aws

数据库开发人员指南

Amazon Redshift

Amazon Redshift: 数据库开发人员指南

Table of Contents

简介

欢迎阅读 Amazon Redshift 数据库开发人员指南。Amazon Redshift 是云中一种完全托管的 PB 级数据仓库服务。Amazon Redshift Serverless 让您可以访问和分析数据，而无需预置数据仓库的常规配置。系统将自动预置资源，数据仓库的容量会智能扩展，即使面对要求最为苛刻且不可预测的工作负载也能提供高速性能。数据仓库空闲时不会产生费用，您只需为实际使用的资源付费。不论数据集的大小如何，您都可以在 Amazon Redshift 查询编辑器 v2 或您最喜欢的商业智能（BI，Business Intelligence）工具中，直接加载数据并开始查询。在易于使用且无需承担管理任务的环境中，享受最佳性价比，使用熟悉的 SQL 功能。

本指南将重点介绍如何使用 Amazon Redshift 创建和管理数据仓库。如果您是数据库的设计者、软件开发人员或管理员，本指南会为您提供设计、构建、查询和维护数据仓库所需的信息。

主题

先决条件

在使用本指南之前，您应阅读 Amazon Redshift Serverless，其中介绍了如何完成以下任务。

- 使用 Amazon Redshift Serverless 创建数据仓库。
- 使用 Amazon Redshift 查询编辑器 v2 加载示例数据
- 从 Amazon S3 加载数据。

您还应该知道如何使用 SQL 客户端，并且对 SQL 语言有基本的了解。

您是数据库开发人员吗？

如果您是首次使用 Amazon Redshift 的用户，我们建议您先阅读 Amazon Redshift Serverless，以了解如何开始使用。

如果您是数据库用户、数据库设计人员、数据库开发人员或数据库管理员，下表将帮助您查找所需的内容。

如果要...	我们建议...
了解 Amazon Redshift 数据仓库的内部架构。	系统和架构概述 概括介绍 Amazon Redshift 的内部架构。 如果您想要更全面地了解 Amazon Redshift Web 服务，请转到 Amazon Redshift 产品详细信息页面。
创建数据库、表、用户和其他数据库对象。	常见数据库任务简要介绍了 SQL 开发的基础知识。 Amazon Redshift SQL 包含有关 Amazon Redshift SQL 命令和函数以及其他 SQL 元素的语法和示例。 设计表的 Amazon Redshift 最佳实践 简要介绍我们有关选择排序键、分配键和压缩编码的建议。
了解如何设计表以实现最佳性能。	使用自动表优化 详细介绍有关压缩表列中的数据以及选择分配键和排序键的注意事项。
加载数据。	加载数据 介绍从 Amazon DynamoDB 表或 Amazon S3 存储桶中存储的平面文件加载大型数据集的过程。 Amazon Redshift 加载数据的最佳实践 提供有关快速高效加载数据的提示。
管理用户、组和数据库安全。	管理数据库安全性 涵盖数据库安全主题。
监控和优化系统性能。	系统表和视图参考 详细介绍您可以从中查询数据库状态并监控查询内容与流程的系统表和视图。 您还应该查阅 Amazon Redshift 管理指南，了解如何使用 AWS Management Console检查系统运行状况、监控指标以及对集群进行备份和还原。
分析和报告来自超大型数据集的信息。	许多受欢迎的软件供应商正在让自己的产品和服务针对 Amazon Redshift 进行认证，以使您能够继续使用您现在使用的工具。有关更多信息，请参阅 Amazon Redshift 合作伙伴页面。 SQL 参考 涵盖有关 Amazon Redshift 支持的 SQL 表达式、命令和函数的所有详细信息。

如果要...	我们建议...
与 Amazon Redshift 资源和表交互。	请参阅《Amazon Redshift Serverless API 指南》、《Amazon Redshift API 指南》和《Amazon Redshift 数据 API 指南》，详细了解如何以编程方式与资源交互和运行操作。
按照教程进一步熟悉 Amazon Redshift。	按照 Amazon Redshift 的教程中的内容，了解有关 Amazon Redshift 功能的更多信息。

系统和架构概述

Amazon Redshift 数据仓库是一个企业级的关系数据库查询和管理系统。

Amazon Redshift 支持与多种类型的应用程序（包括业务情报 (BI)、报告、数据和分析工具）建立客户端连接。

在运行分析查询时，您将在多阶段操作中检索、比较和计算大量数据以产生最终结果。

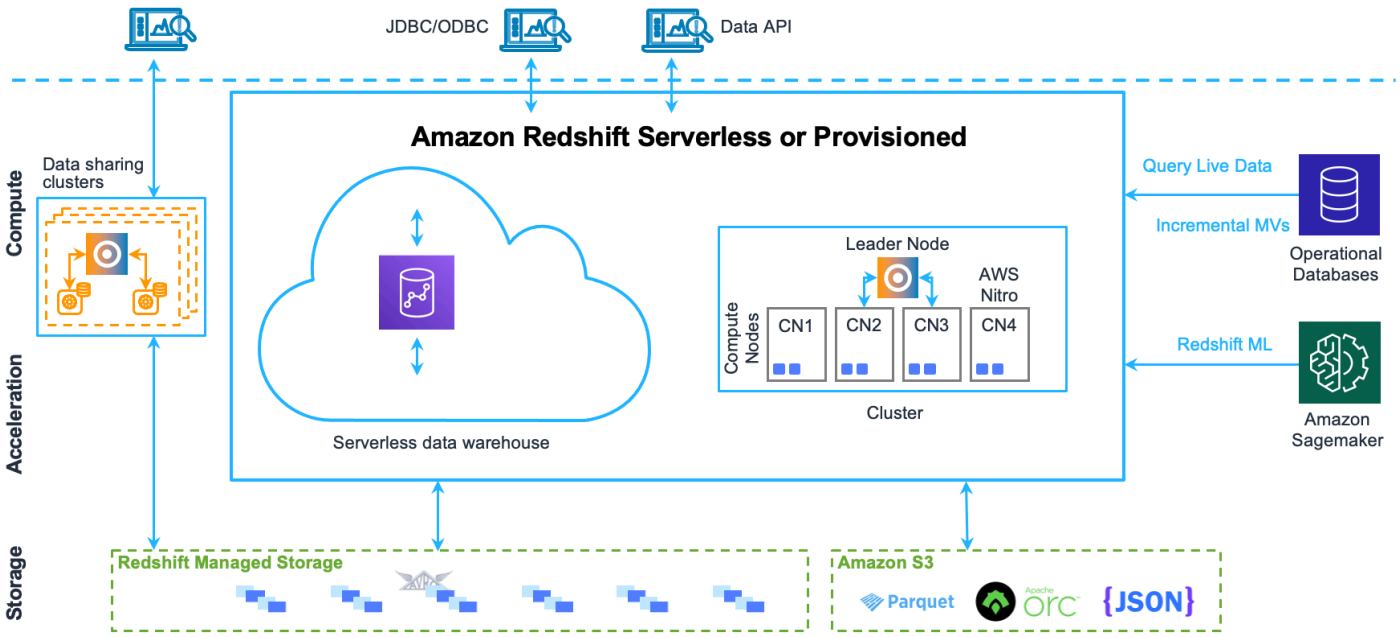
Amazon Redshift 通过大规模并行处理、列式数据存储和非常高效且具有针对性的数据压缩编码方案的组合，实现高效存储和最优查询性能。此部分介绍了 Amazon Redshift 系统架构。

主题

- 数据仓库系统架构
- Performance
- 列式存储
- 工作负载管理
- 将 Amazon Redshift 与其他服务一起使用

数据仓库系统架构

此部分介绍 Amazon Redshift 数据仓库架构的元素，如下图所示。



客户端应用程序

Amazon Redshift 与各种数据加载和 ETL（提取、转换和加载）工具以及业务情报 (BI) 报告、数据挖掘和分析工具集成。Amazon Redshift 基于开放标准 PostgreSQL，因此，大多数现有 SQL 客户端应用程序仅做最少量的更改。有关 Amazon Redshift SQL 和 PostgreSQL 之间的重要差异的信息，请参阅Amazon Redshift 和 PostgreSQL。

集群

Amazon Redshift 数据仓库的核心基础设施组件是集群。

集群包含一个或多个计算节点。如果集群预置有两个或更多计算节点，则一个额外的领导节点将协调这些计算节点并处理外部通信。您的客户端应用程序仅直接与领导节点交互。计算节点对于外部应用程序是透明的。

领导节点

领导节点管理与客户端程序的通信以及与计算节点的所有通信。它分析和制定执行计划以实施数据库操作，特别是获得复杂查询的结果所需执行的一系列步骤。根据执行计划，领导节点编译节点、将编译后的节点分发给计算节点，并将部分数据分配给每个计算节点。

领导节点仅在查询引用计算节点上存储的表时，才将 SQL 语句分发给计算节点。所有其他查询仅在领导节点上运行。Amazon Redshift 被设计为仅在领导节点上实施特定的 SQL 函数。如果使用这些函数中任一函数的查询引用驻留在计算节点上的表，则此查询将返回一个错误。有关更多信息，请参阅 在领导节点上支持的 SQL 函数。

计算节点

领导节点为执行计划的单个元素编译代码并将代码分配给各个计算节点。计算节点运行编译后的代码，并将中间结果发送回领导节点以便最终聚合。

每个计算节点均拥有自己的专用 CPU 和内存，这都由节点类型决定。当您的工作负载增加时，您可以通过增加节点数和/或升级节点类型来增加集群的计算容量。

Amazon Redshift 提供了多种节点类型以满足您的计算需求。有关每种节点类型的详细信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 集群。

Redshift 托管存储

数据仓库数据存储在单独的存储层 Redshift 托管存储 (RMS) 中。RMS 可以使用 Amazon S3 存储将您的存储扩展到 PB 级。RMS 可让您独立扩展计算和存储并支付费用，因此您只需根据计算需求确定集群规模。它自动使用基于 SSD 的高性能本地存储作为第 1 层缓存。它还利用诸如数据块温度、数据块使用期限和工作负载模式之类的优化功能来实现高性能，同时在需要时自动将存储扩展到 Amazon S3，无需采取任何操作。

节点切片

一个计算节点分为多个切片。将为每个切片分配节点的内存和磁盘空间的一部分，从而处理分配给节点的工作负载的一部分。领导节点管理向切片分发数据的工作，并将任何查询或其他数据库操作的工作负载分配给切片。然后，切片将并行工作以完成操作。

每个节点的切片数由集群的节点大小决定。有关每个节点大小的切片数的更多信息，请转到《Amazon Redshift 管理指南》中的关于集群和节点。

在创建表时，您可以选择将一个列指定为分配键。在将表与数据一起加载时，会根据为表定义的分配键将行分配给节点切片。选择好的分配键将使 Amazon Redshift 能够使用并行处理来加载数据和高效运行查询。有关选择分配键的信息，请参阅选择最佳的分配方式。

内部网络

Amazon Redshift 利用高带宽连接、紧邻和自定义通信协议来提供领导节点和计算节点之间的速度极快的私有网络通信。计算节点在客户端应用程序绝对无法直接访问的独立的、隔离网络上运行。

数据库

一个集群包含一个或多个数据库。用户数据存储在计算节点上。您的 SQL 客户端与领导节点进行通信，进而通过计算节点协调查询运行。

Amazon Redshift 是一个关系数据库管理系统 (RDBMS)，可与其他 RDBMS 应用程序兼容。虽然 Amazon Redshift 提供了与典型 RDBMS 相同的功能（包括在线事务处理 (OLTP) 功能，例如，插入并删除数据），但它已经过优化，可对大型数据集进行高性能的分析和报告。

Amazon Redshift 基于 PostgreSQL。Amazon Redshift 和 PostgreSQL 之间的差别非常大，您在设计和开发数据仓库应用程序时需要注意这一点。有关 Amazon Redshift SQL 与 PostgreSQL 之间的差异的信息，请参阅Amazon Redshift 和 PostgreSQL。

Performance

Amazon Redshift 通过使用这些性能功能来实现极快的查询运行。

主题

大规模并行处理

大规模并行处理 (MPP) 支持对大量数据快速运行最复杂的查询。多个计算节点处理所有查询处理以获得最终结果聚合，运行相同的编译后查询的每个节点的每个核心在整个数据的各个部分进行分段。

Amazon Redshift 将表行分配给计算节点，以便能并行处理数据。通过为每个表选择相应的分配键，可以优化数据分配以均衡工作负载，并最大程度地减少节点间的数据移动。有关更多信息，请参阅 选择最佳的分配方式。

加载平面文件中的数据时将利用并行处理，方式是跨多个节点分配工作负载，同时从多个文件进行读取。有关如何将数据加载到表的更多信息，请参阅Amazon Redshift 加载数据的最佳实践。

列式数据存储

数据库表的列式存储大大降低了总体磁盘 I/O 要求，它是优化分析查询性能的一个重要因素。按列式方式存储数据库表信息将减少磁盘 I/O 请求数与需从磁盘加载的数据量。减少加载到内存中的数据量使 Amazon Redshift 能够在执行查询时执行更多的内存中处理。有关更详细的说明，请参阅 列式存储。

在适当地对列进行排序时，查询处理器能够快速筛选出大型数据块子集。有关更多信息，请参阅 选择最佳的排序键。

数据压缩

数据压缩将降低存储要求，从而减少磁盘 I/O 来提高查询性能。在运行查询时，压缩的数据将读入内存，然后在查询运行期间解压缩。将少量数据加载到内存中使 Amazon Redshift 能够分配更多内存来分析数据。由于列式存储将按顺序存储类似数据，因此 Amazon Redshift 能够应用与列式数据类型关联的自适应压缩编码。对表列启用数据压缩的最佳方式是，允许 Amazon Redshift 在您将表与数据一起加载时应用最优压缩编码。要了解有关使用自动数据压缩的更多信息，请参阅使用自动压缩加载表。

查询优化程序

Amazon Redshift 查询运行引擎集成了 MPP 感知的查询优化程序并采用了面向列式的数据存储。Amazon Redshift 查询优化程序实施大量增强和扩展以便处理通常包含多表联接、子查询和聚合的复杂的分析查询。要了解有关优化查询的更多信息，请参阅优化查询性能。

结果缓存

为了缩短查询运行时间并提高系统性能，Amazon Redshift 在领导节点的内存中缓存特定查询类型的结果。当用户提交查询时，Amazon Redshift 会在结果缓存中检查是否有查询结果的有效缓存副本。如果在结果缓存中找到匹配项，Amazon Redshift 会使用缓存的结果而不执行查询。结果缓存对用户透明。

默认情况下，结果缓存处于打开状态。要为当前会话禁用结果缓存，请将 enable_result_cache_for_session 参数设置为 `off`。

在满足以下所有条件时，Amazon Redshift 将为新查询使用缓存的结果：

- 提交查询的用户具有查询中所用对象的访问权限。
- 查询中的表或视图未更改。
- 查询不使用必须在每次运行时求值的函数，例如 GETDATE。
- 该查询不引用 Amazon Redshift Spectrum 外部表。
- 可能影响查询结果的配置参数未更改。
- 查询的语法与缓存的查询相符。

为了最大限度地提升缓存有效性和资源的使用效率，Amazon Redshift 不缓存一些非常大的查询结果集。Amazon Redshift 会根据多个因素确定是否缓存查询结果。这些因素包括缓存中的条目数以及 Amazon Redshift 集群的实例类型。

要确定查询是否使用了结果缓存，请查询 SVL_QLOG 系统视图。如果查询使用了结果缓存，source_query 列会返回源查询的查询 ID。如果未使用结果缓存，则 source_query 列值为 NULL。

以下示例说明了由 userid 104 和 userid 102 提交的查询使用了来自 userid 100 运行的查询的结果缓存。

```
select userid, query, elapsed, source_query from svl_qlog
where userid > 1
order by query desc;

userid | query  | elapsed  | source_query
-------+--------+----------+-------------
   104 | 629035 |       27 |       628919
   104 | 629034 |       60 |       628900
   104 | 629033 |       23 |       628891
   102 | 629017 |  1229393 |
   102 | 628942 |       28 |       628919
   102 | 628941 |       57 |       628900
   102 | 628940 |       26 |       628891
   100 | 628919 | 84295686 |
   100 | 628900 | 87015637 |
   100 | 628891 | 58808694 |
```

编译后的代码

领导节点跨集群的所有节点分发完全优化的编译后的代码。编译查询将减少与解释器关联的开销，从而加快运行速度，特别是加快复杂查询的运行速度。缓存编译后的代码并在同一个集群的多个会话中进行共享。因此，同一查询的未来运行的速度将更快，通常甚至可使用不同的参数来运行。

查询运行引擎为 JDBC 和 ODBC 连接协议编译不同的代码，这样，使用不同协议的两个客户端会各自产生编译代码的首次成本。不过，使用相同协议的客户端会因共享缓存代码而获益。

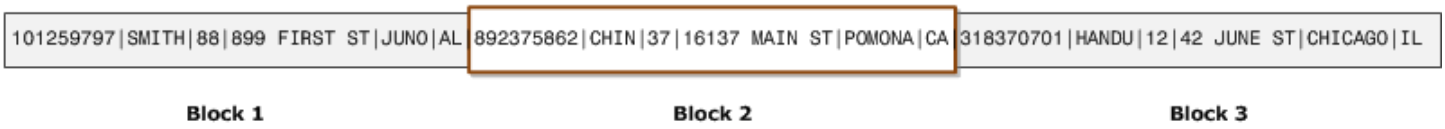
列式存储

数据库表的列式存储大大降低了总体磁盘 I/O 要求，所以是优化分析查询性能的一个重要因素。它减少了您需要从磁盘加载的数据量。

以下一系列图示描述列式数据存储如何实现高效以及如何在将数据检索到内存中实现高效。

此第一个图示说明通常如何将数据库表中的记录（按行）存储到磁盘块中。

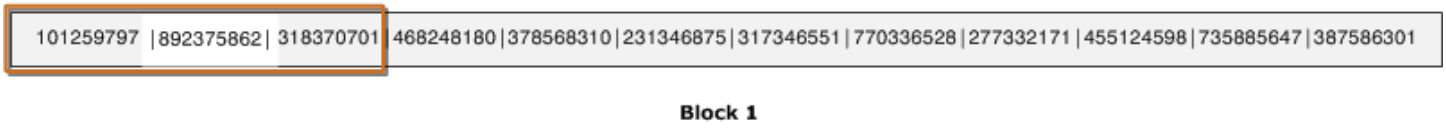
SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



在典型的关系数据库表中，每个行均包含一条记录的字段值。在行式数据库存储中，数据块按顺序存储每个连续列（构成整个行）的值。如果数据块大小小于记录的大小，整个记录的存储可采用多个数据块。如果数据块大小大于记录的大小，整个记录的存储可能采用 1 个以下的数据块，从而导致磁盘空间的使用低效。在在线事务处理 (OLTP) 应用程序中，大多数事务涉及频繁读取和写入整个记录的所有值，通常一次读取和写入一条记录或几条记录。最终，行式存储已针对 OLTP 数据库进行优化。

下一个图示说明，借助列式存储，如何按顺序将每个列的值存储到磁盘块中。

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



使用列式存储，每个数据块可为多个行存储一个列的值。在记录进入系统后，Amazon Redshift 以透明方式将数据转换为每个列的列式存储。

在此简化示例中，借助列式存储，每个数据块将三倍于记录数的列字段值保留为基于行的存储。这意味着，与行式存储相比，为相同数目的记录读取相同数目的列字段值需要三分之一的 I/O 操作数。实际上，使用具有大量列和大量行的表，存储效率甚至会更高。

增加的一个优势是，由于每个块可保留相同类型的数据，因此数据块数据可使用专为列数据类型选择的压缩方案，进一步减少磁盘空间和输入/输出。有关基于数据类型的压缩编码的更多信息，请参阅压缩编码。

磁盘上用于存储数据的空间节省将继续存在，以便检索数据并将其存储在内存中。由于许多数据库操作一次只需访问或操作一个或几个列，您可通过仅检索查询实际所需列的数据块来节省内存空间。其

中，OLTP 事务通常涉及少量记录的一个行中的大多数列或所有列，数据仓库查询通常仅读取大量行的几个列。这意味着，为相同数量的行读取相同数量的列字段值只需要一小部分 I/O 操作。与处理行式块所需的内存相比，它只使用一小部分内存。实际上，通过使用具有大量列和行的表，可大幅提高效率。例如，假定一个表包含 100 个列。使用 5 个列的查询仅需读取表中 5% 的数据。对于大型数据库，可为数十亿或甚至数万亿记录实现此节省。相反，一个行式数据库将读取包含 95 个不需要的列的数据块。

典型的数据库数据块大小介于 2 KB 到 32 KB 之间。Amazon Redshift 使用 1 MB 的块大小，这将提高效率并进一步减少执行任何数据库加载或作为查询运行的一部分的其他操作所需的输入/输出请求数。

工作负载管理

利用 Amazon Redshift 工作负载管理 (WLM)，用户能够灵活地管理工作负载中的优先级，以便时间较短的、快速运行的查询在队列中不会被长时间运行的查询阻碍。

Amazon Redshift WLM 在运行时根据服务类创建查询队列，这将定义各种类型的队列的配置参数，包括内部系统队列和用户可访问的队列。从用户的角度看，用户可访问的服务类和队列在功能上是相同的。为了保持一致，本文档使用队列 一词来表示用户可访问的服务类以及运行时队列。

在运行查询时，WLM 会根据用户的用户组或通过将队列配置中列出的查询组与用户在运行时设置的查询组标签匹配来将查询分配给队列。

> 目前，使用默认参数组的集群的默认行为是使用自动 WLM。自动 WLM 管理查询并发性和内存分配。有关更多信息，请参阅 实施自动 WLM。

对于手动 WLM，Amazon Redshift 配置一个具有并发级别 5 的队列（这将允许同时运行最多 5 个查询）和一个具有并发级别 1 的预定义的超级用户队列。您可以定义最多 8 个队列。每个队列可配置最高 50 的并发级别。所有用户定义的队列（不包括超级用户队列）的最高并发级别总数为 50。

修改 WLM 配置的最简单方法是使用 Amazon Redshift 管理控制台。您还可使用 Amazon Redshift 命令行界面 (CLI) 或 Amazon Redshift API。

有关实施和使用工作负载管理的更多信息，请参阅实施工作负载管理。

将 Amazon Redshift 与其他服务一起使用

Amazon Redshift 与其他 AWS 服务集成，使您能够使用数据安全功能快速可靠地移动、转换和加载数据。

在 Amazon Redshift 和 Amazon S3 之间移动数据

Amazon Simple Storage Service (Amazon S3) 是一种 Web 服务，可在云中存储数据。Amazon Redshift 利用并行处理从 Amazon S3 桶中存储的多个数据文件中读取和加载数据。有关更多信息，请参阅 从 Amazon S3 加载数据。

也可使用并行处理将数据从 Amazon Redshift 数据仓库导出到 Amazon S3 上的多个数据文件中。有关更多信息，请参阅 卸载数据。

结合使用 Amazon Redshift 和 Amazon DynamoDB

Amazon DynamoDB 是一种完全托管的 NoSQL 数据库服务。您可以使用 COPY 命令从单个 Amazon DynamoDB 表加载带数据的 Amazon Redshift 表。有关更多信息，请参阅 从 Amazon DynamoDB 表中加载数据。

通过 SSH 从远程主机中导入数据

您可以在 Amazon Redshift 中使用 COPY 命令从一个或多个远程主机加载数据，例如 Amazon EMR 集群、Amazon EC2 实例或其他计算机。COPY 使用 SSH 连接到远程主机并在远程主机上运行命令以生成数据。Amazon Redshift 支持多个同时连接。COPY 命令从多个主机源并行读取和加载输出。有关更多信息，请参阅 从远程主机中加载数据。

使用 AWS Data Pipeline 自动加载数据

您可以使用 AWS Data Pipeline 来实现数据移动以及数据在 Amazon Redshift 中出入转换的自动化。通过使用 AWS Data Pipeline 的内置计划功能，可以排定和运行重复任务，不必自己编写复杂的数据传输或转换逻辑。例如，您可以将某个重复执行的任务设置为自动将数据从 Amazon DynamoDB 复制到 Amazon Redshift 中。有关指导您完成管道创建以定期将数据从 Amazon S3 移至 Amazon Redshift 的过程的教程，请参阅《AWS Data Pipeline 开发人员指南》中的使用 AWS Data Pipeline 将数据复制到 Amazon Redshift。

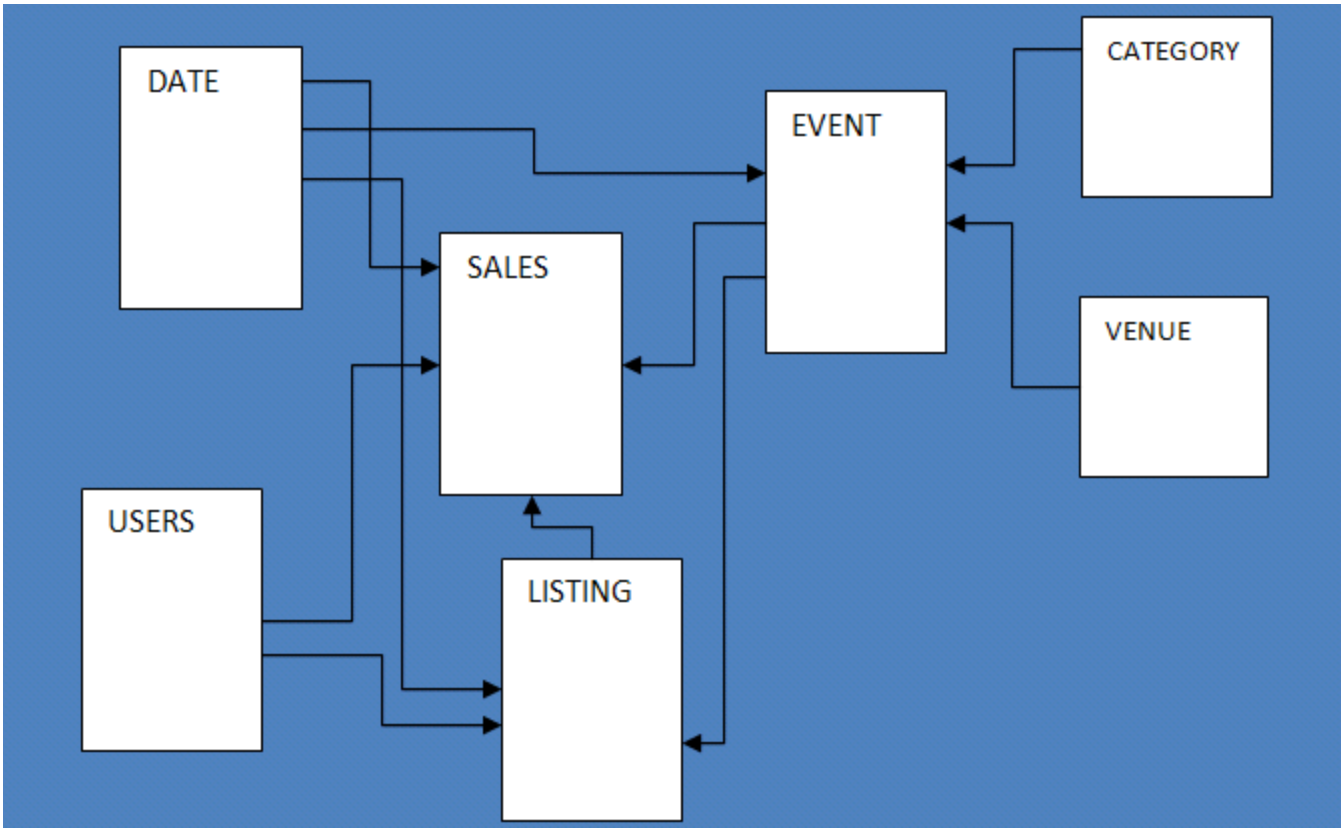
使用 AWS Database Migration Service (AWS DMS) 迁移数据

您可以使用 AWS Database Migration Service 将数据迁移到 Amazon Redshift。AWS DMS 能够将您的数据迁入、迁出使用最为广泛的商用和开源数据库，如 Oracle、PostgreSQL、Microsoft SQL Server、Amazon Redshift、Aurora DB 集群、DynamoDB、Amazon S3、MariaDB 和 MySQL 等。有关更多信息，请参阅将 Amazon Redshift 数据库用作 AWS Database Migration Service 的目标。

示例数据库

主题

Amazon Redshift 文档中的多数示例都使用一个称为 TICKIT 的示例数据库。这个小型数据库包含七个表：两个事实表和五个维度。您可以按照《Amazon Redshift 入门指南》中的步骤 4：从 Amazon S3 将数据加载到 Amazon Redshift 中的步骤加载 TICKIT 数据集。



此示例数据库应用程序帮助分析人员跟踪虚构的 TICKIT 网站的销售活动，用户可以在该网站上在线购买和销售体育赛事、演出和音乐会的门票。具体而言，分析人员可以识别一段时间内门票的变化、卖方

的成功率以及最畅销的活动、场馆和季节。分析人员可以使用这些信息向经常访问该站点的买方和卖方提供奖励，吸引新用户，以及推动广告和促销活动。

例如，下面的查询根据 2008 年销售的门票数，查找圣地亚哥排名前五的卖方：

```
select sellerid, username, (firstname ||' '|| lastname) as name,
city, sum(qtysold)
from sales, date, users
where sales.sellerid = users.userid
and sales.dateid = date.dateid
and year = 2008
and city = 'San Diego'
group by sellerid, username, name, city
order by 5 desc
limit 5;

sellerid | username |       name        |   city    | sum
----------+----------+-------------------+-----------+-----
49977 | JJK84WTE | Julie Hanson      | San Diego |  22
19750 | AAS23BDR | Charity Zimmerman | San Diego |  21
29069 | SVL81MEQ | Axel Grant        | San Diego |  17
43632 | VAG08HKW | Griffin Dodson    | San Diego |  16
36712 | RXT40MKU | Hiram Turner      | San Diego |  14
(5 rows)
```

用于本指南中示例的数据库包含一个小的数据集，其中包含两个事实表，每个表包含的行数不超过 200000，维度范围从 CATEGORY 表中的 11 行到 USERS 表中的大约 50000 行。

具体而言，本指南中的数据库示例演示 Amazon Redshift 表设计的主要功能：

- 数据分布
- 数据排序
- 列式压缩

CATEGORY 表

列名称	数据类型	描述
CATID	SMALLINT	主键，每行的唯一 ID 值。每行表示一种购买和销售其门票的具体活动种类。

列名称	数据类型	描述
CATGROUP	VARCHAR(10)	一组活动的描述性名称，例如 **Shows** 和 **Sports**。
CATNAME	VARCHAR(10)	组内活动种类简短的描述性名称，例如 **Opera** 和 **Musicals**。
CATDESC	VARCHAR(50)	活动种类较长的描述性名称，例如 **Musical theatre**。

DATE 表

列名称	数据类型	描述
DATEID	SMALLINT	主键，每行的唯一 ID 值。每行表示日历年中的一天。
CALDATE	DATE	日历日期，例如 **2008-06-24** 。
DAY	CHAR(3)	星期几（短格式），例如 **SA**。
WEEK	SMALLINT	第几周，例如 **26**。
MONTH	CHAR(5)	月名称（短格式），例如 **JUN**。
QTR	CHAR(5)	第几季度（**1** 到 **4**）。
YEAR	SMALLINT	四位年份 (**2008**)。
HOLIDAY	BOOLEAN	表示当天是否为公共假日（美国）的标志。

EVENT 表

列名称	数据类型	描述
EVENTID	INTEGER	主键，每行的唯一 ID 值。每行表示一个特定时间发生于特定场馆的一项单独活动。
VENUEID	SMALLINT	对 VENUE 表的外键引用。

列名称	数据类型	描述
CATID	SMALLINT	对 CATEGORY 表的外键引用。
DATEID	SMALLINT	对 DATE 表的外键引用。
EVENTNAME	VARCHAR(200)	活动的名称，例如 **Hamlet** 或 **La Traviata**。
STARTTIME	TIMESTAMP	活动的完整日期和开始时间，例如 **2008-10-10 19:30:00**。

VENUE 表

列名称	数据类型	描述
VENUEID	SMALLINT	主键，每行的唯一 ID 值。每行表示活动的一个具体发生场馆。
VENUENAME	VARCHAR(100)	确切的场馆名称，例如 **Cleveland Browns Stadium**。
VENUECITY	VARCHAR(30)	城市名称，例如 **Cleveland** 。
VENUESTATE	CHAR(2)	两个字母组成的州或省缩写（美国和加拿大），例如 **OH**。
VENUESEATS	INTEGER	场馆内提供的最大座位数（如果已知），例如 **73200**。出于演示目的，此列包含一些 Null 值和零。

USERS 表

列名称	数据类型	描述
USERID	INTEGER	主键，每行的唯一 ID 值。每行表示一个已列出或购买至少一项活动的门票的已注册用户（买方和/或卖方）。
USERNAME	CHAR(8)	8 个字符组成的字母数字用户名，例如 **PGL08LJI**。
FIRSTNAME	VARCHAR(30)	用户的名字，例如 **Victor**。
LASTNAME	VARCHAR(30)	用户的姓氏，例如 **Hernandez** 。
CITY	VARCHAR(30)	用户的家所在的城市，例如 **Naperville** 。
STATE	CHAR(2)	用户的家所在的州/省，例如 **GA**。
EMAIL	VARCHAR(100)	用户的电子邮件地址；此列包含随机拉丁值，例如 **turpis@accumsanlaoreet.org** 。
PHONE	CHAR(14)	14 个字符组成的用户电话号码，例如 **(818) 765-4255**。
LIKESPORTS, ...	BOOLEAN	一系列的 10 个不同列，这些列以 **true** 和 **false** 值标识用户的好恶。

LISTING 表

列名称	数据类型	描述
LISTID	INTEGER	主键，每行的唯一 ID 值。每行表示一个特定活动的一批门票的列表。
SELLERID	INTEGER	对 USERS 表的外键引用，标识销售门票的用户。
EVENTID	INTEGER	对 EVENT 表的外键引用。

列名称	数据类型	描述
DATEID	SMALLINT	对 DATE 表的外键引用。
NUMTICKETS	SMALLINT	可供销售的门票数，例如 **2** 或 **20**。
PRICEPERTICKET	DECIMAL(8,2)	单张门票的固定价格，例如 **27.00** 或 **206.00**。
TOTALPRICE	DECIMAL(8,2)	此列表的总价 (NUMTICKETS*PRICEPERTICKET)。
LISTTIME	TIMESTAMP	发布列表的完整日期和时间，例如 **2008-03-18 07:19:35**。

SALES 表

列名称	数据类型	描述
SALESID	INTEGER	主键，每行的唯一 ID 值。每行表示一个特定活动的一张或多张门票的销售，如特定列表中所提供的。
LISTID	INTEGER	对 LISTING 表的外键引用。
SELLERID	INTEGER	对 USERS 表的外键引用（销售门票的用户）。
BUYERID	INTEGER	对 USERS 表的外键引用（购买门票的用户）。
EVENTID	INTEGER	对 EVENT 表的外键引用。
DATEID	SMALLINT	对 DATE 表的外键引用。
QTYSOLD	SMALLINT	已销售的门票数（从 **1** 到 **8**）。（单次交易最多可销售 8 张门票。）
PRICEPAID	DECIMAL(8,2)	支付的总票价，例如 **75.00** 或 **488.00**。单张票价是 PRICEPAID/QTYSOLD。
COMMISSION	DECIMAL(8,2)	企业从销售额中抽取的 15% 佣金，例如 **11.25** 或 **73.20**。卖方得到 85% 的 PRICEPAID 值。

列名称	数据类型	描述
SALETIME	TIMESTAMP	销售完成的完整日期和时间，例如 **2008-05-24 06:21:47**。

Amazon Redshift 最佳实践

在下文中，您可以查找针对计划概念验证、设计表、将数据加载到表中和为 Amazon Redshift 编写查询的最佳实践，还可以查找有关使用 Amazon Redshift Advisor 的讨论。

Amazon Redshift 与其他 SQL 数据库系统不同。为充分实现 Amazon Redshift 架构的优势，您必须专门设计、构建和加载表来使用大规模并行处理、列式数据存储和列式数据压缩。如果数据加载和查询执行时间超过预期或超过您预计等待的时间，您可能忽视了关键信息。

如果您是经验丰富的 SQL 数据库开发人员，我们强烈建议您在开发 Amazon Redshift 数据仓库前查阅主题。

如果您是 SQL 数据库开发新手，本主题并不适合入门。要想开始使用，建议您阅读常见数据库任务并亲自试用示例。

在本主题中，您可以找到最重要的开发原则的概述，以及实施这些原则的具体提示、示例和最佳实践。没有任何一种做法可以适用于所有应用。应在完成数据库设计之前评估所有选项。有关更多信息，请参阅使用自动表优化、加载数据、优化查询性能和参考章节。

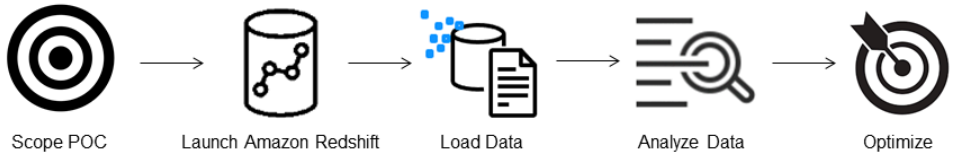
主题

为 Amazon Redshift 执行概念验证（POC）。

Amazon Redshift 是一种流行的云数据仓库，它提供完全托管的基于云的服务，可与组织的 Amazon Simple Storage Service 数据湖、实时流、机器学习（ML）工作流、事务性工作流等集成。以下各节将指导您完成对 Amazon Redshift 执行概念验证（POC）的过程。此处的信息可帮助您为 POC 设定目标，并利用可自动为 POC 预置和配置服务的工具。

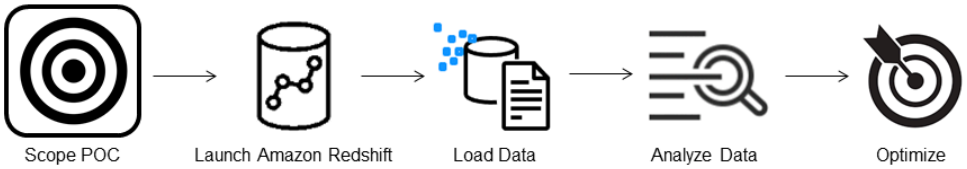
> **Note**
>
> 如需 PDF 格式的信息副本，请选择 Amazon Redshift 资源页面上的运行您自己的 Redshift POC 链接。

执行 Amazon Redshift 的 POC 时，您需要测试、证明和采用各种功能，包括出色的安全功能、弹性扩展、易于集成和摄取以及灵活的去中心化数据架构选项。



请遵循以下步骤，以成功执行 POC。

步骤 1：确定 POC 的范围



执行 POC 时，您可以选择使用自己的数据，也可以选择使用基准测试数据集。选择自己的数据时，您可以对数据运行自己的查询。使用基准测试数据时，基准测试中提供了示例查询。如果您尚未准备好使用自己的数据执行 POC，请参阅使用示例数据集了解更多详细信息。

一般来说，我们建议 Amazon Redshift POC 使用两周的数据。

首先执行以下步骤：

1. 确定您的业务和功能需求，然后进行倒推。常见的示例有：更快的性能、更低的成本、测试新的工作负载或功能，或者将 Amazon Redshift 与其他数据仓库进行比较。
2. 设定具体目标，这些目标将成为 POC 的成功标准。例如，从更快的性能出发，列出您希望加速的前五个流程，并将当前的运行时间和所需的运行时间一并列出。这些可以是报告、查询、ETL 流程、数据摄取，或者您当前的任何棘手问题。

3. 确定运行测试所需的具体范围和构件。您需要哪些数据集才能迁移或持续摄取到 Amazon Redshift 中，以及需要哪些查询和流程来运行测试以根据成功标准进行衡量？ 有两种方式可执行此操作：

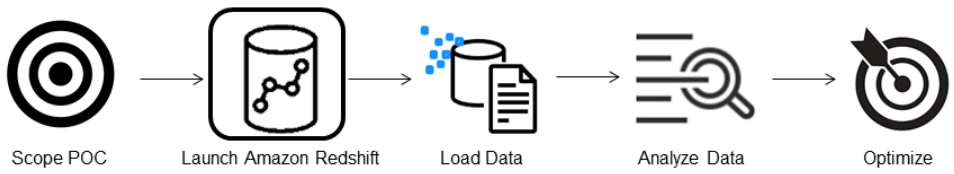
   自带数据

   - 要测试自己的数据，请列出测试成功标准所需的最低可行数据构件清单。例如，如果您当前的数据仓库有 200 个表，但您要测试的报告只需要 20 个表，则仅使用较小的表子集可以更快地运行 POC。

   使用示例数据集

   - 如果您没有准备好自己的数据集，仍可使用行业标准的基准数据集（如 TPC-DS 或 TPC-H）开始对 Amazon Redshift 执行 POC，并运行示例基准测试查询以利用 Amazon Redshift 的强大功能。创建 Amazon Redshift 数据仓库后，可以从其内部访问这些数据集。有关如何访问这些数据集和示例查询的详细说明，请参阅步骤 2：启动 Amazon Redshift。

步骤 2：启动 Amazon Redshift



Amazon Redshift 可通过快速、简单且安全的大规模云数据仓库服务，使您更快地获得见解。您可以通过在 Redshift Serverless 控制台上启动仓库来快速开始工作，在几秒内将数据转化为见解。有了 Redshift Serverless，您就可以专注于交付业务成果，而不必担心管理数据仓库。

设置 Amazon Redshift Serverless

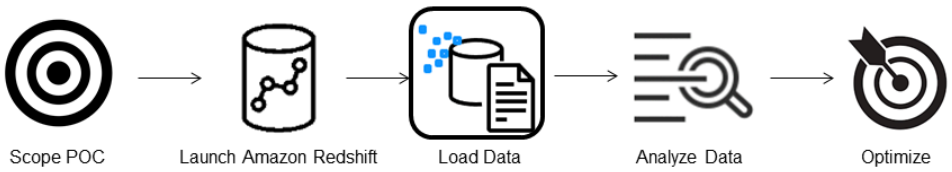
首次使用 Redshift Serverless 时，控制台会引导您完成启动仓库所需的步骤。您也可能有资格获得账户中的 Redshift Serverless 使用积分。有关选择免费试用的更多信息，请参阅 Amazon Redshift 免费试用。按照《Amazon Redshift 入门指南》中的使用 Redshift Serverless 创建数据仓库中的步骤，使用 Redshift Serverless 创建数据仓库。如果您没有想要加载的数据集，该指南还包含有关如何加载示例数据集的步骤。

如果您之前在自己的账户中启动过 Redshift Serverless，请按照《Amazon Redshift 管理指南》中的创建带有命名空间的工作组中的步骤进行操作。仓库可用后，您可以选择加载 Amazon Redshift 中

提供的示例数据。有关使用 Amazon Redshift 查询编辑器 v2 以加载数据的信息，请参阅《Amazon Redshift 管理指南》中的加载示例数据。

如果您自带数据而不是加载示例数据集，请参阅步骤 3：加载数据。

步骤 3：加载数据



启动 Redshift Serverless 后，下一步是为 POC 加载数据。无论您是上传简单的 CSV 文件、从 S3 中摄取半结构化数据，还是直接流式传输数据，Amazon Redshift 都能灵活地将数据从源位置快速轻松地移动到 Amazon Redshift 表中。

选择以下方法之一以加载数据。

上传本地文件

为了快速摄取和分析，您可以使用 Amazon Redshift 查询编辑器 v2 轻松从本地桌面加载数据文件。它能够处理各种格式的文件，如 CSV、JSON、AVRO、PARQUET、ORC 等。要让您的用户以管理员身份使用查询编辑器 v2 从本地桌面加载数据，您必须指定一个通用 Amazon S3 存储桶，并且必须为该用户账户配置适当的权限。您可以遵循使用查询编辑器 V2 在 Amazon Redshift 中轻松安全地加载数据，以获取分步指导。

加载 Amazon S3 文件

要将数据从 Amazon S3 存储桶加载到 Amazon Redshift 中，请首先使用 COPY 命令，指定源 Amazon S3 位置和目标 Amazon Redshift 表。确保正确配置 IAM 角色和权限，以允许 Amazon Redshift 访问指定的 Amazon S3 存储桶。请遵循教程：从 Amazon S3 加载数据，以获取分步指导。您也可以选择查询编辑器 v2 中的加载数据选项，直接从 S3 存储桶加载数据。

持续数据摄取

自动复制（预览版）是 COPY 命令的扩展，可自动从 Amazon S3 存储桶持续加载数据。当您创建 COPY 作业时，Amazon Redshift 会检测何时在指定路径中创建新的 Amazon S3 文件，然后自动加载

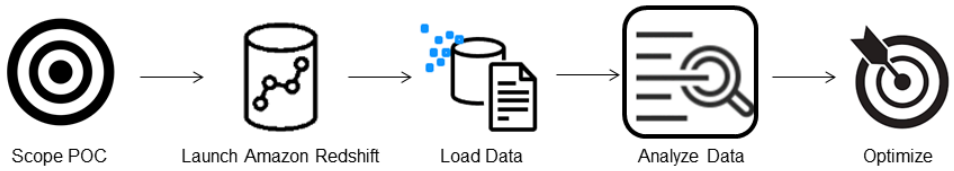
这些文件，无需您的干预。Amazon Redshift 会跟踪加载的文件，以确认它们只加载一次。有关创建 COPY 作业的说明，请参阅COPY JOB（预览版）

> **Note**
>
> 自动复制目前处于预览状态，仅在特定 AWS 区域的预置集群中受支持。要创建用于自动复制的预览集群，请参阅从 Amazon S3 持续摄取文件（预览版）。

加载流数据

串流摄取直接以低延迟、高速度的方式将流数据从 Amazon Kinesis Data Streams 和 Amazon Managed Streaming for Apache Kafka 摄取到 Amazon Redshift 中。Amazon Redshift 串流摄取使用实体化视图，该视图将利用自动刷新功能直接从流中更新。实体化视图映射到流数据来源。在实体化视图定义中，您可以对流数据执行筛选和聚合。有关从流中加载数据的分步指南，请参阅 Amazon Kinesis Data Streams 入门或开始使用 Amazon Managed Streaming for Apache Kafka 串流摄取。

步骤 4：分析数据



创建 Redshift Serverless 工作组和命名空间并加载数据后，您可以通过从 Redshift Serverless 控制台的导航面板打开查询编辑器 v2 来立即运行查询。您可以使用查询编辑器 v2 针对自己的数据集测试查询功能或查询性能。

使用 Amazon Redshift 查询编辑器 v2 进行查询

您可以从 Amazon Redshift 控制台访问查询编辑器 v2。有关如何使用查询编辑器 v2 配置、连接和运行查询的完整指南，请参阅使用 Amazon Redshift 查询编辑器 v2 简化数据分析。

或者，如果您想在 POC 中运行负载测试，则可以通过以下步骤来安装和运行 Apache JMeter。

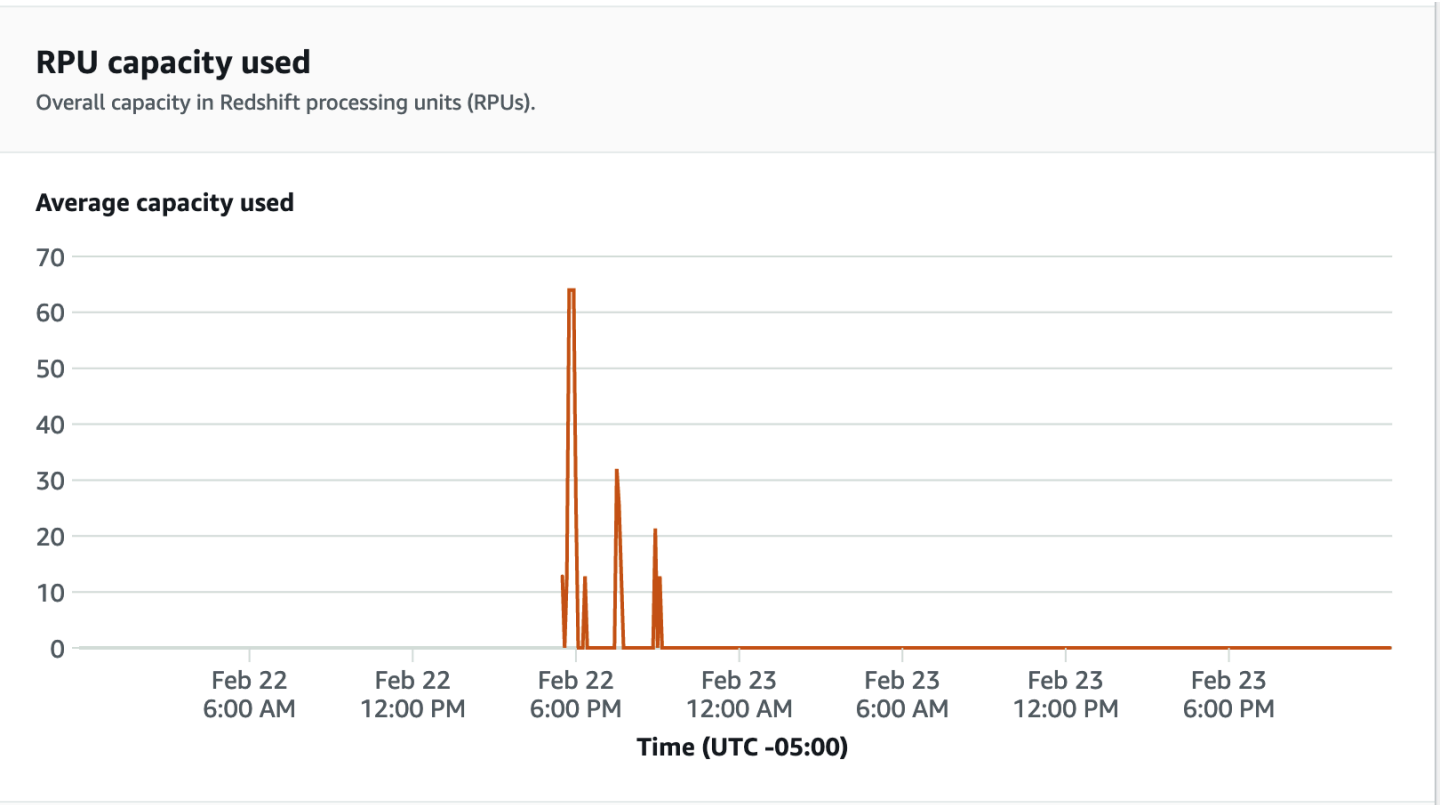
使用 Apache JMeter 运行负载测试

要执行负载测试以模拟“N”个用户同时向 Amazon Redshift 提交查询，可以使用基于 Java 的开源工具 Apache JMeter。

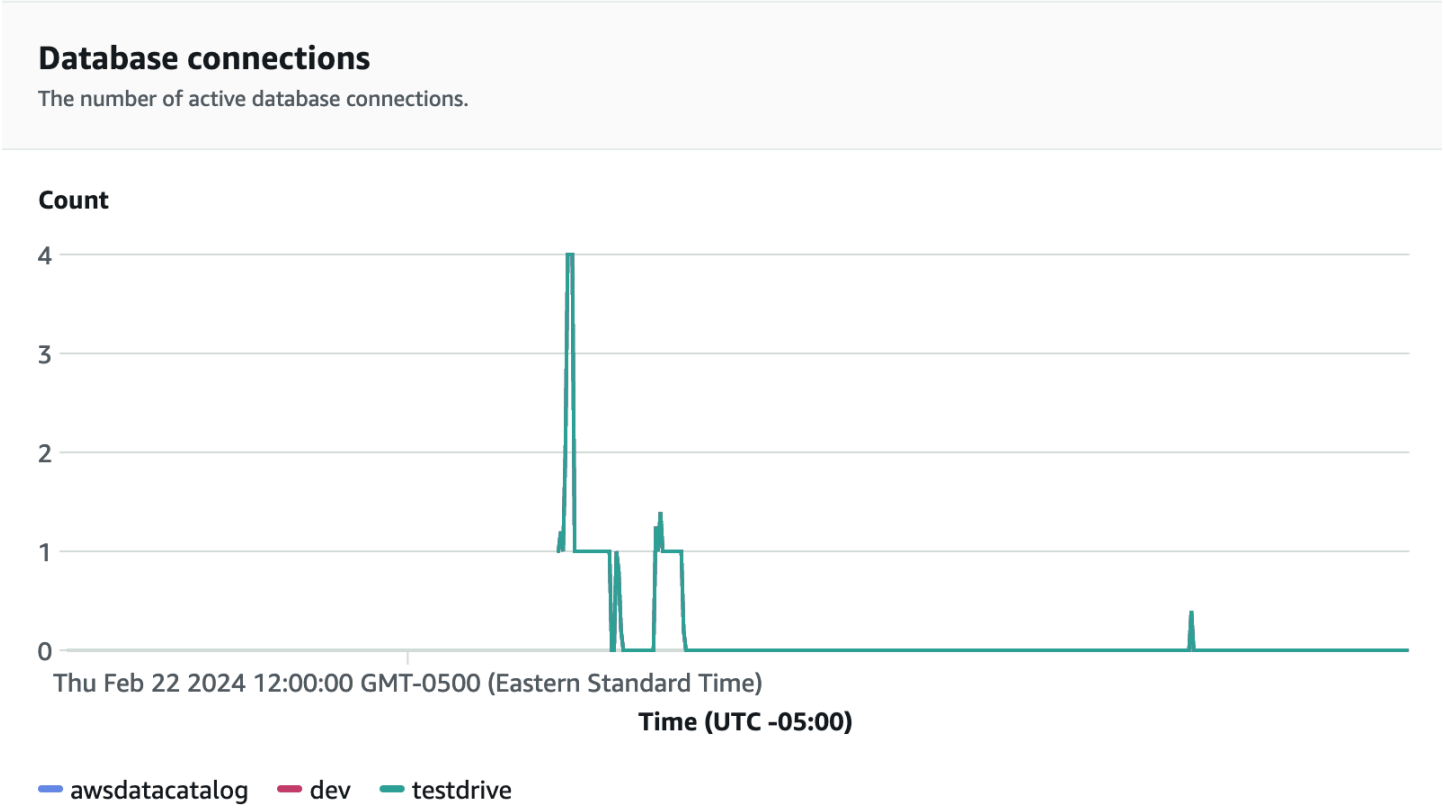
要安装和配置 Apache JMeter 以在 Redshift Serverless 工作组中运行，请按照使用 AWS 分析自动化工具包自动进行 Amazon Redshift 负载测试中的说明进行操作。它使用 AWS 分析自动化工具包（AAA）（一种用于动态部署 Redshift 解决方案的开源工具）来自动启动这些资源。如果您已将自己的数据加载到 Amazon Redshift 中，请务必执行步骤 5 – 自定义 SQL 选项，以确保提供要针对表进行测试的相应 SQL 语句。使用查询编辑器 v2 对每个 SQL 语句进行一次测试，以确保它们运行时没有错误。

完成自定义 SQL 语句并最终确定测试计划后，请进行保存并在 Redshift Serverless 工作组中运行测试计划。要监控测试进度，请打开 Redshift Serverless 控制台，导航到查询和数据库监控，选择查询历史记录选项卡，然后查看有关查询的信息。

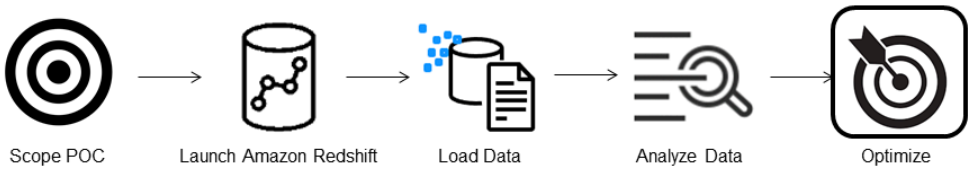
要查看性能指标，请在 Redshift Serverless 控制台上选择数据库性能选项卡，以监控数据库连接和 CPU 利用率等指标。在这里，您可以查看图表以监控使用的 RPU 容量，并观察在工作组上运行负载测试时 Redshift Serverless 如何自动扩展以满足并发工作负载需求。



数据库连接是在运行负载测试时要监控的另一个有用指标，您可以通过该指标来了解工作组在给定时间如何处理大量并发连接以满足不断增长的工作负载需求。



步骤 5：优化



Amazon Redshift 通过提供各种配置和功能来支持各个使用案例，使成千上万的用户能够每天处理 EB 级数据，并为其分析工作负载提供支持。在这些选项之间进行选择时，客户希望寻找可帮助他们确定最优数据仓库配置以支持其 Amazon Redshift 工作负载的工具。

试用方案

您可以使用试用方案在潜在配置上自动重播现有工作负载，并分析相应的输出，以评估要将工作负载迁移到的最佳目标。有关使用试用方案评估不同 Amazon Redshift 配置的信息，请参阅使用 Redshift 试用方案查找适合工作负载的 Amazon Redshift 配置。

设计表的 Amazon Redshift 最佳实践

在规划数据库时，某些关键表设计决策对整体查询性能影响很大。这些设计选择可以减少 I/O 操作数和尽量减少处理查询所需的内存，因而对存储需求以至查询性能也有很大影响。

在本节中，您可以找到最重要的设计决策的总结和用于优化查询性能的最佳实践。使用自动表优化提供了更为详细的表设计选项说明和示例。

主题

选择最佳的排序键

Amazon Redshift 根据排序键将您的数据按照排序顺序存储在磁盘中。Amazon Redshift 查询优化程序在确定最佳查询计划时会使用排序顺序。

> **Note**
>
> 使用自动表优化时，不需要选择表的排序键。有关更多信息，请参阅 使用自动表优化。

关于最佳方法的一些建议如下：

- 要让 Amazon Redshift 选择适当的排序顺序，请指定 AUTO 作为排序键。
- 如果最近使用的数据查询频率最高，则指定时间戳列作为排序键的第一列。

  这样查询会更高效，因为可以跳过该时间范围之外的整个数据块。
- 如果您经常对某列进行范围筛选或相等性筛选，则指定该列作为排序键。

  Amazon Redshift 可以不读取该列的整个数据块。之所以能这样做，是因为它会跟踪每个数据块中存储的最小和最大列值，并且可以跳过不适用于指定范围的数据块。
- 如果您频繁联接表，则指定联接列作为排序键和分配键。

这样，查询优化程序可以选择排序合并联接而不是较慢的哈希联接。因为数据已按联接键排序，所以查询优化程序不用执行排序合并联接的排序阶段。

选择最佳的分配方式

在运行查询时，查询优化程序根据执行任何联接和聚合的需要将行重新分配到计算节点。选择表分配方式的目的是通过在运行查询前将数据放在需要的位置来最大程度地减小重新分配步骤的影响。

> **Note**
>
> 使用自动表优化时，不需要选择表的分配方式。有关更多信息，请参阅 使用自动表优化。

关于最佳方法的一些建议如下：

1. 根据共同列分配事实数据表和一个维度表。

   事实数据表只能有一个分配键。任何通过其他键联接的表都不能与事实数据表并置。根据联接频率和联接行的大小选择一个要并置的维度。将维度表的主键和事实数据表对应的外键指定为 DISTKEY。

2. 根据筛选的数据集的大小选择最大的维度。

   只有用于联接的行才必须分配，因此需要考虑筛选后数据集的大小，而不是表的大小。

3. 在筛选结果集中选择基数高的列。

   例如，如果您在日期列上分配了一个销售表，您应该能获得非常均匀的数据分配，除非您的大多数销售都是季节性的。但是，如果您通常使用范围受限谓词进行筛选以缩小日期期间的范围，则大多数筛选行都将出现在有限的一组切片上并且查询工作负载将偏斜。

4. 将一些维度表改为使用 ALL 分配。

   如果一个维度表不能与事实数据表或其他重要的联接表并置，您可以通过将整个表分配到所有节点来大大提高查询性能。使用 ALL 分配会使存储空间需求成倍增长，并且会增加加载时间和维护操作，所以在选择 ALL 分配前应权衡所有因素。

要让 Amazon Redshift 选择适当的分配方式，请指定 AUTO 作为分配方式。

有关选择分配方式的更多信息，请参阅使用数据分配样式。

让 COPY 选择压缩编码

您可以在创建表时指定压缩编码，但在大多数情况下，自动压缩的效果最好。

ENCODE AUTO 是表的默认设置。将表设置为 ENCODE AUTO 时，Amazon Redshift 会自动管理表中所有列的压缩编码。有关更多信息，请参阅CREATE TABLE 和ALTER TABLE。

在加载操作中，COPY 命令自动分析您的数据并对一个空表应用压缩编码。

自动压缩将在选择压缩编码时平衡整体性能。如果排序键列的压缩率远高于同一查询中的其他列，则范围受限扫描的执行效果可能会很差。因此，自动压缩将选择一个效率较低的压缩编码来让排序键列与其他列保持平衡。

假设您的表的排序键为日期或时间戳，并且表使用了很多大型 varchar 列。在这种情况下，您可能通过完全不压缩排序键列来获取更高的性能。请对表运行 ANALYZE COMPRESSION 命令，然后使用编码创建一个新表，但忽略排序键的压缩编码。

仅当表为空且尚未指定压缩编码时，自动压缩编码才存在性能开销。对于短期存在的表和经常创建的表（如暂存表），可以使用自动压缩加载该表一次，或运行 ANALYZE COMPRESSION 命令。然后使用这些编码创建新表。您可以将编码添加到 CREATE TABLE 语句中或使用 CREATE TABLE LIKE 来创建采用相同编码的新表。

有关更多信息，请参阅 使用自动压缩加载表。

定义主键和外键约束

如果适当，在表之间定义主键和外键约束。即使只是信息性的，查询优化程序也可以使用这些约束来生成更有效的查询计划。

除非您的应用程序强制实施约束，否则不要定义主键和外键约束。Amazon Redshift 不强制实施唯一约束、主键约束和外键约束。

有关 Amazon Redshift 如何使用约束的其他信息，请参阅定义表约束。

使用尽可能小的列大小

不要为了方便而经常使用最大列大小。

而是应考虑您可能存储在列中的最大值，并相应地调整它们的大小。例如，用于存储邮局使用的美国州和地区缩写的 CHAR 列只需要是 CHAR(2)。

在日期列中使用日期/时间数据类型

Amazon Redshift 存储 DATE 和 TIMESTAMP 数据的效率高于 CHAR 或 VARCHAR，这可以改善查询性能。在存储日期/时间信息时，请根据所需精度（而不是字符类型）使用 DATE 或 TIMESTAMP 数据类型。有关更多信息，请参阅 日期时间类型。

Amazon Redshift 加载数据的最佳实践

主题

加载非常大的数据集可能需要花费很长时间，并且会消耗大量计算资源。数据的加载方式还会影响查询性能。本节介绍使用 COPY 命令、批量插入和临时表有效加载数据的最佳实践。

学习数据加载教程

教程：从 Amazon S3 加载数据 指导您逐步将数据上载到 Simple Storage Service（Amazon S3）存储桶，然后使用 COPY 命令将数据加载到您的表中。本教程包含有关解决加载错误的帮助，还比较从单个文件加载和从多个文件加载之间的性能差异。

使用 COPY 命令加载数据

COPY 命令从 Simple Storage Service（Amazon S3）、Amazon EMR、Amazon DynamoDB 或远程主机上的多个数据来源并行加载数据。COPY 加载大量数据的效率要比 INSERT 语句高很多，而且存储数据也更有效率。

有关如何使用 COPY 命令的更多信息，请参阅从 Amazon S3 加载数据和从 Amazon DynamoDB 表中加载数据。

使用一个 COPY 命令从多个文件中加载

Amazon Redshift 可自动从多个压缩数据文件并行加载。您可以使用 Amazon S3 对象前缀或清单文件指定要加载的文件。

但如果您使用多个并发 COPY 命令从多个文件加载一个表，系统会强制 Amazon Redshift 执行序列化加载。如果表已定义一个排序列，这种加载会慢得多并且需要在结束时执行 VACUUM 流程。有关使用 COPY 并行加载数据的更多信息，请参阅从 Amazon S3 加载数据。

加载数据文件

源数据文件采用不同的格式，并使用不同的压缩算法。使用 COPY 命令加载数据时，Amazon Redshift 会加载 Amazon S3 桶前缀引用的所有文件。（前缀是对象键名称开头的一串字符。）如果前缀指的是多个文件或可以拆分的文件，Amazon Redshift 会利用 Amazon Redshift 的 MPP 架构并行加载数据。这将在集群中的节点间划分工作负载。相比之下，当您从无法拆分的文件加载数据时，Amazon Redshift 会强制执行序列化加载，这样速度会慢得多。以下部分介绍将不同文件类型加载到 Amazon Redshift 的建议方法，具体取决于它们的格式和压缩情况。

从可以拆分的文件中加载数据

加载以下文件的数据时，可以自动拆分这些文件：

- 未压缩的 CSV 文件
- 使用 BZIP 压缩的 CSV 文件
- 列式文件（Parquet/ORC）

Amazon Redshift 会自动将 128MB 或更大的文件拆分成数据块。列式文件（特别是 Parquet 和 ORC）如果小于 128MB，则不会被拆分。Redshift 利用并行工作的切片来加载数据。实现快速加载性能。

从无法拆分的文件加载数据

使用 GZIP 等其他压缩算法进行压缩时，JSON 或 CSV 等文件类型不会自动拆分。对于这些文件类型，我们建议您手动将数据拆分为多个大小接近的小文件，这些小文件压缩后的大小介于 1MB 到 1GB 之间。此外，将文件数设为您的集群中切片数的倍数。有关如何将数据拆分为多个文件的更多信息和如何使用 COPY 加载数据的示例，请参阅从 Amazon S3 加载数据。

压缩数据文件

当要压缩大型加载文件时，我们建议您使用 gzip、lzop、bzip2 或 Zstandard 来压缩，并将数据拆分为多个小文件。

在 COPY 命令中指定 GZIP、LZOP、BZIP2 或 ZSTD 选项。此示例从一个用竖线分隔的 lzop 文件加载 TIME 表。

```
copy time
from 's3://mybucket/data/timerows.lzo'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
lzop
delimiter '|';
```

在某些情况下，您无需拆分未压缩的数据文件。有关如何拆分数据的更多信息以及如何使用 COPY 加载数据的示例，请参阅从 Amazon S3 加载数据。

在加载前后验证数据文件

在从 Amazon S3 加载数据之前，首先验证 Amazon S3 桶是否包含所有正确的文件，并且仅包含这些文件。有关更多信息，请参阅 确认在桶中具有正确的文件。

在完成加载操作后，请查询 STL_LOAD_COMMITS 系统表以验证所需文件是否已加载。有关更多信息，请参阅 验证是否正确加载了数据。

使用多行插入

如果不能使用 COPY 命令，而您需要进行 SQL 插入，可以根据情况使用多行插入。如果一次只添加一行或几行，则数据压缩效率低下。

多行插入通过批量进行一系列插入提高性能。下面的示例使用一条 INSERT 语句向一个包含四列的表中插入三行。这仍是少量插入，只是用来说明多行插入的语法。

```
insert into category_stage values
(default, default, default, default),
(20, default, 'Country', default),
(21, 'Concerts', 'Rock', default);
```

有关更多详细信息和示例，请参阅 INSERT。

使用批量插入

带 SELECT 子句使用批量插入操作来实现高性能数据插入。

如果需要将数据或数据子集从一个表移动到另一个表，可以使用 INSERT 和 CREATE TABLE AS 命令。

例如，下面的 INSERT 语句从 CATEGORY 表中选择所有行并将它们插入 CATEGORY_STAGE 表。

```
insert into category_stage
(select * from category);
```

下面的示例创建 CATEGORY_STAGE 作为 CATEGORY 的副本，并将 CATEGORY 中的所有行插入 CATEGORY_STAGE。

```
create table category_stage as
select * from category;
```

按排序键顺序加载数据

按排序键顺序加载数据可以避免对 vacuum 的需要。

如果每批新数据都遵从表中的现有行，您的数据就将按排序顺序正确存储，您不需要运行 vacuum。COPY 在加载时对每批传入数据进行排序，因此您无需对每次加载的行进行预排序。

例如，假设您每天根据当天活动加载数据。如果您的排序键是时间戳列，您的数据将按排序顺序存储。之所以会出现这种顺序，是因为当天的数据总是附加在前一天的数据的后面。有关更多信息，请参阅按排序键顺序加载数据。有关 vacuum 操作的更多信息，请参阅对表执行 vacuum 操作。

加载有序数据块中的数据

如果需要添加大量数据，加载符合排序顺序的有序数据块则不需要使用 Vacuum。

例如，假设您需要加载包含从 2017 年 1 月到 2017 年 12 月的事件的表。假设每个月均在一个单独的文件中，请加载 1 月的行、2 月的行，依此类推。加载完成后，表已完成排序，无需运行 Vacuum。有关更多信息，请参阅 使用时间序列表。

在加载非常大的数据集时，排序操作所需要的空间可能会超出总可用空间。通过加载较小数据块中的数据，每次加载期间使用的中间排序空间会大幅减少。此外，如果 COPY 失败回滚，加载较小的数据块更容易重启。

使用时间序列表

如果数据有固定保留期，可以将数据组织为时间序列表序列。在此类序列中，每个表都相同但包含不同时间范围的数据。

您只需在相应表上运行 DROP TABLE 命令即可轻松删除旧数据。此方法比运行大规模 DELETE 流程快得多，并使您不必运行后续 VACUUM 流程以回收空间。要隐藏数据存储在不同表中这一事实，您可以创建 UNION ALL 视图。删除旧数据时，应优化您的 UNION ALL 视图以移除已删除的表。同样，向新表加载新时间段时，只需将新表添加到视图中。为了通知优化程序跳过对与查询筛选条件不匹配的表的扫描，您的视图定义将筛选与每个表对应的日期范围。

避免在 UNION ALL 视图中包含过多的表。每个附加表都会给查询增加一点处理时间。表不需要使用相同的时间范围。例如，您可能拥有不同时间段（如每日、每月、每年）的表。

如果使用时间序列表以时间戳列作为排序键，则可按排序键顺序高效加载数据。这样做将使 vacuum 不必对数据重新排序。有关更多信息，请参阅 按排序键顺序加载数据。

计划维护时段

如果查询运行时发生计划的维护，查询会终止并回滚，需要重启。将长时间运行的操作（如大型数据加载或 VACUUM 操作）计划在维护时间段之外进行。您也可以通过执行较小增量的数据加载和管理 VACUUM 操作的大小尽可能地降低风险，在需要重启时简化重启。有关更多信息，请参阅加载有序数据块中的数据和对表执行 vacuum 操作。

设计查询的 Amazon Redshift 最佳实践

要最大程度地提高查询性能，请在创建查询时遵循这些建议：

- 根据最佳实践设计表，为查询性能提供坚固的基础。有关更多信息，请参阅 设计表的 Amazon Redshift 最佳实践。
- 避免使用 `select *`。仅包含您需要的列。
- 使用 CASE 条件表达式 执行复杂聚合，而不是从同一表多次选择。

- 除非绝对必要，否则，不要使用交叉联接。这些没有联接条件的联接会导致两个表的笛卡尔积。交叉联接通常作为嵌套循环联接运行，这是最慢的联接类型。
- 如果查询中有一个表只用于谓词条件并且子查询返回的行数较少 (不足 200 行)，可以使用子查询。下面的示例使用子查询来避免联接 LISTING 表。

```
select sum(sales.qtysold)
from sales
where salesid in (select listid from listing where listtime > '2008-12-26');
```

- 使用谓词尽可能限制数据集。
- 在谓词中，尽可能使用成本最低的运算符。比较条件 运算符优先于 LIKE 运算符。LIKE 运算符优先于 SIMILAR TO 或 POSIX 运算符。
- 避免在查询谓词中使用函数。如果使用函数，则需要大量的行来解析查询的中间步骤，从而增加查询成本。
- 如果可能，请使用 WHERE 子句限制数据集。查询计划程序可以使用行顺序来帮助确定哪些记录与条件匹配，从而不必扫描大量磁盘数据块。如果没有它，查询执行引擎必须完全扫描参与的列。
- 添加谓词以筛选参与联接的表（即使谓词应用相同的筛选条件）。查询返回相同的结果集，但 Amazon Redshift 能够在扫描步骤前筛选联接表，然后高效地跳过对这些表数据块的扫描。如果对在联接条件中使用的列进行筛选，则无需指定冗余筛选条件。

  例如，假设您需要联接 SALES 和 LISTING 来查找在 12 月之后列出的票的销售数据并按卖家分组。两个表都按日期排序。下面的查询使用表的公共键联接这两个表，并筛选晚于 12 月 1 日的 `listing.listtime` 值。

```
select listing.sellerid, sum(sales.qtysold)
from sales, listing
where sales.salesid = listing.listid
and listing.listtime > '2008-12-01'
group by 1 order by 1;
```

  WHERE 子句不包含针对 `sales.saletime` 的谓词，因此，执行引擎必须扫描整个 SALES 表。如果您知道哪些筛选条件能够减少参与联接操作的行数，则也请添加这些筛选条件。下面的示例明显地减少了执行时间。

```
select listing.sellerid, sum(sales.qtysold)
from sales, listing
where sales.salesid = listing.listid
and listing.listtime > '2008-12-01'
```

```
and sales.saletime > '2008-12-01'
group by 1 order by 1;
```

- 在 GROUP BY 子句中使用排序键，以便查询计划程序可以使用更高效的聚合。如果查询的 GROUP BY 列表只包含排序键列，并且其中一列也是分配键，则查询可能适合单阶段聚合。GROUP BY 列表中的排序键列必须包含第一个排序键，然后按排序键顺序包含其他需要使用的排序键。例如，可以使用第一个排序键，第一个和第二个排序键，第一个、第二个和第三个排序键，依此类推。不能使用第一个和第三个排序键。

  您可以通过在查询的聚合步骤中运行 EXPLAIN 命令和查找 `XN GroupAggregate` 来确认使用单阶段聚合。
- 如果同时使用 GROUP BY 和 ORDER BY 子句，两个子句中各列的顺序必须相同。也就是，使用下面的方法。

  ```
  group by a, b, c
  order by a, b, c
  ```

  不要使用以下方法。

  ```
  group by b, c, a
  order by a, b, c
  ```

采用 Amazon Redshift Advisor 的建议

为了帮助您提高性能并降低 Amazon Redshift 集群的运营成本，Amazon Redshift Advisor 为您提供了有关要进行的更改的特定建议。Advisor 通过分析集群的性能和使用量指标来制定其定制建议。这些定制建议与操作和集群设置相关。为帮助您设定优化的优先顺序，Advisor 按影响度顺序对建议进行了排名。

Advisor 基于有关性能统计数据或操作数据的观察来制定建议。Advisor 将对集群运行测试以确定测试值是否在指定范围内，从而生成观察。如果测试结果超出该范围，Advisor 将为集群生成观察。同时，Advisor 将创建有关如何将观察到的值恢复到最佳实践范围内的建议。Advisor 仅显示将对性能和操作有显著影响的建议。当 Advisor 确定建议已被采纳时，它将从建议列表中删除它。

例如，假定您的数据仓库包含大量未压缩的表列。在此情况下，您可以通过使用 `ENCODE` 参数指定列压缩来重建表，从而节省集群存储成本。又例如，假设 Advisor 观察到集群将大量数据包含在未压缩的表数据中。在此情况下，它将为您提供 SQL 代码块来查找表列（作为描述如何压缩这些列的压缩和资源的候选项）。

Amazon Redshift 区域

Amazon Redshift Advisor 功能仅在以下 AWS 区域中可用：

- 美国东部（弗吉尼亚北部）区域 (us-east-1)
- 美国东部（俄亥俄）区域 (us-east-2)
- 美国西部（加利福尼亚北部）区域 (us-west-1)
- 美国西部（俄勒冈州）区域 (us-west-2)
- 非洲（开普敦）区域 (af-south-1)
- 亚太地区（香港）区域 (ap-east-1)
- 亚太（海得拉巴）区域（ap-south-2）
- 亚太地区（雅加达）区域 (ap-southeast-3)
- 亚太地区（墨尔本）区域（ap-southeast-4）
- 亚太地区（孟买）区域 (ap-south-1)
- 亚太地区（大阪）区域 (ap-northeast-3)
- 亚太地区（首尔）区域 (ap-northeast-2)
- 亚太地区（新加坡）区域 (ap-southeast-1)
- 亚太地区（悉尼）区域 (ap-southeast-2)
- 亚太地区（东京）区域 (ap-northeast-1)
- 加拿大（中部）区域 (ca-central-1)
- 加拿大西部（卡尔加里）区域 (ca-west-1)
- 中国（北京）区域 (cn-north-1)
- 中国（宁夏）区域 (cn-northwest-1)
- 欧洲（法兰克福）区域 (eu-central-1)
- 欧洲（爱尔兰）区域 (eu-west-1)
- 欧洲（伦敦）区域 (eu-west-2)
- 欧洲（米兰）区域 (eu-south-1)
- 欧洲（巴黎）区域 (eu-west-3)
- 欧洲（西班牙）区域（eu-south-2）
- 欧洲地区（斯德哥尔摩）区域 (eu-north-1)
- 欧洲（苏黎世）区域（eu-central-2）
- 以色列（特拉维夫）区域（il-central-1）

- 中东（巴林）区域 (me-south-1)
- 中东（阿联酋）区域（me-central-1）
- 南美洲（圣保罗）区域 (sa-east-1)

主题

查看 Amazon Redshift Advisor 建议

您可以使用 Amazon Redshift 控制台、Amazon Redshift API 或 AWS CLI 查看 Amazon Redshift Advisor 建议。要查看建议，您必须拥有附加到您的 IAM 角色或身份的 `redshift:ListRecommendations` 权限。

在 Amazon Redshift 预置控制台上查看 Amazon Redshift Advisor 建议

您可以在 AWS Management Console 上查看 Amazon Redshift Advisor 建议

在控制台上查看 Amazon Redshift Advisor 针对 Amazon Redshift 集群提供的相关建议

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择 Advisor。
3. 展开每条建议以查看更多详细信息。在此页面上，您可以对建议进行排序和分组。

通过执行 Amazon Redshift API 操作查看 Amazon Redshift Advisor 建议

您可以使用 Amazon Redshift API 列出 Amazon Redshift Advisor 针对 Amazon Redshift 集群提供的相关建议。通常，您可以使用自己选择的编程语言进行开发和应用，然后使用 AWS SDK 调用 `redshift:ListRecommendations` API。有关更多信息，请参阅《Amazon Redshift API 参考》中的 ListRecommendations。

通过执行 AWS Command Line Interface 操作查看 Amazon Redshift Advisor 建议

您可以使用 AWS Command Line Interface 列出 Amazon Redshift Advisor 针对 Amazon Redshift 集群提供的相关建议。有关更多信息，请参阅《AWS CLI 命令参考》中的 list-recommendations。

Amazon Redshift Advisor 建议

Amazon Redshift Advisor 提供了有关如何优化 Amazon Redshift 集群以提高性能和节省运营成本的建议。您可以在控制台中找到每条建议的说明，如前所示。您可以在以下章节中找到有关这些建议的更多详细信息。

主题

压缩 COPY 加载的 Simple Storage Service（Amazon S3）文件对象

COPY 命令利用 Amazon Redshift 中的大规模并行处理 (MPP) 架构并行读取和加载数据。它可以从 Simple Storage Service（Amazon S3）、DynamoDB 表以及来自一个或多个远程主机的文本输出读取文件。

在加载大量数据时，强烈建议使用 COPY 命令从 S3 加载压缩数据文件。压缩大型数据集可节省将文件上载到 Amazon S3 的时间。COPY 还将在读取文件时解压缩文件，以加快加载过程的速度。

分析

加载大型未压缩数据集的耗时的 COPY 命令通常有机会获得相当大的性能改进。Advisor 分析将识别用于加载大型未压缩数据集的 COPY 命令。在这种情况下，Advisor 将生成对 Amazon S3 中的源文件实施压缩的建议。

建议

确保每个加载大量数据或运行很长时间的 COPY 都从 Amazon S3 中提取未压缩的数据对象。您可以通过以超级用户身份运行以下 SQL 命令来识别从 Amazon S3 加载大量未压缩数据集的 COPY 命令。

```
SELECT
    wq.userid, query, exec_start_time AS starttime, COUNT(*) num_files,
    ROUND(MAX(wq.total_exec_time/1000000.0),2) execution_secs,
    ROUND(SUM(transfer_size)/(1024.0*1024.0),2) total_mb,
    SUBSTRING(querytxt,1,60) copy_sql
FROM stl_s3client s
JOIN stl_query q USING (query)
JOIN stl_wlm_query wq USING (query)
WHERE s.userid>1 AND http_method = 'GET'
    AND POSITION('COPY ANALYZE' IN querytxt) = 0
    AND aborted = 0 AND final_state='Completed'
GROUP BY 1, 2, 3, 7
HAVING SUM(transfer_size) = SUM(data_size)
AND SUM(transfer_size)/(1024*1024) >= 5
ORDER BY 6 DESC, 5 DESC;
```

如果暂存的数据在您加载后保留在 Amazon S3 中（通常位于数据湖架构中），以压缩形式存储此数据可以降低存储成本。

实施提示

- 在压缩后，理想的对象大小在 1–128 MB 之间。
- 您可以使用 gzip、lzop 或 bzip2 格式压缩文件。

隔离多个活动数据库

作为最佳实践，我们建议将 Amazon Redshift 中的数据库与其他数据库隔离。查询在特定数据库中运行且无法访问集群上的任何其他数据库中的数据。但是，您在集群的所有数据库中运行的查询共用同一基础集群存储空间和计算资源。当单个集群包含多个活动数据库时，这些数据库的工作负载通常不相关。

分析

Advisor 分析将审查集群上的所有数据库中是否有在同一时间运行的活动工作负载。如果存在在同一时间运行的活动工作负载，Advisor 将生成考虑将数据库迁移到独立的 Amazon Redshift 集群的建议。

建议

考虑将每个主动查询的数据库移动到独立的专用集群。使用独立的集群可减少资源争用和提高查询性能。之所以如此，是因为它使您能够为每个集群设置大小，以满足每个工作负载的存储、成本和性能需求。此外，不相关的工作负载经常受益于不同的工作负载管理配置。

要识别哪些数据库是主动使用的，您可以作为超级用户运行此 SQL 命令。

```
SELECT database,
  COUNT(*) as num_queries,
  AVG(DATEDIFF(sec,starttime,endtime)) avg_duration,
  MIN(starttime) as oldest_ts,
  MAX(endtime) as latest_ts
FROM stl_query
WHERE userid > 1
GROUP BY database;
```

实施提示

- 由于用户必须专门连接到各个数据库，并且查询只能访问单个数据库，因此将数据库移动到独立集群对用户的影响最小。
- 移动数据库一个方式是执行以下步骤：
  1. 将当前集群的快照暂时还原到相同大小的集群。
  2. 从新集群中删除除目标数据库之外的所有数据库。
  3. 将集群的大小调整为合适的节点类型并针对数据库的工作负载进行计数。

重新分配工作负载管理 (WLM) 内存

Amazon Redshift 将用户查询路由至 实施手动 WLM 以进行处理。工作负载管理 (WLM) 将定义这些查询路由至队列的方式。Amazon Redshift 为每个队列分配一部分集群可用内存。队列的内存在队列的查询槽间分配。

当某个队列所配置的插槽数多于工作负载需要的插槽数时，分配给这些未使用的插槽的内存将得不到充分利用。通过将配置的插槽数减少得与峰值工作负载需求匹配，您可以将未充分利用的内存重新分配到活动插槽，并可以提高查询性能。

分析

Advisor 分析将审查工作负载并发需求以识别具有未使用的插槽的查询队列。当发现以下内容时，Advisor 将生成一个减少队列中的插槽数的建议：

- 在整个分析过程中具有并非完全不活动的插槽的队列。
- 在整个分析过程中具有超过四个插槽（其中至少有两个为非活动插槽）的队列。

建议

通过将配置的插槽数减少得与峰值工作负载需求匹配，您可以将未充分利用的内存重新分配到活动插槽。请考虑为插槽从未得到充分利用的队列减少配置的插槽计数。要识别这些队列，您可以通过作为超级用户运行以下 SQL 命令来比较每个队列的峰值每小时插槽需求。

```
WITH
 generate_dt_series AS (select sysdate - (n * interval '5 second') as dt from (select
 row_number() over () as n from stl_scan limit 17280)),
 apex AS (
     SELECT iq.dt, iq.service_class, iq.num_query_tasks, count(iq.slot_count) as
 service_class_queries, sum(iq.slot_count) as service_class_slots
     FROM
         (select gds.dt, wq.service_class, wscc.num_query_tasks, wq.slot_count
         FROM stl_wlm_query wq
         JOIN stv_wlm_service_class_config wscc ON (wscc.service_class =
 wq.service_class AND wscc.service_class > 5)
         JOIN generate_dt_series gds ON (wq.service_class_start_time <= gds.dt AND
 wq.service_class_end_time > gds.dt)
         WHERE wq.userid > 1 AND wq.service_class > 5) iq
     GROUP BY iq.dt, iq.service_class, iq.num_query_tasks),
     maxes as (SELECT apex.service_class, trunc(apex.dt) as d, date_part(h,apex.dt) as
 dt_h, max(service_class_slots) max_service_class_slots
                     from apex group by apex.service_class, apex.dt,
 date_part(h,apex.dt))
SELECT apex.service_class - 5 AS queue, apex.service_class, apex.num_query_tasks AS
 max_wlm_concurrency, maxes.d AS day, maxes.dt_h || ':00 - ' || maxes.dt_h || ':59' as
 hour, MAX(apex.service_class_slots) as max_service_class_slots
FROM apex
JOIN maxes ON (apex.service_class = maxes.service_class AND apex.service_class_slots =
 maxes.max_service_class_slots)
GROUP BY  apex.service_class, apex.num_query_tasks, maxes.d, maxes.dt_h
ORDER BY apex.service_class, maxes.d, maxes.dt_h;
```

max_service_class_slots 列表示该小时内查询队列中的 WLM 查询插槽的最大数量。如果存在未充分利用的队列，请通过修改参数组来实施插槽减少优化，如《Amazon Redshift 管理指南》中所述。

实施提示

- 如果工作负载在卷中高度可变，请确保分析捕获了峰值利用率期间。如果没有，请重复运行上述 SQL 以监控峰值并发需求。
- 有关解释来自上述 SQL 代码的查询结果的更多详细信息，请参阅 GitHub 上的 wlm_apex_hourly.sql 脚本。

在 COPY 期间跳过压缩分析

当您将数据加载到具有使用 COPY 命令声明的压缩编码的空表中时，Amazon Redshift 将应用存储压缩。此优化确保了即使在由最终用户加载时，集群中的数据也能被高效存储。应用压缩所需的分析可能需要大量时间。

分析

Advisor 分析将检查是否有被自动压缩分析延迟的 COPY 操作。该分析通过对加载中的数据进行采样来确定压缩编码。此采样类似于由 ANALYZE COMPRESSION 命令执行的采样。

当您将数据作为结构化流程的一部分加载时（例如在夜间提取、转换、加载 (ETL) 批处理中），您可以预先定义压缩。您还可以优化表定义以永久跳过此阶段而不会造成任何负面影响。

建议

要通过跳过压缩分析阶段来提高 COPY 响应能力，请实施以下两个选项之一：

- 在创建您要使用 COPY 命令加载的任何表时使用列 ENCODE 参数。
- 通过在 COPY 命令中应用 COMPUPDATE OFF 参数来完全禁用压缩。

最佳解决方案通常是在表创建期间使用列编码，因为此方法还保留了在磁盘上存储压缩数据的好处。您可以使用 ANALYZE COMPRESSION 命令建议压缩编码，但您必须重新创建表以应用这些编码。要自动执行此流程，您可以使用在 GitHub 上找到的 AWSColumnEncodingUtility。

要识别触发了自动压缩分析的最新 COPY 操作，请运行以下 SQL 命令。

```
WITH xids AS (
```

```
    SELECT xid FROM stl_query WHERE userid>1 AND aborted=0
    AND querytxt = 'analyze compression phase 1' GROUP BY xid
    INTERSECT SELECT xid FROM stl_commit_stats WHERE node=-1)
SELECT a.userid, a.query, a.xid, a.starttime, b.complyze_sec,
    a.copy_sec, a.copy_sql
FROM (SELECT q.userid, q.query, q.xid, date_trunc('s',q.starttime)
    starttime, substring(querytxt,1,100) as copy_sql,
    ROUND(datediff(ms,starttime,endtime)::numeric / 1000.0, 2) copy_sec
    FROM stl_query q JOIN xids USING (xid)
    WHERE (querytxt ilike 'copy %from%' OR querytxt ilike '% copy %from%')
    AND querytxt not like 'COPY ANALYZE %') a
LEFT JOIN (SELECT xid,
    ROUND(sum(datediff(ms,starttime,endtime))::numeric / 1000.0,2) complyze_sec
    FROM stl_query q JOIN xids USING (xid)
    WHERE (querytxt like 'COPY ANALYZE %'
    OR querytxt like 'analyze compression phase %')
    GROUP BY xid ) b ON a.xid = b.xid
WHERE b.complyze_sec IS NOT NULL ORDER BY a.copy_sql, a.starttime;
```

实施提示

- 确保在 ETL 过程中创建的所有大尺寸表（例如，暂存表和临时表）声明了除第一个排序键之外的所有列。
- 估计正在为由前面的 SQL 命令标识的每个 COPY 命令加载的表的预计生命周期长短。如果您确信该表仍然非常小，请使用 COMPUPDATE OFF 参数完全禁用压缩。否则，请在使用 COPY 命令加载表之前使用显式压缩创建表。

拆分 COPY 加载的 Simple Storage Service（Amazon S3）对象

COPY 命令利用 Amazon Redshift 中的大规模并行处理 (MPP) 架构在 Simple Storage Service（Amazon S3）上的文件中读取和加载数据。COPY 命令从多个文件并行加载数据，向集群中的节点划分工作负载。要实现最佳吞吐量，我们强烈建议您将数据拆分成多个文件，以便利用并行处理。

分析

Advisor 分析可识别用于加载在 Amazon S3 中暂存的少量文件中包含的大型数据集的 COPY 命令。加载来自若干文件的大型数据集的耗时的 COPY 命令通常有机会获得相当大的性能改进。当 Advisor 发现这些 COPY 命令需要大量时间时，它会提出相关建议，以通过将数据拆分为 Amazon S3 中的额外文件来提高并行度。

建议

在这种情况下，我们建议执行以下操作（按优先顺序列出）：

1. 优化加载的文件数比集群节点数更少的 COPY 命令。
2. 优化加载的文件数比集群切片数更少的 COPY 命令。
3. 优化这样的 COPY 命令：其中的文件数不是集群切片数的倍数。

某些 COPY 命令加载大量数据或运行很长时间。对于这些命令，我们建议您在从 Amazon S3 加载大量数据对象时，其数量应等于集群中的切片数的倍数。要识别 COPY 命令已加载的 S3 对象数，请以超级用户身份运行以下 SQL 代码。

```
SELECT
    query, COUNT(*) num_files,
    ROUND(MAX(wq.total_exec_time/1000000.0),2) execution_secs,
    ROUND(SUM(transfer_size)/(1024.0*1024.0),2) total_mb,
    SUBSTRING(querytxt,1,60) copy_sql
FROM stl_s3client s
JOIN stl_query q USING (query)
JOIN stl_wlm_query wq USING (query)
WHERE s.userid>1 AND http_method = 'GET'
    AND POSITION('COPY ANALYZE' IN querytxt) = 0
    AND aborted = 0 AND final_state='Completed'
GROUP BY query, querytxt
HAVING (SUM(transfer_size)/(1024*1024))/COUNT(*) >= 2
ORDER BY CASE
WHEN COUNT(*) < (SELECT max(node)+1 FROM stv_slices) THEN 1
WHEN COUNT(*) < (SELECT COUNT(*) FROM stv_slices WHERE node=0) THEN 2
ELSE 2+((COUNT(*) % (SELECT COUNT(*) FROM stv_slices))/(SELECT COUNT(*)::DECIMAL FROM
 stv_slices))
END, (SUM(transfer_size)/(1024.0*1024.0))/COUNT(*) DESC;
```

实施提示

- 节点中的切片数取决于集群的节点大小。有关各种节点类型中切片数的更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 中的集群和节点。

- 您可以通过指定一个通用前缀（对于集合，则为前缀键），或通过在清单文件中明确列出文件，从而加载多个文件。有关加载文件的更多信息，请参阅从压缩和未压缩文件中加载数据。
- Amazon Redshift 在拆分工作负载时不会考虑文件大小。拆分您的加载数据文件，使文件大小大约相等，压缩后的文件大小介于 1 MB 和 1 GB 之间。

更新表统计数据

Amazon Redshift 使用基于成本的查询优化程序为查询选择最佳执行计划。成本估算基于使用 ANALYZE 命令收集的表统计数据。当统计数据过时或丢失时，数据库可能会选择一个效率较低的查询执行计划，尤其是对于复杂的查询。保存最新统计数据可帮助复杂的查询在尽可能短的时间内运行。

分析

Advisor 分析可跟踪其统计数据已过时或丢失的表。它将审查与复杂查询关联的表访问元数据。如果使用复杂模式频繁访问的表缺少统计数据，Advisor 将创建关键建议以运行 ANALYZE。如果使用复杂模式频繁访问的表具有过时的统计数据，Advisor 将创建启发式建议以运行 ANALYZE。

建议

每当表内容发生重大变化时，请使用 ANALYZE 更新统计数据。每当使用 COPY 或 INSERT 命令将大量新数据行加载到现有表中时，我们建议运行 ANALYZE。每当使用 UPDATE 或 DELETE 命令修改大量行时，我们也建议运行 ANALYZE。要识别具有缺失或过时的统计数据的表，请以超级用户身份运行以下 SQL 命令。结果将按表的大小顺序排列。

要识别具有缺失或过时的统计数据的表，请以超级用户身份运行以下 SQL 命令。结果将按表的大小顺序排列。

```
SELECT
   ti.schema||'.'||ti."table" tablename,
   ti.size table_size_mb,
   ti.stats_off statistics_accuracy
 FROM svv_table_info ti
 WHERE ti.stats_off > 5.00
 ORDER BY ti.size DESC;
```

实施提示

默认 ANALYZE 阈值为 10%。此默认值意味着，如果自上次 ANALYZE 之后，给定表有不到 10% 的行发生了更改，则 ANALYZE 命令将跳过此表。因此，您可以选择在每个 ETL 流程结束时发出 ANALYZE 分析。采用此方法意味着经常会跳过 ANALYZE，但也确保了 ANALYZE 在需要时运行。

ANALYZE 统计数据对在联接中使用的列（例如，`JOIN tbl_a ON col_b`）或作为谓词的列（例如，`WHERE col_b = 'xyz'`）影响最大。默认情况下，ANALYZE 将收集指定的表中的所有列的统计数据。如果需要，您可以通过仅对受 ANALYZE 影响最大的列运行 ANALYZE 来减少运行该命令所需的时间。您可以运行以下 SQL 命令来识别要用作谓词的列。您还可以通过指定 ANALYZE PREDICATE COLUMNS 来让 Amazon Redshift 选择要分析的列。

```
WITH predicate_column_info as (
SELECT ns.nspname AS schema_name, c.relname AS table_name, a.attnum as col_num,
 a.attname as col_name,
        CASE
            WHEN 10002 = s.stakind1 THEN array_to_string(stavalues1, '||')
            WHEN 10002 = s.stakind2 THEN array_to_string(stavalues2, '||')
            WHEN 10002 = s.stakind3 THEN array_to_string(stavalues3, '||')
            WHEN 10002 = s.stakind4 THEN array_to_string(stavalues4, '||')
            ELSE NULL::varchar
        END AS pred_ts
   FROM pg_statistic s
   JOIN pg_class c ON c.oid = s.starelid
   JOIN pg_namespace ns ON c.relnamespace = ns.oid
   JOIN pg_attribute a ON c.oid = a.attrelid AND a.attnum = s.staattnum)
SELECT schema_name, table_name, col_num, col_name,
       pred_ts NOT LIKE '2000-01-01%' AS is_predicate,
       CASE WHEN pred_ts NOT LIKE '2000-01-01%' THEN (split_part(pred_ts,
 '||',1))::timestamp ELSE NULL::timestamp END as first_predicate_use,
       CASE WHEN pred_ts NOT LIKE '%||2000-01-01%' THEN (split_part(pred_ts,
 '||',2))::timestamp ELSE NULL::timestamp END as last_analyze
FROM predicate_column_info;
```

有关更多信息，请参阅 分析表。

启用短查询加速

短查询加速 (SQA) 让选定的短时查询优先于长时查询。SQA 在专用空间中运行短时查询，因此 SQA 查询不会被迫排在队列中的长时查询后面等待。SQA 仅优先处理用户定义的队列中的短时查询。使用 SQA，短时查询会更快地开始运行，用户会更快地看到结果。

如果您开启 SQA，则可以减少或消除专用于运行短查询的工作负载管理（WLM）队列。此外，长时查询无需与短查询竞争队列中的插槽，因此您可以将 WLM 队列配置为使用较少的查询插槽。当您使用较

低的并发度时，查询吞吐量会增加，而且大多数工作负载的总体系统性能会得到提高。有关更多信息，请参阅 使用短查询加速。

分析

Advisor 检查工作负载模式，并报告 SQA 可以减少延迟的最近查询的数量以及符合 SQA 条件的查询的每日队列时间。

建议

修改 WLM 配置以开启 SQA。Amazon Redshift 使用机器学习算法分析每个有资格的查询。预测质量会随着 SQA 从您的查询模式中学习而改进。有关更多信息，请参阅配置工作负载管理。

当您开启 SQA 时，默认情况下 WLM 会将短查询的最大运行时间设置为动态的。我们建议保留 SQA 最大运行时间的动态设置。

实施提示

要检查是否开启了 SQA，请运行以下查询。如果查询返回一行内容，则说明 SQA 已开启。

```
select * from stv_wlm_service_class_config
where service_class = 14;
```

有关更多信息，请参阅 监控 SQA。

修改表上的分配键

Amazon Redshift 会根据表分配方式在整个集群中分配表中的行。具有 KEY 分配的表需要一个列充当分配键 (DISTKEY)。表中的行会根据其 DISTKEY 列值分配给集群的节点分片。

适当的 DISTKEY 会在每个节点分片上放置相似数量的行，并会经常在联接条件中引用。当表在 DISTKEY 列上联接时，会发生优化联接，从而加快查询性能。

分析

Advisor 会分析您集群的工作负载，以确定表中可从 KEY 分配方式显著获益的最适当的分配键。

建议

Advisor 提供 ALTER TABLE 语句，此类语句根据其分析结果来更改表的 DISTSTYLE 和 DISTKEY。要实现显著的性能优势，请确保实施建议组中的所有 SQL 语句。

使用 ALTER TABLE 重新分配大型表会占用集群资源，并且需要在不同时间进行临时表锁定。在其他集群工作负载较轻时实施每个建议组。有关优化表分配属性的更多详细信息，请参阅 Amazon Redshift 工程高级表设计手册：分配方式和分配键。

有关 ALTER DISTSYLE 和 DISTKEY 的更多信息，请参阅 ALTER TABLE。

> **Note**
>
> 如果您没有看到建议，这并不一定意味着当前的分配方式是最合适的。当数据不足或重新分配的预期好处很小时，Advisor 不会提供建议。
> Advisor 建议适用于特定的表，就算包含同名列的表也不一定适用。除非表中的数据相同，否则就算多个表共享一个列名，表与这些列也可具有不同的特性。
> 如果您看到针对 ETL 作业所创建或删除的暂存表的建议，请修改 ETL 过程以使用 Advisor 建议的分配键。

修改表上的排序键

Amazon Redshift 根据表排序键对表行进行排序。表行的排序基于排序键列值。

通过要求从磁盘读取较少的表块，可以使用适当的排序键对表进行排序，从而可以提高查询性能，尤其是那些具有范围受限制的谓词的查询。

分析

Advisor 会分析您的集群在几天内的工作负载，以便为您的表确定一个有优势的排序键。

建议

Advisor 提供两组 ALTER TABLE 语句，此语句根据其分析更改表的排序键：

- 一组语句，用于更改当前没有排序键的表以添加 COMPOUND 排序键。
- 一组语句，用于将排序键从 INTERLEAVED 更改为 COMPOUND 或者没有排序键。

  使用复合排序键可以显著降低维护开销。具有复合排序键的表不需要昂贵的 VACUUM REINDEX 操作，这些操作对于交错排序是必需的。在实践中，对于绝大多数 Amazon Redshift 工作负载，复合排序键比交错排序键更有效。但是，如果表很小，那么不使用排序键以避免产生排序键存储开销会更有效。

使用 ALTER TABLE 对大型表进行排序时，会消耗集群资源，并且在不同的时间需要表锁。当集群的工作负载适中时，实施每个建议。有关优化表排序键配置的更多详细信息，请参阅 Amazon Redshift 工程高级表设计手册：复合和交错排序键。

有关 ALTER SORTKEY 的更多信息，请参阅 ALTER TABLE。

> **Note**
>
> 如果您没有看到关于表的建议，这并不一定意味着当前配置是最好的。当没有足够的数据或排序的预期好处很小时，Advisor 不会提供建议。
> Advisor 建议适用于特定的表，就算包含相同名称和数据类型的列的表也不一定适用。根据表中的数据和工作负载，共享列名的表可以有不同的建议。

更改列的压缩编码

压缩是可缩减数据存储大小的列级操作。Amazon Redshift 使用压缩功能，通过减少磁盘输入/输出量来节省存储空间并提高查询性能。我们建议根据每列的数据类型和查询模式对其进行最佳压缩编码。通过最佳压缩，查询可以更高效地运行，并且数据库占用的存储空间最少。

分析

Advisor 不断对集群的工作负载和数据库 schema 进行分析，以确定每个表列的最佳压缩编码。

建议

Advisor 提供 ALTER TABLE 语句，此语句根据其分析更改特定列的压缩编码。

使用 ALTER TABLE 更改列压缩编码占用集群资源，并且需要在不同时间进行表锁定。最好在集群工作负载较轻时实施建议。

作为参考，ALTER TABLE 示例 显示了更改列编码的几个语句。

> **Note**
>
> 当没有足够的数据或更改编码的预期好处很小时，Advisor 不会提供建议。

数据类型建议

Amazon Redshift 有一个用于各种使用案例的 SQL 数据类型库。这些包括整数类型，例如 INT 以及存储字符的类型，例如 VARCHAR。Redshift 以优化的方式存储类型，以提供快速访问和良好的查询性

能。此外，Redshift 还提供了针对特定类型的函数，您可以使用这些函数对查询结果进行格式化或执行计算。

分析

Advisor 持续对集群的工作负载和数据库架构进行分析，以确定可从数据类型更改中显著受益的列。

建议

Advisor 提供了使用建议的数据类型添加新列的 ALTER TABLE 语句。随附的 UPDATE 语句会将数据从现有列复制到新列。创建新列并加载数据后，请更改查询和摄入脚本以访问新列。然后利用在 SQL 函数参考中找到的专门针对新数据类型的功能和函数。

将现有数据复制到新列可能需要时间。当集群的工作负载较轻时，我们建议您实施每个 Advisor 建议。请在数据类型引用可用数据类型的列表。

请注意，当没有足够的数据或更改数据类型的预期好处很小时，Advisor 不会提供建议。

Amazon Redshift 的教程

按照这些教程中的步骤了解 Amazon Redshift 功能。

- 教程：从 Amazon S3 加载数据
- 教程：使用 Amazon Redshift Spectrum 查询嵌套数据
- 教程：配置手动工作负载管理 (WLM) 队列
- 教程：将空间 SQL 函数与 Amazon Redshift 配合使用
- Amazon Redshift ML 的教程

使用自动表优化

自动表优化是一种自我调整功能，可通过应用排序键和分配键自动优化表的设计，而无需管理员干预。通过使用自动化功能来调整表的设计，您可以快速入手并获得最快的性能，而无需花费时间手动调整和实施表优化。

自动表优化不断观察查询与表的交互方式。它使用高级人工智能方法选择排序键和分配键，以优化集群工作负载的性能。如果 Amazon Redshift 确定应用键可以提高集群性能，则表将在创建集群之日起数小时内自动更改，对查询的影响最小。

为了充分利用此自动化功能，Amazon Redshift 管理员会创建一个新表，或者更改现有表以使其能够使用自动优化。具有分配样式或排序键 AUTO 的现有表已启用自动化功能。当您对这些表运行查询时，Amazon Redshift 会确定排序键或分配键是否能够提高性能。如果能，Amazon Redshift 会自动修改表，而无需管理员干预。如果运行了最少数量的查询，则会在启动集群后的几小时内应用优化。

如果 Amazon Redshift 确定分配键可以提高查询的性能，则分配样式为 AUTO 的表可以将其分配样式更改为 KEY。

主题

启用自动表优化

预设情况下，未显式定义排序键或分配键时创建的表将设置为 AUTO。在创建表时，您还可以手动显式设置排序键或分配键。如果您设置了排序键或分配键，则不会自动管理该表。

要使现有表能够自动优化，请使用 ALTER 语句选项将表更改为 AUTO。您可以选择为排序键定义自动化，但不能为分配键定义自动化（反之亦然）。如果运行 ALTER 语句将表转换为自动表，则会保留现有的排序键和分配样式。

```
ALTER TABLE table_name ALTER SORTKEY AUTO;
```

```
ALTER TABLE table_name ALTER DISTSTYLE AUTO;
```

有关更多信息，请参阅 ALTER TABLE。

最初，表没有分配键或排序键。分配样式设置为 EVEN 或 ALL，具体取决于表格大小。随着表大小的增大，Amazon Redshift 会应用最佳分配键和排序键。在运行最少数量的查询之后，将在数小时内应用优化。确定排序键优化时，Amazon Redshift 会尝试在表扫描期间优化从磁盘读取的数据块。在确定分配样式优化时，Amazon Redshift 会尝试优化在集群节点之间传输的字节数。

从表中删除自动表优化

您可以从自动优化中删除表。从自动化中删除表包括选择排序键或分配样式。要更改分配样式，请指定特定的分配样式。

```
ALTER TABLE table_name ALTER DISTSTYLE EVEN;
```

```
ALTER TABLE table_name ALTER DISTSTYLE ALL;
```

```
ALTER TABLE table_name ALTER DISTSTYLE KEY DISTKEY c1;
```

要更改排序键，可以定义排序键或选择无。

```
ALTER TABLE table_name ALTER SORTKEY(c1, c2);
```

```
ALTER TABLE table_name ALTER SORTKEY NONE;
```

监控自动表优化的操作

系统视图 SVV_ALTER_TABLE_RECOMMENDATIONS 记录了当前针对表的 Amazon Redshift Advisor 建议。此视图显示针对所有表的建议，包括为自动优化定义的建议和非为自动优化定义的建议。

要查看某张表是否定义为自动优化，请查询系统视图 SVV_TABLE_INFO。条目仅针对当前会话数据库中可见的表显示。每天两次将建议插入视图中，从创建集群后的数小时内开始。建议可用后，将在一小时内开始。在（Amazon Redshift 或您）应用建议后，该建议将不再显示在视图中。

系统视图 SVL_AUTO_WORKER_ACTION 显示了 Amazon Redshift 执行的所有操作的审计日志，以及表的先前状态。

系统视图 SVV_TABLE_INFO 列出了系统中的所有表，以及一个列，用于指示表的排序键和分配样式是否设置为 AUTO。

有关这些系统视图的更多信息，请参阅系统监控（仅已预置）。

使用列压缩

压缩是可缩减数据存储大小的列级操作。压缩能够节约存储空间并减少从存储读取的数据大小，这种方法可以减少磁盘 I/O 量，因此可提高查询性能。

ENCODE AUTO 是表的默认设置。将表设置为 ENCODE AUTO 时，Amazon Redshift 会自动管理表中所有列的压缩编码。有关更多信息，请参阅CREATE TABLE和ALTER TABLE。

但是，如果为表中的任何列指定压缩编码，则表不再设置为 ENCODE AUTO。Amazon Redshift 不再自动管理表中所有列的压缩编码。

当您创建表时，您可以将压缩类型或编码手动应用到表中的列。或者，您可以使用 COPY 命令来自动分析和应用压缩。有关更多信息，请参阅 让 COPY 选择压缩编码。有关应用自动压缩的详细信息，请参阅使用自动压缩加载表。

> **Note**
>
> 我们强烈建议使用 COPY 命令应用自动压缩。

如果新表具有与另一个表相同的数据特征，则可以选择手动应用压缩编码。或者，如果您在测试时发现在自动压缩期间应用的压缩编码不太适合您的数据，就可以这样操作。如果您选择手动应用压缩编码，则可以对已填充的表运行 ANALYZE COMPRESSION 命令并根据其结果选择压缩编码。

要手动应用压缩，您可以在 CREATE TABLE 语句中为各个列指定压缩编码。语法如下所示。

```
CREATE TABLE table_name (column_name
data_type ENCODE encoding-type)[, ...]
```

其中，encoding-type 取自下一部分中的关键字表。

例如，下面的语句会创建一个包含两列的表 PRODUCT。将数据加载到表中后，PRODUCT_ID 列未压缩，但 PRODUCT_NAME 列使用字节词典编码 (BYTEDICT) 压缩。

```
create table product(
product_id int encode raw,
product_name char(20) encode bytedict);
```

您可以使用 ALTER TABLE 命令在向表中添加列时指定其编码。

```
ALTER TABLE table-name ADD [ COLUMN ] column_name column_type ENCODE encoding-type
```

主题

压缩编码

c压缩编码指定在向表中添加行时应用到数据值列的压缩类型。

ENCODE AUTO 是表的默认设置。将表设置为 ENCODE AUTO 时，Amazon Redshift 会自动管理表中所有列的压缩编码。有关更多信息，请参阅CREATE TABLE和ALTER TABLE。

但是，如果为表中的任何列指定压缩编码，则表不再设置为 ENCODE AUTO。Amazon Redshift 不再自动管理表中所有列的压缩编码。

使用 CREATE TABLE 时，为表中的任何列指定压缩编码时会禁用 ENCODE AUTO。如果禁用了 ENCODE AUTO，Amazon Redshift 会自动为您未指定 ENCODE 类型的列分配压缩编码，如下所示：

- 为定义为排序键的列分配 RAW 压缩。
- 为定义为 BOOLEAN、REAL 或 DOUBLE PRECISION 数据类型的列分配 RAW 压缩。
- 定义为 SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIMESTAMP 或 TIMESTAMPTZ 数据类型的列分配了 AZ64 压缩。
- 定义为 CHAR 或 VARCHAR 数据类型的列分配了 LZO 压缩。

在创建表后，您可以使用 ALTER TABLE 更改表的编码。如果您使用 ALTER TABLE 禁用 ENCODE AUTO，Amazon Redshift 将不再自动管理列的压缩编码。所有列都将保留您禁用 ENCODE AUTO 时的压缩编码类型，直到您更改它们或再次启用 ENCODE AUTO。

下表列出了支持的压缩编码和支持该编码的数据类型。

编码类型	CREATE TABLE 和 ALTER TABLE 中的关键字	数据类型
Raw（无压缩）	RAW	所有
AZ64	AZ64	SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIMESTAMP、TIMESTAMPTZ
字节词典	BYTEDICT	SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、CHAR、VARCHAR、DATE、TIMESTAMP、TIMESTAMPTZ
Delta	DELTA DELTA32K	SMALLINT、INT、BIGINT、DATE、TIMESTAMP、DECIMAL INT、BIGINT、DATE、TIMESTAMP、DECIMAL
LZO	LZO	SMALLINT、INTEGER、BIGINT、DECIMAL、CHAR、VARCHAR、DATE、TIMESTAMP、TIMESTAMPTZ、SUPER
Mostlyn	MOSTLY8 MOSTLY16 MOSTLY32	SMALLINT、INT、BIGINT、DECIMAL INT、BIGINT、DECIMAL BIGINT、DECIMAL
Run-length	RUNLENGTH	SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION

编码类型	CREATE TABLE 和 ALTER TABLE 中的关键字	数据类型
		、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP、TIMESTAMPTZ
文本	TEXT255 TEXT32K	仅 VARCHAR 仅 VARCHAR
Zstandard	ZSTD	SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP、TIMESTAMPTZ、SUPER

原始编码

原始编码是指定为排序键的列和定义为 BOOLEAN、REAL 或 DOUBLE PRECISION 数据类型的列的默认编码。使用原始编码时，数据以原始、未压缩的形式存储。

AZ64 编码

AZ64 是 Amazon 的专有压缩编码算法，旨在实现高压缩率和改进的查询处理。AZ64 算法的核心是压缩较小的数据值组，并使用单指令多数据 (SIMD) 指令实现并行处理。使用 AZ64 为数字、日期和时间数据类型实现显著的存储节省和高性能。

使用带有以下数据类型的 CREATE TABLE 和 ALTER TABLE 语句定义列时，您可以使用 AZ64 作为压缩编码：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- DATE
- TIMESTAMP

- TIMESTAMPTZ

字节词典编码

在字节词典编码中，会为磁盘上的每个列值数据块创建单独的唯一值词典。（一个 Amazon Redshift 磁盘数据块占用 1 MB 的空间。）词典最多包含 256 个单字节值，它们作为原始数据值的索引存储。如果单个数据块中存储的值多于 256 个，则多余的值会以原始、未压缩的形式写入该数据块中。针对每个磁盘数据块重复这一过程。

此编码对低基数字符串列非常有效。如果列的数据域少于 256 个唯一值，则该编码是最佳的编码。

如果列使用通过 BYTEDICT 编码的字符串数据类型（CHAR 和 VARCHAR），Amazon Redshift 会直接对压缩数据进行矢量化扫描和谓词评估。这些扫描使用特定于硬件的单指令和多数据 (SIMD) 指令实现并行处理。这大大加快了扫描字符串列的速度。当 CHAR/VARCHAR 列包含长字符串时，字节词典编码的空间效率尤其高。

假设某个表包含数据类型为 CHAR(30) 的 COUNTRY 列。在加载数据时，Amazon Redshift 会创建词典并使用索引值填充 COUNTRY 列。该词典包含已建立索引的唯一值，而表本身则只包含相应值的下标（只占一个字节）。

> **Note**
>
> 固定长度字符列存储有尾随空格。因此，对 CHAR(30) 列使用字节词典编码时，每个压缩值可节约 29 个字节的存储空间。

下表列出了 COUNTRY 列的词典。

唯一数据值	词典索引	大小（固定长度，每个值 30 个字节）
England	0	30
United States of America	1	30
Venezuela	2	30
Sri Lanka	3	30

唯一数据值	词典索引	大小（固定长度，每个值 30 个字节）
Argentina	4	30
Japan	5	30
总计		180

下表列出了 COUNTRY 列中的值。

原始数据值	原始大小（固定长度，每个值 30 个字节）	压缩值（索引）	新的大小（字节）
England	30	0	1
England	30	0	1
United States of America	30	1	1
United States of America	30	1	1
Venezuela	30	2	1
Sri Lanka	30	3	1
Argentina	30	4	1
Japan	30	5	1
Sri Lanka	30	3	1
Argentina	30	4	1
总计	300		10

本示例中的总计压缩大小按照如下方式计算：词典中存储着 6 个不同的条目 (6 * 30 = 180)，表包含 10 个 1 字节的压缩值，总共占用 190 个字节。

增量编码

对于日期时间列，增量编码非常有用。

增量编码通过记录列中各个相邻值之间的差异来压缩数据。这种差异记录在磁盘上每个列值数据块的单独词典中。（一个 Amazon Redshift 磁盘数据块占用 1 MB 的空间。）例如，假设列中包含 10 个从 1 到 10 的整数。第一个存储为 4 字节整数（加上 1 字节标记）。接下来的九个都存储为值为 1 的字节，表示它比前一个值大 1。

增量编码有两个版本：

- DELTA 使用 1 字节的值（8 位整数）记录差值。
- DELTA32K 使用 2 字节值（16 位整数）记录差值。

如果列中的大多数值可用单字节压缩，则 1 字节的变化会非常高效。但是，如果增量较大，则在最坏情况下，这种编码会比存储未压缩数据的效率还低。相似的逻辑也适用于 16 位版本。

如果两个值的差值超出 1 字节范围 (DELTA) 或 2 字节范围 (DELTA32K)，则存储完整的原始值，并包含一个前置的 1 字节标志。1 字节的范围为 -127 到 127；2 字节的范围为 -32K 到 32K。

下表显示了对数值列应用增量编码的原理。

原始数据值	原始大小（字节）	差值（增量）	压缩值	压缩大小（字节）
1	4		1	1+4（标志 + 实际值）
5	4	4	4	1
50	4	45	45	1
200	4	150	150	1+4（标志 + 实际值）
185	4	-15	-15	1

原始数据值	原始大小（字节）	差值（增量）	压缩值	压缩大小（字节）
220	4	35	35	1
221	4	1	1	1
总计	28			15

LZO 编码

LZO 编码提供非常高的压缩率，且具有不错的性能。LZO 编码特别适合用于存储非常长的字符串的 CHAR 和 VARCHAR 列。对于自由格式文本（如产品描述、用户评论或 JSON 字符串），它们特别适合使用。

Mostly 编码

当列的数据类型大于大多数存储值所需时，Mostly 编码会很有用。通过为此类型的列指定 Mostly 编码，您可以将该列中的大部分值压缩成较小的标准存储大小。无法压缩的剩余值以原始形式存储。例如，您可以将 16 位列（如 INT2 列）压缩为 8 位存储。

一般说来，Mostly 编码适用于以下数据类型：

- SMALLINT/INT2（16 位）
- INTEGER/INT（32 位）
- BIGINT/INT8（64 位）
- DECIMAL/NUMERIC（64 位）

请选择适当的 Mostly 编码版本，以契合列的数据类型大小。例如，对于 16 位整数列，可以应用 MOSTLY8。不允许对 16 位数据类型列应用 MOSTLY16 或对 32 位数据类型列应用 MOSTLY32。

当列中相对较多的值无法压缩时，大部分编码的效率可能比不压缩的效率还低。在将其中一个编码应用于列之前，请执行检查。您打算现在加载（以及未来可能加载）的大多数值应该适合下表所示的范围。

编码	压缩的存储大小	可压缩的值的范围（超出范围的值以原始形式存储）
MOSTLY8	1 字节（8 位）	-128 到 127

编码	压缩的存储大小	可压缩的值的范围（超出范围的值以原始形式存储）
MOSTLY16	2 字节（16 位）	-32768 到 32767
MOSTLY32	4 字节（32 位）	-2147483648 到 +2147483647

> **Note**
>
> 对于小数值，忽略小数点以确定该值是否适合范围。例如，将 1234.56 视为 123456，因此，可压缩到 MOSTLY32 列中。

例如，VENUE 表中的 VENUEID 列定义为原始整数列，这意味着其值占用 4 个字节的存储。但是，该列中值的当前范围为 **0** 到 **309**。因此，为 VENUEID 使用 MOSTLY16 编码并重新创建和重新加载该表可将该列中每个值的存储缩减为 2 个字节。

如果另一个表引用的 VENUEID 值主要处于 0 到 127 的范围内，则可以将该外键列编码为 MOSTLY8。在做出选择之前，请对引用表数据运行多个查询，以确定大多数值是处于 8 位、16 位还是 32 位范围内。

下表显示了使用 MOSTLY8、MOSTLY16 和 MOSTLY32 编码时特定数值的压缩大小：

原始值	原始 INT 或 BIGINT 大小（字节）	MOSTLY8 压缩大小（字节）	MOSTLY16 压缩大小（字节）	MOSTLY32 压缩大小（字节）
1	4	1	2	4
10	4	1	2	4
100	4	1	2	4
1000	4	与原始数据大小相同	2	4
10000	4		2	4
20000	4		2	4

原始值	原始 INT 或 BIGINT 大小（字节）	MOSTLY8 压缩大小（字节）	MOSTLY16 压缩大小（字节）	MOSTLY32 压缩大小（字节）
40000	8		与原始数据大小相同	4
100000	8			4
2000000000	8			4

运行长度编码

运行长度编码将连续反复出现的值替换为一个包含该值及连续出现次数（连续出现长度）的令牌。系统会为磁盘上的每个列数值数据块创建单独的唯一值词典。（一个 Amazon Redshift 磁盘数据块占用 1 MB 的空间。）该编码最适合数据值经常反复连续出现的表，例如，当表按这些值排序时。

例如，假设某个大型维度表中的列具有可预测的小型域，如可能值不足 10 个的 COLOR 列。即使没有对数据进行排序，这些值很可能会在整个表中按长序列排列。

我们不建议对指定为排序键的任何列应用运行长度编码。当数据块中包含相似的行数时，范围限制扫描可更好地执行。如果排序键列的压缩率远高于同一查询中的其他列，则范围限制扫描的执行效率可能会很差。

下表使用 COLOR 列的示例来讲解运行长度编码的工作原理。

原始数据值	原始大小（字节）	压缩值（标记）	压缩大小（字节）
Blue	4	{2,Blue}	5
Blue	4		0
Green	5	{3,Green}	6
Green	5		0
Green	5		0
Blue	4	{1,Blue}	5
Yellow	6	{4,Yellow}	7

原始数据值	原始大小（字节）	压缩值（标记）	压缩大小（字节）
Yellow	6		0
Yellow	6		0
Yellow	6		0
总计	51		23

Text255 和 Text32k 编码

Text255 和 text32k 编码对于压缩经常出现相同单词的 VARCHAR 列非常有用。系统会为磁盘上的每个列数值数据块创建单独的唯一单词词典。（一个 Amazon Redshift 磁盘数据块占用 1 MB 的空间。）词典包含列中前 245 个唯一的单词。在磁盘上，这些单词会被替换成代表这 245 个值之一的 1 字节索引值，词典中未表示的任意单词以未压缩方式存储。这一过程会针对每个 1MB 磁盘数据块重复执行。如果索引词在列中的出现频度很高，则该列会产生较高的压缩率。

text32k 编码的原理也是如此，但每个数据块的词典并不捕获特定数量的单词。该词典为其找到的每个唯一单词编制索引，直到组合条目达到 32K 的长度（减去一些开销）。索引值用两个字节存储。

例如，我们来考虑一下 VENUE 表中的 VENUENAME 列。该列中重复出现 **Arena**、**Center**、**Theatre** 之类的单词，且很可能出现在每个数据块中的前 245 个单词中（如果应用 text255 压缩的话）。如果是这样，该列将受益于压缩。因为每当这些单词出现时，它们将只占用 1 字节的存储空间（而不是相对应的 5、6 或 7 个字节）。

Zstandard 编码

Zstandard (ZSTD) 编码提供高压缩率，而且在不同数据集中都有极佳性能。ZSTD 特别适用于存储多种长、短字符串（例如产品描述、用户评论、日志和 JSON 字符串）的 CHAR 和 VARCHAR 列。其中一些算法（例如 Delta 编码或 Mostly 编码）有可能比无压缩使用更多存储空间，ZSTD 不太可能增大磁盘使用率。

ZSTD 支持 SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP 和 TIMESTAMPTZ 数据类型。

测试压缩编码

如果您决定手动指定列编码，则您可能需要针对自己的数据测试不同的编码。

> **Note**
>
> 我们建议您尽可能使用 COPY 命令加载数据，并允许 COPY 命令根据您的数据选择最佳的编码。或者，您也可以使用 ANALYZE COMPRESSION 命令查看推荐针对现有数据采用的编码。有关应用自动压缩的详细信息，请参阅使用自动压缩加载表。

要执行有意义的数据压缩测试，您必须有大量的行。在本示例中，我们将使用从两个表 VENUE 和 LISTING 中进行选择的语句来创建表并插入行。我们将省略通常会联接两个表的 WHERE 子句。结果是 VENUE 表中的每行都联接到 LISTING 表中的所有行（总计超过 3200 万行）。这称作笛卡尔联接，通常不建议使用。但是，对于本例，这是创建多个行的简便方法。如果您现有的表包含想要测试的数据，则您可以跳过这一步。

在我们有一个包含示例数据的表格后，我们创建一个包含七列的表格。每个表格都有不同的压缩编码：raw、bytedict、lzo、run length、text255、text32k 和 zstd。我们通过运行从第一个表选择数据的 INSERT 命令，用完全相同的数据填充每一列。

要测试压缩编码，请执行以下操作：

1. （可选）首先，使用笛卡尔联接创建一个包含大量行的表。如果您想要测试现有的表，则请跳过这一步。

```
create table cartesian_venue(
venueid smallint not null distkey sortkey,
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer);

insert into cartesian_venue
select venueid, venuename, venuecity, venuestate, venueseats
from venue, listing;
```

2. 接下来，使用您要进行比较的编码创建表。

```
create table encodingvenue (
venueraw varchar(100) encode raw,
venuebytedict varchar(100) encode bytedict,
venuelzo varchar(100) encode lzo,
venuerunlength varchar(100) encode runlength,
venuetext255 varchar(100) encode text255,
```

```
venuetext32k varchar(100) encode text32k,
venuezstd varchar(100) encode zstd);
```

3. 使用带 SELECT 子句的 INSERT 语句将相同的数据插入到所有列中。

```
insert into encodingvenue
select venuename as venueraw, venuename as venuebytedict, venuename as venuelzo,
 venuename as venuerunlength, venuename as  venuetext32k, venuename as  venuetext255,
 venuename as venuezstd
from cartesian_venue;
```

4. 验证新表中的行数。

```
select count(*) from encodingvenue

  count
----------
 38884394
(1 row)
```

5. 查询 STV_BLOCKLIST 系统表，比较每列使用的 1MB 磁盘数据块的数量。

   MAX 聚合函数返回每列的最大数据块数。STV_BLOCKLIST 表包含三个系统生成的列的详细信息。本示例在 WHERE 子句中使用 `col < 6` 来排除系统生成的列。

```
select col, max(blocknum)
from stv_blocklist b, stv_tbl_perm p
where (b.tbl=p.id) and name ='encodingvenue'
and col < 7
group by name, col
order by col;
```

   查询返回以下结果。列从零开始编号。根据集群的配置方式，您的结果可能具有不同的编号，但相对大小应相差不大。对于该数据集，您会看到第二列上的 BYTEDICT 编码能够产生最佳结果。这种方法的压缩比优于 20:1。LZO 和 ZSTD 编码也生成了极佳的结果。当然，不同的数据集会导致不同的结果。当列包含较长的文本字符串时，LZO 往往能够产生最佳的压缩结果。

```
 col | max
-----+-----
   0 | 203
   1 |  10
   2 |  22
```

```
   3 | 204
   4 |  56
   5 |  72
   6 |  20
(7 rows)
```

如果您现有的表中包含数据，则您可以使用 ANALYZE COMPRESSION 命令查看针对该表的建议编码。例如，下面的示例显示了对于 VENUE 表的副本 CARTESIAN_VENUE（包含 3800 万行数据）建议的编码。注意，ANALYZE COMPRESSION 为 VENUENAME 列推荐 LZO 编码。ANALYZE COMPRESSION 基于多个因素选择最佳压缩，包括减少百分比。在这种特定情况下，BYTEDICT 提供更好的压缩，但 LZO 也会生成大于 90% 的压缩。

```
analyze compression cartesian_venue;

Table          | Column     | Encoding | Est_reduction_pct
---------------+------------+----------+------------------
reallybigvenue | venueid    | lzo      | 97.54
reallybigvenue | venuename  | lzo      | 91.71
reallybigvenue | venuecity  | lzo      | 96.01
reallybigvenue | venuestate | lzo      | 97.68
reallybigvenue | venueseats | lzo      | 98.21
```

示例：为 CUSTOMER 表选择压缩编码

下面的语句可创建一个包含不同数据类型列的 CUSTOMER 表。此 CREATE TABLE 语句显示了适用于这些列的众多压缩编码可能组合中的一个。

```
create table customer(
custkey int encode delta,
custname varchar(30) encode raw,
gender varchar(7) encode text255,
address varchar(200) encode text255,
city varchar(30) encode text255,
state char(2) encode raw,
zipcode char(5) encode bytedict,
start_date date encode delta32k);
```

下表显示了为 CUSTOMER 表选择的列编码并解释了如此选择的理由：

列	数据类型	编码	说明
CUSTKEY	int	增量	CUSTKEY 由唯一、连续的整数值组成。差值只是一个字节，因此，DELTA 是很好的选择。
CUSTNAME	varchar(30)	raw	CUSTNAME 拥有包含少量重复值的大型域。任何压缩编码可能都无效。
GENDER	varchar(7)	text255	GENDER 是一个包含许多重复值的小型域。Text255 非常适合用于重复出现相同单词的 VARCHAR 列。
ADDRESS	varchar(200)	text255	ADDRESS 是一个大型域，但包含许多重复单词，如 Street Avenue、North、South 等。Text 255 和 text 32k 对于压缩重复出现相同单词的 VARCHAR 列很有用。列长度较短，因此，text255 是很好的选择。
CITY	varchar(30)	text255	CITY 是一个包含部分重复值的大型域。某些城市名称较其他城市名称常用得多。Text255 是很好的选择，

列	数据类型	编码	说明
			理由与 ADDRESS 相同。
STATE	char(2)	raw	在美国，STATE 是一个由 50 个双字符值组成的精确域。Bytedict 编码能够产生一定的压缩效果，但由于列大小只有两个字符，压缩可能还抵不上解压数据产生的开销。
ZIPCODE	char(5)	bytedict	ZIPCODE 是由不足 50000 个唯一值组成的已知域。某些邮政编码较其他邮政编码常用得多。当列包含有限数量的唯一值时，Bytedict 编码非常有效。
START_DATE	date	delta32k 编码	增量编码对于日期时间列非常有用，特别是当行以日期顺序加载时。

使用数据分配样式

将数据加载到表中时，Amazon Redshift 根据表的分配方式将表中的行分配到各个计算节点。在运行查询时，查询优化程序根据执行任何联接和聚合的需要将行重新分配到计算节点。选择表分配方式的目的是，在执行查询前找到数据必须处于的位置，从而最大程度地减小重新分配步骤的影响。

> **Note**
>
> 本部分将介绍 Amazon Redshift 数据库中的数据分配原则。我们建议您使用 DISTSTYLE AUTO 创建表。如果您这样做，则 Amazon Redshift 会使用自动表优化来选择数据分配方式。有关更多信息，请参阅 使用自动表优化。本节的其余部分提供了有关分配方式的详细信息。

主题

数据分配概念

遵循 Amazon Redshift 的一些数据分配概念。

节点和切片

Amazon Redshift 集群是一组节点。集群中的每个节点拥有自己的操作系统、专用内存和专用磁盘存储。有个节点是领导节点，它负责将数据和查询处理任务分配到计算节点。计算节点提供完成这些任务的资源。

计算节点的磁盘存储分成一系列切片。每个节点的切片数取决于集群的节点大小。所有节点均参与并行查询的运行，处理尽可能跨切片均匀分布的数据。有关每个节点大小拥有的切片数的更多信息，请转到《Amazon Redshift 管理指南》中的关于集群和节点。

数据重新分配

将数据加载到表中时，Amazon Redshift 根据表的分配方式将表中的行分配到各个节点切片。作为查询计划的一部分，优化程序确定必须将数据块放置在何处，以最好地运行查询。然后，在查询运行时，数

据被实际移动或重新分配。重新分配可能涉及将特定的行发送到节点以进行联接，或将整个表广播到所有节点。

数据重新分配可能占到查询计划成本的一大部分，其产生的网络流量可能会影响其他数据库操作并降低整个系统的性能。在切实可行的范围内，预测最佳的数据初始放置位置可以尽可能地减少数据分配的影响。

数据分配目标

将数据加载到表中时，Amazon Redshift 根据您在创建表时选择的分配方式将表中的行分配到计算节点和切片。数据分配有两个主要目标：

- 将工作负载均匀地分配到集群中的节点上。不均匀的分配或数据分配偏斜会导致某些节点执行的工作比其他节点多，从而影响查询性能。
- 在查询运行时，尽量减少数据移动。如果参与联接或聚合的行已在节点上与其在其他表中的联接行并置，则优化程序在查询执行期间不必重新分配过多的数据。

您为数据库选择的分配策略会对查询性能、存储需求、数据加载和维护产生重大影响。通过为每张表选择最佳的分配方式，您可以均衡数据分配并显著提高整个系统的性能。

分配方式

创建表时，您可以指定以下分配方式之一：AUTO、EVEN、KEY 或 AUTO。

如果未指定分配方式，Amazon Redshift 使用自动 (AUTO) 分配。

AUTO 分配

在自动 (AUTO) 分配方式下，Amazon Redshift 基于表数据大小分配最佳分配方式。例如，如果指定 AUTO 分配方式，Amazon Redshift 最初向小型表指定的是 ALL 分配方式。当表变大时，Amazon Redshift 可能会将分配方式更改为 KEY，选择主键（或复合主键的列）作为分配键。如果表变大且没有任何一列适合用作分配键，Amazon Redshift 会将分配方式更改为 EVEN。分配方式的更改在后台进行，对用户查询的影响极小。

要查看 Amazon Redshift 自动执行的更改表分配键的操作，请参阅SVL_AUTO_WORKER_ACTION。要查看有关更改表分配键的当前建议，请参阅SVV_ALTER_TABLE_RECOMMENDATIONS。

要查看应用于表的分配方式，请查询 PG_CLASS_INFO 系统目录视图。有关更多信息，请参阅 查看分配方式。如果未使用 CREATE TABLE 语句指定分配方式，Amazon Redshift 将应用自动 (AUTO) 分配。

EVEN 分配

不管任意特定列中的值是什么，领导节点都以轮询方式向所有切片分配行。当表不参与联接时，适合使用 EVEN 分配。当 KEY 分配和 ALL 分配之间没有明确的选择时，这种方法也很适合。

KEY 分配

根据一列中的值分配行。领导节点会将匹配的值放置到同一个节点切片上。如果基于联接键分配一对表，领导节点会根据联接列中的值在切片上并置行。这样一来，共同列的匹配值将实际存储在一起。

ALL 分配

向每个节点分配整个表的副本。EVEN 分配或 KEY 分配只将表中的部分行放置在每个节点上，而 ALL 分配则确保为该表参与的所有联接并置每一行。

ALL 分配需要以集群中节点数量为倍数的存储空间，因此，它需要长得多的时间来加载、更新或插入数据到多个表中。ALL 分配只适用于移动相对缓慢的表；即更新不频繁、不广泛的表。由于在查询过程中重新分配小表的成本很低，因此将小维度表定义为 DISTSTYLE ALL 没有显著好处。

> **Note**
>
> 当您为某个列指定分配方式后，Amazon Redshift 会在集群级别处理数据分配。Amazon Redshift 不需要、也不支持数据库对象中的数据分区概念。您不需要创建表空间或为表定义分区方案。

在特定场景中，表创建完毕后，您可更改其分配方式。有关更多信息，请参阅 ALTER TABLE。对于在创建表之后无法更改其分配方式的场景，您可以重新创建表并使用深层复制来填充新表。有关更多信息，请参阅执行深层复制。

查看分配方式

要查看表的分配方式，请查询 PG_CLASS_INFO 视图或 SVV_TABLE_INFO 视图。

PG_CLASS_INFO 中的 RELEFFECTIVEDISTSTYLE 列指示表的当前分配方式。如果表使用自动分配，则 RELEFFECTIVEDISTSTYLE 为 10、11 或 12，这指示有效分配方式是 AUTO (ALL)、AUTO (EVEN) 或 AUTO (KEY)。如果表使用自动分配，则分配方式最初可能显示 AUTO (ALL)，然后在表增大时更改为 AUTO (EVEN) 或 AUTO (KEY)。

下表的 RELEFFECTIVEDISTSTYLE 列中提供了每个值的分配方式：

RELEFFECTIVEDISTSTYLE	当前分配方式
0	EVEN
1	KEY
8	ALL
10	AUTO (ALL)
11	AUTO (EVEN)
12	AUTO (KEY)

SVV_TABLE_INFO 中的 DISTSTYLE 列指示表的当前分配方式。如果表使用自动分配，则 DISTSTYLE 为 AUTO (ALL)、AUTO (EVEN) 或 AUTO (KEY)。

下面的示例使用三种分配方式和自动分配创建了四个表，然后查询 SVV_TABLE_INFO 以查看这些分配方式。

```
create table public.dist_key (col1 int)
diststyle key distkey (col1);

insert into public.dist_key values (1);

create table public.dist_even (col1 int)
diststyle even;

insert into public.dist_even values (1);

create table public.dist_all (col1 int)
diststyle all;

insert into public.dist_all values (1);

create table public.dist_auto (col1 int);

insert into public.dist_auto values (1);

select "schema", "table", diststyle from SVV_TABLE_INFO
```

```
where "table" like 'dist%';

        schema    |    table         | diststyle
     -------------+------------------+------------
      public      | dist_key         | KEY(col1)
      public      | dist_even        | EVEN
      public      | dist_all         | ALL
      public      | dist_auto        | AUTO(ALL)
```

评估查询模式

选择分配方式只是数据库设计的一个方面。请在整个系统的情境内考虑分配方式，并将分配与集群大小、压缩编码方法、排序键、表约束等其他重要因素相权衡。

请用尽可能接近真实情况的数据测试系统。

要做出良好的分配方式选择，您必须了解自己的 Amazon Redshift 应用程序的查询模式。找出您系统中代价最为高昂的查询，并使您的初始数据库设计基于这些查询的需求。确定查询总成本的因素有包括：查询需要运行多长时间以及查询需要消耗多少计算资源。确定查询成本的其他因素包括：查询的运行频率以及查询对于其他查询和数据库操作的破坏性。

找出成本最高的查询使用的表，评估其在查询运行时中扮演的角色。考虑表的联接和聚合方式。

使用本部分中的准则为每个表选择分配方式。完成上述操作后，创建这些表，为它们加载尽可能接近真实情况的数据。然后针对这些表测试您希望使用的查询类型。您可以评估查询解释计划，以识别优化机会。比较加载时间、存储空间和查询运行时，以平衡您系统的整体需求。

指定分配方式

本部分中指定分配方式的注意事项和建议使用了星型 schema 作为示例。您的数据库设计可能基于星型 schema、星型 schema 的变体或完全不同的 schema。Amazon Redshift 旨在与您选择的任何 schema 设计高效协作。本部分中的原则适用于任何设计 schema。

1. 为所有表指定主键和外键。

   Amazon Redshift 不强制实施主键和外键约束，但查询优化程序在生成查询计划时会使用它们。如果您设置了主键和外键，则您的应用程序必须维护这些键的有效性。

2. 根据共同列分配事实数据表及其最大的维度表。

   根据参与最常见联接的数据集的大小（而不只是表的大小）选择最大的维度。如果某张表常使用 WHERE 子句筛选，则只有其部分行参与联接。与贡献更多数据的较小的表相比，此类表对重新分

配的影响较小。将维度表的主键和事实数据表对应的外键均指定为 DISTKEY。如果多个表使用相同的分配键，则它们也会与事实数据表并置。事实数据表只能有一个分配键。任何通过其他键联接的表都不能与事实数据表并置。

3. 为其他维度表指定分配键。

   根据主键或外键分配表，具体视它们与其他表最常见的联接方式而定。

4. 评估是否将某些维度表更改为使用 ALL 分配。

   如果一个维度表不能与事实数据表或其他重要的联接表并置，您可以通过将整个表分配到所有节点来大大提高查询性能。使用 ALL 分配会使存储空间需求成倍增长，并且会增加加载时间和维护操作，所以在选择 ALL 分配前应权衡所有因素。下面的部分讲解如何通过评估 EXPLAIN 计划来确定 ALL 分配的候选项。

5. 为剩余的表使用 AUTO 分配。

   如果表大体上为非规范化且不参与联接，或如果您无法明确选择采用其他分配方式，请使用 AUTO 分配。

要让 Amazon Redshift 选择适当的分配方式，请不要明确指定分配方式。

评估查询计划

您可以使用查询计划确定优化分配方式的候选项。

做出初始设计决策后，创建您的表，为它们加载数据，然后测试它们。请使用尽可能接近真实情况的测试数据集。测量加载时间，以用作比较的基准。

评估具有代表性的查询（可代表您希望执行的代价最为高昂的查询）；具体地说，就是使用联接和聚合的查询。比较各设计选项的运行时。如果要比较运行时，请不要将查询的首次执行计算在内，因为第一个运行时包含编译时间。

DS_DIST_NONE

无需重新分配，因为相应的切片已在计算节点上并置。通常，您只有一个 DS_DIST_NONE 步骤：事实数据表与一个维度表之间的联接。

DS_DIST_ALL_NONE

无需重新分配，因为内部联接表使用 DISTSTYLE ALL。每个节点上都有整个表。

DS_DIST_INNER

内部表重新分配。

DS_DIST_OUTER

外部表重新分配。

DS_BCAST_INNER

将整个内部表的副本广播到所有计算节点。

DS_DIST_ALL_INNER

整个内部表重新分配到单个切片，因为外部表使用 DISTSTYLE ALL。

DS_DIST_BOTH

两个表都重新分配。

DS_DIST_NONE 和 DS_DIST_ALL_NONE 都很好。它们表示该步骤无需分配，因为所有联接都已并置。

DS_DIST_INNER 意味着该步骤的成本可能相对较高，因为需要将内部表重新分配到节点。DS_DIST_INNER 表示外部表已根据联接键正确分配。将内部表的分配键设为联接键，以将此转换为 DS_DIST_NONE。在某些情况下，无法根据联接键分配内部表，因为没有根据联接键分配外部表。如果是这种情况，请评估是否对内部表使用 ALL 分配。如果表更新不频繁、不广泛，且表大到足以产生很高的重新分配成本，则将分配方式更改为 ALL 并重新测试。ALL 分配会导致加载时间增加，因此，当您重新测试时，请将加载时间包含在评估因素中。

DS_DIST_ALL_INNER 效果不佳。它表示整个内部表重新分配到单个切片（因为外部表使用 DISTSTYLE ALL），因此，各个节点上都有整个外部表的副本。这会导致在单个节点上低效地顺序执行运行时，而不是使用所有节点来充分利用并行运行时。DISTSTYLE ALL 仅适用于内部联接表。请为外部表指定分配键或使用 EVEN 分配。

DS_BCAST_INNER 和 DS_DIST_BOTH 效果都不好。通常，此类重新分配在表未根据其分配键联接时出现。如果事实数据表还没有分配键，则指定联接列作为这两个表的分配键。如果事实数据表已有基于另一列的分配键，则评估更改分配键以并置该联接能否提高整体性能。如果更改外部表的分配键不是最佳选择，则您可以为内部表指定 DISTSTYLE ALL，从而实现并置。

以下示例显示了具有 DS_BCAST_INNER 和 DS_DIST_NONE 标签的查询计划的部分内容。

```
->  XN Hash Join DS_BCAST_INNER  (cost=112.50..3272334142.59 rows=170771 width=84)
        Hash Cond: ("outer".venueid = "inner".venueid)
```

```
        ->  XN Hash Join DS_BCAST_INNER  (cost=109.98..3167290276.71 rows=172456
 width=47)
              Hash Cond: ("outer".eventid = "inner".eventid)
              ->  XN Merge Join DS_DIST_NONE  (cost=0.00..6286.47 rows=172456 width=30)
                    Merge Cond: ("outer".listid = "inner".listid)
                    ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497
 width=14)
                    ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=24)
```

将维度表更改为使用 DISTSTYLE ALL 后，相同查询的查询计划显示 DS_DIST_ALL_NONE 而不是 DS_BCAST_INNER。此外，联接步骤的相对成本也会有极大的改变。与上一个查询中的 3272334142.59 相比，总成本为 14142.59。

```
->  XN Hash Join DS_DIST_ALL_NONE  (cost=112.50..14142.59 rows=170771 width=84)
        Hash Cond: ("outer".venueid = "inner".venueid)
        ->  XN Hash Join DS_DIST_ALL_NONE  (cost=109.98..10276.71 rows=172456 width=47)
              Hash Cond: ("outer".eventid = "inner".eventid)
              ->  XN Merge Join DS_DIST_NONE  (cost=0.00..6286.47 rows=172456 width=30)
                    Merge Cond: ("outer".listid = "inner".listid)
                    ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497
 width=14)
                    ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=24)
```

查询计划示例

本示例演示如何评估查询计划以找到优化分配的机会。

使用 EXPLAIN 命令运行下面的查询以生成查询计划。

```
explain
select lastname, catname, venuename, venuecity, venuestate, eventname,
month, sum(pricepaid) as buyercost, max(totalprice) as maxtotalprice
from category join event on category.catid = event.catid
join venue on venue.venueid = event.venueid
join sales on sales.eventid = event.eventid
join listing on sales.listid = listing.listid
join date on sales.dateid = date.dateid
join users on users.userid = sales.buyerid
group by lastname, catname, venuename, venuecity, venuestate, eventname, month
having sum(pricepaid)>9999
order by catname, buyercost desc;
```

在 TICKIT 数据库中，SALES 是事实数据表，LISTING 是其最大的维度。为并置这些表，SALES 根据 LISTID（LISTING 的外键）分配，LISTING 根据其主键 LISTID 分配。下面的示例显示了 SALES 和 LISTING 的 CREATE TABLE 命令。

```
create table sales(
 salesid integer not null,
 listid integer not null distkey,
 sellerid integer not null,
 buyerid integer not null,
 eventid integer not null encode mostly16,
 dateid smallint not null,
 qtysold smallint not null encode mostly8,
 pricepaid decimal(8,2) encode delta32k,
 commission decimal(8,2) encode delta32k,
 saletime timestamp,
 primary key(salesid),
 foreign key(listid) references listing(listid),
 foreign key(sellerid) references users(userid),
 foreign key(buyerid) references users(userid),
 foreign key(dateid) references date(dateid))
        sortkey(listid,sellerid);

create table listing(
 listid integer not null distkey sortkey,
 sellerid integer not null,
 eventid integer not null encode mostly16,
 dateid smallint not null,
 numtickets smallint not null encode mostly8,
 priceperticket decimal(8,2) encode bytedict,
 totalprice decimal(8,2) encode mostly32,
 listtime timestamp,
 primary key(listid),
 foreign key(sellerid) references users(userid),
 foreign key(eventid) references event(eventid),
 foreign key(dateid) references date(dateid));
```

在下面的查询计划中，基于 SALES 和 LISTING 的联接的合并联接步骤显示 DS_DIST_NONE，表明该步骤无需重新分配。但是，上移查询计划，另一个内部联接显示 DS_BCAST_INNER，表明内部表作为查询执行的一部分广播。使用键分配只能并置一对表，因此，五个表必须重新广播。

```
QUERY PLAN
XN Merge  (cost=1015345167117.54..1015345167544.46 rows=1000 width=103)
```

```
 Merge Key: category.catname, sum(sales.pricepaid)
 ->  XN Network  (cost=1015345167117.54..1015345167544.46 rows=170771 width=103)
       Send to leader
       ->  XN Sort  (cost=1015345167117.54..1015345167544.46 rows=170771 width=103)
             Sort Key: category.catname, sum(sales.pricepaid)
             ->  XN HashAggregate  (cost=15345150568.37..15345152276.08 rows=170771
width=103)
                   Filter: (sum(pricepaid) > 9999.00)
                    ->  XN Hash Join DS_BCAST_INNER  (cost=742.08..15345146299.10
rows=170771 width=103)
                          Hash Cond: ("outer".catid = "inner".catid)
                          ->  XN Hash Join DS_BCAST_INNER
(cost=741.94..15342942456.61 rows=170771 width=97)
                                Hash Cond: ("outer".dateid = "inner".dateid)
                                ->  XN Hash Join DS_BCAST_INNER
(cost=737.38..15269938609.81 rows=170766 width=90)
                                      Hash Cond: ("outer".buyerid = "inner".userid)
                                      ->  XN Hash Join DS_BCAST_INNER
(cost=112.50..3272334142.59 rows=170771 width=84)
                                            Hash Cond: ("outer".venueid =
"inner".venueid)
                                            ->  XN Hash Join DS_BCAST_INNER
(cost=109.98..3167290276.71 rows=172456 width=47)
                                                  Hash Cond: ("outer".eventid =
"inner".eventid)
                                                  ->  XN Merge Join DS_DIST_NONE
(cost=0.00..6286.47 rows=172456 width=30)
                                                        Merge Cond: ("outer".listid =
"inner".listid)
                                                        ->  XN Seq Scan on listing
(cost=0.00..1924.97 rows=192497 width=14)
                                                        ->  XN Seq Scan on sales
(cost=0.00..1724.56 rows=172456 width=24)
                                                  ->  XN Hash  (cost=87.98..87.98
rows=8798 width=25)
                                                        ->  XN Seq Scan on event
(cost=0.00..87.98 rows=8798 width=25)
                                            ->  XN Hash  (cost=2.02..2.02 rows=202
width=41)
                                                  ->  XN Seq Scan on venue
(cost=0.00..2.02 rows=202 width=41)
                                      ->  XN Hash  (cost=499.90..499.90 rows=49990
width=14)
```

```
                                              ->  XN Seq Scan on users
(cost=0.00..499.90 rows=49990 width=14)
                                  ->  XN Hash  (cost=3.65..3.65 rows=365 width=11)
                                        ->  XN Seq Scan on date  (cost=0.00..3.65
rows=365 width=11)
                            ->  XN Hash  (cost=0.11..0.11 rows=11 width=10)
                                  ->  XN Seq Scan on category  (cost=0.00..0.11 rows=11
width=10)
```

一种解决方案是将这些表更改为使用 DISTSTYLE ALL。

```
ALTER TABLE users ALTER DISTSTYLE ALL;
ALTER TABLE venue ALTER DISTSTYLE ALL;
ALTER TABLE category ALTER DISTSTYLE ALL;
ALTER TABLE date ALTER DISTSTYLE ALL;
ALTER TABLE event ALTER DISTSTYLE ALL;
```

再次使用 EXPLAIN 运行相同的查询，并检查新的查询计划。现在，联接显示 DS_DIST_ALL_NONE，表明无需重新分配（因为数据已通过 DISTSTYLE ALL 分配到每一个节点）。

```
QUERY PLAN
XN Merge  (cost=1000000047117.54..1000000047544.46 rows=1000 width=103)
  Merge Key: category.catname, sum(sales.pricepaid)
  ->  XN Network  (cost=1000000047117.54..1000000047544.46 rows=170771 width=103)
        Send to leader
        ->  XN Sort  (cost=1000000047117.54..1000000047544.46 rows=170771 width=103)
              Sort Key: category.catname, sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=30568.37..32276.08 rows=170771 width=103)
                    Filter: (sum(pricepaid) > 9999.00)
                    ->  XN Hash Join DS_DIST_ALL_NONE  (cost=742.08..26299.10
rows=170771 width=103)
                          Hash Cond: ("outer".buyerid = "inner".userid)
                          ->  XN Hash Join DS_DIST_ALL_NONE  (cost=117.20..21831.99
rows=170766 width=97)
                                Hash Cond: ("outer".dateid = "inner".dateid)
                                ->  XN Hash Join DS_DIST_ALL_NONE
(cost=112.64..17985.08 rows=170771 width=90)
                                      Hash Cond: ("outer".catid = "inner".catid)
                                      ->  XN Hash Join DS_DIST_ALL_NONE
(cost=112.50..14142.59 rows=170771 width=84)
                                            Hash Cond: ("outer".venueid =
"inner".venueid)
```

```
                                              ->  XN Hash Join DS_DIST_ALL_NONE
(cost=109.98..10276.71 rows=172456 width=47)
                                                    Hash Cond: ("outer".eventid =
"inner".eventid)
                                                    ->  XN Merge Join DS_DIST_NONE
(cost=0.00..6286.47 rows=172456 width=30)
                                                          Merge Cond: ("outer".listid =
"inner".listid)
                                                          ->  XN Seq Scan on listing
(cost=0.00..1924.97 rows=192497 width=14)
                                                          ->  XN Seq Scan on sales
(cost=0.00..1724.56 rows=172456 width=24)
                                                    ->  XN Hash  (cost=87.98..87.98
rows=8798 width=25)
                                                          ->  XN Seq Scan on event
(cost=0.00..87.98 rows=8798 width=25)
                                              ->  XN Hash  (cost=2.02..2.02 rows=202
width=41)
                                                    ->  XN Seq Scan on venue
(cost=0.00..2.02 rows=202 width=41)
                                        ->  XN Hash  (cost=0.11..0.11 rows=11 width=10)
                                              ->  XN Seq Scan on category
(cost=0.00..0.11 rows=11 width=10)
                                  ->  XN Hash  (cost=3.65..3.65 rows=365 width=11)
                                        ->  XN Seq Scan on date  (cost=0.00..3.65
rows=365 width=11)
                            ->  XN Hash  (cost=499.90..499.90 rows=49990 width=14)
                                  ->  XN Seq Scan on users  (cost=0.00..499.90 rows=49990
width=14)
```

分配示例

以下示例显示如何根据您在 CREATE TABLE 语句中定义的选项分配数据。

DISTKEY 示例

查看 TICKIT 数据库中的 USERS 表的 schema。USERID 定义为 SORTKEY 列和 DISTKEY 列：

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'users';

    column     |          type          | encoding | distkey | sortkey
---------------+------------------------+----------+---------+---------
```

```
 userid        | integer                | none     | t       |       1
 username      | character(8)           | none     | f       |       0
 firstname     | character varying(30)  | text32k  | f       |       0

...
```

USERID 是该表上分配列的良好选择。如果您查询 SVV_DISKUSAGE 系统视图，就会发现该表的分配非常均匀。列编号从零开始，因此，USERID 为第 0 列。

```
select slice, col, num_values as rows, minvalue, maxvalue
from svv_diskusage
where name='users' and col=0 and rows>0
order by slice, col;

slice| col | rows  | minvalue | maxvalue
-----+-----+-------+----------+----------
0    | 0   | 12496 | 4        | 49987
1    | 0   | 12498 | 1        | 49988
2    | 0   | 12497 | 2        | 49989
3    | 0   | 12499 | 3        | 49990
(4 rows)
```

表包含 49990 行。rows (num_values) 列显示每个切片包含大致相同的行数。minvalue 和 maxvalue 列显示每个切片上值的范围。每个切片都包含几乎整个值范围，因此，很有可能每个切片都会参与筛选用户 ID 范围的查询执行过程。

本示例演示了一个小型测试系统上的分配。切片的总数通常会高得多。

如果您通常使用 STATE 列联接或组合，则您可能会选择根据 STATE 列分配。以下示例显示您使用与 USERS 表相同的数据创建一个新表，但将 DISTKEY 设为 STATE 列的情况。在这种情况下，分配不平均。切片 0（13587 行）的行数比切片 3（10150 行）多约 30%。在大得多的表中，这种程度的分配偏斜可能会对查询处理产生负面影响。

```
create table userskey distkey(state) as select * from users;

select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'userskey' and col=0 and rows>0
order by slice, col;

slice | col | rows  | minvalue | maxvalue
------+-----+-------+----------+----------
    0 |   0 | 13587 |        5 |    49989
```

```
    1 |   0 | 11245 |        2 |    49990
    2 |   0 | 15008 |        1 |    49976
    3 |   0 | 10150 |        4 |    49986
(4 rows)
```

DISTSTYLE EVEN 示例

如果您使用与 USERS 表相同的数据创建一个新表，但将 DISTSTYLE 设为 EVEN，则行总是在所有切片上均匀分配。

```
create table userseven diststyle even as
select * from users;

select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'userseven' and col=0 and rows>0
order by slice, col;

slice | col | rows  | minvalue | maxvalue
------+-----+-------+----------+----------
    0 |   0 | 12497 |        4 |    49990
    1 |   0 | 12498 |        8 |    49984
    2 |   0 | 12498 |        2 |    49988
    3 |   0 | 12497 |        1 |    49989
(4 rows)
```

但是，由于分配不是基于特定的列，查询处理性能可能会下降，特别是当该表联接到其他表时。联接列分配缺失通常会影响能够高效执行的联接操作类型。当两张表依据各自的联接列分配和排序时，联接、聚合和组合操作都能得到优化。

DISTSTYLE ALL 示例

如果您使用与 USERS 表相同的数据创建一个新表，但将 DISTSTYLE 设为 ALL，则所有行都分配到每个节点的第一个切片上。

```
select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'usersall' and col=0 and rows > 0
order by slice, col;

slice | col | rows  | minvalue | maxvalue
------+-----+-------+----------+----------
    0 |   0 | 49990 |        4 |    49990
    2 |   0 | 49990 |        2 |    49990
```

```
(4 rows)
```

使用排序键

> **Note**
>
> 我们建议您使用 SORTKEY AUTO 创建表。如果您这样做，则 Amazon Redshift 会使用自动表优化来选择排序键。有关更多信息，请参阅 使用自动表优化。本节的其余部分提供了有关排序顺序的详细信息。

在创建表时，您可以将其一个或多个列定义为排序键。初次向空表中加载数据时，行以排序顺序存储在磁盘上。有关排序键列的信息会被传递给查询计划程序，后者借助该信息构建能够充分利用数据排序方式的计划。有关更多信息，请参阅 CREATE TABLE。有关创建排序键时最佳实践的信息，请参阅选择最佳的排序键。

排序可实现范围限制谓词的高效处理。Amazon Redshift 将列式数据存储在 1 MB 的磁盘数据块中。每个数据块的最小值和最大值作为元数据的一部分存储。如果查询使用范围限制谓词，则查询处理器可在表扫描期间借助该最小值和最大值快速跳过大量的数据块。例如，假设某张表存储着按日期排序的五年数据，且查询指定了一个月的日期范围。在这种情况下，您可以从扫描中最高消除 98% 的磁盘数据块。如果数据未排序，则该查询就不得不扫描更多（也可能是全部）的磁盘数据块。

您可以指定复合排序键或交错排序键。当查询谓词使用了前缀（有序排序键列的子集）时，复合排序键的效率更高。交错排序键为排序键中的每个列赋予相同的权重，因此，查询谓词可以使用构成该排序键的列的任意子集（可以是任意顺序）。

要了解排序键的选择对查询性能产生的影响，请使用 EXPLAIN 命令。有关更多信息，请参阅 查询计划和执行工作流程。

要定义排序类型，请在您的 CREATE TABLE 或 CREATE TABLE AS 语句中使用 INTERLEAVED 或 COMPOUND 关键字。默认为 COMPOUND。如果使用 INSERT、UPDATE 或 DELETE 操作定期更新表，则建议使用 COMPOUND。INTERLEAVED 排序键可使用多达八个列。根据您的数据和集群大小，VACUUM REINDEX 所需的时间显著大于 VACUUM FULL 所需的时间，因为它需要执行额外的过程来分析交错排序键。对于交错表，排序和合并操作所花费的时间更长，因为与复合排序相比，交错排序可能有更多需要重新排列的行。

要查看某张表的排序键，请查询 SVV_TABLE_INFO 系统视图。

主题

多维数据布局排序（预览版）

以下是预览版中的表多维数据布局排序的预发行文档。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

> **Note**
>
> 此功能仅在使用预览集群或预览工作组时可用。要创建预览版集群，请参阅《Amazon Redshift 管理指南》中的创建预览版集群。要创建预览工作组，请参阅《Amazon Redshift 管理指南》中的创建预览工作组。

多维数据布局排序键是一种基于工作负载中重复谓词的 AUTO 排序键。如果您的工作负载具有重复谓词，那么 Amazon Redshift 可以通过将满足重复谓词的数据行放在同一位置来提高表扫描性能。多维数据布局排序键不是按照严格的列顺序存储表数据，而是通过分析工作负载中出现的重复谓词来存储数据。在一个工作负载中可以找到多个重复谓词。根据您的工作负载，这种排序键可以提高许多谓词的性能。Amazon Redshift 会自动确定是否应将这种排序键方法用于使用 AUTO 排序键定义的表。

例如，假设您有一个按列顺序排列数据的表。可能需要检查许多数据块，以确定它们是否满足工作负载中的谓词。但是，如果数据是按谓词顺序存储在磁盘上的，那么为满足查询需要扫描的数据块就会减少。在这种情况下，使用多维数据布局排序键是有益的。

要查看查询是否使用了多维数据布局键，请查看 SYS_QUERY_DETAIL 视图的 `step_attribute` 列。当值为 `multi-dimensional` 时，多维数据布局用于查询。要查看使用 AUTO 排序键定义的表是否使用了多维数据布局，请查看 SVV_TABLE_INFO 视图的 `sortkey1` 列。当值为 `padb_internal_mddl_key_col` 时，多维数据布局用于表排序键。

要防止 Amazon Redshift 使用多维数据布局排序键，请选择除 SORTKEY AUTO 之外的其他表排序键选项。有关 SORTKEY 选项的更多信息，请参阅 CREATE TABLE。

复合排序键

复合键由排序键定义中列出的所有列组成（顺序即为其排列顺序）。当查询的筛选条件应用了使用排序键前缀的条件（如筛选条件和联接）时，复合排序键最为有用。当查询只依赖于辅助排序列而不引用主列时，复合排序的性能优势会下降。COMPOUND 是默认的排序类型。

复合排序键可以加快联接、GROUP BY 和 ORDER BY 操作，以及使用 PARTITION BY 和 ORDER BY 的窗口函数。例如，当数据在联接列上分配和预先排序时，可以使用合并联接（通常快于哈希联接）。此外，复合排序键还有助于提高压缩率。

在您向已包含数据的排序表中添加行的同时，未排序区域也随之增长，这会对性能产生显著的影响。当表使用交错排序时，特别是当排序列包含单调递增数据（如日期或时间戳列）时，这种影响会更大。定期运行 VACUUM 操作，特别是在加载大量数据后，以重新排序和重新分析数据。有关更多信息，请参阅 管理未排序区域的大小。在执行 vacuum 操作以重新排序数据后，最好运行 ANALYZE 命令以更新查询计划程序的统计元数据。有关更多信息，请参阅 分析表。

交错排序键

交错排序为排序键中的每个列或列的子集赋予相同的权重。如果多个查询使用不同的列作为筛选条件，则通常可以使用交错排序方式来提高这些查询的性能。当查询对辅助排序列使用限制性谓词时，与复合排序相比，交错排序可显著提高查询的性能。

> **Important**
>
> 不要在具有单调递增属性的列（例如，身份列、日期或时间戳）上使用交错排序键。

您应将实施交错排序键获得的性能提升与增加的负载和 vacuum 次数进行权衡。

交错排序对于高选择性查询（在 WHERE 子句中对一个或多个排序键列进行筛选的查询，如 `select c_name from customer where c_region = 'ASIA'`）最为有效。交错排序的优势随着受限制排序列数量的增加而增大。

交错排序对于大型表更为有效。排序应用于每个切片。因此，当某张表大到足以使每个切片占用多个 1 MB 数据块时，交错排序最为有效。在这里，查询处理器可以使用限制性谓词跳过大部分数据块。要查看表使用的数据块数，请查询 STV_BLOCKLIST 系统视图。

对单一列进行排序时，如果该列的值拥有较长的共同前缀，则交错排序的性能要优于复合排序。如都以“http://www”打头的 URL。复合排序键使用前缀中有限数量的字符，因此会产生大量的重复键。交错排序为区域映射值使用了内部压缩方案，使它们能够更好地区分具有较长共同前缀的列值。

当将 Amazon Redshift 预调配的集群迁移到 Amazon Redshift Serverless 时，Redshift 将具有交错排序键和 DISTSTYLE KEY 的表转换为复合排序键。DISTSTYLE 不会变化。有关分配方式的更多信息，请参阅使用数据分配方式。

VACUUM REINDEX

在您向已包含数据的排序表中不断添加行的过程中，性能会逐渐下降。复合排序和交错排序都会出现这种性能下降，但交错表受到的影响更大。VACUUM 可恢复排序顺序，但对于交错表，该操作可能需要花费更长的时间，因为合并新的交错数据可能涉及到修改每一个数据块。

初次加载表时，Amazon Redshift 会分析值在排序键列中的分配，并利用该信息来优化排序键列的交错操作。随着表的增大，排序键列中值的分配可能会发生变化或偏斜，日期或时间戳列的变化或偏斜更加明显。如果偏斜过大，则性能会受到影响。要重新分析排序键并恢复性能，请运行包含 REINDEX 关键字的 VACUUM 命令。对于交错表，由于它必须对数据进行额外的分析，因此，VACUUM REINDEX 需要花费比标准 VACUUM 操作还要长的时间。要查看有关键分配偏斜和上次重建索引时间的信息，请查询 SVV_INTERLEAVED_COLUMNS 系统视图。

有关如何确定运行 VACUUM 的频度和运行 VACUUM REINDEX 的时机的更多信息，请参阅决定是否重建索引。

定义表约束

唯一键、主键和外键约束仅供参考；在您填充表时，Amazon Redshift 并不强制实施它们。例如，如果您将数据插入到具有依赖关系的表中，即使插入操作违反了约束，插入也会成功。但是，主键和外键用作规划提示，如果您应用程序中的 ETL 处理或其他一些处理强制其完整性，则应声明它们。

例如，查询计划器在某些统计计算中使用主键和外键。它这样做是为了推断影响子查询去相关技术的唯一性和引用关系。通过这样做，它可以对大量联接进行排序并消除冗余联接。

计划程序需要利用这些键关系，但它假定 Amazon Redshift 表中的所有键在加载时是有效的。如果您的应用程序允许无效的外键或主键，则某些查询可能返回错误的结果。例如，如果主键不唯一，则 SELECT DISTINCT 查询可能会返回重复的行。如果您不确定自己的表的键约束是否正确，则不要定义它们。不过，如果您确定主键和外键以及唯一性约束有效，则应始终声明它们。

Amazon Redshift 确实会强制实施 NOT NULL 列约束。

有关表约束的更多信息，请参阅 CREATE TABLE。有关如何删除具有依赖关系的表的信息，请参阅 DROP TABLE。

加载数据

主题

COPY 命令是最高效的加载表的方式。您也可以使用 INSERT 命令将数据添加到您的表中，尽管这与使用 COPY 命令相比的效率低得多。COPY 命令能够同时从多个数据文件或多个数据流读取。Amazon Redshift 会将工作负载分配到集群节点，并且并行执行加载操作，包括对行进行排序和跨节点切片分配数据。

> **Note**
>
> Amazon Redshift Spectrum 外部表为只读。您无法对外部表进行 COPY 或 INSERT。

要访问其他 AWS 资源上的数据，您的集群必须有权访问这些资源和有权执行访问数据所需的操作。您可使用 AWS Identity and Access Management（IAM）将用户拥有的访问权限限制为您的集群资源和数据。

在初始数据加载后，如果您添加、修改或删除大量数据，则应随后运行 VACUUM 命令，以便识别您的数据并在数据删除后回收空间。您还应该运行 ANALYZE 命令来更新表统计数据。

本部分介绍如何加载数据和排查数据加载问题，并介绍加载数据的最佳实践。

使用 COPY 命令加载数据

主题

- 凭证和访问权限
- 准备输入数据
- 从 Amazon S3 加载数据
- 从 Amazon EMR 中加载数据
- 从远程主机中加载数据
- 从 Amazon DynamoDB 表中加载数据
- 验证是否正确加载了数据
- 验证输入数据
- 使用自动压缩加载表
- 针对窄表优化存储
- 加载默认列值
- 解决数据加载问题

COPY 命令使用 Amazon Redshift 大规模并行处理 (MPP) 架构从 Amazon S3 上的文件、DynamoDB 表或者来自一个或多个远程主机的文本输出并行读取和加载数据。

> **Note**
>
> 我们强烈建议使用 COPY 命令加载大量数据。使用单个 INSERT 语句填充表可能过于缓慢。此外，如果您的数据在其他 Amazon Redshift 数据库表中已经存在，请使用 INSERT INTO ... SELECT 或 CREATE TABLE AS 来提高性能。有关更多信息，请参阅 INSERT 或 CREATE TABLE AS。

要从另一 AWS 资源加载数据，您的集群必须有权访问相应资源和执行所需操作。

若要授予或撤消使用 COPY 命令将数据加载到表中的特权，请授予或撤消 INSERT 特权。

您的数据需要采用恰当的格式才能加载到 Amazon Redshift 表中。本部分介绍在加载数据前准备和验证数据以及在运行 COPY 语句前对其进行验证的指南。

若要保护文件中的信息，您可以先对数据文件进行加密，然后再将它们上载到 Amazon S3 桶；COPY 将在执行加载时解密数据。您还可以通过向用户提供临时安全凭证来限制对您的加载数据的访问。临时安全凭证可增强安全性，因为它们时效短，过期后无法重复使用。

Amazon Redshift 具有 COPY 内置功能，可以快速加载未压缩的、分隔的数据。但您可以使用 gzip、lzop 或 bzip2 来压缩您的文件，以节约上载文件的时间。

如果 COPY 查询中包含以下关键词，则不支持自动拆分以下格式的未压缩数据：ESCAPE、REMOVEQUOTES 和 FIXEDWIDTH。但是支持 CSV 关键字。

为帮助保护 AWS Cloud 中正在传输的数据，Amazon Redshift 使用硬件加速 SSL 与 Amazon S3 或 Amazon DynamoDB 通信，以执行 COPY、UNLOAD、备份和还原操作。

如果直接从 Amazon DynamoDB 表加载您的表，您可以选择控制自己占用的 Amazon DynamoDB 预配置吞吐量。

作为加载过程的一部分，您可选择让 COPY 分析您的输入数据并对您的表自动应用最佳压缩编码。

凭证和访问权限

要使用其他 AWS 资源（如 Amazon S3、Amazon DynamoDB、Amazon EMR 或 Amazon EC2）加载或卸载数据，您的集群必须有权访问这些资源和执行访问数据所需的操作。例如，要从 Amazon S3 加载数据，COPY 必须具有对桶的 LIST 访问权限以及对桶对象的 GET 访问权限。

要获取访问资源的授权，您的集群必须经过身份验证。您可以选择基于角色的访问控制或基于密钥的访问控制。本节概述了这两种方法。有关完整的详细信息和示例，请参阅访问其他 AWS 资源的权限。

基于角色的访问控制

利用基于角色的访问控制，您的集群将代表您临时代入 AWS Identity and Access Management (IAM) 角色。然后，基于对角色的授权，您的集群可访问所需的 AWS 资源。

我们建议使用基于角色的访问控制，因为除了保护您的 AWS 凭证之外，它还将提供对 AWS 资源和敏感用户数据的更安全、精细的访问控制。

要使用基于角色的访问控制，您必须先使用 Amazon Redshift 服务角色类型创建 IAM 角色，然后将此角色附加到您的集群。此角色至少必须具有 COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限中列出的权限。有关创建 IAM 角色并将其附加到集群的步骤，请参阅《Amazon Redshift 管理指南》中的创建 IAM 角色以允许您的 Amazon Redshift 集群访问 AWS 服务。

通过使用 Amazon Redshift 管理控制台、CLI 或 API，您可将角色添加到集群或查看与集群关联的角色。有关更多信息，请参阅《Amazon Redshift 管理指南》中的使用 IAM 角色授权 COPY 和 UNLOAD 操作。

当您创建 IAM 角色时，IAM 将返回该角色的 Amazon 资源名称（ARN）。要使用 IAM 角色运行 COPY 命令，请使用 IAM_ROLE 参数或 CREDENTIALS 参数提供角色 ARN。

以下 COPY 命令示例使用角色 MyRedshiftRole 的 IAM_ROLE 参数进行身份验证。

```
copy customer from 's3://mybucket/mydata'
iam_role 'arn:aws:iam::12345678901:role/MyRedshiftRole';
```

AWS 用户必须至少具有COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限中列出的权限。

基于密钥的访问控制

利用基于密钥的访问控制，您可以为获得授权可访问包含数据的 AWS 资源的用户提供访问密钥 ID 和秘密访问密钥。

> **Note**
>
> 我们强烈建议使用 IAM 角色进行身份验证而不是提供纯文本访问密钥 ID 和秘密访问密钥。如果您选择基于密钥的访问控制，则不要使用 AWS 账户（根）凭证。应始终创建 IAM 用户并提供该用户的访问密钥 ID 和秘密访问密钥。有关创建 IAM 用户的步骤，请参阅在您的 AWS 账户中创建 IAM 用户。

准备输入数据

如果您的输入数据与将要接收数据的表列不兼容，则 COPY 命令将失败。

请使用以下指南，以帮助确保您的输入数据有效：

- 您的数据只能包含不超过四个字节长度的 UTF-8 字符。
- 确认 CHAR 和 VARCHAR 字符串的长度不超过相应列的长度。VARCHAR 字符串以字节（而不是字符）为单位衡量，因此，如果一个字符串有四个中文字符，每个字符占用四个字节，则需要一个 VARCHAR(16) 列。
- 多字节字符只能用于 VARCHAR 列。确认多字节字符的长度不超过四个字节。
- 确认 CHAR 列的数据仅包含单字节字符。
- 不要加入任何特殊字符或语法来表示记录中的最后一个字段。此字段可能为分隔符。
- 如果数据包括 null 终止符，也称为 NUL (UTF-8 0000) 或二进制零 (0x000)，则可以使用 COPY 命令中的 NULL AS 选项将这些字符作为 NULLS 加载到 CHAR 或 VARCHAR 列中：`null as '\0'` 或 `null as '\000'`。如果您不使用 NULL AS，null 终止符将导致 COPY 失败。

- 如果您的字符串包含特殊字符（如分隔符和嵌入换行符），请对 COPY 命令使用 ESCAPE 选项。
- 确认所有单引号和双引号已正确匹配。
- 确认浮点字符串采用了标准浮点字符格式（例如 12.123）或指数格式（例如 1.0E4）。
- 确认所有时间戳和日期字符串遵循了 DATEFORMAT 和 TIMEFORMAT 字符串 的说明。默认时间戳格式为 YYYY-MM-DD hh:mm:ss，默认日期格式为 YYYY-MM-DD。
- 有关各个日期类型的边界和限制的更多信息，请参阅数据类型。有关多字节字符错误的信息，请参阅多字节字符加载错误

从 Amazon S3 加载数据

主题

- 从压缩和未压缩文件中加载数据
- 将文件上载到 Amazon S3
- 使用 COPY 命令从 Amazon S3 中加载

COPY 命令使用 Amazon Redshift 大规模并行处理 (MPP) 架构从 Amazon S3 桶中的一个或多个文件并行读取和加载数据。在压缩文件的情况下，您可以将数据拆分成多个文件，从而最大程度地利用并行处理。（此规则有例外。加载数据文件中详细介绍了相关内容。）您也可以通过在表上设置分配键，从而最大程度地利用并行处理。有关分配键的更多信息，请参阅使用数据分配样式。

数据将加载到目标表中，一行数据占据表中的一行。数据文件中的字段按从左到右的顺序与表列相匹配。数据文件中的字段可以是固定宽度，也可以用字符分隔；默认分隔符为竖线 (|)。默认情况下，将加载所有表列，但您可以选择定义用逗号分隔的列列表。如果 COPY 命令中指定的列列表中不包括某个表列，则该表列将加载为默认值。有关更多信息，请参阅 加载默认列值。

从压缩和未压缩文件中加载数据

加载压缩数据时，我们建议您将每个表的数据拆分成多个文件。当您加载未压缩的分隔数据时，COPY 命令使用大规模并行处理 (MPP) 和扫描范围，从 Amazon S3 桶的大型文件中加载数据。

从多个压缩文件中加载数据

如果您有压缩数据，我们建议您将每个表的数据拆分成多个文件。COPY 命令可以从多个文件并行加载数据。您可以通过指定一个通用前缀（对于集合，则为前缀键），或通过在清单文件中明确列出文件，从而加载多个文件。

将数据拆分成多个文件，以便文件数是您的集群中的切片数的倍数。这样，Amazon Redshift 就可以在切片之间均匀地拆分数据。每个节点的切片数取决于集群的节点大小。例如，每个 dc2.large 计算节点有两个切片，每个 dc2.8xlarge 计算节点有 16 个切片。有关每个节点大小拥有的切片数的更多信息，请转到《Amazon Redshift 管理指南》中的关于集群和节点。

所有节点均参与并行查询的运行，处理尽可能跨切片均匀分布的数据。如果您的集群有两个 dc2.large 节点，则可以将数据拆分为四个文件或四的倍数个文件。Amazon Redshift 在拆分工作负载时不会考虑文件大小。因此，您需要确保文件大小大致相同，压缩后大小为 1MB 到 1GB。

要使用对象前缀来标识加载文件，为每个文件命名时请加上一个通用前缀。例如，您可以将 `venue.txt` 文件拆分成四个文件，如下所示：

```
venue.txt.1
venue.txt.2
venue.txt.3
venue.txt.4
```

如果您在桶中的一个文件夹中放置多个文件，则可以将该文件夹名称指定为前缀，COPY 会加载该文件夹中的所有文件。如果您使用清单文件明确列出要加载的文件，则这些文件可以位于不同的桶或文件夹中。

有关清单文件的更多信息，请参阅Example: COPY from Amazon S3 using a manifest。

从未压缩的分隔文件中加载数据

当您加载未压缩的、分隔的数据时，COPY 命令会使用 Amazon Redshift 中的大规模并行处理 (MPP) 架构。Amazon Redshift 会自动使用并行工作的切片，从 Amazon S3 桶的大型文件中加载多个范围的数据。必须对文件进行分隔才能进行并行加载。例如，竖线分隔。使用 COPY 命令自动并行加载数据也适用于 CSV 文件。您还可以通过在表上设置分配键，从而充分利用并行处理。有关分配键的更多信息，请参阅使用数据分配样式。

当 COPY 查询包含以下任何关键字时，不支持自动并行加载数据：ESCAPE、REMOVEQUOTES 和 FIXEDWIDTH。

一个或多个文件中的数据将加载到目标表中，一行数据占据表中的一行。数据文件中的字段按从左到右的顺序与表列相匹配。数据文件中的字段可以是固定宽度，也可以用字符分隔；默认分隔符为竖线 (|)。默认情况下，将加载所有表列，但您可以选择定义用逗号分隔的列列表。如果 COPY 命令中指定的列列表中不包括某个表列，则该表列将加载为默认值。有关更多信息，请参阅 加载默认列值。

在数据没有压缩和分隔时，请遵循此一般流程从 Amazon S3 加载数据：

1. 将您的文件上载到 Amazon S3 。
2. 运行 COPY 命令以加载表。
3. 确认数据已正确加载。

有关 COPY 命令的示例，请参阅 COPY 示例。有关加载到 Amazon Redshift 的数据的信息，请查看 STL_LOAD_COMMITS 和 STL_LOAD_ERRORS 系统表。

有关各表中涵盖的节点以及切片的更多信息，请参阅《Amazon Redshift 管理指南》中的关于集群和节点。

将文件上载到 Amazon S3

主题

- 管理数据一致性
- 将加密的数据上载到 Amazon S3
- 确认在桶中具有正确的文件

将文本文件上载到 Amazon S3 时，可以采取以下几种方法：

- 如果您有压缩文件，我们建议您拆分大文件，以便充分利用 Amazon Redshift 中的并行处理。
- 另一方面，COPY 会自动拆分大型未压缩文本分隔文件数据，促进并行并有效分发大型文件中的数据。

创建一个用于存储数据文件的 Amazon S3 桶，然后将数据文件上载到该桶。有关创建桶和上载文件的信息，请参阅《Amazon Simple Storage Service 用户指南》中的使用 Amazon S3 桶。

> **Important**
>
> 必须在与集群相同的 AWS 区域中创建存储数据文件的 Amazon S3 桶，除非您使用 REGION 选项指定 Amazon S3 桶所在的区域。

确保将 S3 IP 范围添加到您的允许列表中。要了解有关所需 S3 IP 范围的更多信息，请参阅网络隔离。

您可以通过以下方式在特定区域中创建 Amazon S3 桶：在使用 Amazon S3 控制台创建桶时选择该区域，或者在使用 Amazon S3 API 或 CLI 创建桶时指定端点。

数据加载后，确认 Amazon S3 上出现了正确的文件。

管理数据一致性

Amazon S3 为所有 AWS 区域中的 Amazon S3 桶上的 COPY、UNLOAD、INSERT（外部表）、CREATE EXTERNAL TABLE AS 和 Amazon Redshift Spectrum 操作提供了强大的先写后读一致性。此外，针对 Amazon S3 Select、Amazon S3 访问控制列表、Amazon S3 对象标签和对象元数据（例如 HEAD 对象）的读取操作具有严格的一致性。有关数据一致性的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的 Amazon S3 数据一致性模型。

将加密的数据上载到 Amazon S3

Amazon S3 支持服务器端加密和客户端加密。本主题将讨论服务器端加密和客户端加密的区别，并介绍将客户端加密用于 Amazon Redshift 的步骤。服务器端加密对 Amazon Redshift 是透明的。

服务器端加密

服务器端加密是静态数据加密，即，Amazon S3 在上载数据时对其进行加密，并在您访问时进行解密。当您使用 COPY 命令加载表时，在 Amazon S3 上从服务器端加密或解密的对象的加载方式没有不同。有关服务器端加密的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的使用服务器端加密。

客户端加密

在客户端加密中，您的客户端应用程序管理数据的加密、加密密钥和相关的工具。您可以使用客户端加密将数据上载到 Amazon S3 桶，然后使用带有 ENCRYPTED 选项的 COPY 命令和私有加密密钥加载数据，以实现更强的安全性。

您使用信封加密来加密您的数据。借助信封加密，您的应用程序可专门处理所有加密。您的私有加密密钥和未加密的数据从来不会发送到 AWS，因此请您务必妥善管理好您的加密密钥。如果您丢失了加密密钥，您将无法解密数据，而且，您无法从 AWS 找回您的加密密钥。信封加密结合了快速对称加密的性能，同时保持了使用非对称密钥进行密钥管理所获得的更强的安全性。Amazon S3 加密客户端生成一次性对称密钥（信封对称密钥）来加密数据，然后由您的根密钥对其进行加密并与您的数据一起存储在 Amazon S3 中。在加载过程中，当 Amazon Redshift 访问您的数据时，将检索加密的对称密钥并使用您的实际密钥对其进行解密，然后解密数据。

要在 Amazon Redshift 中使用 Amazon S3 客户端加密数据，请按照《Amazon Simple Storage Service 用户指南》中的使用客户端加密保护数据所列的步骤操作，并满足您使用的其它要求：

- 对称加密 – AWS SDK for Java `AmazonS3EncryptionClient` 类使用信封加密，如前所述，信封加密是基于对称密钥加密的。使用此类可创建要上载客户端加密数据的 Amazon S3 客户端。

- 256 位 AES 根对称密钥 – 根密钥将对信封密钥进行加密。您将根密钥传递给 `AmazonS3EncryptionClient` 类的实例。保存此密钥，因为您将需要用它来将数据复制到 Amazon Redshift 中。
- 存储加密信封密钥的对象元数据 – 预设情况下，Amazon S3 将信封密钥存储为 `AmazonS3EncryptionClient` 类的对象元数据。存储为对象元数据的加密信封密钥将在加密过程中使用。

> **Note**
>
> 首次使用加密 API 时，如果您收到密码加密错误消息，则您的 JDK 版本可能带有一个 Java Cryptography Extension (JCE) 区域策略文件，该文件将加密和解密转换的最大密钥长度限制为 128 位。有关解决此问题的信息，请转到《Amazon Simple Storage Service 用户指南》中的使用 AWS SDK for Java 指定客户端加密。

有关使用 COPY 命令将客户端加密的文件加载到 Amazon Redshift 表中的信息，请参阅从 Amazon S3 中加载加密的数据文件。

示例：上传客户端加密数据

有关如何使用 AWS SDK for Java 上载客户端加密数据的示例，请转到《Amazon Simple Storage Service 用户指南》中的使用客户端加密保护数据。

第二个选项展示了您若要在 Amazon Redshift 中加载数据而必须在客户端加密期间做出的选择。具体来讲，该示例展示了使用对象元数据来存储加密信封密钥和 256 位 AES 根对称密钥的使用。

本示例代码使用 AWS SDK for Java 创建 256 位 AES 对称根密钥并将它保存到文件。然后，本示例使用首先加密客户端上样本数据的 S3 加密客户端将一个对象上传到 Amazon S3。示例还将下载该对象，并验证数据是否相同。

确认在桶中具有正确的文件

将文件上载到 Amazon S3 桶之后，我们建议您列出桶的内容，以便确认所有正确文件均已存在并且不存在不需要的文件。例如，如果桶 `mybucket` 存储了一个名为 `venue.txt.back` 的文件，则该文件可能会被以下命令意外加载：

```
copy venue from 's3://mybucket/venue' … ;
```

如果您要控制具体加载哪些文件，则可使用清单文件明确列出所需的数据文件。有关使用清单文件的更多信息，请参阅 COPY 命令的 copy_from_s3_manifest_file 选项和 COPY 示例中的Example: COPY from Amazon S3 using a manifest。

有关列出桶内容的更多信息，请参阅《Amazon S3 开发人员指南》中的列出对象键。

使用 COPY 命令从 Amazon S3 中加载

主题

- 使用清单指定数据文件
- 从 Amazon S3 中加载压缩的数据文件
- 从 Amazon S3 中加载固定宽度的数据
- 从 Amazon S3 中加载多字节数据
- 从 Amazon S3 中加载加密的数据文件

使用 COPY 命令从 Amazon S3 上的数据文件并行加载表。您可以使用 Amazon S3 对象前缀或清单文件指定要加载的文件。

使用前缀指定要加载的文件的语法如下所示：

```
copy <table_name> from 's3://<bucket_name>/<object_prefix>'
authorization;
```

清单文件是 JSON 格式的文件，该文件会列出要加载的数据文件。使用清单文件指定要加载的文件的语法如下所示：

```
copy <table_name> from 's3://<bucket_name>/<manifest_file>'
authorization
manifest;
```

数据库中必须已存在要加载的表。有关创建表的信息，请参阅 SQL 参考中的CREATE TABLE。

authorization 的值为您的集群提供访问 Amazon S3 对象所需的 AWS 授权。有关所需权限的信息，请参阅COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限。进行身份验证的首选方法是为具有必要权限的 IAM 角色指定 IAM_ROLE 参数并提供 Amazon 资源名称（ARN）。有关更多信息，请参阅 基于角色的访问控制。

要使用 IAM_ROLE 参数进行身份验证，请替换 *<aws-account-id>* 和 *<role-name>*，如以下语法中所示。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

以下示例使用 IAM 角色进行身份验证。

```
copy customer
from 's3://mybucket/mydata'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

有关其他授权选项的更多信息，请参阅授权参数。

如果您要验证您的数据而不实际加载表，请对 COPY 命令使用 NOLOAD 选项。

以下示例展示了一个名为 venue.txt 的文件中用竖线分隔的数据的前几行。

```
1|Toyota Park|Bridgeview|IL|0
2|Columbus Crew Stadium|Columbus|OH|0
3|RFK Stadium|Washington|DC|0
```

在将文件上载到 Amazon S3 之前，将文件拆分成多个文件，以便 COPY 命令使用并行处理来加载它。文件数应为您的集群中的切片数的倍数。拆分您的加载数据文件，使文件大小大约相等，压缩后的文件大小介于 1 MB 和 1 GB 之间。有关更多信息，请参阅 从压缩和未压缩文件中加载数据。

例如，venue.txt 文件可拆分成四个文件，如下所示：

```
venue.txt.1
venue.txt.2
venue.txt.3
venue.txt.4
```

以下 COPY 命令将使用 Amazon S3 桶 mybucket 中带有前缀“venue”的数据文件中用竖线分隔的数据加载 VENUE 表。

> **Note**
>
> Amazon S3 桶 mybucket 在以下示例中不存在。有关使用现有 Amazon S3 桶中的实际数据的示例 COPY 命令，请参阅加载示例数据。

```
copy venue from 's3://mybucket/venue'
```

```
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

如果带有键前缀“venue”的 Amazon S3 对象不存在，则加载将失败。

使用清单指定数据文件

您可以对数据加载使用清单以确保 COPY 命令加载所有需要的文件，且仅加载需要的文件。您可以使用清单来加载不同桶或文件中的未共享相同前缀的文件。您可以提供明确列出要加载的文件的 JSON 格式的文本文件的名称，而不必提供 COPY 命令的对象路径。清单中的 URL 必须指定桶名称和文件的完整对象路径，而不仅仅是前缀。

有关清单文件的更多信息，请参阅 COPY 示例使用清单指定数据文件。

以下示例展示了用于加载不同桶中的文件名以日期戳开头的文件的 JSON。

```
{
  "entries": [
    {"url":"s3://mybucket-alpha/2013-10-04-custdata", "mandatory":true},
    {"url":"s3://mybucket-alpha/2013-10-05-custdata", "mandatory":true},
    {"url":"s3://mybucket-beta/2013-10-04-custdata", "mandatory":true},
    {"url":"s3://mybucket-beta/2013-10-05-custdata", "mandatory":true}
  ]
}
```

可选的 mandatory 标志指定 COPY 是否应在找不到文件时返回错误。mandatory 的默认值为 false。如果未找到任何文件，则无论 mandatory 设置如何，COPY 都将终止。

以下示例将使用前一个示例中名为 cust.manifest 的清单来运行 COPY 命令。

```
copy customer
from 's3://mybucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

使用 UNLOAD 创建的清单

由 UNLOAD 操作使用 MANIFEST 参数创建的清单可能具有 COPY 操作不需要的键。例如，以下 UNLOAD 清单包含一个 meta 键，该键是 Amazon Redshift Spectrum 外部表所必需的，并用于加载 ORC 或 Parquet 文件格式的数据文件。meta 键包含具有文件实际大小值（以字节为单位）的 content_length 键。COPY 操作只需要 url 密钥和可选 mandatory 密钥。

```
{
  "entries": [
    {"url":"s3://mybucket/unload/manifest_0000_part_00", "meta": { "content_length":
 5956875 }},
    {"url":"s3://mybucket/unload/unload/manifest_0001_part_00", "meta":
 { "content_length": 5997091 }}
 ]
}
```

有关清单文件的更多信息，请参阅Example: COPY from Amazon S3 using a manifest。

从 Amazon S3 中加载压缩的数据文件

要加载使用 gzip、lzop 或 bzip2 压缩的数据文件，请包括相应的选项：GZIP、LZOP 或 BZIP2。

例如，以下命令将从使用 lzop 压缩的文件加载。

```
copy customer from 's3://mybucket/customer.lzo'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' lzop;
```

> **Note**
>
> 如果您使用 lzop 压缩来压缩数据文件并使用 --filter 选项，则 COPY 命令不支持该文件。

从 Amazon S3 中加载固定宽度的数据

固定宽度的数据文件的每个数据列具有统一的长度。固定宽度的数据文件中的每个字段的长度和位置完全相同。对于固定宽度的数据文件中的字符数据（CHAR 和 VARCHAR），您必须包括前导空格或尾随空格作为占位符，以便保持宽度统一。对于整数，您必须使用前导零作为占位符。固定宽度的数据文件没有用于分隔列的分隔符。

要将固定宽度的数据文件加载到现有表，请在 COPY 命令中使用 FIXEDWIDTH 参数。您的表说明必须与 fixedwidth_spec 的值相匹配才能正确加载数据。

要将固定宽度的数据从文件加载到表，请发出以下命令：

```
copy table_name from 's3://mybucket/prefix'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
fixedwidth 'fixedwidth_spec';
```

fixedwidth_spec 参数是一个字符串，其中包含每个列的标识符和每个列的宽度（用冒号分隔）。**column:width** 对由逗号分隔。标识符可以是您选择的任意形式：数字、字母或二者的组合。标识符与表本身无关，因此说明包含列的顺序必须与表中的列顺序相同。

以下两个示例演示了相同说明，第一个使用数字标识符，第二个使用字符串标识符：

```
'0:3,1:25,2:12,3:2,4:6'
```

```
'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6'
```

以下示例演示了固定宽度的样本数据，可使用前面的说明将这些数据加载到 VENUE 表中：

```
1  Toyota Park               Bridgeview  IL0
2  Columbus Crew Stadium     Columbus    OH0
3  RFK Stadium               Washington  DC0
4  CommunityAmerica Ballpark Kansas City KS0
5  Gillette Stadium          Foxborough  MA68756
```

以下 COPY 命令会将此数据集加载到 VENUE 表中：

```
copy venue
from 's3://mybucket/data/venue_fw.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth 'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6';
```

从 Amazon S3 中加载多字节数据

如果您的数据包含非 ASCII 多字节字符（例如中文或西里尔语字符），则必须将该数据加载到 VARCHAR 列。VARCHAR 数据类型支持四字节的 UTF-8 字符，而 CHAR 数据类型仅接受单字节的 ASCII 字符。您不能将五字节或更长的字符加载到 Amazon Redshift 表中。有关 CHAR 和 VARCHAR 的更多信息，请参阅数据类型。

要检查输入文件使用哪种编码，请使用 Linux *file* 命令：

```
$ file ordersdata.txt
```

```
ordersdata.txt: ASCII English text
$ file uni_ordersdata.dat
uni_ordersdata.dat: UTF-8 Unicode text
```

从 Amazon S3 中加载加密的数据文件

您可以使用 COPY 命令加载使用服务器端加密、客户端加密或两种加密方式上载到 Amazon S3 的数据文件。

COPY 命令支持以下 Amazon S3 加密方式：

- 使用 Amazon S3 托管密钥进行服务器端加密 (SSE-S3)
- 使用 AWS KMS keys 的服务器端加密 (SSE-KMS)
- 使用客户端对称根密钥进行客户端加密

COPY 命令不支持以下 Amazon S3 加密方式：

- 使用客户提供的密钥进行服务器端加密 (SSE-C)
- 使用 AWS KMS key 进行客户端加密
- 使用客户提供的对称根密钥进行客户端加密

有关 Amazon S3 加密的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的使用服务器端加密保护数据和使用客户端加密保护数据。

UNLOAD 命令使用 SSE-S3 自动加密文件。您还可以使用 SSE-KMS 或客户托管式对称密钥进行客户端加密的方法进行卸载。有关更多信息，请参阅卸载加密的数据文件。

COPY 命令会自动识别并加载使用 SSE-S3 和 SSE-KMS 进行加密的文件。您可以指定 ENCRYPTED 选项并提供密钥值，加载使用客户端对称根密钥加密的文件。有关更多信息，请参阅 将加密的数据上载到 Amazon S3。

要加载客户端加密数据文件，请使用 MASTER_SYMMETRIC_KEY 参数提供根密钥值，并包括 ENCRYPTED 选项。

```
copy customer from 's3://mybucket/encrypted/customer'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted
```

```
delimiter '|';
```

要加载 gzip、lzop 或 bzip2 压缩的加密数据文件，请随根密钥值和 ENCRYPTED 选项包括 GZIP、LZOP 或 BZIP2 选项。

```
copy customer from 's3://mybucket/encrypted/customer'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted
delimiter '|'
gzip;
```

从 Amazon EMR 中加载数据

您可以使用 COPY 命令从一个具有如下配置的 Amazon EMR 集群并行加载数据：将文本文件作为固定宽度文件、字符分隔文件、CSV 文件或 JSON 格式文件写入到集群的 Hadoop Distributed File System (HDFS)。

从 Amazon EMR 中加载数据的过程

本节演练从 Amazon EMR 集群加载数据的过程。以下各节提供您必须完成每个步骤的详细信息。

- 步骤 1：配置 IAM 权限

  用户必须拥有必要的权限才能创建 Amazon EMR 集群和运行 Amazon Redshift COPY 命令。
- 步骤 2：创建 Amazon EMR 集群

  将集群配置为将文本文件输出到 Hadoop Distributed File System (HDFS)。您需要 Amazon EMR 集群 ID 和集群的主节点公有 DNS（托管集群的 Amazon EC2 实例的端点）。
- 步骤 3：检索 Amazon Redshift 集群公有密钥和集群节点 IP 地址

  公有密钥使 Amazon Redshift 集群节点能够建立与主机的 SSH 连接。您将使用每个集群节点的 IP 地址来配置主机安全组，从而允许使用这些 IP 地址从 Amazon Redshift 集群访问。
- 步骤 4：将 Amazon Redshift 集群公有密钥添加到每个 Amazon EC2 主机的授权密钥文件

  您将 Amazon Redshift 集群公有密钥添加到主机的授权密钥文件，以便让主机识别 Amazon Redshift 集群并接受 SSH 连接。
- 步骤 5：将主机配置为接受 Amazon Redshift 集群的所有 IP 地址

  修改 Amazon EMR 实例的安全组，以添加接受 Amazon Redshift IP 地址的输入规则。

- 步骤 6：运行 COPY 命令以加载数据

  从 Amazon Redshift 数据库运行 COPY 命令，以便将数据加载到 Amazon Redshift 表中。

步骤 1：配置 IAM 权限

用户必须拥有必要的权限才能创建 Amazon EMR 集群和运行 Amazon Redshift COPY 命令。

配置 IAM 权限

1. 为将要创建 Amazon EMR 集群的用户添加以下权限。

```
ec2:DescribeSecurityGroups
ec2:RevokeSecurityGroupIngress
ec2:AuthorizeSecurityGroupIngress
redshift:DescribeClusters
```

2. 为将要运行 COPY 命令的 IAM 角色或用户添加以下权限。

```
elasticmapreduce:ListInstances
```

3. 向 Amazon EMR 集群的 IAM 角色添加以下权限。

```
redshift:DescribeClusters
```

步骤 2：创建 Amazon EMR 集群

COPY 命令从 Amazon EMR Hadoop Distributed File System (HDFS) 上的文件加载数据。当您创建 Amazon EMR 集群时，请将集群配置为将数据文件输出到集群的 HDFS。

要创建 Amazon EMR 集群

1. 在与 Amazon Redshift 集群相同的 AWS 区域中创建 Amazon EMR 集群。

   如果 Amazon Redshift 集群在 VPC 中，则 Amazon EMR 集群必须在同一 VPC 组中。如果 Amazon Redshift 集群使用 EC2-Classic 模式（即，它不在 VPC 中），则 Amazon EMR 集群必须也使用 EC2-Classic 模式。有关更多信息，请参阅《Amazon Redshift 管理指南》中的管理 Virtual Private Cloud (VPC) 中的集群。

2. 将集群配置为将数据文件输出到集群的 HDFS。HDFS 文件名不能包括星号 (*) 或问号 (?)。

> **Important**
>
> 文件名不能包括星号 (*) 或问号 (?)。

3. 在 Amazon EMR 集群配置中，将自动终止选项指定为否，以便集群在 COPY 命令运行时保持可用。

> **Important**
>
> 如果在 COPY 完成前更改或删除了任何数据文件，则您可能会遇到意外结果，或者 COPY 操作可能失败。

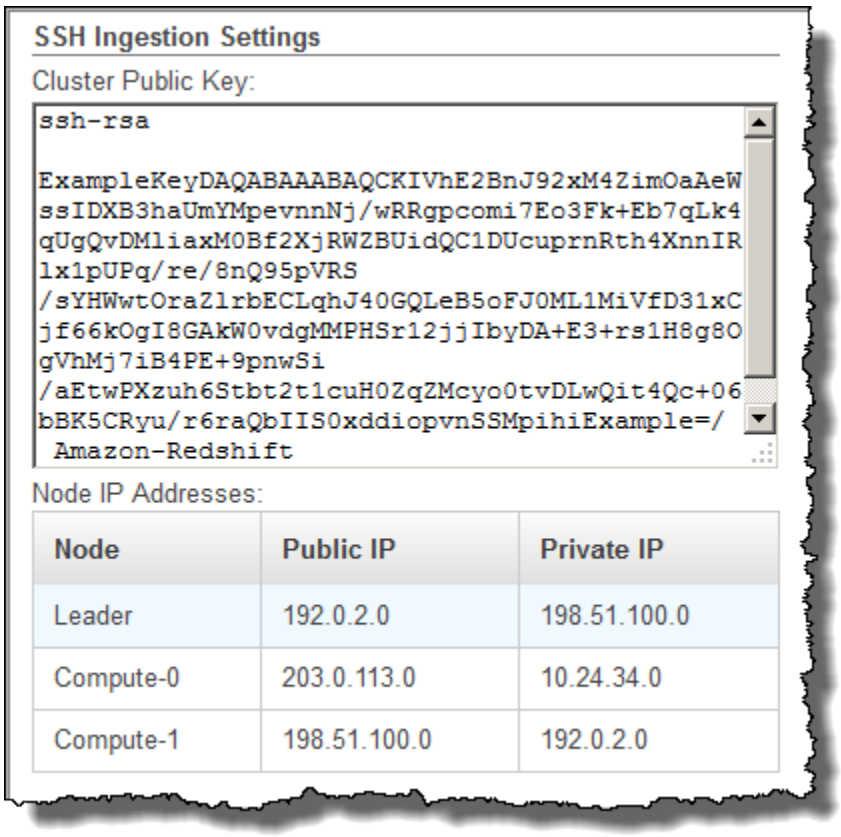
4. 请记下集群 ID 和主节点公有 DNS（托管集群的 Amazon EC2 实例的端点）。您将在后面的步骤中用到这些信息。

步骤 3：检索 Amazon Redshift 集群公有密钥和集群节点 IP 地址

要使用控制台为您的集群检索 Amazon Redshift 集群公有密钥和集群节点 IP 地址

1. 访问 Amazon Redshift 管理控制台。
2. 在导航窗格中选择集群链接。
3. 从列表中选择您的集群。
4. 找到 SSH Ingestion Settings 组。

   记下 Cluster Public Key 和 Node IP addresses 中的值。您将在后面的步骤中用到它们。



您将使用步骤 3 中的私有 IP 地址将 Amazon EC2 主机配置为接受来自 Amazon Redshift 的连接。

若要使用 Amazon Redshift CLI 检索您的集群的集群公有密钥和集群节点 IP 地址，请运行 describe-clusters 命令。例如：

```
aws redshift describe-clusters --cluster-identifier <cluster-identifier>
```

响应将包括 ClusterPublicKey 值和私有 IP 地址及公有 IP 地址的列表，类似于以下内容：

```
{
    "Clusters": [
        {
            "VpcSecurityGroups": [],
            "ClusterStatus": "available",
            "ClusterNodes": [
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "LEADER",
```

```
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-0",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-1",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                }
            ],
            "AutomatedSnapshotRetentionPeriod": 1,
            "PreferredMaintenanceWindow": "wed:05:30-wed:06:00",
            "AvailabilityZone": "us-east-1a",
            "NodeType": "dc2.large",
            "ClusterPublicKey": "ssh-rsa AAAABexamplepublickey...Y3TAl Amazon-
Redshift",
             ...
             ...
}
```

要使用 Amazon Redshift API 检索您的集群的集群公有密钥和集群节点 IP 地址，请使用 `DescribeClusters` 操作。有关更多信息，请参阅《Amazon Redshift CLI 指南》中的 describe-clusters 或《Amazon Redshift API 指南》中的 DescribeClusters。

步骤 4：将 Amazon Redshift 集群公有密钥添加到每个 Amazon EC2 主机的授权密钥文件

您将所有 Amazon EMR 集群节点的集群公有密钥添加到每个主机的授权密钥文件，以便主机识别 Amazon Redshift 并接受 SSH 连接。

要将 Amazon Redshift 集群公有密钥添加到主机的授权密钥文件

1. 使用 SSH 连接访问主机。

   有关使用 SSH 连接到实例的信息，请参阅《Amazon EC2 用户指南》中的连接到您的实例。
2. 从控制台或从 CLI 响应文本复制 Amazon Redshift 公有密钥。
3. 将公有密钥的内容复制并粘贴到主机上的 /home/<ssh_username>/.ssh/authorized_keys 文件中。请包括完整字符串（包含前缀“ssh-rsa”和后缀“Amazon-Redshift”）。例如：

```
ssh-rsa AAAACTP3isxgGzVWoIWpbVvRCOzYdVifMrh… uA70BnMHCaMiRdmvsDOedZDOedZ Amazon-
Redshift
```

步骤 5：将主机配置为接受 Amazon Redshift 集群的所有 IP 地址

若要允许到主机实例的入站流量，请编辑安全组并为每个 Amazon Redshift 集群节点添加一个入站规则。对于 Type，请选择在端口 22 上使用 TCP 协议的 SSH。对于源，请输入您在步骤 3：检索 Amazon Redshift 集群公有密钥和集群节点 IP 地址中检索的 Amazon Redshift 集群节点私有 IP 地址。有关添加规则到 Amazon EC2 安全组的信息，请参阅《Amazon EC2 用户指南》中的为您的实例授权入站流量。

步骤 6：运行 COPY 命令以加载数据

运行 COPY 命令以连接到 Amazon EMR 集群并将数据加载到 Amazon Redshift 表中。Amazon EMR 集群必须继续运行，直到 COPY 命令完成。例如，不要将集群配置为自动终止。

> **Important**
>
> 如果在 COPY 完成前更改或删除了任何数据文件，则您可能会遇到意外结果，或者 COPY 操作可能失败。

在 COPY 命令中，指定 Amazon EMR 集群 ID 和 HDFS 文件路径及文件名。

```
copy sales
from 'emr://myemrclusterid/myoutput/part*' credentials
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

您可以使用通配符星号 (*) 和问号 (?) 作为文件名参数的一部分。例如，`part*` 加载文件 `part-0000`、`part-0001`，等等。如果您仅指定一个文件夹名称，则 COPY 将尝试加载该文件夹中的所有文件。

> **Important**
>
> 如果您使用通配符或仅使用文件夹名称，请确认不会加载不需要的文件，否则 COPY 命令将失败。例如，某些流程可能会将日志文件写入到输出文件夹。

从远程主机中加载数据

您可使用 COPY 命令从一个或多个远程主机并行加载数据，例如 Amazon EC2 实例或其他计算机。COPY 将连接到使用 SSH 的远程主机并在远程主机上运行命令以生成文本输出。

远程主机可以是 Amazon EC2 Linux 实例或配置为接受 SSH 连接的另一台 Unix 或 Linux 计算机。本指南假定您的远程主机是 Amazon EC2 实例。如果过程与其他计算机不同，指南中将会指出差别。

Amazon Redshift 可连接到多台主机，并可以打开到每台主机的多个 SSH 连接。Amazon Redshift 会通过每个连接发送一个唯一命令来生成到主机标准输出的文本输出，然后 Amazon Redshift 会像读取文本文件一样读取它。

开始前的准备工作

在开始之前，您应做好以下准备：

- 您可使用 SSH 连接的一个或多个主机（如 Amazon EC2 实例）。
- 主机上的数据来源。

  您将提供一些命令，Amazon Redshift 集群将在主机上运行这些命令以生成文本输出。在集群连接到主机后，COPY 命令将运行这些命令，从主机的标准输出中读取文本，并将数据并行加载到 Amazon Redshift 表中。文本输出必须采用 COPY 命令可提取的形式。有关更多信息，请参阅准备输入数据。
- 从您的计算机访问主机的权限。

  对于 Amazon EC2 实例，您将使用 SSH 连接来访问主机。您必须访问主机以将 Amazon Redshift 集群的公有密钥添加到主机的授权密钥文件。
- 正在运行的 Amazon Redshift 集群。

  有关如何启动集群的信息，请参阅 Amazon Redshift 入门指南。

加载数据的过程

本节指导您完成从远程主机加载数据的过程。以下各节提供每个步骤中必须完成的操作的详细信息。

- 步骤 1：检索集群公有密钥和集群节点 IP 地址

  公有密钥使 Amazon Redshift 集群节点能够建立与远程主机的 SSH 连接。您将使用每个集群节点的 IP 地址来配置主机安全组或防火墙，从而允许使用这些 IP 地址从 Amazon Redshift 集群进行访问。

- 步骤 2：将 Amazon Redshift 集群公有密钥添加到主机的授权密钥文件

  您将 Amazon Redshift 集群公有密钥添加到主机的授权密钥文件，以便让主机识别 Amazon Redshift 集群并接受 SSH 连接。

- 步骤 3：将主机配置为接受 Amazon Redshift 集群的所有 IP 地址

  对于 Amazon EC2，修改该实例的安全组，以添加接受 Amazon Redshift IP 地址的输入规则。对于其他主机，请修改防火墙，以便 Amazon Redshift 节点能够建立与远程主机的 SSH 连接。

- 步骤 4：获取主机的公有密钥

  您可以选择指定 Amazon Redshift 应使用公有密钥来标识主机。您必须找到公有密钥并将文本复制到您的清单文件中。

- 步骤 5：创建清单文件

  清单是一个 JSON 格式的文本文件，其中包含 Amazon Redshift 连接到主机并提取数据所需的详细信息。

- 步骤 6：将清单文件上载到 Amazon S3 桶

  Amazon Redshift 将读取清单文件并使用该信息连接到远程主机。如果 Amazon S3 桶不在您的 Amazon Redshift 集群所在的区域内，则必须使用 REGION 选项指定数据所在的区域。

- 步骤 7：运行 COPY 命令以加载数据

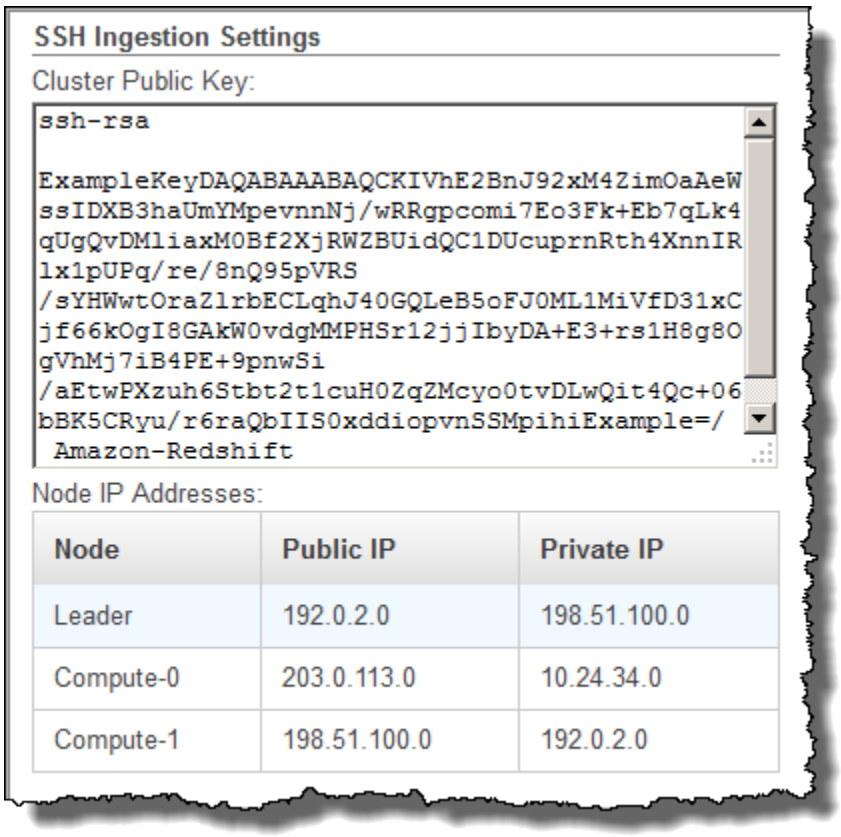
  从 Amazon Redshift 数据库运行 COPY 命令，以便将数据加载到 Amazon Redshift 表中。

步骤 1：检索集群公有密钥和集群节点 IP 地址

使用控制台为您的集群检索集群公有密钥和集群节点 IP 地址

1. 访问 Amazon Redshift 管理控制台。
2. 在导航窗格中选择集群链接。
3. 从列表中选择您的集群。
4. 找到 SSH Ingestion Settings 组。

   记下 Cluster Public Key 和 Node IP addresses 中的值。您将在后面的步骤中用到它们。



您将使用步骤 3 中的 IP 地址将主机配置为接受来自 Amazon Redshift 的连接。根据您连接到的主机的类型以及该主机是否在 VPC 中，您将使用公有 IP 地址或私有 IP 地址。

若要使用 Amazon Redshift CLI 检索您的集群的集群公有密钥和集群节点 IP 地址，请运行 describe-clusters 命令。

例如：

```
aws redshift describe-clusters --cluster-identifier <cluster-identifier>
```

响应将包含 ClusterPublicKey 以及私有和公有 IP 地址的列表，类似于以下内容：

```
{
    "Clusters": [
        {
            "VpcSecurityGroups": [],
            "ClusterStatus": "available",
            "ClusterNodes": [
                {
```

```
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "LEADER",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-0",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-1",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                }
            ],
            "AutomatedSnapshotRetentionPeriod": 1,
            "PreferredMaintenanceWindow": "wed:05:30-wed:06:00",
            "AvailabilityZone": "us-east-1a",
            "NodeType": "dc2.large",
            "ClusterPublicKey": "ssh-rsa AAAABexamplepublickey...Y3TAl Amazon-
Redshift",
             ...
             ...
}
```

若要使用 Amazon Redshift API 检索您的集群的集群公有密钥和集群节点 IP 地址，请使用 DescribeClusters 操作。有关更多信息，请参阅《Amazon Redshift CLI 指南》中的 describe-clusters 或《Amazon Redshift API 指南》中的 DescribeClusters。

步骤 2：将 Amazon Redshift 集群公有密钥添加到主机的授权密钥文件

您将集群公有密钥添加到每个主机的授权密钥文件，以便让主机识别 Amazon Redshift 并接受 SSH 连接。

要将 Amazon Redshift 集群公有密钥添加到主机的授权密钥文件

1. 使用 SSH 连接访问主机。

   有关使用 SSH 连接到实例的信息，请参阅《Amazon EC2 用户指南》中的连接到您的实例。

2. 从控制台或从 CLI 响应文本复制 Amazon Redshift 公有密钥。

3. 将公有密钥的内容复制并粘贴到远程主机上的 /home/<ssh_username>/.ssh/authorized_keys 文件中。<ssh_username> 必须与清单文件中的“username”字段的值匹配。请包括完整字符串（包含前缀“ssh-rsa”和后缀“Amazon-Redshift”）。例如：

```
ssh-rsa AAAACTP3isxgGzVWoIWpbVvRCOzYdVifMrh… uA70BnMHCaMiRdmvsDOedZDOedZ Amazon-Redshift
```

步骤 3：将主机配置为接受 Amazon Redshift 集群的所有 IP 地址

如果要使用 Amazon EC2 实例或 Amazon EMR 集群，请向主机的安全组添加入站规则以允许来自每个 Amazon Redshift 集群节点的流量。对于 Type，请选择在端口 22 上使用 TCP 协议的 SSH。对于源，请输入您在步骤 1：检索集群公有密钥和集群节点 IP 地址中检索的 Amazon Redshift 集群节点 IP 地址。有关添加规则到 Amazon EC2 安全组的信息，请参阅《Amazon EC2 用户指南》中的为您的实例授权入站流量。

请在以下情况下使用私有 IP 地址：

- 您具有一个没有位于 Virtual Private Cloud (VPC) 中的 Amazon Redshift 集群和一个 Amazon EC2 -Classic 实例，二者位于同一 AWS 区域中。
- 您具有一个位于 VPC 中的 Amazon Redshift 集群和一个 Amazon EC2 -VPC 实例，二者位于同一 AWS 区域和同一 VPC 中。

否则，请使用公有 IP 地址。

有关在 VPC 中使用 Amazon Redshift 的更多信息，请参阅《Amazon Redshift 管理指南》中的管理 Virtual Private Cloud (VPC) 中的集群。

步骤 4：获取主机的公有密钥

您可以选择在清单文件中提供主机的公有密钥以便让 Amazon Redshift 可以标识主机。COPY 命令不需要主机公有密钥，但出于安全原因，我们强烈建议使用公有密钥来帮助防止“中间人”攻击。

您可以在以下位置查找主机的公有密钥（其中 <ssh_host_rsa_key_name> 是主机的公有密钥的唯一名称）：

```
:  /etc/ssh/<ssh_host_rsa_key_name>.pub
```

> **Note**
>
> Amazon Redshift 仅支持 RSA 密钥。我们不支持 DSA 密钥。

在步骤 5 中创建清单文件时，您应将公有密钥的文本粘贴到清单文件条目中的“Public Key”字段中。

步骤 5：创建清单文件

COPY 命令可连接到使用 SSH 的多台主机，并可以与每台主机建立多个 SSH 连接。COPY 通过每个主机连接运行一个命令，然后将来自这些命令的输出并行加载到表中。清单文件是 Amazon Redshift 用于连接主机的文本文件，采用 JSON 格式。清单文件指定 SSH 主机端点以及将在主机上运行的用于将数据返回到 Amazon Redshift 的命令。另外，您还可以包含主机公有密钥、登录用户名和每个条目的 mandatory 标志。

在本地计算机上创建清单文件。在后一个步骤中，将文件上载到 Amazon S3。

清单文件应采用以下格式：

```
{
   "entries": [
     {"endpoint":"<ssh_endpoint_or_IP>",
       "command": "<remote_command>",
       "mandatory":true,
       "publickey": "<public_key>",
       "username": "<host_user_name>"},
     {"endpoint":"<ssh_endpoint_or_IP>",
       "command": "<remote_command>",
       "mandatory":true,
       "publickey": "<public_key>",
       "username": "host_user_name"}
    ]
}
```

该清单文件为每个 SSH 连接包含一个 "entries" 结构。每个条目表示一个 SSH 连接。您可以与单台主机建立多个连接或与多台主机建立多个连接。如上所示，字段名称和值均需要使用双引号。唯一的一个不需要双引号的值是 mandatory 字段的布尔值 **`true`** 或 **`false`**。

下面描述了清单文件中的字段。

endpoint

主机的 URL 地址或 IP 地址。例如，“ec2-111-222-333.compute-1.amazonaws.com”或“22.33.44.56”。

命令

将由主机运行的命令，用于生成文本或二进制（gzip、lzop 或 bzip2）输出。该命令可以是用户 "host_user_name" 有权运行的任何命令。命令可以是像打印文件这样简单的命令，也可以查询数据库或启动脚本。输出（文本文件、gzip 二进制文件、lzop 二进制文件或 bzip2 二进制文件）必须采用 Amazon Redshift COPY 命令可摄取的形式。有关更多信息，请参阅 准备输入数据。

publickey

（可选）主机的公有密钥。如果提供了公有密钥，Amazon Redshift 将使用它来标识主机。如果未提供公有密钥，Amazon Redshift 将不会尝试主机标识。例如，如果远程主机的公有密钥是 ssh-rsa AbcCbaxxx…xxxDHKJ root@amazon.com，请在公有密钥字段中输入以下文本：AbcCbaxxx…xxxDHKJ。

mandatory

（可选）指示在连接失败的情况下 COPY 命令是否应失败。默认为 false。如果 Amazon Redshift 未成功建立至少一个连接，COPY 命令将失败。

username

（可选）将用于登录到主机系统并运行远程命令的用户名。用户登录名必须与步骤 2 中用于将公有密钥添加到主机的授权密钥文件的登录名相同。默认用户名为“redshift”。

以下示例显示了用于与同一主机建立四个连接并通过每个连接运行不同的命令的完整清单：

```
{
  "entries": [
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata1.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"},
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata2.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"},
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
```

```
          "command": "cat loaddata3.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"},
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata4.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"}
     ]
}
```

步骤 6：将清单文件上载到 Amazon S3 桶

将清单文件上载到 Amazon S3 桶。如果 Amazon S3 桶不在您的 Amazon Redshift 集群所在的 AWS 区域内，则必须使用 REGION 选项指定清单所在的 AWS 区域。有关创建 Amazon S3 桶并上载文件的信息，请参阅 Amazon Simple Storage Service 用户指南。

步骤 7：运行 COPY 命令以加载数据

运行 COPY 命令以连接到主机并将数据加载到 Amazon Redshift 表中。在 COPY 命令中，指定清单文件的显式 Amazon S3 对象路径并包含 SSH 选项。例如，

```
copy sales
from 's3://mybucket/ssh_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|'
ssh;
```

> **Note**
>
> 如果使用自动压缩，则 COPY 命令将执行两次数据读取，这意味着它将运行两次远程命令。第一次读取用于提供压缩分析的样本，第二次读取实际加载数据。如果运行远程命令两次可能会由于潜在副作用而导致问题，则应关闭自动压缩。要关闭自动压缩，请运行 COPY 命令，同时将 COMPUPDATE 选项设置为 OFF。有关更多信息，请参阅 使用自动压缩加载表。

从 Amazon DynamoDB 表中加载数据

您可以使用 COPY 命令从单个 Amazon DynamoDB 表加载带数据的表。

> **Important**
>
> 提供数据的 Amazon DynamoDB 表必须在与集群相同的 AWS 区域中创建，除非您使用 REGION 选项指定 Amazon DynamoDB 表所在的 AWS 区域。

COPY 命令使用 Amazon Redshift 大规模并行处理 (MPP) 架构从 Amazon DynamoDB 表中并行读取和加载数据。您可以通过在 Amazon Redshift 表上设置分配样式，从而最大程度地利用并行处理。有关更多信息，请参阅 使用数据分配样式。

> **Important**
>
> 当 COPY 命令从 Amazon DynamoDB 表读取数据时，生成的数据传输是该表的预配置吞吐量的一部分。

为避免过度消耗预配置的读取吞吐量，我们建议您不要从生产环境中的 Amazon DynamoDB 表加载数据。如果您确实需要从生产表加载数据，我们建议您设置远低于未使用预配置吞吐量的平均百分比的 READRATIO 选项。低 READRATIO 设置有助于最大程度地减少限制问题。要使用 Amazon DynamoDB 表的所有预配置吞吐量，请将 READRATIO 设置为 100。

COPY 命令使用以下规则，将从 DynamoDB 表检索到的项目中的属性名称与现有 Amazon Redshift 表中的列名称匹配：

- Amazon Redshift 表列与 Amazon DynamoDB 项目属性匹配且不区分大小写。如果 DynamoDB 表中的项目包含仅大小写不同的多个属性（如 Price 和 PRICE），COPY 命令将失败。
- 与 Amazon DynamoDB 表中的属性不匹配的 Amazon Redshift 表列作为 NULL 或空值加载，具体取决于使用 COPY 命令中的 EMPTYASNULL 选项指定的值。
- 将弃用与 Amazon Redshift 表中的列不匹配的 Amazon DynamoDB 属性。属性是在匹配前读取的，因此即使是已弃用的属性也会消耗该表的一部分预配置吞吐量。
- 仅支持具有标量 STRING 和 NUMBER 数据类型的 Amazon DynamoDB 属性。不支持 Amazon DynamoDB BINARY 和 SET 数据类型。如果 COPY 命令尝试加载某个具有不支持的数据类型的属性，该命令将失败。如果该属性与 Amazon Redshift 表列不匹配，则 COPY 不会尝试加载它，它也不会引发错误。

COPY 命令使用以下语法从 Amazon DynamoDB 表加载数据：

```
copy <redshift_tablename> from 'dynamodb://<dynamodb_table_name>'
authorization
readratio '<integer>';
```

authorization 的值是访问 Amazon DynamoDB 表所需的 AWS 凭证。如果这些凭证对应于某个用户，则该用户必须拥有对要加载的 Amazon DynamoDB 表执行 SCAN 和 DESCRIBE 的权限。

authorization 的值为您的集群提供访问 Amazon DynamoDB 表所需的 AWS 授权。必须包括对要加载的 Amazon DynamoDB 表执行 SCAN 和 DESCRIBE 的权限。有关所需权限的更多信息，请参阅COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限。进行身份验证的首选方法是为具有必要权限的 IAM 角色指定 IAM_ROLE 参数并提供 Amazon 资源名称（ARN）。有关更多信息，请参阅 基于角色的访问控制。

要使用 IAM_ROLE 参数进行身份验证，请替换 *<aws-account-id>* 和 *<role-name>*，如以下语法中所示。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

以下示例使用 IAM 角色进行身份验证。

```
copy favoritemovies
from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

有关其他授权选项的更多信息，请参阅授权参数。

如果您要验证您的数据而不实际加载表，请对 COPY 命令使用 NOLOAD 选项。

以下示例加载包含来自 DynamoDB 表 my-favorite-movies-table 的数据的 FAVORITEMOVIES 表。读取活动可消耗多达 50% 的预配置吞吐量。

```
copy favoritemovies from 'dynamodb://my-favorite-movies-table'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

为了最大程度地增加吞吐量，COPY 命令将跨集群中的计算节点从 Amazon DynamoDB 表并行加载数据。

预置的吞吐量与自动压缩

默认情况下，COPY 命令将在您指定没有压缩编码的空目标表时应用自动压缩。自动压缩分析最初将对 Amazon DynamoDB 表中的很多行进行采样。样本大小基于 COMPROWS 参数的值。默认值为每个切片 100000 行。

采样后，将弃用示例行并加载整个表。因此，很多行会被读取两次。有关自动压缩的工作方式的更多信息，请参阅使用自动压缩加载表。

> **Important**
>
> 当 COPY 命令从 Amazon DynamoDB 表读取数据（包括用于采样的行）时，生成的数据传输是该表的预配置吞吐量的一部分。

从 Amazon DynamoDB 中加载多字节数据

如果您的数据包含非 ASCII 多字节字符（例如中文或西里尔语字符），则必须将该数据加载到 VARCHAR 列。VARCHAR 数据类型支持四字节的 UTF-8 字符，而 CHAR 数据类型仅接受单字节的 ASCII 字符。您不能将五字节或更长的字符加载到 Amazon Redshift 表中。有关 CHAR 和 VARCHAR 的更多信息，请参阅数据类型。

验证是否正确加载了数据

在完成加载操作后，请查询 STL_LOAD_COMMITS 系统表以验证所需文件是否已加载。在同一事务中运行 COPY 命令和加载验证，以便在加载出现问题时可以回滚整个事务。

以下查询将返回用于加载 TICKIT 数据库中的表的条目：

```
select query, trim(filename) as filename, curtime, status
from stl_load_commits
where filename like '%tickit%' order by query;

 query |         filename          |          curtime           | status
-------+---------------------------+----------------------------+--------
 22475 | tickit/allusers_pipe.txt  | 2013-02-08 20:58:23.274186 |      1
 22478 | tickit/venue_pipe.txt     | 2013-02-08 20:58:25.070604 |      1
 22480 | tickit/category_pipe.txt  | 2013-02-08 20:58:27.333472 |      1
 22482 | tickit/date2008_pipe.txt  | 2013-02-08 20:58:28.608305 |      1
 22485 | tickit/allevents_pipe.txt | 2013-02-08 20:58:29.99489  |      1
```

```
 22487 | tickit/listings_pipe.txt  | 2013-02-08 20:58:37.632939 |      1
 22489 | tickit/sales_tab.txt      | 2013-02-08 20:58:37.632939 |      1
(6 rows)
```

验证输入数据

要在实际加载 Amazon S3 输入文件或 Amazon DynamoDB 表中的数据之前验证这些数据，请对 COPY 命令使用 NOLOAD 选项。将 NOLOAD 与您要用于加载数据的相同 COPY 命令和选项结合使用。NOLOAD 将检查所有数据的完整性而不用将其加载到数据库中。如果您尝试加载数据，NOLOAD 选项卡会显示将出现的任何错误。

例如，如果您为输入文件指定了不正确的 Amazon S3 路径，则 Amazon Redshift 将显示以下错误。

```
ERROR:  No such file or directory
DETAIL:
-----------------------------------------------
Amazon Redshift error:  The specified key does not exist
code:      2
context:   S3 key being read :
location:  step_scan.cpp:1883
process:   xenmaster [pid=22199]
-----------------------------------------------
```

要针对错误消息排错，请参阅加载错误参考。

有关使用 NOLOAD 选项的示例，请参阅 带有 NOLOAD 选项的 COPY 命令。

使用自动压缩加载表

主题

- 自动压缩的工作方式
- 自动压缩示例

您可以根据您自己的数据评估，手动将压缩编码应用于表中的各个列。或者，您可以使用 COPY 命令并将 COMPUPDATE 设置为 ON，以便根据示例数据自动分析和应用压缩。

您可以在创建并加载全新的表时使用自动压缩。COPY 命令可执行压缩分析。您也可以通过以下方式执行压缩分析而不加载数据或更改对表的压缩：对已填充的表运行 ANALYZE COMPRESSION 命令。

例如，您可以在想要分析对表的压缩以供将来使用时运行 ANALYZE COMPRESSION 命令，同时保留现有数据定义语言 (DDL) 语句。

自动压缩将在选择压缩编码时平衡整体性能。如果排序键列的压缩率远高于同一查询中的其他列，则范围受限扫描的执行效果可能会很差。因此，自动压缩会跳过排序键列上的数据分析阶段，并保留用户定义的编码类型。

如果您尚未明确定义某种编码，则自动压缩会选择 RAW 编码。ANALYZE COMPRESSION 的行为相同。要获得最佳查询性能，请考虑将 RAW 用于排序键。

自动压缩的工作方式

如果 COMPUPDATE 参数为 ON，每次为空目标表运行 COPY 命令并且所有表列具有 RAW 编码或没有编码时，COPY 命令将会应用自动压缩。

要将自动压缩应用于空表而不管其当前压缩编码如何，请运行 COPY 命令，同时将 COMPUPDATE 选项设置为 ON。要关闭自动压缩，请运行 COPY 命令，同时将 COMPUPDATE 选项设置为 OFF。

您不能将自动压缩应用于已包含数据的表。

> **Note**
>
> 自动压缩分析要求加载数据中有足够多的行（每个切片至少 100000 行），以生成有意义的样本。

作为加载事务的一部分，自动压缩将在后台执行以下操作：

1. 从输入文件加载初始的行样本。样本大小基于 COMPROWS 参数的值。默认值为 100000。
2. 为每个列选择压缩选项。
3. 从表中删除样本行。
4. 以所选的压缩编码重新创建表。
5. 使用新编码加载并压缩整个输入文件。

运行 COPY 命令后，表将会完全加载、压缩并可供使用。如果您在以后加载了更多数据，追加的行将根据现有编码进行压缩。

如果您只是想执行压缩分析，请运行 ANALYZE COMPRESSION，这比运行完整的 COPY 操作效率更高。然后，您可以评估结果以决定是使用自动压缩还是手动重新创建表。

只有 COPY 命令才支持自动压缩。或者，您也可以在创建表时手动应用压缩编码。有关手动压缩编码的信息，请参阅使用列压缩。

自动压缩示例

在本示例中，假定 TICKIT 数据库包含 LISTING 表的一个名为 BIGLIST 的副本，并且您希望在该表加载了大约 300 万行时对其应用自动压缩。

加载和自动压缩表

1. 确保表是空的。您只能将自动压缩应用于空表：

```
truncate biglist;
```

2. 使用单个 COPY 命令加载表。尽管表是空的，但之前可能已指定某种编码。为便于 Amazon Redshift 执行压缩分析，请将 COMPUPDATE 参数设置为 ON。

```
copy biglist from 's3://mybucket/biglist.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' COMPUPDATE ON;
```

   由于未指定 COMPROWS 选项，因此将使用默认和推荐的样本大小（100,000 行/切片）。

3. 查看 BIGLIST 表的新 schema 以了解自动选择的编码方案。

```
select "column", type, encoding
from pg_table_def where tablename = 'biglist';

     Column     |            Type             | Encoding
----------------+-----------------------------+----------
 listid         | integer                     | az64
 sellerid       | integer                     | az64
 eventid        | integer                     | az64
 dateid         | smallint                    | none
 numtickets     | smallint                    | az64
 priceperticket | numeric(8,2)                | az64
 totalprice     | numeric(8,2)                | az64
 listtime       | timestamp without time zone | az64
```

4. 验证是否已加载所需的行数：

```
select count(*) from biglist;
```

```
count
---------
3079952
(1 row)
```

以后使用 COPY 或 INSERT 语句向此表追加行时，将应用相同的压缩编码。

针对窄表优化存储

如果您有一个列数非常少但行数非常多的表，三个隐藏的元数据标识列（INSERT_XID、DELETE_XID 和 ROW_ID）将为表消耗不成正比的磁盘空间。

为了优化隐藏列的压缩，请尽可能在单个 COPY 事务中加载表。如果您使用多个单独的 COPY 命令加载表，INSERT_XID 列的压缩效果不会很好。如果您使用了多个 COPY 命令，则必须执行 vacuum 操作，但它不会改进 INSERT_XID 的压缩。

加载默认列值

您可以选择在 COPY 命令中定义一个列列表。如果列列表忽略了表中的某个列，则 COPY 将用 CREATE TABLE 命令中指定的 DEFAULT 选项提供的值加载该列，或加载 NULL（如果未指定 DEFAULT 选项）。

如果 COPY 尝试将 NULL 分配到一个定义为 NOT NULL 的列，COPY 命令将失败。有关分配 DEFAULT 选项的信息，请参阅 CREATE TABLE。

当从 Amazon S3 上的数据文件加载时，列列表中的列必须与数据文件中字段的顺序相同。如果数据文件中的某个字段在列列表中没有对应的列，COPY 命令将失败。

从 Amazon DynamoDB 表加载时，顺序并不重要。将弃用与 Amazon Redshift 表中的列不匹配的 Amazon DynamoDB 属性中的任何字段。

当使用 COPY 命令将 DEFAULT 值加载到表中时，将适用以下限制：

- 如果 IDENTITY 列包含在列列表中，则还必须在 COPY 命令中指定 EXPLICIT_IDS 选项，否则 COPY 命令将失败。同样，如果列列表省略了 IDENTITY 列，并且指定了 EXPLICIT_IDS 选项，COPY 操作将失败。
- 由于给定列的已计算 DEFAULT 表达式对所有已加载行是相同的，因此，使用 RANDOM() 函数的 DEFAULT 表达式会向所有行分配相同的值。

- 包含 CURRENT_DATE 或 SYSDATE 的 DEFAULT 表达式将设置为当前事务的时间戳。

有关示例，请参阅 COPY 示例中的“从带有默认值的文件加载数据”。

解决数据加载问题

主题

本节提供了有关识别和解决数据加载错误的信息。

S3ServiceException 错误

最常见的 s3ServiceException 错误由以下原因导致：凭证字符串的格式有误或凭证字符串不正确，将集群和桶放在不同的 AWS 区域中以及 Amazon S3 权限不足。

本节提供了每种类型的错误的排查信息。

凭证字符串无效

如果凭证字符串的格式不正确，您将收到以下错误消息：

```
ERROR: Invalid credentials. Must be of the format: credentials
'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>
[;token=<temporary-session-token>]'
```

请验证凭证字符串不包含任何空格或换行符，并且括在单引号中。

访问密钥 ID 无效

如果您的访问密钥 ID 不存在，您将收到以下错误消息：

```
[Amazon](500310) Invalid operation: S3ServiceException:The AWS Access Key Id you
 provided does not exist in our records.
```

这通常是复制和粘贴错误。请验证是否正确输入了访问密钥 ID。此外，如果您使用的是临时会话密钥，请检查是否已设置 token 的值。

秘密访问密钥无效

如果您的秘密访问密钥不正确，您将收到以下错误消息：

```
[Amazon](500310) Invalid operation: S3ServiceException:The request signature we
 calculated does not match the signature you provided.
Check your key and signing method.,Status 403,Error SignatureDoesNotMatch
```

这通常是复制和粘贴错误。请验证是否正确输入了秘密访问密钥，并验证它是否为访问密钥 ID 对应的正确密钥。

桶位于不同的区域

COPY 命令中指定的 Amazon S3 桶必须位于集群所在的 AWS 区域。如果 Amazon S3 桶和集群位于不同的区域中，则会收到类似于以下内容的错误：

```
ERROR: S3ServiceException:The bucket you are attempting to access must be addressed
 using the specified endpoint.
```

您可以通过以下方式在特定区域中创建 Amazon S3 桶：在使用 Amazon S3 管理控制台创建桶时选择该区域，或者在使用 Amazon S3 API 或 CLI 创建桶时指定端点。有关更多信息，请参阅 将文件上载到 Amazon S3。

有关 Amazon S3 区域的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的访问桶。

或者，您也可以在 COPY 命令中使用 REGION 选项以指定区域。

访问被拒绝

如果用户没有足够的权限，您将收到以下错误消息：

```
ERROR: S3ServiceException:Access Denied,Status 403,Error AccessDenied
```

一个可能的原因是，凭证标识的用户没有对 Amazon S3 桶的 LIST 和 GET 访问权限。有关其他原因，请参阅《Amazon Simple Storage Service 用户指南》中的排查 Amazon S3 中的拒绝访问（403 禁止）错误。

有关管理用户对桶的访问权限的信息，请参阅《Amazon Simple Storage Service 用户指南》中的 Amazon S3 中的身份和访问管理。

用于解决数据加载问题的系统表

以下 Amazon Redshift 系统表可能有助于排查数据加载问题：

- 查询 STL_LOAD_ERRORS 以发现在特定加载期间发生的错误。
- 查询 STL_FILE_SCAN 以查看特定文件的加载时间或了解是否甚至读取了某个特定文件。
- 查询 STL_S3CLIENT_ERROR 以查找有关从 Amazon S3 传输数据时遇到的错误的详细信息。

查找和诊断加载错误

1. 创建视图或定义返回有关加载错误的详细信息的查询。以下示例将 STL_LOAD_ERRORS 表联接到 STV_TBL_PERM 表以将表 ID 与实际表名称进行匹配。

```
create view loadview as
(select distinct tbl, trim(name) as table_name, query, starttime,
trim(filename) as input, line_number, colname, err_code,
trim(err_reason) as reason
from stl_load_errors sl, stv_tbl_perm sp
where sl.tbl = sp.id);
```

2. 将 COPY 命令中的 MAXERRORS 选项设置为足够大的值，以使 COPY 能够返回有关您的数据的有用值。如果 COPY 遇到错误，则会出现一条错误消息，指示您参阅 STL_LOAD_ERRORS 表以了解详细信息。
3. 查询 LOADVIEW 视图以查看错误详细信息。例如：

```
select * from loadview where table_name='venue';
```

```
  tbl   | table_name | query |         starttime
--------+------------+-------+----------------------------
 100551 | venue      | 20974 | 2013-01-29 19:05:58.365391

|     input      | line_number | colname | err_code |       reason
+----------------+-------------+-------+----------+--------------------
| venue_pipe.txt |           1 |     0 |     1214 | Delimiter not found
```

4. 根据视图返回的信息修复输入文件或加载脚本中的问题。要观察的一些典型加载错误包括：

- 表中的数据类型与输入数据字段中的值不匹配。
- 表中的列数与输入数据中的字段数不匹配。
- 引号不匹配。Amazon Redshift 支持单引号和双引号；但是，必须适当平衡这些引号。
- 输入文件中的日期/时间数据的格式不正确。
- 输入文件中的值超出范围（对于数字列）。
- 某个列的不同值的数量超出了对其压缩编码的限制。

多字节字符加载错误

带有 CHAR 数据类型的列仅接受单字节 UTF-8 字符（字节值最大为 127，即十六进制的 7F），它也是 ASCII 字符集。VARCHAR 列接受多字节 UTF-8 字符，最多四个字节。有关更多信息，请参阅 字符类型。

如果加载数据中的某一行包含对列数据类型无效的字符，则 COPY 将返回一个错误并在 STL_LOAD_ERRORS 系统日志表中记录一行，错误编号为 1220。ERR_REASON 字段包含无效字符的十六进制字节序列。

修复加载数据中的无效字符的替代方法是在加载过程中替换无效字符。要替换无效的 UTF-8 字符，请在 COPY 命令中指定 ACCEPTINVCHARS 选项。如果设置了 ACCEPTINVCHARS 选项，那么指定的字符将替换代码点。如果未设置 ACCEPTINVCHARS 选项，Amazon Redshift 会接受这些字符作为有效的 UTF-8。有关更多信息，请参阅 ACCEPTINVCHARS。

以下代码点列表是有效的 UTF-8，如果未设置 ACCEPTINVCHARS 选项，COPY 操作不会返回错误。但是，这些代码点是无效字符。您可以使用 ACCEPTINVCHARS 选项用指定的字符替换代码点。这些代码点的值范围为 `0xFDD0` 到 `0xFDEF`，最高值为 `0x10FFFF`，以 `FFFE` 或 `FFFF` 结尾：

- `0xFFFE`, `0x1FFFE`, `0x2FFFE`, ..., `0xFFFFE`, `0x10FFFE`
- `0xFFFF`, `0x1FFFF`, `0x2FFFF`, ..., `0xFFFFF`, `0x10FFFF`

以下示例显示了 COPY 尝试将 UTF-8 字符 `e0 a1 c7a4` 加载到 CHAR 列中时的错误原因。

```
Multibyte character not supported for CHAR
(Hint: Try using  VARCHAR). Invalid char: e0 a1 c7a4
```

如果错误与 VARCHAR 数据类型相关，则错误原因将包含错误代码和无效的 UTF-8 十六进制序列。以下示例显示了 COPY 尝试将 UTF-8 a4 加载到 VARCHAR 字段中时的错误原因。

```
String contains invalid or unsupported UTF-8 codepoints.
Bad UTF-8 hex sequence: a4 (error 3)
```

下表列出了 VARCHAR 加载错误的描述和建议解决方法。如果出现了以下错误之一，请将该字符替换为有效的 UTF-8 代码序列或删除该字符。

错误代码	描述
1	UTF-8 字节序列超出了 VARCHAR 支持的最大四字节。
2	UTF-8 字节序列不完整。COPY 在字符串结束之前没有找到多字节字符所需数量的连续字节。
3	UTF-8 单字节字符超出范围。开头字节不能是 254、255 或 128 和 191 之间（含）的任何字符。
4	字节序列中的结尾字节的值超出范围。连续字节必须在 128 和 191 之间（含）。
5	UTF-8 字符作为代理保留。代理代码点（U+D800 至 U+DFFF）无效。
8	字节序列超出最大 UTF-8 码位。
9	UTF-8 字节序列没有匹配的码位。

加载错误参考

如果在从文件中加载数据时出现任何错误，请查询 STL_LOAD_ERRORS 表以识别错误并确定可能的解释。下表列出了在数据加载期间可能出现的所有错误代码：

加载错误代码

错误代码	描述
1200	未知的分析错误。请联系客户支持人员。
1201	在输入文件中找不到字段分隔符。
1202	输入数据拥有的列数多于 DDL 中定义的数量。

错误代码	描述
1203	输入数据拥有的列数少于 DDL 中定义的数量。
1204	输入数据超出了数据类型的可接受范围。
1205	日期格式无效。请参阅 DATEFORMAT 和 TIMEFORMAT 字符串以了解有效的格式。
1206	时间戳格式无效。请参阅 DATEFORMAT 和 TIMEFORMAT 字符串以了解有效的格式。
1207	数据包含了超出所需范围 0-9 的值。
1208	FLOAT 数据类型格式错误。
1209	DECIMAL 数据类型格式错误。
1210	BOOLEAN 数据类型格式错误。
1211	输入行未包含数据。
1212	找不到加载文件。
1213	指定为 NOT NULL 的字段未包含数据。
1214	找不到分隔符。
1215	CHAR 字段错误。
1216	输入行无效。
1217	身份列的值无效。
1218	在使用 NULL AS '\0' 时，某个包含 null 终止符（NUL，即 UTF-8 的 0000）的字段包含了多个字节。
1219	UTF-8 十六进制数包含一个无效位。
1220	字符串包含无效或不受支持的 UTF-8 码位。
1221	文件的编码与 COPY 命令中指定的编码不同。

错误代码	描述
1222	整数值溢出错误。
1223	数据类型无效。
1224	输入数据是格式不正确的 JSON 格式或 super 数据类型。
8001	包含 MANIFEST 参数的 COPY 需要 Amazon S3 对象的完整路径。
9005	指定的结束键无效。

从 Amazon S3 持续摄取文件（预览版）

这是面向预览版中的自动复制（SQL COPY JOB）的预发行文档。文档和特征都可能会更改。我们建议您仅在测试环境中使用此特征，不要在生产环境中使用。公开预览将于 2024 年 7 月 31 日结束。在预览结束两周后，将自动删除预览集群。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

Note

您可以在预览版中创建 Amazon Redshift 集群，以便测试 Amazon Redshift 的新功能。您无法在生产环境中使用这些功能，也无法将预览版集群移动到生产集群或另一个跟踪上的集群。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

在预览版中创建集群

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择预置集群控制面板，然后选择集群。列出您的账户在当前 AWS 区域区域中的集群。列表中的各个列中显示了每个集群的一部分属性。
3. 集群列表页面上会显示一个横幅，其中介绍了预览版。选择创建预览版集群按钮以打开创建集群页面。
4. 输入集群的属性。选择包含要测试的功能的预览版跟踪。我们建议输入的集群名称指明要对该集群进行预览版跟踪。为您的集群选择选项，包括标记为 -preview 的选项，用于要测

试的功能。有关创建集群的一般信息，请参阅《Amazon Redshift 管理指南》中的创建集群。

5. 选择创建集群以在预览模式下创建集群。
6. 当您的预览集群可用时，使用 SQL 客户端加载和查询数据。

您的集群必须使用名为 `preview_2023` 的预览跟踪来创建。使用新集群进行测试，不支持将集群还原到此跟踪。自动复制功能不适用于 Amazon Redshift Serverless 工作组。
在以下 AWS 区域 中提供此预览：

- 美国东部（俄亥俄州）区域 (us-east-2)
- 美国东部（弗吉尼亚州北部）区域 (us-east-1)
- 美国西部（俄勒冈州）区域 (us-west-2)
- 亚太地区（东京）区域 (ap-northeast-1)
- 欧洲地区（斯德哥尔摩）区域 (eu-north-1)
- 欧洲地区（爱尔兰）区域 (eu-west-1)

您可以使用 COPY JOB 将数据从存储在 Amazon S3 中的文件加载到您的 Amazon Redshift 表中。Amazon Redshift 会检测新的 Amazon S3 文件何时添加到 COPY 命令中指定的路径。然后，无需创建外部数据摄取管道，即可自动运行 COPY 命令。Amazon Redshift 会跟踪已加载哪些文件。Amazon Redshift 确定每个 COPY 命令一起进行批处理的文件数。您可以在系统视图中看到生成的 COPY 命令。

COPY JOB 只需定义一次。未来运行使用相同的参数。

您可以使用 CREATE、LIST、SHOW、DROP、ALTER 和 RUN 作业的选项来管理加载操作。有关更多信息，请参阅 COPY JOB（预览版）。

您可以查询系统视图以查看 COPY JOB 状态和进度。提供的视图如下：

- SYS_COPY_JOB（预览版） – 为当前定义的每个 COPY JOB 包含一行。
- STL_LOAD_ERRORS – 包含 COPY 命令中的错误。
- STL_LOAD_COMMITS – 包含用于对 COPY 命令数据加载进行故障排除的信息。
- SYS_LOAD_HISTORY – 包含 COPY 命令的详细信息。
- SYS_LOAD_ERROR_DETAIL – 包含 COPY 命令错误的详细信息。

要获取 COPY JOB 所加载文件的列表，请运行以下示例，替换 *<job_id>*：

```
SELECT job_id, job_name, data_source, copy_query,filename,status, curtime
FROM sys_copy_job copyjob
JOIN stl_load_commits loadcommit
ON copyjob.job_id = loadcommit.copy_job_id
WHERE job_id = <job_id>;
```

使用 DML 命令更新表

Amazon Redshift 支持标准数据操作语言 (DML) 命令（INSERT、UPDATE 和 DELETE），您可以使用这些命令修改表中的行。您还可使用 TRUNCATE 命令执行快速批量删除。

> **Note**
>
> 我们强烈建议您使用 COPY 命令来加载大量数据。使用单个 INSERT 语句填充表可能过于缓慢。此外，如果您的数据在其他 Amazon Redshift 数据库表中已经存在，请使用 INSERT INTO ... SELECT FROM 或 CREATE TABLE AS 来提高性能。有关更多信息，请参阅 INSERT 或 CREATE TABLE AS。

如果您插入、更新或删除了表中的大量行（相对于更改前的行数），请在完成后对表运行 ANALYZE 和 VACUUM 命令。如果在经过一段时间之后您的应用程序中累积了大量小更改，则可能需要安排定期运行 ANALYZE 和 VACUUM 命令。有关更多信息，请参阅分析表和对表执行 vacuum 操作。

更新和插入新数据

您可以使用 MERGE 命令高效地向现有表中添加新数据。执行合并操作，方法是创建暂存表，然后使用本节中描述的方法之一从暂存表更新目标表。有关 MERGE 命令的更多信息，请参阅MERGE。

主题

合并示例使用名为 TICKIT 数据集的 Amazon Redshift 示例数据集。作为先决条件，您可以按照开始使用常见数据库任务中提供的说明设置 TICKIT 表和数据。有关示例数据集的更多详细信息可在示例数据库中找到。

合并方法 1：替换现有行

如果您要覆盖目标表中的所有列，执行合并的最快方法是替换现有行。这将通过使用内部联接删除将要更新的行，从而仅需扫描目标表一次。在删除某些行后，通过从暂存表执行单个插入操作即可使用新行来替换它们。

请在满足以下所有条件时用此方法：

- 您的目标表和暂存表包含相同的列。
- 您要将目标表列中的所有数据替换为所有的暂存表列。
- 您将在合并操作中使用暂存表中的所有行。

如果未满足以上任一条件，请使用下一节所述的“合并方法 2：在不使用 MERGE 的情况下指定列列表”。

如果您不使用暂存表中的所有行，则使用 WHERE 子句筛选 DELETE 和 INSERT 语句，以忽略未更改的行。但是，如果暂存表中的大多数行不会参与合并，我们建议通过单独的步骤中执行 UPDATE 和 INSERT，如本节后面所述。

合并方法 2：在不使用 MERGE 的情况下指定列列表

使用此方法可更新目标表中的特定列，而不是覆盖所有行。此方法比前一个方法消耗的时间更长，因为它需要执行一个额外的更新步骤，并且不使用 MERGE 命令。请在满足以下所有条件时用此方法：

- 并非要更新目标表中的所有列。
- 暂存表中的大多数行在更新中将不会使用。

创建临时的暂存表

暂存表 是一个临时表，用来保存将用于对目标表 进行更改（包括更新和插入）的所有数据。

合并操作需要在暂存表和目标表之间建立联接。要并置联接行，请将暂存表的分配键的列设置为与目标表的分配键的列相同。例如，如果目标表使用一个外键列作为其分配键，则对暂存表的分配键使用相同的列。如果使用 CREATE TABLE LIKE 语句创建一个暂存表，则该暂存表将从父表继承分配键。如果使用 CREATE TABLE AS 语句，则新表不会继承分配键。有关更多信息，请参阅使用数据分配样式。

如果分配键与主键不相同且在合并操作中未更新分配键，则在分配键列上添加一个冗余联接谓词以启用并置联接。例如：

```
where target.primarykey = stage.primarykey
and target.distkey = stage.distkey
```

要验证查询是否将使用并置联接，请使用 EXPLAIN 运行查询并检查所有联接上是否有 DS_DIST_NONE。有关更多信息，请参阅评估查询计划。

替换现有行以执行合并操作

当您运行此过程中详述的合并操作时，请将所有步骤（但创建和删除临时暂存表除外）放在单个事务中。如果任何步骤失败，事务将回滚。使用单个事务还将减少提交次数，从而节省时间和资源。

通过替换现有行执行合并操作

1. 创建暂存表，然后使用要合并的数据填充它，如下面的伪代码所示。

   ```
   create temp table stage (like target);

   insert into stage
   select * from source
   where source.filter = 'filter_expression';
   ```

2. 使用 MERGE 执行与暂存表的内部联接，以更新目标表中与暂存表匹配的行，然后将所有与暂存表不匹配的剩余行插入到目标表中。

   我们建议您在单个 MERGE 命令中运行更新和插入操作。

   ```
   MERGE INTO target
   USING stage [optional alias] on (target.primary_key = stage.primary_key)
   WHEN MATCHED THEN
   UPDATE SET col_name1 = stage.col_name1 , col_name2= stage.col_name2, col_name3 =
    {expr}
   WHEN NOT MATCHED THEN
   INSERT (col_name1 , col_name2, col_name3) VALUES (stage.col_name1, stage.col_name2,
    {expr});
   ```

3. 删除暂存表。

   ```
   drop table stage;
   ```

通过在不使用 MERGE 的情况下指定列列表执行合并操作

当您运行此过程中详述的合并操作时，请将所有步骤放在单个事务中。如果任何步骤失败，事务将回滚。使用单个事务还将减少提交次数，从而节省时间和资源。

通过指定列列表执行合并操作

1. 将整个操作放在单个事务块中。

```
begin transaction;
…
end transaction;
```

2. 创建暂存表，然后使用要合并的数据填充它，如下面的伪代码所示。

```
create temp table stage (like target);
insert into stage
select * from source
where source.filter = 'filter_expression';
```

3. 使用与暂存表的内部联接更新目标表。

   - 在 UPDATE 子句中，显式列出要更新的列。
   - 执行与暂存表的内部联接。
   - 如果分配键与主键不同且分配键将不会更新，请在分配键上添加一个冗余联接。要验证查询是否将使用并置联接，请使用 EXPLAIN 运行查询并检查所有联接上是否有 DS_DIST_NONE。有关更多信息，请参阅评估查询计划。
   - 如果目标表按时间戳排序，请添加谓词以对目标表使用限定范围的扫描。有关更多信息，请参阅设计查询的 Amazon Redshift 最佳实践。
   - 如果您将不在合并中使用所有行，请添加一个子句以筛选要更改的行。例如，添加针对一个或多个列的不等于筛选器以排除未更改的行。
   - 将更新、删除和插入操作放在单个事务块中，以便在出现问题时回滚所有内容。

   例如：

```
begin transaction;

update target
set col1 = stage.col1,
```

```
col2 = stage.col2,
col3 = 'expression'
from stage
where target.primarykey = stage.primarykey
and target.distkey = stage.distkey
and target.col3 > 'last_update_time'
and (target.col1 != stage.col1
or target.col2 != stage.col2
or target.col3 = 'filter_expression');
```

4. 使用与目标表的内部联接从暂存表中删除不需要的行。目标表中的一部分行已经与暂存表中的对应行匹配，而另一些行已在上一个步骤中更新。在任一情况下，都不需要插入它们。

```
delete from stage
using target
where stage.primarykey = target.primarykey;
```

5. 插入暂存表中的剩余行。在 VALUES 子句中使用您在步骤 2 的 UPDATE 语句中使用的同一列列表。

```
insert into target
(select col1, col2, 'expression'
from stage);

end transaction;
```

6. 删除暂存表。

```
drop table stage;
```

合并示例

以下示例将执行一个合并以更新 SALES 表。第一个示例使用了较简单的方法：从目标表中删除所有行，然后插入暂存表中的所有行。第二个示例需要更新目标表中的选定列，因此它包含一个额外的更新步骤。

合并示例使用名为 TICKIT 数据集的 Amazon Redshift 示例数据集。作为先决条件，您可以按照开始使用常见数据库任务指南中提供的说明设置 TICKIT 表和数据。有关示例数据集的更多详细信息可在示例数据库中找到。

合并数据来源示例

本节中的示例需要同时包含更新和插入的样本数据来源。例如，我们将创建一个名为 SALES_UPDATE 的示例表，该表使用 SALES 表中的数据。我们将使用表示 12 月的新销售活动的随机数据填充新表。我们将使用 SALES_UPDATE 示例表在后面的示例中创建暂存表。

```
-- Create a sample table as a copy of the SALES table.

create table tickit.sales_update as
select * from tickit.sales;

-- Change every fifth row to have updates.

update tickit.sales_update
set qtysold = qtysold*2,
pricepaid = pricepaid*0.8,
commission = commission*1.1
where saletime > '2008-11-30'
and mod(sellerid, 5) = 0;

-- Add some new rows to have inserts.
-- This example creates a duplicate of every fourth row.

insert into tickit.sales_update
select (salesid + 172456) as salesid, listid, sellerid, buyerid, eventid, dateid,
 qtysold, pricepaid, commission, getdate() as saletime
from tickit.sales_update
where saletime > '2008-11-30'
and mod(sellerid, 4) = 0;
```

基于匹配键替换现有行的合并示例

以下脚本使用 SALES_UPDATE 表对包含 12 月销售活动的新数据的 SALES 表执行合并操作。此示例替换 SALES 表中具有更新的行。对于此示例，我们将更新 qtysold 和 pricepaid 列，但让 commission 和 saletime 保持不变。

```
MERGE into tickit.sales
USING tickit.sales_update sales_update
on ( sales.salesid = sales_update.salesid
and sales.listid = sales_update.listid
and sales_update.saletime > '2008-11-30'
and (sales.qtysold != sales_update.qtysold
or sales.pricepaid != sales_update.pricepaid))
WHEN MATCHED THEN
```

```
update SET qtysold = sales_update.qtysold,
pricepaid = sales_update.pricepaid
WHEN NOT MATCHED THEN
INSERT (salesid, listid, sellerid, buyerid, eventid, dateid, qtysold , pricepaid,
 commission, saletime)
values (sales_update.salesid, sales_update.listid, sales_update.sellerid,
 sales_update.buyerid, sales_update.eventid,
sales_update.dateid, sales_update.qtysold , sales_update.pricepaid,
 sales_update.commission, sales_update.saletime);

-- Drop the staging table.
drop table tickit.sales_update;

-- Test to see that commission and salestime were not impacted.
SELECT sales.salesid, sales.commission, sales.salestime, sales_update.commission,
 sales_update.salestime
FROM tickit.sales
INNER JOIN tickit.sales_update sales_update
ON
sales.salesid = sales_update.salesid
AND sales.listid = sales_update.listid
AND sales_update.saletime > '2008-11-30'
AND (sales.commission != sales_update.commission
OR sales.salestime != sales_update.salestime);
```

在不使用 MERGE 的情况下指定列列表的合并示例

以下示例将执行合并操作以使用 12 月销售活动的新数据更新 SALES。我们需要同时包含更新和插入的样本数据，以及未更改的行。在此示例中，我们希望更新 QTYSOLD 和 PRICEPAID 列，但让 COMMISSION 和 SALETIME 保持不变。以下脚本使用 SALES_UPDATE 表来对 SALES 表执行合并操作。

```
-- Create a staging table and populate it with rows from SALES_UPDATE for Dec
create temp table stagesales as select * from sales_update
where saletime > '2008-11-30';

-- Start a new transaction
begin transaction;

-- Update the target table using an inner join with the staging table
-- The join includes a redundant predicate to collocate on the distribution key -- A
 filter on saletime enables a range-restricted scan on SALES
```

```
update sales
set qtysold = stagesales.qtysold,
pricepaid = stagesales.pricepaid
from stagesales
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid
and stagesales.saletime > '2008-11-30'
and (sales.qtysold != stagesales.qtysold
or sales.pricepaid != stagesales.pricepaid);

-- Delete matching rows from the staging table
-- using an inner join with the target table

delete from stagesales
using sales
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid;

-- Insert the remaining rows from the staging table into the target table
insert into sales
select * from stagesales;

-- End transaction and commit
end transaction;

-- Drop the staging table
drop table stagesales;
```

执行深层复制

深层复制将使用批量插入重新创建和重新填充表，这将自动对表进行排序。如果表具有大型未排序区域，则深层复制比 vacuum 快得多。如果您可以跟踪并发更新，我们建议您仅在深层复制操作期间进行并发更新。该过程完成后，将增量更新移到新表中。VACUUM 操作支持自动并发更新。

您可以选择四种方法之一来创建原始表的副本：

- 使用原始表 DDL。

  如果 CREATE TABLE DDL 可用，那么这是最快且首选的方法。在创建新表时，您可以指定所有表和列属性，包括主键和外键。您可以使用 SHOW TABLE 函数找到原始 DDL。
- 使用 CREATE TABLE LIKE。

如果原始 DDL 不可用，您可以使用 CREATE TABLE LIKE 来重新创建原始表。新表继承父表的编码、分配键、排序键和 not-null 属性。新表不继承父表的主键和外键属性，但您可以使用 ALTER TABLE 来添加它们。

- 创建一个临时表并截断原始表。

  如果您必须保留父表的主键和外键属性。如果父表有依赖关系，您可以使用 CREATE TABLE ... AS (CTAS) 以创建临时表。然后，截断原始表并从临时表填充它。

  与使用永久表相比，使用临时表可极大地提高性能，但存在丢失数据的风险。在创建临时表的会话结束时将自动删除该临时表。TRUNCATE 将立即提交，即使它在事务块中。如果 TRUNCATE 成功，但会话在接下来的 INSERT 完成前关闭，则数据将丢失。如果数据丢失是不可接受的，请使用永久表。

创建表的副本后，您可能必须授予对新表的访问权限。您可以使用 GRANT 来定义访问权限。要查看和授予表的所有访问权限，您必须是以下人员之一：

- 超级用户。
- 您想复制的表的拥有者。
- 拥有 ACCESS SYSTEM TABLE 权限以查看表的权限且具有所有相关权限的授予权限的用户。

此外，您可能必须授予对于深层复制所在模式的使用权限。如果您的深层复制的模式与原始表的模式不同，也不是 public 模式，则授予使用权限是必需的。要查看和授予使用权限，您必须是以下人员之一：

- 超级用户。
- 可以授予对深层复制模式的 USAGE 权限的用户。

使用原始表 DDL 执行深层复制

1. （可选）通过运行名为 v_generate_tbl_ddl 的脚本来重新创建表 DDL。
2. 使用原始 CREATE TABLE DDL 创建表的副本。
3. 使用 INSERT INTO … SELECT 语句向副本填充原始表中的数据。
4. 检查所授予的对于旧表的权限。您可以在 SVV_RELATION_PRIVILEGES 系统视图中查看这些权限。
5. 如有必要，将旧表的权限授予新表。

6. 向在原始表中具有权限的每个组和用户授予使用权限。如果您的深层复制表处于 public 模式，或者与原始表处于同一模式，则无需执行此步骤。
7. 删除原始表。
8. 使用 ALTER TABLE 语句将副本重命名为原始表名称。

以下示例使用名为 sample_copy 的 SAMPLE 的副本对 SAMPLE 表执行深层复制。

```
--Create a copy of the original table in the sample_namespace namespace using the
 original CREATE TABLE DDL.
create table sample_namespace.sample_copy ( … );

--Populate the copy with data from the original table in the public namespace.
insert into sample_namespace.sample_copy (select * from public.sample);

--Check SVV_RELATION_PRIVILEGES for the original table's privileges.
select * from svv_relation_privileges where namespace_name = 'public' and relation_name
 = 'sample' order by identity_type, identity_id, privilege_type;

--Grant the original table's privileges to the copy table.
grant DELETE on table sample_namespace.sample_copy to group group1;
grant INSERT, UPDATE on table sample_namespace.sample_copy to group group2;
grant SELECT on table sample_namespace.sample_copy to user1;
grant INSERT, SELECT, UPDATE on table sample_namespace.sample_copy to user2;

--Grant usage permission to every group and user that has privileges in the original
 table.
grant USAGE on schema sample_namespace to group group1, group group2, user1, user2;

--Drop the original table.
drop table public.sample;

--Rename the copy table to match the original table's name.
alter table sample_namespace.sample_copy rename to sample;
```

使用 CREATE TABLE LIKE 执行深层复制

1. 使用 CREATE TABLE LIKE 创建新表。
2. 使用 INSERT INTO … SELECT 语句将当前表中的行复制到新表。
3. 检查所授予的对于旧表的权限。您可以在 SVV_RELATION_PRIVILEGES 系统视图中查看这些权限。

4. 如有必要，将旧表的权限授予新表。
5. 向在原始表中具有权限的每个组和用户授予使用权限。如果您的深层复制表处于 public 模式，或者与原始表处于同一模式，则无需执行此步骤。
6. 删除当前表。
7. 使用 ALTER TABLE 语句将新表重命名为原始表名称。

以下示例使用 CREATE TABLE LIKE 对 SAMPLE 表执行深层复制。

```
--Create a copy of the original table in the sample_namespace namespace using CREATE
 TABLE LIKE.
create table sameple_namespace.sample_copy (like public.sample);

--Populate the copy with data from the original table.
insert into sample_namespace.sample_copy (select * from public.sample);

--Check SVV_RELATION_PRIVILEGES for the original table's privileges.
select * from svv_relation_privileges where namespace_name = 'public' and relation_name
 = 'sample' order by identity_type, identity_id, privilege_type;

--Grant the original table's privileges to the copy table.
grant DELETE on table sample_namespace.sample_copy to group group1;
grant INSERT, UPDATE on table sample_namespace.sample_copy to group group2;
grant SELECT on table sample_namespace.sample_copy to user1;
grant INSERT, SELECT, UPDATE on table sample_namespace.sample_copy to user2;

--Grant usage permission to every group and user that has privileges in the original
 table.
grant USAGE on schema sample_namespace to group group1, group group2, user1, user2;

--Drop the original table.
drop table public.sample;

--Rename the copy table to match the original table's name.
alter table sample_namespace.sample_copy rename to sample;
```

通过创建临时表并截断原始表来执行深层复制

1. 使用 CREATE TABLE AS 创建具有原始表中的行的临时表。
2. 截断当前表。
3. 使用 INSERT INTO … SELECT 语句将临时表中的行复制到原始表。

4. 删除临时表。

以下示例通过创建临时表并截断原始表来对 SALES 表执行深层复制。由于原始表仍然存在，因此您无需授予对副本表的权限。

```
--Create a temp table copy using CREATE TABLE AS.
create temp table salestemp as select * from sales;

--Truncate the original table.
truncate sales;

--Copy the rows from the temporary table to the original table.
insert into sales (select * from salestemp);

--Drop the temporary table.
drop table salestemp;
```

分析表

ANALYZE 操作更新查询计划程序用来选择最佳计划的统计元数据。

在大多数情况下，您无需显式运行 ANALYZE 命令。Amazon Redshift 监控您工作负载的更改，并在后台自动更新统计数据。此外，COPY 命令会在将数据加载到空表时自动执行分析。

要明确分析表或整个数据库，请运行 ANALYZE 命令。

主题

- 自动分析
- 分析新表数据
- ANALYZE 命令历史记录

自动分析

Amazon Redshift 持续监控您的数据库，并自动在后台执行分析操作。为了最大限度地降低对系统性能的影响，自动分析将在工作负载较轻的时段运行。

默认情况下会启用自动分析。要关闭自动分析，请通过修改集群的参数组来将 `auto_analyze` 参数设置为 **`false`**。

为了减少处理时间并提高整体系统性能，Amazon Redshift 将跳过对任何修改程度较小的表的自动分析。

分析操作将跳过具有最新统计数据的表。如果您将 ANALYZE 作为提取、转换和加载 (ETL) 工作流的一部分运行，则自动分析将跳过具有最新统计数据的表。类似地，在自动分析更新表的统计数据后，显式 ANALYZE 将跳过表。

分析新表数据

默认情况下，COPY 命令会在将数据加载到空表后执行 ANALYZE。无论表是否为空，您都可以通过设置 STATUPDATE ON 来强制执行 ANALYZE。如果您指定 STATUPDATE OFF，则不会执行 ANALYZE。仅表所有者或超级用户才可以运行 ANALYZE 命令，或在 STATUPDATE 设置为 ON 时运行 COPY 命令。

Amazon Redshift 还分析您使用以下命令创建的新表：

- CREATE TABLE AS (CTAS)
- CREATE TEMP TABLE AS
- SELECT INTO

当您对最初加载其数据后未分析的新表运行查询时，Amazon Redshift 将返回一条警告消息。在后续更新或加载后查询表时，不会出现警告。在您对引用未经分析的表的查询运行 EXPLAIN 命令时，将返回相同的警告消息。

在通过将数据添加到非空表来明显更改表的大小时，您可以明确更新统计数据。可以通过运行 ANALYZE 命令或将 STATUPDATE ON 选项用于 COPY 命令来做到这一点。要查看有关自上次执行 ANALYZE 以来插入或删除的行数的详细信息，请查询 PG_STATISTIC_INDICATOR 系统目录表。

您可以将 ANALYZE 命令的范围指定为下列选项之一：

- 整个当前数据库
- 单个表
- 单个表中的一个或多个特定列
- 有可能在查询中用作谓词的列

ANALYZE 命令将从表中获取行的采样，执行一些计算，并保存生成的列统计数据。预设情况下，Amazon Redshift 将为 DISTKEY 列运行一个采样过程，并为表中所有其他列运行另一个采样过

程。如果您希望为一部分列生成统计数据，则可指定一个逗号分隔的列列表。您可以将 ANALYZE 与 PREDICATE COLUMNS 子句一起运行来跳过用作谓词的列。

ANALYZE 操作是资源密集型的，因此仅对实际需要统计数据更新的表和列运行此类操作。您无需定期或按相同的计划分析所有表中的所有列。如果数据发生重大更改，请分析在以下操作中常用的列：

- 排序和分组操作
- 联接
- 查询谓词

为了减少处理时间并提高整体系统性能，对于具有较低的更改行数百分比（由 analyze_threshold_percent 参数决定）的任何表，Amazon Redshift 将跳过 ANALYZE。默认情况下，分析阈值将设置为 10%。可以通过运行 SET 命令来更改当前会话的分析阈值。

不太可能需要频繁分析的列是表示从未被实际查询的事实和度量以及任何相关属性的列（例如，大量 VARCHAR 列）。例如，请考虑 TICKIT 数据库中的 LISTING 表。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'listing';

column         |        type        | encoding | distkey | sortkey
---------------+--------------------+----------+---------+---------
listid         | integer            | none     | t       | 1
sellerid       | integer            | none     | f       | 0
eventid        | integer            | mostly16 | f       | 0
dateid         | smallint           | none     | f       | 0
numtickets     | smallint           | mostly8  | f       | 0
priceperticket | numeric(8,2)       | bytedict | f       | 0
totalprice     | numeric(8,2)       | mostly32 | f       | 0
listtime       | timestamp with...  | none     | f       | 0
```

如果此表每天加载了大量新记录，则必须定期分析查询中频繁用作联接键的 LISTID 列。如果 TOTALPRICE 和 LISTTIME 是查询中的常用约束，则可在每个工作日分析这些列和分配键。

```
analyze listing(listid, totalprice, listtime);
```

假设应用程序中的卖家和事件变动小得多，且日期 ID 引用仅涵盖两年或三年的一组固定天数。在此情况下，这些列的唯一值将不会发生明显更改。但是，每个唯一值的实例数将平稳增加。

此外，请考虑查询 NUMTICKETS 和 PRICEPERTICKET 度量的频率低于查询 TOTALPRICE 列的频率这种情况。在这种情况下，您可在每个周末对整个表运行一次 ANALYZE 命令，以便更新未每日分析的 5 个列的统计数据：

谓词列

作为指定列列表的便利替代方法，您可以选择仅分析可能用作谓词的列。当您运行查询时，在联接、筛选条件或 group by 子句中使用的任意列将在系统目录中标记为谓词列。当您使用 PREDICATE COLUMNS 子句运行 ANALYZE 时，分析操作仅包括满足以下标准的列：

- 标记为谓词列的列。
- 该列为分配键。
- 该列为排序键的一部分。

如果未将任何表的列标记为谓词，则即使指定了 PREDICATE COLUMNS，ANALYZE 仍将包括所有列。如果未将任何列标记为谓词列，这可能是因为尚未查询表。

在工作负载的查询模式相对稳定时，您可以选择使用 PREDICATE COLUMNS。当查询模式可变并且不同的列频繁用作谓词时，使用 PREDICATE COLUMNS 可能会临时得到过时的统计数据。过时的统计数据可能导致查询运行时间计划不够理想和运行时间较长。不过，当您下次使用 PREDICATE COLUMNS 运行 ANALYZE 时，将包括新的谓词列。

要查看谓词列的详细信息，请使用以下 SQL 创建名为 PREDICATE_COLUMNS 的视图。

```
CREATE VIEW predicate_columns AS
WITH predicate_column_info as (
SELECT ns.nspname AS schema_name, c.relname AS table_name, a.attnum as col_num,
 a.attname as col_name,
        CASE
            WHEN 10002 = s.stakind1 THEN array_to_string(stavalues1, '||')
            WHEN 10002 = s.stakind2 THEN array_to_string(stavalues2, '||')
            WHEN 10002 = s.stakind3 THEN array_to_string(stavalues3, '||')
            WHEN 10002 = s.stakind4 THEN array_to_string(stavalues4, '||')
            ELSE NULL::varchar
        END AS pred_ts
   FROM pg_statistic s
   JOIN pg_class c ON c.oid = s.starelid
   JOIN pg_namespace ns ON c.relnamespace = ns.oid
   JOIN pg_attribute a ON c.oid = a.attrelid AND a.attnum = s.staattnum)
SELECT schema_name, table_name, col_num, col_name,
       pred_ts NOT LIKE '2000-01-01%' AS is_predicate,
```

```
      CASE WHEN pred_ts NOT LIKE '2000-01-01%' THEN (split_part(pred_ts,
 '||',1))::timestamp ELSE NULL::timestamp END as first_predicate_use,
      CASE WHEN pred_ts NOT LIKE '%||2000-01-01%' THEN (split_part(pred_ts,
 '||',2))::timestamp ELSE NULL::timestamp END as last_analyze
FROM predicate_column_info;
```

假设您对 LISTING 表运行以下查询。请注意，LISTID、LISTTIME 和 EVENTID 均用于联接、筛选和 group by 子句中。

```
select s.buyerid,l.eventid, sum(l.totalprice)
from listing l
join sales s on l.listid = s.listid
where l.listtime > '2008-12-01'
group by l.eventid, s.buyerid;
```

当您查询 PREDICATE_COLUMNS 视图时，如下例中所示，可以看到 LISTID、EVENTID 和 LISTTIME 标记为谓词列。

```
select * from predicate_columns
where table_name = 'listing';
```

```
schema_name | table_name | col_num | col_name       | is_predicate |
 first_predicate_use | last_analyze
------------+------------+---------+----------------+--------------
+---------------------+---------------------
public      | listing    |       1 | listid         | true         | 2017-05-05
 19:27:59 | 2017-05-03 18:27:41
public      | listing    |       2 | sellerid       | false        |
  | 2017-05-03 18:27:41
public      | listing    |       3 | eventid        | true         | 2017-05-16
 20:54:32 | 2017-05-03 18:27:41
public      | listing    |       4 | dateid         | false        |
  | 2017-05-03 18:27:41
public      | listing    |       5 | numtickets     | false        |
  | 2017-05-03 18:27:41
public      | listing    |       6 | priceperticket | false        |
  | 2017-05-03 18:27:41
public      | listing    |       7 | totalprice     | false        |
  | 2017-05-03 18:27:41
public      | listing    |       8 | listtime       | true         | 2017-05-16
 20:54:32 | 2017-05-03 18:27:41
```

通过允许查询计划程序选择最佳计划，使统计数据保持最新可提高查询性能。Amazon Redshift 在后台自动刷新统计数据，您也可以明确运行 ANALYZE 命令。如果您选择明确运行 ANALYZE，请执行以下操作：

- 在运行查询之前运行 ANALYZE 命令。
- 在每次定期加载或更新循环结束时，常规性地对数据库运行 ANALYZE 命令。
- 对您创建的任何新表和正在进行重大更改的任何现有表或列运行 ANALYZE 命令。
- 考虑按不同计划对不同类型的表和列运行 ANALYZE 操作，具体取决于它们在查询中的使用和它们的更改倾向。
- 为节省时间和集群资源，在运行 ANALYZE 时请使用 PREDICATE COLUMNS 子句。

您不必在将快照还原到预置集群或无服务器命名空间之后显式运行 ANALYZE 命令，也不必在恢复已暂停的预置集群之后显式运行此命令。在这些情况下，Amazon Redshift 会保留系统表信息，从而无需手动执行 ANALYZE 命令。Amazon Redshift 将继续根据需要运行自动分析操作。

分析操作将跳过具有最新统计数据的表。如果您将 ANALYZE 作为提取、转换和加载 (ETL) 工作流的一部分运行，则自动分析将跳过具有最新统计数据的表。类似地，在自动分析更新表的统计数据后，显式 ANALYZE 将跳过表。

ANALYZE 命令历史记录

了解上次对表或数据库运行 ANALYZE 命令的时间很有用。运行 ANALYZE 命令时，Amazon Redshift 将运行与以下内容类似的多个查询：

```
padb_fetch_sample: select * from table_name
```

查询 STL_ANALYZE 以查看分析操作的历史记录。如果 Amazon Redshift 使用自动分析功能来分析表，则 `is_background` 列将设置为 `t`（真）。否则，它将设置为 `f`（假）。以下示例联接 STV_TBL_PERM 以显示表名称和运行时间详细信息。

```
select distinct a.xid, trim(t.name) as name, a.status, a.rows, a.modified_rows,
 a.starttime, a.endtime
from stl_analyze a
join stv_tbl_perm t  on t.id=a.table_id
where name = 'users'
order by starttime;
```

```
xid    | name  | status          | rows  | modified_rows | starttime           |
 endtime
-------+-------+-----------------+-------+---------------+---------------------
+--------------------
  1582 | users | Full            | 49990 |         49990 | 2016-09-22 22:02:23 |
 2016-09-22 22:02:28
244287 | users | Full            | 24992 |         74988 | 2016-10-04 22:50:58 |
 2016-10-04 22:51:01
244712 | users | Full            | 49984 |         24992 | 2016-10-04 22:56:07 |
 2016-10-04 22:56:07
245071 | users | Skipped         | 49984 |             0 | 2016-10-04 22:58:17 |
 2016-10-04 22:58:17
245439 | users | Skipped         | 49984 |          1982 | 2016-10-04 23:00:13 |
 2016-10-04 23:00:13
(5 rows)
```

或者，您可以运行一个更复杂的查询，该查询将返回在每个包括 ANALYZE 命令的已完成事务中运行的所有语句：

```
select xid, to_char(starttime, 'HH24:MM:SS.MS') as starttime,
datediff(sec,starttime,endtime ) as secs, substring(text, 1, 40)
from svl_statementtext
where sequence = 0
and xid in (select xid from svl_statementtext s where s.text like 'padb_fetch_sample
%' )
order by xid desc, starttime;

xid  |  starttime   | secs |                 substring
-----+--------------+------+-------------------------------------------
1338 | 12:04:28.511 |    4 | Analyze date
1338 | 12:04:28.511 |    1 | padb_fetch_sample: select count(*) from
1338 | 12:04:29.443 |    2 | padb_fetch_sample: select * from date
1338 | 12:04:31.456 |    1 | padb_fetch_sample: select * from date
1337 | 12:04:24.388 |    1 | padb_fetch_sample: select count(*) from
1337 | 12:04:24.388 |    4 | Analyze sales
1337 | 12:04:25.322 |    2 | padb_fetch_sample: select * from sales
1337 | 12:04:27.363 |    1 | padb_fetch_sample: select * from sales
...
```

对表执行 vacuum 操作

Amazon Redshift 可以在后台自动对表进行排序并执行 VACUUM DELETE 操作。要在加载或一系列增量更新操作后清理表，您也可以对整个数据库或对单个表运行 VACUUM 命令。

> **Note**
>
> 只有拥有必要的表权限的用户才能有效地对表执行 vacuum 操作。如果在没有必需的表权限的情况下运行 VACUUM 操作，该操作将成功完成，但不起任何作用。有关能有效运行 VACUUM 操作的有效表权限列表，请参阅VACUUM。
> 出于此原因，我们建议根据需要对各个表执行 vacuum 操作。我们也推荐此方法，因为对整个数据库进行 vacuum 操作可能会消耗大量资源。

自动表排序

Amazon Redshift 在后台自动对数据进行排序以按照其排序键顺序保留表数据。Amazon Redshift 将跟踪您的扫描查询以确定表的哪些部分将从排序中受益。

根据系统上的负载，Amazon Redshift 自动启动排序操作。此自动排序减少了运行 VACUUM 命令以按排序键顺序保留数据的需求。如果您需要按排序键顺序对数据进行完全排序（例如，在加载大量数据之后），则您仍可以手动运行 VACUUM 命令。要通过运行 VACUUM SORT 来确定您的表是否将受益，请监控 `vacuum_sort_benefit` 中的 SVV_TABLE_INFO 列。

Amazon Redshift 跟踪在每个表上使用排序键的扫描查询。Amazon Redshift 估计每个表（如果表已完全排序）在数据扫描和筛选方面的最大改进百分比。此估计值在 `vacuum_sort_benefit` 中的 SVV_TABLE_INFO 列中可见。您可以将此列与 `unsorted` 列结合使用，确定查询何时可以从手动对表运行 VACUUM SORT 中受益。`unsorted` 列反映表的物理排序顺序。`vacuum_sort_benefit` 列指定通过手动运行 VACUUM SORT 对表进行排序的影响。

例如，请考虑以下查询：

```
select "table", unsorted,vacuum_sort_benefit from svv_table_info order by 1;
```

```
 table | unsorted | vacuum_sort_benefit
-------+----------+---------------------
 sales |    85.71 |                5.00
 event |    45.24 |               67.00
```

对于表“sales”，即使该表的物理未排序项约为 86%，其对查询性能的影响也仅为 5%。这可能是因为查询只访问表的一小部分内容，也可能是因为访问表的查询几近于无。对于表“event”，该表的物理未排序项约为 45%。不过，67% 的查询性能影响表明查询访问了表的更大部分内容，或者访问表的查询的数量很多。表“event”可能会从运行 VACUUM SORT 中受益。

自动 vacuum 删除

执行 delete 操作时，会将行标记为删除，但不会删除。Amazon Redshift 会根据数据库表中已删除的行数在后台自动运行 VACUUM DELETE 操作。Amazon Redshift 安排 VACUUM DELETE 在负载减少期间运行，并在高负载期间暂停操作。

主题

vacuum 频率

您应按照所需的频率执行 vacuum 操作以保持一致的查询性能。在确定运行 VACUUM 命令的频率时，请考虑以下因素：

- 在预计集群上的活动最少的时间段（例如，夜晚或指定的数据库管理时段）内运行 VACUUM。
- 在维护时段之外运行 VACUUM 命令。有关更多信息，请参阅计划维护时段。
- 大型未排序区域将导致更长的 vacuum 时间。如果您延迟 vacuum 操作，则 vacuum 操作将需要更长的时间，因为需要识别更多数据。
- VACUUM 是 I/O 密集型操作，因此，完成 vacuum 操作所需的时间越长，它对您的集群上运行的并发查询和其他数据库操作的影响就越大。
- 对于使用交错排序的表，VACUUM 花费的时间更长。要评估是否必须对交错的表重新排序，请查询 SVV_INTERLEAVED_COLUMNS 视图。

排序阶段和合并阶段

Amazon Redshift 按照两个阶段执行 vacuum 操作：首先，它会对未排序区域中的行进行排序；然后，如有必要，会将表结尾处的新排序的行与现有行合并。在对大型表执行 vacuum 操作时，vacuum 操作

将在合并后继续一系列步骤（包括增量排序）。如果操作失败，或者如何 Amazon Redshift 在 vacuum 操作期间脱机，已部分执行 vacuum 操作的表或数据库将处于一致状态，但您必须手动重启 vacuum 操作。增量排序将丢失，但不需要再次对失败前已提交的合并行执行 vacuum 操作。如果未排序区域较大，则浪费的时间可能更多。有关排序阶段和合并阶段的更多信息，请参阅管理合并的行数。

在对表执行 vacuum 操作时，用户可以访问表。您可以在对表执行 vacuum 操作的同时执行查询和写入操作，但如果 DML 和 vacuum 操作同时运行，则二者可能花费更长时间。如果您在 vacuum 操作期间执行 UPDATE 和 DELETE 语句，则系统性能可能会降低。增量合并会临时阻止并发 UPDATE 和 DELETE 操作，而 UPDATE 和 DELETE 反过来会临时阻止对受影响的表执行增量合并步骤。在对表执行完 vacuum 操作前，DDL 操作（例如 ALTER TABLE）将被阻止。

> **Note**
>
> VACUUM 的各种修饰符控制它的工作方式。您可以使用它们来根据当前需求定制 vacuum 操作。例如，使用 VACUUM RECLUSTER 可通过不执行完全合并操作来缩短 vacuum 操作。有关更多信息，请参阅 VACUUM。

vacuum 阈值

默认情况下，当任意表中有 95% 的行已有序时，VACUUM 会为该表跳过排序阶段。跳过排序阶段能够显著提高 VACUUM 的性能。要更改某个表的默认排序阈值，请在运行 VACUUM 命令时包含表名称和 TO threshold PERCENT 参数。

vacuum 类型

有关不同 vacuum 类型的信息，请参阅VACUUM。

管理 vacuum 操作次数

根据您的数据的性质，建议您执行本部分中的做法以最大程度地减少 vacuum 次数。

主题

- 决定是否重建索引
- 管理未排序区域的大小
- 管理合并的行数
- 按排序键顺序加载数据

- 使用时间序列表

决定是否重建索引

通常，您可以使用交错排序样式来显著提高查询性能，但是随着时间的推移，如果排序键列中值的分配更改，性能可能会下降。

最初使用 COPY 或 CREATE TABLE AS 加载空交错表时，Amazon Redshift 自动构建交错索引。如果您最初使用 INSERT 加载交错表，则需要在之后运行 VACUUM REINDEX 以初始化交错索引。

随着时间的推移，在添加带有新排序键值的行后，如果排序键列中值的分布发生更改，性能可能会下降。如果新行主要处于现有排序键值的范围内，则不必重建索引。可以运行 VACUUM SORT ONLY 或 VACUUM FULL 恢复排序顺序。

查询引擎能够使用排序顺序高效地选择处理查询所需扫描的数据块。对于交错排序，Amazon Redshift 将分析排序键列值以确定最佳排序顺序。如果键值的分配在添加行时发生更改或偏移，则排序策略将不再是最佳的排序策略，并且排序的性能优势也会减小。要重新分析排序键分配，您可以运行 VACUUM REINDEX。重建索引操作非常耗时，因此，要决定表是否将从重建索引中获益，请查询 SVV_INTERLEAVED_COLUMNS 视图。

例如，以下查询将显示使用交错排序键的表的详细信息。

```
select tbl as tbl_id, stv_tbl_perm.name as table_name,
col, interleaved_skew, last_reindex
from svv_interleaved_columns, stv_tbl_perm
where svv_interleaved_columns.tbl = stv_tbl_perm.id
and interleaved_skew is not null;

 tbl_id | table_name | col | interleaved_skew | last_reindex
--------+------------+-----+------------------+---------------------
 100048 | customer   |   0 |             3.65 | 2015-04-22 22:05:45
 100068 | lineorder  |   1 |             2.65 | 2015-04-22 22:05:45
 100072 | part       |   0 |             1.65 | 2015-04-22 22:05:45
 100077 | supplier   |   1 |             1.00 | 2015-04-22 22:05:45
(4 rows)
```

`interleaved_skew` 的值是一个比率，指示偏移量。值 1 表示无偏移。如果偏移大于 1.4，VACUUM REINDEX 通常会提高性能，除非偏移是基础集中固有的。

您可以使用 `last_reindex` 中的数据值来确定自上次重建索引以来经历的时间。

管理未排序区域的大小

在将大量新数据加载到已包含数据的表中时，或在例行维护操作不包含对表进行的 vacuum 操作时，未排序区域将增大。要避免长时间运行的 vacuum 操作，请使用以下做法：

- 定期运行 vacuum 操作。

  如果您以较小增量加载您的表（例如，表示表中行总数的一小部分的日常更新），定期运行 VACUUM 将帮助确保各个 vacuum 操作快速执行。
- 首先运行最大加载。

  如果您需要使用多个 COPY 操作加载新表，请首先运行最大加载。当您运行到新的或截断的表中的初始加载时，所有数据将直接加载到已排序区域，因此无需执行 vacuum 操作。
- 截断表而不是删除所有行。

  从表中删除行不会回收行占用的空间，除非您执行 vacuum 操作；不过，截断表将清空表并回收磁盘空间，因此无需执行 vacuum 操作。或者，请删除表并重新创建它。
- 截断或删除测试表。

  如果您正在将少量行加载到表中以进行测试，请在完成此操作后不要删除这些行。相反，作为后续生产加载操作的一部分，请截断表并重新加载这些行。
- 执行深层复制。

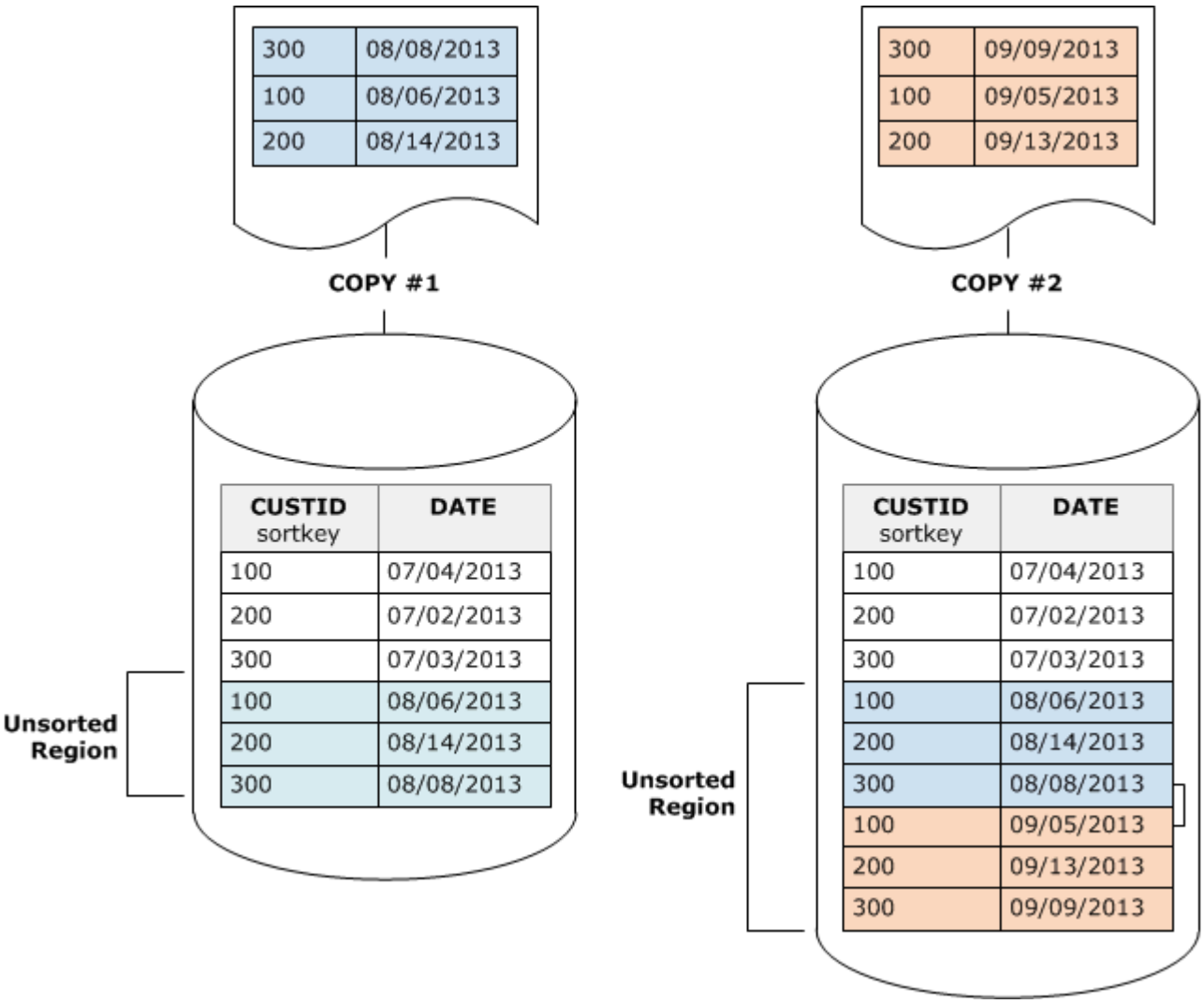
  如果使用复合排序键的表具有大型未排序区域，则深层复制要比 vacuum 快得多。深层复制将使用批量插入来重新创建并重新填充表，这将自动对表进行重新排序。如果表拥有大型未排序区域，深层复制将比真空化快得多。这样做的代价是，您不能在深层复制操作过程中进行并行更新，但可以在真空化时这样做。有关更多信息，请参阅 设计查询的 Amazon Redshift 最佳实践。

管理合并的行数

如果 vacuum 操作需要将新行合并到表的已排序区域，vacuum 所需的时间将随表的增大而增多。您可以通过减少必须合并的行数来提高 vacuum 性能。

在执行 vacuum 操作之前，表包含一个已排序区域（位于表开头处），后跟一个未排序区域（当添加或更新行时，该区域将增大）。如果 COPY 操作添加一组行，则这组新行在添加到表结尾处的未排序区域时将基于排序键进行排序。新行将在其自己的集合中进行排序，而不是在未排序区域内进行排序。

下图说明了两次连续 COPY 操作后的未排序区域，其中排序键为 CUSTID。为简便起见，此示例显示一个复合排序键，但相同的原则适用于交错排序键，只不过未排序区域对交错表的影响更大。



Vacuum 将按两个阶段还原复表的排序顺序：

1. 将未排序区域归入新排序的区域。

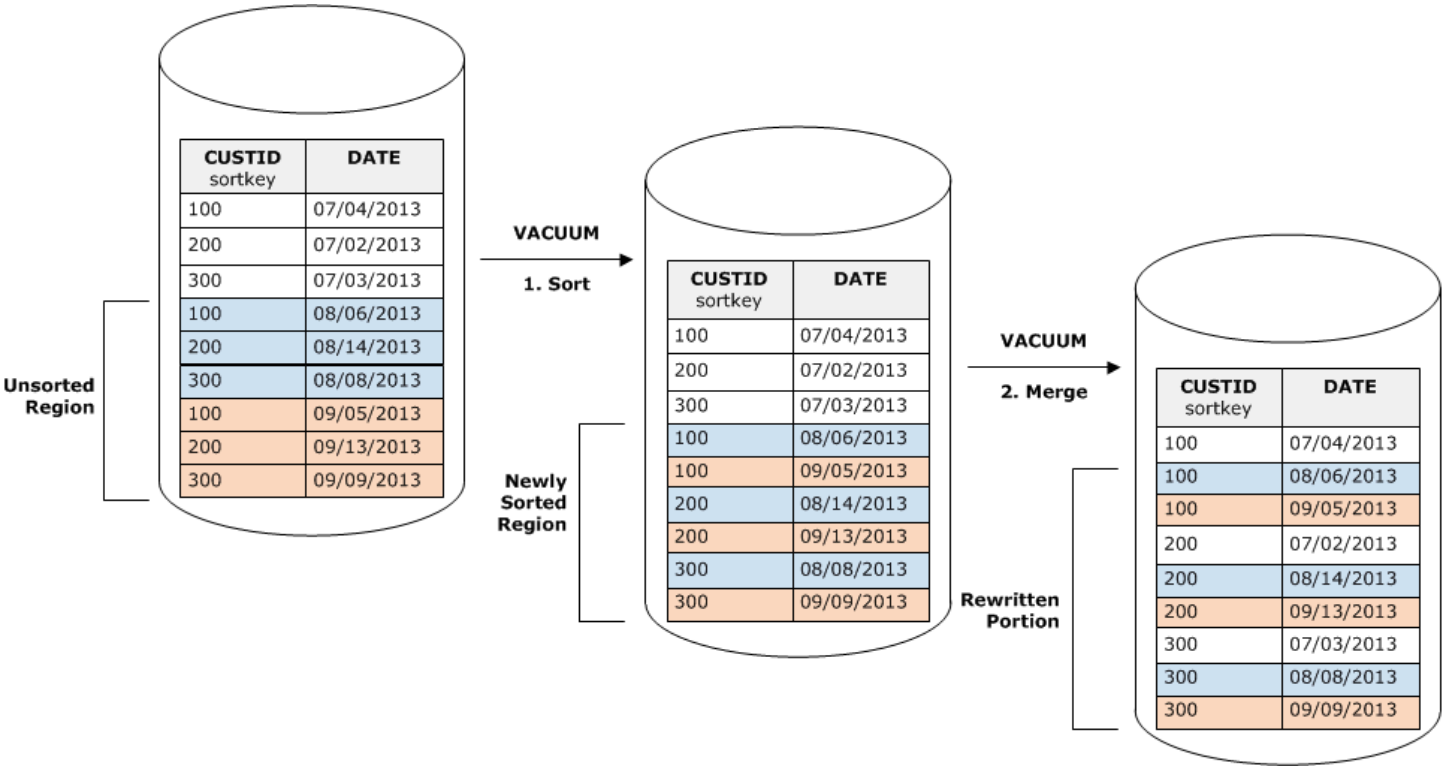
   第一个阶段的成本相对较低，因为仅重写未排序区域。如果新排序区域的排序键值的范围大于现有范围，则仅需要重写新行即可完成 vacuum。例如，如果已排序区域包含的 ID 值介于 1 和 500 之间，并且后续复制操作将添加大于 500 的键值，则仅需重写未排序区域。

2. 将新排序的区域与之前排序的区域合并。

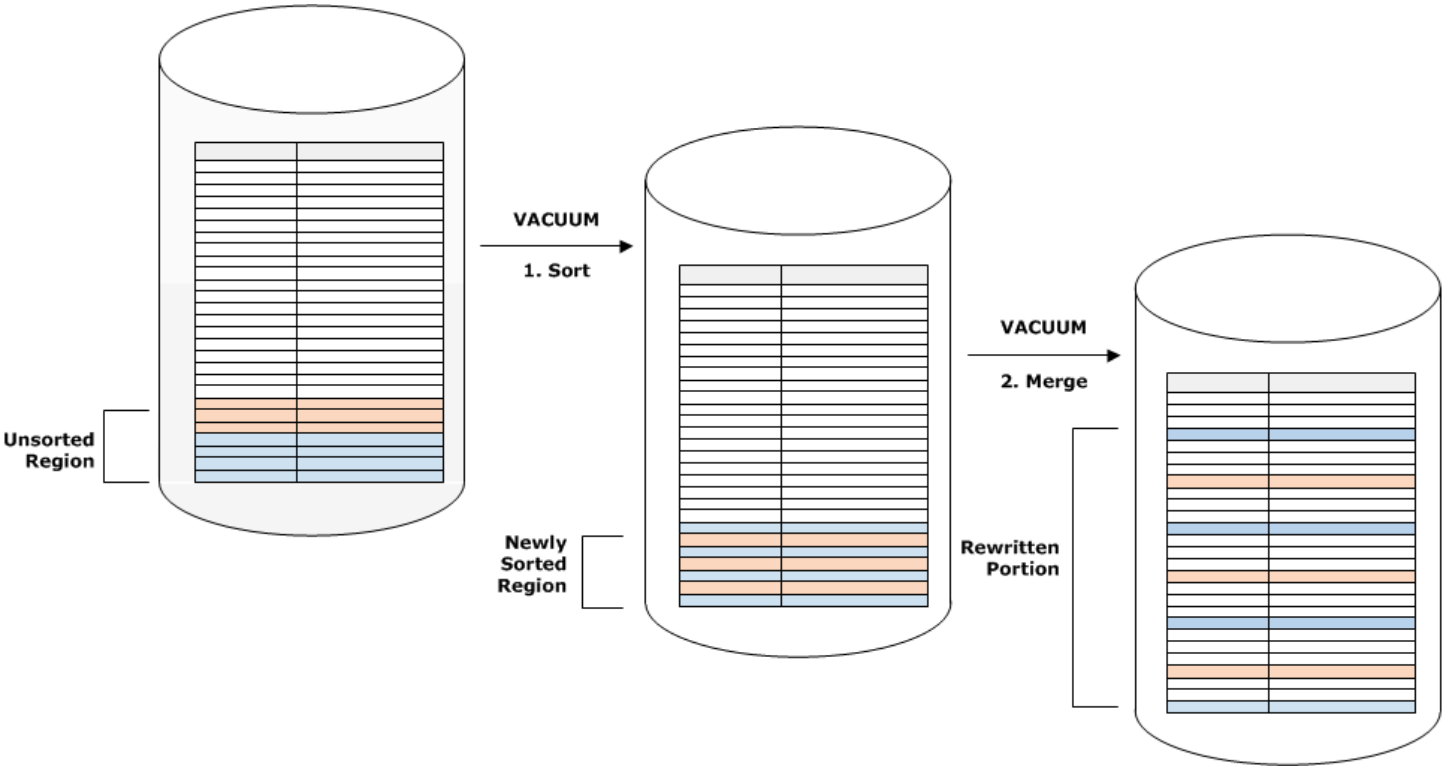
   如果新排序的区域中的键与已排序区域中的键重叠，则 VACUUM 需要合并这些行。从新排序区域（最低排序键处）的开头开始，vacuum 会将从之前排序的区域和新排序的区域中的合并行写入一组新数据块。

新排序键范围与现有排序键重叠的程度将决定之前排序的区域需要被重写的程度。如果未排序键遍布于现有排序范围中，则 vacuum 可能需要重写表的现有部分。

下图说明 vacuum 如何对添加到排序键为 CUSTID 的表中的行进行排序和合并。由于每个复制操作将添加一组键值与现有键重叠的新行，因此几乎需要重写整个表。该图显示单一排序和合并，但在实践中，大型 vacuum 包含一系列增量排序和合并步骤。



如果一组新行中的排序键范围与现有键的范围重叠，则合并阶段的成本将继续随表的增大与表大小成比例的增长，而排序阶段的成本与未排序区域的大小成正比。在这种情况下，合并阶段的成本远远超过排序阶段的成本，如下图所示。



要确定已重新合并的表的比例，请在 vacuum 操作完成后查询 SVV_VACUUM_SUMMARY。以下查询表明，在 CUSTSALES 随时间的推移而增大时，六个连续 vacuum 操作的效果。

```
select * from svv_vacuum_summary
where table_name = 'custsales';

 table_name | xid  | sort_      | merge_     | elapsed_   | row_  | sortedrow_ | block_
  | max_merge_
            |      | partitions | increments | time       | delta | delta      | delta
  | partitions
 -----------+------+------------+------------+------------+-------+------------
+---------+---------------
  custsales | 7072 |          3 |          2 |  143918314 |     0 |   88297472 |   1524
  |      47
  custsales | 7122 |          3 |          3 |  164157882 |     0 |   88297472 |    772
  |      47
  custsales | 7212 |          3 |          4 |  187433171 |     0 |   88297472 |    767
  |      47
  custsales | 7289 |          3 |          4 |  255482945 |     0 |   88297472 |    770
  |      47
  custsales | 7420 |          3 |          5 |  316583833 |     0 |   88297472 |    769
  |      47
```

```
 custsales | 9007 |          3 |          6 |  306685472 |     0 |   88297472 |    772
 |      47
(6 rows)
```

merge_increments 列指明了为每个 vacuum 操作合并的数据量。如果连续 vacuum 操作的合并增量的数量按表大小增长的比例增加，它表示每个 vacuum 操作重新合并的表中的行数正在增加，因为现有排序区域和新排序区域重叠。

按排序键顺序加载数据

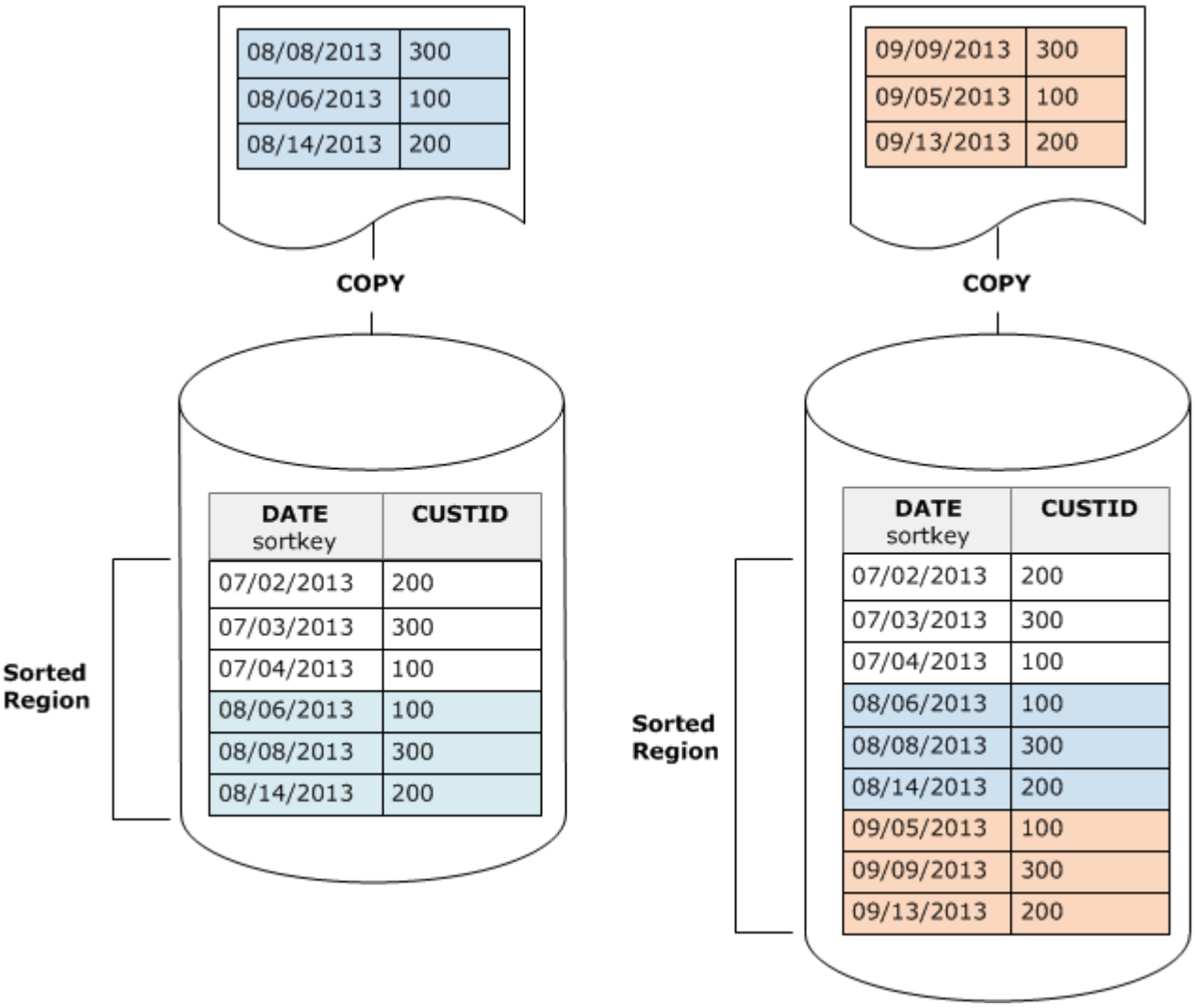
如果您使用 COPY 命令按排序键顺序加载数据，可能会减少甚至消除对 vacuum 的需求。

当满足以下所有条件时，COPY 会向表的有序区域自动添加新行：

- 表使用了只有一个排序列的复合排序键。
- 排序列 NOT NULL。
- 表 100% 有序或为空。
- 所有新行的排序顺序均优先于现有行，包括标记为要删除的行。在此实例中，Amazon Redshift 使用排序键的前八个字节来确定排序顺序。

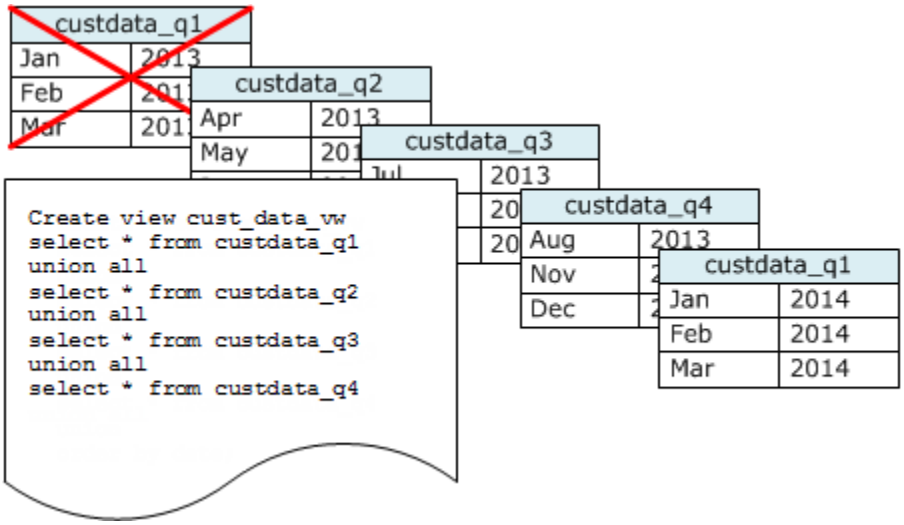
例如，假设您有一个使用客户 ID 和时间记录客户事件的表。如果按客户 ID 进行排序，则增量加载添加的新行的排序键范围可能会与现有范围重叠（如上一个示例中所示），从而导致昂贵的 vacuum 操作。

如果您将排序键设置为时间戳列，新行将按排序顺序追加到表的结尾（如下图所示），从而减少甚至消除对 vacuum 的需求。



使用时间序列表

如果您将数据保留滚动时段，请使用一系列表，如下图所示。



每当添加一组数据时创建一个新表，然后删除系列中最旧的表。您将获得双重好处：

- 避免增加删除行的成本，因为 DROP TABLE 操作比大规模 DELETE 更高效。
- 如果按时间戳对表进行排序，则不需要执行 vacuum 操作。如果每个表包含一个月的数据，则 vacuum 最多必须重写一个月的数据，即使表不是按时间戳排序的也是如此。

您可以创建 UNION ALL 视图供报告查询使用，从而隐藏数据存储在多个表中的事实。如果查询按排序键进行筛选，则查询计划程序可高效地跳过未使用的所有表。UNION ALL 对其他类型的查询可能不太高效，因此您应在所有使用表的查询的环境中评估查询性能。

管理并发写入操作

主题

- 可序列化的隔离
- 写入和读/写操作
- 并发写入示例

Amazon Redshift 允许在以递增方式加载或修改表时对表进行读取操作。

在一些传统的数据仓库和商业智能应用程序中，数据库仅在夜间加载完成后对用户可用。在这种情况下，在正常工作时间内，当运行分析查询和生成报告时不允许更新；但是，越来越多的应用程序可在一天的更长时间内甚至整天保持活动状态，从而导致加载窗口被淘汰。

Amazon Redshift 通过允许在以递增方式加载或修改表时对表进行读取操作，来支持这些类型的应用程序。查询仅查看数据的最新提交的版本或快照，而不是等待提交下一个版本。如果您希望特定查询等待来自另一个写入操作的提交，则必须相应地做出安排。

以下主题介绍一些主要概念和使用案例，它们涉及事务、数据库快照、更新和并发行为。

可序列化的隔离

某些应用程序不仅需要并发查询和加载，还需要能同时对多个表或同一个表进行写入。在此环境中，并发 指的是重叠，而不是安排在完全相同的时间运行。如果两个事务中的第二个事务在第一个提交前开始，则将两个事务视为并发。并发操作可源自由同一用户或不同用户控制的不同会话。

> **Note**
>
> Amazon Redshift 支持默认的自动提交行为，其中，每个单独运行的 SQL 命令都将分别提交。如果您在某个事务数据块中包含一组命令（由 BEGIN 和 END 语句定义），则该数据块将作为一个事务提交，以便您在必要时对其进行回滚。此行为的例外是 TRUNCATE 和 VACUUM 命令，这些命令可自动提交当前事务中所做的所有待定更改。
> 某些 SQL 客户端会自动发出 BEGIN 和 COMMIT 命令，因此客户端控制着是一组语句作为一个事务运行，还是每个单独的语句作为自己的事务运行。检查您正在使用的界面的文档。例如，使用 Amazon Redshift JDBC 驱动程序时，具有包含多个（分号分隔）SQL 命令的查询字符串的 JDBC `PreparedStatement` 将所有语句作为单个事务运行。相比之下，如果您使用 SQL Workbench/J 并设置 AUTO COMMIT ON，则如果您运行多个语句，每个语句都会作为自己的事务运行。

在 Amazon Redshift 中，以保护性方式支持并发写入操作，即对表使用写入锁定和可序列化的隔离原则。可序列化的隔离会保留一种错觉，即对某个表运行的事务是对该表运行的唯一事务。例如，两个同时运行的事务（T1 和 T2）必须生成与以下至少一个项相同的结果：

- T1 和 T2 依次运行。
- T2 和 T1 依次运行。

并发事务彼此不可见；它们不能相互检测对方的更改。每个并发事务将在事务开始时创建数据库快照。在大多数 SELECT 语句、DML 命令（例如 COPY、DELETE、INSERT、UPDATE 和 TRUNCATE）和以下 DDL 命令首次出现时，将在事务中创建数据库快照：

- ALTER TABLE（添加或删除列）

- CREATE TABLE
- DROP TABLE
- TRUNCATE TABLE

如果并发事务的任何序列执行产生与它们的并发执行相同的结果，则这些事务将被视为“可序列化的”且可安全运行。如果这些事务的任何序列执行均不产生相同结果，则执行会破坏可序列性的语句的事务将被中止并回滚。

系统目录表 (PG) 和其他 Amazon Redshift 系统表（STL 和 STV）在事务中未锁定。因此，DDL 和 TRUNCATE 操作引起的数据库对象更改在提交到任何并发事务时均可见。

例如，假设在两个并发事务（T1 和 T2）开始时，数据库中存在表 A。假设 T2 通过从 PG_TABLES 目录表中进行选择来返回表列表。然后 T1 删除表 A 并提交，T2 再次列出这些表。现在不再列出表 A。如果 T2 尝试查询已删除的表，则 Amazon Redshift 将返回“relation does not exist”错误。向 T2 返回表列表或检查表 A 是否存在的目录查询不受与针对用户表的操作相同的隔离规则的约束。

更新这些表的事务在读取已提交 隔离模式下运行。前缀为 PG 的目录表不支持快照隔离。

系统表和目录表的可序列化隔离

对于任何引用用户创建的表或 Amazon Redshift 系统表（STL 或 STV）的 SELECT 查询，还将在事务中创建数据库快照。不引用任何表的 SELECT 查询不会创建新的事务数据库快照。仅在系统目录表 (PG) 上操作的 INSERT、DELETE 和 UPDATE 语句也不会创建新的事务数据库快照。

如何修复可序列化的隔离错误

ERROR:1023 DETAIL：Redshift 中的表上的可序列化隔离冲突

当 Amazon Redshift 检测到可序列化的隔离错误时，您会看到错误消息，如下所示。

```
ERROR:1023 DETAIL: Serializable isolation violation on table in Redshift
```

要解决可序列化的隔离错误，您可以尝试以下方法：

- 重试已取消的事务。

  Amazon Redshift 检测到并发工作负载不可序列化。它建议使应用程序逻辑中存在差异，而这些差异通常可以通过重试遇到错误的事务来解决。如果问题仍然存在，请尝试使用其他方法之一。
- 将任何不必在同一个原子事务中的操作移到事务之外。

当两个事务内的各个操作以可能影响另一个事务的结果的方式相互交叉引用时，此方法适用。例如，以下两个会话各自启动一个事务。

```
Session1_Redshift=# begin;
```

```
Session2_Redshift=# begin;
```

每个事务中的 SELECT 语句的结果可能受另一个事务中的 INSERT 语句影响。换句话说，假设您以任何顺序连续运行以下语句。在每种情况下，如果不是同时运行事务，结果是 SELECT 语句之一另外返回一行。操作顺序运行时，没有哪种顺序可以产生与并发运行时相同的结果。因此，运行的最后一个操作会导致可序列化的隔离错误。

```
Session1_Redshift=# select * from tab1;
Session1_Redshift=# insert into tab2 values (1);
```

```
Session2_Redshift=# insert into tab1 values (1);
Session2_Redshift=# select * from tab2;
```

在许多情况下，SELECT 语句的结果并不重要。换句话说，事务中操作的原子性并不重要。在这些情况下，将 SELECT 语句移到事务之外，如以下示例所示。

```
Session1_Redshift=# begin;
Session1_Redshift=# insert into tab1 values (1)
Session1_Redshift=# end;
Session1_Redshift=# select * from tab2;
```

```
Session2_Redshift # select * from tab1;
Session2_Redshift=# begin;
Session2_Redshift=# insert into tab2 values (1)
Session2_Redshift=# end;
```

在这些示例中，事务中没有交叉引用。两个 INSERT 语句不会相互影响。在这些示例中，至少有一个顺序，其中事务可以顺序运行并产生与并发运行时相同的结果。这意味着事务是可序列化的。

- 通过锁定每个会话中的所有表来强制序列化。

  LOCK 命令阻止可能导致可序列化隔离错误的操作。使用 LOCK 命令时，请确保执行以下操作：

- 锁定受事务影响的所有表，包括受事务内部的只读 SELECT 语句影响的表。
- 无论执行操作的顺序如何，都以相同的顺序锁定表。
- 在执行任何操作之前，在事务开始时锁定所有表。

- 对并发事务使用快照隔离

  将 ALTER DATABASE 命令与快照隔离功能结合使用。有关 ALTER DATABASE 的 SNAPSHOT 参数的更多信息，请参阅参数。

ERROR:1018 DETAIL：关系不存在

当您在不同会话中运行 Amazon Redshift 并发操作时，您会看到错误消息，如下所示。

```
ERROR: 1018 DETAIL: Relation does not exist.
```

Amazon Redshift 中的事务遵循快照隔离。在事务开始后，Amazon Redshift 将拍摄数据库的快照。对于事务的整个生命周期，事务在快照中反映的数据库状态下运行。如果事务从快照中不存在的表读取，则会引发之前显示的 1018 错误消息。即使另一个并发事务在事务拍摄快照后创建表，该事务也无法从新创建的表中读取。

要解决此序列化隔离错误，您可以尝试将事务的起始位置移动到您所知道的该表存在的位置。

如果表是由另一个事务创建的，则此位置至少在该事务提交之后。此外，请确保没有提交可能已删除表的并发事务。

```
session1 = # BEGIN;
session1 = # DROP TABLE A;
session1 = # COMMIT;
```

```
session2 = # BEGIN;
```

```
session3 = # BEGIN;
session3 = # CREATE TABLE A (id INT);
session3 = # COMMIT;
```

```
session2 = # SELECT * FROM A;
```

作为 session2 的读取操作运行的最后一个操作会导致可序列化的隔离错误。当 session2 拍摄快照并且表已被提交的 session1 删除时，会发生此错误。换句话说，即使并发 session3 已创建表，session2 也不会看到该表，因为它不在快照中。

要解决此错误，您可以按以下方式对会话重新排序。

```
session1 = # BEGIN;
session1 = # DROP TABLE A;
session1 = # COMMIT;
```

```
session3 = # BEGIN;
session3 = # CREATE TABLE A (id INT);
session3 = # COMMIT;
```

```
session2 = # BEGIN;
session2 = # SELECT * FROM A;
```

现在，当 session2 拍摄快照时，session3 已经提交，并且表位于数据库中。Session2 可以从表中读取，而不会出现任何错误。

写入和读/写操作

可通过决定何时以及如何运行不同类型的命令来管理并发写入操作的特定行为。以下命令与此讨论相关：

- COPY 命令，可执行加载（初始或增量）
- INSERT 命令，可一次性追加一个或多个行
- UPDATE 命令，可修改现有行
- DELETE 命令，可删除行

COPY 和 INSERT 操作是纯写入操作，而 DELETE 和 UPDATE 操作是读/写操作。（要删除或更新行，必须先读取行。）并发写入操作的结果取决于同时运行的具体命令。针对同一个表的 COPY 和 INSERT 操作将保持等待状态，直到解除锁定，然后它们将正常继续。

UPDATE 和 DELETE 操作的行为方式不同，因为它们在执行任何写入操作前依赖最初的表读取。由于并发事务彼此不可见，因此，UPDATE 和 DELETE 必须读取上次提交的数据的快照。当第一个 UPDATE 或 DELETE 解除其锁定时，第二个 UPDATE 或 DELETE 需要确定它将使用的数据是否可能已过时。它不会过时，因为第二个事务不会在第一个事务解除其锁定之前获取其数据的快照。

并发写入事务的潜在死锁情况

只要事务涉及多个表的更新，同时运行的事务就始终可能在同时尝试写入同一组表时变为死锁状态。事务会在提交或回滚时一次性解除其所有表锁定；而不会逐一放弃锁定。

例如，假设事务 T1 和 T2 在大致相同的时间开始。如果 T1 开始对表 A 进行写入且 T2 开始对表 B 进行写入，则两个事务均可继续而不会发生冲突；但是，如果 T1 完成了对表 A 的写入操作并需要开始对表 B 进行写入，它将无法继续，因为 T2 仍保持对 B 的锁定。相反，如果 T2 完成了对表 B 的写入操作并需要开始对表 A 进行写入，它也无法继续，因为 T1 仍保持对 A 的锁定。由于两个事务都不能在提交其所有写入操作之前解除其锁定，因此两个事务都不能继续。

为了避免这种死锁情况，您需要小心安排并发写入操作。例如，您应始终在各事务中按相同顺序更新表，如果指定了锁定，则应先按相同的顺序锁定表，然后再执行任何 DML 操作。

并发写入示例

以下伪代码示例演示事务如何在并行运行时继续或等待。

到相同表的并发 COPY 操作

事务 1 将行复制到 LISTING 表中：

```
begin;
copy listing from ...;
end;
```

事务 2 在单独的会话中同时开始，并尝试将多个行复制到 LISTING 表中。事务 2 必须等待事务 1 解除对 LISTING 表的写入锁定，然后才能继续。

```
begin;
[waits]
copy listing from ;
end;
```

如果一个或两个事务包含 INSERT 命令而非 COPY 命令，也会产生相同行为。

来自相同表的并发 DELETE 操作

事务 1 从表中删除行：

```
begin;
```

```
delete from listing where ...;
end;
```

事务 2 同时开始并尝试从相同表中删除行。它将成功，因为它会等待事务 1 完成后再尝试删除行。

```
begin
[waits]
delete from listing where ;
end;
```

如果一个或两个事务包含对相同表的 UPDATE 命令而非 DELETE 命令，也会产生相同行为。

具有读取和写入操作组合的并发事务

在此示例中，在提交之前，事务 1 从 USERS 表中删除行、重新加载表、运行 COUNT(*) 查询，然后 ANALYZE：

```
begin;
delete one row from USERS table;
copy ;
select count(*) from users;
analyze ;
end;
```

同时，事务 2 将开始。此事务尝试将额外的行复制到 USERS 表中、分析该表，然后运行与第一个事务相同的 COUNT(*) 查询：

```
begin;
[waits]
copy users from ...;
select count(*) from users;
analyze;
end;
```

第二个事务将成功，因为它必须等待第一个事务完成。其 COUNT 查询将返回基于已完成的加载的计数。

教程：从 Amazon S3 加载数据

在本教程中，您将了解从 Amazon S3 桶中的数据文件将数据加载到您的 Amazon Redshift 数据库表中的完整过程。

在本教程中，您将执行以下操作：

- 下载使用逗号分隔值 (CSV) 格式、字符分隔格式和固定宽度格式的数据文件。
- 创建一个 Amazon S3 桶，然后将数据文件上载到该桶。
- 启动 Amazon Redshift 集群并创建数据库表。
- 使用 COPY 命令从 Amazon S3 上的数据文件加载表。
- 诊断加载错误并修改您的 COPY 命令来更正这些错误。

估计时间：60 分钟

估算费用：集群每小时 1.00 美元

先决条件

您需要以下先决条件：

- 用于启动 Amazon Redshift 集群并在 Amazon S3 中创建桶的 AWS 账户。
- 您从 Amazon S3 加载测试数据的 AWS 凭证（IAM 角色）。如果您需要一个新的 IAM 角色，请转到创建 IAM 角色。
- SQL 客户端，如 Amazon Redshift 控制台查询编辑器。

本教程设计为单独使用。除了本教程之外，还建议您完成以下教程来更全面地了解如何设计和使用 Amazon Redshift 数据库：

- Amazon Redshift 入门指南将指导您完成创建 Amazon Redshift 集群和加载示例数据的过程。

概述

您可以通过使用 INSERT 命令或 COPY 命令来将数据添加到您的 Amazon Redshift 表。在 Amazon Redshift 数据仓库的规模和速度方面，COPY 命令要比 INSERT 命令快许多倍且更高效。

COPY 命令使用 Amazon Redshift 大规模并行处理 (MPP) 架构来从多个数据来源并行读取和加载数据。您可以从 Amazon S3、Amazon EMR 或任何可通过 Secure Shell (SSH) 连接访问的远程主机上的数据文件加载。或者，您可以直接从 Amazon DynamoDB 表加载。

在本教程中，您将使用 COPY 命令来从 Amazon S3 加载数据。此处提到的许多原则也适用于从其他数据来源加载。

要了解有关使用 COPY 命令的更多信息，请参阅以下资源：

- Amazon Redshift 加载数据的最佳实践
- 从 Amazon EMR 中加载数据
- 从远程主机中加载数据
- 从 Amazon DynamoDB 表中加载数据

步骤

步骤 1：创建集群

如果您已有要使用的集群，则可跳过这一步。

在本教程的练习中，将使用四节点集群。

创建集群

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。

   使用导航菜单，选择预置集群控制面板

   > **Important**
   >
   > 请确保您具有执行集群操作所需的权限。有关授予必要权限的信息，请参阅授权 Amazon Redshift 访问 AWS 服务。

2. 在右上角，选择要在其中创建集群的 AWS 区域。在本教程中，选择美国西部（俄勒冈州）。

3. 在导航菜单上，选择集群，然后选择创建集群。此时将显示创建集群页面。
4. 在创建集群页面输入您的集群参数。除更改以下值外，选择您自己的参数值：
   - 选择节点类型 **dc2.large**。
   - 为节点数量选择 **4**。
   - 在集群权限部分中，从可用 IAM 角色中选择一个 IAM 角色。此角色应是您之前创建的有权访问 Amazon S3 的角色。然后，选择关联 IAM 角色以将该角色添加到集群的关联的 IAM 角色列表中。
5. 选择创建集群。

按 Amazon Redshift 入门指南中的步骤操作，以从 SQL 客户端连接到您的集群并测试连接。您无需完成此入门中的剩余步骤，即可创建表、上传数据和尝试示例查询。

下一步

步骤 2：下载数据文件

步骤 2：下载数据文件

在此步骤中，您将一组示例数据文件下载到计算机。在下一个步骤中，您将这些文件上载到 Amazon S3 桶。

下载数据文件

1. 下载压缩文件：LoadingDataSampleFiles.zip。
2. 将文件提取到您计算机上的文件夹中。
3. 验证您的文件夹是否包含以下文件。

```
customer-fw-manifest
customer-fw.tbl-000
customer-fw.tbl-000.bak
customer-fw.tbl-001
customer-fw.tbl-002
customer-fw.tbl-003
customer-fw.tbl-004
customer-fw.tbl-005
customer-fw.tbl-006
customer-fw.tbl-007
customer-fw.tbl.log
```

```
dwdate-tab.tbl-000
dwdate-tab.tbl-001
dwdate-tab.tbl-002
dwdate-tab.tbl-003
dwdate-tab.tbl-004
dwdate-tab.tbl-005
dwdate-tab.tbl-006
dwdate-tab.tbl-007
part-csv.tbl-000
part-csv.tbl-001
part-csv.tbl-002
part-csv.tbl-003
part-csv.tbl-004
part-csv.tbl-005
part-csv.tbl-006
part-csv.tbl-007
```

下一步

步骤 3：将文件上载到 Amazon S3 桶

步骤 3：将文件上载到 Amazon S3 桶

在此步骤中，您将创建一个 Amazon S3 桶并将数据文件上载到该桶。

要将文件上载到 Amazon S3 桶

1. 在 Amazon S3 中创建一个桶。

   有关创建桶的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的创建桶。

   a. 登录到 AWS Management Console，然后通过以下网址打开 Amazon S3 控制台：https://console.aws.amazon.com/s3/。

   b. 选择创建桶。

   c. Choose an AWS 区域。

      在集群所在的区域中创建桶。如果您的集群位于美国西部（俄勒冈州）区域，请选择美国西部（俄勒冈州）区域 (us-west-2)。

   d. 在创建桶对话框的桶名称框中，输入桶名称。

所选的桶名称在 Amazon S3 的所有现有桶名称中必须具有唯一性。确保唯一性的一种办法是以您所在的组织的名称作为您的桶名称的前缀。桶名称必须符合特定规则。有关更多信息，请转至《Amazon Simple Storage Service 用户指南》中的桶限制。

e. 为其余选项选择推荐的默认值。

f. 选择创建桶。

Amazon S3 成功创建桶后，控制台的桶面板中将显示空桶。

2. 创建一个文件夹。

a. 选择新桶的名称。

b. 选择创建文件夹按钮。

c. 将新文件夹命名为 **`load`**。

> **Note**
>
> 您创建的桶不会显示在沙盒中。在本练习中，您将对象添加到实际桶。您需要根据对象在桶中存储的时间支付象征性的费用。有关 Amazon S3 定价的更多信息，请前往 Amazon S3 定价页面。

3. 将数据文件上载到新的 Amazon S3 桶。

a. 选择数据文件夹的名称。

b. 在上传向导中，选择添加文件。

按照 Amazon S3 控制台的说明上载您下载并提取的所有文件。

c. 选择上传。

用户凭证

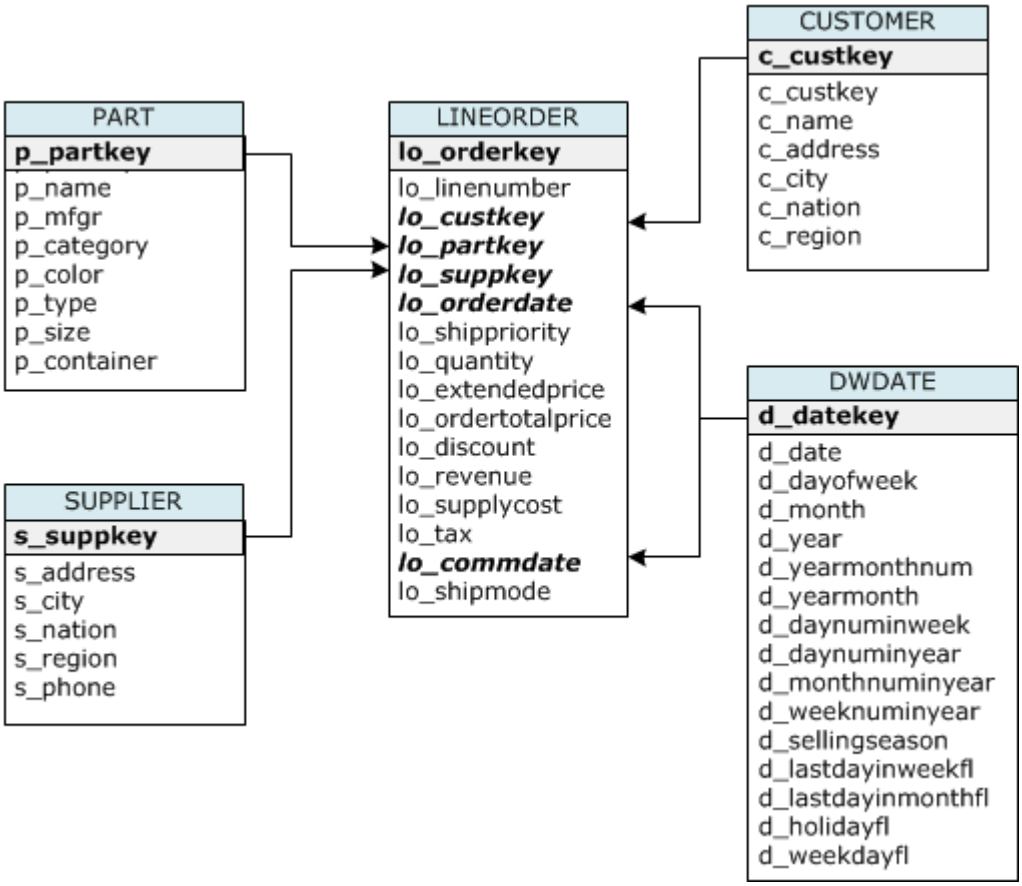
Amazon Redshift COPY 命令必须具有对 Amazon S3 桶中的文件对象的读访问权。如果您使用同一用户凭证来创建 Amazon S3 桶并运行 Amazon Redshift COPY 命令，则 COPY 命令将具有所有必要权限。如果您希望使用其他用户凭证，则可以通过使用 Amazon S3 访问控制来授予访问权限。Amazon Redshift COPY 命令至少需要 ListBucket 和 GetObject 权限才能访问 Amazon S3 桶中的文件对象。有关如何控制对 Amazon S3 资源的访问权限的更多信息，请前往管理您的 Amazon S3 资源的访问权限。

下一步

步骤 4：创建示例表

步骤 4：创建示例表

在本教程中，您将基于 Star Schema Benchmark (SSB) Schema 使用一组表（共 5 个）。下图显示了 SSB 数据模型。



SSB 表可能已存在于当前数据库中。如果是这样的话，请先从数据库中删除这些表，然后再在下一步骤中使用 CREATE TABLE 命令创建这些表。本教程中使用的表可能包含与现有表不同的属性。

创建示例表

1. 要删除 SSB 表，请在 SQL 客户端中运行以下命令。

```
drop table part cascade;
drop table supplier;
drop table customer;
drop table dwdate;
```

```
drop table lineorder;
```

2. 在 SQL 客户端中运行以下 CREATE TABLE 命令。

```
CREATE TABLE part
(
  p_partkey     INTEGER NOT NULL,
  p_name        VARCHAR(22) NOT NULL,
  p_mfgr        VARCHAR(6),
  p_category    VARCHAR(7) NOT NULL,
  p_brand1      VARCHAR(9) NOT NULL,
  p_color       VARCHAR(11) NOT NULL,
  p_type        VARCHAR(25) NOT NULL,
  p_size        INTEGER NOT NULL,
  p_container   VARCHAR(10) NOT NULL
);

CREATE TABLE supplier
(
  s_suppkey   INTEGER NOT NULL,
  s_name      VARCHAR(25) NOT NULL,
  s_address   VARCHAR(25) NOT NULL,
  s_city      VARCHAR(10) NOT NULL,
  s_nation    VARCHAR(15) NOT NULL,
  s_region    VARCHAR(12) NOT NULL,
  s_phone     VARCHAR(15) NOT NULL
);

CREATE TABLE customer
(
  c_custkey      INTEGER NOT NULL,
  c_name         VARCHAR(25) NOT NULL,
  c_address      VARCHAR(25) NOT NULL,
  c_city         VARCHAR(10) NOT NULL,
  c_nation       VARCHAR(15) NOT NULL,
  c_region       VARCHAR(12) NOT NULL,
  c_phone        VARCHAR(15) NOT NULL,
  c_mktsegment   VARCHAR(10) NOT NULL
);

CREATE TABLE dwdate
(
  d_datekey            INTEGER NOT NULL,
  d_date               VARCHAR(19) NOT NULL,
```

```
  d_dayofweek          VARCHAR(10) NOT NULL,
  d_month              VARCHAR(10) NOT NULL,
  d_year               INTEGER NOT NULL,
  d_yearmonthnum       INTEGER NOT NULL,
  d_yearmonth          VARCHAR(8) NOT NULL,
  d_daynuminweek       INTEGER NOT NULL,
  d_daynuminmonth      INTEGER NOT NULL,
  d_daynuminyear       INTEGER NOT NULL,
  d_monthnuminyear     INTEGER NOT NULL,
  d_weeknuminyear      INTEGER NOT NULL,
  d_sellingseason      VARCHAR(13) NOT NULL,
  d_lastdayinweekfl    VARCHAR(1) NOT NULL,
  d_lastdayinmonthfl   VARCHAR(1) NOT NULL,
  d_holidayfl          VARCHAR(1) NOT NULL,
  d_weekdayfl          VARCHAR(1) NOT NULL
);
CREATE TABLE lineorder
(
  lo_orderkey          INTEGER NOT NULL,
  lo_linenumber        INTEGER NOT NULL,
  lo_custkey           INTEGER NOT NULL,
  lo_partkey           INTEGER NOT NULL,
  lo_suppkey           INTEGER NOT NULL,
  lo_orderdate         INTEGER NOT NULL,
  lo_orderpriority     VARCHAR(15) NOT NULL,
  lo_shippriority      VARCHAR(1) NOT NULL,
  lo_quantity          INTEGER NOT NULL,
  lo_extendedprice     INTEGER NOT NULL,
  lo_ordertotalprice   INTEGER NOT NULL,
  lo_discount          INTEGER NOT NULL,
  lo_revenue           INTEGER NOT NULL,
  lo_supplycost        INTEGER NOT NULL,
  lo_tax               INTEGER NOT NULL,
  lo_commitdate        INTEGER NOT NULL,
  lo_shipmode          VARCHAR(10) NOT NULL
);
```

下一步

步骤 5：运行 COPY 命令

步骤 5：运行 COPY 命令

您将运行 COPY 命令来加载 SSB Schema 中的每个表。该 COPY 命令示例演示了使用多个 COPY 命令选项从不同文件格式加载并诊断加载错误。

主题

COPY 命令语法

基本 COPY 命令语法如下所示。

```
COPY table_name [ column_list ] FROM data_source CREDENTIALS access_credentials
 [options]
```

要运行 COPY 命令，您需提供以下值。

表名称

COPY 命令的目标表。该表必须已存在于数据库中。该表可以是临时的或永久的。COPY 命令会将新输入数据追加到该表中的任何现有行。

列列表

默认情况下，COPY 按顺序将源数据中的字段加载到表列中。您可以选择指定一个列列表（该列表是列名称的以逗号分隔的列表）以将数据字段映射到特定列。您在本教程中不使用列列表。有关更多信息，请参阅 COPY 命令参考中的Column List。

数据来源

您可以使用 COPY 命令从 Amazon S3 桶、Amazon EMR 集群、使用 SSH 连接的远程主机或 Amazon DynamoDB 表加载数据。在本教程中，您将从 Amazon S3 桶中的数据文件进行加载。在从 Amazon S3 进行加载时，您必须提供桶的名称和数据文件的位置。为此，请提供数据文件的对象路径或清单文件的位置，该清单文件明确列出了各个数据文件及其位置。

- 键前缀

  存储在 Amazon S3 中的对象由对象键唯一标识，其中包括桶名称、文件夹名称（如果有）和对象名称。键前缀 是指一组带有相同前缀的对象。对象路径是 COPY 命令用于加载共享键前

缀的所有对象的键前缀。例如，键前缀 custdata.txt 可以是一个文件或一组文件，包括 custdata.txt.001、custdata.txt.002 等。

- 清单文件

  在某些情况下，您可能需要加载具有不同前缀的文件，例如，从多个桶或文件夹中进行加载。在其他情况下，您可能需要排除共享前缀的文件。在这些情况下，您可以使用清单文件。清单文件 明确列出了每个加载文件及其唯一对象键。在本教程的后面，您将使用清单文件来加载 PART 表。

凭证

要访问包含要加载的数据的 AWS 资源，您必须为具有足够权限的用户提供 AWS 访问凭证。这些凭证包括 IAM 角色的 Amazon 资源名称（ARN）。要从 Amazon S3 加载数据，凭证必须包括 ListBucket 和 GetObject 权限。如果数据已加密，则需要其它凭证。有关更多信息，请参阅 COPY 命令参考中的授权参数。有关管理访问的更多信息，请转到管理对 Amazon S3 资源的访问权限。

选项

您可以为 COPY 命令指定大量参数，以指定文件格式、管理数据格式、管理错误和控制其他功能。在本教程中，您将使用以下 COPY 命令选项和功能：

- 键前缀

  有关如何通过指定键前缀从多个文件加载的信息，请参阅使用 NULL AS 加载 PART 表。
- CSV 格式

  有关如何加载 CSV 格式的数据的信息，请参阅使用 NULL AS 加载 PART 表。
- NULL AS

  有关如何使用 NULL AS 选项加载 PART 的信息，请参阅使用 NULL AS 加载 PART 表。
- 字符分隔的格式

  有关如何使用 DELIMITER 选项的信息，请参阅使用 REGION 加载 SUPPLIER 表。
- REGION

  有关如何使用 REGION 选项的信息，请参阅使用 REGION 加载 SUPPLIER 表。
- 固定格式宽度

  有关如何从固定宽度数据加载 CUSTOMER 表的信息，请参阅使用 MANIFEST 加载 CUSTOMER 表。

- MAXERROR

  有关如何使用 MAXERROR 选项的信息，请参阅使用 MANIFEST 加载 CUSTOMER 表。
- ACCEPTINVCHARS

  有关如何使用 ACCEPTINVCHARS 选项的信息，请参阅使用 MANIFEST 加载 CUSTOMER 表。
- MANIFEST

  有关如何使用 MANIFEST 选项的信息，请参阅使用 MANIFEST 加载 CUSTOMER 表。
- DATEFORMAT

  有关如何使用 DATEFORMAT 选项的信息，请参阅使用 DATEFORMAT 加载 DWDATE 表。
- GZIP、LZOP 和 BZIP2

  有关如何压缩文件的信息，请参阅使用多个文件加载 LINEORDER 表。
- COMPUPDATE

  有关如何使用 COMPUPDATE 选项的信息，请参阅使用多个文件加载 LINEORDER 表。
- 多个文件

  有关如何加载多个文件的信息，请参阅使用多个文件加载 LINEORDER 表。

加载 SSB 表

您将使用以下 COPY 命令加载 SSB Schema 中的每个表。针对每个表的命令演示了不同的 COPY 选项和疑难解答方法。

要加载 SSB 表，请按以下步骤操作：

1. 替换桶名称和 AWS 凭证
2. 使用 NULL AS 加载 PART 表
3. 使用 REGION 加载 SUPPLIER 表
4. 使用 MANIFEST 加载 CUSTOMER 表
5. 使用 DATEFORMAT 加载 DWDATE 表
6. 使用多个文件加载 LINEORDER 表

替换桶名称和 AWS 凭证

本教程中的 COPY 命令采用以下格式显示。

```
copy table from 's3://<your-bucket-name>/load/key_prefix'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
options;
```

对于每个 COPY 命令，请执行以下操作：

1. 将 *<your-bucket-name>* 替换为您的集群所在区域中的桶名称。

   此步骤假定桶和集群位于同一区域中。或者，您也可以通过对 COPY 命令使用 REGION 选项来指定区域。
2. 用您自己的 AWS 账户 和 IAM 角色替换 *<aws-account-id>* 和 *<role-name>*。用单引号引起来的凭证字符串的区段不得包含任何空格或换行符。请注意，ARN 格式可能与示例略有不同。运行 COPY 命令时，最好从 IAM 控制台复制角色的 ARN，以确保其准确无误。

使用 NULL AS 加载 PART 表

在此步骤中，您将使用 CSV 和 NULL AS 选项加载 PART 表。

COPY 命令可从多个文件并行加载数据，这比从一个文件加载数据快得多。为了演示此原则，本教程中每个表的数据将拆分为 8 个文件，即使这些文件非常小。在后面的步骤中，您将比较从一个文件加载所需的时间与从多个文件加载所需的时间的差异。有关更多信息，请参阅 加载数据文件。

键前缀

您可以通过为文件集指定键前缀，或通过在清单文件中明确列出文件，来从多个文件加载。在此步骤中，您将使用键前缀。在后面的步骤中，您将使用清单文件。键前缀 `'s3://mybucket/load/part-csv.tbl'` 加载 `load` 文件夹中的以下一组文件。

```
part-csv.tbl-000
part-csv.tbl-001
part-csv.tbl-002
part-csv.tbl-003
part-csv.tbl-004
part-csv.tbl-005
part-csv.tbl-006
part-csv.tbl-007
```

CSV 格式

CSV（表示用逗号分隔的值）是一种用于导入和导出电子表格数据的常见格式。CSV 比逗号分隔的格式更灵活，因为它使您能够在字段中包含带引号的字符串。CSV 格式的 COPY 的默认引号字符是双引号 (")，但您可以通过使用 QUOTE AS 选项指定另一种引号字符。在字段中使用引号字符时，应使用另一种引号字符对该字符进行转义。

PART 表中采用 CSV 格式的数据文件中的以下摘录显示了使用双引号引起来的字符串 ("LARGE ANODIZED BRASS")。它还显示一个用引号引起来的字符串中使用两个双引号引起来的字符串 ("MEDIUM ""BURNISHED"" TIN")。

```
15,dark sky,MFGR#3,MFGR#47,MFGR#3438,indigo,"LARGE ANODIZED BRASS",45,LG CASE
22,floral beige,MFGR#4,MFGR#44,MFGR#4421,medium,"PROMO, POLISHED BRASS",19,LG DRUM
23,bisque slate,MFGR#4,MFGR#41,MFGR#4137,firebrick,"MEDIUM ""BURNISHED"" TIN",42,JUMBO
 JAR
```

PART 表的数据包含将导致 COPY 命令失败的字符。在本练习中，您将诊断并纠正错误。

要加载采用 CSV 格式的数据，请将 `csv` 添加到您的 COPY 命令。运行以下命令可加载 PART 表。

```
copy part from 's3://<your-bucket-name>/load/part-csv.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
csv;
```

您可能会收到类似于以下内容的错误消息。

```
An error occurred when executing the SQL command:
copy part from 's3://mybucket/load/part-csv.tbl'
credentials' ...

ERROR: Load into table 'part' failed.  Check 'stl_load_errors' system table for
 details. [SQL State=XX000]

Execution time: 1.46s

1 statement(s) failed.
1 statement(s) failed.
```

要获取有关错误的更多信息，请查询 STL_LOAD_ERRORS 表。以下查询使用 SUBSTRING 函数缩短列以方便阅读，并使用 LIMIT 10 减少返回的行数。您可以调整 `substring(filename,22,25)` 中的值以允许您的桶名称的长度。

```
select query, substring(filename,22,25) as filename,line_number as line,
substring(colname,0,12) as column, type, position as pos, substring(raw_line,0,30) as
 line_text,
substring(raw_field_value,0,15) as field_text,
substring(err_reason,0,45) as reason
from stl_load_errors
order by query desc
limit 10;
```

```
 query  |    filename      | line |  column   |    type    | pos |
--------+--------------------------+-----------+------------+------------+-----+----
 333765 | part-csv.tbl-000 |    1 |           |            |   0 |

 line_text        | field_text |                    reason
------------------+------------+-----------------------------------------------
 15,NUL next,     |            | Missing newline: Unexpected character 0x2c f
```

NULL AS

part-csv.tbl 数据文件使用 NUL 终止符（\x000 或 \x0）来指示 NULL 值。

> **Note**
>
> 尽管 NUL 和 NULL 的拼写非常相似，但两者并不相同。NUL 是一种带代码点 x000 的 UTF-8 字符，通常用于指示记录结束 (EOR)。NULL 是一个表示数据缺失的 SQL 值。

默认情况下，COPY 将 NUL 终止符视为 EOR 字符并终止记录，这通常会导致意外结果或错误。没有在文本数据中指示 NULL 的单个标准方法。因此，使用 NULL AS COPY 命令选项可以指定加载表时替换为 NULL 的字符。在本示例中，您希望 COPY 将 NUL 终止符视为 NULL 值。

> **Note**
>
> 接收 NULL 值的表列必须配置为不可为 null。即，该表列不得包含 CREATE TABLE 规格中的 NOT NULL 约束。

要使用 NULL AS 选项加载 PART，请运行以下 COPY 命令。

```
copy part from 's3://<your-bucket-name>/load/part-csv.tbl'
```

```
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
csv
null as '\000';
```

要验证 COPY 加载的 NULL 值，请运行以下命令以仅选择包含 NULL 的行。

```
select p_partkey, p_name, p_mfgr, p_category from part where p_mfgr is null;
```

```
 p_partkey |  p_name  | p_mfgr | p_category
-----------+----------+--------+------------
        15 | NUL next |        | MFGR#47
        81 | NUL next |        | MFGR#23
       133 | NUL next |        | MFGR#44
(2 rows)
```

使用 REGION 加载 SUPPLIER 表

在此步骤中，您将使用 DELIMITER 和 REGION 选项来加载 SUPPLIER 表。

> **Note**
>
> 我们在 AWS 示例桶中提供了加载 SUPPLIER 表的文件。因此，在此步骤中，您无需上传文件。

字符分隔的格式

字符分隔的文件中的字段由某个特定字符（如竖线字符 (|)、逗号 (,) 或制表符 (\t)）分隔。字符分隔文件的可使用任一 ASCII 字符（包括非打印 ASCII 字符之一）作为分隔符。通过使用 DELIMITER 选项指定分隔符。默认分隔符是竖线字符 (|)。

SUPPLIER 表中数据的以下摘要使用竖线分隔的格式。

```
1|1|257368|465569|41365|19950218|2-HIGH|0|17|2608718|9783671|4|2504369|92072|2|
19950331|TRUCK
1|2|257368|201928|8146|19950218|2-HIGH|0|36|6587676|9783671|9|5994785|109794|6|
19950416|MAIL
```

REGION

只要可能，您应在 Amazon Redshift 集群所在的 AWS 区域中找到加载数据。如果您的数据和集群位于同一区域中，则将减少延迟并避免跨区域数据传输成本。有关更多信息，请参阅Amazon Redshift 加载数据的最佳实践。

如果您必须从另一个 AWS 区域加载数据，可使用 REGION 选项指定从中查找加载数据的 AWS 区域。如果您指定一个区域，则所有加载数据（包括清单文件）必须位于已命名的区域中。有关更多信息，请参阅 REGION。

如果您的集群位于美国东部（弗吉尼亚北部）区域，请运行以下命令来从位于美国西部（俄勒冈州）区域中的 Amazon S3 桶中用竖线分隔的数据加载 SUPPLIER 表。在本示例中，请不要更改桶名称。

```
copy supplier from 's3://awssampledbuswest2/ssbgz/supplier.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
delimiter '|'
gzip
region 'us-west-2';
```

如果您的集群未位于美国东部（弗吉尼亚北部）区域，请运行以下命令来从位于美国东部（弗吉尼亚北部）区域中的 Amazon S3 桶中用竖线分隔的数据加载 SUPPLIER 表。在本示例中，请不要更改桶名称。

```
copy supplier from 's3://awssampledb/ssbgz/supplier.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
delimiter '|'
gzip
region 'us-east-1';
```

使用 MANIFEST 加载 CUSTOMER 表

在此步骤中，您将使用 FIXEDWIDTH、MAXERROR、ACCEPTINVCHARS 和 MANIFEST 选项来加载 CUSTOMER 表。

本练习中的示例数据包含的字符将在 COPY 命令尝试加载数据时导致错误。您将使用 MAXERRORS 选项和 STL_LOAD_ERRORS 系统表来诊断加载错误，然后使用 ACCEPTINVCHARS 和 MANIFEST 选项来消除错误。

固定宽度格式

固定宽度格式将每个字段定义为固定数量的字符，而不是使用分隔符隔开的字段。CUSTOMER 表中数据的以下摘要使用固定宽度格式。

```
1   Customer#000000001   IVhzIApeRb           MOROCCO  0MOROCCO  AFRICA      25-705
2   Customer#000000002   XSTf4,NCwDVaWNe6tE   JORDAN   6JORDAN   MIDDLE EAST 23-453
3   Customer#000000003   MG9kdTD              ARGENTINA5ARGENTINAAMERICA     11-783
```

标签/宽度对的顺序必须与表列的顺序完全一致。有关更多信息，请参阅 FIXEDWIDTH。

CUSTOMER 表数据的固定宽度规范字符串如下所示。

```
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
c_region :12, c_phone:15,c_mktsegment:10'
```

要从固定宽度数据加载 CUSTOMER 表，请运行以下命令。

```
copy customer
from 's3://<your-bucket-name>/load/customer-fw.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
 c_region :12, c_phone:15,c_mktsegment:10';
```

您应收到类似于以下内容的错误消息。

```
An error occurred when executing the SQL command:
copy customer
from 's3://mybucket/load/customer-fw.tbl'
credentials'...

ERROR: Load into table 'customer' failed.  Check 'stl_load_errors' system table for
 details. [SQL State=XX000]

Execution time: 2.95s

1 statement(s) failed.
```

MAXERROR

原定设置情况下，COPY 首次遇到错误时，该命令将失败并返回错误消息。要在测试期间节省时间，您可以使用 MAXERROR 选项来指示 COPY 在跳过指定数量的错误后失败。由于我们预计了首次测试加载 CUSTOMER 表数据时的错误，因此将 `maxerror 10` 添加到 COPY 命令。

要使用 FIXEDWIDTH 和 MAXERROR 选项进行测试，请运行以下命令。

```
copy customer
from 's3://<your-bucket-name>/load/customer-fw.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
 c_region :12, c_phone:15,c_mktsegment:10'
maxerror 10;
```

这一次，您将收到类似于以下内容的警告消息，而不是错误消息。

```
Warnings:
Load into table 'customer' completed, 112497 record(s) loaded successfully.
Load into table 'customer' completed, 7 record(s) could not be loaded.  Check
 'stl_load_errors' system table for details.
```

该警告指示 COPY 遇到了 7 个错误。要查看这些错误，请查询 STL_LOAD_ERRORS 表，如以下示例所示。

```
select query, substring(filename,22,25) as filename,line_number as line,
substring(colname,0,12) as column, type, position as pos, substring(raw_line,0,30) as
 line_text,
substring(raw_field_value,0,15) as field_text,
substring(err_reason,0,45) as error_reason
from stl_load_errors
order by query desc, filename
limit 7;
```

STL_LOAD_ERRORS 查询的结果应类似于以下内容。

```
 query  |         filename          | line |  column   |    type    | pos |
 line_text           | field_text |              error_reason
--------+---------------------------+------+-----------
+------------+-----+-----------------------------------+------------
+--------------------------------------------
 334489 | customer-fw.tbl.log       |    2 | c_custkey | int4       |  -1 | customer-
fw.tbl              | customer-f | Invalid digit, Value 'c', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    6 | c_custkey | int4       |  -1 | Complete
                  | Complete   | Invalid digit, Value 'C', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    3 | c_custkey | int4       |  -1 | #Total rows
                  | #Total row | Invalid digit, Value '#', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    5 | c_custkey | int4       |  -1 | #Status
                  | #Status    | Invalid digit, Value '#', Pos 0, Type: Integ
```

```
 334489 | customer-fw.tbl.log       |    1 | c_custkey | int4       |  -1 | #Load file
                  | #Load file | Invalid digit, Value '#', Pos 0, Type: Integ
 334489 | customer-fw.tbl000        |    1 | c_address | varchar    |  34 | 1
 Customer#000000001  | .Mayag.ezR | String contains invalid or unsupported UTF8
 334489 | customer-fw.tbl000        |    1 | c_address | varchar    |  34 | 1
 Customer#000000001  | .Mayag.ezR | String contains invalid or unsupported UTF8
(7 rows)
```

通过检查结果，您可以看到 error_reasons 列中有以下两条消息：

- ```
  Invalid digit, Value '#', Pos 0, Type: Integ
  ```

  这些错误是由 customer-fw.tbl.log 文件导致的。问题在于它是一个日志文件而不是数据文件且不应加载。可以使用清单文件来避免加载错误文件。

- ```
  String contains invalid or unsupported UTF8
  ```

  VARCHAR 数据类型支持多达 3 个字符的多字节 UTF-8 字符。如果加载数据包含不支持的或无效的字符，则可使用 ACCEPTINVCHARS 选项将每个无效字符替换为指定的替代字符。

另一个加载问题是更难以检测 – 加载生成了意外结果。要调查此问题，请运行以下命令来查询 CUSTOMER 表。

```
select c_custkey, c_name, c_address
from customer
order by c_custkey
limit 10;
```

```
 c_custkey |          c_name           |         c_address
-----------+---------------------------+---------------------------
         2 | Customer#000000002        | XSTf4,NCwDVaWNe6tE
         2 | Customer#000000002        | XSTf4,NCwDVaWNe6tE
         3 | Customer#000000003        | MG9kdTD
         3 | Customer#000000003        | MG9kdTD
         4 | Customer#000000004        | XxVSJsL
         4 | Customer#000000004        | XxVSJsL
         5 | Customer#000000005        | KvpyuHCplrB84WgAi
         5 | Customer#000000005        | KvpyuHCplrB84WgAi
         6 | Customer#000000006        | sKZz0CsnMD7mp4Xd0YrBvx
         6 | Customer#000000006        | sKZz0CsnMD7mp4Xd0YrBvx
```

```
(10 rows)
```

行应是唯一的，但存在重复项。

检查意外结果的另一种方法是验证已加载的行的数量。在我们的示例中，应已加载 100000 行，但加载消息报告已加载 112497 条记录。导致加载了额外的行的原因是，COPY 加载了无关的文件 `customer-fw.tbl0000.bak`。

在本练习中，您将使用清单文件来避免加载错误文件。

ACCEPTINVCHARS

原定设置情况下，在 COPY 遇到列的数据类型不支持的字符时，它会跳过该行并返回错误。有关无效的 UTF-8 字符的信息，请参阅多字节字符加载错误。

您可使用 MAXERRORS 选项忽略错误并继续加载，然后查询 STL_LOAD_ERRORS 以找到无效的字符，并修复数据文件。但是，MAXERRORS 最适合用于解决加载问题，并且通常不应用于生产环境。

ACCEPTINVCHARS 选项通常是用于管理无效字符的更佳选择。ACCEPTINVCHARS 指示 COPY 将每个无效字符替换为指定的有效字符，然后继续加载操作。您可以指定任何有效的 ASCII 字符（NULL 除外）作为替换字符。默认替换字符是问号（?）。COPY 将多字节字符替换为等长的替换字符串。例如，一个 4 字节字符将替换为 `'????'`。

COPY 命令将返回包含无效 UTF-8 字符的行数。它还为每个受影响的行向 STL_REPLACEMENTS 系统表添加一个条目，每个节点分片最多添加 100 个行。还将替换其他无效的 UTF-8 字符，但不会记录这些替换事件。

ACCEPTINVCHARS 仅对 VARCHAR 列有效。

在此步骤中，您将使用替换字符 `'^'` 添加 ACCEPTINVCHARS。

MANIFEST

当您使用键前缀从 Amazon S3 执行 COPY 命令时，可能会加载不需要的表。例如，`'s3://mybucket/load/` 文件夹包含 8 个数据文件，这些文件共享键前缀 `customer-fw.tbl`：`customer-fw.tbl0000`、`customer-fw.tbl0001` 等。但是，同一文件夹也包含无关的文件：`customer-fw.tbl.log` 和 `customer-fw.tbl-0001.bak`。

要确保您加载所有正确的文件且仅有正确的文件，请使用清单文件。清单是采用 JSON 格式的文本文件，可明确列出要加载的每个源文件的唯一对象键。文件对象可位于不同的文件夹或桶中，但它们必须位于同一区域内。有关更多信息，请参阅 MANIFEST。

下面显示了 customer-fw-manifest 文本。

```
{
  "entries": [
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-000"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-001"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-002"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-003"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-004"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-005"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-006"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-007"}
    ]
}
```

使用清单文件加载 CUSTOMER 表的数据

1. 在文本编辑器中打开文件 customer-fw-manifest。
2. 将 *<your-bucket-name>* 替换为您的桶的名称。
3. 保存该文件。
4. 将文件上传到您的桶上的加载文件夹中。
5. 运行以下 COPY 命令。

```
copy customer from 's3://<your-bucket-name>/load/customer-fw-manifest'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
 c_region :12, c_phone:15,c_mktsegment:10'
maxerror 10
acceptinvchars as '^'
manifest;
```

使用 DATEFORMAT 加载 DWDATE 表

在此步骤中，您将使用 DELIMITER 和 DATEFORMAT 选项加载 DWDATE 表。

加载 DATE 和 TIMESTAMP 列时，COPY 应为默认格式，即 YYYY-MM-DD（对于日期）和 YYYY-MM-DD HH:MI:SS（对于时间戳）。如果加载数据不使用默认格式，则可使用 DATEFORMAT 和 TIMEFORMAT 指定格式。

以下摘要显示 DWDATE 表中的日期格式。请注意，列 2 中的日期格式不一致。

```
19920104 1992-01-04          Sunday  January 1992 199201 Jan1992 1 4 4 1...
19920112 January 12, 1992 Monday  January 1992 199201 Jan1992 2 12 12 1...
19920120 January 20, 1992 Tuesday     January 1992 199201 Jan1992 3 20 20 1...
```

DATEFORMAT

您只能指定一种日期格式。如果加载数据包含不一致的格式（可能位于不同的列中），或者格式在加载时未知，则使用带 'auto' 参数的 DATEFORMAT。指定 'auto' 后，COPY 命令将识别任何有效的日期或时间格式并将它转换为默认格式。在使用 DATEFORMAT 和 TIMEFORMAT 字符串时，'auto' 选项将识别一些不受支持的格式。有关更多信息，请参阅 在 DATEFORMAT 和 TIMEFORMAT 中使用自动识别。

要加载 DWDATE 表，请运行以下 COPY 命令。

```
copy dwdate from 's3://<your-bucket-name>/load/dwdate-tab.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
delimiter '\t'
dateformat 'auto';
```

使用多个文件加载 LINEORDER 表

此步骤使用 GZIP 和 COMPUPDATE 选项加载 LINEORDER 表。

在本练习中，您将从一个数据文件中加载 LINEORDER 表，然后从多个文件再次加载该表。通过执行此操作，您可以比较两种方法的加载时间。

> **Note**
>
> 我们在 AWS 示例桶中提供了加载 LINEORDER 表的文件。因此，在此步骤中，您无需上传文件。

GZIP、LZOP 和 BZIP2

您可以使用 gzip、lzop 或 bzip2 压缩格式压缩您的文件。在从压缩文件加载时，COPY 会在加载过程中解压缩这些文件。压缩文件将节省存储空间并缩短上传时间。

COMPUPDATE

当 COPY 加载无压缩编码的空表时，它会分析加载数据以确定最佳编码。然后，它会修改该表以在开始加载前使用这些编码。此分析过程比较费时，最多对每个表执行一次此过程。要节省时间，

您可以通过关闭 COMPUPDATE 来跳过此步骤。为了准确评估 COPY 时间，您将在此步骤中关闭 COMPUPDATE。

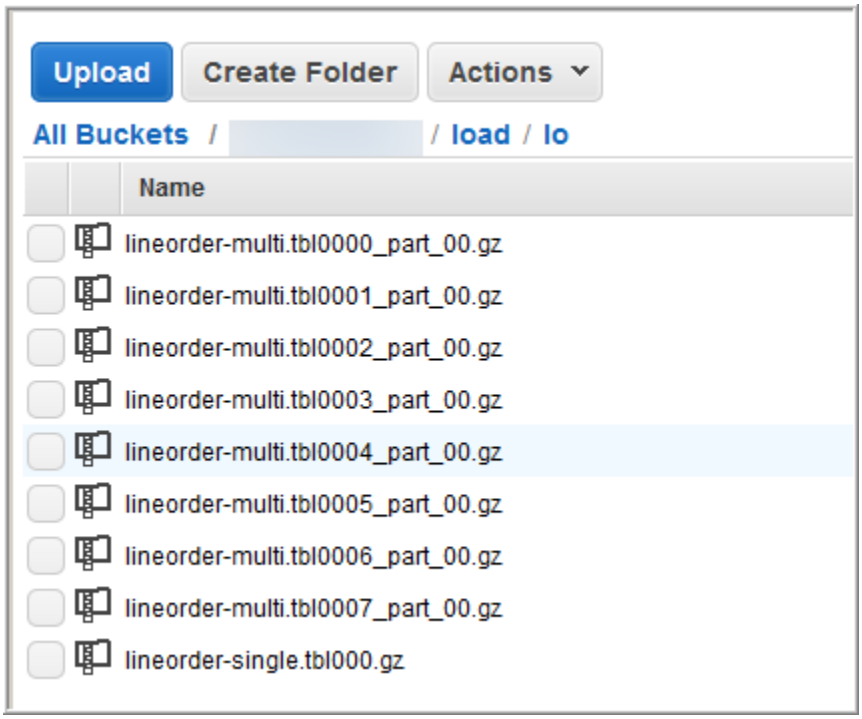
多个文件

COPY 命令可在从多个文件并行加载（而不是从一个文件加载）时非常高效地加载数据。您可以将数据拆分成多个文件，以便文件数是您的集群中的切片数的倍数。这样，Amazon Redshift 将划分工作负载并在切片之间均匀分配数据。每个节点的切片数取决于集群的节点大小。有关每个节点大小的切片数的更多信息，请转到《Amazon Redshift 管理指南》中的关于集群和节点。

例如，本教程中使用的每个 dc2.large 计算节点具有两个切片，因此 4 节点集群具有 8 个切片。在前面的步骤中，加载数据已包含在 8 个文件中，尽管这些文件非常小。在此步骤中，您将比较从一个大文件加载所需的时间与从多个文件加载所需的时间的差异。

您在本教程中使用的文件包含约 1500 万条记录，约占 1.2 GB 空间。这些文件在 Amazon Redshift 范围中非常小，但足以展示从多个文件加载的性能优势。这些文件过大，下载它们并将其上载到 Amazon S3 所需的时间对于本教程来说过多。因此，您将直接从 AWS 示例桶加载这些文件。

以下屏幕截图显示 LINEORDER 的数据文件。



使用多个文件评估 COPY 的性能

1. 从一个文件运行针对 COPY 的以下命令。请不要更改桶名称。

```
copy lineorder from 's3://awssampledb/load/lo/lineorder-single.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
gzip
compupdate off
region 'us-east-1';
```

2. 您的结果应类似于以下内容。请注意执行时间。

```
Warnings:
Load into table 'lineorder' completed, 14996734 record(s) loaded successfully.

0 row(s) affected.
copy executed successfully

Execution time: 51.56s
```

3. 从多个文件运行针对 COPY 的以下命令。请不要更改桶名称。

```
copy lineorder from 's3://awssampledb/load/lo/lineorder-multi.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
gzip
compupdate off
region 'us-east-1';
```

4. 您的结果应类似于以下内容。请注意执行时间。

```
Warnings:
Load into table 'lineorder' completed, 14996734 record(s) loaded successfully.

0 row(s) affected.
copy executed successfully

Execution time: 17.7s
```

5. 比较执行时间。

   在我们的示例中，加载 1500 万条记录所需的时间已从 51.56 秒减少至 17.7 秒（减少了 65.7%）。

这些结果通过使用 4 节点集群获得。如果您的集群具有更多节点，则将加倍节省时间。对于具有数十至数百个节点的典型 Amazon Redshift 集群，差异甚至更大。如果您有一个单节点集群，则执行时间之间的差异很小。

下一步

步骤 6：对数据库执行 vacuum 和分析操作

步骤 6：对数据库执行 vacuum 和分析操作

当添加、删除或修改大量行时，您应运行 VACUUM 命令，然后运行 ANALYZE 命令。vacuum 将从已删除行中恢复空间并还原排序顺序。ANALYZE 命令更新统计元数据，这使查询优化程序能够生成更准确的查询计划。有关更多信息，请参阅 对表执行 vacuum 操作。

如果您按排序键顺序加载数据，则 vacuum 操作的速度会很快。在本教程中，虽然您添加了大量行，但这些行添加到了空表中。在这种情况下，无需重新排序，而且您不删除任何行。COPY 在加载空表后自动更新统计数据，因此您的统计数据应是最新的。但是，要实现出色的事务管理，您需要通过对数据库执行 vacuum 和分析操作来完成本教程。

要对数据库执行 vacuum 和分析操作，请运行以下命令。

```
vacuum;
analyze;
```

下一步

步骤 7：清理资源

步骤 7：清理资源

只要您的集群正在运行，就将继续产生费用。完成本教程后，您应按照《Amazon Redshift 入门指南》中的步骤 5：撤消访问权限并删除示例集群中的步骤操作，以使您的环境返回上一个状态。

如果您希望保留集群，但恢复 SSB 表使用的存储空间，请运行以下命令。

```
drop table part;
drop table supplier;
drop table customer;
drop table dwdate;
```

```
drop table lineorder;
```

下一步

Summary

Summary

在本教程中，您已将数据文件上载到 Amazon S3，然后已使用 COPY 命令将数据从文件加载到 Amazon Redshift 表中。

您已使用以下格式加载数据：

- 字符分隔的
- CSV
- 固定宽度

您使用了 STL_LOAD_ERRORS 系统表来针对加载错误，然后使用了 REGION、MANIFEST、MAXERROR、ACCEPTINVCHARS、DATEFORMAT 和 NULL AS 选项纠正这些错误。

您应用了加载数据的以下最佳实践：

- 使用 COPY 命令加载数据
- 加载数据文件
- 使用一个 COPY 命令从多个文件中加载
- 压缩数据文件
- 在加载前后验证数据文件

有关 Amazon Redshift 最佳实践的更多信息，请参阅以下链接：

- Amazon Redshift 加载数据的最佳实践
- 设计表的 Amazon Redshift 最佳实践
- 设计查询的 Amazon Redshift 最佳实践

卸载数据

主题

要将数据从数据库表卸载到 Amazon S3 存储桶中的一组文件，可以使用包含 SELECT 语句的 UNLOAD 命令。不管加载时使用的是何种数据格式，您都可以按分隔格式或固定宽度格式卸载文本数据。此外，您还可以指定是否创建压缩的 GZIP 文件。

您可以使用临时安全凭证限制用户对您的 Amazon S3 存储桶的访问权限。

将数据卸载到 Amazon S3

Amazon Redshift 将 SELECT 语句的结果拆分到一组文件（每个节点切片分成一个或多个文件），以便并行重新加载数据。或者，您也可以通过添加 PARALLEL OFF 选项指定 UNLOAD 将结果串行写入到一个或多个文件。您可以通过指定 MAXFILESIZE 参数来限制 Amazon S3 中文件的大小。UNLOAD 使用 Amazon S3 服务器端加密 (SSE-S3) 自动加密数据文件。

您可以在 UNLOAD 命令中使用 Amazon Redshift 支持的任意 SELECT 语句，但在外部选择中使用 LIMIT 子句的 SELECT 语句除外。例如，您可以使用包含特定列或使用 WHERE 子句联接多个表的 SELECT 语句。如果查询包含引号（例如，引号中含有文本值），则您需要在查询文本中对其进行转义 (\')。有关更多信息，请参阅 SELECT 命令参考。有关使用 LIMIT 子句的更多信息，请参阅 UNLOAD 命令的 使用说明。

例如，下面的 UNLOAD 命令将 VENUE 表的内容发送到 Amazon S3 存储桶 `s3://mybucket/tickit/unload/`。

```
unload ('select * from venue')
to 's3://mybucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

上面示例创建的文件名包含前缀“`venue_`”。

```
venue_0000_part_00
```

```
venue_0001_part_00
venue_0002_part_00
venue_0003_part_00
```

默认情况下，UNLOAD 根据集群中切片的数量将数据并行写入到多个文件。要将数据写入到单个文件中，请指定 PARALLEL OFF。UNLOAD 按照 ORDER BY 子句（如果使用的话）的绝对排序串行写入数据。数据文件的最大大小为 6.2 GB。如果数据大小超过最大大小，则 UNLOAD 会创建新的文件，每个文件最大可达 6.2 GB。

下面的示例将内容 VENUE 写入到单个文件。文件大小不超过 6.2 GB，因此只需要一个文件。

```
unload ('select * from venue')
to 's3://mybucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off;
```

> **Note**
>
> UNLOAD 命令旨在使用并行处理。对于大多数情况，特别是文件将用于通过 COPY 命令加载表时，我们建议保留 PARALLEL 为启用状态。

假设 VENUE 的总数据大小为 5 GB，下面的示例将 VENUE 的内容写入 50 个文件，每个文件的大小为 100 MB。

```
unload ('select * from venue')
to 's3://mybucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off
maxfilesize 100 mb;
```

如果 Amazon S3 路径字符串中包含前缀，则 UNLOAD 会在文件名中使用该前缀。

```
unload ('select * from venue')
to 's3://mybucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

您可以通过在 UNLOAD 命令中指定 MANIFEST 选项来创建列出卸载文件的清单文件。清单是一个 JSON 格式的文本文件，其中显式列出写入到 Amazon S3 的每个文件的 URL。

下面的示例包含了清单选项。

```
unload ('select * from venue')
to 's3://mybucket/tickit/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

下面的示例显示了四个卸载文件的清单。

```
{
  "entries": [
    {"url":"s3://mybucket/tickit/venue_0000_part_00"},
    {"url":"s3://mybucket/tickit/venue_0001_part_00"},
    {"url":"s3://mybucket/tickit/venue_0002_part_00"},
    {"url":"s3://mybucket/tickit/venue_0003_part_00"}
  ]
}
```

清单文件可用于通过使用包含 MANIFEST 选项的 COPY 来加载相同的文件。有关更多信息，请参阅使用清单指定数据文件。

完成 UNLOAD 操作后，导航到 UNLOAD 写入文件的 Amazon S3 存储桶，确认数据已正确卸载。您会看到一个或多个按切片编号的文件（编号从零开始）。如果您指定了 MANIFEST 选项，则还会看到以“manifest”结尾的文件。例如：

```
mybucket/tickit/venue_0000_part_00
mybucket/tickit/venue_0001_part_00
mybucket/tickit/venue_0002_part_00
mybucket/tickit/venue_0003_part_00
mybucket/tickit/venue_manifest
```

您可以在 UNLOAD 完成后调用 Amazon S3 列表操作，以编程方式获取写入到 Amazon S3 的文件的列表。您还可以查询 STL_UNLOAD_LOG。

下面的查询返回由 UNLOAD 创建的文件的路径名称。PG_LAST_QUERY_ID 函数返回最新的查询。

```
select query, substring(path,0,40) as path
from stl_unload_log
where query=2320
order by path;
```

```
query |             path
-------+---------------------------------------
  2320 | s3://my-bucket/venue0000_part_00
  2320 | s3://my-bucket/venue0001_part_00
  2320 | s3://my-bucket/venue0002_part_00
  2320 | s3://my-bucket/venue0003_part_00
(4 rows)
```

如果数据量非常大，Amazon Redshift 可能会将文件按每个切片分成多个部分。例如：

```
venue_0000_part_00
venue_0000_part_01
venue_0000_part_02
venue_0001_part_00
venue_0001_part_01
venue_0001_part_02
...
```

下面的 UNLOAD 命令中的 SELECT 语句中包含带引号的字符串，因此，引号被转义了 (=\'OH\' ')。

```
unload ('select venuename, venuecity from venue where venuestate=\'OH\' ')
to 's3://mybucket/tickit/venue/ '
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

默认情况下，UNLOAD 宁可执行失败，也不会覆盖目标存储桶中的现有文件。要覆盖现有文件（包括清单文件），请指定 ALLOWOVERWRITE 选项。

```
unload ('select * from venue')
to 's3://mybucket/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
allowoverwrite;
```

卸载加密的数据文件

UNLOAD 会借助 AWS 托管的加密密钥 (SSE-S3) 自动创建使用 Amazon S3 服务器端加密的文件。您还可以指定利用 AWS Key Management Service 密钥进行服务器端加密（SSE-KMS），或利用客户管理的密钥进行客户端加密。UNLOAD 不支持使用客户管理的密钥进行 Amazon S3 服务器端加密。有关更多信息，请参阅使用服务器端加密保护数据。

要利用 AWS KMS 密钥进行服务器端加密来卸载到 Amazon S3，请如下例所示使用 KMS_KEY_ID 参数提供密钥 ID。

```
unload ('select venuename, venuecity from venue')
to 's3://mybucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
KMS_KEY_ID '1234abcd-12ab-34cd-56ef-1234567890ab'
encrypted;
```

如果您希望提供自己的加密密钥，则可以通过使用包含 ENCRYPTED 选项的 UNLOAD 命令在 Amazon S3 中创建客户端加密数据文件。UNLOAD 与 Amazon S3 客户端加密使用相同的信封加密过程。之后，您可以使用包含 ENCRYPTED 选项的 COPY 命令加载加密文件。

该过程的工作方式如下所示：

1. 创建用作私有加密密钥（也称作根对称密钥）的 base64 编码的 256 位 AES 密钥。
2. 发出包含根对称密钥及 ENCRYPTED 选项的 UNLOAD 命令。
3. UNLOAD 生成用于加密您的数据的一次性对称密钥（称作信封对称密钥）和初始化向量 (IV)。
4. UNLOAD 使用您的根对称密钥加密该信封对称密钥。
5. 然后，UNLOAD 将加密数据文件存储在 Amazon S3 中，并将加密信封密钥和 IV 作为对象元数据随每个文件存储。加密信封密钥作为对象元数据 `x-amz-meta-x-amz-key` 存储，IV 作为对象元数据 `x-amz-meta-x-amz-iv` 存储。

有关信封加密过程的更多信息，请参阅使用 AWS SDK for Java 和 Amazon S3 进行客户端数据加密一文。

要卸载加密数据文件，请将根密钥值添加到凭证字符串并包括 ENCRYPTED 选项。如果您使用了 MANIFEST 选项，清单文件也将加密。

```
unload ('select venuename, venuecity from venue')
to 's3://mybucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
manifest
encrypted;
```

要卸载经过 GZIP 压缩的加密数据文件，请包含 GZIP 选项及根密钥值和 ENCRYPTED 选项。

```
unload ('select venuename, venuecity from venue')
```

```
to 's3://mybucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted gzip;
```

要加载加密数据文件，请添加具有相同根密钥值的 MASTER_SYMMETRIC_KEY 参数，并包括 ENCRYPTED 选项。

```
copy venue from 's3://mybucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted;
```

以分隔或固定宽度格式卸载数据

您可以按分隔格式或固定宽度格式卸载数据。默认输出为管道分隔（使用“|”字符）。

下面的示例将逗号指定为分隔符：

```
unload ('select * from venue')
to 's3://mybucket/tickit/venue/comma'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter ',';
```

这会生成如下所示的输出文件：

```
20,Air Canada Centre,Toronto,ON,0
60,Rexall Place,Edmonton,AB,0
100,U.S. Cellular Field,Chicago,IL,40615
200,Al Hirschfeld Theatre,New York City,NY,0
240,San Jose Repertory Theatre,San Jose,CA,0
300,Kennedy Center Opera House,Washington,DC,0
...
```

要将相同的结果集卸载到制表符分隔的文件中，请发出下面的命令：

```
unload ('select * from venue')
to 's3://mybucket/tickit/venue/tab'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t';
```

或者，您也可以使用 FIXEDWIDTH 规范。该规范包含用于每个表列的分隔符及列的宽度（字符数）。UNLOAD 命令宁可失败也不会截断数据，因此，请指定至少与列的最长条目等长的宽度。卸载固定宽度数据的工作原理与卸载分隔数据相似，只不过生成的输出不包含分隔字符。例如：

```
unload ('select * from venue')
to 's3://mybucket/tickit/venue/fw'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth '0:3,1:100,2:30,3:2,4:6';
```

固定宽度输出如下所示：

```
20 Air Canada Centre         Toronto      ON0
60 Rexall Place              Edmonton     AB0
100U.S. Cellular Field       Chicago      IL40615
200Al Hirschfeld Theatre     New York CityNY0
240San Jose Repertory TheatreSan Jose     CA0
300Kennedy Center Opera HouseWashington   DC0
```

有关 FIXEDWIDTH 规范的更多信息，请参阅 UNLOAD 命令。

重新加载卸载的数据

要重新加载卸载操作的结果，您可以使用 COPY 命令。

下面的示例演示了一种简单的情况：使用清单文件卸载 VENUE 表、截断，然后重新加载。

```
unload  ('select * from venue order by venueid')
to 's3://mybucket/tickit/venue/reload_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
delimiter '|';

truncate venue;

copy venue
from 's3://mybucket/tickit/venue/reload_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
delimiter '|';
```

重新加载后，VENUE 表如下所示：

```
select * from venue order by venueid limit 5;

 venueid |         venuename          |  venuecity  | venuestate | venueseats
---------+----------------------------+-------------+------------+-----------
       1 | Toyota Park                | Bridgeview  | IL         |          0
       2 | Columbus Crew Stadium      | Columbus    | OH         |          0
       3 | RFK Stadium                | Washington  | DC         |          0
       4 | CommunityAmerica Ballpark | Kansas City | KS         |          0
       5 | Gillette Stadium           | Foxborough  | MA         |      68756
(5 rows)
```

创建用户定义的函数

您可以使用 SQL SELECT 子句或 Python 程序创建自定义标量用户定义函数 (UDF)。新函数存储在数据库中，可供任何具有足够权限的用户运行。您运行自定义标量 UDF 的方式与运行现有 Amazon Redshift 函数的方式大致相同。

对于 Python UDF，除了使用标准 Python 功能外，您还可以导入您自己的自定义 Python 模块。有关更多信息，请参阅适用于 UDF 的 Python 语言支持。请注意，Python 3 不适用于 Python UDF。要获得 Python 3 对 Amazon Redshift UDF 的支持，请改用 创建标量 Lambda UDF。

您还可以创建 AWS Lambda UDF 以将 Lambda 中定义的自定义函数用作 SQL 查询的一部分。Lambda UDF 使您能够编写复杂的 UDF 并与第三方组件集成。它们还可以帮助您克服当前 Python 和 SQL UDF 的一些局限性。例如，它们可以帮助您访问网络和存储资源，并编写更多完整的 SQL 语句。您可以使用 Lambda 支持的任何编程语言创建 Lambda UDF，如 Java、Go、PowerShell、Node.js、C#、Python 和 Ruby。或者您可以使用自定义运行时。

预设情况下，所有用户都可以执行 UDF。有关权限的更多信息，请参阅 UDF 安全性和权限。

主题

UDF 安全性和权限

要创建 UDF，您必须具有使用 SQL 或 plpythonu (Python) 语言的权限。默认情况下，向 PUBLIC 授予 USAGE ON LANGUAGE SQL 权限，但是，您必须明确授予 USAGE ON LANGUAGE PLPYTHONU 权限才能指定用户或组。

要撤销 SQL 的使用权限，请先从 PUBLIC 撤销使用权限。然后，仅向允许创建 SQL UDF 的特定用户或组授予 SQL 使用权限。以下示例从 PUBLIC 撤销对 SQL 的使用权限。然后它会向用户组 `udf_devs` 授予使用权限。

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

要执行 UDF，您必须拥有每个函数的执行权限。预设情况下，向 PUBLIC 授予运行新 UDF 的权限。要限制使用，请从 PUBLIC 撤销该函数的此权限。然后向特定的个人或组授予权限。

以下示例从 PUBLIC 撤销对函数 f_py_greater 的执行权限。然后它会向用户组 udf_devs 授予使用权限。

```
revoke execute on function f_py_greater(a float, b float) from PUBLIC;
grant execute on function f_py_greater(a float, b float) to group udf_devs;
```

默认情况下，超级用户拥有全部权限。

有关更多信息，请参阅 GRANT和 REVOKE。

创建标量 SQL UDF

标量 SQL UDF 纳入了一个 SQL SELECT 子句，该子句在此函数被调用并返回单个值时执行。CREATE FUNCTION 命令定义以下参数：

- （可选）输入参数。每个参数均必须具有一个数据类型。
- 一个返回数据类型。
- 一个 SQL SELECT 子句。在该 SELECT 子句中，使用 $1、$2 等按照参数在函数定义中的顺序引用输入参数。

输入和返回数据类型可以是任何标准 Amazon Redshift 数据类型。

请勿在 SELECT 子句中包括 FROM 子句。请改为在调用 SQL UDF 的 SQL 语句中包括 FROM 子句。

SELECT 子句不能包含以下任何类型的子句：

- FROM
- INTO
- WHERE
- GROUP BY
- ORDER BY

- LIMIT

标量 SQL 函数示例

以下示例创建一个用于比较两个数并返回较大值的函数。有关更多信息，请参阅CREATE FUNCTION。

```
create function f_sql_greater (float, float)
  returns float
stable
as $$
  select case when $1 > $2 then $1
    else $2
  end
$$ language sql;
```

以下查询将调用新的 f_sql_greater 函数以查询 SALES 表，并返回 COMMISSION 或 PRICEPAID 的 20% (两个值中的较大者)。

```
select f_sql_greater(commission, pricepaid*0.20) from sales;
```

对 UDF 命名

通过在实施前考虑您的 UDF 命名约定，可以避免潜在的冲突和意外结果。由于可以重载函数名称，因此它们可能会与现有和将来的 Amazon Redshift 函数名称发生冲突。本主题讨论重载并介绍一种避免冲突的策略。

重载函数名称

函数由其名称和签名 标识，后者是输入参数的数目和参数的数据类型。同一 schema 中的两个函数可具有相同的名称，前提是它们的签名不同。换句话说，函数名称可以重载。

在执行查询时，查询引擎将根据您提供的参数数目和参数的数据类型来确定要调用哪个函数。您可以使用重载来模拟具有可变数量（不超过 CREATE FUNCTION 命令允许的限制）的参数的函数。

防止 UDF 命名冲突

我们建议您使用前缀 f_ 对所有 UDF 进行命名。Amazon Redshift 保留 f_ 前缀专供 UDF 名称使用。通过使用 f_ 为您的 UDF 名称添加前缀，您可以确保您的 UDF 名称不会与现有或未来的 Amazon

Redshift 内置 SQL 函数名称冲突。例如，通过将新 UDF 命名为 `f_sum`，可以避免与 Amazon Redshift SUM 函数发生冲突。同样，如果您将新函数命名为 `f_fibonacci`，则当 Amazon Redshift 在将来版本中添加名为 FIBONACCI 的函数时可避免发生冲突。

您可以创建与现有 Amazon Redshift 内置 SQL 函数具有相同名称和签名的 UDF，而无需重载函数名，前提是该 UDF 和内置函数存在于不同的 schema 中。由于内置函数存在于系统目录 schema pg_catalog 中，因此您可以在另一个 schema（例如公共或用户定义的 schema）中创建具有相同名称的 UDF。在某些情况下，您可能会调用未显式限定 schema 名称的函数。如果是这样，预设情况下，Amazon Redshift 会首先搜索 pg_catalog schema。因此，内置函数在具有相同名称的新 UDF 之前运行。

您可以通过设置搜索路径以将 pg_catalog 置于结尾处来更改此行为。如果这样做，您的 UDF 会优先于内置函数，但这样做可能会导致意外结果。采用唯一命名策略（例如，使用预留前缀 `f_`）是更可靠的做法。有关更多信息，请参阅 SET和 search_path。

创建标量 Python UDF

标量 Python UDF 纳入了当此函数被调用并返回单个值时执行的一个 Python 程序。CREATE FUNCTION 命令定义以下参数：

- （可选）输入参数。每个参数均必须具有一个名称和一个数据类型。
- 一个返回数据类型。
- 一个可执行的 Python 程序。

输入和返回数据类型可以为 SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE 或 TIMESTAMP。此外，Python UDF 可以使用 ANYELEMENT 数据类型，Amazon Redshift 会根据在运行时提供的参数自动将该数据类型转换为标准数据类型。有关更多信息，请参阅ANYELEMENT 数据类型。

当 Amazon Redshift 查询调用标量 UDF 时，以下步骤将在运行时发生。

1. 函数将输入参数转换为 Python 数据类型。

   有关 Amazon Redshift 数据类型到 Python 数据类型的映射，请参阅Python UDF 数据类型。
2. 函数执行 Python 程序，并传递转换后的输入参数。
3. Python 代码返回一个值。返回值的数据类型必须对应于函数定义所指定的 RETURNS 数据类型。
4. 函数将 Python 返回值转换为指定的 Amazon Redshift 数据类型，然后将该值返回到查询。

> **Note**
>
> Python 3 不适用于 Python UDF。要获得 Python 3 对 Amazon Redshift UDF 的支持，请改用创建标量 Lambda UDF。

标量 Python UDF 示例

以下示例创建一个用于比较两个数并返回较大值的函数。请注意，双美元符号 ($$) 之间代码的缩进是一项 Python 要求。有关更多信息，请参阅CREATE FUNCTION。

```
create function f_py_greater (a float, b float)
  returns float
stable
as $$
  if a > b:
    return a
  return b
$$ language plpythonu;
```

以下查询将调用新的 f_greater 函数以查询 SALES 表，并返回 COMMISSION 或 PRICEPAID 的 20% (两个值中的较大者)。

```
select f_py_greater (commission, pricepaid*0.20) from sales;
```

Python UDF 数据类型

Python UDF 可将任何标准 Amazon Redshift 数据类型用于输入参数和函数的返回值。除标准数据类型外，UDF 还支持数据类型 ANYELEMENT，Amazon Redshift 会根据运行时提供的参数自动将该数据类型转换为标准数据类型。标量 UDF 可返回数据类型 ANYELEMENT。有关更多信息，请参阅ANYELEMENT 数据类型。

在执行期间，Amazon Redshift 会将参数从 Amazon Redshift 数据类型转换为 Python 数据类型以进行处理。然后，它会将返回值从 Python 数据类型转换为相应的 Amazon Redshift 数据类型。有关 Amazon Redshift 数据类型的更多信息，请参阅数据类型。

下表会将 Amazon Redshift 数据类型映射到 Python 数据类型。

Amazon Redshift 数据类型	Python 数据类型
smallint 整数 bigint short long	int
decimal 或 numeric	decimal
double real	float
布尔值	布尔
char varchar	字符串
timestamp	datetime

ANYELEMENT 数据类型

ANYELEMENT 是一种多态数据类型。这意味着，如果使用 ANYELEMENT 为参数的数据类型声明一个函数，则在调用该函数时，该函数可接受任何标准 Amazon Redshift 数据类型作为该参数的输入。ANYELEMENT 参数应设置为调用该函数时实际传递给该参数的数据类型。

如果某个函数使用多个 ANYELEMENT 数据类型，那么在调用该函数时，必须将这些数据类型全部解析为相同的实际数据类型。所有 ANYELEMENT 参数数据类型应设置为传递给 ANYELEMENT 的第一个参数的实际数据类型。例如，声明为 `f_equal(anyelement, anyelement)` 的函数将采用任意两个输入值，只要它们属于相同的数据类型。

如果某个函数的返回值声明为 ANYELEMENT，则至少一个输入参数必须是 ANYELEMENT。该返回值的实际数据类型与为 ANYELEMENT 输入参数提供的实际数据类型相同。

适用于 UDF 的 Python 语言支持

您可以基于 Python 编程语言创建自定义 UDF。Python 2.7 标准库可以在 UDF 中使用，但以下模块除外：

- ScrolledText
- Tix
- Tkinter
- tk
- turtle
- smtpd

除 Python 标准库之外，以下模块也是 Amazon Redshift 实施的一部分：

- numpy 1.8.2
- pandas 0.14.1
- python-dateutil 2.2
- pytz 2014.7
- scipy 0.12.1
- six 1.3.0
- wsgiref 0.1.2

您也可以通过执行 CREATE LIBRARY 命令来导入您自己的自定义 Python 模块并使其在 UDF 中可用。有关更多信息，请参阅导入自定义 Python 库模块。

Important

Amazon Redshift 将阻止通过 UDF 对文件系统进行的所有网络访问和写入访问。

Note

Python 3 不适用于 Python UDF。要获得 Python 3 对 Amazon Redshift UDF 的支持，请改用创建标量 Lambda UDF。

导入自定义 Python 库模块

您使用 Python 语言语法定义标量函数。您可以使用 Python 标准库模块和 Amazon Redshift 预安装的模块。您还可以创建您自己的自定义 Python 库模块并将这些库导入集群，或使用 Python 或第三方的现有库。

无法创建包含与 Python 标准库模块或 Amazon Redshift 预安装的 Python 模块同名的模块的库。如果现有用户安装的库使用与您创建的库相同的 Python 包，则必须先删除现有库，然后再安装新库。

您必须是超级用户或具有 USAGE ON LANGUAGE plpythonu 权限才能安装自定义库；不过，具有创建函数的足够权限的用户可以使用安装的库。您可以查询 PG_LIBRARY 系统目录以查看有关集群上安装的库的信息。

将自定义 Python 模块导入到集群

本部分提供了将自定义 Python 模块导入集群的示例。要执行本部分中的步骤，您必须拥有一个将库包上载到的 Simple Storage Service（Amazon S3）存储桶。然后将包安装到集群。有关创建存储桶的更多信息，请转至《Amazon Simple Storage Service 用户指南》中的创建存储桶。

在此示例中，假设您创建 UDF 以使用数据中的位置和距离。从 SQL 客户端工具连接到您的 Amazon Redshift 集群，并运行以下命令以创建函数。

```
CREATE FUNCTION f_distance (x1 float, y1 float, x2 float, y2 float) RETURNS float
 IMMUTABLE as $$
    def distance(x1, y1, x2, y2):
        import math
        return math.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)

    return distance(x1, y1, x2, y2)
$$ LANGUAGE plpythonu;

CREATE FUNCTION f_within_range (x1 float, y1 float, x2 float, y2 float) RETURNS bool
 IMMUTABLE as $$
    def distance(x1, y1, x2, y2):
        import math
        return math.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)

    return distance(x1, y1, x2, y2) < 20
$$ LANGUAGE plpythonu;
```

请注意，前面的函数中复制了几行代码。此复制是必要的，因为 UDF 无法引用其他 UDF 的内容，并且这两个函数都需要相同的功能。但是，您可以创建自定义库并将函数配置为使用该库，而不是复制多个函数中的代码。

为此，首先请执行以下步骤来创建库包：

1. 创建一个名为 geometry 的文件夹。此文件夹是库的顶级包。
2. 在 geometry 文件夹中，创建一个名为 `__init__.py` 的文件。请注意，该文件名包含两个下划线字符。此文件指示 Python 可初始化此包。
3. 另外，在 geometry 文件夹中，创建一个名为 trig 的文件夹。此文件夹是库的子包。
4. 在 trig 文件夹中，创建另一个名为 `__init__.py` 的文件和一个名为 `line.py` 的文件。在此文件夹中，`__init__.py` 指示 Python 可以初始化子包，并且 `line.py` 是包含库代码的文件。

   您的文件夹和文件结构应类似于以下内容：

   ```
   geometry/
      __init__.py
      trig/
         __init__.py
         line.py
   ```

   有关包结构的更多信息，请参阅 Python 网站上的 Python 教程中的模块。
5. 以下代码包含库的类和成员函数。请将其复制并粘贴到 `line.py`。

   ```
   class LineSegment:
     def __init__(self, x1, y1, x2, y2):
       self.x1 = x1
       self.y1 = y1
       self.x2 = x2
       self.y2 = y2
     def angle(self):
       import math
       return math.atan2(self.y2 - self.y1, self.x2 - self.x1)
     def distance(self):
       import math
       return math.sqrt((self.y2 - self.y1) ** 2 + (self.x2 - self.x1) ** 2)
   ```

在创建包之后，请执行以下操作以准备包并将其上载到 Simple Storage Service（Amazon S3）。

1. 将 geometry 文件夹的内容压缩到一个名为 geometry.zip 的 .zip 文件。请不要包括 geometry 文件夹本身；仅包括该文件夹的内容，如下所示：

```
geometry.zip
   __init__.py
   trig/
      __init__.py
      line.py
```

2. 将 geometry.zip 上载到您的 Simple Storage Service（Amazon S3）存储桶。

> **Important**
>
> 如果 Simple Storage Service（Amazon S3）存储桶不在您的 Amazon Redshift 集群所在的区域内，则必须使用 REGION 选项指定数据所在的区域。有关更多信息，请参阅CREATE LIBRARY。

3. 从您的 SQL 客户端工具运行以下命令可安装库。将 *<bucket_name>* 替换为您的存储桶的名称，并将 *<access key id>* 和 *<secret key>* 替换为您的 AWS Identity and Access Management (IAM) 用户凭证中的访问密钥和秘密访问密钥。

```
CREATE LIBRARY geometry LANGUAGE plpythonu FROM 's3://<bucket_name>/geometry.zip'
 CREDENTIALS 'aws_access_key_id=<access key id>;aws_secret_access_key=<secret key>';
```

在集群中安装库后，您需要配置函数以使用库。为此，请运行以下命令。

```
CREATE OR REPLACE FUNCTION f_distance (x1 float, y1 float, x2 float, y2 float) RETURNS
 float IMMUTABLE as $$
    from trig.line import LineSegment

    return LineSegment(x1, y1, x2, y2).distance()
$$ LANGUAGE plpythonu;

CREATE OR REPLACE FUNCTION f_within_range (x1 float, y1 float, x2 float, y2 float)
 RETURNS bool IMMUTABLE as $$
    from trig.line import LineSegment

    return LineSegment(x1, y1, x2, y2).distance() < 20
$$ LANGUAGE plpythonu;
```

在前面的命令中，`import trig/line` 消除了本部分的原始函数中重复的代码。您可以在多个 UDF 中重复使用此库提供的功能。请注意，要导入模块，您只需指定到子包的路径和模块名称 (`trig/line`)。

UDF 约束

在本主题列出的约束中，您可以在您使用 Amazon Redshift 内置标量函数的任意位置使用 UDF。有关更多信息，请参阅SQL 函数参考。

Amazon Redshift Python UDF 具有以下约束：

- Python UDF 不能访问网络或对文件系统进行读取/写入。
- 用户安装的 Python 库的总大小不能超过 100MB。
- 每个集群可同时运行的 Python UDF 的数目限制为集群的总并发级别的四分之一。例如，如果集群的并发配置为 15，则最多可同时运行 3 个 UDF。在达到上限后，UDF 将在工作负载管理队列中排队以等待执行。SQL UDF 没有并发限制。有关更多信息，请参阅实施工作负载管理。
- 使用 Python UDF 时，Amazon Redshift 不支持 SUPER 和 HLLSKETCH 数据类型。

在 UDF 中记录错误和警告

您可以使用 Python 日志记录模块在 UDF 中创建用户定义的错误和警告消息。执行查询后，您可以查询 SVL_UDF_LOG 系统视图以检索记录的消息。

> **Note**
>
> UDF 日志记录会使用集群资源，可能会影响系统性能。我们建议仅出于开发和故障诊断目的执行日志记录。

在执行查询时，日志处理程序会向 SVL_UDF_LOG 系统视图中写入消息，以及相应的函数名称、节点和切片。日志处理程序针对每个消息、每个切片向 SVL_UDF_LOG 中写入一行。消息被截断为 4096 字节。UDF 日志被限制为每个切片 500 行。当日志已满后，日志处理程序会丢弃较旧的消息并向 SVL_UDF_LOG 中添加警告消息。

> **Note**
>
> Amazon Redshift UDF 日志处理程序使用反斜杠 (\ ) 字符对换行符 (\n)、竖线 (|) 字符和反斜杠 (\ ) 字符进行转义。

默认情况下，UDF 日志级别设置为 WARNING。日志级别为 WARNING、ERROR 和 CRITICAL 的消息会被记录下来。严重性较低 (INFO、DEBUG 和 NOTSET) 的消息会被忽略。要设置 UDF 日志级别，请使用 Python 记录器方法。例如，以下代码将日志级别设置为 INFO。

```
logger.setLevel(logging.INFO)
```

有关使用 Python 日志记录模块的更多信息，请参阅 Python 文档中的 Python 的日志记录设施。

以下示例创建一个名为 f_pyerror 的函数，用于导入 Python 日志记录模块、实例化记录器以及记录错误。

```
CREATE OR REPLACE FUNCTION f_pyerror()
RETURNS INTEGER
VOLATILE AS
$$
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)
logger.info('Your info message here')
return 0
$$ language plpythonu;
```

以下示例查询 SVL_UDF_LOG 以查看上一示例中记录的消息。

```
select funcname, node, slice, trim(message) as message
from svl_udf_log;

  funcname  | query | node | slice |   message
------------+-------+------+-------+-----------------
  f_pyerror | 12345 |     1|     1 | Your info message here
```

创建标量 Lambda UDF

Amazon Redshift 可以将 AWS Lambda 中定义的自定义函数作为 SQL 查询的一部分。您可以使用 Lambda 支持的任何编程语言编写标量 Lambda UDF，如 Java、Go、PowerShell、Node.js、C#、Python 和 Ruby。或者您可以使用自定义运行时。

Lambda UDF 在 Lambda 中定义和管理，您可以控制在 Amazon Redshift 中调用这些 UDF 的访问权限。您可以在同一查询中调用多个 Lambda 函数，也可以多次调用相同的函数。

在支持标量函数的 SQL 语句的任何子句中使用 Lambda UDF。您还可以在任何 SQL 语句（如 SELECT、UPDATE、INSERT 或 DELETE）中使用 Lambda UDF。

> **Note**
>
> 使用 Lambda UDF 可能会产生来自 Lambda 服务的额外费用。是否这样做取决于 Lambda 请求的数量（UDF 调用）和 Lambda 程序执行的总持续时间等因素。但是，在 Amazon Redshift 中使用 Lambda UDF 不需要支付额外费用。有关 AWS Lambda 定价的信息，请参阅 AWS Lambda 定价。
>
> Lambda 请求的数量取决于使用 Lambda UDF 的特定 SQL 语句子句。例如，假设此函数用于 WHERE 子句，如下所示。
>
> ```
> SELECT a, b FROM t1 WHERE lambda_multiply(a, b) = 64; SELECT a, b
> FROM t1 WHERE a*b = lambda_multiply(2, 32)
> ```
>
> 在这种情况下，Amazon Redshift 为每个语句调用第一个 SELECT 语句，并仅调用第二个 SELECT 语句一次。
>
> 但是，在查询的投影部分中使用 UDF 可能只会为结果集中的每个限定行或聚合行调用一次 Lambda 函数。

注册 Lambda UDF

CREATE EXTERNAL FUNCTION 命令创建以下参数：

- （可选）具有数据类型的参数列表。
- 一个返回数据类型。
- 由 Amazon Redshift 调用的外部函数的一个函数名称。
- Amazon Redshift 集群有权代入并调用 Lambda 的一个 IAM 角色。
- Lambda UDF 调用的一个 Lambda 函数名称。

有关 CREATE EXTERNAL FUNCTION 的更多信息，请参阅CREATE EXTERNAL FUNCTION。

此函数的输入和返回数据类型可以是任何标准 Amazon Redshift 数据类型。

Amazon Redshift 确保外部函数可以发送和接收批量参数和结果。

管理 Lambda UDF 安全性和权限

要创建 Lambda UDF，请确保您具有在 LANGUAGE EXFUNC 上使用的权限。您必须将 USAGE ON LANGUAGE EXFUNC 明确授予给特定用户、组或公众，或撤销其 USAGE ON LANGUAGE EXFUNC。

以下示例将 EXFUNC 的使用权限授予 PUBLIC。

```
grant usage on language exfunc to PUBLIC;
```

以下示例将从 PUBLIC 撤销对 exfunc 的使用权限，然后向用户组 lambda_udf_devs 授予使用权限。

```
revoke usage on language exfunc from PUBLIC;
grant usage on language exfunc to group lambda_udf_devs;
```

要运行 Lambda UDF，请确保您对每个调用的函数都拥有权限。预设情况下，向 PUBLIC 授予运行新 Lambda UDF 的权限。要限制使用，请从 PUBLIC 撤销该函数的此权限。然后向特定用户或组授予权限。

以下示例从 PUBLIC 撤销对函数 exfunc_sum 的执行权限。然后它会向用户组 lambda_udf_devs 授予使用权限。

```
revoke execute on function exfunc_sum(int, int) from PUBLIC;
grant execute on function exfunc_sum(int, int) to group lambda_udf_devs;
```

默认情况下，超级用户拥有全部权限。

有关授予和撤消权限的更多信息，请参阅GRANT和REVOKE。

配置 Lambda UDF 的授权参数

CREATE EXTERNAL FUNCTION 命令需要授权才能在 AWS Lambda 中调用 Lambda 函数。要启动授权，请在您运行 CREATE EXTERNAL FUNCTION 命令时指定 AWS Identity and Access Management (IAM) 角色。有关 IAM 角色的更多信息，请参阅《IAM 用户指南》中的 IAM 角色。

如果现有 IAM 角色具有调用附加到集群的 Lambda 函数的权限，则您可以在命令的 IAM_ROLE 参数中替代您的角色 Amazon Resource Name (ARN)。以下部分介绍了在 CREATE EXTERNAL FUNCTION 命令中使用 IAM 角色的步骤。

为 Lambda 创建 IAM 角色

IAM 角色需要调用 Lambda 函数的权限。创建 IAM 角色时，请通过以下方式之一提供权限：

- 创建 IAM 角色时将 `AWSLambdaRole` 策略附加到 Attach permissions policy（附加权限策略）页面上。`AWSLambdaRole` 策略授予调用 Lambda 函数的权限，这是最低要求。有关更多信息和其他策略，请参阅 AWS Lambda 开发人员指南中的AWS Lambda 的基于身份的 IAM 策略。
- 创建您自己的自定义策略以附加到您的 IAM 角色，该角色具有对所有资源或具有该函数 ARN 的特定 Lambda 函数的 `lambda:InvokeFunction` 权限。有关如何创建策略的更多信息，请参阅 IAM 用户指南中的创建 IAM 策略。

  以下示例策略允许在特定 Lambda 函数上调用 Lambda。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Invoke",
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"
        }
    ]
}
```

  有关 Lambda 函数的资源的更多信息，请参阅 IAM API 参考中的 Lambda 操作的资源和条件。

  创建具有所需权限的自定义策略后，您可以在创建 IAM 角色的同时在 Attach permissions policy（附加权限策略）页面上将策略附加到该 IAM 角色。

有关创建 IAM 角色的步骤，请参阅《Amazon Redshift 管理指南》中的授权 Amazon Redshift 代表您访问其他 AWS 服务。

如果您不想创建新的 IAM 角色，您可以将之前提到的权限添加到现有 IAM 角色。

将 IAM 角色与集群关联

将 IAM 角色附加到您的集群。通过使用 Amazon Redshift 管理控制台、CLI 或 API，您可将角色添加到集群或查看与集群关联的角色。有关更多信息，请参阅《Amazon Redshift 管理指南》中的将 IAM 角色与集群关联。

在命令中包含 IAM 角色

在 CREATE EXTERNAL FUNCTION 命令中包含 IAM 角色 ARN。当您创建 IAM 角色时，IAM 将返回该角色的 Amazon Resource Name (ARN)。要指定 IAM 角色，请利用 `IAM_ROLE` 参数提供角色 ARN。以下显示 `IAM_ROLE` 参数的语法。

```
IAM_ROLE 'arn:aws:iam::aws-account-id:role/role-name'
```

要调用位于同一区域内的其他账户中的 Lambda 函数，请参阅在 Amazon Redshift 中串联 IAM 角色。

在 Amazon Redshift 和 AWS Lambda 之间使用 JSON 界面

Amazon Redshift 对于 Amazon Redshift 通信的所有 Lambda 函数使用通用界面。

下表显示了预计将用于 JSON 负载的指定 Lambda 函数的输入字段列表。

字段名称	描述	值范围
request_id	通用唯一标识符 (UUID)，用于唯一标识每个调用请求。	有效的 UUID。
集群	集群的完整 Amazon Resource Name (ARN)。	有效的集群 ARN。
user	进行调用的用户的名称。	有效用户名。
数据库	正在运行查询的数据库的名称。	有效的数据库名称。
external_function	进行调用的外部函数的完全限定名称。	有效的完全限定函数名称。

字段名称	描述	值范围
query_id	正在进行调用的查询的查询 ID。	有效的查询 ID。
num_records	负载中的参数数量。	值为 1 - 2^64。
arguments	指定格式的数据负载。	数组格式的数据必须是 JSON 数组。如果参数的数量大于 1，则每个元素都是一个数组的记录。通过使用数组，Amazon Redshift 可以保留记录在负载中的顺序。

JSON 数组的顺序决定了批处理的顺序。Lambda 函数必须以迭代方式处理参数并生成确切的记录数量。以下是负载的示例。

```
{
  "request_id" : "23FF1F97-F28A-44AA-AB67-266ED976BF40",
  "cluster" : "arn:aws:redshift:xxxx",
  "user" : "adminuser",
  "database" : "db1",
  "external_function": "public.foo",
  "query_id" : 5678234,
  "num_records" : 4,
  "arguments" : [
     [ 1, 2 ],
     [ 3, null],
     null,
     [ 4, 6]
   ]
 }
```

Lambda 函数的返回输出包含以下字段。

字段名称	描述	值范围
success	函数成功或失败的指示。	值为 "true" 或者 "false"。
error_msg	如果成功值为 "false"（如果函数	一个有效的消息。

字段名称	描述	值范围
	失败），则出现错误消息；否则，此字段将被忽略。	
num_records	负载中的记录数。	值为 1 - 2^64。
results	指定格式的调用结果。	不适用

以下是 Lambda 函数输出的示例。

```
{
  "success": true,   // true indicates the call succeeded
  "error_msg" : "my function isn't working",  // shall only exist when success != true
  "num_records": 4,      // number of records in this payload
  "results" : [
     1,
     4,
     null,
     7
   ]
}
```

当您通过 SQL 查询调用 Lambda 函数时，Amazon Redshift 会根据以下注意事项确保连接的安全性：

- GRANT 和 REVOKE 权限。有关 UDF 安全性和权限的更多信息，请参阅UDF 安全性和权限。
- Amazon Redshift 仅向指定的 Lambda 函数提交最小数据集。
- Amazon Redshift 只调用具有指定 IAM 角色的指定 Lambda 函数。

用户定义函数（UDF）的示例使用案例

通过将 Amazon Redshift 与其它组件集成，您可以使用用户定义函数来解决业务问题。以下是其它人如何在其使用案例中使用 UDF 的一些示例：

- 使用 Amazon Redshift Lambda UDF 访问外部组件 – 介绍 Amazon Redshift Lambda UDF 的工作原理，并介绍了创建 Lambda UDF 的过程。

- 借助 Amazon Redshift、Amazon Translate 和 Amazon Comprehend，使用 SQL 函数来翻译和分析文本 – 提供预构建的 Amazon Redshift Lambda UDF，您只需点击几下即可安装这些 UDF，以便翻译、编辑和分析文本字段。
- 使用 Amazon Redshift 访问 Amazon Location Service – 介绍如何将 Amazon Redshift Lambda UDF 与 Amazon Location Service 集成。
- 使用 Amazon Redshift 和 Protegrity 进行数据令牌化 – 介绍如何将 Amazon Redshift Lambda UDF 与 Protegrity Serverless 产品集成。
- Amazon Redshift UDF – Amazon Redshift SQL、Lambda 和 Python UDF 的集合。

在 Amazon Redshift 中创建存储过程

您可以定义 Amazon Redshift 存储过程，使用 PostgreSQL 程序性语言 PL/pgSQL 来执行一组 SQL 查询和逻辑运算。过程存储在数据库中，可供任何具有足够数据库权限的用户运行。

与用户定义的函数 (UDF) 不同，存储过程在 SELECT 查询之外，还可以纳入数据定义语言 (DDL) 和数据操纵语言 (DML)。存储过程无需返回值。您可以使用程序性程语言（包括循环和条件表达式）来控制逻辑流。

有关创建和管理存储过程的 SQL 命令的详细信息，请参阅以下命令主题：

- CREATE PROCEDURE
- ALTER PROCEDURE
- DROP PROCEDURE
- SHOW PROCEDURE
- CALL
- GRANT
- REVOKE
- ALTER DEFAULT PRIVILEGES

主题

- Amazon Redshift 中的存储过程概览
- PL/pgSQL 语言参考

Amazon Redshift 中的存储过程概览

存储过程通常用于针对数据转换、数据验证和特定于业务的逻辑来封装逻辑。通过将多个 SQL 步骤组合到一个存储过程中，可以减少应用程序与数据库之间的往返次数。

如需细粒度的访问控制，您可以创建存储过程来执行函数，而无需向用户提供对基础表的访问权限。例如，只有拥有者或超级用户可以截断表，用户需要写入权限来将数据插入到表中。您无需向用户提供对基础表的权限，而是可以创建执行该任务的存储过程。然后，向用户提供运行存储过程的权限。

具有 DEFINER 安全属性的存储过程使用存储过程拥有者的权限运行。默认情况下，存储过程具有 INVOKER 安全性，这意味着过程使用调用该过程的用户的权限。

要创建存储过程，请使用 CREATE PROCEDURE 命令。要运行过程，请使用 CALL 命令。此部分的后文中提供了示例。

> **Note**
>
> 一些客户端在创建 Amazon Redshift 存储过程时可能会显示以下错误。
>
> ```
> ERROR: 42601: [Amazon](500310) unterminated dollar-quoted string at or near "$$
> ```
>
> 由于客户端无法使用分隔语句的分号和引用的美元符号 ($) 来正确解析 CREATE PROCEDURE 语句，因此会发生此错误。这导致只有此语句的一部分发送到 Amazon Redshift 服务器。通常，您可以通过使用客户端的 `Run as batch` 或 `Execute selected` 选项来解决此错误。
> 例如，使用 Aginity 客户端时，请使用 `Run entire script as batch` 选项。使用 SQL Workbench/J 时，建议使用 124 版。使用 SQL Workbench/J 125 版时，请考虑指定备用分隔符作为解决方法。
> CREATE PROCEDURE 包含以分号 (;) 分隔的 SQL 语句。定义备用分隔符 [如正斜杠 (/)] 并将其放在 CREATE PROCEDURE 语句的末尾，会将整个语句发送到 Amazon Redshift 服务器进行处理。以下为示例。
>
> ```
> CREATE OR REPLACE PROCEDURE test()
> AS $$
> BEGIN
>   SELECT 1 a;
> END;
> $$
> LANGUAGE plpgsql
> ;
> /
> ```
>
> 有关更多信息，请参阅 SQL Workbench/J 文档中的备用分隔符。或者，使用可以更好地支持 CREATE PROCEDURE 语句分析的客户端，例如 Amazon Redshift 控制台中的查询编辑器或 TablePlus。

主题

- 命名存储过程
- 存储过程的安全性和权限

- 返回结果集
- 管理事务
- 捕获错误
- 记录存储过程
- 存储过程支持注意事项

以下示例显示没有输出参数的过程。默认情况下，参数是输入 (IN) 参数。

```
CREATE OR REPLACE PROCEDURE test_sp1(f1 int, f2 varchar)
AS $$
BEGIN
  RAISE INFO 'f1 = %, f2 = %', f1, f2;
END;
$$ LANGUAGE plpgsql;

call test_sp1(5, 'abc');
INFO: f1 = 5, f2 = abc
CALL
```

> **Note**
>
> 在编写存储过程时，我们建议使用最佳实践来保护敏感值：
> 不要在存储过程逻辑中硬编码任何敏感信息。例如，不要在存储过程主体的 CREATE USER 语句中分配用户密码。这会带来安全风险，因为硬编码值会作为架构元数据记录在目录表中。而是应通过参数将诸如密码之类的敏感值作为参量传递给存储过程。
> 有关存储过程的更多信息，请参阅 CREATE PROCEDURE 和在 Amazon Redshift 中创建存储过程。有关目录表的更多信息，请参阅系统目录表。

以下示例显示具有输出参数的过程。参数为 (IN)、输入和输出 (INOUT) 以及输出 (OUT)。

```
CREATE OR REPLACE PROCEDURE test_sp2(f1 IN int, f2 INOUT varchar(256), out_var OUT
 varchar(256))
AS $$
DECLARE
  loop_var int;
BEGIN
  IF f1 is null OR f2 is null THEN
```

```
    RAISE EXCEPTION 'input cannot be null';
  END IF;
  DROP TABLE if exists my_etl;
  CREATE TEMP TABLE my_etl(a int, b varchar);
    FOR loop_var IN 1..f1 LOOP
        insert into my_etl values (loop_var, f2);
        f2 := f2 || '+' || f2;
    END LOOP;
  SELECT INTO out_var count(*) from my_etl;
END;
$$ LANGUAGE plpgsql;


call test_sp2(2,'2019');

         f2          | column2
---------------------+---------
 2019+2019+2019+2019 | 2
(1 row)
```

命名存储过程

如果您定义的过程具有相同的名称，但输入参数数据类型或签名不同，则将创建新的过程。因此会重载过程名称。有关更多信息，请参阅重载过程名称。Amazon Redshift 不支持根据输出参数重载过程。您不能有两个过程，其名称和输入参数数据类型均相同但输出参数类型不同。

拥有者或超级用户可以将存储过程的正文替换为具有相同签名的新正文。要更改某个存储过程的签名或返回类型，请删除存储过程并重新创建它。有关更多信息，请参阅DROP PROCEDURE和CREATE PROCEDURE。

通过在实施前考虑您存储过程的命名约定，可以避免潜在的冲突和意外结果。由于您可以重载过程名称，它们可能与现有及以后的 Amazon Redshift 过程名称冲突。

重载过程名称

过程由其名称和签名标识，后者是输入参数的数目和参数的数据类型。同一 schema 中的两个函数可具有相同的名称，前提是它们的签名不同。换而言之，您可以重载过程名称。

在您运行过程时，查询引擎将根据您提供的参数数目和参数的数据类型来确定要调用哪个过程。您可以使用重载来模拟具有可变数量（不超过 CREATE PROCEDURE 命令允许的限制）的参数的过程。有关更多信息，请参阅CREATE PROCEDURE。

防止命名冲突

我们建议您使用前缀 sp_ 对所有过程进行命名。Amazon Redshift 保留 sp_ 前缀，将其专用于存储过程。通过将 sp_ 作为存储过程名称的前缀，您可以确保过程名称不会与任何现有或以后的 Amazon Redshift 过程名称冲突。

存储过程的安全性和权限

默认情况下，所有用户都有权创建过程。要创建过程，您必须具有语言 PL/pgSQL 的 USAGE 权限，此权限在默认情况下会授予 PUBLIC。默认情况下只有超级用户和拥有者有权调用过程。如果超级用户希望阻止用户创建存储过程，则可对用户运行 PL/pgSQL 上的 REVOKE USAGE。

要调用过程，您必须被授予了过程上的 EXECUTE 权限。默认情况下，新过程的 EXECUTE 权限授予过程拥有者和超级用户。有关更多信息，请参阅GRANT。

创建过程的用户默认是过程的拥有者。默认情况下，拥有者具有过程的 CREATE、DROP 和 EXECUTE 权限。超级用户拥有全部权限。

SECURITY 属性控制过程访问数据库对象的权限。在创建存储过程时，您可以将 SECURITY 属性设置为 DEFINER 或 INVOKER。如果您指定 SECURITY INVOKER，则过程使用调用该过程的用户的权限。如果您指定 SECURITY DEFINER，则过程使用该过程拥有者的权限。默认为 INVOKER。

由于 SECURITY DEFINER 过程使用拥有该过程的用户的权限运行，因此您必须确保没有误用该过程。为确保不会误用 SECURITY DEFINER 过程，请执行以下操作：

- 将 SECURITY DEFINER 过程上的 EXECUTE 授予特定用户，而不是授予 PUBLIC。
- 使用架构名称限定过程需要访问的所有数据库对象。例如，使用 `myschema.mytable` 而不是只有 `mytable`。
- 如果您无法通过 schema 限定其名称，请在创建过程时使用 SET 选项来设置 `search_path`。设置 `search_path` 以排除任何可由不可信用户写入的 schema。此方法防止该过程的任何调用方创建对象（例如，表或视图）来屏蔽过程所要使用的对象。有关 SET 选项的更多信息，请参阅 CREATE PROCEDURE。

以下示例将 `search_path` 设置为 `admin` 以确保从 `user_creds` schema 访问 `admin` 表，而不是从调用方的 `search_path` 中的公共或任何其他架构访问。

```
CREATE OR REPLACE PROCEDURE sp_get_credentials(userid int, o_creds OUT varchar)
AS $$
BEGIN
```

```
  SELECT creds INTO o_creds
  FROM user_creds
  WHERE user_id = $1;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER
-- Set a secure search_path
SET search_path = admin;
```

返回结果集

您可以使用游标或临时表返回结果集。

返回游标

要返回游标，请创建一个过程，该过程具有通过 refcursor 数据类型定义的 INOUT 参数。在调用过程时，向游标提供一个名称。然后，您就可以按名称从游标提取结果。

以下示例创建名为 get_result_set 的过程，该过程具有名为 rs_out 的 INOUT 参数，使用 refcursor 数据类型。该过程使用 SELECT 语句打开游标。

```
CREATE OR REPLACE PROCEDURE get_result_set (param IN integer, rs_out INOUT refcursor)
AS $$
BEGIN
  OPEN rs_out FOR SELECT * FROM fact_tbl where id >= param;
END;
$$ LANGUAGE plpgsql;
```

以下 CALL 命令打开名为 mycursor 的游标。仅在事务中使用游标。

```
BEGIN;
CALL get_result_set(1, 'mycursor');
```

打开游标之后，您可以从游标中提取数据，如下例所示。

```
FETCH ALL FROM mycursor;

    id | secondary_id | name
-------+--------------+---------
     1 |            1 | Joe
     1 |            2 | Ed
```

```
     2 |            1 | Mary
     1 |            3 | Mike
(4 rows)
```

最后，事务完成或回退。

```
COMMIT;
```

存储过程返回的游标需要遵循与 DECLARE CURSOR 中所述相同的约束和性能注意事项。有关更多信息，请参阅游标约束。

以下示例演示从 JDBC，使用 get_result_set 数据类型调用 refcursor 存储过程。文本 'mycursor'（游标的名称）传递到 prepareStatement。结果提取自 ResultSet。

```
static void refcursor_example(Connection conn) throws SQLException {
    conn.setAutoCommit(false);
    PreparedStatement proc = conn.prepareStatement("CALL get_result_set(1,
 'mycursor')");
    proc.execute();
    ResultSet rs = statement.executeQuery("fetch all from mycursor");
    while (rs.next()) {
      int n = rs.getInt(1);
      System.out.println("n " + n);
    }
```

使用临时表

要返回结果，您可以将句柄返回到包含结果行的临时表。客户端可以向存储过程提供名称作为参数。在存储过程内部，可以使用动态 SQL 在临时表上执行操作。下面是一个示例。

```
CREATE PROCEDURE get_result_set(param IN integer, tmp_name INOUT varchar(256)) as $$
DECLARE
  row record;
BEGIN
  EXECUTE 'drop table if exists ' || tmp_name;
  EXECUTE 'create temp table ' || tmp_name || ' as select * from fact_tbl where id <= '
 || param;
END;
$$ LANGUAGE plpgsql;

CALL get_result_set(2, 'myresult');
 tmp_name
```

```
-----------
 myresult
(1 row)

SELECT * from myresult;
 id | secondary_id | name
----+--------------+------
  1 |            1 | Joe
  2 |            1 | Mary
  1 |            2 | Ed
  1 |            3 | Mike
(4 rows)
```

管理事务

您可以创建具有默认事务管理行为或非原子行为的存储过程。

默认模式存储过程事务管理

默认事务模式自动提交行为会导致分别提交每个独自运行的 SQL 命令。对存储过程的调用视为单个 SQL 命令。过程中的 SQL 语句的行为就像在事务块中一样，在调用开始时隐式开始，在调用完成时结束。对其他过程的嵌套调用的处理方式，与对待调用方在相同事务上下文中的任何其他 SQL 语句和操作一样。有关自动提交行为的更多信息，请参阅可序列化的隔离。

但是，假设您从用户指定的事务块（由 BEGIN...COMMIT 定义）中调用存储过程。在这种情况下，存储过程中的所有语句都在用户指定事务的上下文中运行。过程不会在退出时隐式提交。调用方控制过程提交或回退。

如果使用存储过程时遇到任何错误，则将回退在当前事务中进行的所有更改。

您可以在存储过程中使用以下事务控制语句：

- COMMIT – 提交当前事务中完成的所有工作，并隐式开始新事务。有关更多信息，请参阅COMMIT。
- ROLLBACK – 会回滚当前事务中完成的所有工作，并隐式开始新事务。有关更多信息，请参阅ROLLBACK。

TRUNCATE 是可以从存储过程中发出的另一个语句，会影响事务管理。在 Amazon Redshift 中，TRUNCATE 隐式发布提交。此行为在存储过程的上下文中保持相同。从存储过程中发出 TRUNCATE 语句时，它会提交当前事务并开始新事务。有关更多信息，请参阅TRUNCATE。

COMMIT、ROLLBACK 或 TRUNCATE 语句后面的所有语句都会在新事务的上下文中运行，直到遇到 COMMIT、ROLLBACK 或 TRUNCATE 语句或存储过程结束。

在存储过程中使用 COMMIT、ROLLBACK 或 TRUNCATE 语句时，适用以下限制：

- 如果从事务块中调用存储过程，它就无法发出 COMMIT、ROLLBACK 或 TRUNCATE 语句。此限制适用于存储过程自身的主体和任何嵌套过程调用。
- 如果存储过程用 SET config 选项创建，它就无法发出 COMMIT、ROLLBACK 或 TRUNCATE 语句。此限制适用于存储过程自身的主体和任何嵌套过程调用。
- 任何打开的游标（显式或隐式）在处理 COMMIT、ROLLBACK 或 TRUNCATE 语句时会自动关闭。有关显式和隐式游标的约束，请参阅存储过程支持注意事项。

此外，您不能使用动态 SQL 运行 COMMIT 或 ROLLBACK。但是，您可以使用动态 SQL 运行 TRUNCATE。有关更多信息，请参阅动态 SQL。

处理存储过程时，请考虑将 PL/pgSQL 中的 BEGIN 和 END 语句仅用于分组。它们不启动或结束事务。有关更多信息，请参阅数据块。

以下示例演示在从明确的事务块中调用存储过程时的事务行为。从存储过程外部发布了两个插入语句，从内部发布一个语句，这都属于同一个事务 (3382)。用户明确发出提交时，提交事务。

```
CREATE OR REPLACE PROCEDURE sp_insert_table_a(a int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO test_table_a values (a);
END;
$$;

Begin;
  insert into test_table_a values (1);
  Call sp_insert_table_a(2);
  insert into test_table_a values (3);
Commit;

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  | pid |  type   |               stmt_text
--------+------+-----+---------+----------------------------------------
    103 | 3382 | 599 | UTILITY | Begin;
```

```
    103 | 3382 | 599 | QUERY   | insert into test_table_a values (1);
    103 | 3382 | 599 | UTILITY | Call sp_insert_table_a(2);
    103 | 3382 | 599 | QUERY   | INSERT INTO test_table_a values ( $1 )
    103 | 3382 | 599 | QUERY   | insert into test_table_a values (3);
    103 | 3382 | 599 | UTILITY | COMMIT
```

相反，例如从显式事务块外部发出相同的语句并且会话将自动提交设置为 ON 时。在这种情况下，每个语句都在自己的事务中运行。

```
insert into test_table_a values (1);
Call sp_insert_table_a(2);
insert into test_table_a values (3);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  | pid |  type   |
             stmt_text
--------+------+-----+---------
+-----------------------------------------------------------------------------------------------------------------
    103 | 3388 | 599 | QUERY   | insert into test_table_a values (1);
    103 | 3388 | 599 | UTILITY | COMMIT
    103 | 3389 | 599 | UTILITY | Call sp_insert_table_a(2);
    103 | 3389 | 599 | QUERY   | INSERT INTO test_table_a values ( $1 )
    103 | 3389 | 599 | UTILITY | COMMIT
    103 | 3390 | 599 | QUERY   | insert into test_table_a values (3);
    103 | 3390 | 599 | UTILITY | COMMIT
```

以下示例在插入到 test_table_a 中之后发布 TRUNCATE 语句。TRUNCATE 语句发出隐式提交，提交当前事务 (3335) 并启动新事务 (3336)。新事务在过程退出时提交。

```
CREATE OR REPLACE PROCEDURE sp_truncate_proc(a int, b int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO test_table_a values (a);
  TRUNCATE test_table_b;
  INSERT INTO test_table_b values (b);
END;
$$;

Call sp_truncate_proc(1,2);
```

```
select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |  pid  |  type   |
                                        stmt_text
--------+------+-------+---------
+-----------------------------------------------------------------------------------------------
    103 | 3335 | 23636 | UTILITY | Call sp_truncate_proc(1,2);
    103 | 3335 | 23636 | QUERY   | INSERT INTO test_table_a values ( $1 )
    103 | 3335 | 23636 | UTILITY | TRUNCATE test_table_b
    103 | 3335 | 23636 | UTILITY | COMMIT
    103 | 3336 | 23636 | QUERY   | INSERT INTO test_table_b values ( $1 )
    103 | 3336 | 23636 | UTILITY | COMMIT
```

以下示例从嵌套调用中发出 TRUNCATE。TRUNCATE 提交事务 (3344) 的所有外部和内部过程中迄今为止完成所有工作。它会启动新事务 (3345)。新事务在外部过程退出时提交。

```
CREATE OR REPLACE PROCEDURE sp_inner(c int, d int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO inner_table values (c);
  TRUNCATE outer_table;
  INSERT INTO inner_table values (d);
END;
$$;

CREATE OR REPLACE PROCEDURE sp_outer(a int, b int, c int, d int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO outer_table values (a);
  Call sp_inner(c, d);
  INSERT INTO outer_table values (b);
END;
$$;

Call sp_outer(1, 2, 3, 4);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;
```

```
 userid | xid  |  pid  |  type   |
                                          stmt_text
--------+------+-------+---------
+-----------------------------------------------------------------------------------------------------------
    103 | 3344 | 23636 | UTILITY | Call sp_outer(1, 2, 3, 4);
    103 | 3344 | 23636 | QUERY   | INSERT INTO outer_table values ( $1 )
    103 | 3344 | 23636 | UTILITY | CALL sp_inner( $1 , $2 )
    103 | 3344 | 23636 | QUERY   | INSERT INTO inner_table values ( $1 )
    103 | 3344 | 23636 | UTILITY | TRUNCATE outer_table
    103 | 3344 | 23636 | UTILITY | COMMIT
    103 | 3345 | 23636 | QUERY   | INSERT INTO inner_table values ( $1 )
    103 | 3345 | 23636 | QUERY   | INSERT INTO outer_table values ( $1 )
    103 | 3345 | 23636 | UTILITY | COMMIT
```

下面的示例演示了在提交 TRUNCATE 语句时关闭游标 cur1。

```
CREATE OR REPLACE PROCEDURE sp_open_cursor_truncate()
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  cur1 cursor for select * from test_table_a order by 1;
BEGIN
  open cur1;
  TRUNCATE table test_table_b;
  Loop
    fetch cur1 into rec;
    raise info '%', rec.c1;
    exit when not found;
  End Loop;
END
$$;

call sp_open_cursor_truncate();
ERROR: cursor "cur1" does not exist
CONTEXT: PL/pgSQL function "sp_open_cursor_truncate" line 8 at fetch
```

下面的示例发出 TRUNCATE 语句，并且无法从明确的事务块内部调用。

```
CREATE OR REPLACE PROCEDURE sp_truncate_atomic() LANGUAGE plpgsql
AS $$
BEGIN
  TRUNCATE test_table_b;
```

```
END;
$$;

Begin;
  Call sp_truncate_atomic();
ERROR: TRUNCATE cannot be invoked from a procedure that is executing in an atomic
 context.
HINT: Try calling the procedure as a top-level call i.e. not from within an explicit
 transaction block.
Or, if this procedure (or one of its ancestors in the call chain) was created with SET
 config options, recreate the procedure without them.
CONTEXT: SQL statement "TRUNCATE test_table_b"
PL/pgSQL function "sp_truncate_atomic" line 2 at SQL statement
```

以下示例显示，不是超级用户或表所有者的用户可以对表发出 TRUNCATE 语句。用户使用 Security Definer 存储过程执行此操作。该示例显示以下操作：

- user1 创建表 test_tbl。
- user1 创建存储过程 sp_truncate_test_tbl。
- user1 将对于存储过程的 EXECUTE 权限授予 user2。
- user2 运行此存储过程来截断表 test_tbl。该示例显示 TRUNCATE 命令之前和之后的行计数。

```
set session_authorization to user1;
create table test_tbl(id int, name varchar(20));
insert into test_tbl values (1,'john'), (2, 'mary');
CREATE OR REPLACE PROCEDURE sp_truncate_test_tbl() LANGUAGE plpgsql
AS $$
DECLARE
  tbl_rows int;
BEGIN
  select count(*) into tbl_rows from test_tbl;
  RAISE INFO 'RowCount before Truncate: %', tbl_rows;
  TRUNCATE test_tbl;
  select count(*) into tbl_rows from test_tbl;
  RAISE INFO 'RowCount after Truncate: %', tbl_rows;
END;
$$ SECURITY DEFINER;
grant execute on procedure sp_truncate_test_tbl() to user2;
reset session_authorization;
```

```
set session_authorization to user2;
call sp_truncate_test_tbl();
INFO:  RowCount before Truncate: 2
INFO:  RowCount after Truncate: 0
CALL
reset session_authorization;
```

以下示例两次发出 COMMIT。第一个 COMMIT 提交在事务 10363 中完成的所有工作，并隐式启动事务 10364。事务 10364 则由第二个 COMMIT 语句提交。

```
CREATE OR REPLACE PROCEDURE sp_commit(a int, b int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO test_table values (a);
  COMMIT;
  INSERT INTO test_table values (b);
  COMMIT;
END;
$$;

call sp_commit(1,2);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;
 userid |  xid  | pid  |  type   |
                              stmt_text
--------+-------+------+---------
+-----------------------------------------------------------------------------------------------------
    100 | 10363 | 3089 | UTILITY | call sp_commit(1,2);
    100 | 10363 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10363 | 3089 | UTILITY | COMMIT
    100 | 10364 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10364 | 3089 | UTILITY | COMMIT
```

如果 sum_vals 大于 2，以下示例将发出 ROLLBACK 语句。第一个 ROLLBACK 语句回滚事务 10377 中完成的所有工作并启动新事务 10378。事务 10378 则在过程退出时提交。

```
CREATE OR REPLACE PROCEDURE sp_rollback(a int, b int) LANGUAGE plpgsql
AS $$
DECLARE
  sum_vals int;
BEGIN
```

```
  INSERT INTO test_table values (a);
  SELECT sum(c1) into sum_vals from test_table;
  IF sum_vals > 2 THEN
    ROLLBACK;
  END IF;

  INSERT INTO test_table values (b);
END;
$$;

call sp_rollback(1, 2);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

userid |  xid  | pid  |  type   |
                              stmt_text
--------+-------+------+---------
+-----------------------------------------------------------------------------------------------------------
    100 | 10377 | 3089 | UTILITY | call sp_rollback(1, 2);
    100 | 10377 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10377 | 3089 | QUERY   | SELECT sum(c1) from test_table
    100 | 10377 | 3089 | QUERY   | Undoing 1 transactions on table 133646 with current
 xid 10377 : 10377
    100 | 10378 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10378 | 3089 | UTILITY | COMMIT
```

非原子模式存储过程事务管理

在 NONATOMIC 模式下创建的存储过程，与在默认模式下创建的过程，其事务控制行为并不相同。与存储过程之外的 SQL 命令自动提交行为类似，NONATOMIC 过程中的每个 SQL 语句都在自己的事务中运行并自动提交。如果用户在 NONATOMIC 存储过程中启动显式事务块，则该块中的 SQL 语句不会自动提交。事务块控制其中的语句的提交或回滚。

在 NONATOMIC 存储过程中，您可以使用 START TRANSACTIOM 语句打开过程中的显式事务块。但是，如果其中已有一个打开的事务块，则此语句将没有任何效果，因为 Amazon Redshift 不支持子事务。先前的事务会继续执行。

在 NONATOMIC 过程中使用游标 FOR 循环时，请确保在遍历查询的结果之前打开一个显式事务块。否则，在自动提交循环内的 SQL 语句时，游标将关闭。

使用 NONATOMIC 模式行为时的一些注意事项如下：

- 如果没有打开的事务块并且会话启用了自动提交，则存储过程中的每个 SQL 语句都会自动提交。
- 如果从事务块中调用存储过程，您可以发布 COMMIT/ROLLBACK/TRUNCATE 语句来结束事务。这在默认模式下无法实现。
- 您可以发布 START TRANSACTION 语句来启动存储过程中的事务块。

以下示例演示了使用 NONATOMIC 存储过程时的事务行为。以下所有示例的会话都已启用自动提交。

在以下示例中，NONATOMIC 存储过程有两个 INSERT 语句。当在事务块之外调用该过程时，该过程中的每个 INSERT 语句都会自动提交。

```
CREATE TABLE test_table_a(v int);
CREATE TABLE test_table_b(v int);

CREATE OR REPLACE PROCEDURE sp_nonatomic_insert_table_a(a int, b int) NONATOMIC AS
$$
BEGIN
    INSERT INTO test_table_a values (a);
    INSERT INTO test_table_b values (b);
END;
$$
LANGUAGE plpgsql;

Call sp_nonatomic_insert_table_a(1,2);

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |               stmt_text
--------+------+------------+---------+----------------------------------------
      1 | 1792 | 1073807554 | UTILITY | Call sp_nonatomic_insert_table_a(1,2);
      1 | 1792 | 1073807554 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1792 | 1073807554 | UTILITY | COMMIT
      1 | 1793 | 1073807554 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1793 | 1073807554 | UTILITY | COMMIT
(5 rows)
```

但是，当从 BEGIN..COMMIT 块中调用该过程时，所有语句都属于同一个事务（xid=1799）。

```
Begin;
  INSERT INTO test_table_a values (10);
```

```
  Call sp_nonatomic_insert_table_a(20,30);
  INSERT INTO test_table_b values (40);
Commit;

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |                stmt_text
--------+------+------------+---------+-----------------------------------------
      1 | 1799 | 1073914035 | UTILITY | Begin;
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_a values (10);
      1 | 1799 | 1073914035 | UTILITY | Call sp_nonatomic_insert_table_a(20,30);
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_b values (40);
      1 | 1799 | 1073914035 | UTILITY | COMMIT
(7 rows)
```

在此示例中，START TRANSACTION...COMMIT 之间有两条 INSERT 语句。在事务块之外调用该过程时，两条 INSERT 语句位于同一个事务（xid=1866）中。

```
CREATE OR REPLACE PROCEDURE sp_nonatomic_txn_block(a int, b int) NONATOMIC AS
$$
BEGIN
    START TRANSACTION;
    INSERT INTO test_table_a values (a);
    INSERT INTO test_table_b values (b);
    COMMIT;
END;
$$
LANGUAGE plpgsql;

Call sp_nonatomic_txn_block(1,2);

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |               stmt_text
--------+------+------------+---------+----------------------------------------
      1 | 1865 | 1073823998 | UTILITY | Call sp_nonatomic_txn_block(1,2);
      1 | 1866 | 1073823998 | QUERY   | INSERT INTO test_table_a values ( $1 )
```

```
      1 | 1866 | 1073823998 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1866 | 1073823998 | UTILITY | COMMIT
(4 rows)
```

从 BEGIN...COMMIT 块中调用该过程时，该过程中的 START TRANSACTION 不执行任何操作，因为已有一个打开的事务。过程中的 COMMIT 提交当前事务（xid = 1876）并启动新事务。

```
Begin;
  INSERT INTO test_table_a values (10);
  Call sp_nonatomic_txn_block(20,30);
  INSERT INTO test_table_b values (40);
Commit;

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |               stmt_text
--------+------+------------+---------+----------------------------------------
      1 | 1876 | 1073832133 | UTILITY | Begin;
      1 | 1876 | 1073832133 | QUERY   | INSERT INTO test_table_a values (10);
      1 | 1876 | 1073832133 | UTILITY | Call sp_nonatomic_txn_block(20,30);
      1 | 1876 | 1073832133 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1876 | 1073832133 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1876 | 1073832133 | UTILITY | COMMIT
      1 | 1878 | 1073832133 | QUERY   | INSERT INTO test_table_b values (40);
      1 | 1878 | 1073832133 | UTILITY | COMMIT
(8 rows)
```

此示例演示如何处理游标循环。表 test_table_a 具有三个值。目标是遍历这三个值并将这些值插入表 test_table_b 中。如果通过以下方法创建 NONATOMIC 存储过程，则在第一个循环中执行 INSERT 语句后，它将引发错误：游标“cur1”不存在。这是因为 INSERT 的自动提交会关闭打开的游标。

```
insert into test_table_a values (1), (2), (3);

CREATE OR REPLACE PROCEDURE sp_nonatomic_cursor() NONATOMIC
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  cur1 cursor for select * from test_table_a order by 1;
BEGIN
```

```
  open cur1;
  Loop
    fetch cur1 into rec;
    exit when not found;
    raise info '%', rec.v;
    insert into test_table_b values (rec.v);
  End Loop;
END
$$;

CALL sp_nonatomic_cursor();

INFO:  1
ERROR:  cursor "cur1" does not exist
CONTEXT:  PL/pgSQL function "sp_nonatomic_cursor" line 7 at fetch
```

要使游标循环正常工作，请将其置于 START TRANSACTION...COMMIT 之间。

```
insert into test_table_a values (1), (2), (3);

CREATE OR REPLACE PROCEDURE sp_nonatomic_cursor() NONATOMIC
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  cur1 cursor for select * from test_table_a order by 1;
BEGIN
  START TRANSACTION;
  open cur1;
  Loop
    fetch cur1 into rec;
    exit when not found;
    raise info '%', rec.v;
    insert into test_table_b values (rec.v);
  End Loop;
  COMMIT;
END
$$;

CALL sp_nonatomic_cursor();

INFO:  1
INFO:  2
```

```
INFO:  3
CALL
```

捕获错误

当存储过程中的查询或命令导致错误时，后续查询将停止运行，事务进行回滚。您可以使用 EXCEPTION 块处理错误。

> **Note**
>
> 默认行为是，即使存储过程中没有其他产生错误的情形，错误也会导致后续查询停止运行。

```
[ <<label>> ]
[ DECLARE
  declarations ]
BEGIN
  statements
EXCEPTION
  WHEN OTHERS THEN
    statements
END;
```

当发生异常并添加异常处理块时，您可以编写 REAE 语句和大多数其它 PL/pgSQL 语句。例如，您可以使用自定义消息引发异常，也可以将记录插入日志表中。

进入异常处理块时，将回滚当前事务并创建一个新事务来运行块中的语句。如果块中的语句运行没有错误，则会提交事务并重新抛出异常。最后，退出存储过程。

异常块中唯一支持的条件是 OTHERS，它匹配除了查询取消之外的各个错误类型。此外，如果异常处理块中出现错误，错误可由外部异常处理块捕获。

当 NONATOMIC 过程内部出现错误时，如果错误由异常块处理，则不会重新引发错误。请查看 PL/pgSQL 语句 RAISE，以引发由异常处理块捕获的异常。此语句仅在异常处理块中有效。有关更多信息，请参阅RAISE。

使用 CONTINUE 处理程序控制存储过程出错后的处理情况

CONTINUE 处理程序是一种异常处理程序，用于控制 NONATOMIC 存储过程中的执行流程。通过使用此处理程序，您可以在不结束现有语句块的情况下捕获和处理异常。通常，当存储过程中发生错误时，流程将中断，错误将返回给调用者。不过，在某些使用案例中，错误情况并没有严重到需要中断流程的

地步。您可能希望在一个单独的事务中使用自己选择的错误处理逻辑优雅地处理错误，然后继续运行错误后的语句。语法如下。

```
[ DECLARE
  declarations ]
BEGIN
  statements
EXCEPTION
  [ CONTINUE_HANDLER | EXIT_HANDLER ] WHEN OTHERS THEN
    handler_statements
END;
```

您可以利用多个系统表来帮助收集有关各种类型错误的信息。有关更多信息，请参阅 STL_LOAD_ERRORS、STL_ERROR 和 SYS_STREAM_SCAN_ERRORS。您还可以使用其他系统表来进行故障排除。在系统表和视图参考中可以找到有关这些内容的更多信息。

示例

以下示例演示如何在异常处理块中写入语句。存储过程使用默认事务管理行为。

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

CREATE OR REPLACE PROCEDURE update_employee_sp() AS
$$
BEGIN
    UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
    EXECUTE 'select invalid';
EXCEPTION WHEN OTHERS THEN
    RAISE INFO 'An exception occurred.';
    INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp();

INFO:  An exception occurred.
ERROR:  column "invalid" does not exist
CONTEXT:  SQL statement "select invalid"
PL/pgSQL function "update_employee_sp" line 3 at execute statement
```

在此示例中，如果我们调用 update_employee_sp，则会引发信息性消息 An exception occurred.（发生异常。）并将错误消息插入到日志记录表的 employee_error_log 日志中。在存储过程退出之前再次抛出原始异常。以下查询显示了运行该示例所产生的记录。

```
SELECT * from employee;

firstname | lastname
-----------+----------
 Tomas     | Smith

SELECT * from employee_error_log;

          message
-----------------------------------------------
 Error message: column "invalid" does not exist
```

有关 RAISE 的更多信息，包括格式帮助和附加级别列表，请参阅支持的 PL/pgSQL 语句。

以下示例演示如何在异常处理块中写入语句。存储过程使用 NONATOMIC 事务管理行为。在此示例中，过程调用完成后，不会引发任何返回给调用方的错误。由于下一条语句中的错误，UPDATE 语句无法回滚。此时将引发信息性消息，并将错误消息插入到日志记录表中。

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

-- Create the SP in NONATOMIC mode
CREATE OR REPLACE PROCEDURE update_employee_sp_2() NONATOMIC AS
$$
BEGIN
    UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
    EXECUTE 'select invalid';
EXCEPTION WHEN OTHERS THEN
    RAISE INFO 'An exception occurred.';
    INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp_2();
INFO:  An exception occurred.
CALL
```

```
SELECT * from employee;

 firstname | lastname
-----------+----------
 Adam      | Smith
(1 row)

SELECT * from employee_error_log;

                    message
-----------------------------------------------
 Error message: column "invalid" does not exist
(1 row)
```

此示例演示如何创建具有两个子块的过程。调用存储过程时，第一个子块中错误由其异常处理块处理。第一个子块完成后，过程继续执行第二个子块。您可在结果中看到，过程调用完成时没有引发错误。对表 employee 执行的 UPDATE 和 INSERT 操作已提交。来自两个异常块的错误消息都插入到日志记录表中。

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

CREATE OR REPLACE PROCEDURE update_employee_sp_3() NONATOMIC AS
$$
BEGIN
    BEGIN
        UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
        EXECUTE 'select invalid1';
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'An exception occurred in the first block.';
        INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
    END;
    BEGIN
        INSERT INTO employee VALUES ('Edie','Robertson');
        EXECUTE 'select invalid2';
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'An exception occurred in the second block.';
        INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
    END;
END;
$$
```

```
LANGUAGE plpgsql;

CALL update_employee_sp_3();
INFO:  An exception occurred in the first block.
INFO:  An exception occurred in the second block.
CALL

SELECT * from employee;

 firstname | lastname
-----------+-----------
 Adam      | Smith
 Edie      | Robertson
(2 rows)

SELECT * from employee_error_log;

                     message
-------------------------------------------------
 Error message: column "invalid1" does not exist
 Error message: column "invalid2" does not exist
(2 rows)
```

以下示例演示如何使用 CONTINUINE 异常处理程序。此示例创建了两个表，并在存储过程中使用它们。CONTINUE 处理程序控制具有 NONATOMIC 事务管理行为的存储过程中的执行流程。

```
CREATE TABLE tbl_1 (a int);
CREATE TABLE tbl_error_logging(info varchar, err_state varchar, err_msg varchar);

CREATE OR REPLACE PROCEDURE sp_exc_handling_1() NONATOMIC AS
$$
BEGIN
    INSERT INTO tbl_1 VALUES (1);
    -- Expect an error for the insert statement following, because of the invalid value
    INSERT INTO tbl_1 VALUES ("val");
    INSERT INTO tbl_1 VALUES (2);
EXCEPTION CONTINUE_HANDLER WHEN OTHERS THEN
    INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

调用存储过程：

```
CALL sp_exc_handling_1();
```

流程如下：

1. 由于试图在列中插入不兼容的数据类型，因此出现错误。控制权传递到 EXCEPTION 块。进入异常处理块后，当前事务将回滚，并创建一个新的隐式事务来运行其中的语句。
2. 如果 CONTINUE_HANDLER 中的语句运行无误，控制权将传递到紧接着导致异常的语句之后的语句。（如果 CONTINUE_HANDLER 中的语句引发了新的异常，可以在 EXCEPTION 块中使用异常处理程序来处理。）

调用示例存储过程后，表中会包含以下记录：

- 如果您运行 `SELECT * FROM tbl_1;`，它会返回两条记录。其中包含值 1 和 2。
- 如果您运行 `SELECT * FROM tbl_error_logging;`，它会返回一条包含以下值的记录：Encountered error、42703 和 column "val" does not exist in tbl_1。

下面的附加错误处理示例同时使用 EXIT 处理程序和 CONTINUE 处理程序。它创建两个表：一个数据表和一个日志表。它还会创建一个演示错误处理的存储过程：

```
CREATE TABLE tbl_1 (a int);
CREATE TABLE tbl_error_logging(info varchar, err_state varchar, err_msg varchar);

CREATE OR REPLACE PROCEDURE sp_exc_handling_2() NONATOMIC AS
$$
BEGIN
    INSERT INTO tbl_1 VALUES (1);
    BEGIN
        INSERT INTO tbl_1 VALUES (100);
        -- Expect an error for the insert statement following, because of the invalid
 value
        INSERT INTO tbl_1 VALUES ("val");
        INSERT INTO tbl_1 VALUES (101);
    EXCEPTION EXIT_HANDLER WHEN OTHERS THEN
        INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
    END;
    INSERT INTO tbl_1 VALUES (2);
    -- Expect an error for the insert statement following, because of the invalid value
    INSERT INTO tbl_1 VALUES ("val");
    INSERT INTO tbl_1 VALUES (3);
```

```
EXCEPTION CONTINUE_HANDLER WHEN OTHERS THEN
    INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

创建存储过程后，使用以下内容对其进行调用：

```
CALL sp_exc_handling_2();
```

当内部异常块（由内部的 BEGIN 和 END 所括弧）中出现错误时，将由 EXIT 处理程序进行处理。外部块中发生的任何错误都由 CONTINUE 处理程序处理。

调用示例存储过程后，表中会包含以下记录：

- 如果您运行 `SELECT * FROM tbl_1;`，它会返回四条记录，值分别为 1、2、3 和 100。
- 如果您运行 `SELECT * FROM tbl_error_logging;`，它会返回两条记录。它们有以下值：Encountered error、42703 和 column "val" does not exist in tbl_1。

如果 tbl_error_logging 表不存在，则会引发异常。

以下示例演示如何将 CONTINUE 异常处理程序与 FOR 循环一起使用。此示例创建了三个表，并在存储过程的 FOR 循环中使用它们。FOR 循环是结果集变体，这意味着它会遍历查询结果：

```
CREATE TABLE tbl_1 (a int);
INSERT INTO tbl_1 VALUES (1), (2), (3);
CREATE TABLE tbl_2 (a int);
CREATE TABLE tbl_error_logging(info varchar, err_state varchar, err_msg varchar);

CREATE OR REPLACE PROCEDURE sp_exc_handling_loop() NONATOMIC AS
$$
DECLARE
 rec RECORD;
BEGIN
    FOR rec IN SELECT a FROM tbl_1
    LOOP
        IF rec.a = 2 THEN
            -- Expect an error for the insert statement following, because of the
 invalid value
            INSERT INTO tbl_2 VALUES("val");
        ELSE
            INSERT INTO tbl_2 VALUES (rec.a);
```

```
        END IF;
    END LOOP;
EXCEPTION CONTINUE_HANDLER WHEN OTHERS THEN
    INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

调用存储过程：

```
CALL sp_exc_handling_loop();
```

调用示例存储过程后，表中会包含以下记录：

- 如果您运行 `SELECT * FROM tbl_2;`，它会返回两条记录。其中包含值 1 和 3。
- 如果您运行 `SELECT * FROM tbl_error_logging;`，它会返回一条包含以下值的记录：Encountered error、42703 和 column "val" does not exist in tbl_2。

关于 CONTINUE 处理程序的使用说明：

- CONTINUE_HANDLER 和 EXIT_HANDLER 关键字只能在 NONATOMIC 存储过程中使用。
- CONTINUE_HANDLER 和 EXIT_HANDLER 关键字是可选的。EXIT_HANDLER 是默认值。

记录存储过程

有关存储过程的详细信息记录在以下系统表和视图中：

- SVL_STORED_PROC_CALL – 记录有关存储过程的详细信息，包括调用的开始时间和结束时间，以及调用是否在完成之前结束。有关更多信息，请参阅SVL_STORED_PROC_CALL。
- SVL_STORED_PROC_MESSAGES – 将在相应日志记录级别记录由 RAISE 查询发出的存储过程中的消息。有关更多信息，请参阅SVL_STORED_PROC_MESSAGES。
- SVL_QLOG – 针对从存储过程调用的各个查询，记录过程调用的查询 ID。有关更多信息，请参阅SVL_QLOG。
- STL_UTILITYTEXT – 在每次存储过程调用结束之后记录。有关更多信息，请参阅STL_UTILITYTEXT。
- PG_PROC_INFO – 此系统目录视图显示有关存储过程的信息。有关更多信息，请参阅PG_PROC_INFO。

存储过程支持注意事项

以下注意事项在您使用 Amazon Redshift 存储过程时适用。

Amazon Redshift 与 PostgreSQL 的存储过程支持区别

Amazon Redshift 与 PostgreSQL 中的存储过程支持之间存在下列区别：

- Amazon Redshift 不支持子事务，因此对异常处理数据块的支持有限。

注意事项和限制

以下是 Amazon Redshift 中存储过程的注意事项：

- 数据库的最大存储过程数为 10000 个。
- 过程的源代码最大大小为 2 MB。
- 在一个用户会话中，您可以并发打开的显式和隐式游标的最大数量为 1。对 SQL 语句结果集进行迭代的 FOR 循环打开隐式游标。不支持嵌套游标。
- 显式和隐式游标在结果集大小上与标准 Amazon Redshift 游标具有相同的限制。有关更多信息，请参阅游标约束。
- 嵌套调用的最大层数为 16。
- 对于输入参数，过程参数的最大数量为 16，对于输出参数为 32。
- 存储过程中变量的最大数量为 1024。
- 在存储过程内部，不支持任何需要自己的事务上下文的 SQL 命令。示例包括：
  - PREPARE
  - CREATE/DROP DATABASE
  - CREATE EXTERNAL TABLE
  - VACUUM
  - SET LOCAL
  - ALTER TABLE APPEND
- 对于 `registerOutParameter` 数据类型，不支持通过 Java Database Connectivity (JDBC) 驱动程序调用 `refcursor` 方法。有关使用 `refcursor` 数据类型的示例，请参阅。返回结果集

PL/pgSQL 语言参考

Amazon Redshift 中的存储过程基于 PostgreSQL PL/pgSQL 程序性语言，但有一些非常重要的不同之处。在此参考中，您可以找到 Amazon Redshift 所实施 PL/pgSQL 语法的详细信息。有关 PL/pgSQL 的更多信息，请参阅 PostgreSQL 文档中的 PL/pgSQL - SQL 程序性语言。

主题

PL/pgSQL 参考惯例

在此部分中，您可以找到为 PL/pgSQL 存储过程语言编写语法的惯例。

字符	描述
CAPS	采用大写字母的字样为关键字。
[ ]	方括号表示可选参数。方括号中的多个参数表示您可选择任意数量的参数。此外，不同行的括号中的参数表示 Amazon Redshift 分析程序需要按照参数在语法中列出的顺序获取参数。
{ }	大括号表示您需要选择大括号内的参数之一。
|	管道表示您可以在不同参数之间选择。
####	红色斜体字样表示占位符。必须插入适当的值以替换红色斜体字样。
. . .	省略号表示可以重复前面的元素。
'	单引号中的字样表示必须键入引号。

PL/pgSQL 的结构

PL/pgSQL 是一种程序性语言，具有与其他程序性语言相同的许多结构。

主题

数据块

PL/pgSQL 是块结构语言。过程的整个正文在块中定义，其中包含变量声明和 PL/pgSQL 语句。语句还可以包含嵌套块，也就是子块。

使用分号作为声明和语句的结尾。在块或子块的 END 关键字之后添加分号。请勿在关键字 DECLARE 和 BEGIN 之后使用分号。

所有关键字和标识符均可以使用混合大小写来编写。除非括在双引号中，否则标识符隐式转换为小写。

双连字符 (--) 表示注释的开始，一直到这一行的结尾。/* 表示块注释的开始，一直到下一个 */ 为止。块注释不能嵌套。但是，您可以在块注释中包括双连字符注释，而双连字符可以隐藏块注释分隔符 /* 和 */。

在一个块中，语句部分中任何语句都可以是子块。您可以使用子块来逻辑分组，或者将变量本地化到一小组语句中。

```
[ <<label>> ]
[ DECLARE
  declarations ]
BEGIN
  statements
END [ label ];
```

在块之前的声明部分中声明的变量，每次进入块时将初始化为其默认值。换而言之，它们不是在每次函数调用中仅初始化一次。

下面是一个示例。

```
CREATE PROCEDURE update_value() AS $$
DECLARE
  value integer := 20;
BEGIN
```

```
  RAISE NOTICE 'Value here is %', value;  -- Value here is 20
  value := 50;
  --
  -- Create a subblock
  --
  DECLARE
    value integer := 80;
  BEGIN
    RAISE NOTICE 'Value here is %', value;  -- Value here is 80
  END;

  RAISE NOTICE 'Value here is %', value;  -- Value here is 50
END;
$$ LANGUAGE plpgsql;
```

使用标签来标识在 EXIT 语句中使用的块，或者用于限定在块中声明的变量的名称。

请不要将为 PL/pgSQL 中用于分组语句的 BEGIN/END，与用于事务控制的数据库命令相混淆。PL/pgSQL 中的 BEGIN 和 END 仅用于分组。它们不启动或结束事务。

变量声明

在块中声明所有变量，但循环变量除外，此类变量应在块的 DECLARE 部分中声明。变量可使用任意有效的 Amazon Redshift 数据类型。有关支持的数据类型，请参阅数据类型。

PL/pgSQL 变量可以为任意 Amazon Redshift 支持的数据类型，以及 RECORD 和 `refcursor`。有关 RECORD 的更多信息，请参阅 记录类型。有关 `refcursor` 的更多信息，请参阅 游标。

```
DECLARE
name [ CONSTANT ] type [ NOT NULL ] [ { DEFAULT | := } expression ];
```

在下文中您可以找到示例变量声明。

```
customerID integer;
numberofitems numeric(6);
link varchar;
onerow RECORD;
```

对于迭代某个整数范围的 FOR 循环，其循环变量自动声明为整数变量。

如果给出了 DEFAULT 子句，则指定在进入块时赋给变量的初始值。如果未给出 DEFAULT 子句，则变量初始化为 SQL NULL 值。CONSTANT 选项防止向变量赋值，因此其值在块的持续过程中保持为

常数。如果指定了 NOT NULL，则赋值 null 时会导致运行时错误。声明为 NOT NULL 的所有变量必须指定非 null 默认值。

在每次进入块时，对默认值求值。例如，将 now() 分配给类型为 timestamp 的变量会使变量具有当前函数调用的时间，而不是预编译函数时的时间。

```
quantity INTEGER DEFAULT 32;
url VARCHAR := 'http://mysite.com';
user_id CONSTANT INTEGER := 10;
```

refcursor 数据类型是存储过程中游标变量的数据类型。refcursor 值可从存储过程中返回。有关更多信息，请参阅 返回结果集。

别名声明

如果存储过程的签名省略了参数名称，您可以为参数声明别名。

```
name ALIAS FOR $n;
```

内置变量

支持以下内置变量：

- FOUND
- SQLSTATE
- SQLERRM
- GET DIAGNOSTICS integer_var := ROW_COUNT;

FOUND 是一种布尔值类型的特殊变量。FOUND 在每个过程调用中开始为 false。以下类型的语句设置 FOUND：

- SELECT INTO

  如果返回一行，则将 FOUND 设置为 true，未返回行时设置为 false。
- UPDATE、INSERT 和 DELETE

  如果至少影响到一行，则将 FOUND 设置为 true，未影响任何行时设置为 false。

- FETCH

  如果返回一行，则将 FOUND 设置为 true，未返回行时设置为 false。

- FOR 语句

  如果 FOR 语句迭代了一次或多次，则将 FOUND 设置为 true，否则为 false。这适用于 FOR 语句的全部三种变体：整数 FOR 循环、记录集 FOR 循环和动态记录集 FOR 循环。

  FOUND 在 FOR 循环退出时设置。在循环的运行时中，FOR 语句不修改 FOUND。不过，可以通过在循环正文中运行其他语句来更改。

下面是一个示例。

```
CREATE TABLE employee(empname varchar);
CREATE OR REPLACE PROCEDURE show_found()
AS  $$
DECLARE
  myrec record;
BEGIN
  SELECT INTO myrec * FROM employee WHERE empname = 'John';
  IF NOT FOUND THEN
    RAISE EXCEPTION 'employee John not found';
  END IF;
END;
$$ LANGUAGE plpgsql;
```

在异常处理程序中，特殊变量 SQLSTATE 包含对应于所引发异常的错误代码。特殊变量 SQLERRM 包含与异常关联的错误消息。这些变量未在异常处理程序之外定义，如果使用会显示错误。

下面是一个示例。

```
CREATE OR REPLACE PROCEDURE sqlstate_sqlerrm() AS
$$
BEGIN
  UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
  EXECUTE 'select invalid';
  EXCEPTION WHEN OTHERS THEN
  RAISE INFO 'error message SQLERRM %', SQLERRM;
  RAISE INFO 'error message SQLSTATE %', SQLSTATE;
END;
$$ LANGUAGE plpgsql;
```

ROW_COUNT 与 GET DIAGNOSTICS 命令结合使用。它显示向下发送至 SQL 引擎的上一个 SQL 命令处理的行数。

下面是一个示例。

```
CREATE OR REPLACE PROCEDURE sp_row_count() AS
$$
DECLARE
  integer_var int;
BEGIN
  INSERT INTO tbl_row_count VALUES(1);
  GET DIAGNOSTICS integer_var := ROW_COUNT;
  RAISE INFO 'rows inserted = %', integer_var;
END;
$$ LANGUAGE plpgsql;
```

记录类型

RECORD 类型不是真实的数据类型，只是一个占位符。记录类型变量代入在 SELECT 或 FOR 命令期间向其分配的行的实际行结构。记录变量的子结构可在每次分配值时更改。在首次分配值之前，记录变量没有子结构。任何访问其中字段的尝试将引发运行时错误。

```
name RECORD;
```

下面是一个示例。

```
CREATE TABLE tbl_record(a int, b int);
INSERT INTO tbl_record VALUES(1, 2);
CREATE OR REPLACE PROCEDURE record_example()
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
BEGIN
  FOR rec IN SELECT a FROM tbl_record
  LOOP
    RAISE INFO 'a = %', rec.a;
  END LOOP;
END;
$$;
```

支持的 PL/pgSQL 语句

PL/pgSQL 语句使用程序性结构（包括循环和条件表达式）增强 SQL 命令，以控制逻辑流。大部分 SQL 命令都可以使用，包括数据操作语言（DML，Data Manipulation Language）（例如 COPY、UNLOAD 和 INSERT）以及数据定义语言（DDL，Data Definition Language）（例如 CREATE TABLE）。有关完整的 SQL 命令列表，请参阅 SQL 命令。此外，Amazon Redshift 支持以下 PL/pgSQL 语句。

主题

赋值

赋值语句将值赋予变量。表达式必须返回单个值。

```
identifier := expression;
```

使用非标准 = 而不是 := 进行赋值也是可以接受的。

如果表达式的数据类型与变量的数据类型不匹配，或者变量具有大小或精度要求，则将隐式转换结果值。

下面是一个示例。

```
customer_number := 20;
```

```
tip := subtotal * 0.15;
```

SELECT INTO

SELECT INTO 语句将多列（但仅限一行）的结果赋予一个记录变量或赋予标量变量列表。

```
SELECT INTO target select_expressions FROM ...;
```

在前一种语法中，*target* 可以是记录变量，或者是简单变量和记录字段的逗号分隔列表。*select_expressions* 列表和命令的剩余部分与常规 SQL 中相同。

如果将变量列表用作 *target*，则所选值必须与目标的结构完全匹配，否则会出现运行时错误。当记录变量是目标时，它自动将自身配置为查询结果列的行类型。

INTO 子句可以在 SELECT 语句中的几乎任何位置出现。它通常出现在紧挨着 SELECT 子句的后面，或者正好在 FROM 子句之前。即，它显示在 *select_expressions* 列表紧挨着的前面或后面。

如果查询返回零行，则将 NULL 值赋予 *target*。如果查询返回多行，则第一行赋予 *target*，丢弃剩余的行。除非语句包含 ORDER BY，否则第一行不是确定性的。

要确定赋值是否至少返回了一行，请使用特殊 FOUND 变量。

```
SELECT INTO customer_rec * FROM cust WHERE custname = lname;
IF NOT FOUND THEN
  RAISE EXCEPTION 'employee % not found', lname;
END IF;
```

要测试某个记录结果是否为 null，您可以使用有条件的 IS NULL。没有方法确定是否已经丢弃了任何其他行。以下示例处理未返回任何行的情况。

```
CREATE OR REPLACE PROCEDURE select_into_null(return_webpage OUT varchar(256))
AS $$
DECLARE
  customer_rec RECORD;
BEGIN
  SELECT INTO customer_rec * FROM users WHERE user_id=3;
  IF customer_rec.webpage IS NULL THEN
    -- user entered no webpage, return "http://"
    return_webpage = 'http://';
  END IF;
END;
```

```
$$ LANGUAGE plpgsql;
```

无操作

无操作语句 (NULL;) 是不执行任何操作的占位符语句。无操作语句可指示 IF-THEN-ELSE 链的一个分支为空。

```
NULL;
```

动态 SQL

对于在每次从 PL/pgSQL 存储过程运行时会涉及到不同表或不同数据类型的动态命令，可以使用 EXECUTE 语句来生成此类命令。

```
EXECUTE command-string [ INTO target ];
```

在前面的内容中，*command-string* 是生成字符串（文本类型）的表达式，其中包含要运行的命令。*command-string* 值发送到 SQL 引擎。在命令字符串上不进行 PL/pgSQL 变量替换。变量的值必须在命令字符串构造时插入。

> **Note**
>
> 您不能在动态 SQL 中使用 COMMIT 和 ROLLBACK 语句。有关在存储过程中使用 COMMIT 和 ROLLBACK 语句的信息，请参阅管理事务。

使用动态命令时，您通常必须处理单引号的转义。建议您在函数正文中使用美元符号来括起引号中的固定文本。插入结构化查询中的动态值需要特殊处理，因为它们本身可能包含引号。以下示例假定用美元符号整体括起了函数，因此无需两个引号。

```
EXECUTE 'UPDATE tbl SET '
  || quote_ident(colname)
  || ' = '
  || quote_literal(newvalue)
  || ' WHERE key = '
  || quote_literal(keyvalue);
```

前面的示例显示了函数 quote_ident(text) 和 quote_literal(text)。此示例将包含列和表标识符的变量传递到 quote_ident 函数。它还可以在结构化命令中，将包含文本字符串的变量传递

到 quote_literal 函数。这两个函数均采取相应的步骤，分别返回括在双引号或单引号中的输入文本，其中任何嵌入的特殊字符均正确转义。

美元符号括起仅对与括起固定文本有用。请不要按照以下格式编写前述示例。

```
EXECUTE 'UPDATE tbl SET '
  || quote_ident(colname)
  || ' = $$'
  || newvalue
  || '$$ WHERE key = '
  || quote_literal(keyvalue);
```

不能这样做的原因是，如果 newvalue 的内容偶然包含了 $$，示例会中断。您可能选择的任何其他美元符号括起分隔符都会出现这种问题。要安全地括起事先不知道内容的文本，请使用 quote_literal 函数。

Return

RETURN 语句从存储过程返回调用方。

```
RETURN;
```

下面是一个示例。

```
CREATE OR REPLACE PROCEDURE return_example(a int)
AS $$
BEGIN
  FOR b in 1..10 LOOP
    IF b < a THEN
      RAISE INFO 'b = %', b;
    ELSE
      RETURN;
    END IF;
  END LOOP;
END;
$$ LANGUAGE plpgsql;
```

条件：IF

在 Amazon Redshift 使用的 PL/pgSQL 语言中，IF 条件语句可以采用以下格式：

- IF ... THEN

```
IF boolean-expression THEN
  statements
END IF;
```

下面是一个示例。

```
IF v_user_id <> 0 THEN
  UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

- IF ... THEN ... ELSE

```
IF boolean-expression THEN
  statements
ELSE
  statements
END IF;
```

下面是一个示例。

```
IF parentid IS NULL OR parentid = ''
THEN
  return_name = fullname;
  RETURN;
ELSE
  return_name = hp_true_filename(parentid) || '/' || fullname;
  RETURN;
END IF;
```

- IF ... THEN ... ELSIF ... THEN ... ELSE

关键词 ELSIF 也可以拼写成 ELSEIF。

```
IF boolean-expression THEN
  statements
[ ELSIF boolean-expression THEN
  statements
[ ELSIF boolean-expression THEN
  statements
    ...] ]
[ ELSE
```

```
  statements ]
END IF;
```

下面是一个示例。

```
IF number = 0 THEN
  result := 'zero';
ELSIF number > 0 THEN
  result := 'positive';
ELSIF number < 0 THEN
  result := 'negative';
ELSE
  -- the only other possibility is that number is null
  result := 'NULL';
END IF;
```

条件：CASE

在 Amazon Redshift 使用的 PL/pgSQL 语言中，CASE 条件语句可以采用以下格式：

- 简单 CASE

```
CASE search-expression
WHEN expression [, expression [ ... ]] THEN
  statements
[ WHEN expression [, expression [ ... ]] THEN
  statements
  ... ]
[ ELSE
  statements ]
END CASE;
```

简单 CASE 语句根据操作数是否相等提供有条件的执行。

search-expression 值求值一次，接下来与 WHEN 子句中的各个 *expression* 进行比较。如果找到匹配，则对应的 *statements* 运行，然后控制传递到 END CASE 之后的下一个语句。不对后面的 WHEN 表达式求值。如果未找到匹配，则运行 ELSE *statements*。但是，如果没有 ELSE，则将引发 CASE_NOT_FOUND 异常。

下面是一个示例。

```
CASE x
WHEN 1, 2 THEN
  msg := 'one or two';
ELSE
  msg := 'other value than one or two';
END CASE;
```

- 搜索 CASE

```
CASE
WHEN boolean-expression THEN
  statements
[ WHEN boolean-expression THEN
  statements
  ... ]
[ ELSE
  statements ]
END CASE;
```

CASE 的搜索形式提供有条件执行，基于布尔值表达式求值为对还是错。

每个 WHEN 子句的 *boolean-expression* 按顺序求值，直至发现得到 true 的子句。然后，对应的语句运行，然后控制传递到 END CASE 之后的下一个语句。不对后面的 *WHEN* 表达式求值。如果没有找到 true 结果，则运行 ELSE *statements*。但是，如果没有 ELSE，则将引发 CASE_NOT_FOUND 异常。

下面是一个示例。

```
CASE
WHEN x BETWEEN 0 AND 10 THEN
  msg := 'value is between zero and ten';
WHEN x BETWEEN 11 AND 20 THEN
  msg := 'value is between eleven and twenty';
END CASE;
```

Loops

在 Amazon Redshift 使用的 PL/pgSQL 语言中，循环语句可以采用以下格式：

- 简单循环

```
[<<label>>]
LOOP
  statements
END LOOP [ label ];
```

简单循环定义无条件的循环，这种循环将无限重复，直至由 EXIT 或 RETURN 语句终止。在嵌套循环中，EXIT 和 CONTINUE 可以使用可选的标签来指定 EXIT 和 CONTINUE 语句引用哪个循环。

下面是一个示例。

```
CREATE OR REPLACE PROCEDURE simple_loop()
LANGUAGE plpgsql
AS $$
BEGIN
  <<simple_while>>
  LOOP
    RAISE INFO 'I am raised once';
    EXIT simple_while;
    RAISE INFO 'I am not raised';
  END LOOP;
  RAISE INFO 'I am raised once as well';
END;
$$;
```

- 退出循环

```
EXIT [ label ] [ WHEN expression ];
```

如果没有 *label*，则终止最内部的循环，接下来运行 END LOOP 后面的语句。如果有 *label*，则必须是当前或某个更靠外级别的嵌套循环或块的标签。然后，指定的循环或块终止，继续由循环或块对应的 END 之后的语句控制。

如果指定了 WHEN，则循环仅在 *expression* 为 true 时退出。否则，控制将传递到 EXIT 之后的语句。

您可以对所有类型的循环使用 EXIT，它不限于仅由无条件循环使用。

用于 BEGIN 块时，EXIT 将控制传递到块结束之后的语句。为此必须使用标签。在匹配 BEGIN 块时，永远不考虑无标签的 EXIT。

下面是一个示例。

```
CREATE OR REPLACE PROCEDURE simple_loop_when(x int)
LANGUAGE plpgsql
AS $$
DECLARE i INTEGER := 0;
BEGIN
  <<simple_loop_when>>
  LOOP
    RAISE INFO 'i %', i;
    i := i + 1;
    EXIT simple_loop_when WHEN (i >= x);
  END LOOP;
END;
$$;
```

- 继续循环

```
CONTINUE [ label ] [ WHEN expression ];
```

如果未给出 *label*，则执行跳转到最内部循环的下一次迭代。即，跳过循环正文中剩余的所有语句。然后，控制返回到循环控制表达式（如果有），确定是否需要下一次循环迭代。如果存在 *label*，它指定将继续其执行的循环的标签。

如果指定了 WHEN，则循环的下一次迭代仅在 *expression* 为 true 时开始。否则，控制将传递到 CONTINUE 之后的语句。

您可以对所有类型的循环使用 CONTINUE，它不限于仅由无条件循环使用。

```
CONTINUE mylabel;
```

- WHILE 循环

```
[<<label>>]
WHILE expression LOOP
  statements
END LOOP [ label ];
```

只要 *boolean-expression* 求值为 true，WHILE 语句就会重复一组语句。在每个条目即将进入循环正文之前对表达式进行检查。

下面是一个示例。

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP
  -- some computations here
END LOOP;

WHILE NOT done LOOP
  -- some computations here
END LOOP;
```

- FOR 循环（整数变体）

```
[<<label>>]
FOR name IN [ REVERSE ] expression .. expression LOOP
  statements
END LOOP [ label ];
```

FOR 循环（整数变体）创建一个循环，对一系列整数值进行迭代。变量名称自动定义为整数类型，并且仅在循环中存在。忽略循环中任何现有的变量名称定义。给出了范围下限和上限的两个表达式，在进入循环时求值一次。如果您指定 REVERSE，则在每次迭代之后减去步进值而不是累加。

如果下限大于上限（在 REVERSE 情况下则为小于），循环正文不运行。不引发错误。

如果将标签附加到 FOR 循环，则您可以使用限定名称，通过该标签应用整数循环变量。

下面是一个示例。

```
FOR i IN 1..10 LOOP
  -- i will take on the values 1,2,3,4,5,6,7,8,9,10 within the loop
END LOOP;

FOR i IN REVERSE 10..1 LOOP
  -- i will take on the values 10,9,8,7,6,5,4,3,2,1 within the loop
END LOOP;
```

- FOR 循环（结果集变体）

```
[<<label>>]
FOR target IN query LOOP
  statements
END LOOP [ label ];
```

target 是一个记录变量，或者是标量变量的逗号分隔列表。目标依次分配给查询生成的每一行，并为每一行运行循环正文。

FOR 循环（结果集变体）使得存储过程可以迭代查询的结果并相应操作数据。

下面是一个示例。

```
CREATE PROCEDURE cs_refresh_reports() AS $$
DECLARE
  reports RECORD;
BEGIN
  FOR reports IN SELECT * FROM cs_reports ORDER BY sort_key LOOP
    -- Now "reports" has one record from cs_reports
    EXECUTE 'INSERT INTO ' || quote_ident(reports.report_name) || ' ' ||
 reports.report_query;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE plpgsql;
```

- 具有动态 SQL 的 FOR 循环

```
[<<label>>]
FOR record_or_row IN EXECUTE text_expression LOOP
  statements
END LOOP;
```

具有动态 SQL 的 FOR 循环使得存储过程可以迭代动态查询的结果并相应操作数据。

下面是一个示例。

```
CREATE OR REPLACE PROCEDURE for_loop_dynamic_sql(x int)
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  query text;
BEGIN
  query := 'SELECT * FROM tbl_dynamic_sql LIMIT ' || x;
  FOR rec IN EXECUTE query
  LOOP
```

```
    RAISE INFO 'a %', rec.a;
  END LOOP;
END;
$$;
```

游标

您可以不必一次运行整个查询，而是设置游标。游标 封装一个查询，并一次读取几行查询结果。这样做的原因之一是避免在结果包含大量行时过多占用内存。另一个原因是返回对存储过程所创建游标的引用，这使得调用函数可以读取行。此方法提供了从存储过程返回大量行集的高效方式。

要在 NONATOMIC 过程中使用游标，请将游标循环置于 START TRANSACTION...COMMIT 之间。

要设置游标，请先声明游标变量。在 PL/pgSQL 中，对游标的所有访问都通过游标变量进行，该变量始终是特殊数据类型 `refcursor`。`refcursor` 数据类型仅仅存放对游标的引用。

您可以通过声明类型为 `refcursor` 的变量来创建游标变量。或者，您可使用以下游标声明语法。

```
name CURSOR [ ( arguments ) ] FOR query ;
```

在前文中，*arguments*（如果指定）是 *name datatype* 对的逗号分隔列表，每一对定义一个由 *query* 中参数值替换的名称。用于替换这些名称的实际值稍后在打开游标时指定。

下面是一个示例。

```
DECLARE
  curs1 refcursor;
  curs2 CURSOR FOR SELECT * FROM tenk1;
  curs3 CURSOR (key integer) IS SELECT * FROM tenk1 WHERE unique1 = key;
```

这三个变量均具有数据类型 `refcursor`，但第一个可用于任意查询。相反，第二个变量绑定了已经充分指定的查询，最后一个绑定了参数化查询。`key` 值由打开游标时的整数参数值替换。变量 `curs1` 称为未绑定 是因为它未绑定到任何特殊查询。

您必须先打开它，然后才能使用游标检索行。PL/pgSQL 具有三种形式的 OPEN 语句，其中两个使用未绑定游标变量，第三个语句使用绑定游标变量：

- 打开供选择：游标变量已打开，并提供了指定的查询来运行。该游标不能已经打开。此外，它必须已经声明作为未绑定游标（即，作为简单 refcursor 变量）。SELECT 查询的处理方式与 PL/pgSQL 中的 SELECT 语句相同。

```
OPEN cursor_name FOR SELECT ...;
```

下面是一个示例。

```
OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;
```

- 打开供执行：游标变量已打开，并提供了指定的查询来运行。该游标不能已经打开。此外，它必须已经声明作为未绑定游标（即，作为简单 refcursor 变量）。以字符串表达式格式指定查询的方式，与在 EXECUTE 命令中的方式相同。此方法提供了灵活性，使得查询可以在不同运行之间变化。

```
OPEN cursor_name FOR EXECUTE query_string;
```

下面是一个示例。

```
OPEN curs1 FOR EXECUTE 'SELECT * FROM ' || quote_ident($1);
```

- 打开绑定游标：这种类型的 OPEN 用于打开的游标变量，在声明时已经将查询绑定到其上。该游标不能已经打开。只有在声明了游标来获取参数时，才必须显示实际参数值表达式的列表。这些值在查询中提交。

```
OPEN bound_cursor_name [ ( argument_values ) ];
```

下面是一个示例。

```
OPEN curs2;
OPEN curs3(42);
```

打开某个游标之后，您可以使用下述语句来处理它。这些语句不必出现在打开游标的同一个存储过程中。您可将 refcursor 值返回到存储过程之外，让调用函数来处理游标。所有门户在事务结束之后隐式关闭。因此，您只能在事务结束后，使用 refcursor 值来引用打开游标。

- FETCH 从游标将下一行检索到目标中。目标可以是行变量、记录变量，或者是简单变量的逗号分隔列表，就像与 SELECT INTO 一样。对于 SELECT INTO，您可以检查特殊变量 FOUND 以查看是否获取了某行。

  ```
  FETCH cursor INTO target;
  ```

  下面是一个示例。

  ```
  FETCH curs1 INTO rowvar;
  ```

- CLOSE 将关闭打开游标下的门户。您可以使用此语句，在事务结束之前释放资源。您也可以使用此语句再次释放游标成为打开状态。

  ```
  CLOSE cursor;
  ```

  下面是一个示例。

  ```
  CLOSE curs1;
  ```

RAISE

使用 RAISE level 语句报告消息和引发错误。

```
RAISE level 'format' [, variable [, ...]];
```

可能的级别包括 NOTICE、INFO、LOG、WARNING 和 EXCEPTION。EXCEPTION 引发错误，这通常会取消当前事务。其他级别仅生成不同优先级的消息。

在格式字符串中，% 替换为下一个可选参数的字符串表示形式。写作 %% 格式将发出文本 %。当前，可选参数必须为简单变量而不是表达式，格式必须是简单字符串文本。

在以下示例中，v_job_id 的值替换字符串中的 %。

```
RAISE NOTICE 'Calling cs_create_job(%)', v_job_id;
```

使用 RAISE 语句重新引发由异常处理块捕获的异常。此语句仅在 NONATOMIC 模式存储过程的异常处理块中有效。

```
RAISE;
```

事务控制

您可以处理 Amazon Redshift 所用 PL/pgSQL 语言中的事务控制语句。有关在存储过程中使用 COMMIT、ROLLBACK 和 TRUNCATE 的信息，请参阅管理事务。

在 NONATOMIC 模式存储过程中，使用 START TRANSACTION 来启动事务块。

```
START TRANSACTION;
```

> **Note**
>
> PL/pgSQL 语句 START TRANSACTION 与 SQL 命令 START TRANSACTION 有以下不同：
>
> - 在存储过程中，START TRANSATION 不等同于 BEGIN。
> - PL/pgSQL 语句不支持可选的隔离级别和访问权限关键字。

在 Amazon Redshift 中创建实体化视图

在数据仓库环境中，应用程序经常必须要对大型表执行复杂的查询。例如 SELECT 语句，此类语句对包含数十亿行的表执行多表联接和聚合。就系统资源和计算结果所需的时间而言，处理这些查询的成本可能会很高。

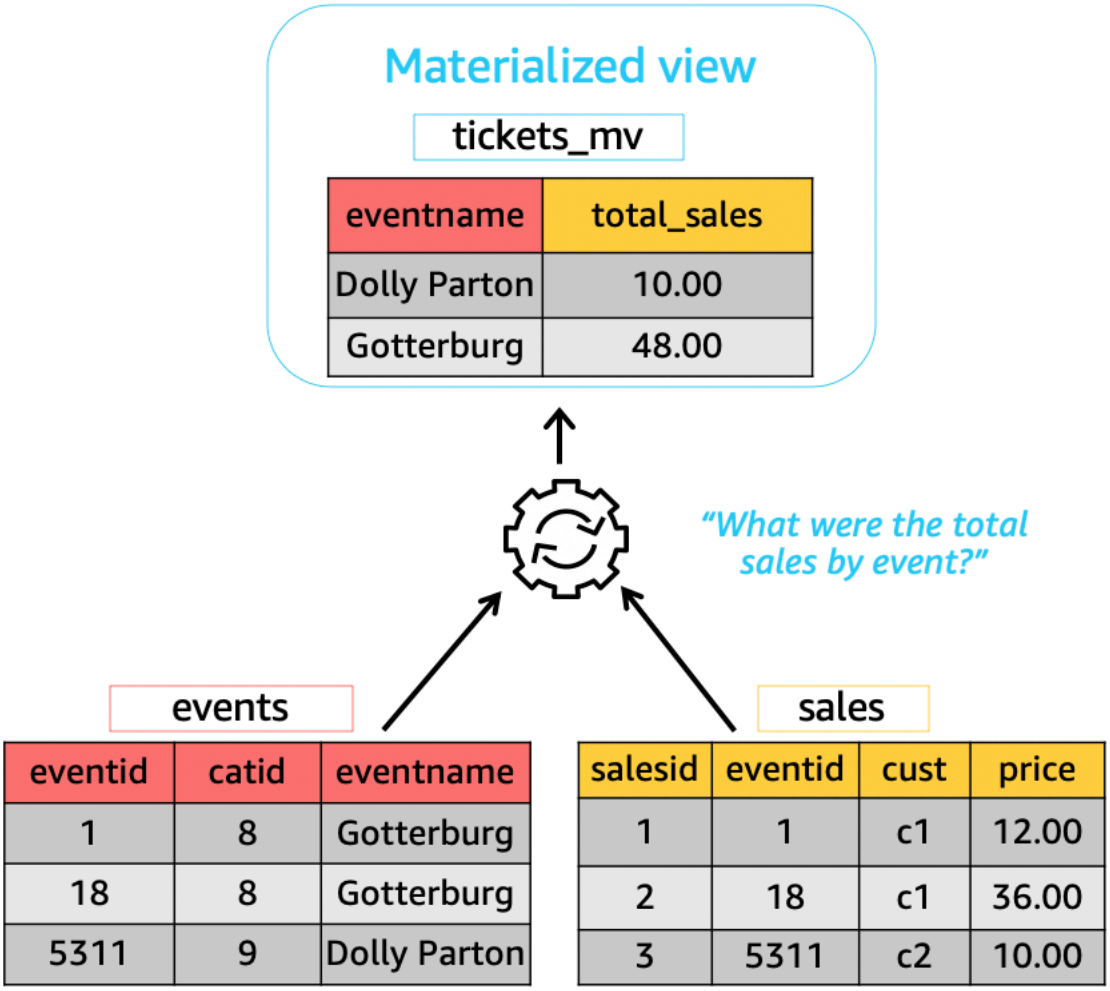
Amazon Redshift 中的实体化视图提供了解决这些问题的方法。实体化视图 包含一个预计算的结果集，该结果集基于对一个或多个基表进行的 SQL 查询。您可以发出 SELECT 语句来查询实体化视图，这与查询数据库中的其他表或视图的方式相同。Amazon Redshift 从实体化视图返回预先计算的结果，根本不必访问基表。从用户的角度来看，与从基表中检索相同的数据相比，查询结果的返回速度要快得多。

实体化视图对于加快可预测和重复的查询特别有用。应用程序可以查询实体化视图并检索预先计算的结果集，而不是对大型表（例如聚合或多联接）执行资源密集型查询。例如，考虑使用一组查询填充控制面板的情况，例如 Amazon QuickSight。该使用案例非常适合实体化视图，因为查询是可预测的，并且会反复重复。

您可以根据其他实体化视图定义实体化视图。使用实体化视图上的实体化视图来扩展实体化视图的功能。通过此方法，现有实体化视图与查询检索数据的基表扮演的角色相同。

此方法对于重复使用不同的聚合或 GROUP BY 选项的预计算联接特别有用。例如，使用实体化视图，该视图将客户信息（包含数百万行）与项目订单详细信息（包含数十亿行）联接起来。该查询费用高昂，可以按需重复计算。对于在此实体化视图之上创建的实体化视图，可以使用不同的 GROUP BY 选项，并且可以与其他表联接。这样做可以节省每次运行昂贵的底层联接所需的计算时间。STV_MV_DEPS表显示了一个实体化视图与其他实体化视图的依赖关系。

在您创建实体化视图时，Amazon Redshift 会运行用户指定的 SQL 语句以从一个或多个基表中收集数据并存储结果集。下图概述了 SQL 查询使用两个基表 `events` 和 `sales` 定义的实体化视图 `tickets_mv`。



然后，您可以在查询中使用这些实体化视图来加快它们的速度。此外，Amazon Redshift 可以自动重写这些查询以使用实体化视图，即使查询没有显式引用实体化视图也是如此。当您无法将查询更改为使用实体化视图时，自动重写查询在增强性能方面特别强大。

要更新实体化视图中的数据，您可以随时使用 REFRESH MATERIALIZED VIEW 语句手动刷新实体化视图。Amazon Redshift 会标识已在一个或多个基表中进行的更改，然后将这些更改应用于实体化视图。由于查询的自动重写要求实体化视图是最新的，因此作为实体化视图的拥有者，请确保在基表发生更改时刷新实体化视图。

Amazon Redshift 提供了几种方法来保持实体化视图的最新状态，以便自动重写。您可以使用自动刷新选项配置实体化视图，以便在更新实体化视图的基表时刷新实体化视图。此自动刷新操作在集群资源可用时运行，以最大限度地减少对其他工作负载的中断。由于自动刷新的计划取决于工作负载，因此您可以更好地控制 Amazon Redshift 刷新实体化视图的时间。您可以使用 Amazon Redshift 调度程序 API 和控制台集成来计划实体化视图刷新任务。有关查询调度的更多信息，请参阅在 Amazon Redshift 控制台上调度查询。

当实体化视图中的最新数据需要服务等级协议 (SLA) 时，执行此操作尤其有用。您还可以手动刷新可以自动刷新的任何实体化视图。有关如何创建实体化视图的信息，请参阅CREATE MATERIALIZED VIEW。

您可以发出 SELECT 语句来查询实体化视图。有关如何查询实体化视图的信息，请参阅查询实体化视图。在基表中插入、更新和删除数据时，结果集最终会变得过时。您可以随时刷新实体化视图，以使用基表中的最新更改对其进行更新。有关如何刷新实体化视图的信息，请参阅REFRESH MATERIALIZED VIEW。

有关用于创建和管理实体化视图的 SQL 命令的详细信息，请参阅以下命令主题：

- CREATE MATERIALIZED VIEW
- ALTER MATERIALIZED VIEW
- REFRESH MATERIALIZED VIEW
- DROP MATERIALIZED VIEW

有关用于监视实体化视图的系统表和视图的信息，请参阅以下主题：

- STV_MV_INFO
- STL_MV_STATE
- SVL_MV_REFRESH_STATUS
- STV_MV_DEPS

主题

- 查询实体化视图
- 自动查询重写以使用实体化视图
- 刷新实体化视图
- 自动实体化视图
- 在实体化视图中使用用户定义的函数 (UDF)
- 串流摄取

查询实体化视图

可以通过将实体化视图名称引用为数据来源（例如表或标准视图）来在任何 SQL 查询中使用实体化视图。

当查询访问实体化视图时，它只会看到最近一次刷新时存储在实体化视图中的数据。因此，查询可能不会从实体化视图的相应基表中看到所有最新更改。

如果其他用户需要查询实体化视图，实体化视图的拥有者必须向这些用户授予 SELECT 权限。其他用户不需要对基础基表具有 SELECT 权限。实体化视图的拥有者还可以撤消其他用户的 SELECT 权限，以防止他们查询实体化视图。

如果实体化视图的拥有者不再对基础基表具有 SELECT 权限，则：

- 拥有者无法再查询实体化视图。
- 对实体化视图具有 SELECT 权限的其他用户无法再查询实体化视图。

以下示例查询 `tickets_mv` 实体化视图。有关用于创建实体化视图的 SQL 命令的更多信息，请参阅CREATE MATERIALIZED VIEW。

```
SELECT sold
FROM tickets_mv
WHERE catgroup = 'Concerts';
```

由于查询结果是预先计算的，因此无需访问基础表（`category`、`event` 和 `sales`）。Amazon Redshift 可以直接从`tickets_mv`返回结果。

自动查询重写以使用实体化视图

您可以在 Amazon Redshift 中使用实体化视图的自动查询重写，让 Amazon Redshift 重写查询以使用实体化视图。这样做可加速查询工作负载，即使对于没有显式引用实体化视图的查询也是如此。当 Amazon Redshift 重写查询时，它只使用最新的实体化视图。

使用说明

要检查查询是否使用自动重写查询，可以检查查询计划或 STL_EXPLAIN。下面显示了 SELECT 语句和原始查询计划的 EXPLAIN 输出。

```
SELECT catgroup, SUM(qtysold) AS sold
FROM category c, event e, sales s
WHERE c.catid = e.catid AND e.eventid = s.eventid
GROUP BY 1;

EXPLAIN
```

```
XN HashAggregate  (cost=920021.24..920021.24 rows=1 width=35)
  ->  XN Hash Join DS_BCAST_INNER  (cost=440004.53..920021.22 rows=4 width=35)
        Hash Cond: ("outer".eventid = "inner".eventid)
        ->  XN Seq Scan on sales s  (cost=0.00..7.40 rows=740 width=6)
        ->  XN Hash  (cost=440004.52..440004.52 rows=1 width=37)
              ->  XN Hash Join DS_BCAST_INNER  (cost=0.01..440004.52 rows=1 width=37)
                    Hash Cond: ("outer".catid = "inner".catid)
                    ->  XN Seq Scan on event e  (cost=0.00..2.00 rows=200 width=6)
                    ->  XN Hash  (cost=0.01..0.01 rows=1 width=35)
                          ->  XN Seq Scan on category c  (cost=0.00..0.01 rows=1
width=35)
```

下面显示了成功自动重写后的 EXPLAIN 输出。此输出包括对查询计划中的实体化视图的扫描，该视图将替换原始查询计划的部分。

```
* EXPLAIN
     XN HashAggregate  (cost=11.85..12.35 rows=200 width=41)
       ->  XN Seq Scan on mv_tbl__tickets_mv__0 derived_table1  (cost=0.00..7.90
rows=790 width=41)
```

只有最新的（新的）实体化视图才会被考虑用于查询的自动重写，而不考虑刷新策略（如自动、计划或手动）。因此，原始查询将返回最新结果。当在查询中显式引用实体化视图时，Amazon Redshift 会访问当前在实体化视图中存储的数据。此数据可能不会反映实体化视图基表中的最新更改。

您可以对集群版本 1.0.20949 或更高版本创建的实体化视图使用自动查询重写。

您可以通过使用 SET mv_enable_aqmv_for_session to FALSE 来停止会话级别的自动查询重写。

限制

以下是使用实体化视图的自动查询重写的限制：

- 自动查询重写适用于不引用或不包含以下任何操作的实体化视图：
  - 子查询
  - Left、right 或 full outer 联接
  - 设置操作
  - 任意聚合函数（除 SUM、COUNT、MIN、MAX 和 AVG 之外）。（这些是仅适用于自动查询重写的聚合函数。）
  - 任意具有 DISTINCT 的聚合函数
  - 任意窗口函数

  - SELECT DISTINCT 或 HAVING 子句
  - 外部表
  - 其他实体化视图
- 自动查询重写将重写引用用户定义的 Amazon Redshift 表的 SELECT 查询。Amazon Redshift 不会重写以下查询：
  - CREATE TABLE AS 语句
  - SELECT INTO 语句
  - 对目录或系统表的查询
  - 具有外部联接或 SELECT DISTINCT 子句的查询
- 如果没有自动重写查询，请检查您是否对指定的实体化视图具有 SELECT 权限，且 mv_enable_aqmv_for_session 选项是否被设置为 TRUE。

  您还可以通过检查 STV_MV_INFO 来检查实体化视图是否符合自动重写查询的条件。有关更多信息，请参阅 STV_MV_INFO。

刷新实体化视图

创建实体化视图时，其内容将反映当时基础数据库表的状态。实体化视图中的数据将保持不变，即使应用程序更改基础表中的数据也是如此。要更新实体化视图中的数据，您可以随时使用 `REFRESH MATERIALIZED VIEW` 语句手动刷新实体化视图。使用此语句时，Amazon Redshift 会标识已在一个或多个基表中进行的更改，然后将这些更改应用于实体化视图。

Amazon Redshift 有两种刷新实体化视图的策略：

- 在许多情况下，Amazon Redshift 可以执行递增刷新。在递增刷新中，Amazon Redshift 将快速识别自上次刷新以来对基表中的数据所做的更改，并更新实体化视图中的数据。在定义实体化视图时，对查询中使用的以下 SQL 结构支持递增刷新：
  - 包含子句 SELECT、FROM、[INNER] JOIN、WHERE、GROUP BY 或 HAVING 的构造。
  - 包含聚合（例如 SUM、MIN、MAX、AVG 和 COUNT）的构造。
  - 大多数内置 SQL 函数（尤其是那些不可变的 SQL 函数），假定它们具有相同的输入参数并始终产生相同的输出。

  基于数据共享表的实体化视图也支持增量刷新。
- 如果无法执行递增刷新，Amazon Redshift 将执行完全刷新。完全刷新 将重新运行基础 SQL 语句，并替换实体化视图中的所有数据。

- Amazon Redshift 自动为实体化视图选择刷新方法，具体取决于用于定义实体化视图的 SELECT 查询。

在实体化视图上刷新实体化视图不是级联过程。换句话说，假设您具有依赖于实体化视图 B 的实体化视图 A。在这种情况下，当调用 REFRESH MATERIALIZED VIEW 时，将使用当前版本的 B 刷新 A，即使 B 已过期。要使 A 完全更新，请在刷新 A 之前，首先在单独的事务中刷新 B。

以下示例显示如何以编程方式为实体化视图创建完全刷新计划。要刷新实体化视图 v，请首先刷新实体化视图 u。要刷新实体化视图 w，请首先刷新实体化视图 u，然后再刷新实体化视图 v。

```
CREATE TABLE t(a INT);
CREATE MATERIALIZED VIEW u AS SELECT * FROM t;
CREATE MATERIALIZED VIEW v AS SELECT * FROM u;
CREATE MATERIALIZED VIEW w AS SELECT * FROM v;

WITH RECURSIVE recursive_deps (mv_tgt, lvl, mv_dep) AS
( SELECT trim(name) as mv_tgt, 0 as lvl, trim(ref_name) as mv_dep
  FROM stv_mv_deps
  UNION ALL
  SELECT R.mv_tgt, R.lvl+1 as lvl, trim(S.ref_name) as mv_dep
  FROM stv_mv_deps S, recursive_deps R
  WHERE R.mv_dep = S.name
)

SELECT mv_tgt, mv_dep from recursive_deps
ORDER BY mv_tgt, lvl DESC;

 mv_tgt | mv_dep
--------+--------
 v      | u
 w      | u
 w      | v
(3 rows)
```

以下示例显示了在依赖于过期的实体化视图的实体化视图上运行 REFRESH MATERIALIZED VIEW 时的信息性消息。

```
create table a(a int);
```

```
create materialized view b as select * from a;
```

```
create materialized view c as select * from b;
```

```
insert into a values (1);
```

```
refresh materialized view c;

INFO:  Materialized view c is already up to date.  However, it depends on another
 materialized view that is not up to date.
```

```
REFRESH MATERIALIZED VIEW b;
INFO:  Materialized view b was incrementally updated successfully.
```

```
REFRESH MATERIALIZED VIEW c;
INFO:  Materialized view c was incrementally updated successfully.
```

Amazon Redshift 当前对实体化视图的递增刷新具有以下限制。

Amazon Redshift 不支持使用以下 SQL 元素通过查询定义的实体化视图的递增刷新：

- OUTER JOIN（RIGHT、LEFT 或 FULL）。
- 集操作 UNION、INTERSECT、EXCEPT 和 MINUS。
- 聚合函数
  MEDIAN、PERCENTILE_CONT、LISTAGG、STDDEV_SAMP、STDDEV_POP、APPROXIMATE COUNT、APPROXIMATE PERCENTILE 以及按位聚合函数。

  > **Note**
  >
  > 支持 COUNT、SUM 和 AVG 聚合函数。

- DISTINCT 聚合函数，如 DISTINCT COUNT、DISTINCT SUM 等等。
- 窗口函数。
- 使用临时表进行查询优化的查询，例如优化常用的子表达式。
- 子查询。
- 在定义实体化视图的查询中引用以下格式的外部表。
  - Delta Lake
  - Hudi

对于使用上述格式以外的格式定义的实体化视图，预览版跟踪支持增量刷新。有关设置预览版集群的更多信息，请参阅《Amazon Redshift 管理指南》中的创建预览版集群。有关设置预览工作组的信息，请参阅《Amazon Redshift 管理指南》中的创建预览工作组。

自动刷新实体化视图

当使用自动刷新选项创建实体化视图或将其更改为具有自动刷新选项时，Amazon Redshift 可以使用其基表中的最新数据自动刷新实体化视图。Amazon Redshift 会在基表更改后尽快自动刷新实体化视图。

为了完成最重要的实体化视图的刷新，同时最大限度地减少对集群中的活动工作负载的影响，Amazon Redshift 会考虑多个因素。这些因素包括当前系统负载、刷新所需的资源、可用集群资源以及实体化视图的使用频率。

Amazon Redshift 会优先考虑您的工作负载而不是自动刷新，并可能会停止自动刷新以保持用户工作负载的性能。此方法可能会延迟某些实体化视图的刷新。在某些情况下，您的实体化视图可能需要更具确定性的刷新行为。如果是这样，请考虑使用 REFRESH MATERIALIZED VIEW 中所述的手动刷新，或使用 Amazon Redshift 调度程序 API 操作或控制台计划刷新。

您可以使用 CREATE MATERIALIZED VIEW 为实体化视图设置自动刷新。还可以使用 AUTO REFRESH 子句自动刷新实体化视图。有关创建实体化视图的更多信息，请参阅CREATE MATERIALIZED VIEW。您可以使用 ALTER MATERIALIZED VIEW 为当前实体化视图开启自动刷新。

刷新实体化视图时，请注意以下事项：

- 即使尚未为实体化视图启用自动刷新，仍可使用 REFRESH MATERIALIZED VIEW 命令显式刷新实体化视图。
- Amazon Redshift 不会自动刷新外部表中定义的实体化视图。
- 对于刷新状态，您可以选中 SVL_MV_REFRESH_STATUS，这将记录用户启动或自动刷新的查询。
- 要在仅重新计算的实体化视图上运行 REFRESH，请确保您对 Schema 具有 CREATE 权限。有关更多信息，请参阅 GRANT。

自动实体化视图

实体化视图是一个提高 Amazon Redshift 查询性能的强大工具。它们通过存储预先计算的结果集来实现这一目的。类似的查询不必每次重新运行相同的逻辑，因为它们可以从现有结果集中检索记录。开发人员和分析师在分析工作负载后创建实体化视图，以确定哪些查询将受益，以及是否值得为每个实体化

视图支付维护费用。随着工作负载的增长或变化，必须对这些实体化视图进行审查，以确保它们继续提供显著的性能优势。

Redshift 中的自动实体化视图 (AutoMV) 功能提供了与用户创建的实体化视图相同的性能优势。Amazon Redshift 使用机器学习持续监控工作负载，并在它们有用的情况下创建新的实体化视图。AutoMV 在创建实体化视图并保持最新状态的成本与查询延迟的预期益处之间实现平衡。系统还监控以前创建的 AutoMV，并在它们不再有用时将它们丢弃。

AutoMV 的行为和功能与用户创建的实体化视图的相同。它们使用相同的标准和限制自动递增刷新。与用户创建的实体化视图一样，自动查询重写以使用实体化视图识别可受益于系统创建的 AutoMV 的查询。它会自动重写这些查询以使用 AutoMV，从而提高查询性能。开发人员无需修订查询即可利用 AutoMV。

> **Note**
>
> 自动实体化视图会间歇性刷新。重写查询以使用 AutoMV 始终会返回最新结果。当 Redshift 检测到数据不是最新时，不会将查询重写为从自动实体化视图读取。相反，查询会从基表中选择最新数据。

任何重复使用的查询的工作负载都可受益于 AutoMV。常见使用案例包括：

- 控制面板 – 控制面板广泛用于快速查看关键业务指标 (KPI)、事件、趋势和其它指标。它们通常具有包含图表和表格的通用布局，但会针对筛选或维度选择操作（如向下钻取）显示不同的视图。控制面板通常有一组重复使用不同参数的常用查询。控制面板查询可极大地受益于自动实体化视图。
- 报告 – 根据业务要求和报告类型，可以以不同频率计划报告查询。此外，可以自动查询，也可按需查询。报告查询的一个共同特征是它们可能会长时间运行并占用大量资源。借助 AutoMV，这些查询不需要在每次运行时重新计算这些查询，从而减少 Redshift 中每个查询的运行时间和资源利用率。

要关闭自动实体化视图，请将 `auto_mv` 参数组更新为 `false`。有关更多信息，请参阅《Amazon Redshift 集群管理指南》中的 Amazon Redshift 参数组。

自动实体化视图的 SQL 作用域和注意事项

- 自动实体化视图可以由查询或子查询启动和创建，前提是它包含 `GROUP BY` 子句或下列聚合函数之一：SUM、COUNT、MIN、MAX 或 AVG。但它不能包含以下任何内容：
  - Left、right 或 full outer 联接

- SUM、COUNT、MIN、MAX 和 AVG 之外的聚合函数。（这些特定函数适用于自动查询重写。）
- 任何包含 DISTINCT 的聚合函数
- 任意窗口函数
- SELECT DISTINCT 或 HAVING 子句
- 其他实体化视图

无法保证满足条件的查询将启动自动实体化视图的创建。系统根据其对工作负载的预期好处和资源维护成本（包括系统更新成本）来确定从哪个候选项创建视图。每个生成的实体化视图可通过自动重写查询来使用。

- 即使 AutoMV 可能由子查询或集合运算符的各个分支启动，生成的实体化视图也不包含子查询或集合运算符。
- 要确定 AutoMV 是否用于查询，请查看 EXPLAIN 计划并在输出中查找 `%_auto_mv_%`。有关更多信息，请参阅 EXPLAIN。
- 外部表（如数据共享和联合表）不支持自动实体化视图。

自动实体化视图的限制

以下是使用自动实体化视图的限制：

- 最大 AutoMV 数 – 集群中每个数据库的自动实体化视图数的上限为 200 个。
- 存储空间和容量 – AutoMV 的一个重要特征是，使用备用后台周期执行它以帮助实现用户工作负载不受影响。如果集群繁忙或存储空间不足，AutoMV 将停止其活动。具体来说，在总集群容量达到 80% 时，不会创建新的自动实体化视图。在总容量达到 90% 时，可能会删除它们，以方便用户工作负载继续运行，而不会降低性能。有关确定集群容量的更多信息，请参阅 STV_NODE_STORAGE_CAPACITY。

自动实体化视图的计费

Amazon Redshift 的自动优化功能可创建和刷新自动实体化视图。对于此过程不收取计算资源费用。自动实体化视图的存储按常规存储费率收费。有关更多信息，请参阅 Amazon Redshift 定价。

其他 资源

以下博客文章进一步解释了自动实体化视图。其中详细说明了如何创建、维护和删除它们。还解释了推动这些决策的基础算法：使用自动实体化视图优化 Amazon Redshift 查询性能。

该视频首先解释了实体化视图，并展示了它们如何提高性能和节约资源。然后，它通过过程流动画和实况演示对自动实体化视图进行了深入的解释。

在实体化视图中使用用户定义的函数 (UDF)

您可以在 Amazon Redshift 实体化视图中使用标量 UDF。在 Python 或 SQL 中定义它们，并在实体化视图定义中引用。

在实体化视图中引用 UDF

以下过程显示如何在实体化视图定义中使用执行简单算术比较的 UDF。

1. 创建一个表，以在实体化视图定义中使用。

```
CREATE TABLE base_table (a int, b int);
```

2. 在 Python 中创建一个标量用户定义函数，该函数返回一个布尔值，指示某个整数是否大于比较的整数。

```
CREATE OR REPLACE FUNCTION udf_python_bool(x1 int, x2 int) RETURNS bool IMMUTABLE
AS $$
  return x1 > x2
$$ LANGUAGE plpythonu;
```

   或者，您可以使用 SQL 创建功能相似的 UDF，用来将结果与第一个数进行比较。

```
CREATE OR REPLACE FUNCTION udf_sql_bool(int, int) RETURNS bool IMMUTABLE
AS $$
  select $1 > $2;
$$ LANGUAGE SQL;
```

3. 创建一个实体化视图，该视图从您创建的表中进行选择并引用 UDF。

```
CREATE MATERIALIZED VIEW mv_python_udf AS SELECT udf_python_bool(a, b) AS a FROM
 base_table;
```

   或者，您可以创建引用 SQL UDF 的实体化视图。

```
CREATE MATERIALIZED VIEW mv_sql_udf AS SELECT udf_sql_bool(a, b) AS a FROM
 base_table;
```

4. 向表中添加数据并刷新实体化视图。

```
INSERT INTO base_table VALUES (1,2), (1,3), (4,2);
```

```
REFRESH MATERIALIZED VIEW mv_python_udf;
```

或者，您可以刷新引用 SQL UDF 的实体化视图。

```
REFRESH MATERIALIZED VIEW mv_sql_udf;
```

5. 查询实体化视图中的数据。

```
SELECT * FROM mv_python_udf ORDER BY a;
```

查询的结果应如下所示：

```
a
-----
false
false
true
```

对于最后一组值，这将返回 `true`，因为列 a 的值 (4) 大于列 b 的值 (2)。

6. 或者，您可以查询引用 SQL UDF 的实体化视图。SQL 函数的结果与 Python 版本的结果匹配。

```
SELECT * FROM mv_sql_udf ORDER BY a;
```

查询的结果应如下所示：

```
a
-----
false
false
true
```

对于要比较的最后一组值，这将返回 `true`。

7. 使用带 CASCADE 的 DROP 语句可删除用户定义的函数以及引用该函数的实体化视图。

```
DROP FUNCTION udf_python_bool(int, int) CASCADE;
```

```
DROP FUNCTION udf_sql_bool(int, int) CASCADE;
```

串流摄取

串流摄取直接以低延迟、高速度的方式将流数据从 Amazon Kinesis Data Streams 和 Amazon Managed Streaming for Apache Kafka 摄取到 Amazon Redshift 预调配或 Amazon Redshift Serverless 实体化视图中。它减少了访问数据所需的时间并降低了存储成本。您可以为 Amazon Redshift 集群或 Amazon Redshift Serverless 配置串流摄取并使用 SQL 语句创建实体化视图，如在 Amazon Redshift 中创建实体化视图中所述。然后，借助实体化视图刷新，每秒可以摄取数百兆字节的数据。从而快速访问快速刷新的外部数据。

数据流

Amazon Redshift 预调配集群或 Amazon Redshift Serverless 工作组是串流使用者。实体化视图是从串流中读取的数据的着陆区，读取的数据将在到达时进行处理。例如，可以通过熟悉的 SQL 来使用 JSON 值并映射到实体化视图的数据列。在刷新实体化视图时，Redshift 会使用已分配 Kinesis 数据分片或 Kafka 分区中的数据，直到该视图与 Kinesis 流的 SEQUENCE_NUMBER 或 Kafka 主题的上一个 Offset 达到平衡。后续的实体化视图刷新会从上次刷新的最新 SEQUENCE_NUMBER 读取数据，直到与串流数据或主题数据相等为止。

串流摄取使用案例

Amazon Redshift 串流摄取的使用案例涉及处理会持续生成（串流）且必须在生成后的短时间内（低延迟）处理的数据。这称为近实时分析。数据来源多种多样，包括 IoT 设备、系统遥测数据，或热点网站或应用程序的点击流数据。

串流摄取注意事项

以下是设置串流摄取环境时有关性能和计费的重要注意事项和最佳实践。

- 自动刷新使用情况和激活 – 实体化视图的自动刷新查询被视为任何其他用户工作负载。自动刷新会在数据串流到达时从串流加载数据。

  对于为串流摄取而创建的实体化视图，可以显式开启自动刷新。为此，请在实体化视图定义中指定 AUTO REFRESH。手动刷新是默认设置。要为现有实体化视图指定自动刷新以进行串流摄

取，可以运行 ALTER MATERIALIZED VIEW 以开启此功能。有关更多信息，请参阅 CREATE MATERIALIZED VIEW 或 ALTER MATERIALIZED VIEW。

- 串流摄取和 Amazon Redshift Serverless – 适用于预置集群上的 Amazon Redshift 串流摄取的相同设置和配置说明也适用于 Amazon Redshift Serverless 上的串流摄取。重要的是将 Amazon Redshift Serverless 配置为必要的 RPU 级别，以便支持具有自动刷新功能的串流摄取和其他工作负载。有关更多信息，请参阅 Amazon Redshift Serverless 的账单。
- 与 Amazon MSK 集群位于不同可用区的 Amazon Redshift 节点 – 当您配置串流摄取时，如果为 Amazon MSK 启用了机架感知功能，则 Amazon Redshift 会尝试连接到同一可用区中的 Amazon MSK 集群。如果您的所有节点都与 Amazon Redshift 集群位于不同的可用区，则会产生跨可用区的数据传输费用。为避免这种情况，请在与 Redshift 预置集群或工作组相同的可用区中至少保留一个 Amazon MSK 代理集群节点。
- 刷新起始位置 – 创建实体化视图后，其初始刷新从 Kinesis 串流的 TRIM_HORIZON 或从 Amazon MSK 主题的偏移 0 开始。
- 数据格式 – 支持的数据格式仅限于可以从 VARBYTE 转换的那些格式。有关更多信息，请参阅VARBYTE 类型 和VARBYTE 运算符。
- 将记录附加到表 - 您可以运行 ALTER TABLE APPEND，以从现有源实体化视图中将行附加到目标表。仅当将实体化视图配置为串流摄取时，此操作才会起作用。有关更多信息，请参阅 ALTER TABLE APPEND。
- 运行 TRUNCATE 或 DELETE - 您可以使用以下几种方法，从用于串流摄取的实体化视图中删除记录：
  - TRUNCATE – 此命令从配置为串流摄取的实体化视图中删除所有行。它不执行表扫描。有关更多信息，请参阅 TRUNCATE。
  - DELETE – 此命令从配置为串流摄取的实体化视图中删除所有行。有关更多信息，请参阅 DELETE。

流式摄取最佳实践和建议

在某些情况下，系统会向您提供有关如何配置流式摄取的选项。我们建议您遵循以下最佳实践。这些实践基于我们自己的测试，以及帮助客户避免导致数据丢失的问题时学到的经验教训。

- 从流式数据中提取值 – 如果您在实体化视图定义中使用 JSON_EXTRACT_PATH_TEXT 函数分解传入的流式 JSON，这可能会对性能和延迟造成极大的影响。这是因为，对每一个使用 JSON_EXTRACT_PATH_TEXT 提取的列，都会重新解析传入的 JSON。之后就会运行所有的数据类型转换、筛选和业务逻辑。这意味着，如果您从 JSON 数据中提取 10 列，每条 JSON 记录都会被解析 10 次，其中包括类型转换和其他逻辑。这会导致摄取延迟时间更长。我们推荐的另一种方

法是使用 JSON_PARSE 函数，将 JSON 记录转换为 Redshift 的 SUPER 数据类型。在流式数据进入实体化视图后，使用 PartiQL 从 JSON 数据的 SUPER 表示形式中提取各个字符串。有关更多信息，请参阅查询半结构化数据。

同样重要的是要注意，JSON_EXTRACT_PATH_TEXT 的数据大小最大值为 64KB。因此，如果任何 JSON 记录大于 64KB，那么使用 JSON_EXTRACT_PATH_TEXT 处理记录会导致错误。

- 将 Amazon Kinesis Data Streams 流或 Amazon MSK 主题映射到 Amazon Redshift 流式摄取实体化视图 – 我们不建议创建多个流式摄取实体化视图，来从单个 Amazon Kinesis Data Streams 流或 Amazon MSK 主题中摄取数据。这是因为，对于 Kinesis Data Streams 流中的每个分片或 Kafka 主题中的每个分区，每个实体化视图都会创建一个使用者。这会导致节流，或者超出流或主题的吞吐量。这还可能导致成本增加，因为您会多次摄取相同的数据。我们建议您为每个流或主题创建一个流式实体化视图。

  如果您的应用场景要求您将一个 KDS 流或 MSK 主题中的数据放入多个实体化视图中，在执行操作之前，请先参考 AWS 大数据博客，尤其是 Best practices to implement near-real-time analytics using Amazon Redshift Streaming Ingestion with Amazon MSK。

使用串流摄取与在 Amazon S3 中暂存数据进行比较

有多种选项可以将数据流式传输到 Amazon Redshift 或 Amazon Redshift Serverless。两个众所周知的选项是串流摄取（如本主题所述），或者使用 Firehose 设置到 Amazon S3 的传输流。以下列表描述了每种方法：

1. 从 Kinesis Data Streams 或 Amazon Managed Streaming for Apache Kafka 到 Amazon Redshift 或 Amazon Redshift Serverless 的串流摄取涉及配置实体化视图来接收数据。
2. 使用 Kinesis Data Streams 将数据传输到 Amazon Redshift 并通过 Firehose 进行流式传输的过程涉及将源流连接到 Amazon Data Firehose，然后等待 Firehose 在 Amazon S3 中暂存数据。此过程使用不同大小的批次，缓冲间隔长度各不相同。在流式传输到 Amazon S3 后，Firehose 会启动一个 COPY 命令来加载数据。

通过串流摄取，您可以绕过第二个过程所需的几个步骤：

- 您不必将数据发送到 Amazon Data Firehose 传输流，因为通过串流摄取，数据可以直接从 Kinesis Data Streams 发送到 Redshift 数据库中的实体化视图。
- 您不必将流式传输的数据存入 Amazon S3，因为串流摄取数据会直接传送到 Redshift 实体化视图。
- 您不必编写和运行 COPY 命令，因为实体化视图中的数据是直接从流中刷新的。将数据从 Amazon S3 加载到 Redshift 并不是该过程的一部分。

请注意，串流摄取仅限于来自 Amazon Kinesis Data Streams 的流和来自 Amazon MSK 的主题。要从 Kinesis Data Streams 流式传输到 Amazon Redshift 以外的目标，可能需要一个 Firehose 传输流。有关更多信息，请参阅 Sending Data to an Amazon Data Firehose Delivery Stream。

注意事项

以下是串流摄取到 Amazon Redshift 的注意事项。

特征或操作	描述
Kafka 主题长度限制	不能使用名称超过 128 个字符（不包括引号）的 Kafka 主题。有关更多信息，请参阅名称和标识符。
实体化视图的增量刷新和联接	实体化视图必须以递增方式维护。Kinesis 或 Amazon MSK 无法完全重新计算，因为在默认情况下，它们不会保留过去 24 小时或 7 天的串流或主题历史记录。您可以在 Kinesis 或 Amazon MSK 中设置更长的数据留存期。但是，这会导致需要更多的维护和成本。此外，在 Kinesis 流或 Amazon MSK 主题上创建的实体化视图目前不支持联接。创建串流或主题的实体化视图后，您可以创建另一个实体化视图，以用于将串流实体化视图联接到其他实体化视图、表或视图。 有关更多信息，请参阅 REFRESH MATERIALIZED VIEW。
记录解析	Amazon Redshift 串流摄取不支持解析 Kinesis Producer Library 聚合的记录（KPL 关键概念 - 聚合）。聚合的记录摄取后将存储为二进制协议缓冲区数据。（有关更多信息，请参阅协议缓冲。）根据向 Kinesis 推送数据的方式不同，您可能需要关闭此功能。
解压缩	VARBYTE 目前不支持任何解压缩方法。因此，无法在 Redshift 中查询包含压缩数据的记录。在将数据推送到 Kinesis 流或 Amazon MSK 主题之前解压缩。
最大记录大小	Amazon Redshift 可以从 Kinesis 或 Amazon MSK 摄取的任何记录字段的最大大小略小于 1MB。以下几点详细说明了该行为：

特征或操作	描述
	• 最大 VARBYTE 长度 – 对于串流摄取，VARBYTE 类型支持最大长度为 1,024,000 字节的数据。Kinesis 将有效负载限制在 1 MB 以内。 • 消息限制 – 默认 Amazon MSK 配置将消息限制在 1 MB 以内。此外，如果消息包括标头，则数据量限制为 1,048,470 字节。使用默认设置，摄取没有问题。但是，您可以将 Kafka 的最大消息大小更改为更大的值，从而更改 Amazon MSK 的最大消息大小。在这种情况下，Kafka 记录的键/值字段或标头可能会超过大小限制。这些记录会引起错误，并且不会摄取。 **Note** 对于来自 Kinesis 或 Amazon MSK 的串流摄取，Amazon Redshift 支持最大 1,024,000 字节的数据大小，尽管 Amazon Redshift 支持最大 16 MB 的 VARBYTE 数据类型。
错误记录	在每种情况下，如果由于数据的大小超过最大大小而无法将记录摄取到 Redshift，则会跳过该记录。在这种情况下，实体化视图刷新仍然会成功，并且每条错误记录的一段会写入到 SYS_STREAM_SCAN_ERRORS 系统表中。系统不会跳过由业务逻辑导致的错误，例如计算错误或类型转换产生的错误。在向实体化视图定义添加逻辑之前，请仔细测试逻辑，以避免出现这些错误。
Amazon MSK 多 VPC 私有连接	Redshift 串流摄取目前不支持 Amazon MSK 多 VPC 私有连接。另外，您也可以使用 VPC 对等连接来连接 VPC，或使用 AWS Transit Gateway 通过中央枢纽连接 VPC 和本地网络。上述任一方法都可以让 Redshift 与 Amazon MSK 集群或与另一个 VPC 中的 Amazon MSK Serverless 进行通信。

开始使用 Amazon Kinesis Data Streams 串流摄取

Amazon Redshift 串流摄取的设置涉及创建映射到流数据来源的外部 Schema 以及创建引用外部 Schema 的实体化视图。Amazon Redshift 串流摄取支持将 Kinesis Data Streams 作为源。因此，在配置串流摄取之前，您必须有可用的 Kinesis Data Streams 源。如果还没有源，请按照 Kinesis 文档 Amazon Kinesis Data Streams 入门中的说明执行操作，或者按照通过 AWS 管理控制台创建流中的说明在控制台上创建一个源。

Amazon Redshift 串流摄取使用实体化视图，该视图将在 REFRESH 运行时直接从流中更新。实体化视图映射到流数据来源。在实体化视图定义中，您可以对流数据执行筛选和聚合。串流摄取实体化视图（基本实体化视图）只能引用一个流，但是您可以创建额外的实体化视图，以与基本实体化视图和其他实体化视图或表连接使用。

> **Note**
>
> 串流摄取和 Amazon Redshift Serverless – 本主题中的配置步骤同时适用于预调配的 Amazon Redshift 集群和 Amazon Redshift Serverless。有关更多信息，请参阅 串流摄取注意事项。

假设您有 Kinesis Data Streams 流可用，第一步是使用 CREATE EXTERNAL SCHEMA 在 Amazon Redshift 中定义一个 Schema，并引用某个 Kinesis Data Streams 资源。之后，要访问流中的数据，请在实体化视图中定义 STREAM。您可以用半结构化的 SUPER 格式存储流记录，或者定义一个会将数据转换为 Redshift 数据类型的 Schema。当您查询实体化视图时，返回的记录是流的时间点视图。

1. 使用允许 Amazon Redshift 集群或 Amazon Redshift Serverless 工作组代入该角色的信任策略创建 IAM 角色。有关如何为 IAM 角色配置信任策略的信息，请参阅授权 Amazon Redshift 代表您访问其他 AWS 服务。创建后的角色应具有以下 IAM 策略，提供与 Amazon Kinesis 数据流进行通信的权限。

   来自 Kinesis 数据流的未加密流 IAM 策略

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:GetShardIterator",
```

```
                "kinesis:GetRecords",
                "kinesis:DescribeStream"
            ],
            "Resource": "arn:aws:kinesis:*:0123456789:stream/*"
        },
        {
            "Sid": "ListStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:ListStreams",
                "kinesis:ListShards"
            ],
            "Resource": "*"
        }
    ]
}
```

来自 Kinesis 数据流的加密流 IAM 策略

```
{
 "Version": "2012-10-17",
 "Statement": [{
   "Sid": "ReadStream",
   "Effect": "Allow",
   "Action": [
    "kinesis:DescribeStreamSummary",
    "kinesis:GetShardIterator",
    "kinesis:GetRecords",
    "kinesis:DescribeStream"
   ],
   "Resource": "arn:aws:kinesis:*:0123456789:stream/*"
  },
  {
   "Sid": "DecryptStream",
   "Effect": "Allow",
   "Action": [
    "kms:Decrypt"
   ],
   "Resource": "arn:aws:kms:us-
east-1:0123456789:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  },
        {
   "Sid": "ListStream",
```

```
    "Effect": "Allow",
    "Action": [
     "kinesis:ListStreams",
     "kinesis:ListShards"
    ],
    "Resource": "*"
   }
 ]
}
```

2. 检查您的 VPC，并确认您的 Amazon Redshift 集群或 Amazon Redshift Serverless 拥有使用 NAT 网关或互联网网关通过互联网到达 Kinesis Data Streams 端点的路由。如果您想让 Redshift 和 Kinesis Data Streams 之间的流量保持在 AWS 网络内，可以考虑使用 Kinesis 接口 VPC 端点。有关更多信息，请参阅将 Amazon Kinesis Data Streams 与接口 VPC 端点结合使用。
3. 在 Amazon Redshift 中，创建外部 Schema 以将 Kinesis 中的数据映射到某个 Schema。

```
CREATE EXTERNAL SCHEMA kds
FROM KINESIS
IAM_ROLE { default | 'iam-role-arn' };
```

Kinesis Data Streams 的串流摄取不需要身份验证类型。它使用在 CREATE EXTERNAL SCHEMA 语句中定义的 IAM 角色来发出 Kinesis Data Streams 请求。

可选：使用 REGION 关键字指定 Amazon Kinesis Data Streams 或 Amazon MSK 流所在的区域。

```
CREATE EXTERNAL SCHEMA kds
FROM KINESIS
REGION 'us-west-2'
IAM_ROLE { default | 'iam-role-arn' };
```

在此示例中，区域指定了源流的位置。IAM_ROLE 就是一个示例。

4. 创建一个实体化视图以使用流数据。使用如下语句，如果无法解析记录，则会导致错误。如果您不想跳过错误记录，请使用类似这样的命令。

```
CREATE MATERIALIZED VIEW my_view AUTO REFRESH YES AS
SELECT *
FROM kds.my_stream_name;
```

以下示例定义了采用 JSON 格式的源数据的实体化视图。该视图可验证传入数据的 JSON 格式是否正确。Kinesis 流名称区分大小写，可以包含大写字母和小写字母。要从具有大写名称的流中摄取，可以在数据库级别将配置 `enable_case_sensitive_identifier` 设置为 `true`。有关更多信息，请参阅名称和标识符和 enable_case_sensitive_identifier。

```
CREATE MATERIALIZED VIEW my_view AUTO REFRESH YES AS
SELECT approximate_arrival_timestamp,
partition_key,
shard_id,
sequence_number,
refresh_time,
JSON_PARSE(kinesis_data) as kinesis_data
FROM kds.my_stream_name
WHERE CAN_JSON_PARSE(kinesis_data);
```

要开启自动刷新，请使用 `AUTO REFRESH YES`。默认行为是手动刷新。请注意，当您使用 CAN_JSON_PARSE 时，可能会跳过无法解析的记录。

元数据列包括以下内容：

元数据列	数据类型	描述
approximate_arrival_timestamp	不带时区的时间戳	记录插入 Kinesis 流的大致时间
partition_key	varchar(256)	Kinesis 用于将记录分配给分片的键
shard_id	char(20)	从中检索记录的串流内的分片的唯一标识符
sequence_number	varchar(128)	Kinesis 分片中的记录的唯一标识符
refresh_time	不带时区的时间戳	刷新开始的时间
kinesis_data	varbyte	来自 Kinesis 串流的记录

请务必注意，如果您的实体化视图定义中有业务逻辑，那么在某些情况下，业务逻辑错误可能会导致串流摄取被阻止。这可能会导致您不得不删除实体化视图，然后重新创建。为避免这种情况，我们建议您尽可能简化逻辑，并在摄取数据后对数据进行大部分业务逻辑检查。

5. 刷新视图，这会调用 Redshift 从串流中读取数据并将数据加载到实体化视图中。

```
REFRESH MATERIALIZED VIEW my_view;
```

6. 在实体化视图中查询数据。

```
select * from my_view;
```

开始使用 Amazon Managed Streaming for Apache Kafka 串流摄取

Amazon Redshift 串流摄取的目的是简化将串流数据直接从串流服务摄取到 Amazon Redshift 或 Amazon Redshift Serverless 的过程。这适用于 Amazon MSK 和 Amazon MSK Serverless 以及 Kinesis。使用 Amazon Redshift 串流摄取时，在将串流数据摄取到 Redshift 之前，无需在 Amazon S3 中暂存 Kinesis Data Streams 流或 Amazon MSK 主题。

在技术层面上，来自 Amazon Kinesis Data Streams 和 Amazon Managed Streaming for Apache Kafka 的串流摄取以低延迟、高速度的方式将串流或主题数据摄取到 Amazon Redshift 实体化视图中。设置完成后，使用实体化视图刷新，可以接收大量数据。

通过执行以下步骤，为 Amazon MSK 设置 Amazon Redshift 串流摄取：

1. 创建映射到串流数据来源的外部 Schema。
2. 创建引用外部 Schema 的实体化视图。

在配置 Amazon Redshift 串流摄取之前，您必须有可用的 Amazon MSK 源。如果您没有源，请按照开始使用 Amazon MSK 中的说明进行操作。

> **Note**
>
> 串流摄取和 Amazon Redshift Serverless – 本主题中的配置步骤同时适用于预调配的 Amazon Redshift 集群和 Amazon Redshift Serverless。有关更多信息，请参阅 串流摄取注意事项。

设置 IAM 并从 Kafka 执行串流摄取

假设您有可用的 Amazon MSK 集群，第一步是使用 CREATE EXTERNAL SCHEMA 在 Redshift 中定义一个架构并引用 Kafka 主题作为数据来源。之后，要访问主题中的数据，请在实体化视图中定义 STREAM。您可以用半结构化 SUPER 格式存储来自主题的记录，或者定义一个会将数据转换为 Amazon Redshift 数据类型的 Schema。当您查询实体化视图时，返回的记录是主题的时间点视图。

1. 使用允许 Amazon Redshift 集群或 Amazon Redshift Serverless 代入 IAM 角色的信任策略创建该角色。有关如何为 IAM 角色配置信任策略的信息，请参阅授权 Amazon Redshift 代表您访问其他 AWS 服务。创建角色后，它应具有以下 IAM 策略，从而提供与 Amazon MSK 集群进行通信的权限。如果您使用 Amazon MSK，则所需的策略取决于集群上使用的身份验证方法。有关 Amazon MSK 中可用的身份验证方法，请参阅 Apache Kafka API 的身份验证和授权。

   使用未经身份验证的访问权限时的 Amazon MSK IAM 策略：

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "VisualEditor0",
               "Effect": "Allow",
               "Action": [
                   "kafka:GetBootstrapBrokers"
               ],
               "Resource": "*"
           }
       ]
   }
   ```

   使用 IAM 身份验证时的 Amazon MSK IAM 策略：

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "MSKIAMpolicy",
               "Effect": "Allow",
               "Action": [
                   "kafka-cluster:ReadData",
                   "kafka-cluster:DescribeTopic",
                   "kafka-cluster:Connect"
   ```

```
            ],
            "Resource": [
                "arn:aws:kafka:*:0123456789:cluster/MyTestCluster/*",
                "arn:aws:kafka:*:0123456789:topic/MyTestCluster/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeGroup"
            ],
            "Resource": [
                "arn:aws:kafka:*:0123456789:group/MyTestCluster/*"
            ]
        },
        {
            "Sid": "MSKPolicy",
            "Effect": "Allow",
            "Action": [
                "kafka:GetBootstrapBrokers"
            ],
            "Resource": "*"
        }
    ]
}
```

2. 检查您的 VPC，并确认您的 Amazon Redshift 集群或 Amazon Redshift Serverless 拥有通往 Amazon MSK 集群的路由。Amazon MSK 集群的入站安全组规则应允许 Amazon Redshift 集群或 Amazon Redshift Serverless 工作组的安全组。如果您使用 Amazon MSK，则您指定的端口取决于集群上使用的身份验证方法。有关更多信息，请参阅端口信息和从 AWS 内但在 VPC 外部访问。

   请注意，对于串流摄取，不支持通过 mTLS 进行客户端身份验证。有关更多信息，请参阅限制。

   下表显示了为从 Amazon MSK 进行串流摄取所要设置的免费配置选项：

Amazon Redshift 配置	Amazon MSK 配置	要在 Redshift 和 Amazon MSK 之间打开的端口
AUTHENTICATION NONE	TLS 传输已禁用	9092

Amazon Redshift 配置	Amazon MSK 配置	要在 Redshift 和 Amazon MSK 之间打开的端口
AUTHENTICATION NONE	TLS 传输已启用	9094
AUTHENTICATION IAM	IAM	9098/9198

Amazon Redshift 身份验证是在 CREATE EXTERNAL SCHEMA 语句中设置的。

如果 Amazon MSK 集群启用了相互传输层安全性协议 (mTLS) 身份验证，则将 Amazon Redshift 配置为使用 AUTHENTICATION NONE 会指示它使用端口 9094 进行未经身份验证的访问。但是，由于 mTLS 身份验证正在使用该端口，因此这一过程将失败。因此，当您使用 mTLS 时，我们建议您切换到 AUTHENTICATION IAM。

3. 在 Amazon Redshift 集群或 Amazon Redshift Serverless 工作组中启用增强型 VPC 路由。有关更多信息，请参阅启用增强型 VPC 路由。

> **Note**
>
> 为了检索 Amazon MSK 引导程序代理 URL，Amazon Redshift 使用附加的 IAM 角色提供的权限进行 GetBootstrapBrokers API 调用。请注意，为了使此请求在启用增强型 VPC 路由时获得成功，您的 Amazon Redshift 预调配集群或 Amazon Redshift Serverless 工作组的子网必须具有 NAT 网关或互联网网关。您的网络 ACL 和上述子网的安全组出站规则还必须允许访问 Amazon MSK API 服务端点。有关更多信息，请参阅 Amazon Managed Streaming for Apache Kafka 端点和限额。

4. 在 Amazon Redshift 中，创建一个外部 Schema 以映射到 Amazon MSK 集群。

```
CREATE EXTERNAL SCHEMA MySchema
FROM MSK
IAM_ROLE { default | 'iam-role-arn' }
AUTHENTICATION { none | iam }
CLUSTER_ARN 'msk-cluster-arn';
```

在 FROM 子句中，Amazon MSK 表示模式映射来自托管式 Kafka 服务的数据。

当您创建外部模式时，Amazon MSK 的串流摄取提供以下身份验证类型：

- 无 – 指定没有身份验证步骤。

- iam – 指定 IAM 身份验证。选择此选项时，请确保 IAM 角色具有 IAM 身份验证的权限。

串流摄取不支持其他 Amazon MSK 身份验证方法，例如 TLS 身份验证或用户名和密码。

CLUSTER_ARN 指定要从中进行流式传输的 Amazon MSK 集群。

5. 创建一个实体化视图以使用来自主题的数据。如果您不想跳过错误记录，请使用类似此示例的 SQL 命令。

```
CREATE MATERIALIZED VIEW MyView AUTO REFRESH YES AS
SELECT *
FROM MySchema."mytopic";
```

以下示例定义了一个包含 JSON 源数据的实体化视图。请注意，以下视图会确认数据是有效 JSON 和 utf8。Kafka 主题名称区分大小写，可以包含大写字母和小写字母。要从具有大写名称的主题中摄取，可以在数据库级别将配置 enable_case_sensitive_identifier 设置为 true。有关更多信息，请参阅名称和标识符和 enable_case_sensitive_identifier。

```
CREATE MATERIALIZED VIEW MyView AUTO REFRESH YES AS
SELECT kafka_partition,
 kafka_offset,
 kafka_timestamp_type,
 kafka_timestamp,
 kafka_key,
 JSON_PARSE(kafka_value) as kafka_data,
 kafka_headers,
 refresh_time
FROM MySchema."mytopic"
WHERE CAN_JSON_PARSE(kafka_value);
```

要开启自动刷新，请使用 AUTO REFRESH YES。默认行为是手动刷新。

元数据列包括以下内容：

元数据列	数据类型	描述
kafka_partition	bigint	来自 Kafka 主题的记录的分区 ID

元数据列	数据类型	描述
kafka_offset	bigint	Kafka 主题中给定分区的记录的偏移
kafka_timestamp_type	char(1)	Kafka 记录中使用的时间戳类型： • C – 客户端的记录创建时间 (CREATE_TIME) • L – Kafka 服务器端的记录追加时间 (LOG_APPEND_TIME) • U – 记录创建时间不可用 (NO_TIMESTAMP_TYPE)
kafka_timestamp	不带时区的时间戳	记录的时间戳值
kafka_key	varbyte	Kafka 记录的键
kafka_value	varbyte	从 Kafka 收到的记录
kafka_headers	super	从 Kafka 收到的记录的标头
refresh_time	不带时区的时间戳	刷新开始的时间

请务必注意，如果您的实体化视图定义中有业务逻辑，那么在某些情况下，业务逻辑错误可能会导致串流摄取被阻止。这可能会导致您不得不删除实体化视图，然后重新创建。为避免这种情况，我们建议您尽可能简化业务逻辑，并在摄取数据后对数据运行额外的逻辑。

6. 刷新视图，这会调用 Amazon Redshift 从主题中读取数据并将数据加载到实体化视图中。

```
REFRESH MATERIALIZED VIEW MyView;
```

7. 在实体化视图中查询数据。

```
select * from MyView;
```

当 REFRESH 运行时，直接从主题更新实体化视图。您创建映射到 Kafka 主题数据来源的实体化视图。在实体化视图定义中，您可以对数据执行筛选和聚合。串流摄取实体化视图（基本实体化视图）只能引用一个 Kafka 主题，但是您可以创建额外的实体化视图，以与基本实体化视图和其他实体化视图或表连接使用。

有关串流摄取限制的更多信息，请参阅 注意事项。

使用 Kinesis 进行电动汽车充电站数据流摄取教程

此过程演示如何从名为 ev_station_data 的 Kinesis 流中摄取数据。该流包含来自不同电动汽车充电站的消费数据，采用 JSON 格式。Schema 定义得很好。此示例演示了如何将数据存储为原始 JSON 格式，以及如何在摄取时将 JSON 数据转换为 Amazon Redshift 数据类型。

创建者设置

1. 使用 Amazon Kinesis Data Streams，按照以下步骤创建一个名为 ev_station_data 的流。对于容量模式，选择按需。有关更多信息，请参阅通过 AWS 管理控制台创建流。
2. Amazon Kinesis Data Generator 可以帮助您生成测试数据以供您的流使用。按照工具中详细说明的步骤开始使用，然后使用以下数据模板生成数据：

```
{

   "_id" : "{{random.uuid}}",
   "clusterID": "{{random.number(
        {   "min":1,
            "max":50
        }
    )}}",
    "connectionTime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
    "kWhDelivered": "{{commerce.price}}",
    "stationID": "{{random.number(
        {   "min":1,
            "max":467
        }
    )}}",
      "spaceID": "{{random.word}}-{{random.number(
        {   "min":1,
            "max":20
        }
    )}}",
```

```
   "timezone": "America/Los_Angeles",
   "userID": "{{random.number(
        {   "min":1000,
            "max":500000
        }
    )}}"
}
```

流数据中的每个 JSON 对象都包含以下属性。

```
{
    "_id": "12084f2f-fc41-41fb-a218-8cc1ac6146eb",
    "clusterID": "49",
    "connectionTime": "2022-01-31 13:17:15",
    "kWhDelivered": "74.00",
    "stationID": "421",
    "spaceID": "technologies-2",
    "timezone": "America/Los_Angeles",
    "userID": "482329"
}
```

Amazon Redshift 设置

以下步骤演示了如何配置实体化视图以摄取数据。

1. 创建外部 Schema 以将 Kinesis 中的数据映射到某个 Redshift 对象。

   ```
   CREATE EXTERNAL SCHEMA evdata FROM KINESIS
   IAM_ROLE 'arn:aws:iam::0123456789:role/redshift-streaming-role';
   ```

   有关如何配置 IAM 角色的更多信息，请参阅开始使用 Amazon Kinesis Data Streams 串流摄取。

2. 创建一个实体化视图以使用流数据。以下示例演示了定义实体化视图以摄取 JSON 源数据的两种方法。

   首先，以半结构化的 SUPER 格式存储流记录。在此示例中，JSON 源存储在 Redshift 中，并未转换为 Redshift 类型。

   ```
   CREATE MATERIALIZED VIEW ev_station_data AS
       SELECT approximate_arrival_timestamp,
   ```

```
    partition_key,
    shard_id,
    sequence_number,
    json_parse(kinesis_data) as payload
    FROM evdata."ev_station_data" WHERE can_json_parse(kinesis_data);
```

而在下面的实体化视图定义中，实体化视图具有在 Redshift 中定义的 Schema。实体化视图按照来自流的 UUID 值分配，并按 approximatearrivaltimestamp 值存储。

```
CREATE MATERIALIZED VIEW ev_station_data_extract DISTKEY(6) sortkey(1) AUTO REFRESH
 YES AS
    SELECT refresh_time,
    approximate_arrival_timestamp,
    partition_key,
    shard_id,
    sequence_number,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'_id',true)::character(36)
 as ID,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'clusterID',true)::varchar(30)
 as clusterID,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'connectionTime',true)::varchar(
 as connectionTime,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'kWhDelivered',true)::DECIMAL(10
 as kWhDelivered,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'stationID',true)::DECIMAL(10,2)
 as stationID,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'spaceID',true)::varchar(100)
 as spaceID,
    json_extract_path_text(from_varbyte(kinesis_data,
 'utf-8'),'timezone',true)::varchar(30)as timezone,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'userID',true)::varchar(30)
 as userID
    FROM evdata."ev_station_data"
    WHERE LENGTH(kinesis_data) < 65355;
```

查询流

1. 查询刷新后的实体化视图以获取使用情况统计信息。

```
SELECT to_timestamp(connectionTime, 'YYYY-MM-DD HH24:MI:SS') as connectiontime
,SUM(kWhDelivered) AS Energy_Consumed
,count(distinct userID) AS #Users
from ev_station_data_extract
group by  to_timestamp(connectionTime, 'YYYY-MM-DD HH24:MI:SS')
order by 1 desc;
```

2. 查看结果。

```
connectiontime         energy_consumed #users
2022-02-08 16:07:21+00   4139         10
2022-02-08 16:07:20+00   5571         10
2022-02-08 16:07:19+00   8697         20
2022-02-08 16:07:18+00   4408         10
2022-02-08 16:07:17+00   4257         10
2022-02-08 16:07:16+00   6861         10
2022-02-08 16:07:15+00   5643         10
2022-02-08 16:07:14+00   3677         10
2022-02-08 16:07:13+00   4673         10
2022-02-08 16:07:11+00   9689         20
```

在 AWS Glue Data Catalog 中创建视图（预览版）

以下是预览版 Data Catalog for Amazon Redshift 中的预发行文档视图。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

您可以在预览版中创建 Amazon Redshift 集群，以便测试 Amazon Redshift 的新功能。您无法在生产环境中使用这些功能，也无法将预览版集群移动到生产集群或另一个跟踪上的集群。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

在预览版中创建集群

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择预置集群控制面板，然后选择集群。列出您的账户在当前 AWS 区域 区域中的集群。列表中的各个列中显示了每个集群的一部分属性。
3. 集群列表页面上会显示一个横幅，其中介绍了预览版。选择创建预览版集群按钮以打开创建集群页面。
4. 输入集群的属性。选择包含要测试的功能的预览版跟踪。我们建议输入的集群名称指明要对该集群进行预览版跟踪。为您的集群选择选项，包括标记为 -preview 的选项，用于要测试的功能。有关创建集群的一般信息，请参阅《Amazon Redshift 管理指南》中的创建集群。
5. 选择创建集群以在预览模式下创建集群。

   > **Note**
   >
   > `preview_2023` 跟踪是最新可用的预览版跟踪。此版本仅支持创建具有 RA3 节点类型的集群。不支持节点类型 DC2 以及任何更早的节点类型。

6. 当您的预览集群可用时，使用 SQL 客户端加载和查询数据。

Data Catalog 视图预览功能仅在以下区域中可用。

- 美国东部（俄亥俄州）(us-east-2)
- 美国东部（弗吉尼亚州北部）(us-east-1)

- 美国西部（北加利福尼亚）(us-west-1)
- 亚太地区（东京）(ap-northeast-1)
- 欧洲地区（爱尔兰）(eu-west-1)
- 欧洲地区（斯德哥尔摩）(eu-north-1)

您也可以创建预览工作组来测试 Data Catalog 视图。您无法在生产中使用这些功能，也无法将您的工作组移至其他工作组。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。有关如何创建预览工作组的说明，请参阅创建预览工作组。

通过在 AWS Glue Data Catalog 中创建视图，您可以创建单一的通用视图架构和元数据对象，以便在 Amazon Athena 和 Amazon EMR Spark 等引擎中使用。这样做可以让您在数据湖和数据仓库中使用相同的视图来适应您的使用案例。Data Catalog 中的视图是特殊的，因为它们被归类为定义者视图，其中访问权限由创建视图的用户定义，而不是由查询视图的用户定义。以下是一些在 Data Catalog 中创建视图的使用案例和好处：

- 创建一个视图，根据用户所需的权限来限制数据访问。例如，您可以使用 Data Catalog 中的视图来防止不在 HR 部门工作的员工查看个人身份信息 (PII)。
- 确保用户无法访问不完整的记录。通过对 Data Catalog 中的视图应用某些筛选器，可以确保 Data Catalog 中视图内的数据记录始终完整。
- Data Catalog 视图还具有安全优势，即确保用于创建视图的查询定义必须完整才能创建视图。这种安全性优势意味着 Data Catalog 中的视图不容易受到恶意玩家的 SQL 命令的影响。
- Data Catalog 中的视图支持与普通视图相同的优点，例如允许用户访问视图，而不向用户提供底层表。

要在 Data Catalog 中创建视图，必须有 Spectrum 外部表、包含在 Lake Formation 管理的数据共享中的对象或 Apache Iceberg 表。

Data Catalog 视图定义存储于 AWS Glue Data Catalog。可以使用 AWS Lake Formation 通过资源授权、列授权或基于标签的访问控制来授予访问权限。有关在 Lake Formation 中授予和撤销访问权限的更多信息，请参阅授予和撤销对 Data Catalog 资源的权限。

先决条件

在 Data Catalog 中创建视图之前，请确保已完成以下先决条件：

- 确保您的 IAM 角色具有以下信任策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
               "Service": [
                    "glue.amazonaws.com",
                    "lakeformation.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

- 您还需要以下传递角色策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1",
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com",
                        "lakeformation.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

- 最后，您还需要以下权限。
  - Glue:GetDatabase

- Glue:GetDatabases
- Glue:CreateTable
- Glue:GetTable
- Glue:UpdateTable
- Glue:DeleteTable
- Glue:GetTables
- Glue:SearchTables
- Glue:BatchGetPartition
- Glue:GetPartitions
- Glue:GetPartition
- Glue:GetTableVersion
- Glue:GetTableVersions

端到端示例

首先基于 Data Catalog 数据库创建外部架构。

```
CREATE EXTERNAL SCHEMA IF NOT EXISTS external_schema FROM DATA CATALOG DATABASE
 'external_data_catalog_db'
IAM_ROLE 'arn:aws:iam::123456789012:role/sample-role';
```

现在您可以创建 Data Catalog 视图。

```
CREATE EXTERNAL PROTECTED VIEW external_schema.remote_view
AS SELECT * FROM external_schema.remote_table;
```

然后，您可以开始查询视图。

```
SELECT * FROM external_schema.remote_view;
```

有关与 Data Catalog 中的视图相关的 SQL 命令的更多信息，请参阅 CREATE EXTERNAL VIEW、ALTER EXTERNAL VIEW 和 DROP EXTERNAL VIEW。

注意事项和限制

以下是适用于在 Data Catalog 中创建的视图的注意事项和限制。

- 无法创建基于其他视图的 Data Catalog 视图。
- 在 Data Catalog 视图中只能有 10 个基表。
- 视图的定义者必须对基表拥有完全 SELECT GRANTABLE 权限。
- 视图只能包含 Lake Formation 对象和内置对象。不允许在视图中使用以下对象。
  - 系统表
  - 用户定义的函数 (UDF)
  - 不在 Lake Formation 管理的数据共享中的 Redshift 表、视图、实体化视图和后期绑定视图。
- 视图不能包含嵌套的 Redshift Spectrum 表。
- 只能使用双点符号来查询视图。不支持从外部安装的数据库查询 Lake Formationviews。
- Redshift 视图中引用的 Lake Formation 表的 ARN 长度必须少于 127 个字符。
- 视图基础对象的 AWS Glue 表示形式必须与视图位于相同 AWS 账户和区域中。

在 Amazon Redshift 中查询空间数据

空间数据描述几何体在定义空间（空间参照系）中的位置和形状。Amazon Redshift 支持具有 GEOMETRY 和 GEOGRAPHY 数据类型的空间数据，其中包含空间数据和数据的空间参考系统标识符 (SRID)。

空间数据包含可用于表示地理要素的几何数据。此类数据的示例包括天气报告、地图方向、包含地理位置的推特、店铺位置以及航空公司路线。空间数据在业务分析、报告和预测中起着重要的作用。

您可以使用 Amazon Redshift SQL 函数查询空间数据。空间数据包含对象的几何值。

GEOMETRY 数据类型操作在笛卡尔平面上工作。尽管空间参考系统标识符 (SRID) 存储在对象内部，但该 SRID 只是坐标系的标识符，在处理 GEOMETRY 对象的算法中没有任何作用。对 GEOGRAPHY 数据类型执行的操作将对象内部的坐标视为球体上的球面坐标。此球体由 SRID 定义，该 SRID 引用了地理空间参考系统。默认情况下，GEOGRAPHY 数据类型是使用空间参考 (SRID) 4326 创建的，参照世界大地测量系统 (WGS) 84。有关 SRID 的更多信息，请参阅维基百科中的空间参考系统。

您可以使用 ST_Transform 函数来转换来自各种空间参考系统的坐标。坐标转换完成后，您还可以在两者之间使用简单的转换，只需使用地理 SRID 对输入 GEOMETRY 进行编码即可。此转换仅复制坐标，无需进一步转换。例如：

```
SELECT ST_AsEWKT(ST_GeomFromEWKT('SRID=4326;POINT(10 20)')::geography);
```

```
st_asewkt
------------------------
 SRID=4326;POINT(10 20)
```

为了更好地理解 GEOMETRY 和 GEOGRAPHY 数据类型之间的区别，考虑使用世界大地测量系统 (WGS) 84 来计算柏林机场 (BER) 和旧金山机场 (SFO) 之间的距离。使用 GEOGRAPHY 数据类型，结果以米为单位。使用 SRID 4326 GEOMETRY 数据类型时，结果以度为单位，无法转换为米，因为一度的距离取决于地球几何体的位置。

GEOGRAPHY 数据类型的计算主要用于现实圆形地球计算，如一个国家的无误差精确面积。但是，它们的计算成本要高得多。因此，ST_Transform 可以将坐标转换为适当的局部投影坐标系，实现更快的 GEOMETRY 数据类型计算。

使用空间数据，您可以运行查询以执行以下操作：

- 找出两点之间的距离。
- 检查一个区域 (多边形) 是否包含另一个区域。
- 检查一条线串是否与另一条线串或多边形相交。

您可以使用 GEOMETRY 数据类型来保存空间数据的值。Amazon Redshift 中的 GEOMETRY 值可以定义二维 (2D)、三维 (3DZ)、带度量的二维 (3DM) 和四维 (4D) 几何体基元数据类型：

- 二维 (2D) 几何体由平面中的两个笛卡尔坐标 (x, y) 指定。
- 二维 (2D) 几何体由空间中的三个笛卡尔坐标 (x, y, z) 指定。
- 带测量 (3DM) 的二维几何体由三个坐标 (x, y, m) 指定，其中前两个坐标是平面中的笛卡尔坐标，第三个是测量值。
- 四维 (4D) 几何体由四个坐标 (x, y, z, m) 指定，其中前三个坐标是空间中的笛卡尔坐标，第四个是测量值。

有关几何体基元数据类型的更多信息，请参阅 Wikipedia 中的几何体的已知文本表示。

您可以使用 GEOGRAPHY 数据类型来保存空间数据的值。Amazon Redshift 中的 GEOGRAPHY 值可以定义二维 (2D)、三维 (3DZ)、带度量的二维 (3DM) 和四维 (4D) 几何体基元数据类型：

- 二维 (2D) 几何体由球体上的经纬度坐标指定。
- 三维 (3DZ) 几何体由球体上的经纬度和高度坐标指定。
- 带测量的二维 (3DM) 几何体由三个坐标（经度、纬度、度量）指定，其中前两个坐标是球面角坐标，第三个是测量值。
- 四维 (4D) 几何体由四个坐标（经度、纬度、高度、度量）指定，其中前三个坐标是经度、纬度和高度，第四个是测量值。

有关地理坐标系的更多信息，请参阅维基百科中的地理坐标系和球坐标系。

GEOMETRY 和 GEOGRAPHY 数据类型具有以下子类型：

- POINT
- LINESTRING
- POLYGON
- MULTIPOINT
- MULTILINESTRING

- MULTIPOLYGON
- GEOMETRYCOLLECTION

有一些 Amazon Redshift SQL 函数支持以下几何数据的表示形式：

- GeoJSON
- 已知文本 (WKT)
- 扩展的已知文本 (EWKT)
- 已知二进制 (WKB) 表示
- 扩展的已知二进制文件 (EWKB)

您可以在 GEOMETRY 和 GEOGRAPHY 数据类型之间转换。

以下 SQL 将线串从 GEOMETRY 转换到 GEOGRAPHY。

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)')::geography);
```

```
 st_asewkt
---------------------------------------------
 SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

以下 SQL 将线串从 GEOMETRY 转换到 GEOGRAPHY。

```
SELECT ST_AsEWKT(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)')::geometry);
```

```
 st_asewkt
---------------------------------------------
 SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

Amazon Redshift 提供了许多 SQL 函数来查询空间数据。除了 ST_IsValid 函数之外，接受 GEOMETRY 对象作为参数的空间函数期望该 GEOMETRY 对象是有效的几何体。如果 GEOMETRY 或 GEOGRAPHY 对象无效，则空间函数的行为未定义。有关有效性的更多信息，请参阅几何有效性。

有关用于查询空间数据的 SQL 函数的详细信息，请参阅空间函数。

有关加载空间数据的详细信息，请参阅加载 GEOMETRY 或 GEOGRAPHY 数据类型的列。

主题

教程：将空间 SQL 函数与 Amazon Redshift 配合使用

本教程演示了如何在 Amazon Redshift 中使用某些空间 SQL 函数。

为此，请使用空间 SQL 函数查询两个表。本教程使用公共数据集中的数据，这些数据集将租赁住宿的位置数据与德国柏林的邮政编码相关联。

主题

先决条件

在此教程中，您需要以下资源：

- 您可以访问和更新的现有 Amazon Redshift 集群和数据库。在现有集群中，您可以创建表、加载示例数据并运行 SQL 查询以演示空间函数。您的集群应拥有至少两个节点。要了解如何创建集群，请按照 Amazon Redshift 入门指南中的步骤操作。
- 要使用 Amazon Redshift 查询编辑器，请确保您的集群位于支持查询编辑器的 AWS 区域。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 使用查询编辑器查询数据库。
- Amazon Redshift 集群的 AWS 凭证，以允许其从 Amazon S3 加载测试数据。有关如何访问其他 AWS 服务（如 Amazon S3）的信息，请参阅授权 Amazon Redshift 访问 AWS 服务。
- 名为 `mySpatialDemoRole` 的 AWS Identity and Access Management (IAM) 角色，它具有附加的托管策略 `AmazonS3ReadOnlyAccess`，用于读取 Amazon S3 数据。要创建具有从 Amazon

S3 桶加载数据的权限的角色，请参阅《Amazon Redshift 管理指南》中的使用 IAM 角色授权 COPY、UNLOAD 和 CREATE EXTERNAL SCHEMA 操作。

- 创建 IAM 角色 mySpatialDemoRole 后，该角色需要与您的 Amazon Redshift 集群关联。有关如何创建该关联的更多信息，请参阅《Amazon Redshift 管理指南》中的使用 IAM 角色授权 COPY、UNLOAD 和 CREATE EXTERNAL SCHEMA 操作。

步骤 1：创建表并加载测试数据

本教程使用的源数据是名为 accommodations.csv 和 zipcodes.csv 的文件。

accommodations.csv 文件是来自 insideairbnb.com 的开源数据。zipcodes.csv 文件提供的邮政编码是德国柏林-勃兰登堡国家统计研究所 (Amt für Statistik Berlin-Brandenburg) 的开源数据。这两个数据来源均根据知识共享许可证提供。数据仅限于德国柏林区域。这些文件位于 Amazon S3 公有桶中，可用于本教程。

您可以选择从以下 Amazon S3 链接下载源数据：

- accommodations 表的源数据。
- zipcode 表的源数据。

使用以下过程创建表并加载测试数据。

要创建表并加载测试数据

1. 打开 Amazon Redshift 查询编辑器。有关使用查询编辑器的更多信息，请参阅《Amazon Redshift 管理指南》中的使用查询编辑器查询数据库。
2. 删除本教程使用的任何表（如果它们已存在于数据库中）。有关更多信息，请参阅 步骤 3：清理资源。
3. 创建 accommodations 表来存储每个住宿地的地理位置（经度和纬度）、列表名称和其他业务数据。

   本教程将探索德国柏林的房间出租情况。shape 列存储住宿位置的地理点。其他列包含有关租赁的信息。

   要创建 accommodations 表，在 Amazon Redshift 查询编辑器中运行以下 SQL 语句。

```
CREATE TABLE public.accommodations (
```

```
  id INTEGER PRIMARY KEY,
  shape GEOMETRY,
  name VARCHAR(100),
  host_name VARCHAR(100),
  neighbourhood_group VARCHAR(100),
  neighbourhood VARCHAR(100),
  room_type VARCHAR(100),
  price SMALLINT,
  minimum_nights SMALLINT,
  number_of_reviews SMALLINT,
  last_review DATE,
  reviews_per_month NUMERIC(8,2),
  calculated_host_listings_count SMALLINT,
  availability_365 SMALLINT
);
```

4. 在查询编辑器中创建 zipcode 表来存储柏林邮政编码。

   邮政编码在 wkb_geometry 列中被定义为多边形。其余列描述了有关邮政编码的其他空间元数据。

   要创建 zipcode 表，在 Amazon Redshift 查询编辑器中运行以下 SQL 语句。

```
CREATE TABLE public.zipcode (
  ogc_field INTEGER PRIMARY KEY NOT NULL,
  wkb_geometry GEOMETRY,
  gml_id VARCHAR(256),
  spatial_name VARCHAR(256),
  spatial_alias VARCHAR(256),
  spatial_type VARCHAR(256)
 );
```

5. 为这些表加载示例数据。

   本教程的示例数据位于为所有经过身份验证的 AWS 用户提供读取访问权限的 Amazon S3 桶中。确保您提供了有效的 AWS 凭证，以允许访问 Amazon S3 。

   要将测试数据加载到表中，请运行以下 COPY 命令。将 *account-number* 替换为您自己的 AWS 账号。用单引号引起来的凭证字符串的区段不能包含任何空格或换行符。

```
COPY public.accommodations
FROM 's3://redshift-downloads/spatial-data/accommodations.csv'
DELIMITER ';'
```

```
IGNOREHEADER 1 REGION 'us-east-1'
CREDENTIALS 'aws_iam_role=arn:aws:iam::account-number:role/mySpatialDemoRole';
```

```
COPY public.zipcode
FROM 's3://redshift-downloads/spatial-data/zipcode.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
CREDENTIALS 'aws_iam_role=arn:aws:iam::account-number:role/mySpatialDemoRole';
```

6. 通过运行以下命令验证每个表都已正确加载。

```
select count(*) from accommodations;
```

```
select count(*) from zipcode;
```

以下结果显示每个测试数据表的行数。

表名称	行数
住宿	22,248
邮政编码	190

步骤 2：查询空间数据

创建并加载表后，可以使用 SQL SELECT 语句对其进行查询。以下查询演示了您可以检索的一些信息。您可以编写许多其他查询，这些查询使用空间函数来满足您的需求。

要查询空间数据

1. 查询以获取存储在 accommodations 表中的列表总数，如下所示。空间参考系统是世界测地系统 (WGS) 84，它具有唯一的空间参考标识符 4326。

```
SELECT count(*) FROM public.accommodations WHERE ST_SRID(shape) = 4326;
```

```
 count
-------
```

```
 22248
```

2. 获取已知文本 (WKT) 格式的几何体对象以及一些其他属性。此外，您还可以验证此邮政编码数据是否也存储在使用空间参考 ID (SRID) 4326 的世界测地系统 (WGS) 84 中。空间数据必须存储在同一空间参考系统中才能实现互操作。

```
SELECT ogc_field, spatial_name, spatial_type, ST_SRID(wkb_geometry),
 ST_AsText(wkb_geometry)
FROM public.zipcode
ORDER BY spatial_name;
```

```
ogc_field  spatial_name  spatial_type  st_srid  st_astext
-------------------------------------------------------------
0          10115        Polygon        4326     POLYGON((...))
4          10117        Polygon        4326     POLYGON((...))
8          10119        Polygon        4326     POLYGON((...))
...
(190 rows returned)
```

3. 以 GeoJSON 格式选择柏林米特 (10117)（柏林的一个区）的面、其维度和此面中的点数。

```
SELECT ogc_field, spatial_name, ST_AsGeoJSON(wkb_geometry),
 ST_Dimension(wkb_geometry), ST_NPoints(wkb_geometry)
FROM public.zipcode
WHERE spatial_name='10117';
```

```
ogc_field  spatial_name  spatial_type
 st_dimension  st_npoint
-------------------------------------------------------------------------------------------
4          10117         {"type":"Polygon", "coordinates":[[[...]]]}    2
    331
```

4. 运行以下 SQL 命令，查看距离勃兰登堡门 500 米范围内的住宿地数量。

```
SELECT count(*)
FROM public.accommodations
WHERE ST_DistanceSphere(shape, ST_GeomFromText('POINT(13.377704 52.516431)', 4326))
 < 500;
```

```
count
------
  29
```

5. 通过运行以下查询，从存储在列为附近的住宿地数据中获取勃兰登堡门的粗略位置。

   此查询需要一个子选择。它会导致不同的计数，因为请求的位置与之前的查询不同，因为它更接近住宿地。

```
WITH poi(loc) as (
  SELECT st_astext(shape) FROM accommodations WHERE name LIKE '%brandenburg gate%'
)
SELECT count(*)
FROM accommodations a, poi p
WHERE ST_DistanceSphere(a.shape, ST_GeomFromText(p.loc, 4326)) < 500;
```

```
count
------
  60
```

6. 运行以下查询以显示勃兰登堡门周围所有住宿地的详细信息，按价格降序排列。

```
SELECT name, price, ST_AsText(shape)
FROM public.accommodations
WHERE ST_DistanceSphere(shape, ST_GeomFromText('POINT(13.377704 52.516431)', 4326))
 < 500
ORDER BY price DESC;
```

```
name                                                price  st_astext
-------------------------------------------------------------------------------------------
DUPLEX APARTMENT/PENTHOUSE in 5* LOCATION! 7583        300
 POINT(13.3826510209548 52.5159819722552)
DUPLEX-PENTHOUSE IN FIRST LOCATION! 7582              300
 POINT(13.3799997083855 52.5135918444834)
...
(29 rows returned)
```

7. 运行以下查询以检索最昂贵的住宿地及其邮政编码。

```
SELECT
  a.price, a.name, ST_AsText(a.shape),
  z.spatial_name, ST_AsText(z.wkb_geometry)
FROM accommodations a, zipcode z
WHERE price = 9000 AND ST_Within(a.shape, z.wkb_geometry);
```

```
price   name                                    st_astext
     spatial_name      st_astext
-------------------------------------------------------------------------------------------
9000    Ueber den Dächern Berlins Zentrum    POINT(13.334436985013
 52.4979779501538)    10777             POLYGON((13.3318284987227
 52.4956021172799,...
```

8. 使用子查询计算住宿的最高、最低或中位价格。

   以下查询按邮政编码列出住宿的中位价格。

```
SELECT
  a.price, a.name, ST_AsText(a.shape),
  z.spatial_name, ST_AsText(z.wkb_geometry)
FROM accommodations a, zipcode z
WHERE
  ST_Within(a.shape, z.wkb_geometry) AND
  price = (SELECT median(price) FROM accommodations)
ORDER BY a.price;
```

```
price name                              st_astext
 spatial_name   st_astext
-------------------------------------------------------------------------------------------

45    "Cozy room Berlin-Mitte"        POINT(13.3864349535358 52.5292016386514)
 10115          POLYGON((13.3658598465795 52.535659581048,...
...
(723 rows returned)
```

9. 运行以下查询以检索柏林列出的住宿数量。要查找热门点，请按邮政编码对这些热点进行分组，并按供应量排序。

```
SELECT z.spatial_name as zip, count(*) as numAccommodations
FROM public.accommodations a, public.zipcode z
WHERE ST_Within(a.shape, z.wkb_geometry)
GROUP BY zip
ORDER BY numAccommodations DESC;
```

```
zip   numaccommodations
---------------------------
10245 872
10247 832
10437 733
10115 664
...
(187 rows returned)
```

步骤 3：清理资源

只要您的集群在运行，就将继续产生费用。完成本教程后，您可以删除您的示例集群。

如果您希望保留集群，但恢复测试数据表使用的存储空间，请执行以下命令以删除表。

```
drop table public.accommodations cascade;
```

```
drop table public.zipcode cascade;
```

将 shapefile 加载到 Amazon Redshift

您可以使用 COPY 命令将存储在 Amazon S3 中的 Esri shapefile 摄取到 Amazon Redshift 表中。shapefile 以向量格式存储地理要素的几何位置和属性信息。shapefile 格式可以在空间上描述空间对象，如点、线和面。有关 shapefile 的更多信息，请参阅 Wikipedia 中的 Shapefile。

COPY 命令支持数据格式参数 SHAPEFILE。预设情况下，shapefile 的第一列是 GEOMETRY 或 IDENTITY 列。所有后续列都遵循 shapefile 中指定的顺序。但是，目标表不需要位于此精确布局中，因为您可以使用 COPY 列映射来定义顺序。有关 COPY 命令 shapefile 支持的信息，请参阅SHAPEFILE。

在某些情况下，生成的几何体大小可能大于在 Amazon Redshift 中存储几何体的最大值。如果是这样，您可以使用 COPY 选项 SIMPLIFY 或 SIMPLIFY AUTO 来简化摄入过程中的几何体，如下所示：

- 指定 SIMPLIFY *tolerance* 以使用 Ramer-Douglas-Peucker 算法和给定的容差简化摄入过程中的所有几何体。
- 指定没有容差的 SIMPLIFY AUTO，以便仅使用 Ramer-Douglas-Peucker 算法简化大于最大尺寸的几何体。此方法可计算足够大的最小容差，以便在最大大小限制范围内存储对象。
- 指定 SIMPLIFY AUTO *max_tolerance*，以便仅使用 Ramer-Douglas-Peucker 算法和自动计算的容差简化大于最大尺寸的几何体。此方法可确保该容差不超过最大容差。

有关 GEOMETRY 数据值的最大大小的信息，请参阅将空间数据与 Amazon Redshift 一起使用时的注意事项。

在某些情况下，容差足够低，记录不能缩小到 GEOMETRY 数据值的最大大小以下。在这种情况下，您可以使用 COPY 命令的 MAXERROR 选项忽略所有或高达一定数量的摄入错误。

COPY 命令还支持加载 GZIP shapefile。为此，请指定 COPY GZIP 参数。使用此选项，所有 shapefile 组件必须独立压缩并共享相同的压缩后缀。

如果 shapefile 中存在投影描述文件 (.prj)，Redshift 会使用它来确定空间参考系统 ID (SRID)。如果 SRID 有效，则生成的几何体将分配此 SRID。如果与输入几何相关的 SRID 值不存在，则生成的几何体的 SRID 值为零。通过将 SET read_srid_on_shapefile_ingestion 的状态设为 OFF，您可以在会话级别禁用空间参考系统 ID 的自动检测。

查询 SYS_SPATIAL_SIMPLIFY 或 SVL_SPATIAL_SIMPLIFY 系统视图，以查看已简化的记录以及计算出的容差。当您指定 SIMPLIFY *tolerance* 时，此视图包含每个 COPY 操作的记录。否则，它将包含每个简化几何体的记录。有关更多信息，请参阅SYS_SPATIAL_SIMPLIFY或SVL_SPATIAL_SIMPLIFY。

有关加载 shapefile 的示例，请参阅将 shapefile 加载到 Amazon Redshift。

Amazon Redshift 空间数据的术语

以下术语用于描述某些 Amazon Redshift 空间函数。

边界框

几何体或地理的边界框被定义为几何体或地理中所有点坐标范围的交叉乘积（跨维度）。对于二维几何体，边界框是一个矩形，完全包含几何体中的所有点。例如，多边形 POLYGON((0 0,1 0,0 2,0 0)) 的边界框是由点 (0, 0) 和 (1, 2) 定义其左下角和右上角的矩形。Amazon Redshift 在几何体内预先计算并存储边界框，以加快几何谓词和空间连接的速度。例如，如果两个几何体的边界框不相交，则这两个几何体不能相交，并且它们不能位于使用 ST_Intersects 谓语的空间连接的结果集中。

您可以使用空间函数添加 (AddBBox)、删除 (DropBBox)，并确定对边界框的支持 (SupportsBBox)。Amazon Redshift 支持所有几何体子类型的边界框的预计算。

以下示例显示了如何更新表中的现有几何体以使用边界框存储它们。如果您的集群处于集群版本 1.0.26809 或更高版本，则预设情况下会使用预先计算的边界框创建所有新几何体。

```
UPDATE my_table SET geom = AddBBox(geom) WHERE SupportsBBox(geom) = false;
```

更新现有几何体后，建议您对更新后的表运行 VACUUM 命令。有关更多信息，请参阅 VACUUM。

要设置会话期间是否使用边界框对几何体进行编码，请参阅default_geometry_encoding。

几何有效性

Amazon Redshift 使用的几何算法假定输入几何体是有效的几何体。如果算法的输入无效，则结果是未定义的。以下部分介绍 Amazon Redshift 为每个几何体子类型使用的几何有效性定义。

Point

如果满足以下任意条件，则认为点为有效：

- 该点是空点。
- 所有点坐标都是有限的浮点数。

点可以是空点。

线串

如果满足以下任意条件，则线串视为有效：

- 线串为空；也就是说，它不包含点。
- 非空线串中的所有点都具有包含有限浮点数的坐标。
- 线串如果不为空，则必须是一维的；也就是说，它不能退化为点。

线串不能包含空点。

线串可以具有重复的连续点。

线串可以具有自交叉点。

面

如果满足以下任意条件，则面被视为有效：

- 面为空；也就是说，它不包含环。
- 如果面不为空，则在满足以下任意条件时有效：
  - 面的所有环都有效。如果满足以下所有条件，则环被视为有效：
    - 环的所有点都具有包含有限浮点数的坐标。
    - 环闭合；也就是说，它的第一个点和最后一个点重合。
    - 环没有任何自交叉点。
    - 环是二维的。
  - 面的环具有一致的方向。也就是说，如果您遍历任何环，则面的内部位于您的右侧或左侧。这意味着，如果面的外环是顺时针方向或逆时针方向，则面的所有内环必须具有相同的逆时针方向或顺时针方向。
  - 所有内部环必须位于面的外部环内。
  - 内部环不能嵌套；也就是说，内部环不能位于另一个内部环内。
  - 内部环和外部环只能在有限数量的点上相交。
  - 必须简单地连接面的内部。

面不能包含空点。

多点

如果满足以下任意条件，则多点被视为有效：

- 多点为空；也就是说，它不包含点。
- 多点不为空，根据点有效性定义，所有点均有效。

多点可包含一个或多个空点。

多点可以具有重复的点。

多线串

如果满足以下任意条件，则多线串被视为有效：

- 多线串为空；也就是说，它不包含线串。
- 根据线串有效性定义，非空多线串中的所有线串都有效。

只包含空线串的非空多线串被认为是有效的。

多线串中的空线串不会影响其有效性。

多线串可以具有包含重复连续点的线串。

多线串可以具有自交叉点。

多线串不能包含空点。

多面

如果满足以下任意条件，则多面被视为有效：

- 多面不包含任何面（它为空）。
- 多面不是空的，且以下所有条件均成立：
  - 多面中的所有面都有效。
  - 多面中任何两个面都没有在无限数量的点上相交。特别是，这意味着任何两个面的内部都不能相交，并且它们只能接触到有限数量的点。

多面中的空面不会使多面失效。

多面不能包含空点。

几何体集合

如果满足以下任意条件，则几何体集合被视为有效：

- 几何体集合为空；也就是说，它不包含任何几何体。
- 非空几何体集合中的所有几何体都有效。

尽管采用递归方式，该定义仍然适用于嵌套几何体集合。

几何体集合可以包含空点和带有空点的多点。

几何简单性

Amazon Redshift 使用的几何算法假定输入几何体是有效的几何体。如果算法的输入无效，则简单性检查是未定义的。以下部分介绍 Amazon Redshift 为每个几何体子类型使用的几何有效性定义。

Point

如果满足以下任意条件，则有效点被视为简单：

- 一个有效的点始终被认为是简单的。
- 一个空点被认为是简单的。

线串

如果满足以下任意条件，则有效线串被视为简单：

- 线串为空。
- 线串不为空，且满足以下所有条件：
  - 它没有重复的连续点。
  - 它没有自交叉点，除了可能的第一点和最后一点之外，它可以重合。换句话说，除边界点之外，线串不能有自交点。

面

如果有效的面不包含任何重复的连续点，则视为简单面。

多点

如果满足以下任意条件，则有效多点被视为简单：

- 多点为空；也就是说，它不包含点。
- 多点中的任意两个非空点均不重合。

Multilinestring

如果满足以下任意条件，则有效多线串被视为简单：

- 多线串为空。
- 多线串不为空，且满足以下所有条件：
  - 它的所有线串都是简单的。
  - 多线串的任何两个线串都不相交，除非在作为两个线串边界点的点处。

仅由空线串组成的非空多线串被视为空。

多线串中的空线串不会影响其简单性。

多线串中的闭合线串不能与多线串中的任何其他线串相交。

多线串不能具有包含重复连续点的线串。

多面

如果有效的多面不包含任何重复的连续点，则该多面被认为是简单的。

几何体集合

如果满足以下任意条件，则将有效的几何体集合视为简单：

- 几何体集合为空；也就是说，它不包含任何几何体。
- 非空几何体集合中的所有几何体都是简单的。

尽管采用递归方式，该定义仍然适用于嵌套几何体集合。

H3

H3 是一种分层地理空间索引网格系统，它提供了一种将空间坐标索引到平方米分辨率的方法。索引数据可以跨不同的数据集进行联接，并以不同的精度进行聚合。H3 支持一系列基于网格的算法和优化，包括最近邻、最短路径、梯度平滑等。H3 索引指的是可以是六边形或五边形的单元格。根据分辨率，空间按层次细分。H3 支持 16 种分辨率，从 0 到 15（含）。0 表示最粗糙，15 表示最精细。

Amazon Redshift 提供以下 H3 空间函数：

- H3_FromLongLat
- H3_FromPoint
- H3_Polyfill

将空间数据与 Amazon Redshift 一起使用时的注意事项

以下是将空间数据与 Amazon Redshift 一起使用时的注意事项：

- GEOMETRY 或 GEOGRAPHY 对象的最大大小为 1048447 字节。
- Amazon Redshift Spectrum 本身不支持空间数据。因此，您不能创建或更改具有 GEOMETRY 或 GEOGRAPHY 列的外部表。
- Python 用户定义的函数 (UDF) 的数据类型不支持 GEOMETRY 或 GEOGRAPHY 数据类型。
- 您不能使用 GEOMETRY 或 GEOGRAPHY 列作为 Amazon Redshift 表的排序键或分配键。
- 您不能使用 SQL ORDER BY、GROUP BY 或 DISTINCT 子句中的 GEOMETRY 或 GEOGRAPHY 列。
- 您不能使用许多 SQL 函数中的 GEOMETRY 或 GEOGRAPHY 列。

- 您不能对每种格式的 GEOMETRY 或 GEOGRAPHY 列执行 UNLOAD 操作。您可以对 GEOMETRY 或 GEOGRAPHY 列执行 UNLOAD 操作，使其转换为文本或逗号分隔值 (CSV) 文件。执行此操作将以十六进制 EWKB 格式写入 GEOMETRY 或 GEOGRAPHY 数据。如果 EWKB 数据的大小大于 4 MB，则会出现警告，因为以后无法将数据加载到表中。
- GEOMETRY 或 GEOGRAPHY 数据支持的压缩编码是 RAW。
- 使用 JDBC 或 ODBC 驱动程序时，请使用自定义类型映射。在这种情况下，客户端应用程序必须具有有关 ResultSet 对象的哪些参数是 GEOMETRY 或 GEOGRAPHY 对象的信息。ResultSetMetadata 操作返回类型 VARCHAR。
- 要从 SHAPEFILE 复制地理日期，首先摄取 GEOMETRY 列，然后将对象转换为 GEOGRAPHY 对象。

以下非空间函数可以接受 GEOMETRY 或 GEOGRAPHY 类型的输入，或 GEOMETRY 或 GEOGRAPHY 类型的列：

- 聚合函数 COUNT
- 条件表达式 COALESCE 和 NVL
- CASE 表达式
- GEOMETRY 与 GEOGRAPHY 的默认编码为 RAW。有关更多信息，请参阅 压缩编码。

在 Amazon Redshift 中使用联合查询来查询数据

通过在 Amazon Redshift 中使用联合查询，您可以跨操作数据库、数据仓库和数据湖查询和分析数据。利用联合查询功能，您可以将来自 Amazon Redshift 的对外部数据库中的实时数据的查询与跨 Amazon Redshift 和 Amazon S3 环境的查询相结合。联合查询可以使用 Amazon RDS for PostgreSQL、Amazon Aurora PostgreSQL 兼容版本、Amazon RDS for MySQL 和 Amazon Aurora MySQL 兼容版本中的外部数据库。

可以使用联合查询将实时数据整合到业务情报 (BI) 和报告应用程序中。例如，要使 Amazon Redshift 能够更轻松地摄入数据，您可以使用联合查询来执行以下操作：

- 直接查询操作数据库。
- 快速应用转换。
- 将数据加载到目标表中，而无需复杂的提取、转换、加载 (ETL) 管道。

为了减少网络上的数据移动并提高性能，Amazon Redshift 将联合查询的计算部分直接分发到远程操作数据库中。Amazon Redshift 还根据需要使用其并行处理能力来支持运行这些查询。

在运行联合查询时，Amazon Redshift 首先从领导节点建立与 RDS 或 Aurora DB 集群数据库实例的客户端连接来检索表元数据。从计算节点中，Amazon Redshift 使用下推谓词发出子查询并检索结果行。然后，Amazon Redshift 在计算节点之间分配结果行以供进一步处理。

有关发送到 Amazon Aurora PostgreSQL 数据库或 Amazon RDS for PostgreSQL 数据库的查询的详细信息将记录在系统视图 SVL_FEDERATED_QUERY 中。

主题

开始使用对 PostgreSQL 的联合查询

要创建联合查询，请遵循以下常规方法：

1. 设置 Amazon Redshift 集群与 Amazon RDS 或 Aurora PostgreSQL 数据库实例的连接。

   为此，请确保 RDS PostgreSQL 或 Aurora PostgreSQL 数据库实例可以接受来自 Amazon Redshift 集群的连接。我们建议您的 Amazon Redshift 集群和 Amazon RDS 或 Aurora PostgreSQL 实例位于相同的 Virtual Private Cloud (VPC) 和子网组中。这样一来，您就可以将 Amazon Redshift 集群的安全组添加到 RDS 或 Aurora PostgreSQL 数据库实例的安全组的入站规则中。

   您还可以设置 VPC 对等连接或其他联网来允许 Amazon Redshift 与 RDS 或 Aurora PostgreSQL 实例建立连接。有关 VPC 联网的更多信息，请参阅以下内容。

   - 《Amazon VPC 对等连接指南》中的什么是 VPC 对等连接？
   - 《Amazon RDS 用户指南》中的在 VPC 中使用数据库实例

   > **Note**
   >
   > 在某些情况下，您必须启用增强型 VPC 路由：例如，如果您的 Amazon Redshift 集群与 RDS 或 Aurora PostgreSQL 实例位于不同的 VPC，或者如果它们位于相同 VPC 而您的路由需要增强型路由。否则，运行联合查询时可能会收到超时错误。

2. 在 AWS Secrets Manager 中为 RDS PostgreSQL 和 Aurora PostgreSQL 数据库设置密钥。然后，在 AWS Identity and Access Management (IAM) 访问策略和角色中引用密钥。有关更多信息，请参阅 创建密钥和 IAM 角色以使用联合查询。

   > **Note**
   >
   > 如果您的集群使用增强型 VPC 路由，您可能需要为 AWS Secrets Manager 配置接口 VPC 端点。当您的 Amazon Redshift 集群的 VPC 和子网无权访问公有 AWS Secrets Manager 端点时，需要执行此操作。当您使用 VPC 接口端点时，VPC 中的 Amazon Redshift 集群和 AWS Secrets Manager 之间的通信将从 VPC 私密路由到端点接口。有关更多信息，请参阅《Amazon VPC 用户指南》中的创建接口端点。

3. 将之前创建的 IAM 角色应用于 Amazon Redshift 集群。有关更多信息，请参阅 创建密钥和 IAM 角色以使用联合查询。
4. 使用外部 schema 连接到 RDS PostgreSQL 和 Aurora PostgreSQL 数据库。有关更多信息，请参阅 CREATE EXTERNAL SCHEMA。有关如何使用联合查询的示例，请参阅使用联合查询的示例。

5. 运行您的引用外部 schema 的 SQL 查询，该 schema 引用了 RDS PostgreSQL 和 Aurora PostgreSQL 数据库。

开始使用 AWS CloudFormation 联合查询 PostgreSQL

您可以使用联合查询，以跨操作数据库进行查询。在本入门指南中，您可以使用示例 AWS CloudFormation 堆栈自动进行设置，以启用从 Amazon Redshift 集群到 Aurora PostgreSQL 无服务器数据库的联合查询。无需运行 SQL 语句，即可快速启动并运行，从而配置资源。

堆栈创建了一个外部架构，引用了 Aurora PostgreSQL 实例，其中包括带示例数据的表。您可以从 Redshift 集群中查询外部架构中的表。

相反，如果您想通过运行 SQL 语句来设置外部架构，以便开始使用联合查询，而不使用 CloudFormation，请参阅开始使用对 PostgreSQL 的联合查询。

运行 CloudFormation 堆栈进行联合查询之前，请确保您拥有已开启数据 API 的 Amazon Aurora PostgreSQL 兼容版无服务器数据库。您可以在数据库属性中打开数据 API。如果找不到设置，请仔细检查是否正在运行 Aurora PostgreSQL 的无服务器实例。另外，请确保您有一个使用 RA3 节点的 Amazon Redshift 集群。我们建议 Redshift 集群和无服务器 Aurora PostgreSQL 实例位于相同的 Virtual Private Cloud (VPC) 和子网组中。这样一来，您就可以将 Amazon Redshift 集群的安全组添加到 Aurora PostgreSQL 数据库实例的安全组的入站规则中。

有关开始设置 Amazon Redshift 集群的更多信息，请参阅 Amazon Redshift 预置集群。有关使用 CloudFormation 设置资源的更多信息，请参阅 AWS CloudFormation 是什么？。有关设置 Aurora DB 集群数据库的更多信息，请参阅创建 Aurora DB cluster Serverless v1 数据库集群。

为 Redshift 联合查询启动 CloudFormation 堆栈

使用以下过程启动适用于 Amazon Redshift 的 CloudFormation 堆栈，以启用联合查询。执行此操作之前，请确保您已设置 Amazon Redshift 集群和无服务器 Aurora PostgreSQL 实例。

启动 CloudFormation 堆栈，以进行联合查询

1. 在这里单击启动 CFN 堆栈，以启动 AWS Management Console 中的 CloudFormation 服务。

   如果出现登录提示，请登录。

   堆栈创建过程启动，引用存储在 Amazon S3 中的 CloudFormation 模板文件。CloudFormation 模板指的是声明组建堆栈的 AWS 资源的 JSON 格式文本文件。
2. 选择下一步，输入堆栈详细信息。

3. 在集群参数下，请输入以下内容：

   - Amazon Redshift 集群名称，例如 **ra3-consumer-cluster**
   - 特定的数据库名称，例如 **dev**
   - 数据库用户名，例如 **consumeruser**

   同时输入 Aurora DB 集群数据库参数，包括用户、数据库名称、端口和端点。我们建议使用测试集群以及测试无服务器数据库，因为堆栈会创建多个数据库对象。

   选择下一步。

   此时将显示堆栈选项。
4. 选择下一步以接受默认设置。
5. 在功能下，选择我确认 AWS CloudFormation 可能会创建 IAM 资源。
6. 选择创建堆栈。

选择创建堆栈。CloudFormation 配置模板资源（大约需要 10 分钟）并创建外部架构。

如果创建堆栈时发生错误，请执行以下操作：

- 查看 CloudFormation 事件选项卡，以获取可以帮助解决错误的信息。
- 确保您输入了正确的 Redshift 集群名称、数据库名称和数据库用户名。同时检查 Aurora PostgreSQL 实例的参数。
- 确保您的集群有 RA3 节点。
- 确保您的数据库和 Redshift 集群位于同一子网和安全组中。

从外部架构中查询数据

要使用以下程序，请确保您拥有在所述集群和数据库运行查询所需的权限。

要通过联合查询查询外部数据库

1. 使用客户端工具（如 Redshift 查询编辑器）连接创建堆栈时输入的 Redshift 数据库。
2. 查询堆栈创建的外部架构。

```
select * from svv_external_schemas;
```

SVV_EXTERNAL_SCHEMAS 视图返回有关可用外部架构的信息。在这种情况下，返回堆栈创建的外部架构，myfederated_schema。如有任何设置，可能还会返回其它外部架构。该视图同时返回该架构的关联数据库。数据库是您在创建堆栈时输入的 Aurora DB 集群数据库。堆栈将一个名为 category 的表，以及另一个名为 sales 的表添加到 Aurora DB 集群数据库中。

3. 在引用 Aurora PostgreSQL 数据库的外部架构中对表运行 SQL 查询。以下示例显示了一个查询。

```
SELECT count(*) FROM myfederated_schema.category;
```

category 表返回了几条记录。您也可以从 sales 表返回记录。

```
SELECT count(*) FROM myfederated_schema.sales;
```

有关更多示例，请参阅 使用联合查询的示例。

开始使用对 MySQL 的联合查询

要创建对 MySQL 数据库的联合查询，请遵循以下常规方法：

1. 设置 Amazon Redshift 集群与 Amazon RDS 或 Aurora MySQL 数据库实例的连接。

   为此，请确保 RDS MySQL 或 Aurora MySQL 数据库实例可以接受来自 Amazon Redshift 集群的连接。我们建议您的 Amazon Redshift 集群和 Amazon RDS 或 Aurora MySQL 实例位于相同的 Virtual Private Cloud (VPC) 和子网组中。这样一来，您就可以将 Amazon Redshift 集群的安全组添加到 RDS 或 Aurora MySQL 数据库实例的安全组的入站规则中。

   您还可以设置 VPC 对等连接或其他联网来允许 Amazon Redshift 与 RDS 或 Aurora MySQL 实例建立连接。有关 VPC 联网的更多信息，请参阅以下内容。

   - 《Amazon VPC 对等连接指南》中的什么是 VPC 对等连接？
   - 《Amazon RDS 用户指南》中的在 VPC 中使用数据库实例

   > **Note**
   >
   > 如果您的 Amazon Redshift 集群位于与您的 RDS 或 Aurora MySQL 实例不同的 VPC 中，则启用增强型 VPC 路由。否则，运行联合查询时可能会收到超时错误。

2. 在 AWS Secrets Manager 中为您的 RDS MySQL 和 Aurora MySQL 数据库设置密钥。然后，在 AWS Identity and Access Management (IAM) 访问策略和角色中引用密钥。有关更多信息，请参阅创建密钥和 IAM 角色以使用联合查询。

   > **Note**
   >
   > 如果您的集群使用增强型 VPC 路由，您可能需要为 AWS Secrets Manager 配置接口 VPC 端点。当您的 Amazon Redshift 集群的 VPC 和子网无权访问公有 AWS Secrets Manager 端点时，需要执行此操作。当您使用 VPC 接口端点时，VPC 中的 Amazon Redshift 集群和 AWS Secrets Manager 之间的通信将从 VPC 私密路由到端点接口。有关更多信息，请参阅《Amazon VPC 用户指南》中的创建接口端点。

3. 将之前创建的 IAM 角色应用于 Amazon Redshift 集群。有关更多信息，请参阅 创建密钥和 IAM 角色以使用联合查询。
4. 使用外部 schema 连接到 RDS MySQL 和 Aurora MySQL 数据库。有关更多信息，请参阅 CREATE EXTERNAL SCHEMA。有关如何使用联合查询的示例，请参阅使用联合查询 与 MySQL 的示例。
5. 运行您的引用外部 schema 的 SQL 查询，该 schema 引用了 RDS MySQL 和 Aurora MySQL 数据库。

创建密钥和 IAM 角色以使用联合查询

以下步骤演示如何创建密钥和 IAM 角色以便与联合查询一起使用。

先决条件

请确保您具有以下先决条件来创建密钥和 IAM 角色，以便与联合查询一起使用：

- 具有用户名和密码身份验证的 RDS PostgreSQL、Aurora PostgreSQL 数据库实例、RDS MySQL 或 Aurora MySQL 数据库实例。
- 具有支持联合查询的集群维护版本的 Amazon Redshift 集群。

使用 AWS Secrets Manager 创建密钥（用户名和密码）

1. 使用拥有您的 RDS 或 Aurora DB 集群实例的账户登录 Secrets Manager 控制台。
2. 选择存储新密钥。

3. 选择RDS 数据库凭证磁贴。对于用户名和密码，请输入实例的值。确认或选择加密密钥的值。然后，选择您的密钥将访问的 RDS 数据库。

> **Note**
>
> 我们建议使用默认加密密钥 (DefaultEncryptionKey)。如果您使用自定义加密密钥，则必须将用于访问密钥的 IAM 角色添加为密钥用户。

4. 输入密钥的名称，使用默认选项继续执行创建步骤，然后选择存储。
5. 查看您的密钥，并记下您为标识密钥而创建的密钥 ARN 值。

使用密钥创建安全策略

1. 登录AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 使用 JSON 创建策略类似于以下内容。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

要检索密钥，您需要列出和读取操作。我们建议您将资源限制为您创建的特定密钥。为此，请使用密钥的 Amazon 资源名称 (ARN) 来限制资源。您还可以使用 IAM 控制台上的可视化编辑器指定权限和资源。

3. 为策略指定一个名称并完成创建。
4. 导航到IAM 角色。
5. 为 Redshift - 可自定义创建 IAM 角色。
6. 将您刚创建的 IAM 策略附加到现有 IAM 角色，或者创建新的 IAM 角色并附加策略。
7. 在 IAM 角色的信任关系选项卡上，确认该角色包含信任实体 `redshift.amazonaws.com`。
8. 记下您创建的角色 ARN。此 ARN 有权访问密钥。

要将 IAM 角色附加到您的 Amazon Redshift 集群

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群。这将列出您的账户在当前 AWS 区域中的集群。
3. 在列表中选择集群名称可查看有关集群的更多详细信息。
4. 对于操作，请选择管理 IAM 角色。这将显示管理 IAM 角色页面。
5. 将您的 IAM 角色添加到集群。

使用联合查询的示例

以下示例显示了如何运行联合查询。使用连接到 Amazon Redshift 数据库的 SQL 客户端运行 SQL。

将联合查询与 PostgreSQL 结合使用的示例

以下示例演示如何设置引用 Amazon Redshift 数据库、Aurora PostgreSQL 数据库和 Amazon S3 的联合查询。此示例说明联合查询的工作原理。要在您自己的环境中运行联合查询，请对该查询进行相应更改，使其适合您的环境。有关执行此操作的先决条件，请参阅开始使用对 PostgreSQL 的联合查询。

创建引用 Aurora PostgreSQL 数据库的外部 schema。

```
CREATE EXTERNAL SCHEMA apg
```

```
FROM POSTGRES
DATABASE 'database-1' SCHEMA 'myschema'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

创建另一个引用 Amazon S3 的外部 schema，该 schema 使用 Amazon Redshift Spectrum。此外，将使用架构的权限授予 public。

```
CREATE EXTERNAL SCHEMA s3
FROM DATA CATALOG
DATABASE 'default' REGION 'us-west-2'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-S3';

GRANT USAGE ON SCHEMA s3 TO public;
```

显示 Amazon Redshift 表中的行计数。

```
SELECT count(*) FROM public.lineitem;

  count
----------
25075099
```

显示 Aurora PostgreSQL 表中的行计数。

```
SELECT count(*) FROM apg.lineitem;

count
-------
11760
```

显示 Amazon S3 中的行计数。

```
SELECT count(*) FROM s3.lineitem_1t_part;

   count
------------
6144008876
```

从 Amazon Redshift、Aurora PostgreSQL 和 Amazon S3 创建表视图。此视图用于运行联合查询。

```
CREATE VIEW lineitem_all AS
  SELECT
 l_orderkey,l_partkey,l_suppkey,l_linenumber,l_quantity,l_extendedprice,l_discount,l_tax,l_retu
        l_shipdate::date,l_commitdate::date,l_receiptdate::date,
 l_shipinstruct ,l_shipmode,l_comment
  FROM s3.lineitem_1t_part
  UNION ALL SELECT * FROM public.lineitem
  UNION ALL SELECT * FROM apg.lineitem
     with no schema binding;
```

使用谓词显示视图 `lineitem_all` 中的行计数以限制结果。

```
SELECT count(*) from lineitem_all WHERE l_quantity = 10;

   count
-----------
123373836
```

了解每年 1 月份一件商品的销量。

```
SELECT extract(year from l_shipdate) as year,
       extract(month from l_shipdate) as month,
       count(*) as orders
FROM lineitem_all
WHERE extract(month from l_shipdate) = 1
AND l_quantity < 2
GROUP BY 1,2
ORDER BY 1,2;

 year | month | orders
------+-------+---------
 1992 |     1 |  196019
 1993 |     1 | 1582034
 1994 |     1 | 1583181
 1995 |     1 | 1583919
 1996 |     1 | 1583622
 1997 |     1 | 1586541
 1998 |     1 | 1583198
 2016 |     1 |   15542
```

```
 2017 |     1 |   15414
 2018 |     1 |   15527
 2019 |     1 |     151
```

使用混合大小写名称的示例

要查询具有数据库、schema、表或列混合大小写名称的受支持 PostgreSQL 远程数据库，请将 `enable_case_sensitive_identifier` 设置为 `true`。有关设置此会话参数的更多信息，请参阅enable_case_sensitive_identifier。

```
SET enable_case_sensitive_identifier TO TRUE;
```

数据库和 schema 名称通常是小写的。以下示例说明如何连接到受支持的 PostgreSQL 远程数据库，该数据库具有数据库和 schema 的小写名称以及表和列的混合大小写名称。

创建引用具有小写数据库名称 (`dblower`) 和小写 schema 名称 (`schemalower`) 的 Aurora PostgreSQL 数据库的外部 schema。

```
CREATE EXTERNAL SCHEMA apg_lower
FROM POSTGRES
DATABASE 'dblower' SCHEMA 'schemalower'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

在运行查询的会话中，将 `enable_case_sensitive_identifier` 设置为 `true`。

```
SET enable_case_sensitive_identifier TO TRUE;
```

运行联合查询以从 PostgreSQL 数据库中选择所有数据。表 (`MixedCaseTab`) 和列 (`MixedCaseName`) 具有混合大小写的名称。结果是一行 (`Harry`)。

```
select * from apg_lower."MixedCaseTab";
```

```
 MixedCaseName
-------
 Harry
```

以下示例说明如何连接到受支持的 PostgreSQL 远程数据库，该数据库具有数据库、schema、表和列的混合大小写名称。

将 enable_case_sensitive_identifier 设置为 true，然后再创建外部 schema。如果 enable_case_sensitive_identifier 未在创建外部 schema 之前设置为 true，则会发生数据库不存在错误。

创建引用具有混合大小写数据库名称 (UpperDB) 和 schema 名称 (UpperSchema) 的 Aurora PostgreSQL 数据库的外部 schema。

```
CREATE EXTERNAL SCHEMA apg_upper
FROM POSTGRES
DATABASE 'UpperDB' SCHEMA 'UpperSchema'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

运行联合查询以从 PostgreSQL 数据库中选择所有数据。表 (MixedCaseTab) 和列 (MixedCaseName) 具有混合大小写的名称。结果是一行 (Harry)。

```
select * from apg_upper."MixedCaseTab";
```

```
 MixedCaseName
-------
 Harry
```

使用联合查询 与 MySQL 的示例

以下示例演示如何设置引用 Aurora MySQL 数据库的联合查询。此示例说明联合查询的工作原理。要在您自己的环境中运行联合查询，请对该查询进行相应更改，使其适合您的环境。有关执行此操作的先决条件，请参阅开始使用对 MySQL 的联合查询。

该示例取决于以下先决条件：

- 在 Aurora MySQL 数据库的 Secrets Manager 中设置的密钥。在 IAM 访问策略和角色中引用该密钥。有关更多信息，请参阅 创建密钥和 IAM 角色以使用联合查询。
- 设置 Amazon Redshift 和 Aurora MySQL 的链接的安全组。

创建引用 Aurora MySQL 数据库的外部 schema。

```
CREATE EXTERNAL SCHEMA amysql
FROM MYSQL
DATABASE 'functional'
URI 'endpoint to remote hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

运行一个 Aurora MySQL 表的 SQL SELECT，以在 Aurora MySQL 中显示员工表中的一行。

```
SELECT level FROM amysql.employees LIMIT 1;

 level
-------
     8
```

Amazon Redshift 与支持的 PostgreSQL 和 MySQL 数据库之间的数据类型差异

下表显示 Amazon Redshift 数据类型与相应 Amazon RDS PostgreSQL 或 Aurora PostgreSQL 数据类型的映射。

Amazon Redshift 数据类型	RDS PostgreSQL 或 Aurora PostgreSQL 数据类型	描述
SMALLINT	SMALLINT	有符号的二字节整数
INTEGER	INTEGER	有符号的四字节整数
BIGINT	BIGINT	有符号的八字节整数
DECIMAL	DECIMAL	可选精度的精确数字
REAL	REAL	单精度浮点数
DOUBLE PRECISION	DOUBLE PRECISION	双精度浮点数

Amazon Redshift 数据类型	RDS PostgreSQL 或 Aurora PostgreSQL 数据类型	描述
BOOLEAN	BOOLEAN	逻辑布尔值 (true/false)
CHAR	CHAR	固定长度字符串
VARCHAR	VARCHAR	具有用户定义的限制的可变长度字符串，
DATE	DATE	日历日期（年、月、日）
TIMESTAMP	TIMESTAMP	日期和时间（没有时区）
TIMESTAMPTZ	TIMESTAMPTZ	日期和时间（有时区）
GEOMETRY	PostGIS GEOMETRY	空间数据

在 Amazon Redshift 中，以下 RDS PostgreSQL 和 Aurora PostgreSQL 数据类型将转换为 VARCHAR(64K)：

- JSON、JSONB
- 数组
- BIT、BIT VARYING
- BYTEA
- 复合类型
- 日期和时间类型 INTERVAL、TIME、TIME WITH TIMEZONE
- 枚举类型
- 货币类型
- 网络地址类型
- 数字类型 SERIAL、BIGSERIAL、SMALLSERIAL 和 MONEY
- 对象标识符类型

- pg_lsn 类型
- 伪类型
- 范围类型
- 文本搜索类型
- TXID_SNAPSHOT
- UUID
- XML 类型

下表显示 Amazon Redshift 数据类型与相应 Amazon RDS MySQL 或 Aurora MySQL 数据类型的映射。

Amazon Redshift 数据类型	RDS MySQL 或 Aurora MySQL 数据类型	描述
BOOLEAN	TINYINT(1)	逻辑布尔值（true 或 false）
SMALLINT	TINYINT(UNSIGNED)	有符号的二字节整数
SMALLINT	SMALLINT	有符号的二字节整数
INTEGER	SMALLINT UNSIGNED	有符号的四字节整数
INTEGER	MEDIUMINT (UNSIGNED)	有符号的四字节整数
INTEGER	INT	有符号的四字节整数
BIGINT	INT UNSIGNED	有符号的八字节整数
BIGINT	BIGINT	有符号的八字节整数
DECIMAL	BIGINT UNSIGNED	可选精度的精确数字
DECIMAL	DECIMAL(M,D)	可选精度的精确数字
REAL	FLOAT	单精度浮点数
DOUBLE PRECISION	DOUBLE	双精度浮点数

Amazon Redshift 数据类型	RDS MySQL 或 Aurora MySQL 数据类型	描述
CHAR	CHAR	固定长度字符串
VARCHAR	VARCHAR	具有用户定义的限制的可变长度字符串，
DATE	DATE	日历日期（年、月、日）
TIME	TIME	时间（没有时区）
TIMESTAMP	TIMESTAMP	日期和时间（没有时区）
TIMESTAMP	DATETIME	时间（没有时区）
VARCHAR(4)	YEAR	表示年份的可变长度字符

当时间数据超出 (00:00:00 – 24:00:00) 范围时会出错。

在 Amazon Redshift 中，以下 RDS MySQL 和 Aurora MySQL 数据类型将转换为 VARCHAR(64K)：

- BIT
- BINARY
- VARBINARY
- TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB
- TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT
- ENUM
- SET
- SPATIAL

使用 Amazon Redshift 访问联合数据时的注意事项

某些 Amazon Redshift 功能不支持对联合数据进行访问。您可以在下面找到相关的限制和注意事项。

以下是将联合查询与 Amazon Redshift 结合使用时的限制和注意事项：

- 联合查询支持对外部数据来源进行读取访问。无法在外部数据来源中写入或创建数据库对象。
- 在某些情况下，您可能访问与 Amazon Redshift 不同的 AWS 区域中的 Amazon RDS 或 Aurora DB 集群数据库。在这些情况下，您通常会因跨 AWS 地区传输数据而产生网络延迟和计费费用。我们建议您将 Aurora 全局数据库用于与 Amazon Redshift 集群位于同一 AWS 区域的本地端点。Aurora 全局数据库使用专用基础设施在任意两个 AWS 区域间进行基于存储的复制，典型延迟不到 1 秒。
- 考虑访问 Amazon RDS 或 Aurora DB 集群的成本。例如，使用此功能访问 Aurora DB 集群时，Aurora DB 集群费用基于 IOPS。
- 联合查询不允许从 RDS 或 Aurora DB 集群访问 Amazon Redshift。
- 联合查询仅在同时提供 Amazon Redshift 和 Amazon RDS 或 Aurora DB 集群的 AWS 区域中可用。
- 联合查询当前不支持 `ALTER SCHEMA`。要更改 schema，请依次使用 `DROP` 和 `CREATE EXTERNAL SCHEMA`。
- 联合查询不适用于并发扩展。
- 联合查询当前不支持通过 PostgreSQL 外部数据包装程序进行访问。
- 对 RDS MySQL 或 Aurora MySQL 的联合查询支持在 READ COMMITTED 级别的事务隔离。
- 如果未指定，Amazon Redshift 会在端口 3306 上连接到 RDS for MySQL 或 Aurora MySQL。在为 MySQL 创建外部架构之前，请确认 MySQL 端口号。
- 如果未指定，Amazon Redshift 会在 5432 端口上连接到 RDS PostgreSQL 或 Aurora PostgreSQL。在为 PostgreSQL 创建外部架构之前，请确认 PostgreSQL 端口号。
- 从 MySQL 中获取 TIMESTAMP 和 DATE 数据类型时，零值被视为 NULL。
- 如果使用 Aurora DB 集群数据库读取器端点，则可能会出现“快照无效”错误。采用以下方法之一可以避免这种情况：
  - 使用特定的 Aurora DB 集群实例端点（而不是使用 Aurora DB 集群端点）。此方法对来自 PostgreSQL 数据库的结果使用 REPEATABLE READ 事务隔离。
  - 使用 Aurora DB 集群读取器端点并将会话的 `pg_federation_repeatable_read` 设置为 false。此方法对来自 PostgreSQL 数据库的结果使用 READ COMMITTED 事务隔离。有关 Aurora DB 集群读取器端点的更多信息，请参阅《Amazon Aurora 用户指南》中的 Aurora DB 集群端点类型。有关 `pg_federation_repeatable_read` 的信息，请参阅pg_federation_repeatable_read。

以下是使用对 PostgreSQL 数据库的联合查询时事务的注意事项：

- 如果查询由联合表组成，则领导节点会在远程数据库上启动 READ ONLY REPEATABLE READ 事务。此事务在 Amazon Redshift 事务的持续时间内保留。
- 领导节点通过调用 `pg_export_snapshot` 创建远程数据库快照，并对受影响的表进行读锁定。
- 计算节点启动事务，并使用在领导节点上创建的快照来向远程数据库发出查询。

支持的联合数据库版本

Amazon Redshift 外部 schema 可以引用外部 RDS PostgreSQL 或 Aurora PostgreSQL 中的数据库。当它这样做时，以下限制将适用：

- 创建引用 Aurora DB 集群的外部架构时，Aurora PostgreSQL 数据库必须为版本 9.6 或更高版本。
- 创建引用 Amazon RDS 的外部 schema 时，Amazon RDS PostgreSQL 数据库必须为版本 9.6 或更高版本。

Amazon Redshift 外部 schema 可以引用外部 RDS MySQL 或 Aurora MySQL 中的数据库。当它这样做时，以下限制将适用：

- 创建引用 Aurora DB 集群的外部架构时，Aurora MySQL 数据库必须为版本 5.6 或更高版本。
- 创建引用 Amazon RDS 的外部 schema 时，RDS MySQL 数据库必须为版本 5.6 或更高版本。

使用 Amazon Redshift Spectrum 查询外部数据

通过使用 Amazon Redshift Spectrum，您可以从 Amazon S3 中的文件有效地查询和检索结构化和半结构化数据，而不必将数据加载到 Amazon Redshift 表中。Redshift Spectrum 查询采用了大规模并行以便针对大型数据集极快地运行。很多处理发生在 Redshift Spectrum 层中，而大多数数据位于 Amazon S3 中。多个集群可同时查询 Amazon S3 上的同一数据集，而无需为每个集群复制数据。

主题

Amazon Redshift Spectrum 概览

Amazon Redshift Spectrum 驻留在独立于您的集群的专用 Amazon Redshift 服务器上。Amazon Redshift 将很多计算密集型任务（如谓词筛选和聚合）下推到 Redshift Spectrum 层。因此，Redshift Spectrum 查询使用的集群处理容量比其他查询的少得多。Redshift Spectrum 还可智能地扩展。基于您的查询的需求，Redshift Spectrum 可能能够使用数千个实例来利用大规模并行处理。

您通过定义您的文件的结构并将文件作为外部数据目录中的表注册来创建 Redshift Spectrum 表。外部数据目录可以是 AWS Glue、Amazon Athena 附带的数据目录或您自己的 Apache Hive 元存储。您可

使用数据定义语言 (DDL) 命令或使用连接到外部数据目录的任何其他工具从 Amazon Redshift 创建和管理外部表。对外部数据目录进行的更改将立即对您的任何 Amazon Redshift 集群可用。

您也可在一个或多个列上为外部表分区。将分区定义为外部表的一部分可以提高性能。提高性能的原因是，Amazon Redshift 查询优化程序消除了不包含查询数据的分区。

在定义 Redshift Spectrum 表之后，您可以像查询和联接任何其他 Amazon Redshift 表一样查询和联接这些表。Redshift Spectrum 不支持对外部表的更新操作。您可将 Redshift Spectrum 表添加到多个 Amazon Redshift 集群并在同一AWS区域的任何集群中查询 Amazon S3 上的相同数据。更新 Amazon S3 数据文件后，立即可从您的任何 Amazon Redshift 集群查询到该数据。

您访问的 AWS Glue 数据目录可能已加密以提高安全性。如果 AWS Glue 目录已加密，您需要使用 AWS Glue 的 AWS Key Management Service (AWS KMS) 密钥来访问 AWS Glue 目录。AWS Glue 目录加密并非在所有AWS区域中都可用。有关受支持的 AWS 区域的列表，请参阅 AWS Glue 开发人员指南中的 AWS Glue 的加密和安全访问。有关 AWS Glue 数据目录加密的更多信息，请参阅 AWS Glue 开发人员指南中的加密您的 AWS Glue 数据目录。

> **Note**
>
> 您无法使用用于标准 Amazon Redshift 表（如 PG_TABLE_DEF、STV_TBL_PERM、PG_CLASS 或 information_schema）的同一资源查看 Redshift Spectrum 表的详细信息。如果您的商业智能或分析工具无法识别 Redshift Spectrum 外部表，请将您的应用程序为配置查询 SVV_EXTERNAL_TABLES 和 SVV_EXTERNAL_COLUMNS。

Amazon Redshift Spectrum 区域

除区域特定文档中另有说明的外，Redshift Spectrum 已在提供 Amazon Redshift 的 AWS 区域开放。有关商业区域中 AWS 区域的可用性，请参阅《Amazon Web Services 一般参考》中 Redshift API 的服务端点。

Amazon Redshift Spectrum 注意事项

在使用 Amazon Redshift Spectrum 时，请注意以下事项：

- Amazon Redshift 集群与 Amazon S3 桶必须位于同一AWS区域。

- Redshift Spectrum 不支持预置集群的增强型 VPC 路由。要访问 Amazon S3 数据，您可能需要执行其他配置步骤。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 Redshift Spectrum 与增强型 VPC 路由。
- Redshift Spectrum 支持 Amazon S3 接入点别名。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的为您的接入点使用桶式别名。但是，Redshift Spectrum 不支持使用 Amazon S3 接入点别名的 VPC。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 Redshift Spectrum 与增强型 VPC 路由。
- 您不能对外部表执行更新或删除操作。要在指定架构中创建新的外部表，可以使用 CREATE EXTERNAL TABLE。有关 CREATE EXTERNAL TABLE 的更多信息，请参阅CREATE EXTERNAL TABLE。要将 SELECT 查询的结果插入到外部目录中的现有外部表中，可以使用 INSERT（外部表）。有关 INSERT（外部表）的更多信息，请参阅 INSERT（外部表）。
- 除非您使用的是为 AWS Lake Formation 启用的 AWS Glue Data Catalog，否则您无法控制用户对于外部表的权限。相反，您可以授予和撤销对外部 schema 的权限。有关如何使用 AWS Lake Formation 的更多信息，请参阅将 Redshift Spectrum 与 AWS Lake Formation 结合使用。
- 要运行 Redshift Spectrum 查询，数据库用户必须有权在数据库中创建临时表。以下示例将数据库 `spectrumdb` 的临时权限授予 `spectrumusers` 用户组。

  ```
  grant temp on database spectrumdb to group spectrumusers;
  ```

  有关更多信息，请参阅 GRANT。
- 使用 Athena 数据目录或 AWS Glue 数据目录作为元数据存储时，请参阅《Amazon Redshift 管理指南》中的配额和限制。
- Redshift Spectrum 不支持在 Amazon EMR 中使用 Kerberos。

Amazon Redshift Spectrum 入门

在本教程中，您将了解如何使用 Amazon Redshift Spectrum 直接从 Amazon S3 上的文件中查询数据。如果您已经有一个集群和一个 SQL 客户端，您通过极少的设置即可完成本教程。

> **Note**
>
> Redshift Spectrum 查询将产生额外的费用。本教程中运行示例查询的费用极低。有关定价的更多信息，请参阅 Amazon Redshift Spectrum 定价。

先决条件

要使用 Redshift Spectrum，您需要一个 Amazon Redshift 集群和一个连接到集群的 SQL 客户端，供您运行 SQL 命令。该集群和 Amazon S3 中的数据文件必须位于同一 AWS 区域中。

有关如何创建 Amazon Redshift 集群的信息，请参阅《Amazon Redshift 入门指南》中的Amazon Redshift 预置集群。有关连接到集群的方法的信息，请参阅《Amazon Redshift 入门指南》中的连接到 Amazon Redshift 数据仓库。

在下面的一些示例中，示例数据位于美国东部（弗吉尼亚州北部）(us-east-1)，因此您需要一个也位于 us-east-1 的集群。或者，您可以使用 Amazon S3 将以下桶和文件夹中的数据对象复制到集群所在的 AWS 区域的桶中：

- `s3://redshift-downloads/tickit/spectrum/customers/*`
- `s3://redshift-downloads/tickit/spectrum/sales_partition/*`
- `s3://redshift-downloads/tickit/spectrum/sales/*`
- `s3://redshift-downloads/tickit/spectrum/salesevent/*`

运行类似于以下内容的 Amazon S3 命令，将位于美国东部（弗吉尼亚州北部）的示例数据复制到您的 AWS 区域。在运行命令之前，请创建桶并在此桶中创建文件夹，以匹配您的 Amazon S3 copy 命令。Amazon S3 copy 命令的输出确认文件已复制到所需 AWS 区域中的 *bucket-name*。

```
aws s3 cp s3://redshift-downloads/tickit/spectrum/ s3://bucket-name/tickit/spectrum/ --
copy-props none --recursive
```

使用 AWS CloudFormation 的 Redshift Spectrum 入门

作为以下步骤的替代方法，您可以访问 Redshift Spectrum DataLake AWS CloudFormation 模板，以创建一个包含您可查询的 Amazon S3 桶的堆栈。有关更多信息，请参阅 启动您的 AWS CloudFormation 堆栈，然后在 Amazon S3 中查询您的数据。

Redshift Spectrum 入门分步指南

要开始使用 Amazon Redshift Spectrum，请执行以下步骤：

- 第 1 步 为 Amazon Redshift 创建一个 IAM 角色
- 步骤 2：将 IAM 角色与集群相关联

- 步骤 3：创建外部架构和外部表
- 步骤 4：在 Amazon S3 中查询数据

第 1 步 为 Amazon Redshift 创建一个 IAM 角色

您的集群需要授权才能访问您在 AWS Glue 或 Amazon Athena 中的外部数据目录以及您在 Amazon S3 中的数据文件。要提供授权，您需要引用附加到集群的 AWS Identity and Access Management (IAM) 角色。有关将角色用于 Amazon Redshift 的更多信息，请参阅使用 IAM 角色授权 COPY 和 UNLOAD 操作。

> **Note**
>
> 在某些情况下，您可以将 Athena Data Catalog 迁移到 AWS Glue Data Catalog。如果您的集群在支持 AWS Glue 的 AWS 区域内，并且您在 Athena Data Catalog 中拥有 Redshift Spectrum 外部表，则可执行此操作。要将 AWS Glue 数据目录用于 Redshift Spectrum，您可能需要更改您的 IAM 策略。有关更多信息，请参阅《Athena 用户指南》中的升级到 AWS Glue Data Catalog。

为 Amazon Redshift 创建角色时，请选择以下方法之一：

- 如果您将 Redshift Spectrum 与 Athena Data Catalog 或 AWS Glue Data Catalog 结合使用，请按照要为 Amazon Redshift 创建一个 IAM 角色 中概述的步骤操作。
- 如果您将 Redshift Spectrum 与为 AWS Lake Formation 启用的 AWS Glue Data Catalog 结合使用，请按照以下程序中概括的步骤操作：
  - 要使用为 AWS Lake Formation 启用的 AWS Glue Data Catalog 为 Amazon Redshift 创建 IAM 角色
  - 授予对表的 SELECT 权限以在 Lake Formation 数据库中进行查询

要为 Amazon Redshift 创建一个 IAM 角色

1. 打开 IAM 控制台。
2. 在导航窗格中，选择角色。
3. 选择创建角色。
4. 选择 AWS 服务作为可信实体，然后选择 Redshift 作为使用案例。

5. 在其他 AWS 服务的使用案例下，选择 Redshift - 可自定义，然后选择下一步。
6. 此时显示添加权限策略页面。选择 AmazonS3ReadOnlyAccess 和 AWSGlueConsoleFullAccess（如果使用的是 AWS Glue 数据目录）。或选择 AmazonAthenaFullAccess（如果使用的是 Athena Data Catalog）。选择下一步。

   > **Note**
   >
   > AmazonS3ReadOnlyAccess 策略为您的集群提供对所有 Amazon S3 桶的只读访问。要仅授予 AWS 示例数据桶的访问权限，请创建新策略并添加以下权限。
   >
   > ```
   > {
   >     "Version": "2012-10-17",
   >     "Statement": [
   >         {
   >             "Effect": "Allow",
   >             "Action": [
   >                 "s3:Get*",
   >                 "s3:List*"
   >             ],
   >             "Resource": "arn:aws:s3:::redshift-downloads/*"
   >         }
   >     ]
   > }
   > ```

7. 对于角色名称，输入您角色的名称，例如 **myspectrum_role**。
8. 检查信息，然后选择 Create role。
9. 在导航窗格中，选择角色。选择新角色的名称以查看摘要，然后将 Role ARN 复制到剪贴板。该值是您刚创建的角色的 Amazon 资源名称 (ARN)。您将在创建用于引用 Amazon S3 上的数据文件的外部表时使用此值。

要使用为 AWS Lake Formation 启用的 AWS Glue Data Catalog 为 Amazon Redshift 创建 IAM 角色

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择策略。

   如果这是您首次选择 Policies，则会显示 Welcome to Managed Policies 页面。选择开始使用。
3. 选择创建策略。
4. 选择以在 JSON 选项卡上创建策略。

5. 粘贴在以下 JSON 策略文档中，该策略授予对 Data Catalog 的访问权限，但拒绝对 Lake Formation 的管理员权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RedshiftPolicyForLF",
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "lakeformation:GetDataAccess"
            ],
            "Resource": "*"
        }
    ]
}
```

6. 完成后，选择审核对策略进行审核。策略验证程序将报告任何语法错误。
7. 在查看策略页面上，为名称输入 **myspectrum_policy**，以命名您正在创建的策略。输入描述（可选）。查看策略摘要以查看您的策略授予的权限。然后，选择创建策略以保存您的工作。

   在创建策略之后，您可以向您的用户提供访问权限。

要提供访问权限，请为您的用户、组或角色添加权限：

- AWS IAM Identity Center 中的用户和群组：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

授予对表的 SELECT 权限以在 Lake Formation 数据库中进行查询

1. 通过 https://console.aws.amazon.com/lakeformation/ 中打开 Lake Formation 控制台。
2. 在导航窗格中，选择数据湖权限，然后选择授予。
3. 按照《AWS Lake Formation 开发人员指南》的使用命名资源方法授予表权限中的说明进行操作。提供以下信息：
   - 对于 IAM 角色，选择您创建的 IAM 角色 `myspectrum_role`。运行 Amazon Redshift 查询编辑器时，它使用此 IAM 角色来获取数据权限。

   > **Note**
   >
   > 要授予对启用了 Lake Formation 的 Data Catalog 中的表的 SELECT 权限以进行查询，请执行以下操作：
   >
   > - 在 Lake Formation 中注册数据的路径。
   > - 在 Lake Formation 中授予用户对该路径的权限。
   > - 创建的表可在 Lake Formation 中注册的路径中找到。
4. 选择授权。

> **Important**
>
> 作为最佳实践，仅允许通过 Lake Formation 权限访问底层 Amazon S3 对象。要防止未经批准的访问，请删除授予针对 Lake Formation 以外的 Amazon S3 对象的任何权限。如果您在设置 Lake Formation 之前曾访问了 Amazon S3 对象，请删除之前设置的任何 IAM 策略或桶权限。有关更多信息，请参阅将 AWS Glue 数据权限升级到 AWS Lake Formation 模型和 Lake Formation 权限。

步骤 2：将 IAM 角色与集群相关联

现在，您已拥有一个 IAM 角色，该角色授权 Amazon Redshift 为您访问外部 Data Catalog 和 Amazon S3。此时，您必须将该角色与您的 Amazon Redshift 集群关联。

将 IAM 角色与集群关联

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择要更新的集群的名称。
3. 对于操作，请选择管理 IAM 角色。这将显示IAM 角色页面。
4. 选择输入 ARN，然后输入 ARN 或 IAM 角色，或从列表中选择 IAM 角色。然后，选择添加 IAM 角色以将该角色添加到已附加的 IAM 角色列表中。
5. 选择完成将 IAM 角色与集群关联。将修改集群以完成更改。

步骤 3：创建外部架构和外部表

在外部 schema 中创建外部表。外部 schema 引用了外部数据目录中的数据库并提供了 IAM 角色 ARN（代表您授权您的集群访问 Amazon S3）。您可在 Amazon Athena Data Catalog、AWS Glue Data Catalog 或 Apache Hive 元存储（如 Amazon EMR）中创建外部数据库。在此示例中，您将在创建外部 schema Amazon Redshift 时在 Amazon Athena Data Catalog 中创建外部数据库。有关更多信息，请参阅 为 Amazon Redshift Spectrum 创建外部 schema。

创建外部 schema 和外部表

1. 要创建外部架构，请将以下命令中的 IAM 角色 ARN 替换为您在步骤 1 中创建的角色 ARN。然后在 SQL 客户端中运行该命令。

```
create external schema myspectrum_schema
from data catalog
database 'myspectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
create external database if not exists;
```

2. 要创建外部表，请运行以下 CREATE EXTERNAL TABLE 命令。

> **Note**
>
> 您的集群和 Amazon S3 桶必须位于同一个 AWS 区域中。对于此示例 CREATE EXTERNAL TABLE 命令，包含示例数据的 Amazon S3 桶位于美国东部（弗吉尼亚州北部）AWS 区域中。要查看源数据，请下载 sales_ts.000 文件。

您可以修改此示例以在不同的 AWS 区域中运行。在您所需的 AWS 区域中创建 Amazon S3 桶。使用 Amazon S3 copy 命令复制销售数据。然后，将示例 CREATE EXTERNAL TABLE 命令中的 location 选项更新到您的桶。

```
aws s3 cp s3://redshift-downloads/tickit/spectrum/sales/ s3://bucket-name/
tickit/spectrum/sales/ --copy-props none --recursive
```

Amazon S3 copy 命令的输出确认文件已复制到所需 AWS 区域中的 *bucket-name*。

```
copy: s3://redshift-downloads/tickit/spectrum/sales/sales_ts.000 to
 s3://bucket-name/tickit/spectrum/sales/sales_ts.000
```

```
create external table myspectrum_schema.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/'
table properties ('numRows'='172000');
```

步骤 4：在 Amazon S3 中查询数据

在创建外部表之后，您可使用用于查询其他 Amazon Redshift 表的同一 SELECT 语句查询外部表。这些 SELECT 语句查询包括联接表、聚合数据和筛选谓词。

要在 Amazon S3 中查询数据

1. 获取 MYSPECTRUM_SCHEMA.SALES 表中的行数。

```
select count(*) from myspectrum_schema.sales;
```

```
count
------
172462
```

2. 作为最佳实践，将较大的事实数据表保存在 Amazon S3 中并将较小的维度表保存在 Amazon Redshift 中。如果您已加载了加载数据中的示例数据，您的数据库中将有一个名为 EVENT 的表。如果没有，请使用以下命令创建 EVENT 表。

```
create table event(
eventid integer not null distkey,
venueid smallint not null,
catid smallint not null,
dateid smallint not null sortkey,
eventname varchar(200),
starttime timestamp);
```

3. 通过将以下 COPY 命令中的 IAM 角色 ARN 替换为第 1 步 为 Amazon Redshift 创建一个 IAM 角色中创建的角色 ARN 来加载 EVENT 表。您可以选择从 AWS 区域 us-east-1 中的 Amazon S3 桶下载并查看 allevents_pipe.txt 的源数据。

```
copy event from 's3://redshift-downloads/tickit/allevents_pipe.txt'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
delimiter '|' timeformat 'YYYY-MM-DD HH:MI:SS' region 'us-east-1';
```

以下示例将外部 Amazon S3 表 MYSPECTRUM_SCHEMA.SALES 与本地 Amazon Redshift 表 EVENT 联接，以查找排名前十位的活动的销量总额。

```
select top 10 myspectrum_schema.sales.eventid,
 sum(myspectrum_schema.sales.pricepaid) from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
    289 | 51846.00
```

```
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
   6471 | 47997.00
   2118 | 47863.00
    984 | 46780.00
   7851 | 46661.00
   5638 | 46280.00
```

4. 查看上一查询的查询计划。注意针对 Amazon S3 上的数据执行的 S3 Seq Scan、S3 HashAggregate 和 S3 Query Scan 步骤。

```
explain
select top 10 myspectrum_schema.sales.eventid,
 sum(myspectrum_schema.sales.pricepaid)
from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
QUERY PLAN


------------------------------------------------------------------------------
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)


  ->  XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)


        Merge Key: sum(sales.derived_col2)


        ->  XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)


              Send to leader
```

```
               ->  XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200
width=31)

                     Sort Key: sum(sales.derived_col2)


                     ->  XN HashAggregate  (cost=1055770620.49..1055770620.99
rows=200 width=31)

                           ->  XN Hash Join DS_BCAST_INNER
(cost=3119.97..1055769620.49 rows=200000 width=31)

                                 Hash Cond: ("outer".derived_col1 = "inner".eventid)


                                 ->  XN S3 Query Scan sales  (cost=3010.00..5010.50
rows=200000 width=31)

                                       ->  S3 HashAggregate  (cost=3010.00..3010.50
rows=200000 width=16)

                                             ->  S3 Seq Scan myspectrum_schema.sales
location:"s3://redshift-downloads/tickit/spectrum/sales" format:TEXT
(cost=0.00..2150.00 rows=172000 width=16)
                                                   Filter: (pricepaid > 30.00)


                                 ->  XN Hash  (cost=87.98..87.98 rows=8798 width=4)


                                       ->  XN Seq Scan on event  (cost=0.00..87.98
rows=8798 width=4)
```

启动您的 AWS CloudFormation 堆栈，然后在 Amazon S3 中查询您的数据

创建 Amazon Redshift 集群并连接到集群后，您可以安装 Redshift Spectrum DataLake AWS CloudFormation 模板，然后查询您的数据。

CloudFormation 安装 Redshift Spectrum Getting Started DataLake 模板，并创建一个包含以下信息的堆栈：

- 与 Redshift 集群关联的角色 `myspectrum_role`

- 外部架构 myspectrum_schema
- Amazon S3 桶中的外部表 sales
- 已加载数据的 Redshift 表 event

启动您的 Redshift Spectrum Getting Started DataLake CloudFormation 堆栈：

1. 选择启动 CFN 堆栈。CloudFormation 控制台打开并选定 DataLake.yml 模板。

   您还可以下载和自定义 Redshift Spectrum Getting Started DataLake CloudFormation CFN 模板，然后打开 CloudFormation 控制台 (https://console.aws.amazon.com/cloudformation)，并使用自定义模板创建堆栈。
2. 选择 Next（下一步）。
3. 在参数下，输入 Amazon Redshift 集群名称、数据库名称和您的数据库用户名。
4. 选择下一步。

   此时将显示堆栈选项。
5. 选择下一步以接受原定设置。
6. 检查信息，然后在功能下选择我确认 AWS CloudFormation 可能会创建 IAM 资源。
7. 选择创建堆栈。

如果在创建堆栈时发生错误，请参阅以下信息：

- 查看 CloudFormation 事件选项卡，以获取可以帮助解决错误的信息。
- 删除 DataLake CloudFormation 堆栈后再重试操作。
- 确保您已连接到 Amazon Redshift 数据库。
- 确保您输入了 Amazon Redshift 集群名称、数据库名称和数据库用户名的正确信息。

在 Amazon S3 中查询您的数据

使用用于查询其它 Amazon Redshift 表的同一 SELECT 语句查询外部表。这些 SELECT 语句查询包括联接表、聚合数据和筛选谓词。

以下查询会返回 myspectrum_schema.sales 外部表中的行数。

```
select count(*) from myspectrum_schema.sales;
```

```
count
------
172462
```

将外部表与本地表联接

以下示例将外部表 myspectrum_schema.sales 与本地表 event 联接以查找排名前十的活动的销量总额。

```
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid)
 from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
   6471 | 47997.00
   2118 | 47863.00
    984 | 46780.00
   7851 | 46661.00
   5638 | 46280.00
```

查看查询计划

查看上一查询的查询计划。注意针对 Amazon S3 上的数据执行的 S3 Seq Scan、S3 HashAggregate 和 S3 Query Scan 步骤。

```
explain
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid)
from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
```

```
order by 2 desc;
```

```
QUERY PLAN

------------------------------------------------------------------------------
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)

  ->  XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

        Merge Key: sum(sales.derived_col2)

        ->  XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

              Send to leader

              ->  XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

                    Sort Key: sum(sales.derived_col2)

                    ->  XN HashAggregate  (cost=1055770620.49..1055770620.99 rows=200
 width=31)

                          ->  XN Hash Join DS_BCAST_INNER  (cost=3119.97..1055769620.49
 rows=200000 width=31)

                                Hash Cond: ("outer".derived_col1 = "inner".eventid)

                                ->  XN S3 Query Scan sales  (cost=3010.00..5010.50
 rows=200000 width=31)

                                      ->  S3 HashAggregate  (cost=3010.00..3010.50
 rows=200000 width=16)
```

```
                                          ->  S3 Seq Scan spectrum.sales
location:"s3://redshift-downloads/tickit/spectrum/sales" format:TEXT
(cost=0.00..2150.00 rows=172000 width=16)
                                                Filter: (pricepaid > 30.00)


                               ->  XN Hash  (cost=87.98..87.98 rows=8798 width=4)


                                     ->  XN Seq Scan on event  (cost=0.00..87.98
rows=8798 width=4)
```

适用于 Amazon Redshift Spectrum 的 IAM 策略

预设情况下，Amazon Redshift Spectrum 在支持 AWS Glue 的 AWS 区域中使用 AWS Glue Data Catalog。在其他 AWS 区域中，Redshift Spectrum 使用 Athena Data Catalog。您的集群需要授权才能访问您在 AWS Glue 或 Athena 中的外部数据目录以及您在 Amazon S3 中的数据文件。您通过引用附加到集群的 AWS Identity and Access Management (IAM) 角色来提供授权。如果您使用 Apache Hive 元存储管理您的数据目录，则无需提供对 Athena 的访问权限。

您可以串联角色，以便集群可以承担其他未附加到集群的角色。有关更多信息，请参阅 在 Amazon Redshift Spectrum 中链接 IAM 角色。

您访问的 AWS Glue 目录可能已加密以提高安全性。如果 AWS Glue 目录已加密，则您需要使用 AWS KMS 的 AWS Glue 密钥来访问 AWS Glue Data Catalog。有关更多信息，请参阅 AWS Glue 开发人员指南中的加密您的 AWS Glue Data Catalog。

主题

- Amazon S3 权限
- 跨账户 Amazon S3 权限
- 授予或限制使用 Redshift Spectrum 的访问权限的策略
- 授予最低权限的策略
- 在 Amazon Redshift Spectrum 中链接 IAM 角色
- 控制对 AWS Glue 数据目录的访问

Amazon S3 权限

您的集群至少需要对 Amazon S3 桶的 GET 和 LIST 访问权限。如果您的桶与您的集群没有位于同一个AWS账户中，则您的桶还必须授权您的集群访问数据。有关更多信息，请参阅代表您授权 Amazon Redshift 访问其他 AWS 服务。

> **Note**
>
> Amazon S3 桶不能使用将访问限制为仅通过特定 VPC 端点的桶策略。

以下策略将授予对任何 Amazon S3 桶的 GET 和 LIST 访问权限。该策略允许访问 Amazon S3 桶以进行 Redshift Spectrum 和 COPY 操作。

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": ["s3:Get*", "s3:List*"],
  "Resource": "*"
 }]
}
```

以下策略将授予对您的名为 myBucket 的 Amazon S3 桶的 GET 和 LIST 访问权限。

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": ["s3:Get*", "s3:List*"],
  "Resource": "arn:aws:s3:::myBucket/*"
 }]
}
```

跨账户 Amazon S3 权限

要向 Redshift Spectrum 授权访问属于其他 AWS 账户的 Amazon S3 桶中的数据，请将以下策略添加到 Amazon S3 桶中。有关更多信息，请参阅授予跨账户桶权限。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "Example permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::redshift-account:role/spectrumrole"
            },
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListMultipartUploadParts",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::bucketname",
                "arn:aws:s3:::bucketname/*"
            ]
        }
    ]
}
```

授予或限制使用 Redshift Spectrum 的访问权限的策略

要仅使用 Redshift Spectrum 授予对 Amazon S3 桶的访问权限，请包括允许访问用户代理 AWS Redshift/Spectrum 的条件。以下策略仅允许 Redshift Spectrum 访问 Amazon S3 桶。它排除其他访问，例如 COPY 操作。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["s3:Get*", "s3:List*"],
        "Resource": "arn:aws:s3:::myBucket/*",
                "Condition": {"StringEquals": {"aws:UserAgent": "AWS Redshift/
Spectrum"}}
    }]
}
```

同样，您可能需要创建 IAM 角色来允许 COPY 操作，但排除 Redshift Spectrum 访问。为此，请包含拒绝用户代理 **AWS Redshift/Spectrum** 访问权限的条件。以下策略允许访问 Amazon S3 桶，但排除了 Redshift Spectrum。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["s3:Get*", "s3:List*"],
        "Resource": "arn:aws:s3:::myBucket/*",
                "Condition": {"StringNotEquals": {"aws:UserAgent": "AWS Redshift/
Spectrum"}}
    }]
}
```

授予最低权限的策略

以下策略授予将 Redshift Spectrum 用于 Amazon S3 、AWS Glue 和 Athena 所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListMultipartUploadParts",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::bucketname",
                "arn:aws:s3:::bucketname/folder1/folder2/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
```

```
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

如果您将 Athena 用于数据目录，而不是 AWS Glue，则策略需要完全 Athena 访问权限。以下策略将授予对 Athena 资源的访问权限。如果您的外部数据库位于 Hive 元存储中，则您不需要 Athena 访问权限。

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": ["athena:*"],
  "Resource": ["*"]
 }]
}
```

在 Amazon Redshift Spectrum 中链接 IAM 角色

当您将角色附加到集群时，集群可以代入该角色以您的名义访问 Amazon S3、Athena 和 AWS Glue。如果附加到集群的角色无法访问必要的资源，则可以串联到另一个角色 (可能属于其他账户)。然后，您的集群临时代入串联的角色来访问数据。您还可以通过串联角色来授予跨账户访问权限。您可以串联最多 10 个角色。链中的每个角色都会代入链中的下一个角色，直到集群承担位于链尾的角色。

要串联角色，您可以在角色之间建立信任关系。代入另一个角色的角色必须具有允许其代入指定角色的权限策略。反过来，传递权限的角色必须具有允许其将权限传递给另一个角色的信任策略。有关更多信息，请参阅在 Amazon Redshift 中链接 IAM 角色。

当您运行 CREATE EXTERNAL SCHEMA 命令时，可以通过包括一个逗号分隔的角色 ARN 列表来串联角色。

> **Note**
>
> 串联角色的列表不能包含空格。

在以下示例中，MyRedshiftRole 附加到集群。MyRedshiftRole 代入角色 AcmeData，该角色属于账户 111122223333。

```
create external schema acme from data catalog
database 'acmedb' region 'us-west-2'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole,arn:aws:iam::111122223333:role/
AcmeData';
```

控制对 AWS Glue 数据目录的访问

如果将 AWS Glue 用于数据目录，则可以使用 IAM 策略将细粒度访问控制应用于 AWS Glue Data Catalog。例如，您可能希望仅将少数数据库和表公开给特定的 IAM 角色。

以下各部分描述了一些 IAM 策略，它们针对存储在 AWS Glue 数据目录中的数据进行各种级别的访问。

主题

- 数据库操作的策略
- 表操作的策略
- 分区操作的策略

数据库操作的策略

如果要授予用户查看和创建数据库的权限，则同时需要数据库和 AWS Glue 数据目录的访问权限。

以下示例查询创建一个数据库。

```
CREATE EXTERNAL SCHEMA example_db
FROM DATA CATALOG DATABASE 'example_db' region 'us-west-2'
IAM_ROLE 'arn:aws:iam::redshift-account:role/spectrumrole'
CREATE EXTERNAL DATABASE IF NOT EXISTS
```

以下 IAM 策略提供创建数据库所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabase",
                "glue:CreateDatabase"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:catalog"
            ]
        }
    ]
}
```

以下示例查询列出了当前数据库。

```
SELECT * FROM SVV_EXTERNAL_DATABASES WHERE
databasename = 'example_db1' or databasename = 'example_db2';
```

以下 IAM 策略提供列出当前数据库所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabases"
            ],
            "Resource": [
```

```
                "arn:aws:glue:us-west-2:redshift-account:database/example_db1",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db2",
                "arn:aws:glue:us-west-2:redshift-account:catalog"

            ]
        }
    ]
}
```

表操作的策略

如果您要授予用户查看、创建、删除、更改表或对表执行其他操作的权限，则用户需要多种访问权限。他们需要自行访问表、表所属的数据库以及目录。

以下示例查询创建外部表。

```
CREATE EXTERNAL TABLE example_db.example_tbl0(
    col0 INT,
    col1 VARCHAR(255)
) PARTITIONED BY (part INT) STORED AS TEXTFILE
LOCATION 's3://test/s3/location/';
```

以下 IAM 策略提供创建外部表所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
```

```
}
```

以下的每一个示例查询都列出当前外部表。

```
SELECT * FROM svv_external_tables
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

```
SELECT * FROM svv_external_columns
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

```
SELECT parameters FROM svv_external_tables
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

以下 IAM 策略提供列出当前外部表所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/
example_tbl0",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl1"

            ]
```

```
        }
    ]
}
```

以下示例查询修改现有表。

```
ALTER TABLE example_db.example_tbl0
SET TABLE PROPERTIES ('numRows' = '100');
```

以下 IAM 策略提供修改现有表所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTable",
                "glue:UpdateTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"

            ]
        }
    ]
}
```

以下示例查询删除现有表。

```
DROP TABLE example_db.example_tbl0;
```

以下 IAM 策略提供删除现有表所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:DeleteTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

分区操作的策略

如果要授予用户执行分区级操作（查看、创建、删除、更改等）的权限，则他们需要对分区所属的表具有权限。他们还需要相关数据库和 AWS Glue Data Catalog 的权限。

以下示例查询创建一个分区。

```
ALTER TABLE example_db.example_tbl0
ADD PARTITION (part=0) LOCATION 's3://test/s3/location/part=0/';
ALTER TABLE example_db.example_t
ADD PARTITION (part=1) LOCATION 's3://test/s3/location/part=1/';
```

以下 IAM 策略提供创建分区所需的最低权限。

```
 {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "glue:GetTable",
                "glue:BatchCreatePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

以下示例查询列出了当前分区。

```
SELECT * FROM svv_external_partitions
WHERE schemname = 'example_db' AND
tablename = 'example_tbl0'
```

以下 IAM 策略提供列出当前分区所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetPartitions",
                "glue:GetTables",
                "glue:GetTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
```

```
}
```

以下示例查询修改现有分区。

```
ALTER TABLE example_db.example_tbl0 PARTITION(part='0')
SET LOCATION 's3://test/s3/new/location/part=0/';
```

以下 IAM 策略提供修改现有分区所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetPartition",
                "glue:UpdatePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

以下示例查询删除现有分区。

```
ALTER TABLE example_db.example_tbl0 DROP PARTITION(part='0');
```

以下 IAM 策略提供删除现有分区所需的最低权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:DeletePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

将 Redshift Spectrum 与 AWS Lake Formation 结合使用

您可以使用 AWS Lake Formation 对存储在 Amazon S3 中的数据集中定义和实施数据库级、表级和列级访问策略。当您的数据在使用 Lake Formation 启用的 AWS Glue Data Catalog 中注册时，可以使用多个服务对其查询，包括 Redshift Spectrum。

Lake Formation 提供 Data Catalog 的安全性和治理。在 Lake Formation 内，您可以授予和撤消对于 Data Catalog 对象的权限，如数据库、表、列和底层 Amazon S3 存储。

> **Important**
>
> 您只能在提供 Lake Formation 的 AWS 区域中将 Redshift Spectrum 与启用了 Lake Formation 的 Data Catalog 结合使用。有关可用区域的列表，请参阅《AWS 一般参考》中的 AWS Lake Formation 端点和限额。

通过将 Redshift Spectrum 与 Lake Formation 结合使用，您可以执行以下操作：

- 将 Lake Formation 用作集中的位置，您可以在其中授予和撤消对数据湖中所有数据的权限和访问控制策略。Lake Formation 提供权限层次结构来控制对 Data Catalog 中的数据库和表的访问权限。有关更多信息，请参阅《AWS Lake Formation 开发人员指南》中的 Lake Formation 权限概览。

- 创建外部表并对数据湖中的数据运行查询。在您账户中的用户可以运行查询之前，数据湖账户管理员会向 Lake Formation 注册包含源数据的现有 Amazon S3 路径。管理员还可以创建表并向您的用户授予权限。可以授予针对数据库、表或列的访问权限。管理员可以使用 Lake Formation 中的数据筛选条件，对存储在 Amazon S3 中的敏感数据进行精细的访问控制。有关更多信息，请参阅 使用数据筛选条件实现行级和单元格级安全性。

  在 Data Catalog 中注册数据后，每当用户尝试运行查询时，Lake Formation 都会验证该特定委托人对于表的访问权限。Lake Formation 成会将临时凭据发送给 Redshift Spectrum，此时查询运行。
- 使用通过 `GetCredentials` 或 `GetClusterCredentials` 获得的 IAM 凭证对自动挂载的 AWS Glue Data Catalog 运行 Redshift Spectrum 查询，并按数据库用户（IAMR:username 或 IAM:username）管理 Lake Formation 权限。

当您将 Redshift Spectrum 与为 Lake Formation 启用的 Data Catalog 结合使用时，必须满足以下条件之一：

- 与集群关联且有权访问 Data Catalog 的 IAM 角色。
- 为管理对外部资源的访问而配置的联合 IAM 身份。有关更多信息，请参阅使用联合身份管理 Amazon Redshift 对本地资源和 Amazon Redshift 外部表的访问权限。

> **Important**
>
> 将 Redshift Spectrum 与为 Lake Formation 启用的 Data Catalog 结合使用时，不能链接 IAM 角色。

要了解有关设置 AWS Lake Formation 与 Redshift Spectrum 结合使用所需步骤的更多信息，请参阅《AWS Lake Formation 开发人员指南》中的教程：从 Lake Formation 中的 JDBC 源创建数据湖。具体而言，请参阅使用 Amazon Redshift Spectrum 查询数据湖中的数据，了解有关与 Redshift Spectrum 集成的详细信息。本主题中使用的数据和 AWS 资源取决于本教程中的先前步骤。

使用数据筛选条件实现行级和单元格级安全性

您可以在 AWS Lake Formation 中定义数据筛选条件，以控制您的 Redshift Spectrum 查询对在数据目录中定义的数据的行级和单元格级访问权限。要设置此行为，请执行以下任务：

- 使用以下信息在 Lake Formation 中创建数据筛选条件：
  - 一种列规范，其中包含要在查询结果中包括或排除的列的列表。

- 行筛选表达式，用于指定要包含在查询结果中的行。

  有关如何创建数据筛选条件的更多信息，请参阅《AWS Lake Formation 开发人员指南》中的 Lake Formation 中的数据筛选条件。
- 在 Amazon Redshift 中创建一个外部表，该表引用启用了 Lake Formation 的数据目录中的表。有关如何使用 Redshift Spectrum 查询 Lake Formation 表的详细信息，请参阅《AWS Lake Formation 开发人员指南》中的使用 Amazon Redshift Spectrum 查询数据湖中的数据。

在 Amazon Redshift 中定义表后，您可以查询 Lake Formation 表，并仅访问数据筛选条件允许的行和列。

有关如何在 Lake Formation 中设置行级和单元格级安全性，然后使用 Redshift Spectrum 进行查询的详细指南，请参阅将 Amazon Redshift Spectrum 与在 AWS Lake Formation 中定义的行级和单元格级安全策略结合使用。

为 Amazon Redshift Spectrum 中的查询创建数据文件

您用于 Amazon Redshift Spectrum 中的查询的数据文件通常与您用于其他应用程序的文件类型相同。例如，相同类型的文件与 Amazon Athena、Amazon EMR 和 Amazon QuickSight 一起使用。您可以直接从 Amazon S3 以数据的原始格式查询数据。为此，数据文件的格式必须是 Redshift Spectrum 支持的格式，并且位于集群可访问的 Amazon S3 桶中。

包含数据文件的 Amazon S3 桶与 Amazon Redshift 集群必须位于同一 AWS 区域。有关支持的 AWS 区域的信息，请参阅Amazon Redshift Spectrum 区域。

Redshift Spectrum 的数据格式

Redshift Spectrum 支持以下结构化和半结构化数据格式。

文件格式	列式	支持并行读取	拆分单位
Parquet	是	是	行组
ORC	是	是	Stripe
RCFile	是	是	行组
TextFile	否	是	行

文件格式	列式	支持并行读取	拆分单位
SequenceFile	否	是	行或块
RegexSerde	否	是	行
OpenCSV	否	是	行
AVRO	否	是	Block
Ion	否	否	不适用
JSON	否	否	不适用

在上表中，标题表示以下内容：

- 列式 – 文件格式是否在物理上以列式结构存储数据，而不是以行式结构存储数据。
- 支持并行读取 – 文件格式是否支持读取文件中的各个块。读取单个块可以跨多个独立 Redshift Spectrum 请求对文件进行分布式处理，而不必在单个请求中读取整个文件。
- 拆分单位 – 对于可以并行读取的文件格式，拆分单位是单个 Redshift Spectrum 请求可以处理的最小数据块。

> **Note**
>
> 文本文件中的时间戳值的格式必须为 `yyyy-MM-dd HH:mm:ss.SSSSSS`，如以下时间戳值所示：`2017-05-01 11:30:59.000000`。

我们建议使用列式存储文件格式（例如 Apache Parquet）。利用列式存储文件格式，您可以通过仅选择所需的列来最大程度地减少 Amazon S3 外部的数据传输。

Redshift Spectrum 的压缩类型

要减少存储空间、提高性能和最大程度地降低成本，我们强烈建议您压缩数据文件。Redshift Spectrum 基于文件扩展名识别文件压缩类型。

Redshift Spectrum 支持以下压缩类型和扩展名。

压缩算法	文件扩展名	支持并行读取
Gzip	.gz	否
Bzip2	.bz2	是
Snappy	.snappy	否

可以在不同的级别应用压缩。最常见的情况是，压缩整个文件或压缩文件中的单个块。在文件级压缩列格式不会产生性能优势。

必须满足以下条件，Redshift Spectrum 才能并行读取文件：

- 文件格式支持并行读取。
- 文件级压缩（如果有）支持并行读取。

文件中的单个拆分单位是否通过可并行读取的压缩算法进行压缩并不重要，因为每个拆分单位均由单个 Redshift Spectrum 请求处理。上述情况的一个示例是通过 Snappy 压缩的 Parquet 文件。使用 Snappy 压缩 Parquet 文件中的各个行组，但文件的顶层结构保持未压缩状态。在此情况下，可以并行读取文件，因为每个 Redshift Spectrum 请求均可从 Amazon S3 中读取和处理单个行组。

Redshift Spectrum 的加密

Redshift Spectrum 以透明方式解密使用以下加密选项加密的数据文件：

- 使用由 Amazon S3 管理的 AES-256 加密密钥的服务器端加密 (SSE-S3)。
- 具有由 AWS Key Management Service 管理的密钥的服务器端加密 (SSE-KMS)。

Redshift Spectrum 不支持 Amazon S3 客户端加密。有关服务器端加密的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的使用服务器端加密保护数据。

Amazon Redshift 使用大规模并行处理 (MPP) 实现对大量数据的复杂查询操作的快速执行。Redshift Spectrum 将同一原则延伸到了查询外部数据，并按需使用多个 Redshift Spectrum 实例来扫描文件。将文件放在每个表的单独的文件夹中。

您可通过执行以下操作来优化数据以执行并行处理：

- 如果您的文件格式或压缩不支持并行读取，请将大文件拆分为多个小文件。我们建议使用介于 64 MB 和 1 GB 之间的文件大小。
- 将所有文件保持在大致相同的大小。如果某些文件大于其他文件，Redshift Spectrum 将无法均匀分配工作负载。

为 Amazon Redshift Spectrum 创建外部 schema

所有外部表都必须在您使用 CREATE EXTERNAL SCHEMA 语句创建的外部 schema 中创建。

> **Note**
>
> 某些应用程序将术语数据库 和 schema 互换使用。在 Amazon Redshift 中，我们使用术语 schema。

Amazon Redshift 外部 schema 引用了外部数据目录中的外部数据库。您可在 Amazon Redshift、Amazon Athena、AWS Glue Data Catalog 或在 Apache Hive 元存储（如 Amazon EMR）中创建外部数据库。如果您在 Amazon Redshift 中创建了外部数据库，该数据库将驻留在 Athena Data Catalog 中。要在 Hive 元存储中创建数据库，您需要在您的 Hive 应用程序中创建数据库。

Amazon Redshift 需要授权才能代表您访问 Athena 中的 Data Catalog 和 Amazon S3 中的数据文件。要提供授权，您首先应创建 AWS Identity and Access Management (IAM) 角色。然后，您应将该角色附加到您的集群并在 Amazon Redshift CREATE EXTERNAL SCHEMA 语句中为该角色提供 Amazon 资源名称 (ARN)。有关 授权的更多信息，请参阅适用于 Amazon Redshift Spectrum 的 IAM 策略。

> **Note**
>
> 如果您当前在 Athena Data Catalog 中有 Redshift Spectrum 外部表，则可以将您的 Athena Data Catalog 迁移到 AWS Glue Data Catalog。要将 AWS Glue 数据目录用于 Redshift Spectrum，您可能需要更改您的 IAM 策略。有关更多信息，请参阅《Amazon Athena 用户指南》中的升级到 AWS Glue Data Catalog。

要在创建外部 schema 的同时创建外部数据库，请指定 FROM DATA CATALOG 并在您的 CREATE EXTERNAL DATABASE 语句中包含 CREATE EXTERNAL SCHEMA 子句。

以下示例使用外部数据库 spectrum_schema 创建名为 spectrum_db 的外部 schema。

```
create external schema spectrum_schema from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
create external database if not exists;
```

如果您使用 Athena 管理您的数据目录，请指定 Athena 数据库名称和 Athena Data Catalog 所在的 AWS 区域。

以下示例使用 Athena Data Catalog 中的默认 `sampledb` 数据库创建外部 schema。

```
create external schema athena_schema from data catalog
database 'sampledb'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
region 'us-east-2';
```

> **Note**
>
> `region` 参数将引用 Athena Data Catalog 所在的 AWS 区域而不是数据文件在 Amazon S3 中的位置。

如果使用 Hive 元存储（例如 Amazon EMR）来管理数据目录，则必须将安全组配置为允许集群之间的流量。

在 CREATE EXTERNAL SCHEMA 语句中，指定 `FROM HIVE METASTORE` 并包含元存储的 URI 和端口号。以下示例创建一个使用名为 `hive_db` 的 Hive 元存储数据库的外部 schema。

```
create external schema hive_schema
from hive metastore
database 'hive_db'
uri '172.10.10.10' port 99
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
```

要查看集群的外部 schema，请查询 PG_EXTERNAL_SCHEMA 目录表或 SVV_EXTERNAL_SCHEMAS 视图。下面的示例查询联接 PG_EXTERNAL_SCHEMA 和 PG_NAMESPACE 的 SVV_EXTERNAL_SCHEMAS。

```
select * from svv_external_schemas
```

有关完整的命令语法和示例，请参阅 CREATE EXTERNAL SCHEMA。

在 Amazon Redshift Spectrum 中使用外部目录

Amazon Redshift Spectrum 外部数据库和外部表的元数据存储在外部数据目录中。预设情况下，Redshift Spectrum 元数据存储在 Athena Data Catalog 中。您可在 Athena 控制台中查看和管理 Redshift Spectrum 数据库和表。

您还可使用 Hive 数据定义语言 (DDL) 通过 Athena 或 Hive 元存储（如 Amazon EMR）创建和管理外部数据库和外部表。

> **Note**
>
> 我们建议使用 Amazon Redshift 在 Redshift Spectrum 中创建和管理外部数据库和外部表。

在 Athena 和 AWS Glue 中查看 Redshift Spectrum 数据库

您可以通过将 CREATE EXTERNAL DATABASE IF NOT EXISTS 子句作为 CREATE EXTERNAL SCHEMA 语句的一部分来创建外部数据库。在此情况下，外部数据库元数据将存储在您的数据目录中。您创建的由外部 schema 限定的外部表的元数据也存储在您的 Data Catalog 中。

Athena 和 AWS Glue 将为每个受支持的 AWS 区域保留一个数据目录。要查看表元数据，请登录 Athena 或 AWS Glue 控制台。在 Athena 中，依次选择数据来源和您的 AWS Glue，然后查看数据库的详细信息。在 AWS Glue 中，依次选择数据库和您的外部数据库，然后查看数据库的详细信息。

如果您使用 Athena 创建和管理您的外部表，请使用 CREATE EXTERNAL SCHEMA 注册数据库。例如，以下命令将注册名为 `sampledb` 的 Athena 数据库。

```
create external schema athena_sample
from data catalog
database 'sampledb'
iam_role 'arn:aws:iam::123456789012:role/mySpectrumRole'
region 'us-east-1';
```

当您查询 SVV_EXTERNAL_TABLES 系统视图时，您将看到 Athena `sampledb` 数据库中的表以及您在 Amazon Redshift 中创建的表。

```
select * from svv_external_tables;
```

```
schemaname    | tablename        | location
```

```
--------------+-----------------
+-------------------------------------------------------
athena_sample | elb_logs          | s3://athena-examples/elb/plaintext
athena_sample | lineitem_1t_csv   | s3://myspectrum/tpch/1000/lineitem_csv

athena_sample | lineitem_1t_part | s3://myspectrum/tpch/1000/lineitem_partition

spectrum      | sales             | s3://redshift-downloads/tickit/spectrum/sales

spectrum      | sales_part        | s3://redshift-downloads/tickit/spectrum/sales_part
```

注册 Apache Hive 元存储数据库

如果您在 Apache Hive 元存储中创建外部表，则可使用 CREATE EXTERNAL SCHEMA 在 Redshift Spectrum 中注册这些表。

在 CREATE EXTERNAL SCHEMA 语句中，指定 FROM HIVE METASTORE 子句并提供 Hive 元存储 URI 和端口号。IAM 角色必须包含对 Amazon S3 的访问权限，但无需任何 Athena 权限。以下示例注册 Hive 元存储。

```
create external schema if not exists hive_schema
from hive metastore
database 'hive_database'
uri 'ip-10-0-111-111.us-west-2.compute.internal' port 9083
iam_role 'arn:aws:iam::123456789012:role/mySpectrumRole';
```

使您的 Amazon Redshift 集群能够访问您的 Amazon EMR 集群

如果 Hive 元存储位于 Amazon EMR 中，则您必须为您的 Amazon Redshift 集群授予访问 Amazon EMR 集群的权限。为此，请创建一个 Amazon EC2 安全组。之后，应允许从您的 Amazon Redshift 集群的安全组和您的 Amazon EMR 集群的安全组到 EC2 安全组的所有入站流量。然后，您应将 EC2 安全组添加到您的 Amazon Redshift 集群和您的 Amazon EMR 集群。

查看您的 Amazon Redshift 集群的安全组名称

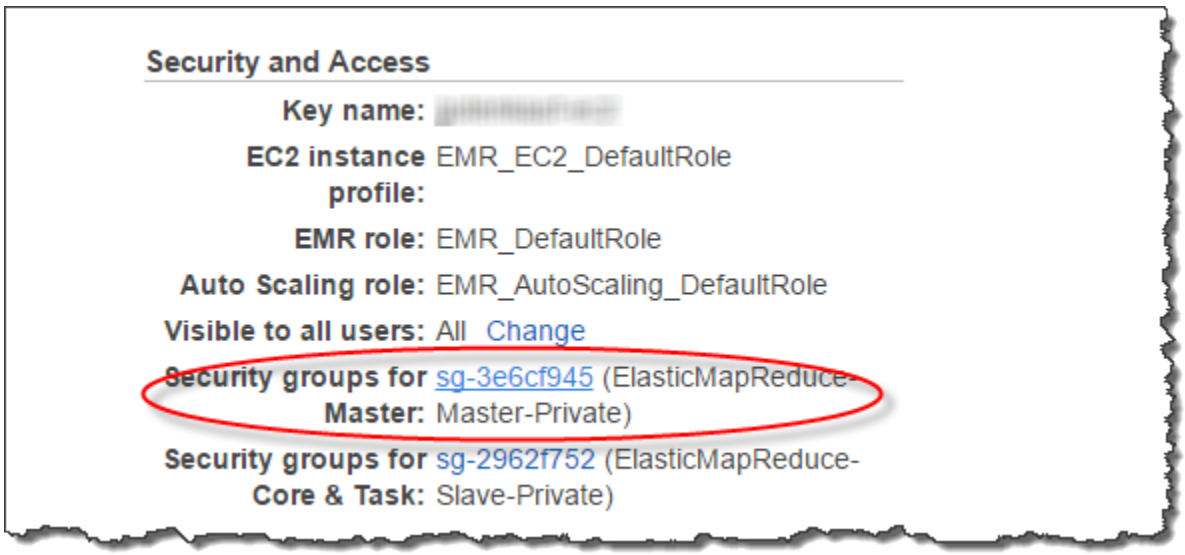
要显示安全组，请执行以下操作：

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后从列表中选择该集群以打开其详细信息。
3. 选择属性并查看网络和安全设置部分。

4. 在 VPC 安全组中找到您的安全组并记下它。

查看 Amazon EMR 主节点安全组名称

1. 打开您的 Amazon EMR 集群。有关更多信息，请参阅《Amazon EMR 管理指南》中的使用安全配置设置集群安全性。
2. 在安全与访问下，记下 Amazon EMR 主节点安全组名称。



创建或修改 Amazon EC2 安全组以允许 Amazon Redshift 和 Amazon EMR 之间的连接：

1. 在 Amazon EC2 控制面板中，选择安全组。有关更多信息，请参阅《Amazon EC2 用户指南》中的安全组规则。
2. 选择创建安全组。
3. 如果使用的是 VPC，选择 Amazon Redshift 和 Amazon EMR 集群所在的 VPC。
4. 添加一条入站规则。
    1. 对于类型，选择自定义 TCP。
    2. 对于 Source，选择 Custom。
    3. 输入您的 Amazon Redshift 安全组的名称。
5. 再添加一条入站规则。
    1. 对于 Type，选择 TCP。
    2. 对于端口范围，输入 9083。

> **Note**
>
> EMR HMS 的默认端口是 9083。如果 HMS 使用了其他的端口，请在入站规则和外部模式定义中指定该端口。

3. 对于 Source，选择 Custom。

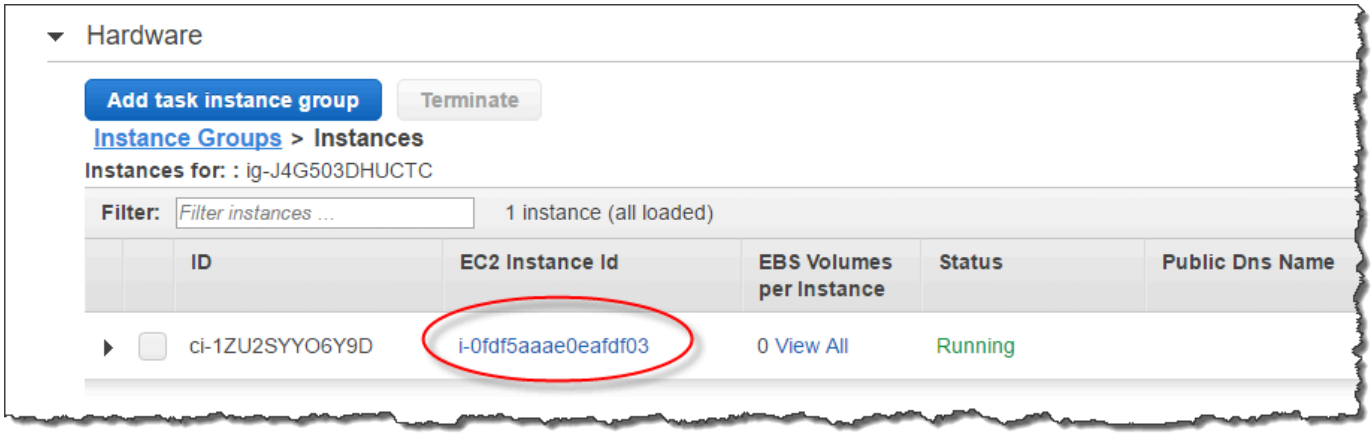
6. 输入安全组名称和描述。
7. 选择创建安全组。

将您在上一步中创建的 Amazon EC2 安全组添加到您的 Amazon Redshift 集群：

1. 在 Amazon Redshift 中，选择您的集群。
2. 选择属性。
3. 查看网络和安全设置，然后选择编辑。
4. 在 VPC 安全组中，选择新的安全组名称。
5. 选择保存更改。

将 Amazon EC2 安全组添加到您的 Amazon EMR 集群：

1. 在 Amazon EMR 中，选择您的集群。有关更多信息，请参阅《Amazon EMR 管理指南》中的使用安全配置设置集群安全性。
2. 在 Hardware 下，选择主节点的链接。
3. 选择 EC2 实例 ID列中的链接。



4. 依次选择操作、安全和更改安全组。

5. 在关联的安全组中，选择新安全组，然后选择添加安全组。
6. 选择保存。

为 Redshift Spectrum 创建外部表

在外部 schema 中创建一个外部表。要创建外部表，您必须是外部 schema 的所有者或是超级用户。要移交外部 schema 的所有权，请使用 ALTER SCHEMA 更改所有者。以下示例将 spectrum_schema schema 的所有者更改为 newowner。

```
alter schema spectrum_schema owner to newowner;
```

要运行 Redshift Spectrum 查询，您需要以下权限：

- schema 的使用权限
- 在当前数据库中创建临时表的权限

以下示例将 schema spectrum_schema 的使用权限授予 spectrumusers 用户组。

```
grant usage on schema spectrum_schema to group spectrumusers;
```

以下示例将数据库 spectrumdb 的临时权限授予 spectrumusers 用户组。

```
grant temp on database spectrumdb to group spectrumusers;
```

您可在 Amazon Redshift、AWS Glue、Amazon Athena 或 Apache Hive 元存储中创建外部表。有关更多信息，请参阅《AWS Glue 开发人员指南》中的开始使用 AWS Glue、《Amazon Athena 用户指南》中的入门或《Amazon EMR 开发人员指南》中的 Apache Hive。

如果您的外部表是在 AWS Glue、Athena 或 Hive 元存储中定义的，您首先应创建引用外部数据库的外部 schema。然后，您可通过使用 schema 名称作为表名称的前缀来在 SELECT 语句中引用外部表，无需在 Amazon Redshift 中创建表。有关更多信息，请参阅 为 Amazon Redshift Spectrum 创建外部 schema。

要允许 Amazon Redshift 在 AWS Glue Data Catalog 中查看表，添加 glue:GetTable 到 Amazon Redshift IAM 角色。否则，您可能会遇到类似以下内容的错误。

```
RedshiftIamRoleSession is not authorized to perform: glue:GetTable on resource: *;
```

例如，假设您有一个在 Athena 外部目录中定义的名为 `lineitem_athena` 的外部表。在这种情况下，您可以定义一个名为 `athena_schema` 的外部 schema，然后使用以下 SELECT 语句查询表。

```
select count(*) from athena_schema.lineitem_athena;
```

要在 Amazon Redshift 中定义外部表，请使用 CREATE EXTERNAL TABLE 命令。外部表语句定义了表列、您的数据文件的格式和您的数据在 Amazon S3 中的位置。Redshift Spectrum 扫描指定的文件夹和任意子文件夹中的文件。Redshift Spectrum 将忽略隐藏文件以及以句点、下划线或哈希标记 (.、_ 或 #) 开头或以波形符结尾的文件。

以下示例在名为 `spectrum` 的 Amazon Redshift 外部 schema 中创建一个名为 SALES 的表。数据位于制表符分隔的文本文件中。

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/'
table properties ('numRows'='172000');
```

要查看外部表，请查询 SVV_EXTERNAL_TABLES 系统视图。

Pseudocolumns

原定设置情况下，Amazon Redshift 使用伪列 `$path`、`$size` 和 `$spectrum_oid` 创建外部表。选择 `$path` 列可针对查询返回的每行查看 Amazon S3 上数据文件的路径，而选择 `$size` 列可以查看每行的数据文件的大小。`$spectrum_oid` 列提供了使用 Redshift Spectrum 执行相关查询的功能。有关示例，请参阅示例：在 Redshift Spectrum 中执行相关的子查询。必须用双引号分隔 `$path`、`$size`

和 $spectrum_oid 列名称。SELECT * 子句不返回伪列。如下例所示，必须在查询中显式包含 $path、$size 和 $spectrum_oid 列名称。

```
select "$path", "$size", "$spectrum_oid"
from spectrum.sales_part where saledate = '2008-12-01';
```

您可以通过将 spectrum_enable_pseudo_columns 配置参数设置为 false 来禁用为会话创建伪列的功能。有关更多信息，请参阅 spectrum_enable_pseudo_columns。还可以通过将 enable_spectrum_oid 设置为 false 来仅禁用 $spectrum_oid 伪列。有关更多信息，请参阅 enable_spectrum_oid。然而，禁用 $spectrum_oid 伪列也会禁用使用 Redshift Spectrum 进行相关查询的支持。

> **Important**
>
> 选择 $size、$path 或 $spectrum_oid 将产生费用，因为 Redshift Spectrum 会扫描 Amazon S3 中的数据文件来确定结果集的大小。有关更多信息，请参阅 Amazon Redshift 定价。

Pseudocolumns 示例

以下示例将为外部表返回相关数据文件的总大小。

```
select distinct "$path", "$size"
from spectrum.sales_part;

 $path                                                                      | $size
----------------------------------------------------------------------------+-------
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/ |  1616
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/ |  1444
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/ |  1644
```

对 Redshift Spectrum 外部表进行分区

当您对数据进行分区时，可通过按分区键进行筛选来限制 Redshift Spectrum 扫描的数据量。您可按任何键对数据进行分区。

常见做法是根据时间对数据进行分区。例如，您可以选择按年、月、日和小时进行分区。如果您的数据来自多个源，您可以按数据来源标识符和日期进行分区。

以下过程介绍如何对您的数据进行分区。

对您的数据进行分区

1. 根据分区键将您的数据存储在 Amazon S3 文件夹中。

   为每个分区值创建一个文件夹并使用分区键和值为该文件夹命名。例如，如果您按日期分区，您可能具有名为 `saledate=2017-04-01`、`saledate=2017-04-02` 等的文件夹。Redshift Spectrum 扫描分区文件夹和任意子文件夹中的文件。Redshift Spectrum 将忽略隐藏文件以及以句点、下划线或哈希标记 (.、_ 或 #) 开头或以波形符结尾的文件。
2. 创建外部表并在 PARTITIONED BY 子句中指定分区键。

   分区键不能是表列的名称。数据类型可以为 SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE 或 TIMESTAMP 数据类型。
3. 添加分区。

   通过使用 ALTER TABLE … ADD PARTITION，添加每个分区，以指定分区列和键值以及分区文件夹在 Amazon S3 中的位置。您可以使用单个 ALTER TABLE … ADD 语句添加多个分区。以下示例为 `'2008-01'` 和 `'2008-03'` 添加分区。

```
alter table spectrum.sales_part add
partition(saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/
saledate=2008-01/'
partition(saledate='2008-03-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/
saledate=2008-03/';
```

> **Note**
>
> 如果使用 AWS Glue 目录，则可以使用单个 ALTER TABLE 语句最多添加 100 个分区。

分区数据示例

在此示例中，您将创建一个由单个分区键进行分区的外部表和一个由两个分区键进行分区的外部表。

本示例的示例数据位于为所有经过身份验证的 AWS 用户提供读取访问权限的 Amazon S3 桶中。您的集群和外部数据文件必须位于同一 AWS 区域中。示例数据桶位于美国东部（弗吉尼亚州北部）区域

(us-east-1) 中。要使用 Redshift Spectrum 访问数据，您的集群必须同样位于 us-east-1 中。要列出 Amazon S3 中的文件夹，请运行以下命令。

```
aws s3 ls s3://redshift-downloads/tickit/spectrum/sales_partition/
```

```
PRE saledate=2008-01/
PRE saledate=2008-03/
PRE saledate=2008-04/
PRE saledate=2008-05/
PRE saledate=2008-06/
PRE saledate=2008-12/
```

如果您还没有外部架构，请运行以下命令。使用您的 AWS Identity and Access Management (IAM) 角色替换 Amazon 资源名称 (ARN)。

```
create external schema spectrum
from data catalog
database 'spectrumdb'
iam_role 'arn:aws:iam::123456789012:role/myspectrumrole'
create external database if not exists;
```

示例 1：使用单个分区键进行分区

在以下示例中，您将创建按月分区的外部表。

要创建按月分区的外部表，请运行以下命令。

```
create external table spectrum.sales_part(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (saledate char(10))
row format delimited
fields terminated by '|'
```

```
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales_partition/'
table properties ('numRows'='172000');
```

要添加分区，请运行以下 ALTER TABLE 命令。

```
alter table spectrum.sales_part add
partition(saledate='2008-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/'

partition(saledate='2008-03')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/'

partition(saledate='2008-04')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-04/';
```

要从分区表中选择数据，请运行以下查询。

```
select top 5 spectrum.sales_part.eventid, sum(spectrum.sales_part.pricepaid)
from spectrum.sales_part, event
where spectrum.sales_part.eventid = event.eventid
  and spectrum.sales_part.pricepaid > 30
  and saledate = '2008-01'
group by spectrum.sales_part.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
   4124 | 21179.00
   1924 | 20569.00
   2294 | 18830.00
   2260 | 17669.00
   6032 | 17265.00
```

要查看外部表分区，请查询 SVV_EXTERNAL_PARTITIONS 系统视图。

```
select schemaname, tablename, values, location from svv_external_partitions
where tablename = 'sales_part';
```

```
schemaname | tablename  | values      | location
```

```
-----------+------------+-------------
+-------------------------------------------------------------------------
spectrum   | sales_part | ["2008-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-01
spectrum   | sales_part | ["2008-03"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-03
spectrum   | sales_part | ["2008-04"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-04
```

示例 2：使用多个分区键进行分区

要创建按 date 和 eventid 分区的外部表，请运行以下命令。

```
create external table spectrum.sales_event(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (salesmonth char(10), event integer)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/salesevent/'
table properties ('numRows'='172000');
```

要添加分区，请运行以下 ALTER TABLE 命令。

```
alter table spectrum.sales_event add
partition(salesmonth='2008-01', event='101')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-01/
event=101/'

partition(salesmonth='2008-01', event='102')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-01/
event=102/'

partition(salesmonth='2008-01', event='103')
```

```
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-01/
event=103/'

partition(salesmonth='2008-02', event='101')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-02/
event=101/'

partition(salesmonth='2008-02', event='102')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-02/
event=102/'

partition(salesmonth='2008-02', event='103')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-02/
event=103/'

partition(salesmonth='2008-03', event='101')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-03/
event=101/'

partition(salesmonth='2008-03', event='102')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-03/
event=102/'

partition(salesmonth='2008-03', event='103')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-03/
event=103/';
```

运行以下查询可选择已分区表中的数据。

```
select spectrum.sales_event.salesmonth, event.eventname,
 sum(spectrum.sales_event.pricepaid)
from spectrum.sales_event, event
where spectrum.sales_event.eventid = event.eventid
  and salesmonth = '2008-02'
 and (event = '101'
 or event = '102'
 or event = '103')
group by event.eventname, spectrum.sales_event.salesmonth
order by 3 desc;
```

```
salesmonth | eventname       | sum
-----------+-----------------+--------
2008-02    | The Magic Flute | 5062.00
```

```
2008-02    | La Sonnambula    | 3498.00
2008-02    | Die Walkure      |  534.00
```

将外部表列映射到 ORC 列

可以使用 Amazon Redshift Spectrum 外部表从 ORC 格式的文件查询数据。优化的行列式 (ORC) 格式是一种支持嵌套数据结构的列式存储文件格式。有关查询嵌套数据的更多信息，请参阅使用 Amazon Redshift Spectrum 查询嵌套数据。

创建引用 ORC 文件中的数据的外部表时，将外部表中的每个列映射到 ORC 数据中的列。为此，请使用以下方法之一：

- 按位置映射
- 按列名映射

按列名映射是默认设置。

按位置映射

使用位置映射，外部表中定义的第一列映射到 ORC 数据文件中的第一列，第二列映射到第二列，依此类推。按位置映射时，要求外部表和 ORC 文件中的列顺序匹配。如果列的顺序不匹配，可以按名称映射列。

> **Important**
>
> 在早期版本中，Redshift Spectrum 默认使用位置映射。如果需要继续对现有表使用位置映射，请将表属性 `orc.schema.resolution` 设置为 `position`，如以下示例所示。
>
> ```
> alter table spectrum.orc_example
> set table properties('orc.schema.resolution'='position');
> ```

例如，表 `SPECTRUM.ORC_EXAMPLE` 定义如下。

```
create external table spectrum.orc_example(
int_col int,
float_col float,
nested_col struct<
  "int_col" : int,
```

```
  "map_col" : map<int, array<float >>
   >
) stored as orc
location 's3://example/orc/files/';
```

表结构可以按以下方式抽象化。

```
• 'int_col' : int
• 'float_col' : float
• 'nested_col' : struct
   o 'int_col' : int
   o 'map_col' : map
      - key : int
      - value : array
         - value : float
```

底层 ORC 文件具有以下文件结构。

```
• ORC file root(id = 0)
   o 'int_col' : int (id = 1)
   o 'float_col' : float (id = 2)
   o 'nested_col' : struct (id = 3)
      - 'int_col' : int (id = 4)
      - 'map_col' : map (id = 5)
         - key : int (id = 6)
         - value : array (id = 7)
            - value : float (id = 8)
```

在此示例中，您可以严格按位置将外部表中的每个列映射到 ORC 文件中的列。下面显示了映射。

外部表列名称	ORC 列 ID	ORC 列名称
int_col	1	int_col
float_col	2	float_col
nested_col	3	nested_col
nested_col.int_col	4	int_col
nested_col.map_col	5	map_col

外部表列名称	ORC 列 ID	ORC 列名称
nested_col.map_col.key	6	NA
nested_col.map_col.value	7	NA
nested_col.map_col.value.item	8	NA

按列名映射

使用名称映射，可以将外部表中的列使用相同的名称映射到同一级别的 ORC 文件中的指定列。

例如，假设您要将上一个示例 SPECTRUM.ORC_EXAMPLE 中的表映射到使用以下文件结构的 ORC 文件。

```
• ORC file root(id = 0)
   o 'nested_col' : struct (id = 1)
      - 'map_col' : map (id = 2)
         - key : int (id = 3)
         - value : array (id = 4)
            - value : float (id = 5)
      - 'int_col' : int (id = 6)
   o 'int_col' : int (id = 7)
   o 'float_col' : float (id = 8)
```

使用位置映射，Redshift Spectrum 会尝试进行以下映射。

外部表列名称	ORC 列 ID	ORC 列名称
int_col	1	struct
float_col	7	int_col
nested_col	8	float_col

使用前面的位置映射查询表时，SELECT 命令在类型验证时会失败，因为结构不同。

您可以使用列名映射将同一外部表映射到前面示例中显示的两个文件结构。表列 int_col、float_col 和 nested_col 按列名映射到 ORC 文件中具有相同名称的列。外部表中名

为 nested_col 的列是 struct 列，其子列名为 map_col 和 int_col。子列也按列名正确映射到 ORC 文件中的相应列。

为 Apache Hudi 中管理的数据创建外部表

要查询 Apache Hudi 写入时复制 (CoW) 格式的数据，您可以使用 Amazon Redshift Spectrum 外部表。Hudi 写入时复制表是存储在 Amazon S3 中的 Apache Parquet 文件的集合。您可以在 Apache Hudi 版本 0.5.2、0.6.0、0.7.0、0.8.0、0.9.0、0.10.0、0.10.1、0.11.0 和 0.11.1 中读取写入时复制 (CoW) 表，这些表通过 Insert、Delete 和 Upsert Write 操作创建和修改。例如，不支持引导表。有关更多信息，请参阅开源 Apache Hudi 文档中的写入时复制表。

创建引用 Hudi CoW 格式的数据的外部表时，将外部表中的每个列映射到 Hudi 数据中的列。映射按列完成。

分区和未分区 Hudi 表的数据定义语言 (DDL) 语句与其他 Apache Parquet 文件格式的语句类似。对于 Hudi 表，您可以将 INPUTFORMAT 定义为 org.apache.hudi.hadoop.HoodieParquetInputFormat。LOCATION 参数必须指向包含 .hoodie 文件夹的 Hudi 表基本文件夹，建立 Hudi 提交时间线时需要此文件夹。在某些情况下，Hudi 表上的 SELECT 操作可能会失败，并显示消息未找到有效的 Hudi 提交时间线。如果是这样，请检查 .hoodie 文件夹是否位于正确的位置，并且包含有效的 Hudi 提交时间线。

> **Note**
>
> Apache Hudi 格式仅在您使用 AWS Glue Data Catalog 时受到支持。当您使用 Apache Hive 元数据仓作为外部目录时，它不受支持。

用于定义未分区表的 DDL 采用以下格式。

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://s3-bucket/prefix'
```

用于定义分区表的 DDL 采用以下格式。

```
CREATE EXTERNAL TABLE tbl_name (columns)
```

```
PARTITIONED BY(pcolumn1 pcolumn1-type[,...])
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://s3-bucket/prefix'
```

若要将分区添加到分区的 Hudi 表，请运行 ALTER TABLE ADD PARTITION 命令，其中 LOCATION 参数指向具有属于该分区的文件的 Amazon S3 子文件夹。

用于添加分区的 DDL 采用以下格式。

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION 's3://s3-bucket/prefix/partition-path'
```

为 Delta Lake 中托管的数据创建外部表

要查询 Delta Lake 表中的数据，您可以使用 Amazon Redshift Spectrum 外部表。

要从 Redshift Spectrum 访问 Delta Lake 表，请在查询之前生成清单。Delta Lake 清单包含构成 Delta Lake 表的一致快照的文件列表。在分区表中，每个分区都有一个清单。Delta Lake 表是存储在 Amazon S3 中的 Apache Parquet 文件的集合。有关更多信息，请参阅开源 Delta Lake 文档中的 Delta Lake。

创建引用 Delta Lake 表中的数据的外部表时，将外部表中的每个列映射到 Delta Lake 数据中的列。映射按列名称完成。

分区和未分区的 Delta Lake 表的 DDL 与其他 Apache Parquet 文件格式的 DDL 类似。对于 Delta Lake 表，您可以将 INPUTFORMAT 定义为 `org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat`，并将 OUTPUTFORMAT 定义为 `org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat`。LOCATION 参数必须指向表基本文件夹中的清单文件夹。如果 Delta Lake 表上的 SELECT 操作失败，对于可能的原因，请参阅Delta Lake 表的限制和故障排除。

用于定义未分区表的 DDL 采用以下格式。

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
```

```
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://s3-bucket/prefix/_symlink_format_manifest'
```

用于定义分区表的 DDL 采用以下格式。

```
CREATE EXTERNAL TABLE tbl_name (columns)
PARTITIONED BY(pcolumn1 pcolumn1-type[,...])
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://s3-bucket>/prefix/_symlink_format_manifest'
```

若要将分区添加到已分区的 Delta Lake 表，请运行 ALTER TABLE ADD PARTITION 命令，其中 LOCATION 参数指向包含分区清单的 Amazon S3 子文件夹。

用于添加分区的 DDL 采用以下格式。

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION
's3://s3-bucket/prefix/_symlink_format_manifest/partition-path'
```

或者运行直接指向 Delta Lake 清单文件的 DDL。

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION
's3://s3-bucket/prefix/_symlink_format_manifest/partition-path/manifest'
```

Delta Lake 表的限制和故障排除

从 Redshift Spectrum 查询 Delta Lake 表时，请考虑以下事项：

- 如果清单指向不再存在的快照或分区，则查询将失败，直到生成新的有效清单。例如，这可能因基础表上的 VACUUM 操作所导致，
- Delta Lake 清单仅提供分区级的一致性。

下表说明了查询 Delta Lake 表时出现某些错误的一些潜在原因。

错误消息	可能的原因
桶 s3-bucket-1 中的 Delta Lake 清单不能包含桶 s3-bucket-2 中的条目。	清单条目指向与指定桶不同的 Amazon S3 桶中的文件。
Delta Lake 文件应位于同一文件夹中。	清单条目指向具有与指定的 Amazon S3 前缀不同的文件。
未找到 Delta Lake 清单 manifest-path 中所列的文件 filename。	在 Amazon S3 中找不到清单中列出的文件。
获取 Delta Lake 清单时出错。	在 Amazon S3 中找不到清单。
S3 路径无效。	清单文件中的条目不是有效的 Amazon S3 路径，或者清单文件已损坏。

在 Amazon Redshift 上使用 Apache Iceberg 表

您可以使用 Redshift Spectrum 或 Redshift Serverless 来查询在 AWS Glue Data Catalog 中编目的 Apache Iceberg 表。Apache Iceberg 是一种用于数据湖的开源表格式。有关更多信息，请参阅 Apache Iceberg 文档中的 Apache Iceberg。

Amazon Redshift 为查询 Apache Iceberg 表提供事务一致性。在使用 Amazon Redshift 运行查询时，您可以使用符合 ACID（原子性、一致性、隔离、持久性）的服务（例如 Amazon Athena 和 Amazon EMR）来操作表中的数据。Amazon Redshift 可以使用存储在 Apache Iceberg 元数据中的表统计数据，来优化查询计划并减少查询处理期间的文件扫描。使用 Amazon Redshift SQL，您可以将 Redshift 表与数据湖表联接起来。

开始在 Amazon Redshift 上使用 Iceberg 表：

1. 使用兼容的服务（如 Amazon Athena 或 Amazon EMR）在 AWS Glue Data Catalog 数据库上创建 Apache Iceberg 表。要使用 Athena 创建 Iceberg 表，请参阅《Amazon Athena User Guide》中的Using Apache Iceberg tables。
2. 使用允许访问您的数据湖的关联 IAM 角色，创建一个 Amazon Redshift 集群或 Redshift Serverless 工作组。有关如何创建集群或工作组的信息，请参阅《Amazon Redshift 入门指南》中的 Amazon Redshift 预置集群和 Redshift Serverless。

3. 使用查询编辑器 v2 或第三方 SQL 客户端连接到您的集群或工作组。有关如何使用查询编辑器 v2 进行连接的信息，请参阅《Amazon Redshift 管理指南》中的使用 SQL 客户端工具连接到 Amazon Redshift 数据仓库。
4. 在 Amazon Redshift 数据库中为包含 Iceberg 表的特定数据目录数据库创建外部架构。有关创建外部架构的信息，请参阅为 Amazon Redshift Spectrum 创建外部 schema。
5. 运行 SQL 查询以访问您创建的外部架构中的 Iceberg 表。

在 Amazon Redshift 上使用 Apache Iceberg 表时的注意事项

在将 Amazon Redshift 与 Iceberg 表一起使用时，请考虑以下事项：

- Iceberg 版本支持 – Amazon Redshift 支持对以下版本的 Iceberg 表运行查询：
  - 版本 1 定义了如何使用不可变数据文件管理大型分析表。
  - 版本 2 增加了支持行级更新和删除的功能，同时保持现有数据文件不变，并使用删除文件处理表数据更改。

  有关版本 1 表和版本 2 表之间的区别，请参阅 Apache Iceberg 文档中的Format version changes。
- 仅限查询 – Amazon Redshift 支持对 Apache Iceberg 表进行只读访问。它支持事务一致性选择查询。您可以在 AWS Glue Data Catalog 中使用像 Amazon Athena 这样的服务来定义和更新 Iceberg 表的架构。
- 添加分区 – 您无需为 Apache Iceberg 表手动添加分区。Amazon Redshift 会自动检测到 Apache Iceberg 表中的新分区，无需手动操作即可更新表定义中的分区。分区规格的任何更改也将自动应用于您的查询，无需任何用户干预。
- 将 Iceberg 数据摄取到 Amazon Redshift 中 – 您可以使用 INSERT INTO 或 CREATE TABLE AS 命令，将数据从 Iceberg 表导入到本地 Amazon Redshift 表中。您目前无法使用 COPY 命令将 Apache Iceberg 表的内容摄取到本地 Amazon Redshift 表中。
- 实体化视图 – 您可以在 Apache Iceberg 表上创建实体化视图，就像在 Amazon Redshift 中为任何其他外部表创建实体化视图一样。其他数据湖表格式的相同注意事项也适用于 Apache Iceberg 表。目前不支持对数据湖表执行增量更新、自动刷新、自动查询重写和自动 MV。
- AWS Lake Formation 精细访问控制 – Amazon Redshift 支持对 Apache Iceberg 表进行 AWS Lake Formation 精细访问控制。
- 用户定义的数据处理参数 – Amazon Redshift 支持对 Apache Iceberg 表使用用户定义的数据处理参数。您可以在现有文件上使用用户定义的数据处理参数，来定制要在外部表中查询的数据，以避免扫描错误。这些参数提供了处理表架构与文件中实际数据之间不匹配的功能。您也可以在 Apache Iceberg 表上使用用户定义的数据处理参数。

- 数据共享 – Amazon Redshift 数据共享目前不支持数据湖表，包括 Apache Iceberg 表。
- 时空旅行查询 – Apache Iceberg 表目前不支持时空旅行查询。
- 定价 – 当您从集群访问 Iceberg 表时，您需要按照 Redshift Spectrum 定价付费。当您从工作组访问 Iceberg 表时，您需要按 Redshift Serverless 定价付费。有关 Redshift Spectrum 和 Redshift Serverless 定价的信息，请参阅 Amazon Redshift 定价。

主题

Apache Iceberg 表支持的数据类型

Amazon Redshift 可以查询包含以下数据类型的 Iceberg 表：

```
binary
boolean
date
decimal
double
float
int
list
long
map
string
struct
timestamp without time zone
```

有关 Iceberg 数据类型的更多信息，请参阅 Apache 文档中的 Schemas for Iceberg。

下表显示了 Amazon Redshift 数据类型与 Iceberg 表数据类型之间的关系。

Iceberg 类型	Amazon Redshift 类型	注意
boolean	boolean	
-	tinyint	Amazon Redshift 中的 Iceberg 表不支持。

Iceberg 类型	Amazon Redshift 类型	注意
-	smallint	Amazon Redshift 中的 Iceberg 表不支持。
int	int	在 Amazon Redshift SQL 语句中，此类型为 INTEGER。
long	bigint	
double	double	
float	float	
decimal(P, S)	decimal(P, S)	P 表示精度，S 表示小数位数。
-	char	Redshift Spectrum 中的 Iceberg 表不支持。
string	string	在 Amazon Redshift SQL 语句中，此类型为 VARCHAR。
binary	binary	
date	date	
time	-	
timestamp	timestamp	
timestamptz	-	Redshift Spectrum 目前不支持 timestamptz 类型。
list<E>	array	
map<K,V>	map	
struct<...>	struct	
fixed(L)	-	Redshift Spectrum 目前不支持 fixed(L) 类型。

有关 Amazon Redshift 中的数据类型的更多信息，请参阅数据类型。

提高 Amazon Redshift Spectrum 查询性能

查看查询计划以了解已推至 Amazon Redshift Spectrum 层的步骤。

以下步骤与 Redshift Spectrum 查询相关：

- S3 Seq Scan
- S3 HashAggregate
- S3 Query Scan
- Seq Scan PartitionInfo
- Partition Loop

以下示例显示了针对将外部表与本地表联接的查询的查询计划。注意已针对 Amazon S3 上的数据运行的 S3 Seq Scan 和 S3 HashAggregate 步骤。

```
explain
select top 10 spectrum.sales.eventid, sum(spectrum.sales.pricepaid)
from spectrum.sales, event
where spectrum.sales.eventid = event.eventid
and spectrum.sales.pricepaid > 30
group by spectrum.sales.eventid
order by 2 desc;
```

```
QUERY PLAN


------------------------------------------------------------------------------
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)


  ->  XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)


        Merge Key: sum(sales.derived_col2)
```

```
        ->  XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

              Send to leader

              ->  XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

                    Sort Key: sum(sales.derived_col2)

                    ->  XN HashAggregate  (cost=1055770620.49..1055770620.99 rows=200
width=31)

                          ->  XN Hash Join DS_BCAST_INNER  (cost=3119.97..1055769620.49
rows=200000 width=31)

                                Hash Cond: ("outer".derived_col1 = "inner".eventid)

                                ->  XN S3 Query Scan sales  (cost=3010.00..5010.50
rows=200000 width=31)

                                      ->  S3 HashAggregate  (cost=3010.00..3010.50
rows=200000 width=16)

                                            ->  S3 Seq Scan spectrum.sales
location:"s3://redshift-downloads/tickit/spectrum/sales" format:TEXT
(cost=0.00..2150.00 rows=172000 width=16)
                                                  Filter: (pricepaid > 30.00)

                                ->  XN Hash  (cost=87.98..87.98 rows=8798 width=4)

                                      ->  XN Seq Scan on event  (cost=0.00..87.98
rows=8798 width=4)
```

注意查询计划中的以下元素：

- S3 Seq Scan 节点显示筛选条件 pricepaid > 30.00 已在 Redshift Spectrum 层中处理。

`XN S3 Query Scan` 节点下的筛选节点指示 Amazon Redshift 中基于从 Redshift Spectrum 层返回的数据的谓词处理。

- `S3 HashAggregate` 节点指示 Redshift Spectrum 层中针对分组依据子句 (`group by spectrum.sales.eventid`) 的聚合。

以下是提高 Redshift Spectrum 性能的方式：

- 使用 Apache Parquet 格式化数据文件。Parquet 以列格式存储数据，因此 Redshift Spectrum 可通过扫描消除不需要的列。当数据为文本文件格式时，Redshift Spectrum 需要扫描整个文件。
- 使用多个文件以针对并行处理进行优化。让文件大小超过 64 MB。通过将文件保持在大致相同的大小来避免数据大小偏斜。有关 Apache Parquet 文件和配置建议的信息，请参阅《Apache Parquet 文档》中的文件格式：配置。
- 在查询中尽可能使用最少的列。
- 将您的大型事实数据表放入 Amazon S3 中并将常用的较小的维度表保存在本地 Amazon Redshift 数据库中。
- 通过设置 TABLE PROPERTIES numRows 参数来更新外部表统计数据。使用 CREATE EXTERNAL TABLE 或 ALTER TABLE 设置 TABLE PROPERTIES numRows 参数以反映表中的行数。Amazon Redshift 不分析外部表来生成表统计数据，查询优化程序会使用这些统计数据来生成查询计划。如果没有为外部表设置表统计数据，则 Amazon Redshift 会生成查询执行计划。Amazon Redshift 是基于“外部表较大，本地表较小”的假设生成此计划。
- Amazon Redshift 查询计划程序会将谓词和聚合尽可能推至 Redshift Spectrum 查询层。如果从 Amazon S3 返回了大量数据，则处理将受您的集群的资源的限制。Redshift Spectrum 将自动扩展以处理大型请求。因此，当您可以将处理推至 Redshift Spectrum 层时，您的整体性能就会提高。
- 编写查询以使用有资格推至 Redshift Spectrum 层的筛选条件和聚合。

  下面是可推至 Redshift Spectrum 层的某些操作的示例：

  - GROUP BY 子句
  - 比较条件和模式匹配条件 (如 LIKE)。
  - 聚合函数 (如 COUNT、SUM、AVG、MIN 和 MAX)。
  - 字符串函数。

  无法推至 Redshift Spectrum 层的操作包括 DISTINCT 和 ORDER BY。
- 使用分区限制扫描的数据。根据您最常用的查询谓词为您的数据分区，然后通过筛选分区列来减少分区。有关更多信息，请参阅 对 Redshift Spectrum 外部表进行分区。

查询 SVL_S3PARTITION 以查看分区和合格分区总计。

- 使用 AWS Glue 的统计数据生成器来计算 AWS Glue Data Catalog 表的列级统计数据。AWS Glue 为 Data Catalog 中的表生成统计数据后，Amazon Redshift Spectrum 会自动使用这些统计数据来优化查询计划。有关使用 AWS Glue 计算列级统计数据的更多信息，请参阅《AWS Glue 开发人员指南》中的使用列统计数据。

设置数据处理选项

您可以在创建外部表时设置表参数，以定制需在外部表中查询的数据。否则，可能会发生扫描错误。有关更多信息，请参阅CREATE EXTERNAL TABLE中的 TABLE PROPERTIES。有关示例，请参阅数据处理示例。有关错误的列表，请参阅SVL_SPECTRUM_SCAN_ERROR。

创建外部表时，可以设置以下 TABLE PROPERTIES，以便为外部表中查询的数据指定输入处理。

- 使用 `column_count_mismatch_handling`，以便确定文件包含的行值是否小于或大于外部表定义中指定的列数。
- 使用 `invalid_char_handling`，以便为包含 VARCHAR、CHAR 和字符串数据的列中的无效字符指定输入处理。当您为 `invalid_char_handling` 指定 REPLACE 时，您可以指定要使用的替换字符。
- 使用 `numeric_overflow_handling`，以便为包含整数和十进制数据的列指定转换溢出处理。
- `surplus_bytes_handling` 为包含 VARBYTE 数据的列中的超额字节指定输入处理。
- 使用 `surplus_char_handling`，以便为包含 VARCHAR、CHAR 和字符串数据的列中的多余字符指定输入处理。

您可以设置配置选项来取消超过最大错误数的查询。有关更多信息，请参阅 spectrum_query_maxerror。

示例：在 Redshift Spectrum 中执行相关的子查询

您可以在 Redshift Spectrum 中执行相关的子查询。`$spectrum_oid` 伪列提供了使用 Redshift Spectrum 执行相关查询的功能。要执行关联的子查询，必须启用伪列 `$spectrum_oid`，但它不会显示在 SQL 语句中。有关更多信息，请参阅 Pseudocolumns。

要为此示例创建外部架构和外部表，请参阅Amazon Redshift Spectrum 入门。

以下是 Redshift Spectrum 中相关子查询的示例。

```
select *
from myspectrum_schema.sales s
where exists
( select *
from myspectrum_schema.listing l
where l.listid = s.listid )
order by salesid
limit 5;
```

```
salesid  listid   sellerid   buyerid   eventid   dateid  qtysold  pricepaid
 commission   saletime
1        1        36861      21191     7872      1875    4        728         109.2
    2008-02-18 02:36:48
2        4        8117       11498     4337      1983    2        76          11.4
    2008-06-06 05:00:16
3        5        1616       17433     8647      1983    2        350         52.5
    2008-06-06 08:26:17
4        5        1616       19715     8647      1986    1        175         26.25
    2008-06-09 08:38:52
5        6        47402      14115     8240      2069    2        154         23.1
    2008-08-31 09:17:02
```

在 Amazon Redshift Spectrum 中监控指标

您可使用以下系统视图监控 Amazon Redshift Spectrum 查询：

- SVL_S3QUERY

  使用 SVL_S3QUERY 视图获取有关段和节点切片级别的 Redshift Spectrum 查询 (S3 查询) 的详细信息。

- SVL_S3QUERY_SUMMARY

  使用 SVL_S3QUERY_SUMMARY 视图可获取已在系统上运行的所有 Amazon Redshift Spectrum 查询（S3 查询）的汇总。

下面是要在 SVL_S3QUERY_SUMMARY 中查找的一些内容：

- 由 Redshift Spectrum 查询处理的文件的数量。
- 从 Amazon S3 扫描到的字节数。Redshift Spectrum 查询的成本反映在从 Amazon S3 扫描到的数据量中。
- 已从 Redshift Spectrum 层返回到集群的字节的数量。返回大量数据可能影响系统性能。
- Redshift Spectrum 请求的最长持续时间和平均持续时间。长时间运行的请求可能表示存在瓶颈。

Amazon Redshift Spectrum 中的查询故障排除

接下来，您可以找到一个快速参考，该参考确定并解决了您可能遇到的有关 Amazon Redshift Spectrum 查询的一些常见问题。要查看 Redshift Spectrum 查询生成的错误，请查询 SVL_S3LOG 系统表。

主题

超过了重试次数

如果 Amazon Redshift Spectrum 请求超时，将取消并重新提交请求。在 5 次重试失败之后，查询将失败并返回以下错误。

```
error:  Spectrum Scan Error: Retries exceeded
```

可能的原因包括：

- 文件大小过大 (大于 1 GB)。检查您在 Amazon S3 中的文件大小并查找大文件和文件大小偏斜。将大文件分成若干个大小在 100 MB 和 1 GB 之间的小文件。尝试使文件的大小大致相同。
- 网络吞吐量低。稍后尝试您的查询。

访问受限制

Amazon Redshift Spectrum 受其他 AWS 服务的服务配额的限制。在高使用率下，Redshift Spectrum 请求可能需要降低速度，这会导致以下错误。

```
error:  Spectrum Scan Error: Access throttled
```

可能存在两种类型的限制：

- Amazon S3 限制的访问权限。
- 访问受 AWS KMS 限制。

错误上下文提供了有关限制类型的更多详细信息。在下文中，您可以找到此限制的原因和可能的解决方案。

Amazon S3 限制的访问权限

如果对某个前缀的读取请求速率太高，Amazon S3 可能会限制 Redshift Spectrum 请求。有关您可以在 Amazon S3 中达到的 GET/HEAD 请求速率的信息，请参阅《Amazon Simple Storage Service 用户指南》中的优化 Amazon S3 性能。Amazon S3 GET/HEAD 请求速率会考虑前缀上的所有 GET/HEAD 请求，因此，访问同一前缀的不同应用程序共享总请求速率。

如果 Redshift Spectrum 请求经常受到 Amazon S3 的限制，请减少 Redshift Spectrum 向 Amazon S3 发出的 Amazon S3 GET/HEAD 请求的数量。为此，请尝试将小文件合并为大文件。我们建议使用大小为 64 MB 或更大的文件。

还可以考虑对 Redshift Spectrum 表进行分区，以便从早期筛选中受益，并减少在 Amazon S3 中访问的文件数。有关更多信息，请参阅 对 Redshift Spectrum 外部表进行分区。

访问受 AWS KMS 限制

如果您使用服务器端加密（SSE-S3 或 SSE-KMS）将数据存储在 Amazon S3 中，则 Amazon S3 将为 Redshift Spectrum 访问的每个文件向 AWS KMS 发起 API 操作。这些请求将计入您的加密操作配额；有关更多信息，请参阅 AWS KMS 请求配额。有关 SSE-S3 和 SSE-KMS 的更多信息，请参阅

《Amazon Simple Storage Service 用户指南》中的使用服务器端加密保护数据和使用在 AWS KMS 中存储 KMS 密钥的服务器端加密保护数据。

减少 Redshift Spectrum 向 AWS KMS 发出的请求数的第一步是减少访问的文件数。为此，请尝试将小文件合并为大文件。我们建议使用大小为 64 MB 或更大的文件。

如果您的 Redshift Spectrum 请求经常受到 AWS KMS 的限制，请考虑请求增加加密操作的 AWS KMS 请求速率的配额。要请求提高限额，请参阅《Amazon Web Services 一般参考》中的AWS 服务限制。

超出资源限制

Redshift Spectrum 对请求可使用的内存量强制实施上限。需要更多内存的 Redshift Spectrum 请求将失败，并导致以下错误。

```
error:  Spectrum Scan Error: Resource limit exceeded
```

以下两个常见原因可能会导致 Redshift Spectrum 请求超出其内存限额：

- Redshift Spectrum 处理大量无法拆分成小型数据块的数据。
- 大型聚合步骤由 Redshift Spectrum 处理。

我们建议使用支持拆分大小为 128 MB 或更小的并行读取的文件格式。请参阅为 Amazon Redshift Spectrum 中的查询创建数据文件以了解支持的文件格式以及数据文件创建的通用指南。在使用不支持并行读取的文件格式或压缩算法时，我们建议将文件大小保持在 64 MB 和 128 MB 之间。

未返回分区表的行

如果您的查询未返回已分区外部表中的任何行，请检查是否已为此外部表添加分区。Redshift Spectrum 仅扫描位于已使用 `ALTER TABLE … ADD PARTITION` 显式添加的 Amazon S3 位置的文件。查询 SVV_EXTERNAL_PARTITIONS 视图以查找现有分区。为每个丢失的分区运行 `ALTER TABLE … ADD PARTITION`。

未授权错误

验证集群的 IAM 角色是否允许访问 Amazon S3 文件对象。如果您的外部数据库在 Amazon Athena 上，请验证 IAM 角色是否允许访问 Athena 资源。有关更多信息，请参阅 适用于 Amazon Redshift Spectrum 的 IAM 策略。

数据格式不兼容

对于列式文件格式（例如 Apache Parquet）列类型嵌有数据。CREATE EXTERNAL TABLE 定义中的列类型必须与数据文件的列类型匹配。如果存在不匹配的情况，您会收到类似以下内容的错误：

```
File 'https://s3bucket/location/file has an incompatible Parquet schema
for column ‘s3://s3bucket/location.col1'. Column type: VARCHAR, Par
```

由于消息长度存在限制，错误消息可能会被截断。要检索完整的错误消息 (包括列名和列类型)，请查询 SVL_S3LOG 系统视图。

以下示例查询上次完成的查询的 SVL_S3LOG。

```
select message
from svl_s3log
where query = pg_last_query_id()
order by query,segment,slice;
```

以下是显示完整错误消息的结果的示例。

```
                            message
-----------------------------------------------------------------------------------------
Spectrum Scan Error. File 'https://s3bucket/location/file has an incompatible
Parquet schema for column ' s3bucket/location.col1'.
Column type: VARCHAR, Parquet schema:\noptional int64 l_orderkey [i:0 d:1 r:0]\n
```

要更正该错误，请将外部表修改为与 Parquet 文件的列类型匹配。

在 Amazon Redshift 中使用 Hive DDL 时遇到语法错误

对于 CREATE EXTERNAL TABLE，Amazon Redshift 支持与 Hive DDL 类似的数据定义语言 (DDL)。但是，这两种类型的 DDL 并不总是完全相同。如果您复制 Hive DDL 以创建或修改 Amazon Redshift 外部表，则可能遇到语法错误。下面是 Amazon Redshift 和 Hive DDL 之间的差别的示例：

- Amazon Redshift 需要单引号 (')，而 Hive DDL 支持双引号 (")。
- Amazon Redshift 不支持 STRING 数据类型。请改用 VARCHAR。

创建临时表的权限

要运行 Redshift Spectrum 查询，数据库用户必须有权在数据库中创建临时表。以下示例将数据库 spectrumdb 的临时权限授予 spectrumusers 用户组。

```
grant temp on database spectrumdb  to group spectrumusers;
```

有关更多信息，请参阅 GRANT。

范围无效

Redshift Spectrum 预计 Amazon S3 中属于外部表的文件在查询期间不会被覆盖。如果被覆盖，则可能会导致以下错误。

```
Error: HTTP response error code: 416 Message: InvalidRange The requested range is not
 satisfiable
```

为避免出现错误，请确保 Amazon S3 文件在使用 Redshift Spectrum 查询时不会被覆盖。

Parquet 版本号无效。

Redshift Spectrum 会检查它访问的每个 Apache Parquet 文件的元数据。如果检查失败，则可能会导致与以下内容相似的错误：

```
File 'https://s3.region.amazonaws.com/s3bucket/location/file has an invalid version
 number
```

导致检查失败的两个常见原因如下：

- Parquet 文件在查询过程中已被覆盖（请参阅范围无效）。
- Parquet 文件损坏。

教程：使用 Amazon Redshift Spectrum 查询嵌套数据

概述

Amazon Redshift Spectrum 支持以 Parquet、ORC、JSON 和 Ion 文件格式查询嵌套数据。Redshift Spectrum 使用外部表访问数据。可以创建使用复杂数据类型 struct、array 和 map 的外部表。

例如，假定您的数据文件在名为 customers 的文件夹中包含 Amazon S3 中的以下数据。尽管没有单个根元素，但此示例数据中的每个 JSON 对象都表示表中的一行。

```
{"id": 1,
 "name": {"given": "John", "family": "Smith"},
 "phones": ["123-457789"],
 "orders": [{"shipdate": "2018-03-01T11:59:59.000Z", "price": 100.50},
            {"shipdate": "2018-03-01T09:10:00.000Z", "price": 99.12}]
}
{"id": 2,
 "name": {"given": "Jenny", "family": "Doe"},
 "phones": ["858-8675309", "415-9876543"],
 "orders": []
}
{"id": 3,
 "name": {"given": "Andy", "family": "Jones"},
 "phones": [],
 "orders": [{"shipdate": "2018-03-02T08:02:15.000Z", "price": 13.50}]
}
```

您可以使用 Amazon Redshift Spectrum 来查询文件中的嵌套数据。下面的教程将向您展示如何使用 Apache Parquet 数据实现这一功能。

有关教程先决条件、步骤和嵌套数据使用案例，请参阅以下主题：

- 先决条件
- 步骤 1：创建包含嵌套数据的外部表
- 步骤 2：使用 SQL 扩展在 Amazon S3 中查询嵌套数据
- 嵌套数据使用案例
- 嵌套数据限制（预览版）
- 序列化复杂嵌套 JSON

先决条件

如果您尚未使用 Redshift Spectrum，请按照 Amazon Redshift Spectrum 入门中的步骤操作，然后继续。

要创建外部架构，请将以下命令中的 IAM 角色 ARN 替换为您在创建 IAM 角色中创建的角色 ARN。然后在 SQL 客户端中运行该命令。

```
create external schema spectrum
from data catalog
database 'myspectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
create external database if not exists;
```

步骤 1：创建包含嵌套数据的外部表

您可以通过从 Amazon S3 进行下载来查看源数据。

要创建本教程所需的外部表，请运行以下命令。

```
CREATE EXTERNAL TABLE spectrum.customers (
  id     int,
  name   struct<given:varchar(20), family:varchar(20)>,
  phones array<varchar(20)>,
  orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/tickit/spectrum/customers/';
```

在前述示例中，外部表 spectrum.customers 使用 struct 和 array 数据类型定义具有嵌套数据的列。Amazon Redshift Spectrum 支持以 Parquet、ORC、JSON 和 Ion 文件格式查询嵌套数据。STORED AS 参数是适用于 Apache Parquet 文件的 PARQUET。LOCATION 参数必须引用包含嵌套数据或文件的 Amazon S3 文件夹。有关更多信息，请参阅CREATE EXTERNAL TABLE。

可以在任何级别嵌套 array 和 struct 类型。例如，您可以定义一个名为 toparray 的列，如以下示例所示。

```
toparray array<struct<nestedarray:
         array<struct<morenestedarray:
         array<string>>>>>
```

您也可以为 struct 列嵌套 x 类型，如以下示例所示。

```
x struct<a: string,
         b: struct<c: integer,
                   d: struct<e: string>
                  >
```

```
        >
```

步骤 2：使用 SQL 扩展在 Amazon S3 中查询嵌套数据

Redshift Spectrum 支持通过对 Amazon Redshift SQL 语法的扩展来查询 array、map 和 struct 复杂类型。

扩展 1：访问 struct 列

您可以使用将字段名称与路径相连的点表示法从 struct 列提取数据。例如，以下查询返回客户的姓氏和名字。名字通过长路径 c.name.given 进行访问。姓氏通过长路径 c.name.family 进行访问。

```
SELECT c.id, c.name.given, c.name.family
FROM   spectrum.customers c;
```

前述的查询返回以下数据。

```
id | given | family
---|-------|-------
1  | John  | Smith
2  | Jenny | Doe
3  | Andy  | Jones
(3 rows)
```

struct 可以是另一个 struct 的列，而后者可能是任何级别的另一个 struct 的列。访问如此深的嵌套 struct 的路径可以是任意长度。例如，请查看以下示例中的 x 列的定义。

```
x struct<a: string,
         b: struct<c: integer,
                   d: struct<e: string>
                  >
        >
```

您可以按 x.b.d.e 方式访问 e 中的数据。

扩展 2：FROM 子句中的范围扩展数组

您可以通过在 FROM 子句中指定 array 列来代替表名称，以提取 array 列（扩展后包括 map 列）中的数据。扩展应用于主查询的 FROM 子句，也应用于子查询的 FROM 子句。

您可以按位置（例如 c.orders[0]）引用 array 元素。（预览版）

通过将范围扩展 arrays 与联接结合使用，您可以实现各种取消嵌套，如下面的使用案例中所述。

使用内部联接取消嵌套

以下查询为具有订单的客户选择客户 ID 和订单发货日期。FROM 子句中的 SQL 扩展 c.orders o 取决于别名 c。

```
SELECT c.id, o.shipdate
FROM   spectrum.customers c, c.orders o
```

对于具有订单的每个客户 c，FROM 子句为客户 c 的每个订单 o 返回一行。该行将客户行 c 和订单行 o 合并起来。然后，SELECT 子句只保留 c.id 和 o.shipdate。结果如下所示。

```
id|      shipdate
--|----------------------
1 |2018-03-01  11:59:59
1 |2018-03-01  09:10:00
3 |2018-03-02  08:02:15
(3 rows)
```

别名 c 提供对客户字段的访问，而别名 o 提供对订单字段的访问。

语义类似于标准 SQL。您可以将 FROM 子句视为执行以下嵌套循环，然后 SELECT 选择要输出的字段。

```
for each customer c in spectrum.customers
  for each order o in c.orders
     output c.id and o.shipdate
```

因此，如果客户没有订单，则客户不会显示在结果中。

您还可以将其视为对 customers 表和 orders 数组执行 JOIN 的 FROM 子句。实际上，您还可以编写查询，如下面的示例所示。

```
SELECT c.id, o.shipdate
FROM   spectrum.customers c INNER JOIN c.orders o ON true
```

> **Note**
>
> 如果存在名为 c 的 schema 且具有名为 orders 的表，则 c.orders 引用表 orders，而不是 customers 的数组列。

使用左侧联接取消嵌套

以下查询输出所有客户名称及其订单。如果客户未下订单，则仍返回客户的名称。但在这种情况下，订单列为 NULL，如下面 Jenny Doe 的示例所示。

```
SELECT c.id, c.name.given, c.name.family, o.shipdate, o.price
FROM   spectrum.customers c LEFT JOIN c.orders o ON true
```

前述的查询返回以下数据。

```
id  |  given  | family  |    shipdate          | price
----|---------|---------|----------------------|--------
 1  |  John   | Smith   | 2018-03-01  11:59:59 | 100.5
 1  |  John   | Smith   | 2018-03-01  09:10:00 |  99.12
 2  |  Jenny  | Doe     |                      |
 3  |  Andy   | Jones   | 2018-03-02  08:02:15 |  13.5
 (4 rows)
```

扩展 3：使用别名直接访问标量数组

当 FROM 子句中的别名 p 范围扩展到标量数组时，查询将 p 的值视为 p。例如，以下查询生成成对的客户名称和电话号码。

```
SELECT c.name.given, c.name.family, p AS phone
FROM   spectrum.customers c LEFT JOIN c.phones p ON true
```

前述的查询返回以下数据。

```
given  |  family  |  phone
-------|----------|-----------
John   | Smith    | 123-4577891
Jenny  | Doe      | 858-8675309
Jenny  | Doe      | 415-9876543
Andy   | Jones    |
```

```
(4 rows)
```

扩展 4：访问映射的元素

Redshift Spectrum 将 map 数据类型视为 array 类型，其中包含具有 key 列和 value 列的 struct 类型。key 必须是 scalar；值可以是任何数据类型。

例如，以下代码使用 map 创建外部表来存储电话号码。

```
CREATE EXTERNAL TABLE spectrum.customers2 (
  id     int,
  name   struct<given:varchar(20), family:varchar(20)>,
  phones map<varchar(20), varchar(20)>,
  orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/tickit/spectrum/customers/';
```

由于 map 类型的行为类似于具有 key 和 value 列的 array 类型，因此您可以将前面的 schema 视为如下内容。

```
CREATE EXTERNAL TABLE spectrum.customers3 (
  id     int,
  name   struct<given:varchar(20), family:varchar(20)>,
  phones array<struct<key:varchar(20), value:varchar(20)>>,
  orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/tickit/spectrum/customers/';
```

以下查询返回具有手机号码的客户名称，并返回每个名称对应的号码。映射查询被视为等同于查询 struct 类型的嵌套 array。仅当您按前面的说明创建了外部表时，以下查询才返回数据。

```
SELECT c.name.given, c.name.family, p.value
FROM   spectrum.customers c, c.phones p
WHERE  p.key = 'mobile';
```

> **Note**
>
> map 的 key 是 Ion 和 JSON 文件类型的 string。

嵌套数据使用案例

您可以使用常用的 SQL 功能组合前面介绍的扩展。下面的使用案例阐述了一些常见的组合。这些示例帮助演示如何使用嵌套数据。它们不是本教程的组成部分。

主题

提取嵌套数据

您可以使用 CREATE TABLE AS 语句从包含复杂数据类型的外部表中提取数据。以下查询使用 LEFT JOIN 从外部表中提取所有客户及其电话号码，并将它们存储在 Amazon Redshift 表 CustomerPhones 中。

```
CREATE TABLE CustomerPhones AS
SELECT  c.name.given, c.name.family, p AS phone
FROM    spectrum.customers c LEFT JOIN c.phones p ON true;
```

使用子查询聚合嵌套数据

您可以使用子查询聚合嵌套数据。以下示例说明了此方法。

```
SELECT c.name.given, c.name.family, (SELECT COUNT(*) FROM c.orders o) AS ordercount
FROM   spectrum.customers c;
```

将返回以下数据。

```
given   |  family  |  ordercount
--------|----------|--------------
 Jenny  |  Doe     |       0
 John   |  Smith   |       2
 Andy   |  Jones   |       1
 (3 rows)
```

> **Note**
>
> 当您通过按父行进行分组来聚合嵌套数据时，最高效的方法是前面示例中所示的方法。在该示例中，c.orders 的嵌套行按其父行 c 分组。或者，如果您知道 id 对于每个 customer 是唯一的且 o.shipdate 从不为 Null，则您可以按下例中所示进行聚合。但一般而言，这种方法的效率比不上前面的示例。

```
SELECT    c.name.given, c.name.family, COUNT(o.shipdate) AS ordercount
FROM      spectrum.customers c LEFT JOIN c.orders o ON true
GROUP BY  c.id, c.name.given, c.name.family;
```

您也可以在 FROM 子句中使用一个子查询来编写查询，此子查询引用原级查询的别名 (c) 并提取数组数据。以下示例演示了此方法。

```
SELECT c.name.given, c.name.family, s.count AS ordercount
FROM   spectrum.customers c, (SELECT count(*) AS count FROM c.orders o) s;
```

联接 Amazon Redshift 和嵌套数据

您还可以在外部表中联接 Amazon Redshift 数据以及嵌套数据。例如，假设您在 Amazon S3 中具有以下嵌套数据。

```
CREATE EXTERNAL TABLE spectrum.customers2 (
  id      int,
  name    struct<given:varchar(20), family:varchar(20)>,
  phones  array<varchar(20)>,
  orders  array<struct<shipdate:timestamp, item:int>>
);
```

此外，假设您在 Amazon Redshift 中具有下面的表。

```
CREATE TABLE prices (
  id int,
  price double precision
);
```

以下查询根据前面的内容查找每个客户的采购总数和购买金额。下例仅用于举例说明。仅当您创建了前面介绍的表时，它才返回数据。

```
SELECT   c.name.given, c.name.family, COUNT(o.date) AS ordercount, SUM(p.price) AS
 ordersum
FROM     spectrum.customers2 c, c.orders o, prices p ON o.item = p.id
GROUP BY c.id, c.name.given, c.name.family;
```

嵌套数据限制（预览版）

> **Note**
>
> 以下列表中标记的限制（预览版）仅适用于在以下区域创建的预览集群和预览工作组。
>
> - 美国东部（俄亥俄州）(us-east-2)
> - 美国东部（弗吉尼亚州北部）(us-east-1)
> - 美国西部（北加利福尼亚）(us-west-1)
> - 亚太地区（东京）(ap-northeast-1)
> - 欧洲地区（爱尔兰）(eu-west-1)
> - 欧洲地区（斯德哥尔摩）(eu-north-1)
>
> 有关设置预览版集群的信息，请参阅《Amazon Redshift 管理指南》中的创建预览版集群。有关设置预览工作组的信息，请参阅《Amazon Redshift 管理指南》中的创建预览工作组。

以下限制适用于嵌套数据：

- `array` 或 `map` 类型可以包含其他 `array` 或 `map` 类型，前提是对嵌套 `arrays` 或 `maps` 的查询不返回 `scalar` 值。（预览版）
- Amazon Redshift Spectrum 只支持将复杂数据类型用作外部表。
- 子查询结果列必须是顶级列。（预览版）
- 如果 `OUTER JOIN` 表达式引用嵌套表，则它只能引用该表及其嵌套数组（和映射）。如果 `OUTER JOIN` 表达式不引用嵌套表，它可以引用任何数量的非嵌套表。
- 如果子查询中的 `FROM` 子句引用一个嵌套表，则它无法引用任何其他表。
- 如果子查询依赖于引用父表的嵌套表，那么子查询只能在 `FROM` 子句中使用父表。您无法在任何其他子句中使用此父项，如 `SELECT` 或 `WHERE` 子句。例如，以下查询无法运行，因为子查询的 `SELECT` 子句引用父表 `c`。

```
SELECT c.name.given
FROM   spectrum.customers c
WHERE (SELECT COUNT(c.id) FROM c.phones p WHERE p LIKE '858%') > 1;
```

以下查询之所以有效，是因为只在子查询的 FROM 子句中使用了父 c。

```
SELECT c.name.given
FROM   spectrum.customers c
WHERE (SELECT COUNT(*) FROM c.phones p WHERE p LIKE '858%') > 1;
```

- 在 FROM 子句之外的任何位置访问嵌套数据的子查询必须返回单个值。唯一的例外是 WHERE 子句中的 (NOT) EXISTS 运算符。
- 不支持 (NOT) IN。
- 所有嵌套类型的最大嵌套深度均为 100。该限制适用于所有文件格式（Parquet、ORC、Ion 和 JSON）。
- 访问嵌套数据的聚合子查询只能引用 FROM 子句中的 arrays 和 maps，而不能引用外部表。
- 不支持查询 Redshift Spectrum 表中嵌套数据的伪列。有关更多信息，请参阅Pseudocolumns。
- 通过在 FROM 子句中指定数组列或映射列来提取这些列中的数据时，如果值为 scalar，则只能从这些列中选择值。例如，以下查询都尝试从数组内部 SELECT 元素。选择 arr.a 的查询之所以起作用，是因为 arr.a 是一个 scalar 值。第二个查询不起作用，因为 array 是从 FROM 子句中的 s3.nested table 提取的数组。（预览版）

```
SELECT array_column FROM s3.nested_table;

array_column
-----------------
[{"a":1},{"b":2}]

SELECT arr.a FROM s3.nested_table t, t.array_column arr;

arr.a
-----
1

--This query fails to run.
SELECT array FROM s3.nested_table tab, tab.array_column array;
```

不能在 FROM 子句中使用本身来自另一个数组或映射的数组或映射。要选择嵌套在其他数组中的数组或其他复杂结构，请考虑在 SELECT 语句中使用索引。

序列化复杂嵌套 JSON

本教程中演示的方法的替代方法是以序列化 JSON 格式查询顶级嵌套集合列。您可以通过 Redshift Spectrum 使用序列化以 JSON 格式检查、转换和摄取嵌套数据。ORC、JSON、Ion 和 Parquet 格式支持此方法。使用会话配置参数 json_serialization_enable 配置序列化行为。设置时，复杂的 JSON 数据类型将序列化为 VARCHAR(65535)。嵌套的 JSON 可以通过 JSON 函数 访问。有关更多信息，请参阅json_serialization_enable。

例如，如果不设置 json_serialization_enable，则以下访问嵌套列的查询直接失败。

```
SELECT * FROM spectrum.customers LIMIT 1;

=> ERROR:  Nested tables do not support '*' in the SELECT clause.

SELECT name FROM spectrum.customers LIMIT 1;

=> ERROR:  column "name" does not exist in customers
```

设置 json_serialization_enable 允许直接查询顶级集合。

```
SET json_serialization_enable TO true;

SELECT * FROM spectrum.customers order by id LIMIT 1;

id | name                                  | phones         | orders
---+---------------------------------------+----------------
+-----------------------------------------------------------------------------------------------------------------
1  | {"given": "John", "family": "Smith"} | ["123-457789"] | [{"shipdate":
 "2018-03-01T11:59:59.000Z", "price": 100.50}, {"shipdate": "2018-03-01T09:10:00.000Z",
 "price": 99.12}]

SELECT name FROM spectrum.customers order by id LIMIT 1;

name
---------
{"given": "John", "family": "Smith"}
```

在序列化嵌套 JSON 时，请考虑以下项目。

- 当集合列被序列化为 VARCHAR(65535) 时，不能再将其嵌套子字段作为查询语法的一部分直接访问（即在筛选器子句中）。但是，JSON 函数可用于访问嵌套的 JSON。
- 不支持以下专门化表示：
  - ORC 联合
  - 具有复杂类型键的 ORC 映射
  - Ion 数据报
  - Ion SEXP
- 时间戳以 ISO 序列化字符串的形式返回。
- 基本映射键被提升为字符串（例如 1 到 "1"）。
- 顶级 null 值被序列化为 NULL。
- 如果序列化溢出最大 VARCHAR 大小 65535，则单元格将设置为 NULL。

序列化包含 JSON 字符串的复杂类型

预设情况下，嵌套集合中包含的字符串值被序列化为转义 JSON 字符串。当字符串为有效的 JSON 时，转义可能是不可取的。相反，您可能希望直接将嵌套子元素或 VARCHAR 字段编写为 JSON。通过 `json_serialization_parse_nested_strings` 会话级别配置启用此行为。设置 `json_serialization_enable` 和 `json_serialization_parse_nested_strings` 时，有效的 JSON 值将被内联序列化，没有转义字符。当该值是无效的 JSON 时，它会被转义，就好像未设置 `json_serialization_parse_nested_strings` 配置值一样。有关更多信息，请参阅json_serialization_parse_nested_strings。

例如，假设前面示例中的数据包含 JSON 作为 `name` VARCHAR(20) 字段中的 `structs` 复杂类型：

```
name
---------
{"given": "{\"first\":\"John\",\"middle\":\"James\"}", "family": "Smith"}
```

当设置 `json_serialization_parse_nested_strings` 时，`name` 列序列化如下：

```
SET json_serialization_enable TO true;
SET json_serialization_parse_nested_strings TO true;
SELECT name FROM spectrum.customers order by id LIMIT 1;

name
```

```
---------
{"given": {"first":"John","middle":"James"}, "family": "Smith"}
```

而不是像这样进行转义：

```
SET json_serialization_enable TO true;
SELECT name FROM spectrum.customers order by id LIMIT 1;

name
---------
{"given": "{\"first\":\"John\",\"middle\":\"James\"}", "family": "Smith"}
```

在 Amazon Redshift 中使用 HyperLogLog 草图

HyperLogLog 是一种用于估计多集基数的算法。基数指的是多集中的不同值的数量。例如，在集合 {4,3,6,2,2,6,4,3,6,2,2,3} 中，基数为 4，不同值为 4、3、6 和 2。

HyperLogLog 算法的精度（也称为 m 值）可能会影响估计基数的精度。在基数估计期间，Amazon Redshift 使用默认精度值 15。对于较小的数据集，此值最多可为 26。因此，平均相对误差范围介于 0.01—0.6% 之间。

在计算多集的基数时，HyperLogLog 算法会生成一个称为 HLL 草图的构造。HLL 草图对有关多集中不同值的信息进行封装。Amazon Redshift 数据类型 HLLSKETCH 表示此类草图值。此数据类型可用于在 Amazon Redshift 表中存储草图。此外，Amazon Redshift 还支持可以作为聚合函数和标量函数应用于 HLLSKETCH 值的运算。您可以使用这些函数来提取 HLLSKETCH 的基数并组合多个 HLLSKETCH 值。

当从大型数据集中提取基数时，HLLSKETCH 数据类型可提供显著的查询性能优势。您可以使用 HLLSKETCH 值预聚合这些数据集并将它们存储在表中。Amazon Redshift 可以直接从存储的 HLLSKETCH 值中提取基数，而无需访问基础数据集。

处理 HLL 草图时，Amazon Redshift 会执行优化，以最大限度地减少草图占用的内存空间，并最大限度地提高提取基数的精度。Amazon Redshift 对 HLL 草图使用两种表示法：稀疏和密集。HLLSKETCH 以稀疏格式开始。当插入新值时，它的大小会增加。当其大小达到密集表示的大小后，Amazon Redshift 会自动将草图从稀疏转换为密集。

当草图采用稀疏格式时，Amazon Redshift 会以 JSON 格式导入、导出和打印 HLLSKETCH。当草图采用密集格式时，Amazon Redshift 会以 Base64 字符串格式导入、导出和打印 HLLSKETCH。有关 UNLOAD 的更多信息，请参阅卸载 HLLSKETCH 数据类型。要导入文本或逗号分隔值 (CSV) 数据到 Amazon Redshift，请使用 COPY 命令。有关更多信息，请参阅 加载 HLLSKETCH 数据类型。

有关与 HyperLogLog 一起使用的函数的信息，请参阅HyperLogLog 函数。

主题

- 注意事项
- 限制
- 示例

注意事项

以下是在 Amazon Redshift 中使用 HyperLogLog 的注意事项：

- 以下非 HyperLogLog 函数可以接受 HLLSKETCH 类型的输入或 HLLSKETCH 类型的列：
  - 聚合函数 COUNT
  - 条件表达式 COALESCE 和 NVL
  - CASE 表达式
- 支持的编码为 RAW。
- 您可以对具有 HLLSKETCH 列的表执行 UNLOAD 操作，将其转换为文本或 CSV。您可以使用 UNLOAD HLLSKETCH 列来写入 HLLSKETCH 数据。Amazon Redshift 以 JSON 格式显示稀疏表示的数据，或以 Base64 格式显示密集表示的数据。有关 UNLOAD 的更多信息，请参阅卸载 HLLSKETCH 数据类型。

  下面显示了用于以 JSON 格式表示的稀疏 HyperLogLog 草图的格式。

  ```
  {"version":1,"logm":15,"sparse":{"indices":
  [15099259,33107846,37891580,50065963],"values":[2,3,2,1]}}
  ```

- 您可以使用 COPY 命令将文本或 CSV 数据导入 Amazon Redshift。有关更多信息，请参阅 加载 HLLSKETCH 数据类型。
- HLLSKETCH 的默认编码为 RAW。有关更多信息，请参阅 压缩编码。

限制

以下是在 Amazon Redshift 中使用 HyperLogLog 的限制：

- Amazon Redshift 表不支持将 HLLSKETCH 列作为 Amazon Redshift 表的排序键或分配键。
- Amazon Redshift 不支持 ORDER BY、GROUP BY 或 DISTINCT 子句中的 HLLSKETCH 列。
- 您只能将 HLLSKETCH 列卸载为文本或 CSV 格式。然后，Amazon Redshift 会以 JSON 格式或 Base64 格式写入 HLLSKETCH 数据。有关 UNLOAD 的更多信息，请参阅UNLOAD。
- Amazon Redshift 仅支持精度（日志值）为 15 的 HyperLogLog 草图。
- JDBC 和 ODBC 驱动程序不支持 HLLSKETCH 数据类型。因此，结果集使用 VARCHAR 来表示 HLLSKETCH 值。
- Amazon Redshift Spectrum 本身不支持 HLLSKETCH 数据。因此，您不能创建或更改具有 HLLSKETCH 列的外部表。

- Python 用户定义的函数 (UDF) 的数据类型不支持 HLLSKETCH 数据类型。有关 Python UDF 的更多信息，请参阅创建标量 Python UDF。

示例

示例：在子查询中返回基数

以下示例为名为 Sales 表中的子查询中的每个草图返回基数。

```
CREATE TABLE Sales (customer VARCHAR, country VARCHAR, amount BIGINT);
INSERT INTO Sales VALUES ('David Joe', 'Greece', 14.5),  ('David Joe', 'Greece',
 19.95), ('John Doe', 'USA', 29.95), ('John Doe', 'USA', 19.95), ('George Spanos',
 'Greece', 9.95), ('George Spanos', 'Greece', 2.95);
```

以下查询为每个国家/地区的客户生成 HLL 草图并提取基数。这显示了来自每个国家/地区的独特客户。

```
SELECT hll_cardinality(sketch), country
FROM (SELECT hll_create_sketch(customer) AS sketch, country
        FROM Sales
        GROUP BY country) AS hll_subquery;

hll_cardinality | country
----------------+---------
            1   | USA
            2   | Greece
 ...
```

示例：从子查询中的组合草图返回 HLLSKETCH 类型

以下示例返回一个表示子查询中各个草图组合的单个 HLLSKETCH 类型。这些草图通过使用 HLL_COMBINE 聚合函数进行组合。

```
SELECT hll_combine(sketch)
FROM (SELECT hll_create_sketch(customers) AS sketch
        FROM Sales
        GROUP BY country) AS hll_subquery

                                        hll_combine
-----------------------------------------------------------------------------------------
```

```
 {"version":1,"logm":15,"sparse":{"indices":[29808639,35021072,47612452],"values":
[1,1,1]}}
(1 row)
```

示例：通过合并多个草图返回 HyperLogLog 草图

对于以下示例，假设表 page-users 存储用户在给定网站上访问的每个页面的预聚合草图。此表中的每一行都包含一个 HyperLogLog 草图，该草图表示显示所访问页面的所有用户 ID。

```
page_users
-- +----------------+-------------+--------------+
-- | _PARTITIONTIME | page         | sketch |
-- +----------------+-------------+--------------+
-- | 2019-07-28     | homepage     | CHAQkAQYA... |
-- | 2019-07-28     | Product A    | CHAQxPnYB... |
-- +----------------+-------------+--------------+
```

以下示例将预聚合的多个草图联合起来，并生成单个草图。此草图封装了每个草图封装的集体基数。

```
SELECT hll_combine(sketch) as sketch
FROM page_users
```

该输出值看上去类似于以下内容。

```
-- +----------------------------------------+
-- | sketch |
-- +----------------------------------------+
-- | CHAQ3sGoCxgCIAuCB4iAIBgTIBgqgIAgAwY.... |
-- +----------------------------------------+
```

创建新草图时，您可以使用 HLL_CARDINALITY 函数获取集体的不同值，如下所示。

```
SELECT hll_cardinality(sketch)
FROM (
  SELECT
  hll_combine(sketch) as sketch
  FROM page_users
) AS hll_subquery
```

该输出值看上去类似于以下内容。

```
-- +-------+
-- | count |
-- +-------+
-- | 54356 |
-- +-------+
```

示例：使用外部表在 S3 数据上生成 HyperLogLog 草图

以下示例将缓存 HyperLogLog 草图，以避免直接访问 Amazon S3 进行基数估计。

您可以在定义为保存 Amazon S3 数据的外部表中预聚合和缓存 HyperLogLog 草图。通过执行此操作，您可以提取基数估计值，而无需访问基础基数数据。

例如，假设您已将一组制表符分隔文本文件卸载到 Amazon S3 中。您可以运行以下查询在名为 `spectrum` 的 Amazon Redshift 外部架构中定义名为 `sales` 的外部表。此示例的 Amazon S3 存储桶位于以下 AWS 区域：美国东部（弗吉尼亚州北部）。

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid smallint,
buyerid smallint,
eventid integer,
dateid integer,
qtysold integer,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t' stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/';
```

假设您想计算在任意日期购买商品的不同买家。为此，以下示例为一年中的每一天生成买家 ID 草图，并将结果存储在 Amazon Redshift 表中`hll_sales`。

```
CREATE TABLE hll_sales AS
SELECT saletime, hll_create_sketch(buyerid) AS sketch
FROM spectrum.sales
GROUP BY saletime;
```

```
SELECT TOP 5 * FROM hll_sales;
```

该输出值看上去类似于以下内容。

```
-- hll_sales

-- | saletime        | sketch
    |
-- +-----------------
+-------------------------------------------------------------------------+
-- | 7/22/2008 8:30  | {"version":1,"logm":15,"sparse":{"indices":[9281416],"values":
[1]}}
-- | 2/19/2008 0:38  | {"version":1,"logm":15,"sparse":{"indices":[48735497],"values":
[3]}}
-- | 11/5/2008 4:49  | {"version":1,"logm":15,"sparse":{"indices":[27858661],"values":
[1]}}
-- | 10/27/2008 4:08 | {"version":1,"logm":15,"sparse":{"indices":[65295430],"values":
[2]}}
-- | 2/16/2008 9:37  | {"version":1,"logm":15,"sparse":{"indices":[56869618],"values":
[2]}}
-- +----------------
 +-------------------------------------------------------------------------+
```

以下查询显示了 2008 年感恩节之后的星期五购买商品的不同买家的估计数量。

```
SELECT hll_cardinality(hll_combine(sketch)) as distinct_buyers
FROM hll_sales
WHERE trunc(saletime) = '2008-11-28';
```

该输出值看上去类似于以下内容。

```
distinct_buyers
---------------
386
```

假设您需要在特定日期范围内购买商品的不同用户数量。例如，从感恩节后的星期五到下一个星期一。为此，以下查询使用 hll_combine 聚合函数。此函数使您能够避免重复计算在选定范围内超过一天购买商品的买家。

```
SELECT hll_cardinality(hll_combine(sketch)) as distinct_buyers
```

```
FROM hll_sales
WHERE saletime BETWEEN '2008-11-28' AND '2008-12-01';
```

该输出值看上去类似于以下内容。

```
distinct_buyers
---------------
1166
```

要保留最新的 hll_sales表，请在每天结束时运行以下查询。这样做会根据今天购买商品的买家的 ID 生成一个 HyperLogLog 草图，并将其添加到 hll_sales 表。

```
INSERT INTO hll_sales
SELECT saletime, hll_create_sketch(buyerid)
FROM spectrum.sales
WHERE TRUNC(saletime) = to_char(GETDATE(), 'YYYY-MM-DD')
GROUP BY saletime;
```

跨数据库查询数据

通过使用 Amazon Redshift 中的跨数据库查询，您可以在 Amazon Redshift 集群中跨数据库进行查询。通过跨数据库查询，您可以从 Amazon Redshift 集群中的任何数据库查询数据，而无论您连接到哪个数据库。跨数据库查询消除了数据副本并简化了数据组织，从而可以支持同一数据仓库中的多个业务组。

通过使用跨数据库查询，您可以执行以下操作：

- 跨 Amazon Redshift 集群中的数据库查询数据。

  您不仅可以从连接到的数据库进行查询，还可以从您有权访问的任何其他数据库中读取。

  查询任何其他未连接的数据库上的数据库对象时，您只有对这些数据库对象的读取访问权限。您可以使用跨数据库查询访问 Amazon Redshift 集群上任何数据库中的数据，而无需连接到该特定数据库。这样做可以帮助您快速、轻松地查询和联接跨 Amazon Redshift 集群中的多个数据库分布的数据。

  您还可以在单个查询中联接多个数据库的数据集，并使用业务情报 (BI) 或分析工具分析数据。您可以继续使用标准 Amazon Redshift SQL 命令为用户设置精细表级别的访问控制。通过这样做，您可以帮助确保用户只能看到他们拥有权限的数据的相关子集。
- 查询对象。

  您可以使用用三部分表示法表示的完全限定对象名来查询其他数据库对象。任何数据库对象的完整路径均由三个组成部分组成：数据库名称、schema 和对象的名称。您可以使用完整路径表示法 *database_name*.*schema_name*.*object_name* 从任何其他数据库访问任何对象。要访问特定列，请使用*database_name*.*schema_name*.*object_name*.*column_name*。

  您还可以使用外部 schema 表示法为另一个数据库中的 schema 创建别名。此外部 schema 引用另一个数据库和 schema 对。查询可以使用外部 schema 表示法 *external_schema_name*.*object_name* 访问其他数据库对象。

  在同一个只读查询中，您可以查询各种数据库对象，如用户表、常规视图、具体化视图和来自其他数据库的后期绑定视图。
- 管理权限。

  对 Amazon Redshift 集群中任何数据库中的对象具有访问权限的用户可以查询这些对象。您可以使用 GRANT 命令为用户和用户组授予权限。当用户不再需要访问特定数据库对象时，您也可以使用 REVOKE 命令撤销权限。

- 使用元数据和 BI 工具。

  您可以创建一个外部 schema，以引用同一 Amazon Redshift 集群内另一个 Amazon Redshift 数据库中的 schema。要了解信息，请参阅 CREATE EXTERNAL SCHEMA 命令。

  创建外部 schema 引用后，Amazon Redshift 会在 SVV_EXTERNAL_TABLES 和 SVV_EXTERNAL_COLUMNS 中的其他数据库的 schema 下显示表，以供工具探索元数据。

  要将跨数据库查询与 BI 工具集成，您可以使用以下系统视图。这些功能可帮助您查看 Amazon Redshift 集群上连接数据库和其他数据库中的对象元数据的相关信息。

  以下是显示 Amazon Redshift 集群中所有数据库的所有 Amazon Redshift 对象和外部对象的系统视图：

  - SVV_ALL_COLUMNS
  - SVV_ALL_SCHEMAS
  - SVV_ALL_TABLES

  以下是显示 Amazon Redshift 集群中所有数据库的所有 Amazon Redshift 对象的系统视图：

  - SVV_REDSHIFT_COLUMNS
  - SVV_REDSHIFT_DATABASES
  - SVV_REDSHIFT_FUNCTIONS
  - SVV_REDSHIFT_SCHEMAS
  - SVV_REDSHIFT_TABLES

主题

注意事项

在 Amazon Redshift 中使用跨数据库查询功能时，请考虑以下事项：

- Amazon Redshift 支持对 ra3.4xlarge、ra3.16xlarge 和 ra3.xlplus 节点类型进行跨数据库查询。

- Amazon Redshift 支持在同一 Amazon Redshift 集群中的一个或多个数据库中连接表或视图中的数据。
- Amazon Redshift Serverless 支持与 Amazon Redshift 集群相同的跨数据库功能，因此，您可以跨无服务器命名空间中的一个或多个数据库联接来自表或视图的数据。
- 连接数据库上的事务中的所有查询都会读取另一个数据库中与事务开始时处于相同状态的数据。此方法有助于跨数据库提供查询事务一致性。Amazon Redshift 支持跨数据库查询的事务一致性。
- 要跨数据库获取元数据，请使用 SVV_ALL* 和 SVV_REDSHIFT* 元数据视图。您无法使用三部分表示法或外部架构来查询 information_schema 和 pg_catalog 下的跨数据库元数据表或视图。

限制

在 Amazon Redshift 中使用跨数据库查询功能时，请注意以下限制：

- 查询任何其他未连接的数据库上的数据库对象时，您只有对这些数据库对象的读取访问权限。
- 您无法查询在引用另一个数据库对象的其他数据库上创建的视图。
- 您只能在集群中的其他数据库的对象上创建后期绑定视图和具体化视图。您无法在集群中其他数据库的对象上创建常规视图。
- Amazon Redshift 不支持具有列级权限的表进行跨数据库查询。
- Amazon Redshift 不支持 AWS Glue 或联合数据库上的查询目录对象。要查询这些对象，请首先创建引用每个数据库中的这些外部数据来源的外部 schema。
- 不支持对具有交错排序键的表运行跨数据库查询。

使用跨数据库查询的示例

使用以下示例帮助了解如何设置引用 Amazon Redshift 数据库的跨数据库查询。

首先，请在您的 Amazon Redshift 集群中创建数据库 db1 和 db2 以及用户 user1 和 user2。有关更多信息，请参阅CREATE DATABASE和CREATE USER。

```
--As user1 on db1
CREATE DATABASE db1;

CREATE DATABASE db2;

CREATE USER user1 PASSWORD 'Redshift01';
```

```
CREATE USER user2 PASSWORD 'Redshift01';
```

作为 db1 上的 user1，授予对 user2 的访问权限，然后将值插入 table1 中。有关更多信息，请参阅GRANT和INSERT。

```
--As user1 on db1
CREATE TABLE table1 (c1 int, c2 int, c3 int);

GRANT SELECT ON table1 TO user2;

INSERT INTO table1 VALUES (1,2,3),(4,5,6),(7,8,9);
```

作为 db2 上的 user2，使用三部分表示法在 db2 中运行跨数据库查询。

```
--As user2 on db2
SELECT * from db1.public.table1 ORDER BY c1;
c1 | c2  | c3
---+-----+----
1  |  2  | 3
4  |  5  | 6
7  |  8  | 9
(3 rows)
```

作为 db2 上的 user2，创建外部 schema 并在 db2 中使用外部 schema 表示法运行跨数据库查询。

```
--As user2 on db2
CREATE EXTERNAL SCHEMA db1_public_sch
FROM REDSHIFT DATABASE 'db1' SCHEMA 'public';

SELECT * FROM db1_public_sch.table1 ORDER BY c1;

c1  | c2 | c3
----+----+----
1   | 2  | 3
4   | 5  | 6
7   | 8  | 9
(3 rows)
```

要创建不同的视图并授予对这些视图的权限，作为 db1 上的 user1，请执行以下操作。

```
--As user1 on db1
CREATE VIEW regular_view AS SELECT c1 FROM table1;
```

```
GRANT SELECT ON regular_view TO user2;


CREATE MATERIALIZED VIEW mat_view AS SELECT c2 FROM table1;

GRANT SELECT ON mat_view TO user2;


CREATE VIEW late_bind_view AS SELECT c3 FROM public.table1 WITH NO SCHEMA BINDING;

GRANT SELECT ON late_bind_view TO user2;
```

作为 db2 上的 user2，使用三部分表示法运行以下跨数据库查询以查看特定视图。

```
--As user2 on db2
SELECT * FROM db1.public.regular_view;
c1
----
1
4
7
(3 rows)

SELECT * FROM db1.public.mat_view;
c2
----
8
5
2
(3 rows)

SELECT * FROM db1.public.late_bind_view;
c3
----
3
6
9
(3 rows)
```

作为 db2 上的 user2，使用外部 schema 表示法运行以下跨数据库查询以查询后期绑定视图。

```
--As user2 on db2
```

```
SELECT * FROM db1_public_sch.late_bind_view;
c3
----
3
6
9
(3 rows)
```

作为 db2 上的 user2，在单个查询中使用连接的表运行以下命令。

```
--As user2 on db2
CREATE TABLE table1 (a int, b int, c int);

INSERT INTO table1 VALUES (1,2,3), (4,5,6), (7,8,9);

SELECT a AS col_1, (db1.public.table1.c2 + b) AS sum_col2, (db1.public.table1.c3 + c)
 AS sum_col3 FROM db1.public.table1, table1 WHERE db1.public.table1.c1 = a;
col_1 | sum_col2 | sum_col3
------+----------+----------
1     | 4        | 6
4     | 10       | 12
7     | 16       | 18
(3 rows)
```

以下示例列出了集群上的所有数据库。

```
select database_name, database_owner, database_type
from svv_redshift_databases
where database_name in ('db1', 'db2');

 database_name | database_owner | database_type
---------------+----------------+---------------
 db1           |            100 | local
 db2           |            100 | local
(2 rows)
```

以下示例列出了集群上所有数据库的所有 Amazon Redshift schema。

```
select database_name, schema_name, schema_owner, schema_type
from svv_redshift_schemas
where database_name in ('db1', 'db2');
```

```
 database_name |    schema_name     | schema_owner | schema_type
---------------+--------------------+--------------+-------------
 db1           | pg_catalog         |            1 | local
 db1           | public             |            1 | local
 db1           | information_schema |            1 | local
 db2           | pg_catalog         |            1 | local
 db2           | public             |            1 | local
 db2           | information_schema |            1 | local
(6 rows)
```

以下示例列出了集群上所有数据库的所有 Amazon Redshift 表或视图。

```
select database_name, schema_name, table_name, table_type
from svv_redshift_tables
where database_name in ('db1', 'db2') and schema_name in ('public');

 database_name | schema_name |     table_name      | table_type
---------------+-------------+---------------------+------------
 db1           | public      | late_bind_view      | VIEW
 db1           | public      | mat_view            | VIEW
 db1           | public      | mv_tbl__mat_view__0 | TABLE
 db1           | public      | regular_view        | VIEW
 db1           | public      | table1              | TABLE
 db2           | public      | table2              | TABLE
(6 rows)
```

以下示例列出了集群上所有数据库的所有 Amazon Redshift 和外部 schema。

```
select database_name, schema_name, schema_owner, schema_type
from svv_all_schemas where database_name in ('db1', 'db2') ;

 database_name |    schema_name     | schema_owner | schema_type
---------------+--------------------+--------------+-------------
 db1           | pg_catalog         |            1 | local
 db1           | public             |            1 | local
 db1           | information_schema |            1 | local
 db2           | pg_catalog         |            1 | local
 db2           | public             |            1 | local
 db2           | information_schema |            1 | local
 db2           | db1_public_sch     |            1 | external
(7 rows)
```

以下示例列出了集群上所有数据库的所有 Amazon Redshift 和外部表。

```
select database_name, schema_name, table_name, table_type
from svv_all_tables
where database_name in ('db1', 'db2') and schema_name in ('public');

 database_name | schema_name |     table_name      | table_type
---------------+-------------+---------------------+------------
 db1           | public      | regular_view        | VIEW
 db1           | public      | mv_tbl__mat_view__0 | TABLE
 db1           | public      | mat_view            | VIEW
 db1           | public      | late_bind_view      | VIEW
 db1           | public      | table1              | TABLE
 db2           | public      | table2              | TABLE
(6 rows)
```

将跨数据库查询与查询编辑器结合使用

您可以使用跨数据库查询访问 Amazon Redshift 集群上任何数据库中的数据，而无需连接到该特定数据库。在任何其他未连接的数据库上运行跨数据库查询时，您只有对这些数据库对象的读取访问权限。

您可以使用用三部分表示法表示的完全限定对象名来查询其他数据库对象。任何数据库对象的完整路径均由三个组成部分组成：数据库名称、schema 和对象的名称。示例是 *database_name.schema_name.object_name*。

将跨数据库查询与查询编辑器 v2 结合使用

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在 Amazon Redshift 查询编辑器 v2 中创建一个集群以使用跨数据库查询。有关更多信息，请参阅《Amazon Redshift 管理指南》中的创建集群。
3. 启用对具有适当权限的查询编辑器的访问权限。有关更多信息，请参阅《Amazon Redshift 管理指南》中的使用查询编辑器 v2 查询数据库。
4. 在导航菜单上，选择查询编辑器 v2，然后连接到集群中的数据库。

   当您首次连接到查询编辑器 v2 时，预设情况下，Amazon Redshift 会显示所连接数据库的资源。
5. 选择您有权查看这些其他数据库的数据库对象的其他数据库。要查看对象，请确保您拥有相应的权限。选择数据库后，Amazon Redshift 将显示数据库中的 schema 列表。

   选择一个 schema 以查看该 schema 中的数据库对象列表。

> **Note**
>
> Amazon Redshift 不直接支持属于 AWS Glue 或联合数据库一部分的查询目录对象。要查询这些对象，请首先创建引用每个数据库中的这些外部数据来源的外部 schema。
> 使用三部分表示法的 Amazon Redshift 跨数据库查询不支持 schema `information_schema` 和 `pg_catalog` 下的元数据表，因为这些元数据视图特定于数据库。

6. （可选）筛选所选 schema 的表或视图的列表。

在 Amazon Redshift 中共享数据

使用 Amazon Redshift 数据共享，您可以在 Amazon Redshift 集群、工作组、AWS 账户和 AWS 区域之间安全地共享对实时数据的访问权限，而无需手动移动或复制数据。由于数据是实时的，只要进行了更新，所有用户就都可以在 Amazon Redshift 中看到最新、最一致的信息。

您可以在预置集群、无服务器工作组、可用区、AWS 账户和 AWS 区域之间共享数据。您可以在集群类型之间共享，也可以在预调配集群和无服务器之间共享。

Amazon Redshift 中的多仓库写入（预览版）

您可以跨同一 AWS 账户内的不同 Amazon Redshift 集群或 Amazon Redshift Serverless 工作组共享数据库对象以进行读取和写入，也可以在不同 AWS 账户之间进行共享。您也可以跨区域写入数据。您可以为不同的表授予 SELECT、INSERT 和 UPDATE 等权限，为不同的架构授予 USAGE 和 CREATE 权限。一旦提交写入事务，所有仓库都可获得实时数据。

有关在 PREVIEW_2023 库中配置数据共享功能的更多信息，请参阅共享对数据的写入访问权限（预览版）。

> **Note**
>
> 目前，ra3.xlplus 集群无法通过数据共享实现多仓库写入。要使用此功能，请创建 ra3.4xl 集群、ra3.16xl 集群或 Amazon Redshift Serverless 工作组。

Amazon Redshift 数据共享概述

使用数据共享，您可以在 Amazon Redshift 集群之间安全、轻松地共享实时数据。

有关如何开始使用数据共享以及使用 AWS Management Console 管理数据共享的信息，请参阅管理数据共享任务。

Amazon Redshift 数据共享使用案例

Amazon Redshift 数据共享对于以下使用案例尤其有用：

- 支持不同类型的业务关键型工作负载 – 使用与多个业务情报 (BI) 或分析集群共享数据的中央提取、转换和加载 (ETL) 集群。此方法提供读取工作负载隔离和单个工作负载的退款。您可以根据特定于工作负载的价格和性能要求调整单个工作负载计算的大小并对其进行扩展。

- 启用跨组协作 – 跨团队和业务组实现无缝协作，以实现更广泛的分析、数据科学和跨产品影响分析。
- 提供数据即服务 – 在整个组织中共享数据即服务。
- 在环境之间共享数据 – 在开发环境、测试环境和生产环境之间共享数据。您可以在不同的粒度级别下共享数据，以提高团队敏捷性。
- 使用 Amazon Redshift 授权访问数据 – 在 AWS Data Exchange 目录中列出 Amazon Redshift 数据集，客户可以在几分钟内找到、订阅和查询这些数据集。

数据共享写入访问权限使用案例（预览版）

写入数据共享有几个重要的使用案例：

- 更新生产者上的业务源数据 – 您可以将数据作为一项服务在整个组织内共享，但随后使用者也可以对源数据执行操作。例如，他们可以回传最新值或确认收到数据。这些只不过是几个可能的业务使用案例。
- 在生产者中插入其他记录 – 使用者可以向原始源数据添加记录。如果需要，可以将其标记为来自使用者。

有关如何对数据共享执行写入操作的具体信息，请参阅共享对数据的写入访问权限（预览版）。

在 Amazon Redshift 中共享不同级别的数据

借助 Amazon Redshift，您可以在不同级别共享数据。这些级别包括数据库、schema、表、视图（包括常规视图、后期绑定视图和实体化视图）和 SQL 用户定义函数 (UDF)。您可以为给定数据库创建多个数据共享。一个数据共享可以包含来自创建共享的数据库中多个 schema 的对象。

通过在共享数据方面具有这种灵活度，您可以获得精细访问控制。您可以为需要访问 Amazon Redshift 数据的不同用户和企业定制此控制。

在 Amazon Redshift 中管理数据一致性

Amazon Redshift 在所有创建器和使用者集群上提供事务一致性，并与所有使用者共享最新且一致的数据视图。

您可以不断更新创建器集群上的数据。事务中的使用者集群上的所有查询都读取共享数据的相同状态。Amazon Redshift 不考虑在使用者集群上的事务开始后提交的创建器集群上的另一个事务更改的数据。在生产者集群上提交数据更改后，使用者集群上的新事务可以立即查询更新的数据。

强一致性移除了在数据共享过程中可能包含无效结果的低保真度业务报告的风险。这个因素对于财务分析或结果可能用于编制用于训练机器学习模型的数据集而言尤为重要。

在 Amazon Redshift 中使用数据共享的注意事项

以下是使用 Amazon Redshift 数据共享时的注意事项。有关数据共享限制的信息，请参阅数据共享的限制。

- 跨区域数据共享会产生额外的跨区域数据传输费用。同一区域内的数据传输不会产生这些费用，只有跨区域的数据传输才会产生这些费用。有关更多信息，请参阅 管理跨区域数据共享的成本控制。
- 作为数据共享用户，您只能继续连接到本地集群数据库。您无法连接到通过数据共享创建的数据库，但可以从这些数据库进行读取。
- 使用者需要支付查询创建者数据所需的所有计算和跨区域数据传输费用。创建者需要为其预置集群或无服务器命名空间中的数据底层存储付费。
- 对共享数据的查询性能取决于使用者集群的计算容量。

管理集群加密

要跨 AWS 账户 共享数据，必须对生产者和使用者集群进行加密。

在 Amazon Redshift 中，您可以为集群开启数据库加密，以帮助保护静态数据。为集群开启加密时，会对集群及其快照的数据块和系统元数据进行加密。您可以在启动集群时开启加密，也可以修改未加密的集群以使用 AWS Key Management Service (AWS KMS) 加密。有关 Amazon Redshift 数据库加密的更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 数据库加密。

为了保护传输中的数据，所有数据都在传输过程中通过生产者集群的加密模式进行加密。加载数据时，使用者集群采用此加密模式。然后，使用者集群作为普通加密集群运行。创建者和使用者之间的通信也使用共享密钥模式进行加密。有关传输中加密的更多信息，请参阅传输中加密。

数据共享的限制

以下是在 Amazon Redshift 中使用数据共享时的限制：

- 数据共享在所有预置 ra3 集群类型（ra3.16xlarge、ra3.4xlarge 和 ra3.xlplus）和 Amazon Redshift Serverless 上均受支持。其他集群类型不支持数据共享。
- 对于跨账户以及跨区域数据共享，必须对创建者和使用者集群和无服务器命名空间进行加密。这是出于安全目的。但是，二者不需要共享相同的加密密钥。
- 通过数据共享，您只能共享 SQL UDF。不支持 Python 和 Lambda UDF。

- 如果生产者数据库有特定的排序规则，请在使用者数据库中使用相同的排序规则设置。
- Amazon Redshift 不支持将外部架构、表或外部表上的后期绑定视图添加到数据共享。
- Amazon Redshift 不支持生产者集群上的嵌套 SQL 用户定义的函数。
- Amazon Redshift 不支持共享具有交错排序键的表以及引用具有交错排序键的表的视图。
- 使用者无法将数据共享对象添加到另一个数据共享。此外，使用者无法将引用数据共享对象的视图添加到另一个数据共享。
- Amazon Redshift 不支持访问在“准备”和“执行”访问之间发生了并发 DDL 的数据共享对象。
- Amazon Redshift 不支持通过数据共享来共享存储过程。
- Amazon Redshift 不支持共享元数据系统视图和系统表。

可进行数据共享的区域

下表列出了数据共享功能的可用性。

区域	同区域数据共享	跨区域数据共享	AWS Lake Formation 监管的数据共享
美国东部（弗吉尼亚州北部）(us-east-1)	是	是	是
美国东部（俄亥俄州）(us-east-2)	是	是	是
美国西部（北加利福尼亚）(us-west-1)	是	是	是
美国西部（俄勒冈州）(us-west-2)	是	是	是
亚太地区（孟买）(ap-south-1)	是	是	是
亚太地区（海得拉巴）(ap-south-2)	是	否	否
亚太地区（东京）(ap-northeast-1)	是	是	是

区域	同区域数据共享	跨区域数据共享	AWS Lake Formation 监管的数据共享
亚太地区（新加坡）(ap-southeast-1)	是	是	是
亚太地区（悉尼）(ap-southeast-2)	是	是	是
亚太地区（雅加达）；（ap-southeast-3）	是	否	否
亚太地区（墨尔本）(ap-southeast-4)	是	否	否
亚太地区（首尔）(ap-northeast-2)	是	是	是
亚太地区（大阪）(ap-northeast-3)	是	否	否
非洲（开普敦）(af-south-1)	是	是	不支持
加拿大西部（卡尔加里）（ca-west-1）	是	否	否
加拿大（中部）(ca-central-1)	是	是	是
欧洲地区（法兰克福）(eu-central-1)	是	是	是
欧洲（苏黎世）(eu-central-2)	是	否	否
欧洲地区（爱尔兰）(eu-west-1)	是	是	是

区域	同区域数据共享	跨区域数据共享	AWS Lake Formation 监管的数据共享
欧洲（伦敦）(eu-west-2)	是	是	是
欧洲地区（巴黎）（eu-west-3）	是	是	是
欧洲（米兰）(eu-south-1)	是	否	否
欧洲（西班牙）(eu-south-2)	是	否	否
欧洲地区（斯德哥尔摩）(eu-north-1)	是	是	是
中东（阿联酋）(me-central-1)	是	否	否
中东（巴林）(me-south-1)	是	否	否
以色列（特拉维夫）（il-central-1）	是	否	否
南美洲（圣保罗）（sa-east-1）	是	是	是
AWS GovCloud（美国东部）(us-gov-east-1)	是	否	是
AWS GovCloud（美国西部）(us-gov-west-1)	是	否	是

数据共享多仓库写入的区域可用性

在 PREVIEW_2023 库中，数据共享具有写入操作功能和更精细的共享功能。有关如何配置这些内容的更多信息，请参阅共享对数据的写入访问权限（预览版）。有关提供预览功能的区域的信息，请参阅可进行数据共享的区域（预览版）。

什么是数据共享？

数据共享是在 Amazon Redshift 中共享数据的单位。使用数据共享可在同一个 AWS 账户或不同 AWS 账户中共享数据。还可以在不同 Amazon Redshift 集群间共享数据，以进行读取。

每个数据共享都与 Amazon Redshift 集群上的特定数据库相关联。

生产者集群管理员可以创建数据共享并添加数据共享对象，以与其他集群共享数据（称为出站共享）。使用者集群管理员可以从其他集群接收数据共享，称为入站共享。有关生产者和使用者的详细信息，请参阅数据共享生产者和使用者。

数据共享对象是来自集群上特定数据库的对象，生产者集群管理员可以将其添加到要与数据使用者共享的数据共享中。对于数据使用者来说，数据共享对象是只读的。数据共享对象的示例包括表、视图和用户定义的函数。您可以随时在创建数据共享或编辑数据共享时将数据共享对象添加到数据共享。

当调整集群大小或暂停生产者集群时，数据共享将继续工作。

数据共享有不同的类型。

主题

- 标准数据共享
- AWS Data Exchange 数据共享
- AWS Lake Formation 托管式数据共享
- 数据共享生产者和使用者

标准数据共享

通过标准数据共享，您可以在预调配集群、无服务器工作组、可用区、AWS 账户和 AWS 区域之间共享数据。您可以在集群类型之间共享，也可以在预调配集群和 Amazon Redshift Serverless 之间共享。

要共享数据，请注意以下预调配集群、无服务器命名空间和 AWS 账户标识符：

- 预调配集群命名空间是标识 Amazon Redshift 预调配集群的标识符。命名空间全局唯一标识符 (GUID) 会在创建预调配集群期间自动创建并附加到集群。命名空间 Amazon 资源名称（ARN）的格

式为 arn:{partition}:redshift:{region}:{account-id}:namespace:{namespace-guid}。您可以在 Amazon Redshift 控制台的集群详细信息页面上查看预调配集群的命名空间。

在数据共享工作流中，命名空间 GUID 值和集群命名空间 ARN 用于与 AWS 账户中的集群共享数据。您还可以通过使用 `current_namespace` 函数为当前集群查找命名空间。

- 无服务器命名空间 是标识 Amazon Redshift Serverless 的标识符。命名空间全局唯一标识符 (GUID) 会在创建 Amazon Redshift Serverless 期间自动创建并附加到实例。无服务器命名空间 ARN 的格式为 arn:{partition}:redshift-serverless:{region}:{account-id}:namespace/{namespace-guid}。
- AWS 账户可以是数据共享的使用者，每一个都用 12 位数的 AWS 账户 ID 表示。

对于标准数据共享，请考虑以下事项：

- 删除创建器集群时，Amazon Redshift 会删除由创建者集群创建的数据共享。备份和还原创建器集群时，创建的数据共享仍然保留在已恢复的集群上。但是，授予其他集群的数据共享权限在还原的集群上不再有效。将数据共享的使用权限重新授予所需的使用者集群。使用者集群上的使用者数据库指向拍摄快照所在的原始集群中的数据共享。要从还原的集群中查询共享数据，使用者集群管理员可创建一个不同的数据库。或者，管理员可以删除并重新创建现有的使用者数据库，以使用新还原的集群中的数据共享。
- 当从快照中删除并还原使用者集群时，以前共享到此集群的访问权限将不再有效且可见。如果还原的使用者集群上仍然需要访问数据共享，则创建者集群管理员必须再次授予已恢复的使用者集群使用数据共享的权限。使用者集群管理员必须删除从非活动数据共享创建的任何过时的使用者数据库。然后，在创建者重新授予权限后，管理员必须从数据共享中重新创建使用者数据库。由于恢复的集群上的集群命名空间 GUID 与原始集群不同，因此在从备份还原使用者或生产者集群时重新授予数据共享权限。

AWS Data Exchange 数据共享

AWS Data Exchange 数据共享是通过 AWS Data Exchange 共享您的数据的授权单位。AWS 管理与 AWS Data Exchange 订阅以及 Amazon Redshift 数据共享使用情况相关的所有计费和付款。经批准的数据提供商可以添加 AWS Data Exchange 数据共享到 AWS Data Exchange 产品。当客户订阅带 AWS Data Exchange 数据共享的产品时，他们可以访问该产品中的数据共享。

AWS Data Exchange for Amazon Redshift 使您可以通过 AWS Data Exchange 轻松获得 Amazon Redshift 数据的访问许可。当客户订阅具备 AWS Data Exchange 数据共享的产品时，AWS Data Exchange 自动在该产品中包含的所有 AWS Data Exchange 数据共享中将该客户添加为数据使用者。自动生成发票，并通过 AWS Marketplace Entitlement Service 集中收取付款并自动支付。

提供商可以在 Amazon Redshift 中以精细级别许可数据，例如架构、表、视图和用户定义的函数。您可以使用跨多个 AWS Data Exchange 产品的相同 AWS Data Exchange 数据共享。使用者可以使用添加到 AWS Data Exchange 数据共享的任何对象。创建者可以查看由 AWS Data Exchange 使用 Amazon Redshift API 操作、SQL 命令和 Amazon Redshift 控制台代表其管理的所有 AWS Data Exchange 数据共享。订阅产品 AWS Data Exchange 数据共享的客户对数据共享中的对象具有只读访问权限。

想要使用第三方创建者数据的客户可以浏览 AWS Data Exchange 目录，以浏览和订阅 Amazon Redshift 中的数据集。在客户的 AWS Data Exchange 订阅处于活跃状态后，他们可以从集群中的数据共享创建数据库并在 Amazon Redshift 中查询数据。

AWS Data Exchange 数据共享的工作原理

以创建者管理员的身份管理 AWS Data Exchange 数据共享。

如果您是数据创建者（也称为 AWS Data Exchange 上的提供商），您可以创建连接到 Amazon Redshift 数据库的 AWS Data Exchange 数据共享。要在 AWS Data Exchange 添加对产品的 AWS Data Exchange 数据共享，您必须是注册的 AWS Data Exchange 提供商。

有关如何开始使用 AWS Data Exchange 数据共享的更多信息，请参阅在 AWS Data Exchange 上共享许可的 Amazon Redshift 数据。

以活跃 AWS Data Exchange 订阅的使用者身份使用 AWS Data Exchange 数据共享。

如果您是活动 AWS Data Exchange 订阅的使用者（也称为 AWS Data Exchange 上的订阅者），您可以浏览 AWS Data Exchange 控制台上的 AWS Data Exchange 目录，以发现包含 AWS Data Exchange 数据共享的产品。

订阅包含 AWS Data Exchange 数据共享的产品后，从集群数据共享中创建数据库。然后，您可以直接在 Amazon Redshift 中查询数据，无需提取、转换和加载数据。

有关如何开始使用 AWS Data Exchange 数据共享的更多信息，请参阅在 AWS Data Exchange 上共享许可的 Amazon Redshift 数据。

对于 AWS Data Exchange 数据共享，请考虑以下事项：

- 删除生产者集群时，Amazon Redshift 会删除由生产者集群创建的数据共享。备份和还原创建器集群时，创建的数据共享仍然保留在已恢复的集群上。为了使数据订阅者能够继续访问数据，请再次创建 AWS Data Exchange 数据共享并将其发布到产品的数据集中。使用者集群上的使用者数据库指向拍摄快照所在的原始集群中的数据共享。要从还原的集群中查询共享数据，使用者集群管理员会创

建一个不同的数据库，或删除并重新创建现有的使用者数据库，以使用新还原的集群中的 AWS Data Exchange 数据共享。

- 当从快照中删除并还原使用者集群时，以前共享到此集群的访问权限将不再有效且可见。使用者集群管理员必须删除从非活动数据共享创建的任何过时的使用者数据库，并在生产者重新授予权限后，从数据共享中重新创建使用者数据库。由于还原的集群上的集群命名空间 GUID 与原始集群不同，因此，在从备份还原生产者集群时重新授予数据共享权限。
- 如果您有任何 AWS Data Exchange 数据共享，我们建议您不要删除集群。执行这种类型的更改可能会违反 AWS Data Exchange 中的数据产品条款。

使用 Amazon Redshift AWS Data Exchange 时的注意事项

使用 Amazon Redshift AWS Data Exchange 时，请考虑以下事项：

- 创建者和使用者都必须使用 RA3 实例类型才能使用 Amazon Redshift 数据共享。创建者必须将 RA3 实例类型与最新的 Amazon Redshift 集群版本一起使用。
- 必须对创建者和使用者集群进行加密。
- 您必须注册为 AWS Data Exchange 提供商才可在 AWS Data Exchange 上发布产品，包括包含 AWS Data Exchange 数据共享的产品。有关更多信息，请参阅以提供商身份开始使用。
- 您无需成为注册 AWS Data Exchange 提供商即可通过 AWS Data Exchange 查找、订阅和查询 Amazon Redshift 数据。
- 要控制对数据的访问，请创建可公开访问设置已开启的 AWS Data Exchange 数据共享。要更改 AWS Data Exchange 数据共享，以关闭可公开访问的设置，请将会话变量设置为允许 ALTER DATASHARE SET PUBLICACCESSIBLE FALSE。有关更多信息，请参阅 ALTER DATASHARE 使用说明。
- 创建者无法从 AWS Data Exchange 数据共享手动添加或删除使用者，因为对数据共享的访问是基于对包含 AWS Data Exchange 数据共享的 AWS Data Exchange 产品的活跃订阅。
- 创建者无法查看使用者运行的 SQL 查询。他们只能通过只有创建者才能访问的 Amazon Redshift 表查看元数据，例如查询数量或使用者查询的对象。有关更多信息，请参阅 监控和审计 Amazon Redshift 中的数据共享。
- 我们建议您将数据共享设置为可公开访问。如果您不这样设置，在 AWS Data Exchange 上具备可公开访问的使用者集群的订阅者将无法使用您的数据共享。
- 我们建议您不要使用 DROP DATASHARE 语句删除共享给其它 AWS 账户 的 AWS Data Exchange 数据共享。如果您这样做，有权访问数据共享的 AWS 账户 将失去访问权限。此操作不可逆。执行这种类型的更改可能会违反 AWS Data Exchange 中的数据产品条款。如果您要删除 AWS Data Exchange 数据共享，请参阅DROP DATASHARE 使用说明。

- 对于跨区域数据共享，您可以创建 AWS Data Exchange 数据共享，以共享许可数据。
- 当使用来自其他区域的数据时，使用者需要支付从生产者区域到使用者区域的跨区域数据传输费。

AWS Lake Formation 托管式数据共享

使用 AWS Lake Formation，您可以集中定义和强制执行 Amazon Redshift 数据共享的数据库、表、列和行级访问权限，并限制用户对数据共享内对象的访问。通过 Lake Formation 共享数据，您可以在 Lake Formation 中定义权限，并将这些权限应用于任何数据共享及其对象。例如，如果您有一个包含员工信息的表，则可以使用 Lake Formation 的列级筛选条件，防止非人力资源部门的员工查看社会保险号等个人身份信息（PII）。有关数据筛选条件的更多信息，请参阅《AWS Lake Formation 开发人员指南》中的 Lake Formation 中的数据筛选和单元格级安全性。

您还可以在 Lake Formation 中使用标签来配置 Lake Formation 上资源的权限。有关更多信息，请参阅 Lake Formation 基于标签的访问控制。

Amazon Redshift 目前支持在同一账户内或不同账户间共享时通过 Lake Formation 共享数据。目前不支持跨区域共享。

以下是有关如何使用 Lake Formation 控制数据共享权限的简要概述：

1. 在 Amazon Redshift 中，创建者集群或工作组管理员在创建者集群或工作组上创建数据共享，并向 Lake Formation 账户授予使用权限。
2. 创建者集群或工作组管理员授权 Lake Formation 账户访问数据共享。
3. Lake Formation 管理员发现并注册数据共享。他们还必须发现他们有权访问的 AWS Glue ARN，并将数据共享与 AWS Glue Data Catalog ARN 关联起来。如果您使用 AWS CLI，则可以通过 Redshift CLI 操作 `describe-data-shares` 和 `associate-data-share-consumer` 发现和接受数据共享。要注册数据共享，请使用 Lake Formation CLI 操作 `register-resource`。
4. Lake Formation 管理员在 AWS Glue Data Catalog 中创建联合数据库，并配置 Lake Formation 权限以控制用户对数据共享中对象的访问。有关 AWS Glue 中的联合数据库的更多信息，请参阅管理 Amazon Redshift 数据共享中数据的权限。
5. Lake Formation 管理员发现他们有权访问的 AWS Glue 数据库，并将数据共享与 AWS Glue Data Catalog ARN 关联起来。
6. Redshift 管理员发现他们有权访问的 AWS Glue 数据库 ARN，使用 AWS Glue 数据库 ARN 在 Amazon Redshift 使用者集群中创建外部数据库，并将使用权限授予使用 IAM 凭证进行身份验证的数据库用户，以便开始查询 Amazon Redshift 数据库。
7. 数据库用户可以使用视图 SVV_EXTERNAL_TABLES 和 SVV_EXTERNAL_COLUMNS 来查找 AWS Glue 数据库中他们有权访问的所有表或列，然后他们可以查询 AWS Glue 数据库的表。

8. 当生产者集群或工作组管理员决定不再与使用者集群共享数据时，生产者集群管理员可以撤销使用权限、取消授权或者从 Redshift 中删除数据共享。系统不会自动删除 Lake Formation 中的关联权限和对象。

有关以生产者集群或工作组管理员身份与 AWS Lake Formation 共享数据共享的更多信息，请参阅以创建者身份使用 Lake Formation 托管的数据共享。要使用生产者集群或工作组中的共享数据，请参阅以使用者身份使用 Lake Formation 托管的数据共享。

将 AWS Lake Formation 与 Amazon Redshift 一起使用时的注意事项和限制

以下是通过 Lake Formation 共享 Amazon Redshift 数据时的注意事项和限制。有关数据共享注意事项和限制的信息，请参阅在 Amazon Redshift 中使用数据共享时的注意事项。有关 Lake Formation 限制的信息，请参阅关于在 Lake Formation 中使用 Amazon Redshift 数据共享的说明。

- 目前不支持跨区域与 Lake Formation 共享数据共享。
- 如果为共享关系上的用户定义了列级别筛选条件，则执行 `SELECT *` 操作将仅返回用户有权访问的列。
- 不支持 Lake Formation 中单元级别的筛选条件。
- 如果您创建了视图及其表并与 Lake Formation 共享，则可以配置筛选条件来管理表的访问权限，在使用者集群用户访问共享对象时，Amazon Redshift 会强制执行 Lake Formation 定义的策略。当用户访问与 Lake Formation 共享的视图时，Redshift 仅强制执行在视图上定义的 Lake Formation 策略，而不是视图中包含的表上定义的策略。但是，当用户直接访问表时，Redshift 会在表上强制执行已定义的 Lake Formation 策略。
- 如果某个共享表配置了 Lake Formation 筛选条件，则无法基于该表对使用者创建实体化视图。
- Lake Formation 管理员必须具有数据湖管理员权限和接受数据共享所需的权限。
- 生产者使用者集群必须为使用最新 Amazon Redshift 集群版本或无服务器工作组的 RA3 集群，才能通过 Lake Formation 共享数据共享。
- 必须对创建者和使用者集群进行加密。
- 将数据共享内容共享给 Lake Formation 时，忽略生产者集群或工作组中实现的 Redshift 行级和列级访问控制策略。Lake Formation 管理员必须在 Lake Formation 中配置这些策略。生产者集群或工作组管理员可以使用 ALTER TABLE 命令关闭表的 RLS。
- 只有同时有权访问 Redshift 和 Lake Formation 的用户才能通过 Lake Formation 共享数据共享。

数据共享生产者和使用者

数据创建器（也称为数据共享创建器或数据共享生成器）是您想要从中共享数据的集群。创建者集群管理员和数据库拥有者可以使用 CREATE DATASHARE 命令创建数据共享。您可以从该数据库中添加架构、表、视图和 SQL 用户定义的函数（UDF）等对象，而您希望生产者集群与使用者集群共享这些对象。

AWS Data Exchange 数据共享的数据提供商（也称为 AWS Data Exchange 上的提供商）可通过 AWS Data Exchange 许可数据。经批准的提供商可以将 AWS Data Exchange 数据共享添加到 AWS Data Exchange 产品。

当客户订阅带 AWS Data Exchange 数据共享的产品时，AWS Data Exchange 自动将该客户添加为该产品包含的所有 AWS Data Exchange 数据共享上的数据使用者。当客户的订阅结束时，AWS Data Exchange 还将从 AWS Data Exchange 数据共享中删除所有客户。AWS Data Exchange 还自动管理使用 AWS Data Exchange 数据共享的付费产品的计费、发票、付款收集和付款分配事宜。有关更多信息，请参阅 AWS Data Exchange 数据共享。如需注册为 AWS Data Exchange 数据提供商，请参阅以提供商身份开始使用。

数据使用者（也称为数据共享使用者或数据共享使用者）是从创建器集群接收数据共享的集群。

共享数据的 Amazon Redshift 集群可以位于相同或不同的 AWS 账户或不同的 AWS 区域中，因此，您可以跨组织共享数据并与其他各方协作。使用者集群管理员将收到其被授权使用的数据共享，并查看每个数据共享的内容。要使用共享的数据，使用者集群管理员可从数据共享中创建 Amazon Redshift 数据库。然后，管理员将数据库的权限分配给使用者集群中的用户和角色。授予权限后，用户和角色可以将共享对象列为标准元数据查询的一部分，以及使用者集群上的本地数据。他们可以立即开始查询。

如果您是活动 AWS Data Exchange 订阅的使用者（也称为 AWS Data Exchange 上的订阅者），您可以在 Amazon Redshift 中查找、订阅和查询最新的精细数据，无需提取、转换和加载数据。有关更多信息，请参阅 AWS Data Exchange 数据共享。

Amazon Redshift 中的数据共享工作原理

管理不同状态下的数据共享

使用跨账户数据共享，需要您执行操作的数据共享具有不同状态。您的数据共享可以具有活动状态、需要操作或非活动状态。

下面介绍了每个数据共享状态及其需要的操作：

- 创建器集群管理员创建数据共享时，生产者集群上的数据共享状态为待处理授权。创建器集群管理员可以授权数据使用者访问数据共享。使用者集群管理员没有任何操作。
- 当创建器集群管理员为数据共享授权时，创建器集群上的数据共享状态为已授权。创建器集群管理员没有任何操作。当数据共享的数据使用者至少存在一个关联时，数据共享状态会从已授权变为活动状态。

  使用者集群上的数据共享状态随后变为可用（需要在 Amazon Redshift 控制台上执行操作）。使用者集群管理员可以将数据共享与数据使用者相关联，或拒绝数据共享。使用者集群管理员还可以使用 AWS CLI 命令 `describeDatashareforConsumer` 查看数据共享的状态。或者，管理员可以使用 CLI 命令 `describeDatashare` 并提供数据共享 Amazon 资源名称（ARN）以查看数据共享的状态。
- 当使用者集群管理员将数据共享与数据使用者相关联时，创建器集群上的数据共享状态将变为活动状态。当数据共享的数据使用者至少存在一个关联时，数据共享状态会从已授权变为活动状态。创建器集群管理员无需执行任何操作。

  使用者集群上的数据共享状态变为活动状态。使用者集群管理员不需要执行任何操作。
- 当使用者集群管理员从数据共享中删除使用者关联时，数据共享状态将变为活动状态或已授权。当数据共享至少与另一个数据使用者存在一个关联时，状态变为活动状态。当任何使用者与创建器集群上的数据共享没有任何关联时，状态变为已授权。创建器集群管理员没有任何操作。

  如果所有关联均被删除，使用者集群上的数据共享状态变为需要操作。当数据共享可供使用者使用时，使用者集群管理员可以将数据共享与数据使用者重新关联。
- 当使用者集群管理员拒绝数据共享时，创建器集群上的数据共享状态变为需要操作，使用者集群上的数据共享状态变为已拒绝。创建器集群管理员可以重新授权数据共享。使用者集群管理员没有任何操作。
- 当创建器集群管理员从数据共享中删除授权时，创建器集群上的数据共享的状态将变为需要操作。如果需要，创建器集群管理员可以选择重新授权数据共享。使用者集群管理员不需要执行任何操作。

共享数据共享

仅当您在不同的 Amazon Redshift 预置集群或无服务器工作组之间共享数据时才需要使用数据共享。在同一个集群中，只要您对另一个数据库中的对象具有所需的权限，就可以使用由三部分组成的简单表示法 `database.schema.table` 查询另一个数据库。

在 Amazon Redshift 中管理数据共享的权限

作为创建器集群管理员，您可以保留对正在共享的数据集的控制权。您可以将新对象添加到数据共享中或将其从数据共享中删除。您还可以授予或撤消对使用者集群、AWS 账户或 AWS 区域的整体数据共享的访问权。当权限被撤消时，使用者集群立即失去对共享对象的访问权限，并且不能在 SVV_DATASHARES 中的 INBOUND 数据共享列表中看到它们。

以下示例创建数据共享 `salesshare`，添加架构 `public`，然后将表 `public.tickit_sales_redshift` 添加到 `salesshare`。它还授予指定的集群命名空间对 `salesshare` 的使用权限。

```
CREATE DATASHARE salesshare;

ALTER DATASHARE salesshare ADD SCHEMA public;

ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;

GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

对于 CREATE DATASHARE，超级用户和数据库拥有者可以创建数据共享。有关更多信息，请参阅 CREATE DATASHARE。对于 ALTER DATASHARE，对要添加或删除的数据共享具有所需权限的数据共享拥有者可以更改数据共享。有关更多信息，请参阅 ALTER DATASHARE。

作为创建器管理员，当您删除数据共享时，它将停止在使用者集群上列出。在使用者集群上通过删除的数据共享中创建的数据库和 schema 引用继续存在，其中没有对象。使用者集群管理员必须手动删除这些数据库。

在使用者方面，使用者集群管理员可以通过从数据共享创建数据库来确定哪些用户和角色应该访问共享数据。根据您在创建数据库时选择的选项，您可以按如下方式控制对数据库的访问。有关从数据共享创建数据库的更多信息，请参阅CREATE DATABASE。

在不使用 WITH PERMISSIONS 子句的情况下创建数据库

管理员可以在数据库或 schema 级别控制访问权限。要控制 Schema 级别的访问权限，管理员必须通过从数据共享创建的 Amazon Redshift 数据库创建外部 Schema。

以下示例授予在数据库级别和 schema 级别访问共享表的权限。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE sales_db SCHEMA 'public';

GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

要进一步限制访问，您可以基于共享对象创建视图，以仅显示必要的数据。然后，您可以使用这些视图向用户和角色授予访问权限。

授予用户访问数据库或 Schema 的权限后，他们将有权访问该数据库或 Schema 中的所有共享对象。

使用 WITH PERMISSIONS 子句创建数据库

在授予对数据库或架构的使用权限后，管理员可以使用与本地数据库或架构相同的权限授予过程来进一步控制访问权限。如果没有单个对象的权限，用户即使获得了 USAGE 权限，也无法访问数据共享数据库或架构中的任何对象。

以下示例授予在数据库级别访问共享表的权限。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
GRANT USAGE FOR SCHEMAS IN DATABASE sales_db TO Bob;
GRANT SELECT ON sales_db.public.tickit_sales_redshift TO Bob;
```

在授予用户访问数据库或架构的权限后，还需要授予他们访问数据库或架构中任何对象的相关权限。

使用 WITH PERMISSIONS 进行精细共享（预览版）

允许集群或 Serverless 工作组查询数据共享

此步骤假设数据共享来自另一个集群或您账户中的 Amazon Redshift Serverless 命名空间，或者来自另一个账户并已与您正在使用的命名空间相关联。

1. 使用者数据库管理员可以从数据共享中创建数据库。

   ```
   CREATE DATABASE my_ds_db [WITH PERMISSIONS] FROM DATASHARE my_datashare OF
    NAMESPACE 'abc123def';
   ```

   如果您使用 WITH PERMISSIONS 创建数据库，则可以向不同的用户和角色授予对数据共享对象的精细权限。否则，所有获得对数据共享数据库的 USAGE 权限的用户和角色都将获得对数据共享数据库中所有对象的所有权限。

2. 以下内容显示如何向 Redshift 数据库用户或角色授予权限。您必须连接到本地数据库才能运行这些语句。如果在运行 grant 语句之前在数据共享数据库上执行 USE 命令，则无法运行这些语句。

```
GRANT USAGE ON DATABASE my_ds_db TO ROLE data_eng;
GRANT CREATE, USAGE ON SCHEMA my_ds_db.my_shared_schema TO ROLE data_eng;
GRANT ALL ON ALL TABLES IN SCHEMA my_ds_db.my_shared_schema TO ROLE data_eng;

GRANT USAGE ON DATABASE my_ds_db TO bi_user;
GRANT USAGE ON SCHEMA my_ds_db.my_shared_schema TO bi_user;
GRANT SELECT ON my_ds_db.my_shared_schema.table1 TO bi_user;
```

在 Amazon Redshift 数据共享中使用视图

创建器集群可以共享常规视图、后期绑定视图和实体化视图。共享常规或后期绑定视图时，不必共享基表。下表介绍了数据共享如何支持视图。

视图名称	可以将此视图添加到数据共享中吗？	使用者是否可以在跨集群的数据共享对象上创建此视图？
常规视图	是	否
后期绑定视图	是	是
实体化视图	是	是，但仅限于完整刷新

以下查询显示了数据共享支持的常规视图的输出。有关常规视图定义的信息，请参阅CREATE VIEW。

```
SELECT * FROM tickit_db.public.myevent_regular_vw
ORDER BY eventid LIMIT 5;

   eventid  |  eventname
  ----------+-------------
     3835   | LeAnn Rimes
     3967   | LeAnn Rimes
     4856   | LeAnn Rimes
     4948   | LeAnn Rimes
```

```
     5131   | LeAnn Rimes
```

以下查询显示了数据共享支持的后期绑定视图的输出。有关后期绑定视图定义的信息，请参阅CREATE VIEW。

```
SELECT * FROM tickit_db.public.event_lbv
ORDER BY eventid LIMIT 5;

 eventid | venueid | catid | dateid |          eventname           |      starttime
 --------+---------+-------+--------+------------------------------
+---------------------
     1   |   305   |   8   |  1851  |        Gotterdammerung       | 2008-01-25
 14:30:00
     2   |   306   |   8   |  2114  |           Boris Godunov      | 2008-10-15
 20:00:00
     3   |   302   |   8   |  1935  |              Salome          | 2008-04-19
 14:30:00
     4   |   309   |   8   |  2090  |  La Cenerentola (Cinderella) | 2008-09-21
 14:30:00
     5   |   302   |   8   |  1982  |          Il Trovatore        | 2008-06-05
 19:00:00
```

以下查询显示了数据共享支持的实体化视图的输出。有关实体化视图定义的信息，请参阅CREATE MATERIALIZED VIEW。

```
SELECT * FROM tickit_db.public.tickets_mv;

  catgroup | qtysold
 ----------+---------
  Concerts | 195444
    Shows  | 149905
```

您可以在创建器集群中的所有租户之间维护公用表。您还可以将按维度列筛选的数据子集，例如 tenant_id（account_id 或 namespace_id），共享到使用者集群中。为此，您可以在这些 ID 列上使用筛选器，例如 current_aws_account = tenant_id，以在基表上定义一个视图。在使用者方面，当您查询视图时，只能看到符合您账户条件的行。为此，您可以使用 Amazon Redshift 上下文函数 current_aws_account 和 current_namespace。

以下查询返回当前 Amazon Redshift 集群所在的账户 ID。如果您已连接到 Amazon Redshift，则可以运行此查询。

```
select current_user, current_aws_account;

current_user | current_aws_account
-------------+--------------------
dwuser       |    111111111111
(1row)
```

以下查询返回当前 Amazon Redshift 集群的命名空间。如果已连接到数据库，则可以运行此查询。

```
select current_user, current_namespace;

current_user | current_namespace
-------------+--------------------------------------
dwuser       | 86b5169f-01dc-4a6f-9fbb-e2e24359e9a8
(1 row)
```

对数据共享中的实体化视图进行增量刷新

共享基表时，Amazon Redshift 支持对使用者数据共享中的实体化视图进行增量刷新。增量刷新是一项操作，其中 Amazon Redshift 可识别上次刷新后发生的一个或多个基表中的更改，并仅更新实体化视图中的相应记录。有关此行为的更多信息，请参阅 CREATE MATERIALIZED VIEW。

使用 IAM 策略管理对数据共享 API 操作的访问

要控制对数据共享 API 操作的访问，请使用基于 IAM 操作的策略。有关如何管理 IAM 策略的信息，请参阅《IAM 用户指南》中的管理 IAM 策略。

有关使用数据共享 API 操作所需的权限信息，请参阅《Amazon Redshift 管理指南》中的使用数据共享 API 操作所需的权限。

为了提高跨账户数据共享的安全性，您可以将条件密钥 `ConsumerIdentifier` 用于 `AuthorizeDataShare` 和 `DeauthorizeDataShare` API 操作。这样您就可以明确控制哪些 AWS 账户 可以调用两个 API 操作。

您可以拒绝授权或取消授权任何不属于您自己账户的使用者数据共享。为此，请指定 IAM 策略中的 AWS 账户 数字。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Deny",
            "Action": [
                "redshift:AuthorizeDataShare",
                "redshift:DeauthorizeDataShare"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "redshift:ConsumerIdentifier": "555555555555"
                }
            }
        }
    ]
}
```

您可以允许拥有 DataShareArn **testshare2** 的创建者明确与拥有 IAM 策略中 111122223333 的 AWS 账户 使用者共享。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "redshift:AuthorizeDataShare",
                "redshift:DeauthorizeDataShare"
            ],
            "Resource": "arn:aws:redshift:us-
east-1:666666666666:datashare:af06285e-8a45-4ee9-b598-648c218c8ff1/testshare2",
            "Condition": {
                "StringEquals": {
                    "redshift:ConsumerIdentifier": "111122223333"
                }
            }
        }
    ]
}
```

查询数据共享

在 Amazon Redshift 中访问共享数据

您可以使用标准 SQL 接口、JDBC 或 ODBC 驱动程序以及数据 API 来发现共享数据。您还可以通过熟悉的业务情报 (BI) 和分析工具查询具有高性能的数据。您可以通过引用其他 Amazon Redshift 数据库中的对象来执行查询，这些数据库既是您有权访问的集群的本地对象，也可以从您的集群中远程访问。

您可以简单地通过与集群中的本地数据库保持连接来完成此操作。然后您可以从数据共享创建使用者数据库以使用共享数据。

完成此操作之后，您可以执行连接数据集的跨数据库查询。您可以使用 3 部分表示法（*consumer_database_name*.*schema_name*.*table_name*）查询使用者数据库中的对象。您还可以使用指向使用者数据库中架构的外部架构链接进行查询。您可以在同一查询中查询本地数据以及从其它集群共享的数据。此类查询可以引用来自当前已连接数据库和其它未连接数据库中的对象，包括从数据共享创建的使用者数据库。

在 Amazon Redshift 中访问数据共享元数据

为了帮助集群管理员发现数据共享，Amazon Redshift 提供了一组元数据视图来列出数据共享。这些视图列出了在您的集群中创建的数据共享，以及从同一账户内的其他集群和其他账户，或其他 AWS 区域（预览版）接收的数据共享。这些视图将显示以下信息：

- 由集群共享和接收的数据共享
- 数据共享中数据库对象的内容，包括基本共享元数据、对象和使用者

使用 SVV_DATASHARES 查看在集群中创建（出站）并与其他人共享（入站）的所有数据共享的列表。有关更多信息，请参阅 SVV_DATASHARES。

使用 SVV_DATASHARE_CONSUMERS 查看数据使用者的列表。有关更多信息，请参阅 SVV_DATASHARE_CONSUMERS。

使用 SVV_DATASHARE_OBJECTS 查看在集群中创建（出站）并与其他人共享（入站）的所有数据共享中的对象的列表。有关更多信息，请参阅 SVV_DATASHARE_OBJECTS。

将 Amazon Redshift 数据共享与业务情报工具集成

要将数据共享与业务情报 (BI) 工具集成，我们建议您使用 Amazon Redshift JDBC 或 ODBC 驱动程序。

Amazon Redshift JDBC 和 ODBC 驱动程序支持驱动程序中的 `GetCatalogs` API 操作，以返回所有数据库的列表，包括从数据共享创建的数据库。驱动程序还支持下游操作，例如 `GetSchemas`、`GetTables` 等，这些操作从 `GetCatalogs` 返回的所有数据库中返回数据。即使在调用中没有明确指定目录时，驱动程序也会提供此支持。有关 JDBC 或 ODBC 驱动程序的更多信息，请参阅《Amazon Redshift 管理指南》中的配置 Amazon Redshift 中的连接。

您无法直接连接到通过数据共享创建的使用者数据库。连接到您的集群上的本地数据库。如果您的工具中有连接切换用户界面，则数据库列表应仅包含本地集群数据库。该列表应排除从数据共享创建的使用者数据库，以提供最佳体验。您可以使用 SVV_REDSHIFT_DATABASES 视图中的选项来筛选数据库。

监控和审计 Amazon Redshift 中的数据共享

通过审计数据共享，创建器可以跟踪数据共享的演变。例如，审计有助于跟踪何时创建了数据共享、添加或删除了对象以及向 Amazon Redshift 集群、AWS 账户或 AWS 区域授予或撤消了权限。

除了审计之外，创建器和使用者还会跟踪各种粒度（如账户、集群和对象级别）下的数据共享使用情况。有关跟踪使用情况和审计视图的更多信息，请参阅SVL_DATASHARE_CHANGE_LOG和SVL_DATASHARE_USAGE_PRODUCER。

您可以通过查询系统视图来监控数据共享。

1. 希望共享数据的创建者集群管理员创建 Amazon Redshift 数据共享。然后，创建者集群管理员添加所需的数据库对象。这些对象可能是数据共享的架构、表和视图，并指定要与之共享对象的使用者列表。

   使用以下系统视图查看用于跟踪生产者和/或使用者集群上数据共享的变化和使用情况的合并视图：

   - SYS_DATASHARE_CHANGE_LOG
   - SYS_DATASHARE_USAGE_CONSUMER
   - SYS_DATASHARE_USAGE_PRODUCER

   使用以下系统视图查看出站数据共享的数据共享对象和数据使用者信息：

   - SVV_DATASHARES
   - SVV_DATASHARE_CONSUMERS

- SVV_DATASHARE_OBJECTS

2. 使用者集群管理员查看其被授予使用的数据共享，并通过使用 SVV_DATASHARES 查看入站数据共享来查看每个数据共享的内容。

   要使用共享的数据，每个使用者集群管理员可从数据共享中创建 Amazon Redshift 数据库。然后，管理员将权限分配给使用者集群中的适当用户和角色。用户和角色可以通过查看以下元数据系统视图将共享对象列为标准元数据查询的一部分，并可立即开始查询数据。

   - SVV_REDSHIFT_COLUMNS
   - SVV_REDSHIFT_DATABASES
   - SVV_REDSHIFT_FUNCTIONS
   - SVV_REDSHIFT_SCHEMAS
   - SVV_REDSHIFT_TABLES

   要查看 Amazon Redshift 本地和共享 schema 以及外部 schema 的对象，请使用以下元数据系统视图对其进行查询。

   - SVV_ALL_COLUMNS
   - SVV_ALL_SCHEMAS
   - SVV_ALL_TABLES

Amazon Redshift 数据共享与 AWS CloudTrail 集成

数据共享与 AWS CloudTrail 集成。CloudTrail 服务提供用户、角色或 AWS 服务在 Amazon Redshift 中所执行的操作记录。CloudTrail 将数据共享的所有 API 调用作为事件捕获。捕获的调用中包括通过 AWS CloudTrail 控制台的调用以及对数据共享操作的代码调用。有关 Amazon Redshift 与 AWS CloudTrail 集成的更多信息，请参阅使用 CloudTrail 进行日志记录。

有关使用 CloudTrail 的更多信息，请参阅 CloudTrail 的工作原理。

管理数据共享任务

您可以使用 SQL 界面或 Amazon Redshift 控制台开始使用数据共享。

主题

- 使用 SQL 接口管理数据共享
- 使用控制台管理数据共享

- 使用 AWS CloudFormation 管理数据共享
- 使用控制台管理写入数据共享（预览版）

使用 SQL 接口管理数据共享

您可以在 AWS 账户内、跨账户或跨 AWS 区域，在不同的 Amazon Redshift 集群间共享数据以进行读取。

主题

- 共享对 AWS 账户内数据的读取访问权限
- 共享对数据的写入访问权限（预览版）
- 跨 AWS 账户共享数据
- 跨 AWS 区域共享数据
- 在 AWS Data Exchange 上共享许可的 Amazon Redshift 数据
- 使用 AWS Lake Formation 托管的数据共享

共享对 AWS 账户内数据的读取访问权限

您可以在一个 AWS 账户 内的不同 Amazon Redshift 集群间共享数据，以进行读取。

要以创建器集群管理员或数据库拥有者的身份共享数据以用于读取目的

1. 在您的集群中创建数据共享。有关更多信息，请参阅 CREATE DATASHARE。

   ```
   CREATE DATASHARE salesshare;
   ```

   集群超级用户和数据库拥有者可以创建数据共享。在创建过程中，每个数据共享都与数据库相关联。只有该数据库中的对象才能在该数据共享中共享。可以在具有相同或不同粒度对象的同一数据库上创建多个数据共享。集群可以创建的数据共享数量没有限制。

   您还可以使用 Amazon Redshift 控制台创建数据共享。有关更多信息，请参阅 创建数据共享。
2. 委派权限以对数据共享进行操作。有关更多信息，请参阅 GRANT 或 REVOKE。

   以下示例授予 dbuser 对于 salesshare 的权限。

   ```
   GRANT ALTER, SHARE ON DATASHARE salesshare TO dbuser;
   ```

集群超级用户和数据共享的拥有者可以向其它用户授予或撤消对数据共享的修改权限。

3. 将对象添加到数据共享或从数据共享中删除对象。要将对象添加到数据共享中，请在添加对象之前添加 schema。当您添加 schema 时，Amazon Redshift 不会在其下添加所有对象。确保明确添加这些内容。有关更多信息，请参阅 ALTER DATASHARE。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

您还可以将视图添加到数据共享。

```
CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
 public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;
```

使用 ALTER DATASHARE 共享给定 schema 中的 schema 以及表、视图和函数。超级用户、数据共享拥有者或对数据共享具有 ALTER 或 ALL 权限的用户可以更改数据共享以向其中添加对象或从中删除对象。用户应具有向数据共享中添加对象或从中删除对象的权限。用户还应该是对象的拥有者，或者对这些对象具有 SELECT、USAGE 或 ALL 权限。

也可以使用 GRANT 向数据共享添加对象。此示例演示如何：

```
GRANT SELECT ON TABLE public.tickit_sales_redshift TO DATASHARE salesshare;
```

此语法在功能上等同于 `ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;`。

使用 INCLUDENEW 子句将在指定 schema 中创建的任何未来表、视图或 SQL 用户定义函数 (UDF) 添加到数据共享中。只有超级用户才可以更改每个数据共享-schema 对的此属性。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

您还可以使用 Amazon Redshift 控制台添加或从数据共享中删除对象。有关更多信息，请参阅将数据共享对象添加到数据共享、从数据共享中删除数据共享对象和编辑在您的账户中创建的数据共享。

4. 将使用者添加到数据共享或从数据共享中删除使用者。以下示例将使用者集群命名空间添加到 salesshare 中。命名空间是账户中的使用者集群的命名空间全局唯一标识符（GUID）。有关更多信息，请参阅 GRANT 或 REVOKE。

```
GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

您只能将权限授予 GRANT 语句中的一个数据共享使用者。

集群超级用户和数据共享对象的拥有者或对数据共享具有 SHARE 权限的用户可以将使用者添加到数据共享或从中删除使用者。为此，他们使用 GRANT USAGE 或 REVOKE USAGE。

要查找当前看到的集群的命名空间，您可以使用 SELECT CURRENT_NAMESPACE 命令。要查找同一个 AWS 账户中不同集群的命名空间，请转到 Amazon Redshift 控制台集群详细信息页面。在该页面上，找到新添加的命名空间字段。

您还可以使用 Amazon Redshift 控制台添加或从数据共享中删除数据使用者。有关更多信息，请参阅将数据使用者添加到数据共享和从数据共享中删除数据使用者。

5. （可选）向数据共享中添加安全限制。以下示例显示允许具有公有 IP 访问权限的使用者集群读取数据共享。有关更多信息，请参阅 ALTER DATASHARE。

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE = TRUE;
```

您可以在创建数据共享后修改有关使用者类型的属性。例如，您可以定义希望使用给定数据共享中的数据的集群不能公开访问。来自不符合数据共享中指定的安全限制的使用者集群的查询将在查询运行时被拒绝。

您还可以使用 Amazon Redshift 控制台编辑数据共享。有关更多信息，请参阅 编辑在您的账户中创建的数据共享。

6. 列出在集群中创建的数据共享，并查看数据共享的内容。

以下示例显示名为 salesshare 的数据共享的信息。有关更多信息，请参阅DESC DATASHARE和SHOW DATASHARES。

```
DESC DATASHARE salesshare;


 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name           |   include_new
```

```
------------------+--------------------------------------+------------
+------------+-------------+---------------------------------+------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_users_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_venue_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_category_redshift|
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_date_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_event_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_listing_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_sales_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | schema      | public                         |  t
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | view        | public.sales_data_summary_view |
```

以下示例显示创建器集群中的出站数据共享。

```
SHOW DATASHARES LIKE 'sales%';
```

该输出值看上去类似于以下内容。

```
share_name | share_owner  | source_database | consumer_database | share_type |
 createdate      | is_publicaccessible  | share_acl | producer_account |
 producer_namespace
-----------+-------------+-----------------+-------------------+------------
+--------------------+---------------------+-----------+-----------------
+--------------------------------------
salesshare |    100       | dev             |                   |  OUTBOUND
 | 2020-12-09 02:27:08 |          True        |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

有关更多信息，请参阅DESC DATASHARE和SHOW DATASHARES。

您还可以使用 SVV_DATASHARES、SVV_DATASHARE_CONSUMERS 和 SVV_DATASHARE_OBJECTS 来查看数据共享、数据共享内的对象以及数据共享使用者。

7. 删除数据共享。有关更多信息，请参阅 DROP DATASHARE。

   您可以随时使用 DROP DATASHARE 删除数据共享对象。集群超级用户和数据共享拥有者可以删除数据共享。

   以下示例将删除名为 salesshare 的数据共享。

   ```
   DROP DATASHARE salesshare;
   ```

   您还可以使用 Amazon Redshift 控制台删除数据共享。有关更多信息，请参阅 删除在您的账户中创建的数据共享。

8. 使用 ALTER DATASHARE 可以在任何时间点从数据共享中删除对象。使用 REVOKE USAGE ON 可撤销某些使用者对数据共享的权限。它可以撤消对数据共享内的对象的 USAGE 权限，并立即停止对所有使用者集群的访问。列出数据共享和元数据查询（如列出数据库和表）不会在撤销访问权限后返回共享对象。

   ```
   ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
   ```

   您还可以使用 Amazon Redshift 控制台编辑数据共享。有关更多信息，请参阅 编辑在您的账户中创建的数据共享。

9. 如果您不想再与使用者共享数据，则撤消从命名空间对数据共享的访问权限。

   ```
   REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE
    '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
   ```

   您还可以使用 Amazon Redshift 控制台编辑数据共享。有关更多信息，请参阅 编辑在您的账户中创建的数据共享。

要以使用者集群管理员的身份共享数据以进行读取

1. 列出可供您使用的数据共享并查看数据共享的内容。有关更多信息，请参阅DESC DATASHARE和SHOW DATASHARES。

   以下示例显示指定创建器命名空间的入站数据共享的信息。当您以使用者集群管理员身份运行 DESC DATASHARE 时，您必须指定 NAMESPACE 选项以查看入站数据共享。

   ```
   DESC DATASHARE salesshare OF NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
   ```

```
 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name           |   include_new
------------------+-------------------------------------+------------
+------------+-------------+---------------------------------+------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_users_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_venue_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_category_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_date_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_event_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_listing_redshift  |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_sales_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | schema      | public                          |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | view        | public.sales_data_summary_view  |
```

只有集群超级用户才可以执行此操作。您还可以使用 SVV_DATASHARES 查看数据共享，使用 SVV_DATASHARE_OBJECTS 查看数据共享内的对象。

以下示例显示使用者集群中的入站数据共享。

```
SHOW DATASHARES LIKE 'sales%';


 share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
------------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+------------------
+--------------------------------------
 salesshare |             |                 |                   | INBOUND
    |            |         t           |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

2. 作为数据库超级用户，您可以创建引用数据共享的本地数据库。有关更多信息，请参阅 CREATE DATABASE。

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

如果您想更精细地控制对本地数据库中对象的访问权限，请在创建数据库时使用 WITH PERMISSIONS 子句。这允许您在步骤 4 中为数据库中的对象授予对象级权限。

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

您可以通过查询 SVV_REDSHIFT_DATABASES 视图查看从数据共享中创建的数据库。您无法连接到从数据共享创建的这些数据库，它们是只读的。但是，您可以连接到使用者集群上的本地数据库并执行跨数据库查询，以查询从数据共享创建的数据库中的数据。您不能基于从现有数据共享创建的数据库对象创建数据共享。但是，您可以将数据复制到使用者集群上的单独表中，执行所需的任何处理，然后共享创建的新对象。

您还可以使用 Amazon Redshift 控制台从数据共享中创建数据库。有关更多信息，请参阅 通过数据共享创建数据库。

3. （可选）创建外部 schema，以引用导入到使用者集群上的使用者数据库中的特定 schema 并为其分配精细权限。有关更多信息，请参阅 CREATE EXTERNAL SCHEMA。

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA
 'public';
```

4. 根据需要，向使用者集群中的用户和角色授权对从数据共享创建的数据库和架构引用的权限。有关更多信息，请参阅GRANT或REVOKE。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

如果创建数据库时不使用 WITH PERMISSIONS，则只能将从数据共享创建的整个数据库的权限分配给用户和角色。在某些情况下，您需要对根据数据共享创建的数据库对象子集进行精细控制。如果是这样，您可以创建一个外部 schema 引用，该引用指向数据共享中的特定 schema（如上一步所述），并在 schema 级别提供精细权限。

您还可以基于共享的对象创建后期绑定视图，并使用这些视图来分配精细权限。您还可以考虑让创建器集群为您创建具有所需精细度的额外的数据共享。

如果您在步骤 2 中使用 WITH PERMISSIONS 创建了数据库，则必须为共享数据库中的对象分配对象级权限。只有 USAGE 权限的用户在获得其他对象级权限之前，无法访问使用 WITH PERMISSIONS 创建的数据库中的任何对象。

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

5. 在数据共享中查询共享对象中的数据。

对使用者集群上的使用者数据库和架构具有权限的用户和角色可以浏览和导航任何共享对象的元数据。他们还可以浏览和导航使用者集群中的本地对象。为此，他们使用 JDBC 或 ODBC 驱动程序或 SVV_ALL 和 SVV_REDSHIFT 视图。

创建器集群在数据库中可能有许多 schema、表和每个 schema 中的视图。使用者端的用户只能看到通过数据共享提供的对象的子集。这些用户无法从创建器集群中看到整个元数据。此方法有助于通过数据共享提供精细的元数据安全控制。

您将继续连接到本地集群数据库。但现在，您也可以使用三部分 database.schema.table 表示法从数据共享创建的数据库和 schema 中读取。您可以跨您可见的任何数据库和所有数据库执行查询。这些数据库可以是集群上的本地数据库，也可以是通过数据共享创建的数据库。使用者集群无法连接到从数据共享创建的数据库。

您可以使用完全资格认证来访问数据。有关更多信息，请参阅 使用跨数据库查询的示例。

```
SELECT * FROM sales_db.public.tickit_sales_redshift ORDER BY 1,2 LIMIT 5;

 salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime
---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------
       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |    728.00 |
   109.20 | 2008-02-18 02:36:48
       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |     76.00 |
    11.40 | 2008-06-06 05:00:16
       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |    350.00 |
    52.50 | 2008-06-06 08:26:17
       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |    175.00 |
    26.25 | 2008-06-09 08:38:52
```

```
      5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |    154.00 |
   23.10 | 2008-08-31 09:17:02
```

您只能在共享对象上使用 SELECT 语句。但是，您可以通过查询来自不同本地数据库中的共享对象的数据，在使用者集群中创建表。

除查询之外，使用者还可以对共享对象创建视图。仅支持后期绑定视图或实体化视图。Amazon Redshift 不支持共享数据的常规视图。使用者创建的视图可跨越多个本地数据库或通过数据共享创建的数据库。有关更多信息，请参阅 CREATE VIEW。

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

共享对数据的写入访问权限（预览版）

您可以跨同一 AWS 账户内的不同 Amazon Redshift 集群或 Amazon Redshift Serverless 工作组共享数据库对象以进行读取和写入，也可以在不同账户和不同区域之间进行共享。本主题中的过程展示了如何设置包含写入权限的数据共享。您可以为不同的表授予 SELECT、INSERT 和 UPDATE 等权限，为架构授予 USAGE 和 CREATE 权限。一旦提交写入事务，所有仓库都可获得实时数据。生产者账户管理员可以确定特定的命名空间或区域是否获得对数据的只读、读写或任何访问权限。

以下各节介绍如何配置数据共享。这些过程假设您在处理预置集群或 Amazon Redshift Serverless 工作组中的数据库。

只读数据共享与读写数据共享

以前，数据共享中的对象在所有情况下都是只读的。写入数据共享中的对象是一项新功能。只有当生产者特别授予数据共享中对象的 INSERT 或 CREATE 等写入权限时，数据共享中的对象才会启用写入功能。此外，对于跨账户共享，生产者必须授权数据共享以进行写入，而使用者必须关联特定的集群和工作组才能写入。详细信息将在本主题的后续章节中介绍。

您可以向数据共享授予的权限（预览版）

在数据共享上下文中，您可以向它们授予不同的对象类型和各种权限。

架构：

- USAGE
- CREATE

表：

- SELECT
- INSERT
- UPDATE
- DELETE
- TRUNCATE
- DROP
- REFERENCES

函数：

- EXECUTE

数据库：

- CREATE

预览版数据共享的要求和限制

- 连接 – 必须直接连接到数据共享数据库或运行 USE 命令才能写入数据共享。但是，我们很快就会启用使用由三部分组成的表示法进行此操作的功能。
- 可用性 – 您必须使用 Serverless 工作组、ra3.4xl 集群或 ra3.16xl 集群才能使用此功能。计划支持 ra3.xlplus 集群。
- 元数据发现 – 如果您是通过 Redshift JDBC、ODBC 或 Python 驱动程序直接连接到数据共享数据库的使用者，您可以通过以下方式查看目录数据：

  - SQL SHOW 命令。
  - 查询 information_schema 表和视图。
  - 查询 SVV 元数据视图。
- 数据 API – 您无法通过数据 API 连接到数据共享数据库。我们将很快提供这方面的支持。
- 权限可见性 – 使用者看不到授予数据共享的权限。我们将很快添加这一功能。
- 加密 – 对于跨账户数据共享，必须对生产者和使用者集群进行加密。
- 隔离级别 – 您数据库的隔离级别必须是快照隔离，以便允许其他 Serverless 工作组和集群对其进行写入。
- 自动操作 – 写入数据共享对象的使用者不会触发自动分析操作。因此，在向表中插入数据后，生产者必须手动运行分析才能更新表统计信息。否则，查询计划可能无法达到最佳效果。
- 多语句查询和事务 – 目前不支持事务块之外的多语句查询。因此，如果您使用的是像 dbeaver 这样的查询编辑器，并且有多个写入查询，则需要将查询封装在显式 BEGIN...END 事务语句中。

支持的 SQL 语句

通过写入共享数据的公开预览版支持以下语句：

- BEGIN | START TRANSACTION
- END | COMMIT | ROLLBACK
- COPY without COMPUPDATE
- { CREATE | DROP } SCHEMA
- { CREATE | DROP | SHOW } TABLE
- CREATE TABLE table_name AS
- DELETE
- { GRANT | REVOKE } privilege_name ON OBJECT_TYPE object_name TO consumer_user
- INSERT
- SELECT
- INSERT INTO SELECT
- TRUNCATE
- UPDATE
- 超级数据类型列

不支持的语句类型 - 不支持以下语句类型：

- 写入生产者时，对使用者仓库进行多语句查询。
- 并发扩展查询从使用者写入生产者。
- 自动复制作业从使用者写入生产者。
- 流式作业从使用者写入生产者。
- 使用者在生产者集群上创建零 ETL 集成表。有关零 ETL 集成的更多信息，请参阅使用零 ETL 集成。
- 用交错排序键写入表。

以生产者账户管理员的身份在具有写入权限的账户内共享数据（预览版）

以前，数据共享中的对象在所有情况下都是只读的。写入数据共享中的对象是一项新功能。只有当生产者特别授予数据共享中对象的 INSERT 或 CREATE 等写入权限时，数据共享中的对象才会启用写入功能。详细信息将在本主题的后续章节中介绍。

如果您要查找只读数据存储的现有文档，请访问 Amazon Redshift 跨集群共享数据。

要开始数据共享，生产者上的管理员会创建一个数据共享并向其中添加对象：

1. 生产者数据库所有者或超级用户创建数据共享。数据共享是数据库对象、权限和使用者的逻辑容器。（使用者是指您的账户和其他账户中的集群或 Amazon Redshift Serverless 命名空间。）每个数据共享都与在其中创建的数据库相关联，并且只能添加该数据库中的对象。下面的命令将创建一个数据共享：

   ```
   CREATE DATASHARE my_datashare [PUBLICACCESSIBLE = TRUE];
   ```

   设置 PUBLICACCESSIBLE = TRUE 允许使用者从可公开访问的集群和预置工作组中查询您的数据共享。如果您不想允许查询，请将其省略或明确设置为 false。

   数据共享所有者必须对要添加到数据共享的架构授予 USAGE 权限。GRANT 命令是新的。它用于授予对架构的各种操作，包括 CREATE 和 USAGE。架构中包含共享对象：

   ```
   CREATE SCHEMA myshared_schema1;
   CREATE SCHEMA myshared_schema2;

   GRANT USAGE ON SCHEMA myshared_schema1 TO DATASHARE my_datashare;
   GRANT CREATE, USAGE ON SCHEMA myshared_schema2 TO DATASHARE my_datashare;
   ```

或者，管理员可以继续运行 ALTER 命令将架构添加到数据共享。以这种方式添加架构时，仅授予 USAGE 权限。

```
ALTER DATASHARE my_datashare ADD SCHEMA myshared_schema1;
```

2. 管理员添加架构后，可以授予对架构中对象的数据共享权限。这些权限可以是读取和写入权限。GRANT ALL 示例显示了如何授予所有权限。

```
GRANT SELECT, INSERT ON TABLE myshared_schema1.table1, myshared_schema1.table2,
 myshared_schema2.table1
TO DATASHARE my_datashare;

GRANT ALL ON TABLE myshared_schema1.table4 TO DATASHARE my_datashare;
```

您可以继续运行 ALTER DATASHARE 等命令来添加表。这样做时，只会对添加的对象授予 SELECT 权限。

```
ALTER DATASHARE my_datashare ADD TABLE myshared_schema1.table1,
 myshared_schema1.table2, myshared_schema2.table1;
```

3. 管理员将数据共享的使用权限授予账户中的特定命名空间。您可以在集群详细信息页面、Amazon Redshift Serverless 命名空间详细信息页面中或通过运行命令 `SELECT current_namespace;` 来找到作为 ARN 一部分的命名空间 ID。有关更多信息，请参阅 CURRENT_NAMESPACE。

```
GRANT USAGE ON DATASHARE my_datashare TO NAMESPACE '86b5169f-012a-234b-9fbb-
e2e24359e9a8';
```

跨账户共享数据写入权限（预览版）

此文档为预发行文档，介绍了通过 Amazon Redshift 数据共享功能进行多数据仓库写入，该功能在 PREVIEW_2023 版本的公开预览版中提供。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

如果您尚未在 PREVIEW_2023 库中创建数据共享，则转到共享对数据的写入访问权限（预览版）以开始使用。

以使用者数据安全管理员身份关联共享数据（预览版）

以下是通过 PREVIEW_2023 版本中公开预览版 Amazon Redshift 数据共享功能写入多数据仓库的预发行文档。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

如果您尚未在 PREVIEW_2023 库中创建数据共享，则转到共享对数据的写入访问权限（预览版）以开始使用。

先决条件：如果数据共享是与其他账户共享的，则本节中的步骤将在生产者管理员授予对共享数据库对象的特定操作，以及生产者安全管理员授权访问后执行。

使用者安全管理员确定以下内容：

- 账户中的所有命名空间、账户中特定区域的命名空间或特定命名空间是否可以访问数据共享。
- 如果命名空间可以访问数据共享，这些命名空间是否有写入权限。

使用者安全管理员可以通过控制台、CLI 或 API 关联数据共享。如果通过 CLI，管理员使用以下命令：

```
associate-data-share-consumer
--data-share-arn <value>
--consumer-identifier <value>
[--allow-writes | --no-allow-writes]
```

有关该命令的更多信息，请参阅 associate-data-share-consumer。

在将数据共享与命名空间关联时，使用者安全管理员必须明确将 `allow-writes` 设置为 true，以允许使用 INSERT 和 UPDATE 命令。如果不这样做，则用户只能执行读取操作，例如 SELECT、USAGE 或 EXECUTE 权限。

您可以通过使用不同的值再次调用 `associate-data-share-consumer` 来更改数据共享的命名空间关联。旧的关联会被新的关联覆盖，因此，如果您最初关联和设置 `allow-writes`，但关联并指定 `no-allow-writes`，或者干脆不指定值，则使用者的写入权限将被撤销。

以生产者安全管理员身份授权写入数据共享（预览版）

以下是通过 PREVIEW_2023 版本中公开预览版 Amazon Redshift 数据共享功能写入多数据仓库的预发行文档。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

如果您尚未在 PREVIEW_2023 库中创建数据共享，则转到共享对数据的写入访问权限（预览版）以开始使用。

> **Note**
>
> 这仅适用于账户之间共享数据共享的情况。

生产者安全管理员确定以下内容：

- 其他账户是否可以访问数据共享。
- 如果账户有权访问数据共享，则该账户是否具有写入权限。

授权数据共享需要以下 IAM 权限：

redshift:AuthorizeDataShare

您可以通过 CLI 调用或 API 授权使用和写入权限：

```
authorize-data-share
--data-share-arn <value>
--consumer-identifier <value>
[--allow-writes | --no-allow-writes]
```

有关该命令的更多信息，请参阅 authorize-data-share。

使用者标识符可以是：

- 十二位数的 AWS 账户 ID。
- 命名空间标识符 ARN。

请注意，在授权步骤中不会授予写入权限。授权写入数据共享只是允许账户拥有数据共享管理员授予的写入权限。如果管理员不允许写入，则特定使用者只能拥有 SELECT、USAGE 和 EXECUTE 权限。

您可以通过再次调用 `authorize-data-share` 来更改数据共享使用者的授权，但要使用不同的值。新授权将覆盖旧授权。因此，如果您最初授权并允许写入，但重新授权并指定 `no-allow-writes` 或干脆不指定值，则使用者的写入权限将被撤销。

可进行数据共享的区域（预览版）

以下是通过 PREVIEW_2023 版本中公开预览版 Amazon Redshift 数据共享功能写入多数据仓库的预发行文档。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

如果您尚未在 PREVIEW_2023 库中创建数据共享，则转到共享对数据的写入访问权限（预览版）以开始使用。

以下区域可进行数据共享（预览版）：

- 美国东部（弗吉尼亚州北部）(us-east-1)
- 美国东部（俄亥俄州）(us-east-2)
- 美国西部（俄勒冈州）(us-west-2)
- 亚太地区（东京）(ap-northeast-1)
- 欧洲地区（爱尔兰）(eu-west-1)
- 欧洲地区（斯德哥尔摩）(eu-north-1)

跨 AWS 账户共享数据

您可以跨 AWS 账户共享数据以进行读取。跨 AWS 账户共享数据的工作方式类似于在账户内共享数据。区别在于，需要双向握手来跨 AWS 账户共享数据。创建器账户管理员可以授权使用者账户访问数据共享，也可以选择不授权任何访问权限。要使用经授权的数据共享，使用者账户管理员可以关联该数据共享。管理员可以将数据共享与整个 AWS 账户或使用者账户中的特定集群关联，或拒绝数据共享。有关在账户中共享数据的更多信息，请参阅共享对 AWS 账户内数据的读取访问权限。

数据共享可以拥有数据使用者，他们是同一账户或不同 AWS 账户中的集群命名空间。您无需创建单独的数据共享以进行账户内共享和跨账户共享。

对于跨账户数据共享，必须对创建器和使用者集群进行加密。

与 AWS 账户共享数据时，生产者集群管理员将作为一个实体与 AWS 账户进行共享。使用者集群管理员可以决定使用者账户中的哪些集群命名空间可以访问数据共享。

主题

- 创建者集群管理员操作
- 使用者账户管理员操作
- 使用者集群管理员操作

创建者集群管理员操作

如果您是创建器集群管理员或数据库拥有者 – 按照以下步骤操作：

1. 在集群中创建数据共享，并将数据共享对象添加到数据共享。有关如何创建数据共享和将数据集对象添加到数据共享的更多详细步骤，请参阅共享对 AWS 账户内数据的读取访问权限。有关 CREATE DATASHARE 和 ALTER DATASHARE 的信息，请参阅CREATE DATASHARE和ALTER DATASHARE。

   以下示例将不同的数据共享对象添加到数据共享 `salesshare` 中：

   ```
   -- Add schema to datashare
   ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;

   -- Add table under schema to datashare
   ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;

   -- Add view to datashare
   ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;

   -- Add all existing tables and views under schema to datashare (does not include
    future table)
   ALTER DATASHARE salesshare ADD ALL TABLES in schema public;
   ```

   您还可以使用 Amazon Redshift 控制台创建或编辑数据共享。有关更多信息，请参阅创建数据共享和编辑在您的账户中创建的数据共享。
2. 委派权限以对数据共享进行操作。有关更多信息，请参阅 GRANT 或 REVOKE。

   以下示例授予 `dbuser` 对于 `salesshare` 的权限。

```
GRANT ALTER, SHARE ON DATASHARE salesshare TO dbuser;
```

集群超级用户和数据共享的拥有者可以向其它用户授予或撤消对数据共享的修改权限。

3. 将使用者添加到数据共享或从数据共享中删除使用者。以下示例将 AWS 账户 ID 添加到 salesshare。有关更多信息，请参阅 GRANT 或 REVOKE。

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '123456789012';
```

您只能将权限授予 GRANT 语句中的一个数据使用者。

集群超级用户和数据共享对象的拥有者或对数据共享具有 SHARE 权限的用户可以将使用者添加到数据共享或从中删除使用者。为此，他们使用 GRANT USAGE 或 REVOKE USAGE。

您还可以使用 Amazon Redshift 控制台添加或从数据共享中删除数据使用者。有关更多信息，请参阅将数据使用者添加到数据共享和从数据共享中删除数据使用者。

4. （可选）如果您不想再与使用者共享数据，请撤消 AWS 账户对数据共享的访问权限。

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '123456789012';
```

如果您是创建器账户管理员 – 按照以下步骤操作：

将使用权授予 AWS 账户后，数据共享状态为 pending_authorization。创建器账户管理员应使用 Amazon Redshift 控制台授权数据共享，并选择数据使用者。

登录 https://console.aws.amazon.com/redshiftv2/。然后选择要授权哪些数据使用者访问数据共享或从中删除授权。授权的数据使用者会收到通知，以便对数据共享采取操作。如果您要以数据使用者身份添加集群命名空间，则不必执行授权。数据使用者获得授权后，可以访问数据共享对象并创建使用者数据库以查询数据。有关更多信息，请参阅 授权或删除数据共享中的授权。

使用者账户管理员操作

如果您是使用者账户管理员 – 按照以下步骤操作：

要将从其他账户共享的一个或多个数据共享与您的整个 AWS 账户或您账户中的特定集群命名空间关联，请使用 Amazon Redshift 控制台。

登录 https://console.aws.amazon.com/redshiftv2/。然后，将从其他账户共享的一个或多个数据共享与您的整个 AWS 账户或您账户中的特定集群命名空间关联。有关更多信息，请参阅 关联数据共享。

关联 AWS 账户 或特定集群命名空间后，数据共享将可供使用。您还可以随时更改数据共享关联。将关联从单个集群命名空间更改到 AWS 账户 时，Amazon Redshift 会使用 AWS 账户 信息覆盖集群命名空间。将关联从 AWS 账户 更改到特定集群命名空间时，Amazon Redshift 会使用集群命名空间信息覆盖 AWS 账户 信息。账户中的所有集群命名空间都可以访问数据。

使用者集群管理员操作

如果您是使用者集群管理员 – 按照以下步骤操作：

1. 列出提供给您的数据共享并查看数据共享的内容。仅当创建器集群管理员已授权数据共享且使用者集群管理员已接受并关联数据共享时，数据共享的内容才可用。有关更多信息，请参阅DESC DATASHARE和SHOW DATASHARES。

   以下示例显示指定创建器命名空间的入站数据共享的信息。当您以使用者集群管理员身份运行 DESC DATAHSARE 时，您必须指定 NAMESPACE 以及账户 ID 以查看入站数据共享。对于出站数据共享，请指定数据共享名称。

```
SHOW DATASHARES LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
-----------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+-----------------
+--------------------------------------
salesshare |             |                 |                   | INBOUND    |
            |        t            |           | 123456789012    | 'dd8772e1-
d792-4fa4-996b-1870577efc0d'
```

```
DESC DATASHARE salesshare OF ACCOUNT '123456789012' NAMESPACE 'dd8772e1-
d792-4fa4-996b-1870577efc0d';


 producer_account |          producer_namespace          | share_type | share_name |
 object_type |           object_name
------------------+--------------------------------------+------------+------------
+-------------+---------------------------------
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_users_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_venue_redshift
```

```
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_category_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_date_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_event_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_listing_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_sales_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 schema      | public
(8 rows)
```

只有集群超级用户才可以执行此操作。您还可以使用 SVV_DATASHARES 查看数据共享，使用 SVV_DATASHARE_OBJECTS 查看数据共享内的对象。

以下示例显示使用者集群中的入站数据共享。

```
SELECT * FROM SVV_DATASHARES WHERE share_name LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
-----------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+------------------
+--------------------------------------
salesshare |             |                 |                   | INBOUND    |
            |        t            |           | 123456789012      | 'dd8772e1-
d792-4fa4-996b-1870577efc0d'
```

```
SELECT * FROM SVV_DATASHARE_OBJECTS WHERE share_name LIKE 'sales%';
 share_type | share_name | object_type |           object_name           |
 producer_account |          producer_namespace
------------+------------+-------------+---------------------------------
+------------------+--------------------------------------
 INBOUND    | salesshare | table       | public.tickit_users_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_venue_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_category_redshift |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
```

```
 INBOUND    | salesshare | table       | public.tickit_date_redshift     |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_event_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_listing_redshift  |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_sales_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | schema      | public                          |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
(8 rows)
```

2. 创建引用数据共享的本地数据库。从数据共享中创建数据库时，请指定 NAMESPACE 和账户 ID。有关更多信息，请参阅 CREATE DATABASE。

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF ACCOUNT '123456789012'
 NAMESPACE 'dd8772e1-d792-4fa4-996b-1870577efc0d';
```

   如果您想更精细地控制对本地数据库中对象的访问权限，请在创建数据库时使用 WITH PERMISSIONS 子句。这允许您在步骤 4 中为数据库中的对象授予对象级权限。

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF ACCOUNT
 '123456789012' NAMESPACE 'dd8772e1-d792-4fa4-996b-1870577efc0d';
```

   您可以通过查询 SVV_REDSHIFT_DATABASES 视图查看从数据共享中创建的数据库。您无法连接到从数据共享创建的这些数据库，它们是只读的。但是，您可以连接到使用者集群上的本地数据库，并对从数据共享创建的数据库中的数据执行跨数据库查询。您不能基于从现有数据共享创建的数据库对象创建数据共享。但是，您可以将数据复制到使用者集群上的单独表中，执行所需的任何处理，然后共享创建的新对象。

3. （可选）创建外部 schema，以引用导入到使用者集群上的使用者数据库中的特定 schema 并为其分配精细权限。有关更多信息，请参阅 CREATE EXTERNAL SCHEMA。

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA
 'public';
```

4. 根据需要，向使用者集群中的用户或角色授权对从数据共享创建的数据库和架构引用的权限。有关更多信息，请参阅GRANT或REVOKE。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

如果创建数据库时不使用 WITH PERMISSIONS，则只能将从数据共享创建的整个数据库的权限分配给用户或角色。在某些情况下，您需要对根据数据共享创建的数据库对象子集进行精细控制。如果是这样，您可以创建一个外部架构引用，指向数据共享中的特定架构（如上一步所述）。然后，您可以在架构级别提供精细权限。您还可以基于共享的对象创建后期绑定视图，并使用这些视图来分配精细权限。您还可以考虑让创建器集群为您创建具有所需精细度的额外的数据共享。您可以根据需要为从数据共享中创建的数据库创建尽可能多的架构引用。

如果您在步骤 2 中使用 WITH PERMISSIONS 创建了数据库，则必须为共享数据库中的对象分配对象级权限。只有 USAGE 权限的用户在获得其他对象级权限之前，无法访问使用 WITH PERMISSIONS 创建的数据库中的任何对象。

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

5. 在数据共享中查询共享对象中的数据。

   对使用者集群上的使用者数据库和架构具有权限的用户和角色可以浏览和导航任何共享对象的元数据。他们还可以浏览和导航使用者集群中的本地对象。为此，请使用 JDBC 或 ODBC 驱动程序或 SVV_ALL 和 SVV_REDSHIFT 视图。

   创建器集群在数据库中可能有许多 schema、表和每个 schema 中的视图。使用者端的用户只能看到通过数据共享提供的对象的子集。这些用户无法从创建者集群中看到整个元数据。此方法有助于通过数据共享提供精细的元数据安全控制。

   您将继续连接到本地集群数据库。但现在，您也可以使用三部分 database.schema.table 表示法从数据共享创建的数据库和 schema 中读取。您可以跨您可见的任何数据库和所有数据库执行查询。这些数据库可以是集群上的本地数据库，也可以是通过数据共享创建的数据库。使用者集群无法连接到从数据共享创建的数据库。

   您可以使用完全资格认证来访问数据。有关更多信息，请参阅 使用跨数据库查询的示例。

   ```
   SELECT * FROM sales_db.public.tickit_sales_redshift;
   ```

   您只能在共享对象上使用 SELECT 语句。但是，您可以通过查询来自不同本地数据库中的共享对象的数据，在使用者集群中创建表。

除执行查询之外，使用者还可以对共享对象创建视图。仅支持后期绑定视图和实体化视图。Amazon Redshift 不支持共享数据的常规视图。使用者创建的视图可跨越多个本地数据库或通过数据共享创建的数据库。有关更多信息，请参阅 CREATE VIEW。

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

跨 AWS 区域共享数据

您可以在 AWS 区域 内的不同 Amazon Redshift 集群间共享数据，以进行读取。通过跨区域数据共享，您可以跨 AWS 区域 共享数据，无需手动复制数据。您无需将数据卸载到 Amazon S3 中，并将数据复制到新的 Amazon Redshift 集群中或执行跨区域快照复制。

通过跨区域数据共享，您可以在同一个 AWS 账户 或不同 AWS 账户 中共享数据，即使集群位于不同区域。与位于相同 AWS 账户但不同 AWS 区域的 Amazon Redshift 集群共享数据时，请遵循与在 AWS 账户中共享数据相同的工作流。有关更多信息，请参阅 共享对 AWS 账户内数据的读取访问权限。

如果共享数据的集群位于不同的 AWS 账户 和 AWS 区域，您可以遵循与跨 AWS 账户 共享数据相同的工作流程，并在使用者集群中包括区域级别的关联。跨区域数据共享支持与整个 AWS 账户、整个 AWS 区域，或者 AWS 区域 中的特定集群命名空间的数据共享关联。有关在 AWS 账户 账户中共享数据的更多信息，请参阅跨 AWS 账户共享数据。

当使用来自其他区域的数据时，使用者需要支付从生产者区域到使用者区域的跨区域数据传输费。

要使用数据共享，使用者账户管理员可以通过以下三种方式之一关联数据共享。

- 与跨其所有 AWS 区域 的整个 AWS 账户 关联
- 与 AWS 账户 中的特定 AWS 区域 关联
- 与 AWS 区域 中的特定集群命名空间关联

若管理员选择整个 AWS 账户，该账户中跨不同 AWS 区域 的所有现有和未来的集群命名空间均有权访问数据共享。使用者账户管理员还可以选择区域内的特定 AWS 区域 或集群命名空间，以授予其对数据共享的访问权限。

如果您是创建者集群管理员或数据库所有者，请创建数据共享、将数据库对象和数据使用者添加到数据共享中，并向数据使用者授予权限。有关更多信息，请参阅 创建者集群管理员操作。

如果您是创建者账户管理员，则应使用 AWS Command Line Interface（AWS CLI）或 Amazon Redshift 控制台授权数据共享，并选择数据使用者。

如果您是使用者账户管理员 – 按照以下步骤操作：

要将从其他账户共享的一个或多个数据共享与您的整个 AWS 账户或 AWS 区域中的特定 AWS 区域或集群命名空间关联，请使用 Amazon Redshift 控制台。

借助跨区域数据共享，您可以使用 AWS Command Line Interface（AWS CLI）或 Amazon Redshift 控制台在特定的 AWS 区域中添加集群。

要指定一个或多个 AWS 区域，您可以使用 `associate-data-share-consumer` CLI 命令以及可选的 `consumer-region` 选项。

以下示例使用 CLI 将 `Salesshare` 与整个 AWS 账户关联，并选择了 `associate-entire-account` 选项。您一次只能关联一个区域。

```
aws redshift associate-data-share-consumer
--region {PRODUCER_REGION}
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--associate-entire-account
```

以下示例将 `Salesshare` 关联到美国东部（俄亥俄州）区域 (`us-east-2`)。

```
aws redshift associate-data-share-consumer
--region {PRODUCER_REGION}
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:0123456789012:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--consumer-region 'us-east-2'
```

以下示例将 `Salesshare` 与另一个 AWS 账户在亚太地区（悉尼）区域（`ap-southeast-2`）的一个特定使用者集群命名空间关联。

```
aws redshift associate-data-share-consumer
```

```
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--consumer-arn 'arn:aws:redshift:ap-southeast-2:{CONSUMER_ACCOUNT}:namespace:
{ConsumerImmutableClusterId}'
```

您可以使用 Amazon Redshift 控制台将数据共享与整个 AWS 账户 或 AWS 区域 中的特定 AWS 区域或集群命名空间关联。要执行此操作，请登录 https://console.aws.amazon.com/redshiftv2/。然后将从其它账户共享的一个或多个数据共享与您的整个 AWS 账户、整个 AWS 区域，或 AWS 区域 中的特定集群命名空间关联。有关更多信息，请参阅 关联数据共享。

关联 AWS 账户 或特定集群命名空间后，数据共享将可供使用。您还可以随时更改数据共享关联。将关联从单个集群命名空间更改到 AWS 账户 时，Amazon Redshift 会使用 AWS 账户 信息覆盖集群命名空间。将关联从 AWS 账户 更改到特定集群命名空间时，Amazon Redshift 会使用集群命名空间信息覆盖 AWS 账户 信息。将关联从整个 AWS 账户 更改到特定 AWS 区域和集群命名空间时，Amazon Redshift 会使用特定区域和集群命名空间信息覆盖 AWS 账户 信息。

如果您是使用者集群管理员，您可以创建引用数据共享的本地数据库，并根据需要向使用者集群中的用户或角色授予对从数据共享创建的数据库的访问权限。您还可以对共享对象创建视图，并创建外部架构，以引用导入到使用者集群上的使用者数据库中的特定架构并为其分配精细权限。有关更多信息，请参阅 使用者集群管理员操作。

管理跨区域数据共享的成本控制

当使用来自其他区域的数据时，使用者需要支付从生产者区域到使用者区域的跨区域数据传输费。不同区域的数据传输价格不同。收费基于每次成功运行查询时扫描的数据字节数。有关 Amazon Redshift 定价的更多信息，请参阅 Amazon Redshift 定价。

您需要按字节数付费，不足一兆字节按一兆字节付费，每个查询最低按 10MB 付费。您可以设置有关查询使用量的成本控制，以及查看集群上每个查询传输的数据量。

要监控和控制跨区域数据共享的使用情况和相关使用成本，您可以创建每日、每周、每月使用限制，并定义 Amazon Redshift 在达到这些限制时自动执行的操作，以帮助保持预算的可预测性。要详细了解 Amazon Redshift 中的使用限制，请参阅管理 Amazon Redshift 中的使用限制。

根据您设置的使用限制，Amazon Redshift 执行的操作可能是在某个系统表记录一个事件、发送 CloudWatch 告警并通过 Amazon SNS 通知管理员，或者停止跨区域数据共享的继续使用。有关相关操作的更多信息，请参阅管理 Amazon Redshift 中的使用限制。

要在 Amazon Redshift 控制台上定义使用限制，请在集群的操作下选择配置使用限制。您可以通过自动生成的 CloudWatch 指标监控使用情况趋势，并利用自动生成的 CloudWatch 指标，在使用量超过您定义的限制时从集群性能或监控选项卡获得提示。您还可以使用 AWS CLI 或者 Amazon Redshift API

操作以编程方式创建、修改和删除使用限制。有关更多信息，请参阅管理 Amazon Redshift 中的使用限制。

在 AWS Data Exchange 上共享许可的 Amazon Redshift 数据

创建 AWS Data Exchange 数据共享并将其添加到 AWS Data Exchange 产品，提供商可以在 Amazon Redshift 中授权数据，当用户拥有活动 AWS Data Exchange 订阅时，可以在 Amazon Redshift 中发现、订阅和查询最新数据。

将 AWS Data Exchange 数据共享添加到 AWS Data Exchange 产品后，订阅开始后，使用者即可自动访问产品的数据共享，并且只要订阅处于活跃状态，就可保持访问。

以创建者的身份使用 AWS Data Exchange 数据共享

如果您是创建者集群管理员，请按照以下步骤管理 Amazon Redshift 控制台上的 AWS Data Exchange 数据共享：

1. 在集群中创建数据共享，以共享 AWS Data Exchange 上的数据，并授予 AWS Data Exchange 数据共享访问权限。

   集群超级用户和数据库拥有者可以创建数据共享。在创建过程中，每个数据共享都与数据库相关联。只有该数据库中的对象才能在该数据共享中共享。可以在具有相同或不同粒度对象的同一数据库上创建多个数据共享。集群可以创建的数据共享数量没有限制。

   您还可以使用 Amazon Redshift 控制台创建数据共享。有关更多信息，请参阅 创建数据共享。

   运行 CREATE DATASHARE 语句时，使用 MANAGEDBY ADX 选项隐式授予 AWS Data Exchange 对数据共享的访问权限。这表明 AWS Data Exchange 管理此数据共享。创建新的数据共享时，您只能使用 MANAGEDBY ADX 选项。不能使用 ALTER DATASHARE 语句修改现有的数据共享，以添加 MANAGEDBY ADX 选项。使用 MANAGEDBY ADX 选项创建数据共享后，只有 AWS Data Exchange 可以访问和管理数据共享。

   ```
   CREATE DATASHARE salesshare
   [[SET] MANAGEDBY [=] {ADX} ];
   ```

2. 将对象添加到数据共享中。创建者管理员继续管理 AWS Data Exchange 数据共享中可用的数据共享对象。

   要将对象添加到数据共享中，请在添加对象之前添加 schema。当您添加 schema 时，Amazon Redshift 不会在其下添加所有对象。您必须显式添加它们。有关更多信息，请参阅 ALTER DATASHARE。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

您还可以将视图添加到数据共享。

```
CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
 public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;
```

使用 ALTER DATASHARE 共享给定 schema 中的 schema 以及表、视图和函数。超级用户、数据共享拥有者或对数据共享具有 ALTER 或 ALL 权限的用户可以更改数据共享以向其中添加对象或从中删除对象。用户应具有向数据共享中添加对象或从中删除对象的权限。用户还应该是对象的拥有者，或者对这些对象具有 SELECT、USAGE 或 ALL 权限。

使用 INCLUDENEW 子句将在指定 schema 中创建的任何未来表、视图或 SQL 用户定义函数 (UDF) 添加到数据共享中。只有超级用户才可以更改每个数据共享-schema 对的此属性。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

您还可以使用 Amazon Redshift 控制台添加或从数据共享中删除对象。有关更多信息，请参阅将数据共享对象添加到数据共享、从数据共享中删除数据共享对象和编辑 AWS Data Exchange 数据共享。

3. 要授权对 AWS Data Exchange 数据共享的访问权限，请执行以下操作之一：

   - 通过使用 `aws redshift authorize-data-share` API 中的 `ADX` 关键词明确授权对 AWS Data Exchange 数据共享的访问权限。这样可允许 AWS Data Exchange 识别服务账户中的数据共享，并管理与数据共享关联的使用者。

     ```
     aws redshift authorize-data-share
     --data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
     {PRODUCER_CLUSTER_NAMESPACE}/salesshare
     --consumer-identifier ADX
     ```

您可以将条件密钥 ConsumerIdentifier 用于 AuthorizeDataShare 和 DeauthorizeDataShare，以明确允许或拒绝 AWS Data Exchange 调用 IAM 策略中的两个 API。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Deny",
            "Action": [
                "redshift:AuthorizeDataShare",
                "redshift:DeauthorizeDataShare"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "redshift:ConsumerIdentifier": "ADX"
                }
            }
        }
    ]
}
```

- 使用 Amazon Redshift 控制台授权或删除授权 AWS Data Exchange 数据共享。有关更多信息，请参阅 授权或删除数据共享中的授权。
- 或者，您可以在将数据共享导入 AWS Data Exchange 数据集时隐式授予访问 AWS Data Exchange 数据共享的权限。

要删除 AWS Data Exchange 数据共享访问授权，请使用 aws redshift deauthorize-data-share API 操作中的 ADX 关键词。这样即可允许 AWS Data Exchange 识别服务账户中的数据共享，并从数据共享中删除关联。

```
aws redshift deauthorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier ADX
```

4. 列出在集群中创建的数据共享，并查看数据共享的内容。

以下示例显示名为 SalesShare 的数据共享的信息。有关更多信息，请参阅DESC DATASHARE和SHOW DATASHARES。

```
DESC DATASHARE salesshare;

 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name          |   include_new
-------------------+--------------------------------------+------------
+------------+-------------+---------------------------------+-------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_users_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_venue_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_category_redshift|
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_date_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_event_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_listing_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_sales_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | schema      | public                         |  t
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | view        | public.sales_data_summary_view |
```

以下示例显示创建器集群中的出站数据共享。

```
SHOW DATASHARES LIKE 'sales%';
```

该输出值看上去类似于以下内容。

```
share_name | share_owner  | source_database | consumer_database | share_type |
 createdate      | is_publicaccessible  | share_acl | producer_account |
 producer_namespace
-----------+--------------+-----------------+-------------------+------------
+---------------------+----------------------+-----------+-----------------
+--------------------------------------
```

```
salesshare |    100       | dev             |                    |  OUTBOUND
 | 2020-12-09 02:27:08 |          True        |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

有关更多信息，请参阅DESC DATASHARE和SHOW DATASHARES。

您还可以使用 SVV_DATASHARES、SVV_DATASHARE_CONSUMERS 和 SVV_DATASHARE_OBJECTS 来查看数据共享、数据共享内的对象以及数据共享使用者。

5. 删除数据共享。我们建议您不要使用 DROP DATASHARE 语句删除共享给其他 AWS 账户 的 AWS Data Exchange 数据共享。这些账户将失去对数据共享的访问权限。此操作不可逆。这可能会违反 AWS Data Exchange 中的数据产品优惠条款。如果您要删除 AWS Data Exchange 数据共享，请参阅DROP DATASHARE 使用说明。

   以下示例将删除名为 SalesShare 的数据共享。

   ```
   DROP DATASHARE salesshare;
   ERROR:  Drop of ADX-managed datashare salesshare requires session variable
    datashare_break_glass_session_var to be set to value '620c871f890c49'
   ```

   要允许删除 AWS Data Exchange 数据共享，请设置 datashare_break_glass_session_var 变量，然后再次运行 DROP DATASHARE 语句。如果您要删除 AWS Data Exchange 数据共享，请参阅DROP DATASHARE 使用说明。

   您还可以使用 Amazon Redshift 控制台删除数据共享。有关更多信息，请参阅 删除在您的账户中创建的 AWS Data Exchange 数据共享。

6. 使用 ALTER DATASHARE 可以在任何时间点从数据共享中删除对象。使用 REVOKE USAGE ON 可撤销某些使用者对数据共享的权限。它可以撤消对数据共享内的对象的 USAGE 权限，并立即停止对所有使用者集群的访问。列出数据共享和元数据查询（如列出数据库和表）不会在撤销访问权限后返回共享对象。

   ```
   ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
   ```

   您还可以使用 Amazon Redshift 控制台编辑数据共享。有关更多信息，请参阅 编辑 AWS Data Exchange 数据共享。

7. 授予或撤消 AWS Data Exchange 数据共享的 GRANT USAGE。您不能授予或撤消使用 AWS Data Exchange 数据共享的 GRANT USAGE。以下示例显示了在向 AWS 账户授予对数据共享（由 AWS Data Exchange 管理）的 GRANT USAGE 权限时出现的错误。

```
CREATE DATASHARE salesshare MANAGEDBY ADX;
```

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '012345678910';
ERROR:  Permission denied to add/remove consumer to/from datashare salesshare.
 Datashare consumers are managed by ADX.
```

有关更多信息，请参阅 GRANT 或 REVOKE。

如果您是创建者集群管理员，请按照以下步骤创建数据共享产品，并在 AWS Data Exchange 控制台上发布：

- AWS Data Exchange 数据共享创建后，创建者应创建新的数据集、导入资产、创建修订版本，并创建和发布新产品。

  使用 Amazon Redshift 控制台创建数据集。有关更多信息，请参阅 在 AWS Data Exchange 上创建数据集。

  有关更多信息，请参阅在 AWS Data Exchange 上提供数据产品。

以使用者的身份使用 AWS Data Exchange 数据共享

如果您是使用者，请按照以下步骤发现包含 AWS Data Exchange 数据共享的数据产品，并查询 Amazon Redshift 数据：

1. 在 AWS Data Exchange 控制台探索和订阅包含 AWS Data Exchange 数据共享的数据产品。

   订阅开始后，您可以访问作为资产导入到数据集（包含 AWS Data Exchange 数据共享）的已许可 Amazon Redshift 数据。

   有关如何开始使用包含 AWS Data Exchange 数据共享的数据产品的更多信息，请参阅在 AWS Data Exchange 上订阅数据产品。
2. 如果需要，请在 Amazon Redshift 控制台上创建一个 Amazon Redshift 集群。

   有关如何创建集群的信息，请参阅创建集群。
3. 列出可供您使用的数据共享并查看数据共享的内容。有关更多信息，请参阅DESC DATASHARE和SHOW DATASHARES。

以下示例显示指定创建器命名空间的入站数据共享的信息。当您以使用者集群管理员身份运行 DESC DATASHARE 时，您必须指定 ACCOUNT 和 NAMESPACE 选项以查看入站数据共享。

```
DESC DATASHARE salesshare of ACCOUNT '123456789012' NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';

 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name           |   include_new
-------------------+--------------------------------------+------------
+------------+-------------+----------------------------------+------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_users_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_venue_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_category_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_date_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_event_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_listing_redshift  |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_sales_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | schema      | public                          |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | view        | public.sales_data_summary_view  |
```

只有集群超级用户才可以执行此操作。您还可以使用 SVV_DATASHARES 查看数据共享，使用 SVV_DATASHARE_OBJECTS 查看数据共享内的对象。

以下示例显示使用者集群中的入站数据共享。

```
SHOW DATASHARES LIKE 'sales%';


 share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
```

```
------------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+-----------------
+--------------------------------------
 salesshare |             |                 |                   | INBOUND
    |            |         t           |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

4. 创建引用数据共享的本地数据库。您必须指定 ACCOUNT 和 NAMESPACE 选项才能创建 AWS Data Exchange 数据共享D 本地数据库。有关更多信息，请参阅 CREATE DATABASE。

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF ACCOUNT '123456789012'
 NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

如果您想更精细地控制对本地数据库中对象的访问权限，请在创建数据库时使用 WITH PERMISSIONS 子句。这允许您在步骤 6 中为数据库中的对象授予对象级权限。

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF ACCOUNT
 '123456789012' NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

您可以通过查询 SVV_REDSHIFT_DATABASES 视图查看从数据共享中创建的数据库。您无法连接到从数据共享创建的这些数据库，它们是只读的。但是，您可以连接到使用者集群上的本地数据库，并对从数据共享创建的数据库中的数据执行跨数据库查询。您不能基于从现有数据共享创建的数据库对象创建数据共享。但是，您可以将数据复制到使用者集群上的单独表中，执行所需的任何处理，然后共享创建的新对象。

您还可以使用 Amazon Redshift 控制台从数据共享中创建数据库。有关更多信息，请参阅 通过数据共享创建数据库。

5. （可选）创建外部 schema，以引用导入到使用者集群上的使用者数据库中的特定 schema 并为其分配精细权限。有关更多信息，请参阅 CREATE EXTERNAL SCHEMA。

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA
 'public';
```

6. 根据需要，向使用者集群中的用户或角色授权对从数据共享创建的数据库和架构引用的权限。有关更多信息，请参阅GRANT或REVOKE。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

如果创建数据库时不使用 WITH PERMISSIONS，则只能将从数据共享创建的整个数据库的权限分配给用户和角色。在某些情况下，您需要对根据数据共享创建的数据库对象子集进行精细控制。如果是这样，您可以创建一个外部 schema 引用，该引用指向数据共享中的特定 schema（如上一步所述），并在 schema 级别提供精细权限。

您还可以基于共享的对象创建后期绑定视图，并使用这些视图来分配精细权限。您还可以考虑让创建器集群为您创建具有所需精细度的额外的数据共享。您可以根据需要为从数据共享中创建的数据库创建尽可能多的架构引用。

如果您在步骤 4 中使用 WITH PERMISSIONS 创建了数据库，则必须为共享数据库中的对象分配对象级权限。只有 USAGE 权限的用户在获得其他对象级权限之前，无法访问使用 WITH PERMISSIONS 创建的数据库中的任何对象。

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

7. 在数据共享中查询共享对象中的数据。

   对使用者集群上的使用者数据库和架构具有权限的用户和角色可以浏览和导航任何共享对象的元数据。他们还可以浏览和导航使用者集群中的本地对象。为此，他们使用 JDBC 或 ODBC 驱动程序或 SVV_ALL 和 SVV_REDSHIFT 视图。

   创建器集群在数据库中可能有许多 schema、表和每个 schema 中的视图。使用者端的用户只能看到通过数据共享提供的对象的子集。这些用户无法从创建器集群中看到整个元数据。此方法有助于通过数据共享提供精细的元数据安全控制。

   您将继续连接到本地集群数据库。但现在，您也可以使用三部分 database.schema.table 表示法从数据共享创建的数据库和 schema 中读取。您可以跨您可见的任何数据库和所有数据库执行查询。这些数据库可以是集群上的本地数据库，也可以是通过数据共享创建的数据库。使用者集群无法连接到从数据共享创建的数据库。

   您可以使用完全资格认证来访问数据。有关更多信息，请参阅 使用跨数据库查询的示例。

```
SELECT * FROM sales_db.public.tickit_sales_redshift ORDER BY 1,2 LIMIT 5;

 salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime
```

```
---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------
       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |    728.00 |
   109.20 | 2008-02-18 02:36:48
       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |     76.00 |
    11.40 | 2008-06-06 05:00:16
       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |    350.00 |
    52.50 | 2008-06-06 08:26:17
       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |    175.00 |
    26.25 | 2008-06-09 08:38:52
       5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |    154.00 |
    23.10 | 2008-08-31 09:17:02
```

您只能在共享对象上使用 SELECT 语句。但是，您可以通过查询来自不同本地数据库中的共享对象的数据，在使用者集群中创建表。

除查询之外，使用者还可以对共享对象创建视图。仅支持后期绑定视图或实体化视图。Amazon Redshift 不支持共享数据的常规视图。使用者创建的视图可跨越多个本地数据库或通过数据共享创建的数据库。有关更多信息，请参阅 CREATE VIEW。

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

使用 AWS Lake Formation 托管的数据共享

将数据共享到 AWS Lake Formation 让您可以集中定义 Amazon Redshift 数据共享的 AWS Lake Formation 权限，并限制用户对数据共享内对象的访问。

以创建者身份使用 Lake Formation 托管的数据共享

作为生产者集群或工作组管理员，请按照以下步骤将数据共享共享到 Lake Formation：

1. 在集群中创建数据共享，并授权 AWS Lake Formation 访问数据共享。

只有集群超级用户和数据库拥有者才可以创建数据共享。在创建过程中，每个数据共享都与数据库相关联。只有该数据库中的对象才能在该数据共享中共享。可以在具有相同或不同粒度对象的同一数据库上创建多个数据共享。集群可以创建的数据共享数量没有限制。

```
CREATE DATASHARE salesshare;
```

2. 将对象添加到数据共享。生产者集群或工作组管理员继续管理可用的数据共享对象。要将对象添加到数据共享中，请在添加对象之前添加 schema。当您添加 schema 时，Amazon Redshift 不会在其下添加所有对象。您必须显式添加它们。有关更多信息，请参阅更改数据共享。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

您还可以将视图添加到数据共享。支持的视图为标准视图、后期绑定视图和实体化视图。

```
CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
 public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
```

使用 ALTER DATASHARE 共享 Schema 以及给定 Schema 中的表和视图。超级用户、数据共享拥有者或对数据共享具有 ALTER 或 ALL 权限的用户可以更改数据共享以向其中添加对象或从中删除对象。数据库用户应该是对象的拥有者，或者对这些对象具有 SELECT、USAGE 或 ALL 权限。

使用 INCLUDENEW 子句将在指定 Schema 中创建的任何新表和视图添加到数据共享。只有超级用户才可以更改每个数据共享-schema 对的此属性。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

3. 向 Lake Formation 管理员账户授予数据共享的访问权限。

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '012345678910' VIA DATA CATALOG;
```

要撤销使用，请使用以下命令。

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '012345678910' VIA DATA CATALOG;
```

4. 使用 `aws redshift authorize-data-share` API 操作授予对 Lake Formation 数据共享的访问权限。这样可让 Lake Formation 识别服务账户中的数据共享，并管理与数据共享关联的使用者。

```
aws redshift authorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier {"DataCatalog/<consumer-account-id>"}
```

要从 Lake Formation 托管的数据共享中删除授权，请使用 `aws redshift deauthorize-data-share` API 操作。这样，您可以让 AWS Lake Formation 识别服务账户中的数据共享，并删除授权。

```
aws redshift deauthorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier {"DataCatalog/<consumer-account-id>"}
```

在任何时候，如果生产者集群或工作组管理员决定不再需要与使用者集群或工作组共享数据，他们可以使用 DROP DATASHARE 删除数据共享、取消对数据共享的授权或者撤销数据共享权限。系统不会自动删除 Lake Formation 中的关联权限和对象。

```
DROP DATASHARE salesshare;
```

授权 Lake Formation 账户管理数据共享后，Lake Formation 管理员可以发现共享的数据共享，将数据共享与数据目录 ARN 关联，并在链接到数据共享的 AWS Glue Data Catalog 中创建数据库。要使用 AWS CLI 关联数据共享，请使用命令 associate-data-share-consumer。要跨 AWS 区域共享数据共享，请在 `associate-data-share-consumer` 命令中指定 `--region` 参数，或使用 AWS 控制台来选择您的数据使用者。以下示例演示如何跨区域共享 Lake Formation 托管式数据共享。

```
aws redshift associate-data-share-consumer --region <region-1>
--data-share-arn 'arn:aws:redshift:us-
east-1:12345678912:datashare:035c45ea-61ce-86f0-8b75-19ac6102c3b7/sample_share'
--consumer-arn 'arn:aws:glue:<region-1>:111912345678:catalog'
```

Lake Formation 管理员还必须创建本地资源，用于定义数据共享中的对象应如何映射到 Lake Formation 中的对象。有关发现数据共享以及创建本地资源的更多信息，请参阅管理 Amazon Redshift 数据共享中数据的权限。

以使用者身份使用 Lake Formation 托管的数据共享

AWS Lake Formation 管理员发现数据共享邀请并在 AWS Glue Data Catalog 中创建链接到数据共享的数据库后，使用者集群或工作组管理员可以将集群与数据共享和 AWS Glue Data Catalog 中的数据库相关联，创建使用者集群或工作组的本地数据库，并向 Amazon Redshift 使用者集群或工作组中的用户和角色授予访问权限，以便开始查询。按照这些步骤设置查询权限。

1. 如果需要，在 Amazon Redshift 控制台上创建一个 Redshift 集群，用作使用者集群或工作组。有关如何创建集群的信息，请参阅创建集群。
2. 要列出用户可以访问 AWS Glue Data Catalog 使用者集群或工作组中的哪些数据库，请运行 SHOW DATABASES 命令。

   ```
   SHOW DATABASES FROM DATA CATALOG [ACCOUNT <account-id>,<account-id2>] [LIKE
    <expression>]
   ```

   执行此命令会列出 Data Catalog 中可用的资源，例如 AWS Glue 数据库的 ARN、数据库名称和有关数据共享的信息。
3. 使用来自 SHOW DATABASES 的 AWS Glue 数据库 ARN 在使用者集群或工作组中创建本地数据库。有关更多信息，请参阅创建数据库。

   ```
   CREATE DATABASE lf_db FROM ARN <lake-formation-database-ARN> WITH [NO] DATA CATALOG
    SCHEMA [<schema>];
   ```
4. 根据需要，向使用者集群或工作组中的用户和角色授予对从数据共享创建的数据库和架构引用的权限。有关更多信息，请参阅授权或撤销。请注意，通过 CREATE USER 命令创建的用户无法访问数据共享中已共享到 Lake Formation 的对象。只有同时具有 Redshift 和 Lake Formation 访问权限的用户才可以访问已与 Lake Formation 共享的数据共享。

   ```
   GRANT USAGE ON DATABASE sales_db TO IAM:Bob;
   ```

   作为使用者集群或工作组管理员，您只能将对从数据共享创建的整个数据库的权限分配给您的用户和角色。在某些情况下，您需要对根据数据共享创建的数据库对象子集进行精细控制。

您还可以基于共享的对象创建后期绑定视图，并使用这些视图来分配精细权限。您还可以考虑让生产者集群或工作组为您创建具有所需精细度的额外的数据共享。您可以为从数据共享中创建的数据库创建尽可能多的 schema 引用。

5. 数据库用户可以使用视图 SVV_EXTERNAL_TABLES 和 SVV_EXTERNAL_COLUMNS 来查找 AWS Glue 数据库中的所有共享表或列

```
SELECT * from svv_external_tables WHERE redshift_database_name = 'lf_db';

SELECT * from svv_external_columns WHERE redshift_database_name = 'lf_db';
```

6. 在数据共享中查询共享对象中的数据。

   对使用者集群或工作组上的使用者数据库和架构具有权限的用户和角色可以浏览和导航任何共享对象的元数据。他们还可以浏览和导航使用者集群或工作组中的本地对象。为此，他们可以使用 JDBC 或 ODBC 驱动程序或 SVV_ALL 和 SVV_EXTERNAL 视图。

```
SELECT * FROM lf_db.schema.table;
```

   您只能在共享对象上使用 SELECT 语句。但是，您可以通过查询来自不同本地数据库中的共享对象的数据，在使用者集群中创建表。

```
// Connect to a local cluster database

// Create a view on shared objects and access it.

CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

使用控制台管理数据共享

使用 Amazon Redshift 控制台管理在您的账户中创建或从其他账户共享的数据共享。

您需要权限才能创建、编辑或删除数据共享。有关信息，请参阅在 Amazon Redshift 中管理数据共享的权限。

- 如果您是创建器集群管理员，则可以创建数据共享、添加数据使用者、添加数据共享对象、从数据共享创建数据库、编辑数据共享或从 CLUSTERS 选项卡中删除数据共享。

  在导航菜单中，导航到集群选项卡，从集群列表中选择一个集群。然后，执行以下操作之一：

  - 选择数据共享选项卡，从在我的命名空间中创建的数据共享部分选择数据共享。然后，执行以下操作之一：
    - 创建数据共享

      创建数据共享时，您可以添加数据共享对象或数据使用者。有关更多信息，请参阅将数据共享对象添加到数据共享和将数据使用者添加到数据共享。
    - 编辑在您的账户中创建的数据共享
    - 删除在您的账户中创建的数据共享
  - 选择数据共享，并从其他集群中的数据共享部分选择数据共享。然后，执行以下操作之一：
    - 创建数据共享
    - 通过数据共享创建数据库
  - 选择数据库，然后从数据库部分选择一个数据库。然后选择创建数据共享。有关更多信息，请参阅通过数据共享创建数据库。

> **Note**
>
> 要查看数据库及数据库内的对象或查看集群中的数据共享，请连接到数据库。有关更多信息，请参阅 连接到数据库。

连接到数据库

连接到数据库以查看数据库和此集群中的数据库内的对象，或查看数据共享。

用于连接到指定数据库的用户凭证必须具有必要的权限才能查看所有数据共享。

如果没有本地连接，请执行以下操作之一：

- 在集群详细信息页面上，从数据库选项卡的数据库或数据共享对象部分中，选择连接到数据库以查看集群中的数据库对象。

- 在集群详细信息页面上，从数据共享选项卡中，执行下列操作之一：
  - 在来自其他集群的数据共享部分中，选择连接到数据库以查看来自其他集群的数据共享。
  - 在在我的集群中创建的数据共享部分中，选择连接到数据库以查看您的集群中的数据共享。
- 在连接到数据库窗口中，执行以下操作之一：
  - 如果选择创建新连接，请选择 AWS Secrets Manager 以使用存储的密钥对连接的访问进行身份验证。

    或者，选择临时凭证以使用数据库凭证对连接的访问进行身份验证。指定数据库名称和数据库用户的值。

    选择连接。
  - 选择使用最近的连接以连接到您具备必要权限的另一个数据库。

    Amazon Redshift 会自动建立连接。

建立数据库连接后，您可以开始创建数据共享、查询数据共享或从数据共享创建数据库。

创建数据共享

创建数据共享

作为创建器集群管理员，您可以从“集群详细信息”页面中的数据库或数据共享选项卡中创建数据共享。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 在集群详细信息页面上，执行以下操作之一：
   - 从数据库选项卡上的数据库部分中，选择数据库。此时将会显示数据库详细信息页面。

     选择创建数据共享。您只能从本地数据库创建数据共享。如果您未连接到数据库，则会显示连接到数据库页面。按连接到数据库中的步骤连接到数据库。如果最近存在连接，则会显示创建数据共享页面。
   - 如果您没有数据库连接，则从数据共享选项卡上的数据共享部分中，连接到数据库。

     在在我的集群中创建的数据共享部分中，选择创建数据共享。此时会显示“创建数据共享”页面。
4. 在数据共享信息部分，选择以下选项之一：

- 选择数据共享创建数据共享，以便在不同的 Amazon Redshift 集群或在同一个 AWS 账户 或不同 AWS 账户 中共享数据，从而用于读取。
- 选择 AWS Data Exchange 数据共享创建数据共享，以通过 AWS Data Exchange 许可您的数据。

5. 指定数据共享名称、数据库名称和可公开访问的值。

   更改数据库名称时，请建立新的数据库连接。

6. 在数据共享对象部分中，选择添加。此时会显示“添加数据共享”页面。要向数据共享中添加对象，请按将数据共享对象添加到数据共享操作。
7. 在数据使用者部分，您可以选择发布到 Redshift 账户，或者发布到 AWS Glue Data Catalog，这将开始通过 Lake Formation 共享数据的过程。将数据共享发布到 Redshift 账户即意味着与另一个充当使用者集群的 Redshift 账户共享数据。

   > **Note**
   >
   > 创建数据共享后，您无法编辑配置以发布到其他选项。

8. 选择创建数据共享。

Amazon Redshift 会创建数据共享。创建数据共享之后，您可以从数据共享中创建数据库。

将数据共享对象添加到数据共享

向数据共享中添加一个或多个对象。对于数据使用者来说，数据共享对象是只读的。

您可以在不添加数据共享对象的情况下创建数据共享，并在稍后添加对象。

只有在向数据共享中添加至少一个对象时，数据共享才会变为活动状态。

1. 从数据共享列表中选择要向其添加对象的数据共享。
2. 选择添加。将出现“添加数据共享对象”页面。
3. 在添加其它数据共享对象之前，至少向数据共享添加至少一个 schema。通过选择添加并重复添加多个 schema。
4. 您可以选择从指定 schema 中添加所选对象类型的所有现有对象，也可以从指定 schema 中添加特定的单个对象。选择对象类型，例如表和视图或用户定义的函数。
5. 您可以选择添加并重复以添加指定的 schema 和数据共享对象，并继续添加另一个对象。

将数据使用者添加到数据共享

您可以将一个或多个数据使用者添加到数据共享。数据使用者可以是唯一标识 Amazon Redshift 集群或 AWS 账户 的集群命名空间。

您必须明确选择禁用或启用将您的数据共享与具有公共访问权限的集群共享。

- 选择将集群命名空间添加到数据共享。命名空间是 Amazon Redshift 集群的全局唯一标识符 (GUID)。
- 选择向数据共享添加 AWS 账户。指定的 AWS 账户 必须具有对数据共享的访问权限。

授权或删除数据共享中的授权

作为创建者集群管理员，选择要授权哪些数据使用者访问数据共享或从中删除授权。授权的数据使用者会收到通知，以便对数据共享采取操作。如果您要以数据使用者身份添加集群命名空间，则不必执行授权。

先决条件：要授权或删除对数据共享的授权，必须至少将一个数据使用者添加到数据共享中。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择数据共享。此时将会显示数据共享列表页面。
3. 选择在我的账户中。
4. 在我的账户中的数据共享部分中，执行以下操作之一：
   - 选择要授权的一个或多个使用者集群。此时将显示“授权数据使用者”页面。然后选择授权。

     如果在创建数据共享时选择了发布到 AWS Glue Data Catalog，则只能向 Lake Formation 账户授予数据共享的权限。

     对于 AWS Data Exchange 数据共享，一次只能授权一个数据共享。

     当您授权 AWS Data Exchange 数据共享时，您正在与 AWS Data Exchange 服务共享数据共享，并允许 AWS Data Exchange 代表您管理对数据共享的访问权限。当使用者订阅产品时，AWS Data Exchange 通过将使用者作为数据使用者添加到 AWS Data Exchange 数据共享，从而允许对使用者进行访问。AWS Data Exchange 对数据共享没有读取权限。
   - 选择要对其删除授权的一个或多个使用者集群。然后选择删除授权。

数据使用者获得授权后，可以访问数据共享对象并创建使用者数据库以查询数据。

删除授权后，数据使用者将立即失去对数据共享的访问权限。

以使用者身份管理来自其他账户的数据共享

关联数据共享

作为使用者集群管理员，您可以将从其它账户共享的一个或多个数据共享与您的整个 AWS 账户或您账户中的特定集群命名空间关联。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择数据共享。此时将会显示数据共享列表页面。
3. 选择来自其他账户。
4. 在来自其它账户的数据共享部分中，选择要关联的数据共享，并选择关联。出现“关联数据共享”页面时，请选择以下关联类型之一：
   - 选择整个 AWS 账户，以将您的 AWS 账户中不同 AWS 区域的所有现有和未来的集群命名空间关联到该数据共享。然后选择关联。

     如果将数据共享发布到 AWS Glue Data Catalog，则只能将数据共享与整个 AWS 账户关联。
   - 选择特定 AWS 区域和集群命名空间，以将一个或多个 AWS 区域和特定集群命名空间关联到该数据共享。
     a. 选择添加区域，将特定 AWS 区域和集群命名空间添加至数据共享。此时将显示添加 AWS 区域页面。
     b. 选择 AWS 区域。
     c. 请执行下列操作之一：
        - 选择添加所有集群命名空间，将此区域中的所有现有和未来的集群命名空间添加至数据共享。
        - 选择添加特定集群命名空间将该区域中的一个或多个特定集群命名空间与数据共享关联。
        - 选择一个或多个集群命名空间，然后选择添加 AWS 区域。
     d. 选择关联。

如果您要将数据共享与 Lake Formation 账户相关联，请前往 Lake Formation 控制台创建数据库，然后定义数据库权限。有关更多信息，请参阅《AWS Lake Formation 开发人员指南》中的为 Amazon Redshift 数据共享设置权限。创建 AWS Glue 数据库或联合数据库后，您可以使用查询编辑器 v2 或任何首选 SQL 客户端与您的使用者集群一起查询数据。有关更多信息，请参阅 以使用者身份使用 Lake Formation 托管的数据共享。

关联数据共享后，数据共享将变为可用状态。

您还可以随时更改数据共享关联。将关联从特定 AWS 区域和集群命名空间更改到整个 AWS 账户时，Amazon Redshift 会使用 AWS 账户信息覆盖特定区域和集群命名空间信息。然后，AWS 账户中的所有 AWS 区域和集群命名空间即可访问数据共享。

将关联从特定集群命名空间更改为特定 AWS 区域中的所有集群命名空间后，此区域中的所有集群命名空间均可访问数据共享。

从数据使用者中移除数据共享的关联

作为使用者集群管理员，您可以从数据使用者中删除数据共享的关联。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择数据共享。此时将会显示数据共享列表页面。
3. 选择来自其他账户。
4. 在来自其他账户的数据共享部分中，选择要从数据使用者中删除关联的数据共享。
5. 在数据使用者部分中，选择要删除关联的一个或多个数据使用者。然后选择删除关联。
6. 当显示“删除关联”页面时，选择删除关联。

删除关联后，数据使用者将失去对数据共享的访问权限。您可以随时更改数据使用者关联。

拒绝数据共享

作为使用者集群管理员，您可以拒绝任何状态为可用或活动的数据共享。拒绝数据共享后，使用者集群用户会失去对数据共享的访问权限。在您调用 `DescribeDataSharesForConsumer` API 操作时，Amazon Redshift 不会返回被拒绝的数据共享。如果生产者集群管理员运行 `DescribeDataSharesForProducer` API 操作，他们将看到数据共享被拒绝。数据共享被拒绝后，生产者集群管理员可以再次将数据共享授权给使用者集群，而使用者集群管理员可以选择将其 AWS 账户与数据共享关联，也可以拒绝该数据共享。

如果您的 AWS 账户与数据共享关联，并且有一个由 Lake Formation 管理的数据共享关联等待处理，则拒绝由 Lake Formation 管理的数据共享关联也会拒绝原始数据共享。要拒绝特定关联，生产者集群管理员可以从指定数据共享移除授权。此操作不会影响其他数据共享。

要拒绝数据共享，请使用 AWS 控制台、API 操作 `RejectDataShare` 或 AWS CLI 中的 `reject-datashare`。

要使用 AWS 控制台拒绝数据共享，请执行以下操作：

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择数据共享。
3. 选择来自其他账户。
4. 在来自其他账户的数据共享部分中，选择要拒绝的数据共享。当显示拒绝数据共享页面时，请选择拒绝。

拒绝数据共享之后，您无法恢复更改。Amazon Redshift 将从列表中移除数据共享。要再次查看数据共享，生产者管理员必须再次对其进行授权。

管理现有数据共享

查看数据共享

从 DATASHARES 或 CLUSTERS 选项卡中查看数据共享。

- 使用 DATASHARES 选项卡列出您的账户或其他账户中的数据共享。
  - 要查看在您的账户中创建的数据共享，请选择在我的账户中，然后选择要查看的数据共享。
  - 要查看从其他账户共享的数据共享，请选择来自其他账户，然后选择要查看的数据共享。
- 使用 CLUSTERS 选项卡列出您的集群中或其他集群中的数据共享。

  连接到数据库。有关更多信息，请参阅 连接到数据库。

  然后从来自其他集群的数据共享或在我的集群中创建的数据共享部分中选择一个数据共享，以查看其详细信息。

从数据共享中删除数据共享对象

您可以按照以下过程从数据共享中删除一个或多个对象。

要从数据共享中删除一个或多个对象

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。

3. 选择数据共享。
4. 在在我的账户中创建的数据共享部分中，选择连接到数据库。有关更多信息，请参阅 连接到数据库。
5. 选择要编辑的数据共享，然后选择编辑。此时将会显示数据共享详细信息页面。
6. 要将一个或多个数据共享对象从数据共享中删除，请执行下列操作之一：
   - 要从数据共享中删除 schema，请选择一个或多个 schema。然后选择删除。Amazon Redshift 会从数据共享中删除指定的 schema 和指定 schema 的所有对象。
   - 要从数据共享中删除表和视图，请选择一个或多个表和视图。然后选择删除。或者，选择按 schema 删除以删除指定 schema 中的所有表和视图。
   - 要从数据共享中删除用户定义的函数，请选择一个或多个用户定义的函数。然后选择删除。或者，选择按 schema 删除以删除指定 schema 中的所有用户定义函数。

从数据共享中删除数据使用者

您可以从数据共享中删除一个或多个数据使用者。数据使用者可以是唯一标识 Amazon Redshift 集群或 AWS 账户的集群命名空间。

从集群命名空间 ID 或 AWS 账户中选择一个或多个数据使用者，然后选择删除。

Amazon Redshift 会从数据共享中删除指定的数据使用者。他们将立即失去对数据共享的访问权限。

编辑在您的账户中创建的数据共享

使用控制台编辑在您的账户中创建的数据共享。首先连接到数据库，以查看在您的账户中创建的数据共享列表。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 选择数据共享。
4. 在在我的账户中创建的数据共享部分中，选择连接到数据库。有关更多信息，请参阅 连接到数据库。
5. 选择要编辑的数据共享，然后选择编辑。此时将会显示数据共享详细信息页面。
6. 在数据共享对象或数据使用者部分中进行任何更改。

> **Note**
>
> 如果您选择将数据共享发布到 AWS Glue Data Catalog，则无法编辑配置以将数据共享发布到其他 Amazon Redshift 账户。

7. 选择保存更改。

Amazon Redshift 会使用更改更新您的数据共享。

删除在您的账户中创建的数据共享

使用控制台删除在您的账户中创建的数据共享。首先连接到数据库，以查看在您的账户中创建的数据共享列表。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 选择数据共享。此时将会显示数据共享列表。
4. 在在我的账户中创建的数据共享部分中，选择连接到数据库。有关更多信息，请参阅 连接到数据库。
5. 选择要删除的一个或多个数据共享，然后选择删除。此时会显示“删除数据共享”页面。

   删除与 Lake Formation 共享的数据共享不会自动删除 Lake Formation 中的相关权限。要删除权限，请前往 Lake Formation 控制台。
6. 键入 Delete 以确认删除指定的数据共享。
7. 选择 Delete。

删除数据共享后，数据共享使用者将失去对数据共享的访问权限。

查询数据共享

通过数据共享创建数据库

要开始在数据共享中查询数据，请从数据共享中创建一个数据库。您只能从指定的数据共享中创建一个数据库。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 选择数据共享。此时将会显示数据共享列表。
4. 在来自其他集群的数据共享部分中，选择连接到数据库。有关更多信息，请参阅 连接到数据库。
5. 选择要从中创建数据库的数据共享，然后选择从数据共享中创建数据库。此时将显示“从数据共享创建数据库”页面。
6. 在数据库名称中，指定数据库名称。数据库名称必须为 1-64 个字母数字字符（仅限小写），且不能是保留字。
7. 选择创建。

创建数据库之后，您可以查询数据库中的数据。

管理 AWS Data Exchange 数据共享

在 AWS Data Exchange 上创建数据集

在 AWS Data Exchange 上创建数据集。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 选择数据共享。
4. 在在我的账户中创建的数据共享部分中，选择 AWS Data Exchange 数据共享。
5. 选择在 AWS Data Exchange 上创建数据集。有关更多信息，请参阅发布新产品。

编辑 AWS Data Exchange 数据共享

使用控制台编辑 AWS Data Exchange 数据共享。首先连接到数据库，以查看在您的账户中创建的数据共享列表。

对于 AWS Data Exchange 数据分享，您无法更改数据使用者。

要编辑 AWS Data Exchange 数据共享的可公开访问设置，请使用查询编辑器 v2。Amazon Redshift 会生成一个随机的一次性值来设置会话变量，以允许关闭此设置。有关更多信息，请参阅 ALTER DATASHARE 使用说明。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 在导航器菜单中，选择编辑器，然后选择查询编辑器 v2。
4. 如果这是您首次使用查询编辑器 v2，请配置 AWS 账户。默认情况下，AWS 拥有的密钥用于加密资源。有关配置 AWS 账户 的更多信息，请参阅《Amazon Redshift 管理指南》中的配置您的 AWS 账户。
5. 要连接您的集群所在的 AWS Data Exchange 数据共享，请选择数据库以及树视图面板中的集群名称。如果出现提示，请输入连接参数。
6. 复制以下 SQL 语句。以下示例更改了 Salesshare 数据共享的可公开访问设置。

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
```

7. 要运行复制的 SQL 语句，请选择查询，然后将复制的 SQL 语句粘贴到查询区域中。然后，选择运行。

   出现以下情况时会显示错误：

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
ERROR:  Alter of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value 'c670ba4db22f4b'
```

   ‘c670ba4db22f4b’是一个随机的一次性值，当发生不推荐的操作时，Amazon Redshift 会生成该值。

8. 将以下示例语句复制并粘贴到查询区域中。然后运行命令。该 SET datashare_break_glass_session_var 命令应用权限，允许对 AWS Data Exchange 数据共享进行不推荐的操作。

```
SET datashare_break_glass_session_var to 'c670ba4db22f4b';
```

9. 再次运行 ALTER DATASHARE 语句。

```
ALTER DATASHARE salesshare;
```

Amazon Redshift 会使用更改更新您的数据共享。

删除在您的账户中创建的 AWS Data Exchange 数据共享

使用控制台删除在您的账户中创建的 AWS Data Exchange 数据共享。首先连接到数据库，以查看在您的账户中创建的数据共享列表。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 在导航器菜单中，选择编辑器，然后选择查询编辑器 v2。
4. 如果这是您首次使用查询编辑器 v2，请配置 AWS 账户。默认情况下，AWS 拥有的密钥用于加密资源。有关配置 AWS 账户 的更多信息，请参阅《Amazon Redshift 管理指南》中的配置您的 AWS 账户。
5. 要连接您的集群所在的 AWS Data Exchange 数据共享，请选择数据库以及树视图面板中的集群名称。如果出现提示，请输入连接参数。
6. 复制以下 SQL 语句。以下示例将删除 SalesShare 数据共享。

```
DROP DATASHARE salesshare
```

7. 要运行复制的 SQL 语句，请选择查询，然后将复制的 SQL 语句粘贴到查询区域中。然后，选择运行。

   出现以下情况时会显示错误：

```
ERROR:  Drop of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value '620c871f890c49'
```

   ‘620c871f890c49’是一个随机的一次性值，当发生不推荐的操作时，Amazon Redshift 会生成该值。
8. 将以下示例语句复制并粘贴到查询区域中。然后运行命令。该 SET datashare_break_glass_session_var 命令应用权限，允许对 AWS Data Exchange 数据共享进行不推荐的操作。

```
SET datashare_break_glass_session_var to '620c871f890c49';
```

9. 再次运行 DROP DATASHARE 语句。

```
DROP DATASHARE salesshare;
```

删除数据共享后，数据共享使用者将失去对数据共享的访问权限。

删除共享 AWS Data Exchange 数据共享可能会违反 AWS Data Exchange 中的数据产品条款。

使用 AWS CloudFormation 管理数据共享

您可以通过使用配置 AWS 资源的 AWS CloudFormation 堆栈，自动执行数据共享设置。CloudFormation 堆栈设置同一个 AWS 账户中两个 Amazon Redshift 集群之间的数据共享。因此，您可以在不运行 SQL 语句来预调配资源的情况下开始数据共享。

堆栈在您指定的集群上创建了数据共享。数据共享包括表格和只读示例数据。这些数据可以被另一个 Amazon Redshift 集群读取。

如果您想通过运行 SQL 语句在 AWS 账户中开始共享数据，以便在不使用 CloudFormation 的情况下设置数据共享并授予权限，请参阅共享对 AWS 账户内数据的读取访问权限。

运行数据共享 CloudFormation 堆栈之前，您必须使用有权创建 IAM 角色和 Lambda 函数的用户进行登录。您还需要同一个账户中的两个 Amazon Redshift 集群。一个作为创建者，用于共享示例数据，另一个作为使用者，用于读取该数据。对于这些集群的主要要求是每个集群均需使用 RA3 节点。有关其他要求，请参阅在 Amazon Redshift 中使用数据共享的注意事项。

有关开始设置 Amazon Redshift 集群的更多信息，请参阅 Amazon Redshift 预置集群。有关使用 CloudFormation 自动进行设置的更多信息，请参阅什么是 AWS CloudFormation？。

> **Important**
>
> 在启动 CloudFormation 堆栈之前，请确保在同一账户中有两个 Amazon Redshift 集群，并且这些集群使用 RA3 节点。确保每个集群都有一个数据库和一个超级用户。有关更多信息，请参阅CREATE DATABASE 和superuser。

要启动您的 CloudFormation 堆栈，以进行 Amazon Redshift 数据共享，请执行以下操作：

1. 单击启动 CFN 堆栈，将您带到 AWS Management Console 中的 CloudFormation 服务。

   如果出现登录提示，请登录。

   堆栈创建过程启动，引用存储在 Amazon S3 中的 CloudFormation 模板文件。CloudFormation 模板指的是声明组建堆栈的 AWS 资源的 JSON 格式文本文件。有关 CloudFormation 模板的更多信息，请参阅了解模板基础知识。
2. 选择下一步，输入堆栈详细信息。

3. 根据参数中的每个集群，输入以下内容：

   - 您的 Amazon Redshift 集群名称，例如 **ra3-consumer-cluster**
   - 您的数据库名称，例如 **dev**
   - 数据库用户的名称，例如 **consumeruser**

   我们建议使用测试集群，因为堆栈会创建多个数据库对象。

   选择下一步。
4. 此时将显示堆栈选项。

   选择下一步以接受默认设置。
5. 在功能下，选择我确认 AWS CloudFormation 可能会创建 IAM 资源。
6. 选择创建堆栈。

CloudFormation 使用模板构建 Amazon Redshift 堆栈大约需要 10 分钟时间，会创建一个名为 `myproducer_share` 的数据共享。堆栈在堆栈详细信息中指定的数据库中创建数据共享。只有该数据库中的对象才能共享。

如果创建堆栈时发生错误，请执行以下操作：

- 确保为每个 Redshift 集群输入了正确的集群名称、数据库名称和数据库用户名。
- 确保您的集群有 RA3 节点。
- 请确保您以具有创建 IAM 角色和 Lambda 函数的权限的用户身份登录。有关创建 IAM 角色的更多信息，请参阅创建 IAM 角色。有关创建 Λ 函数的策略的更多信息，请参阅函数开发。

查询您创建的数据共享

要使用以下过程，请确保您拥有在所述每个集群上运行查询所需的权限。

要查询您的数据共享：

1. 使用客户端工具（如 Amazon Redshift 查询编辑器 v2），连接到创建 CloudFormation 堆栈时输入的数据库上的生产者集群。
2. 查询数据共享。

```
SHOW DATASHARES;
```

```
+-----------------+------------+----------------+------------------
+-----------+-----------+--------------------+----------+-----------------
+-------------------------------------+
|    share_name    | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace          |
+-----------------+------------+----------------+------------------
+-----------+-----------+--------------------+----------+-----------------
+-------------------------------------+
| myproducer_share | 100         | sample_data_dev | myconsumer_db     | INBOUND
  | NULL       | true               | NULL      |  producer-acct   |        your-
producer-namespace       |
+-----------------+------------+----------------+------------------
+-----------+-----------+--------------------+----------+-----------------
+-------------------------------------+
```

前面的命令返回堆栈创建的数据共享的名称，名为 myproducer_share。它还会返回与数据共享 myconsumer_db 关联的数据库名称。

复制创建者命名空间标识符，以便在后面的步骤中使用。

3. 描述数据共享中的对象。

```
DESC DATASHARE myproducer_share;

+-----------------+-------------------------------------+-----------
+-----------------+------------+-------------------------------------
+-------------+
| producer_account |          producer_namespace          | share_type |
 share_name    | object_type |             object_name             | include_new |
+-----------------+-------------------------------------+-----------
+-----------------+------------+-------------------------------------
+-------------+
|   producer-acct  |        your-producer-namespace       | OUTBOUND   |
 myproducer_share | schema      | myproducer_schema                   | true
 |
|   producer-acct  |        your-producer-namespace       | OUTBOUND   |
 myproducer_share | table       | myproducer_schema.tickit_sales      | NULL
 |
|   producer-acct  |        your-producer-namespace       | OUTBOUND   |
 myproducer_share | view        | myproducer_schema.ticket_sales_view | NULL
 |
```

```
+-----------------+---------------------------------------+-----------
+-----------------+-------------+-------------------------------------
+-------------+
```

描述数据共享时，它会返回表和视图属性。堆栈会将包含示例数据的表和视图添加到创建者数据库中，例如 `tickit_sales` 和 `tickit_sales_view`。有关 TICKIT 示例数据库的更多信息，请参阅示例数据库。

您无需委托数据共享权限即可运行查询。堆栈会授予必要权限。

4. 使用客户端工具连接使用者集群。描述数据共享，指定创建者的命名空间。

```
DESC DATASHARE myproducer_share OF NAMESPACE '<namespace id>'; --specify the unique
 identifier for the producer namespace

+-----------------+---------------------------------------+-----------
+-----------------+-------------+-------------------------------------
+-------------+
| producer_account |          producer_namespace          | share_type |
 share_name    | object_type |             object_name             | include_new |
+-----------------+---------------------------------------+-----------
+-----------------+-------------+-------------------------------------
+-------------+
|   producer-acct  |        your-producer-namespace       | INBOUND    |
 myproducer_share | schema      | myproducer_schema                   | NULL
 |
|   producer-acct  |        your-producer-namespace       | INBOUND    |
 myproducer_share | table       | myproducer_schema.tickit_sales      | NULL
 |
|   producer-acct  |        your-producer-namespace       | INBOUND    |
 myproducer_share | view        | myproducer_schema.ticket_sales_view | NULL
 |
+-----------------+---------------------------------------+-----------
+-----------------+-------------+-------------------------------------
+-------------+
```

5. 您可以通过指定数据共享的数据库和架构来查询数据共享中的表。有关更多信息，请参阅 使用跨数据库查询的示例。以下查询从 TICKIT 示例数据库的 SALES 表中返回销售和卖家数据。有关更多信息，请参阅 SALES 表。

```
SELECT * FROM myconsumer_db.myproducer_schema.tickit_sales_view;
```

```
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
|       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |       728 |
     109.2 | 2008-02-18 02:36:48 |
|       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |        76 |
      11.4 | 2008-06-06 05:00:16 |
|       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |       350 |
      52.5 | 2008-06-06 08:26:17 |
|       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |       175 |
     26.25 | 2008-06-09 08:38:52 |
|       5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |       154 |
      23.1 | 2008-08-31 09:17:02 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

> **Note**
>
> 查询会根据共享架构中的视图运行。您无法直接连接到从数据共享创建的数据库。它们是只读的。

6. 要运行包含聚合的查询，请使用以下示例。

```
SELECT * FROM myconsumer_db.myproducer_schema.tickit_sales ORDER BY 1,2 LIMIT 5;

+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
|       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |       728 |
     109.2 | 2008-02-18 02:36:48 |
|       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |        76 |
      11.4 | 2008-06-06 05:00:16 |
|       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |       350 |
      52.5 | 2008-06-06 08:26:17 |
|       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |       175 |
     26.25 | 2008-06-09 08:38:52 |
```

```
|       5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |       154 |
      23.1 | 2008-08-31 09:17:02 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

该查询从示例 TICKIT 数据中返回销售和卖家数据。

有关数据共享查询的更多示例，请参阅共享对 AWS 账户内数据的读取访问权限。

使用控制台管理写入数据共享（预览版）

此文档为预发行文档，介绍了通过 Amazon Redshift 数据共享功能进行多数据仓库写入，该功能在 PREVIEW_2023 版本的公开预览版中提供。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

有关设置 PREVIEW_2023 库的更多信息，请参阅以下任一内容：

- 对于 Amazon Redshift Serverless 预览版：创建预览版工作组
- 对于 Amazon Redshift 预置集群预览版：创建预览版集群

有关开始使用数据共享的更多信息，请转到共享对数据的写入访问权限（预览版）。

使用 Amazon Redshift 控制台管理在您的账户中创建或从其他账户共享的数据共享。

连接到数据库（预览版）

此文档为预发行文档，介绍了通过 Amazon Redshift 数据共享功能进行多数据仓库写入，该功能在 PREVIEW_2023 版本的公开预览版中提供。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

有关开始使用数据共享的更多信息，请转到共享对数据的写入访问权限（预览版）。

连接到数据库以查看数据库和此集群中的数据库内的对象，或查看数据共享。

用于连接到指定数据库的用户凭证必须具有必要的权限才能查看所有数据共享。

如果没有本地连接，请执行以下操作之一：

- 在集群详细信息页面上，从数据库选项卡的数据库或数据共享对象部分中，选择连接到数据库以查看集群中的数据库对象。
- 在集群详细信息页面上，从数据共享选项卡中，执行下列操作之一：
  - 在来自其他集群的数据共享部分中，选择连接到数据库以查看来自其他集群的数据共享。
  - 在在我的集群中创建的数据共享部分中，选择连接到数据库以查看您的集群中的数据共享。
- 在连接到数据库窗口中，执行以下操作之一：
  - 如果选择创建新连接，请选择 AWS Secrets Manager 以使用存储的密钥对连接的访问进行身份验证。

    或者，选择临时凭证以使用数据库凭证对连接的访问进行身份验证。指定数据库名称和数据库用户的值。

    选择连接。
  - 选择使用最近的连接以连接到您具备必要权限的另一个数据库。

    Amazon Redshift 会自动建立连接。

建立数据库连接后，您可以开始创建数据共享、查询数据共享或从数据共享创建数据库。

创建数据共享和添加对象（预览版）

创建数据共享

此文档为预发行文档，介绍了通过 Amazon Redshift 数据共享功能进行多数据仓库写入，该功能在 PREVIEW_2023 版本的公开预览版中提供。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

有关开始使用数据共享的更多信息，请转到共享对数据的写入访问权限（预览版）。

作为创建器集群管理员，您可以从“集群详细信息”页面中的数据库或数据共享选项卡中创建数据共享。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。

3. 在集群详细信息页面上，执行以下操作之一：

   - 从数据库选项卡上的数据库部分中，选择数据库。此时将会显示数据库详细信息页面。

     选择创建数据共享。您只能从本地数据库创建数据共享。如果您未连接到数据库，则会显示连接到数据库页面。按连接到数据库（预览版）中的步骤连接到数据库。如果最近存在连接，则会显示创建数据共享页面。

   - 如果您没有数据库连接，则从数据共享选项卡上的数据共享部分中，连接到数据库。

     在在我的集群中创建的数据共享部分中，选择创建数据共享。此时会显示创建数据共享页面。

4. 在这里，您可以添加各种类型的数据库对象。选择添加按钮以添加对象。此时将显示对话框。执行以下步骤：

   1. 选择一个或多个架构。这样做可以使架构中的对象可供添加。
   2. 从架构中选择对象类型。

      在这里，您可以选择几个选项来添加对象：

      - 从架构中添加特定对象 – 如果选择此选项，则会按名称列出各个对象。您可以选择对象并将其添加到数据共享中。例如，您可以根据需要添加特定的表和存储过程。然后，所选架构中的表和存储过程将包含在数据共享中。后续步骤将进一步解释权限的设置。继续使用视图和其他类型，选择要添加的对象。
      - 将选定对象类型中的所有现有对象添加到架构 – 此选项将添加所有对象。

   3. 您也可以选择是否要添加未来对象。如果您选择将添加到架构中的数据共享对象包含在内，则意味着，添加到该架构中的对象会自动添加到数据共享中。
   4. 选择添加以完成该部分并添加对象。它们将列在数据共享对象下。
   5. 添加对象后，您可以选择单个对象并编辑其权限。如果您选择架构，则会出现一个对话框，询问您是否要添加限定范围权限。这使得架构中的每个现有或新增对象都具有一组适用于该对象类型的预选权限。例如，管理员可以设置所有添加的表都具有 SELECT 和 UPDATE 权限。
   6. 配置架构权限后，您可以浏览其他对象类型并选择其权限。例如，您可以向特定表添加 UPDATE 权限。
   7. 在数据使用者部分，您可以添加命名空间或添加 AWS 账户作为数据共享的使用者。
   8. 选择创建数据共享以保存您的更改。

   创建数据共享后，它会出现在我的命名空间中创建的数据共享下的列表中。如果从列表中选择数据共享，则可以查看其使用者、对象和其他属性。

将数据使用者添加到数据共享

您可以将一个或多个数据使用者添加到数据共享。数据使用者可以是唯一标识 Amazon Redshift 集群或 AWS 账户 的集群命名空间。

您必须明确选择禁用或启用将您的数据共享与具有公共访问权限的集群共享。

- 选择将集群命名空间添加到数据共享。命名空间是 Amazon Redshift 集群的全局唯一标识符 (GUID)。
- 选择向数据共享添加 AWS 账户。指定的 AWS 账户 必须具有对数据共享的访问权限。

授权或删除数据共享中的授权（预览版）

此文档为预发行文档，介绍了通过 Amazon Redshift 数据共享功能进行多数据仓库写入，该功能在 PREVIEW_2023 版本的公开预览版中提供。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

有关开始使用数据共享的更多信息，请转到共享对数据的写入访问权限（预览版）。

作为创建者集群管理员，选择要授权哪些数据使用者访问数据共享或从中删除授权。授权的数据使用者会收到通知，以便对数据共享采取操作。如果您要以数据使用者身份添加集群命名空间，则不必执行授权。

先决条件：要授权或删除对数据共享的授权，必须至少将一个数据使用者添加到数据共享中。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择数据共享。在这里，您可以看到一个名为数据共享使用者的列表。选择要授权的一个或多个使用者集群。然后选择授权。
3. 此时将出现授权账户对话框。您可以从几种授权类型中进行选择。
   - [集群名称或工作组名称] 只读 - 这意味着即使数据共享创建者授予了写入权限，使用者也没有写入权限。
   - [集群名称或工作组名称] 读写 - 这意味着使用者可以拥有创建者授予的所有权限，包括写入权限。

4. 选择保存。

您也可以用使用者身份授权 AWS Data Exchange。

1. 如果在创建数据共享时选择了发布到 AWS Glue Data Catalog，则只能向 Lake Formation 账户授予数据共享的权限。

   对于 AWS Data Exchange 数据共享，一次只能授权一个数据共享。

   当您授权 AWS Data Exchange 数据共享时，您正在与 AWS Data Exchange 服务共享数据共享，并允许 AWS Data Exchange 代表您管理对数据共享的访问权限。当使用者订阅产品时，AWS Data Exchange 通过将使用者作为数据使用者添加到 AWS Data Exchange 数据共享，从而允许对使用者进行访问。AWS Data Exchange 对数据共享没有读取权限。
2. 选择保存。

数据使用者获得授权后，可以访问数据共享对象并创建使用者数据库以查询数据。

删除授权：

选择要对其删除授权的一个或多个使用者集群。然后选择删除授权。

删除授权后，数据使用者将立即失去对数据共享的访问权限。

以使用者身份关联或拒绝数据共享（预览版）

关联数据共享

> 此文档为预发行文档，介绍了通过 Amazon Redshift 数据共享功能进行多数据仓库写入，该功能在 PREVIEW_2023 版本的公开预览版中提供。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

有关开始使用数据共享的更多信息，请转到共享对数据的写入访问权限（预览版）。

作为使用者集群管理员，您可以将从其它账户共享的一个或多个数据共享与您的整个 AWS 账户或您账户中的特定集群命名空间关联。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。

2. 在导航菜单上，选择数据共享。此时将会显示数据共享列表页面。选择来自其他账户。
3. 在来自其它账户的数据共享部分中，选择要关联的数据共享，并选择关联。出现关联数据共享页面时，请选择以下关联类型之一：

   - 选择整个 AWS 账户，以将您的 AWS 账户中不同 AWS 区域的所有现有和未来的集群命名空间关联到该数据共享。

     如果将数据共享发布到 AWS Glue Data Catalog，则只能将数据共享与整个 AWS 账户关联。
4. 您可以从这里选择允许的权限。可选项有：

   - 只读 - 如果您选择只读，则使用者无法使用诸如 UPDATE 或 INSERT 之类的写入权限，即使这些权限是在生产者上授予和授权的。
   - 读写 – 使用者数据共享用户将拥有生产者授予和授权的所有权限，包括读取和写入权限。
5. 或者，选择特定 AWS 区域和集群命名空间，以将一个或多个 AWS 区域和特定集群命名空间关联到该数据共享。选择添加区域，将特定 AWS 区域和集群命名空间添加至数据共享。此时将显示添加 AWS 区域页面。
6. 选择 AWS 区域。
7. 请执行下列操作之一：

   - 选择添加所有集群命名空间，将此区域中的所有现有和未来的集群命名空间添加至数据共享。
   - 选择添加特定集群命名空间将该区域中的一个或多个特定集群命名空间与数据共享关联。
   - 选择一个或多个集群命名空间，然后选择添加 AWS 区域。
8. 选择关联。

生产者可以返回并更改授权设置，这可能会影响使用者的关联设置。

如果您要将数据共享与 Lake Formation 账户相关联，请前往 Lake Formation 控制台创建数据库，然后定义数据库权限。有关更多信息，请参阅《AWS Lake Formation 开发人员指南》中的为 Amazon Redshift 数据共享设置权限。创建 AWS Glue 数据库或联合数据库后，您可以使用查询编辑器 v2 或任何首选 SQL 客户端与您的使用者集群一起查询数据。

关联数据共享后，数据共享将变为可用状态。

您还可以随时更改数据共享关联。将关联从特定 AWS 区域和集群命名空间更改到整个 AWS 账户时，Amazon Redshift 会使用 AWS 账户信息覆盖特定区域和集群命名空间信息。然后，AWS 账户中的所有 AWS 区域和集群命名空间即可访问数据共享。

将关联从特定集群命名空间更改为特定 AWS 区域中的所有集群命名空间后，此区域中的所有集群命名空间均可访问数据共享。

从数据使用者中移除数据共享的关联

作为使用者集群管理员，您可以从数据使用者中删除数据共享的关联。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择数据共享。此时将会显示数据共享列表页面。
3. 选择来自其他账户。
4. 在来自其他账户的数据共享部分中，选择要从数据使用者中删除关联的数据共享。
5. 在数据使用者部分中，选择要删除关联的一个或多个数据使用者。然后选择删除关联。
6. 当显示“删除关联”页面时，选择删除关联。

删除关联后，数据使用者将失去对数据共享的访问权限。您可以随时更改数据使用者关联。

拒绝数据共享

作为使用者集群管理员，您可以拒绝任何状态为可用或活动的数据共享。拒绝数据共享后，使用者集群用户会失去对数据共享的访问权限。在您调用 `DescribeDataSharesForConsumer` API 操作时，Amazon Redshift 不会返回被拒绝的数据共享。如果生产者集群管理员运行 `DescribeDataSharesForProducer` API 操作，他们将看到数据共享被拒绝。数据共享被拒绝后，生产者集群管理员可以再次将数据共享授权给使用者集群，而使用者集群管理员可以选择将其 AWS 账户与数据共享关联，也可以拒绝该数据共享。

如果您的 AWS 账户与数据共享关联，并且有一个由 Lake Formation 管理的数据共享关联等待处理，则拒绝由 Lake Formation 管理的数据共享关联也会拒绝原始数据共享。要拒绝特定关联，生产者集群管理员可以从指定数据共享移除授权。此操作不会影响其他数据共享。

要拒绝数据共享，请使用 AWS 控制台、API 操作 `RejectDataShare` 或 AWS CLI 中的 `reject-datashare`。

要使用 AWS 控制台拒绝数据共享，请执行以下操作：

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择数据共享。

3. 选择来自其他账户。
4. 在来自其他账户的数据共享部分中，选择要拒绝的数据共享。当显示拒绝数据共享页面时，请选择拒绝。

拒绝数据共享之后，您无法恢复更改。Amazon Redshift 将从列表中移除数据共享。要再次查看数据共享，生产者管理员必须再次对其进行授权。

管理现有数据共享（预览版）

此文档为预发行文档，介绍了通过 Amazon Redshift 数据共享功能进行多数据仓库写入，该功能在 PREVIEW_2023 版本的公开预览版中提供。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

有关开始使用数据共享的更多信息，请转到共享对数据的写入访问权限（预览版）。

查看数据共享

从 DATASHARES 或 CLUSTERS 选项卡中查看数据共享。

- 使用 DATASHARES 选项卡列出您的账户或其他账户中的数据共享。
  - 要查看在您的账户中创建的数据共享，请选择在我的账户中，然后选择要查看的数据共享。
  - 要查看从其他账户共享的数据共享，请选择来自其他账户，然后选择要查看的数据共享。
- 使用 CLUSTERS 选项卡列出您的集群中或其他集群中的数据共享。

  连接到数据库。有关更多信息，请参阅 连接到数据库（预览版）。

  然后从来自其他集群的数据共享或在我的集群中创建的数据共享部分中选择一个数据共享，以查看其详细信息。

从数据共享中删除数据共享对象

您可以按照以下过程从数据共享中删除一个或多个对象。

要从数据共享中删除一个或多个对象

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。

2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 选择数据共享。
4. 在在我的账户中创建的数据共享部分中，选择连接到数据库。有关更多信息，请参阅 连接到数据库（预览版）。
5. 选择要编辑的数据共享，然后选择编辑。此时将会显示数据共享详细信息页面。
6. 要将一个或多个数据共享对象从数据共享中删除，请执行下列操作之一：
   - 要从数据共享中删除 schema，请选择一个或多个 schema。然后选择删除。Amazon Redshift 会从数据共享中删除指定的 schema 和指定 schema 的所有对象。
   - 要从数据共享中删除表和视图，请选择一个或多个表和视图。然后选择删除。或者，选择按 schema 删除以删除指定 schema 中的所有表和视图。
   - 要从数据共享中删除用户定义的函数，请选择一个或多个用户定义的函数。然后选择删除。或者，选择按 schema 删除以删除指定 schema 中的所有用户定义函数。

从数据共享中删除数据使用者

您可以从数据共享中删除一个或多个数据使用者。数据使用者可以是唯一标识 Amazon Redshift 集群或 AWS 账户的集群命名空间。

从集群命名空间 ID 或 AWS 账户中选择一个或多个数据使用者，然后选择删除。

Amazon Redshift 会从数据共享中删除指定的数据使用者。他们将立即失去对数据共享的访问权限。

编辑在您的账户中创建的数据共享

使用控制台编辑在您的账户中创建的数据共享。首先连接到数据库，以查看在您的账户中创建的数据共享列表。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 选择数据共享。
4. 在在我的账户中创建的数据共享部分中，选择连接到数据库。
5. 选择要编辑的数据共享，然后选择编辑。此时将会显示数据共享详细信息页面。
6. 在数据共享对象或数据使用者部分中进行任何更改。

> **Note**
>
> 如果您选择将数据共享发布到 AWS Glue Data Catalog，则无法编辑配置以将数据共享发布到其他 Amazon Redshift 账户。

7. 选择保存更改。

Amazon Redshift 会使用更改更新您的数据共享。

删除在您的账户中创建的数据共享

使用控制台删除在您的账户中创建的数据共享。首先连接到数据库，以查看在您的账户中创建的数据共享列表。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 选择数据共享。此时将会显示数据共享列表。
4. 在在我的账户中创建的数据共享部分中，选择连接到数据库。
5. 选择要删除的一个或多个数据共享，然后选择删除。此时会显示“删除数据共享”页面。

   删除与 Lake Formation 共享的数据共享不会自动删除 Lake Formation 中的相关权限。要删除权限，请前往 Lake Formation 控制台。
6. 键入 Delete 以确认删除指定的数据共享。
7. 选择 Delete。

删除数据共享后，数据共享使用者将失去对数据共享的访问权限。

查询数据共享（预览版）

以下是通过 PREVIEW_2023 版本中公开预览版 Amazon Redshift 数据共享功能写入多数据仓库的预发行文档。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版服务参与。

有关开始使用数据共享的更多信息，请转到共享对数据的写入访问权限（预览版）。

通过数据共享创建数据库

要开始在数据共享中查询数据，请从数据共享中创建一个数据库。您只能从指定的数据共享中创建一个数据库。

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择集群，然后选择您的集群。此时会显示集群详细信息页面。
3. 选择数据共享。此时将会显示数据共享列表。
4. 在来自其他集群的数据共享部分中，选择连接到数据库。有关更多信息，请参阅 连接到数据库（预览版）。
5. 选择要从中创建数据库的数据共享，然后选择从数据共享中创建数据库。此时将显示“从数据共享创建数据库”页面。
6. 在数据库名称中，指定数据库名称。数据库名称必须为 1-64 个字母数字字符（仅限小写），且不能是保留字。
7. 选择创建。

创建数据库后，您可以查询数据库中的数据或执行写入操作，前提是这些操作已由使用者管理员授予、授权和关联。

在 Amazon Redshift 中摄取和查询半结构化数据

通过使用 Amazon Redshift 中的半结构化数据支持，您可以摄取并将半结构化数据存储在 Amazon Redshift 数据仓库中。通过使用 SUPER 数据类型和 PartiQL 语言，Amazon Redshift 扩展了数据仓库功能，以便与 SQL 和 NoSQL 数据源集成。通过这种方式，Amazon Redshift 可以对关系数据和半结构化存储数据（如 JSON）进行高效分析。

Amazon Redshift 提供两种形式的半结构化数据支持：SUPER 数据类型和 Amazon Redshift Spectrum。

如果需要以低延迟插入或更新小批量的 JSON 数据，请使用 SUPER 数据类型。此外，当查询需要强一致性、可预测的查询性能、复杂的查询支持以及易于使用不断发展的 schema 和无 schema 数据时，请使用 SUPER。

相比之下，如果您的数据查询需要与其他 AWS 服务以及主要存储在 Amazon S3 中以用于归档的数据集成，则将 Amazon Redshift Spectrum 与开放文件格式结合使用。

SUPER 数据类型的使用案例

Amazon Redshift 中使用 SUPER 数据类型的半结构化数据支持可提供卓越的性能、灵活度和易用性。以下使用案例帮助演示如何将半结构化数据支持用于 SUPER。

快速灵活地插入 JSON 数据 – Amazon Redshift 支持可以解析 JSON 并将其存储为 SUPER 值的快速事务。insert 事务的运行速度比在表中执行相同的插入操作速度最高快五倍，这些表将 SUPER 的属性分解为常规列。例如，假设传入的 JSON 的格式为 {“a”:..., “b”:..., “c”“..., ...}。通过将传入的 JSON 存储在带有单个 SUPER 列 S 的表 TJ 中，而不是将其存储到具有列“a”、“b”、“c”等的常规表 TR 中，可以多次提高插入性能。当 JSON 中有数百个属性时，SUPER 数据类型的性能优势就会变得非常明显。

另外，SUPER 数据类型不需要常规 schema。您不需要在存储传入的 JSON 之前对其自检和清理。例如，假设传入的 JSON 具有字符串“c”属性，其他属性具有整数“c”属性，但没有 SUPER 数据类型。在这种情况下，您必须分隔 c_string 和 c_int 列或清理数据。相比之下，使用 SUPER 数据类型，所有 JSON 数据都在摄入期间存储，而不会丢失信息。稍后，您可以使用 SQL 的 PartiQL 扩展来分析信息。

用于发现的灵活查询 – 将半结构化数据（如 JSON）存储到 SUPER 数据值后，您可以在不强加 schema 的情况下查询它。您可以使用 PartiQL 动态键入和宽松语义来运行查询并发现所需的深度嵌套数据，而无需在查询之前强加 schema。

对常规实体化视图的提取、加载、转换 (ETL) 操作的灵活查询 – 将无 schema 和半结构化数据存储到 SUPER 中后，您可以使用 PartiQL 实体化视图对数据进行自检并将其分解为实体化视图。

带有分解数据的实体化视图是经典分析案例的性能和可用性优势的一个很好的示例。当您对分解数据执行分析时，Amazon Redshift 实体化视图的列式组织可提供更好的性能。此外，需要对提取的数据使用常规 schema 的用户和业务情报 (BI) 工具可以使用视图（具体化或虚拟）作为数据的常规 schema 表示。

在 PartiQL 实体化视图将在 JSON 或 SUPER 中找到的数据提取到传统的列式实体化视图之后，您可以查询这些实体化视图。有关 SUPER 数据类型如何用于实体化视图的更多信息，请参阅将 SUPER 数据类型用于具体化视图。

您可以将动态数据掩蔽策略应用于 SUPER 类型列路径上的 `scalar` 值。有关动态数据掩蔽的更多信息，请参阅动态数据掩蔽。有关为 SUPER 数据类型使用动态数据掩蔽的信息，请参阅对 SUPER 数据类型路径使用动态数据掩蔽。（预览版）

有关 SUPER 数据类型的信息，请参阅SUPER 类型。

有关使用 SUPER 数据类型的示例，请参阅本主题的小节，从SUPER sample 数据集开始。

SUPER 数据类型使用的概念

接下来，您可以找到一些 Amazon Redshift Super 数据类型概念。

了解 Amazon Redshift 中的 SUPER 数据类型是什么 – SUPER 数据类型是一种 Amazon Redshift 数据类型，允许存储包含 Amazon Redshift 标量以及可能的嵌套数组和结构的无 schemal 数组和结构。SUPER 数据类型可以本地存储不同格式的半结构化数据，例如 JSON 或源自面向文档源的数据。您可以添加一个新的 SUPER 列来存储半结构化数据，并编写访问 SUPER 列的查询以及通常的标量列。有关 SUPER 数据类型的更多信息，请参阅SUPER 类型。

将无 schema JSON 摄取到 SUPER – 通过灵活的半结构化 SUPER 数据类型，Amazon Redshift 可以接收并将无 schema JSON 摄取到 SUPER 值中。例如，Amazon Redshift 可以将 JSON 值 [10.5, "first"] 摄取到一个 SUPER 值 [10.5, ‘first’] 中，该值是一个包含 Amazon Redshift 十进制 10.5 和 varchar“first”的数组。Amazon Redshift 可以使用 COPY 命令或 JSON 解析函数将 JSON 摄取到 SUPER 值中，例如 json_parse('[10.5, "first"]')。COPY 和 `json_parse` 在预设情况下均使用严格的解析语义摄取 JSON。您还可以使用数据库数据本身构建包括数组和结构在内的 SUPER 值。

在摄取无 schema JSON 的不规则结构时，SUPER 列不需要 schema 修改。例如，在分析点击流时，您最初在属性为“IP” 和“time”的 SUPER 列中存储“click”结构。您可以添加属性“customer id”，而无需更改 schema，以便摄取此类更改。

用于 SUPER 数据类型的本机格式是一种二进制格式，需要的空间小于其文本形式的 JSON 值。这可以在查询时更快地摄入和对 SUPER 值进行运行时处理。

使用 PartiQL 查询 SUPER 数据— PartiQL 是一个向后兼容的 SQL-92 扩展，许多 AWS 目前都在使用它。通过使用 PartiQL，熟悉的 SQL 结构可以将对经典的表格式 SQL 数据和 SUPER 的半结构化数据的访问无缝结合在一起。您可以执行对象和数组导航以及非嵌套数组。PartiQL 扩展了标准 SQL 语言，以声明方式表达和处理嵌套和多值数据。

PartiQL 是 SQL 的扩展，其中 SUPER 列的嵌套和无 schema 数据是一等公民。PartiQL 不要求在查询编译期间对所有查询表达式进行类型检查。当访问 SUPER 列内的实际数据类型时，此方法使包含 SUPER 数据类型的查询表达式能够在查询执行期间动态键入。此外，PartiQL 在宽松模式下运行，其中类型不一致不会导致失败，而是会返回 null。无 schema 和松宽松查询处理的结合使得 PartiQL 非常适合提取、加载、传输 (ELT) 应用程序，在这些应用程序中，SQL 查询可以评估在 SUPER 列中摄取的 JSON 数据。

与 Redshift Spectrum 集成 – 通过 JSON、Parquet 和其他具有嵌套数据的格式运行 Redshift Spectrum 查询时，Amazon Redshift 支持 PartiQL 的多个方面。Redshift Spectrum 仅支持具有 schema 的嵌套数据。例如，使用 Redshift Spectrum，您可以声明您的 JSON 数据在 schema ARRAY<STRUCT<a:INTEGER, b:DECIMAL(5,2)>> 中具有属性 nested_schemaful_example。此属性的 schema 确定数据始终包含一个数组，该数组包含一个带有整数 a 和小数 b 的结构。如果数据更改为包含更多属性，则类型也会发生变化。相比之下，SUPER 数据类型不需要 schema。您可以存储具有不同属性或类型的结构元素的数组。另外，一些值可以存储在数组之外。

有关支持 SUPER 数据类型的函数的信息，请参阅以下内容：

- ABS 函数
- CEILING（或 CEIL）函数
- FLOOR 函数
- ROUND 函数
- SIGN 函数
- TRUNC 函数

SUPER 数据的注意事项

在使用 SUPER 数据时，请考虑以下事项：

- 使用 JDBC 驱动程序版本 1.2.50、ODBC 驱动程序版本 1.4.17 或更高版本以及 Amazon Redshift Python 驱动程序版本 2.0.872 或更高版本。

有关 JDBC 驱动程序的信息，请参阅配置 JDBC 连接。

有关 ODBC 驱动程序的信息，请参阅配置 ODBC 连接。

- 通过 SUPER sample 数据集 查找以下主题中使用的 schema 示例。
- 以下主题中使用的所有 SQL 代码示例都包含用于下载的相同 S3 前缀。其中包括数据定义语言 (DDL) 和 COPY 语句，以及某些使用 SUPER 的 TPC-H 修改后查询。

  要查看或下载 SQL 文件，请执行下列操作之一：

  - 下载 SUPER 教程 SQL 文件和 TPC-H 文件。
  - 使用 Amazon S3 CLI，运行以下命令。您可以使用自己的目标路径。

```
aws s3 cp s3://redshift-downloads/semistructured/tutorialscripts/semistructured-
tutorial.sql /target/path
aws s3 cp s3://redshift-downloads/semistructured/tutorialscripts/
super_tpch_queries.sql /target/path
```

有关 SUPER 配置的更多信息，请参阅SUPER 配置。

SUPER sample 数据集

用于提取和查询示例的表 schema 和数据模型定义如下。

```
/*customer-orders-lineitem*/
CREATE TABLE customer_orders_lineitem
(c_custkey bigint
,c_name varchar
,c_address varchar
,c_nationkey smallint
,c_phone varchar
,c_acctbal decimal(12,2)
,c_mktsegment varchar
,c_comment varchar
,c_orders super
);

/* Datamodel of documents to be stored in c_orders Super column would be as follows*/
ARRAY < STRUCT < o_orderkey:bigint
                          ,o_orderstatus:string
                          ,o_totalprice:double
```

```
                            ,o_orderdate:string
                            ,o_orderpriority:string
                            ,o_clerk:string
                            ,o_shippriority:int
                            ,o_comment:string
                            ,o_lineitems:ARRAY < STRUCT < l_partkey:bigint
                                                        ,l_suppkey:bigint
                                                        ,l_linenumber:int
                                                        ,l_quantity:double
                                                        ,l_extendedprice:double
                                                        ,l_discount:double
                                                        ,l_tax:double
                                                        ,l_returnflag:string
                                                        ,l_linestatus:string
                                                        ,l_shipdate:string
                                                        ,l_commitdate:string
                                                        ,l_receiptdate:string
                                                        ,l_shipinstruct:string
                                                        ,l_shipmode:string
                                                        ,l_comment:string
                                                        > >
                             > >

/*part*/
CREATE TABLE part
(
 p_partkey bigint
 ,p_name varchar
 ,p_mfgr varchar
 ,p_brand varchar
 ,p_type varchar
 ,p_size int
 ,p_container varchar
 ,p_retailprice decimal(12,2)
 ,p_comment varchar
 );

/*region-nations*/
CREATE TABLE region_nations
(
 r_regionkey smallint
 ,r_name varchar
 ,r_comment varchar
 ,r_nations super
```

```
 );

 /* Datamodel of documents to be stored in r_nations Super column would be as follows*/
 ARRAY < STRUCT < n_nationkey:int,n_name:string,n_comment:string > >

/*supplier-partsupp*/
CREATE TABLE supplier_partsupp
(
 s_suppkey bigint
 ,s_name varchar
 ,s_address varchar
 ,s_nationkey smallint
 ,s_phone varchar
 ,s_acctbal double precision
 ,s_comment varchar
 ,s_partsupps super
 );

 /* Datamodel of documents to be stored in s_partsupps Super column would be as
 follows*/
 ARRAY < STRUCT <
 ps_partkey:bigint,ps_availqty:int,ps_supplycost:double,ps_comment:string > >
```

将半结构化数据加载到 Amazon Redshift

使用 SUPER 数据类型在 Amazon Redshift 中保留和查询分层数据和通用数据。Amazon Redshift 引入 `json_parse` 函数来解析 JSON 格式的数据并将其转换为 SUPER 表示形式。Amazon Redshift 还支持使用 COPY 命令加载 SUPER 列。受支持的文件格式包括 JSON、Avro、文本、逗号分隔值 (CSV) 格式、Parquet 和 ORC。

有关以下示例中使用的表的信息，请参阅SUPER sample 数据集。

有关 `json_parse` 函数的信息，请参阅JSON_PARSE 函数。

SUPER 数据类型的默认编码是 ZSTD。

将 JSON 文档解析为 SUPER 列

您可以使用 `json_parse` 函数将 JSON 数据插入或更新到 SUPER 列中。该函数以 JSON 格式解析数据，并将其转换为 SUPER 数据类型，您可以在 INSERT 或 UPDATE 语句中使用该数据类型。

以下示例将 JSON 数据插入到 SUPER 列中。如果 json_parse 函数在查询中缺失，Amazon Redshift 将该值视为单个字符串，而不是必须解析的 JSON 格式的字符串。

如果您更新了 SUPER 数据列，Amazon Redshift 会要求将完整的文档传递给列值。Amazon Redshift 不支持部分更新。

```
INSERT INTO region_nations VALUES(0,
   'lar deposits. blithely final packages cajole. regular waters are final requests.
 regular accounts are according to',
   'AFRICA',
   JSON_PARSE('{"r_nations":[
      {"n_comment":" haggle. carefully final deposits detect slyly agai",
         "n_nationkey":0,
         "n_name":"ALGERIA"
      },
      {"n_comment":"ven packages wake quickly. regu",
         "n_nationkey":5,
         "n_name":"ETHIOPIA"
      },
      {"n_comment":" pending excuses haggle furiously deposits. pending, express pinto
 beans wake fluffily past t",
         "n_nationkey":14,
         "n_name":"KENYA"
      },
      {"n_comment":"rns. blithely bold courts among the closely regular packages use
 furiously bold platelets?",
         "n_nationkey":15,
         "n_name":"MOROCCO"
      },
      {"n_comment":"s. ironic, unusual asymptotes wake blithely r",
         "n_nationkey":16,
         "n_name":"MOZAMBIQUE"
      }
   ]
}'));
```

使用 COPY 在 Amazon Redshift 中加载 SUPER 列

在以下部分中，您可以了解使用 COPY 命令将 JSON 数据加载到 Amazon Redshift 的不同方法。

从 JSON 和 Avro 复制数据

通过在 Amazon Redshift 中使用半结构化数据支持，您可以加载 JSON 文档，而无需将 JSON 结构的属性分解为多列。

Amazon Redshift 提供了两种使用 COPY 摄取 JSON 文档的方法，即使是完全或部分未知的 JSON 结构：

1. 使用 noshred 选项将从 JSON 文档派生的数据存储到单个 SUPER 数据列中。当 schema 未知或预计将更改时，此方法非常有用。因此，此方法可以更容易地将整个元组存储在单个 SUPER 列中。
2. 使用 auto 或 jsonpaths 选项将 JSON 文档分解为多个 Amazon Redshift 列。属性可以是 Amazon Redshift 标量或 SUPER 值。

您可以将这些选项与 JSON 或 Avro 格式一起使用。

分解前的 JSON 对象的最大大小为 4 MB。

将 JSON 文档复制到单个 SUPER 数据列

要将 JSON 文档复制到单个 SUPER 数据列中，请创建包含单个 SUPER 数据列的表。

```
CREATE TABLE region_nations_noshred (rdata SUPER);
```

将 Amazon S3 中的数据复制到单个 SUPER 数据列中。要将 JSON 源数据摄取到单个 SUPER 数据列中，请在 FORMAT JSON 子句中指定 noshred 选项。

```
COPY region_nations_noshred FROM 's3://redshift-downloads/semistructured/tpch-nested/
data/json/region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'noshred';
```

在 COPY 成功摄取 JSON 后，您的表具有包含整个 JSON 对象数据的 rdata SUPER 数据列。摄取的数据将维护 JSON 层次结构的所有属性。但是，叶子会转换为 Amazon Redshift 标量类型，以实现高效的查询处理。

使用以下查询检索原始 JSON 字符串。

```
SELECT rdata FROM region_nations_noshred;
```

当 Amazon Redshift 生成一个 SUPER 数据列时，它可以通过 JSON 序列化使用 JDBC 作为字符串进行访问。有关更多信息，请参阅 序列化复杂嵌套 JSON。

将 JSON 文档复制到多个 SUPER 数据列

您可以将 JSON 文档拆分为多个列，这些列可以是 SUPER 数据列或 Amazon Redshift 标量类型。Amazon Redshift 会将 JSON 对象的不同部分分布到不同的列。

```
CREATE TABLE region_nations
(
 r_regionkey smallint
 ,r_name varchar
 ,r_comment varchar
 ,r_nations super
 );
```

要将上一个示例的数据复制到表中，请在 FORMAT JSON 子句中指定 AUTO 选项，以将 JSON 值拆分到多个列中。COPY 将顶级 JSON 属性与列名匹配，并允许嵌套值作为 SUPER 值（如 JSON 数组和对象）摄取。

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'auto';
```

当 JSON 属性名称混合使用大小写时，请在 FORMAT JSON 子句中指定 `auto ignorecase` 选项。有关 COPY 命令的更多信息，请参阅使用“auto ignorecase”选项从 JSON 数据中加载。

在某些情况下，列名和 JSON 属性之间存在不匹配的情况，或者要加载的属性嵌套超过一层深度。如果是这样，请使用 `jsonpaths` 文件将 JSON 属性手动映射到 Amazon Redshift 列。

```
CREATE TABLE nations
(
 regionkey smallint
 ,name varchar
 ,comment super
 ,nations super
 );
```

假设您想要将数据加载到列名与 JSON 属性不匹配的表中。在下面的示例中，nations 表就是这样一个表。您可以创建一个 jsonpaths 文件，该文件将属性路径按其在 jsonpaths 数组中的位置映射到表列中。

```
{"jsonpaths": [
       "$.r_regionkey",
       "$.r_name",
       "$.r_comment",
       "$.r_nations
    ]
}
```

jsonpaths 文件的位置用作 FORMAT JSON 的参数。

```
COPY nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 's3://redshift-downloads/semistructured/tpch-nested/data/jsonpaths/
nations_jsonpaths.json';
```

使用以下查询可访问显示传播到多个列的数据的表。使用 JSON 格式打印 SUPER 数据列。

```
SELECT r_regionkey,r_name,r_comment,r_nations[0].n_nationkey FROM region_nations ORDER
 BY 1,2,3 LIMIT 1;
```

Jsonpath 文件将 JSON 文档中的字段映射到表格列。您可以提取其它列，例如分配键和排序键，同时仍将完整文档作为 SUPER 列加载。以下查询将完整文档加载到国家/地区列。name 列为排序键，而 regionkey 列为分配键。

```
CREATE TABLE nations_sorted (
    regionkey smallint,
    name varchar,
    nations super
) DISTKEY(regionkey) SORTKEY(name);
```

根 jsonpath "$" 映射到文档的根目录，如下所示：

```
{"jsonpaths": [
       "$.r_regionkey",
       "$.r_name",
```

```
        "$"
    ]
}
```

jsonpaths 文件的位置用作 FORMAT JSON 的参数。

```
COPY nations_sorted FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 's3://redshift-downloads/semistructured/tpch-nested/data/jsonpaths/
nations_sorted_jsonpaths.json';
```

从文本和 CSV 复制数据

Amazon Redshift 将文本和 CSV 格式的 SUPER 列表示为序列化 JSON。要使用正确的类型信息加载 SUPER 列，则需要有效的 JSON 格式。取消引用对象、数组、数字、布尔值和空值。用双引号将字符串值括起来。SUPER 列对文本和 CSV 格式使用标准转义规则。对于 CSV 格式，分隔符将根据 CSV 标准进行转义。对于文本格式，如果选定的分隔符也可能出现在 SUPER 字段中，请在 COPY 和 UNLOAD 期间使用 ESCAPE 选项。

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/csv/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT CSV;
```

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/text/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
DELIMITER ','
ESCAPE;
```

从列格式的 Parquet 和 ORC 中复制数据

如果您的半结构化或嵌套数据已经以 Apache Parquet 或 Apache ORC 格式提供，则可以使用 COPY 命令将数据摄取到 Amazon Redshift 中。

Amazon Redshift 表格结构应与 Parquet 或 ORC 文件的列数和列数据类型相匹配。通过在 COPY 命令中指定 SERIALIZETOJSON，您可以将文件中与表中的 SUPER 列对齐的任何列类型加载为 SUPER。这包括结构和数组类型。

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/
parquet/region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT PARQUET SERIALIZETOJSON;
```

以下示例使用 ORC 格式。

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/orc/
region_nation'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT ORC SERIALIZETOJSON;
```

当日期或时间数据类型的属性位于 ORC 中时，Amazon Redshift 会在 SUPER 中对它们进行编码时将其转换为 varchar。

卸载半结构化数据

您可以以不同格式将包含 SUPER 数据列的表卸载到 Amazon S3。

主题

- 以 CSV 或文本格式卸载半结构化数据
- 以 Parquet 格式卸载半结构化数据

以 CSV 或文本格式卸载半结构化数据

您可以将带有 SUPER 数据列的表以逗号分隔值 (CSV) 或文本格式卸载到 Amazon S3 中。Amazon Redshift 使用导航和非嵌套子句的组合，以 CSV 或文本格式将 SUPER 数据格式的分层数据卸载到 Amazon S3。随后，您可以根据已卸载的数据创建外部表，并使用 Redshift Spectrum 对其进行查询。有关使用 UNLOAD 和所需的 IAM 权限的信息，请参阅UNLOAD。

在运行下面的示例之前，请使用 将半结构化数据加载到 Amazon Redshift 中的进程填充 region_nations 表。有关以下示例中使用的表的信息，请参阅SUPER sample 数据集。

以下示例将数据卸载到 Amazon S3 中。

```
UNLOAD ('SELECT * FROM region_nations')
TO 's3://xxxxxx/'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3-Write'
DELIMITER AS '|'
```

```
GZIP
ALLOWOVERWRITE;
```

与用户定义的字符串表示 null 值的其他数据类型不同，Amazon Redshift 使用 JSON 格式导出 SUPER 数据列，并根据 JSON 格式将其表示为 null。因此，SUPER 数据列会忽略 UNLOAD 命令中使用的 NULL [AS] 选项。

以 Parquet 格式卸载半结构化数据

您可以将包含 SUPER 数据列的表以 Parquet 格式卸载到 Amazon S3。Amazon Redshift 将 Parquet 格式的超级列表示为 JSON 数据类型。这使得半结构化数据可以以 Parquet 格式表示。您可以使用 Redshift Spectrum 查询这些列，或使用 COPY 命令将它们提取回 Amazon Redshift。有关使用 UNLOAD 和所需的 IAM 权限的信息，请参阅UNLOAD。

以下示例以 Parquet 格式将数据卸载到 Amazon S3。

```
UNLOAD ('SELECT * FROM region_nations')
TO 's3://xxxxxx/'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3-Write'
FORMAT PARQUET;
```

查询半结构化数据

Amazon Redshift 使用 PartiQL 语言提供对关系数据、半结构化数据和嵌套数据的 SQL 兼容访问。

PartiQL 使用动态类型进行操作。这种方法可以对结构化、半结构化和嵌套数据集的组合进行直观的筛选、联接和聚合。在访问嵌套数据时，PartiQL 语法使用点记法和数组下标进行路径导航。它还使 FROM 子句项能够对数组进行迭代并用于非嵌套操作。以下内容介绍了将 SUPER 数据类型的使用与路径和数组导航、取消嵌套、逆透视转换和联接相结合的不同查询模式。

有关以下示例中使用的表的信息，请参阅SUPER sample 数据集。

导航

Amazon Redshift 使用 PartiQL 分别通过 [...] 括号和点符号来支持对数组和结构的导航。此外，您还可以使用点记法将导航混合到结构中，使用括号符号将数组混合到结构中。例如，以下示例假定 `c_orders` SUPER 数据列是一个具有结构的数组，并且属性名为 `o_orderkey`。

要提取 `customer_orders_lineitem` 表中的数据，请运行以下命令。将 IAM 角色替换为您自己的凭证。

```
COPY customer_orders_lineitem FROM 's3://redshift-downloads/semistructured/tpch-nested/
data/json/customer_orders_lineitem'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'auto';

SELECT c_orders[0].o_orderkey FROM customer_orders_lineitem;
```

Amazon Redshift 还使用表别名作为表示法的前缀。以下示例是与上一个示例相同的查询。

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

您可以在所有类型的查询中使用点和括号符号，例如筛选、联接和聚合。您可以在通常存在列引用的查询中使用这些符号。以下示例使用筛选结果的 SELECT 语句。

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0]. o_orderkey IS NOT
 NULL;
```

以下示例在 GROUP BY 和 ORDER BY 子句中使用括号和点导航：

```
SELECT c_orders[0].o_orderdate,
       c_orders[0].o_orderstatus,
       count(*)
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderkey IS NOT NULL
GROUP BY c_orders[0].o_orderstatus,
         c_orders[0].o_orderdate
ORDER BY c_orders[0].o_orderdate;
```

取消嵌套查询

为了取消嵌套查询，Amazon Redshift 使用 PartiQL 语法迭代 SUPER 数组。它通过使用查询的 FROM 子句导航数组来实现这一点。使用前面的示例，以下示例对 c_orders 的属性值进行迭代。

```
SELECT c.*, o FROM customer_orders_lineitem c, c.c_orders o;
```

取消嵌套语法是 FROM 子句的扩展。在标准 SQL 中，FROM 子句 x （AS） y 表示 y 迭代关系 x 中的每个元组。在这种情况下，x 指的是关系，而 y 指的是关系 x 的别名。同样，使用 FROM 子句项 x （AS） y 进行取消嵌套的 PartiQL 语法表示 y 迭代（SUPER）数组表达式 x 中的每个（SUPER）值。在这种情况下，x 是一个 SUPER 表达式，而 y 是 x 的别名。

左侧操作数也可以使用点和括号表示法进行常规导航。在上一个示例中，customer_orders_lineitem c 是对 customer_order_lineitem 基表的迭代，c.c_orders o 是对 c.c_orders 数组的迭代。要迭代作为数组中的数组的 o_lineitems 属性，必须添加多个子句。

```
SELECT c.*, o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

Amazon Redshift 还在使用 AT 关键字迭代数组时支持数组索引。子句 x AS y AT z 迭代数组 x 并生成字段 z，，即数组索引。以下示例演示数组索引的工作原理：

```
SELECT c_name,
       orders.o_orderkey AS orderkey,
       index AS orderkey_index
FROM customer_orders_lineitem c, c.c_orders AS orders AT index
ORDER BY orderkey_index;

c_name             | orderkey | orderkey_index
-------------------+----------+----------------
Customer#000008251 | 3020007  |        0
Customer#000009452 | 4043971  |        0
  (2 rows)
```

以下示例对标量数组进行迭代：

```
CREATE TABLE bar AS SELECT json_parse('{"scalar_array": [1, 2.3, 45000000]}') AS data;

SELECT index, element FROM bar AS b, b.data.scalar_array AS element AT index;

 index | element
-------+----------
     0 | 1
     1 | 2.3
     2 | 45000000
(3 rows)
```

以下示例对多个级别的数组进行迭代。该示例使用多个 unnest 子句来迭代到最内层的数组。f.multi_level_array AS 数组迭代 multi_level_array。数组 AS 元素是对 multi_level_array 中的数组的迭代。

```
CREATE TABLE foo AS SELECT json_parse('[[1.1, 1.2], [2.1, 2.2], [3.1, 3.2]]') AS
 multi_level_array;
```

```
SELECT array, element FROM foo AS f, f.multi_level_array AS array, array AS element;

   array   | element
-----------+---------
 [1.1,1.2] | 1.1
 [1.1,1.2] | 1.2
 [2.1,2.2] | 2.1
 [2.1,2.2] | 2.2
 [3.1,3.2] | 3.1
 [3.1,3.2] | 3.2
(6 rows)
```

有关 FROM 子句的更多信息，请参阅FROM 子句。

对象逆透视

为执行逆透视，Amazon Redshift 使用 PartiQL 语法迭代 SUPER 对象。它使用带有 UNPIVOT 关键字的查询的 FROM 子句来执行此操作。在这种情况下，表达式是 `c.c_orders[0]` 对象。示例查询会遍历此对象返回的每个属性。

```
SELECT attr as attribute_name, json_typeof(val) as value_type
FROM customer_orders_lineitem c, UNPIVOT c.c_orders[0] AS val AT attr
WHERE c_custkey = 9451;

 attribute_name  | value_type
-----------------+------------
 o_orderstatus   | string
 o_clerk         | string
 o_lineitems     | array
 o_orderdate     | string
 o_shippriority  | number
 o_totalprice    | number
 o_orderkey      | number
 o_comment       | string
 o_orderpriority | string
(9 rows)
```

与取消嵌套一样，逆透视语法也是 FROM 子句的扩展。不同之处在于，逆透视的语法使用 UNPIVOT 关键字来表示它正在迭代对象而不是数组。它使用 AS `value_alias` 迭代对象内的所有值并使用 AT `attribute_alias` 迭代所有属性。请考虑以下语法片段：

```
UNPIVOT expression AS value_alias [ AT attribute_alias ]
```

Amazon Redshift 支持在单个 FROM 子句中使用对象反转置和数组取消嵌套，如下所示：

```
SELECT attr as attribute_name, val as object_value
FROM customer_orders_lineitem c, c.c_orders AS o, UNPIVOT o AS val AT attr
WHERE c_custkey = 9451;
```

当您使用对象逆透视时，Amazon Redshift 不支持关联的逆透视。具体来说，假设您有一个案例，其中在不同查询级别有多个逆透视示例，并且内部逆透视引用了外部逆透视。Amazon Redshift 不支持此类多重逆透视。

有关 FROM 子句的更多信息，请参阅FROM 子句。有关演示如何使用 PIVOT 和 UNPIVOT 查询结构化数据的示例，请参阅 PIVOT 和 UNPIVOT 示例。

动态键入

动态键入不需要显式转换从点和括号路径中提取的数据。Amazon Redshift 使用动态键入处理无 schemal SUPER 数据，无需在查询中使用数据类型之前声明数据类型。动态键入使用导航到 SUPER 数据列的结果，而无需将其显式转换为 Amazon Redshift 类型。动态键入在联接和 GROUP BY 子句中最有用。以下示例使用 SELECT 语句，该语句不需要将点和括号表达式显式转换为常见的 Amazon Redshift 类型。有关类型兼容性和转换的信息，请参阅类型兼容性和转换。

```
SELECT c_orders[0].o_orderkey
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderstatus = 'P';
```

当 c_orders[0].o_orderstatus 为字符串“P”时，此查询中的等号计算为 `true`。在所有其他情况下，等号计算为 `false`，包括等式参数为不同类型的情况。

动态和静态键入

如果不使用动态键入，则无法确定 c_orders[0].o_orderstatus 是字符串、整数还是结构。您只能确定 c_orders[0].o_orderstatus 是 SUPER 数据类型，它可以是 Amazon Redshift 标量、数组或结构。c_orders[0].o_orderstatus 的静态类型是 SUPER 数据类型。传统上，类型在 SQL 中是隐式的静态类型。

Amazon Redshift 使用动态键入来处理无 schema 数据。当查询计算数据时，c_orders[0].o_orderstatus 是一种特定的类型。例如，在 customer_orders_lineitem 的第一条记录

上评估 c_orders[0].o_orderstatus 可能会导致一个整数。对第二条记录进行评估可能会导致字符串。它们都是表达式的动态类型。

当将 SQL 运算符或函数与具有动态类型的点和括号表达式一起使用时，Amazon Redshift 生成的结果类似于将标准 SQL 运算符或的函数与相应的静态类型结合使用。在此示例中，当路径表达式的动态类型为字符串时，与字符串“P”进行比较是有意义的。只要 c_orders[0].o_orderstatus 的动态类型是除字符串外的任何其他数据类型，相等性都返回 false。当使用键入错误的参数时，其他函数返回 null。

以下示例使用静态键入编写上一个查询：

```
SELECT c_custkey
FROM customer_orders_lineitem
WHERE CASE WHEN JSON_TYPEOF(c_orders[0].o_orderstatus) = 'string'
           THEN c_orders[0].o_orderstatus::VARCHAR = 'P'
           ELSE FALSE END;
```

请注意，相等谓词和比较谓词之间存在以下区别。在上一个示例中，如果用小于或等于谓词替换相等谓词，则语义生成 null 而不是 false。

```
SELECT c_orders[0]. o_orderkey
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderstatus <= 'P';
```

在此示例中，如果 c_orders[0].o_orderstatus 是一个字符串，则如果它的字母顺序等于或小于“P”，Amazon Redshift 返回 true。如果 Amazon Redshift 按字母顺序大于“P”，则返回 false。但是，如果 c_orders[0].o_orderstatus 不是字符串，则 Amazon Redshift 会返回 null，因为 Amazon Redshift 无法比较不同类型的值，如以下查询所示：

```
SELECT c_custkey
FROM customer_orders_lineitem
WHERE CASE WHEN JSON_TYPEOF(c_orders[0].o_orderstatus) = 'string'
           THEN c_orders[0].o_orderstatus::VARCHAR <= 'P'
           ELSE NULL END;
```

动态键入并不排除具有最低可比性的类型的比较。例如，您可以将 CHAR 和 VARCHAR Amazon Redshift 标量类型转换为 SUPER。它们与字符串类似，包括忽略类似于 Amazon Redshift CHAR 和 VARCHAR 类型的尾随空格字符。同样地，整数、小数和浮点值可与 SUPER 值进行比较。特别是对于小数列，每个值也可以具有不同的小数位数。Amazon Redshift 仍将它们视为动态类型。

Amazon Redshift 还支持对深度相等的对象和数组进行相等运算，例如深入评估对象或数组以及比较所有属性。小心使用深度相等计算，因为执行深度相等的过程可能很耗时。

对联接使用动态键入

对于联接，动态键入会自动匹配值与不同的动态类型，而无需执行长 CASE WHEN 分析以找出可能显示的数据类型。例如，假定您的组织随着时间的推移更改了它用于部分键的格式。

发出的初始整数部分键被字符串部分键（如“A55”）替换，然后再次替换为数组部分键，例如字符串和数字组合形成的 [‘X’, 10]。Amazon Redshift 不必对部分键执行冗长的案例分析，并且可以如以下示例所示使用联接。

```
SELECT c.c_name
    ,l.l_extendedprice
    ,l.l_discount
FROM customer_orders_lineitem c
    ,c.c_orders o
    ,o.o_lineitems l
    ,supplier_partsupp s
    ,s.s_partsupps ps
WHERE l.l_partkey = ps.ps_partkey
AND c.c_nationkey = s.s_nationkey
ORDER BY c.c_name;
```

下面的示例显示了，如果不使用动态键入，同一个查询会多么复杂和低效：

```
SELECT c.c_name
    ,l.l_extendedprice
    ,l.l_discount
FROM customer_orders_lineitem c
    ,c.c_orders o
    ,o.o_lineitems l
    ,supplier_partsupp s
    ,s.s_partsupps ps
WHERE CASE WHEN IS_INTEGER(l.l_partkey) AND IS_INTEGER(ps.ps_partkey)
           THEN l.l_partkey::integer = ps.ps_partkey::integer
           WHEN IS_VARCHAR(l.l_partkey) AND IS_VARCHAR(ps.ps_partkey)
           THEN l.l_partkey::varchar = ps.ps_partkey::varchar
           WHEN IS_ARRAY(l.l_partkey) AND IS_ARRAY(ps.ps_partkey)
                AND IS_VARCHAR(l.l_partkey[0]) AND IS_VARCHAR(ps.ps_partkey[0])
                AND IS_INTEGER(l.l_partkey[1]) AND IS_INTEGER(ps.ps_partkey[1])
           THEN l.l_partkey[0]::varchar = ps.ps_partkey[0]::varchar
```

```
                AND l.l_partkey[1]::integer = ps.ps_partkey[1]::integer
           ELSE FALSE END
AND c.c_nationkey = s.s_nationkey
ORDER BY c.c_name;
```

宽松语义

预设情况下，在导航无效时，SUPER 值的导航操作返回 null，而不是返回错误。如果 SUPER 值不是对象，或者如果 SUPER 值是一个对象，但不包含查询中使用的属性名称，则对象导航无效。例如，以下查询访问 SUPER 数据列 cdata 中的无效属性名称：

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

如果 SUPER 值不是数组或数组索引超出界限，则数组导航返回 null。以下查询返回 null，因为 c_orders[1][1] 超出了界限。

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

在使用动态键入转换 SUPER 值时，宽松语义特别有用。如果转换无效，将 SUPER 值转换为错误的类型将返回 null 而不是错误。例如，以下查询返回 null，因为它不能将对象属性 o_orderstatus 的字符串值“Good”转换为 INTEGER。Amazon Redshift 针对 VARCHAR 到 INTEGER 的转换返回错误，但不针对 SUPER 转换返回错误。

```
SELECT c.c_orders.o_orderstatus::integer FROM customer_orders_lineitem c;
```

自检类型

SUPER 数据列支持返回有关 SUPER 值的动态类型和其他类型信息的检查函数。最常见的示例是返回具有布尔值、数字、字符串、对象、数组或 null 的 VARCHAR 的 JSON_TYPEOF 标量函数，具体取决于 SUPER 值的动态类型。Amazon Redshift 支持以下针对 SUPER 数据列的布尔函数：

- DECIMAL_PRECISION
- DECIMAL_SCALE
- IS_ARRAY
- IS_BIGINT
- IS_CHAR
- IS_DECIMAL

- IS_FLOAT
- IS_INTEGER
- IS_OBJECT
- IS_SCALAR
- IS_SMALLINT
- IS_VARCHAR
- JSON_TYPEOF

如果输入值为 null，所有这些函数都返回 false。IS_SCALAR、IS_OBJECT 和 IS_ARRAY 是相互排斥的，涵盖除 null 之外的所有可能的值。

要推理与数据对应的类型，Amazon Redshift 使用 JSON_TYPEOF 函数，该函数返回 SUPER 值的类型（顶级），如以下示例所示：

```
SELECT JSON_TYPEOF(r_nations) FROM region_nations;
 json_typeof
 -------------
 array
(1 row)
```

```
SELECT JSON_TYPEOF(r_nations[0].n_nationkey) FROM region_nations;
 json_typeof
 -------------
 number
```

Amazon Redshift 将此字符串视为单个长字符串，类似于将此值插入 VARCHAR 列而不是 SUPER。由于该列是 SUPER，因此单个字符串仍然是一个有效的 SUPER 值，且差异体现在 JSON_TYPEOF 中：

```
SELECT IS_VARCHAR(r_nations[0].n_name) FROM region_nations;
 is_varchar
-------------
 true
(1 row)
```

```
SELECT r_nations[4].n_name FROM region_nations
WHERE CASE WHEN IS_INTEGER(r_nations[4].n_nationkey)
           THEN r_nations[4].n_nationkey::INTEGER = 15
```

```
            ELSE false END;
```

Order by（排序依据）

Amazon Redshift 不会定义具有不同动态类型的值之间的 SUPER 比较。作为字符串的 SUPER 值既不小于也不大于作为数字的 SUPER 值。要将 ORDER BY 子句与 SUPER 列一起使用，Amazon Redshift 定义了在 Amazon Redshift 使用 ORDER BY 子句对 SUPER 值进行排名时，需要观察的不同类型的总排序。动态类型之间的顺序是布尔值、数字、字符串、数组、对象。以下示例显示不同类型的顺序：

```
INSERT INTO region_nations VALUES
(100,'name1','comment1','AWS'),
(200,'name2','comment2',1),
(300,'name3','comment3',ARRAY(1, 'abc', null)),
(400,'name4','comment4',-2.5),
(500,'name5','comment5','Amazon');

SELECT r_nations FROM region_nations order by r_nations;

r_nations
----------------
 -2.5
 1
 "Amazon"
 "AWS"
 [1,"abc",null]
(5 rows)
```

有关 ORDER BY 子句的更多信息，请参阅ORDER BY 子句。

运算符和函数

Amazon Redshift 为 SUPER 运算符和函数提供以下功能支持。

算术运算符

SUPER 值支持使用动态类型的所有基本算术运算符 +、-、*、/、%。运算的结果类型保持为 SUPER。对于所有运算符（二进制运算符 + 除外），输入操作数必须是数字。否则，Amazon Redshift 返回 null。当 Amazon Redshift 运行这些运算符且动态类型不改变时，小数和浮点值之间的区别将保留。但是，使用乘法和除法时，小数位数会发生变化。算术溢出仍然会导致查询错误，它们不会

更改为 null。如果输入是数字，则二进制运算符 + 执行加法；如果输入是字符串，则执行串联。如果一个操作数是字符串，另一个操作数是数字，则结果为 null。如果 SUPER 值不是以下示例所示的数字，则一元前缀运算符 + 和-返回 null：

```
SELECT (c_orders[0]. o_orderkey + 0.5) * c_orders[0]. o_orderkey / 10 AS math FROM
 customer_orders_lineitem;
           math
----------------------------
 1757958232200.1500
(1 row)
```

动态键入允许 SUPER 中的十进制值具有不同的小数位数。Amazon Redshift 将十进制值视为不同的静态类型，并允许所有数学运算。Amazon Redshift 根据操作数的小数位数动态计算产生的小数位数。如果其中一个操作数是浮点数，则 Amazon Redshift 会将另一个操作数升级为浮点数，并以浮点数的形式生成结果。

算术函数

Amazon Redshift 支持 SUPER 列的以下算术函数。如果输入不是数字，则它们返回 null：

- FLOOR。有关更多信息，请参阅 FLOOR 函数。
- CEIL 和 CEILING。有关更多信息，请参阅 CEILING (或 CEIL) 函数。
- ROUND。有关更多信息，请参阅 ROUND 函数。
- TRUNC。有关更多信息，请参阅 TRUNC 函数。
- ABS。有关更多信息，请参阅 ABS 函数。

以下示例使用算术函数查询数据：

```
SELECT x, FLOOR(x), CEIL(x), ROUND(x)
FROM (
    SELECT (c_orders[0]. o_orderkey + 0.5) * c_orders[0].o_orderkey / 10 AS x
    FROM customer_orders_lineitem
    );

         x          |     floor     |     ceil      |     round
--------------------+---------------+---------------+---------------
 1389636795898.0500  | 1389636795898  | 1389636795899  | 1389636795898
```

ABS 函数在 FLOOR、CEIL 时保留输入小数的小数位数。ROUND 消除了输入小数的小数位数。

数组函数

Amazon Redshift 支持以下数组组合和实用程序函数数组、array_concat、subarray、array_flatten、get_array_length、and split_to_array。

您可以使用 ARRAY 函数（包括其他 SUPER 值）通过 Amazon Redshift 数据类型中的值构建 SUPER 数组。以下示例使用可变数函数 ARRAY：

```
SELECT ARRAY(1, c.c_custkey, NULL, c.c_name, 'abc') FROM customer_orders_lineitem c;
                              array
 ------------------------------------------------------
[1,8401,null,""Customer#000008401"",""abc""]
[1,9452,null,""Customer#000009452"",""abc""]
[1,9451,null,""Customer#000009451"",""abc""]
[1,8251,null,""Customer#000008251"",""abc""]
[1,5851,null,""Customer#000005851"",""abc""]
(5 rows)
```

以下示例将数组串联与 ARRAY_CONCAT 函数结合使用：

```
SELECT ARRAY_CONCAT(JSON_PARSE('[10001,10002]'),JSON_PARSE('[10003,10004]'));

             array_concat
-------------------------------------
 [10001,10002,10003,10004]
(1 row)
```

以下示例将数组操作与 SUBARRAY 函数一起使用，该函数返回输入数组的子集。

```
SELECT SUBARRAY(ARRAY('a', 'b', 'c', 'd', 'e', 'f'), 2, 3);

   subarray
---------------
 ["c","d","e"]
(1 row))
```

以下示例使用 ARRAY_FLATTEN 将多个级别的数组合并到单个数组中：

```
SELECT x, ARRAY_FLATTEN(x) FROM (SELECT ARRAY(1, ARRAY(2, ARRAY(3, ARRAY())))  AS x);

     x           | array_flatten
 ----------------+---------------
```

```
 [1,[2,[3,[]]]]  | [1,2,3]
(1 row)
```

数组函数 ARRAY_CONCAT 和 ARRAY_FLATTEN 使用动态键入规则。如果输入不是数组，则它们返回 null 而不是错误。GET_ARRAY_LENGTH 函数在给定对象或数组路径的情况下返回 SUPER 数组的长度。

```
SELECT c_name
FROM customer_orders_lineitem
WHERE GET_ARRAY_LENGTH(c_orders) = (
    SELECT MAX(GET_ARRAY_LENGTH(c_orders))
    FROM customer_orders_lineitem
    );
```

以下示例使用 SPLIT_TO_ARRAY 将字符串拆分为字符串数组。函数将分隔符用作可选参数。如果没有分隔符，则默认值为逗号。

```
SELECT SPLIT_TO_ARRAY('12|345|6789', '|');

   split_to_array
---------------------
 ["12","345","6789"]
(1 row)
```

SUPER 配置

当您使用 Amazon Redshift SUPER 数据类型和 PartiQL 时，请注意 SUPER 配置的以下注意事项。

宽松和严格的 SUPER 模式

当您查询 SUPER 数据时，路径表达式可能不匹配实际的 SUPER 数据结构。当您尝试访问对象或数组元素的不存在成员时，如果您的查询在默认宽松模式下运行，Amazon Redshift 将返回 NULL 值。如果您在严格模式下运行查询，Amazon Redshift 将返回错误。可以将以下会话参数设置为打开或关闭宽松模式。

以下示例使用会话参数启用宽松模式。

```
SET navigate_super_null_on_error=ON;  --default lax mode for navigation

SET cast_super_null_on_error=ON;  --default lax mode for casting
```

```
SET parse_super_null_on_error=OFF;  --default strict mode for ingestion
```

访问具有大写和混合大小写字段名称或属性的 JSON 字段

当 JSON 属性名称采用大写或混合大小写时，您必须能够以区分大小写的方式处理 SUPER 类型结构。为此，您可以将 enable_case_sensitive_identifier 配置为 TRUE，并将大写和混合大小写的属性名称用双引号括起来。您也可以将 enable_case_sensitive_super_attribute 配置为 TRUE。在这种情况下，您可以在查询中使用大写和混合大小写的属性名称，而不必用双引号将其括起来。

以下示例说明如何设置 enable_case_sensitive_identifier 来查询数据。

```
SET enable_case_sensitive_identifier to TRUE;

-- Accessing JSON attribute names with uppercase and mixedcase names
SELECT json_table.data."ITEMS"."Name",
       json_table.data."price"
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 Name | price
------+-------
 "TV" | 345
(1 row)

RESET enable_case_sensitive_identifier;

-- After resetting the above configuration, the following query accessing JSON
 attribute names with uppercase and mixedcase names should return null (if in lax
 mode).
SELECT json_table.data."ITEMS"."Name",
       json_table.data."price"
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 name | price
------+-------
      | 345
(1 row)
```

以下示例说明如何设置 enable_case_sensitive_super_attribute 来查询数据。

```
SET enable_case_sensitive_super_attribute to TRUE;
 -- Accessing JSON attribute names with uppercase and mixedcase names

SELECT json_table.data.ITEMS.Name,
       json_table.data.price
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 name | price
 -----+-------
 "TV" | 345
(1 row)

 RESET enable_case_sensitive_super_attribute;

 -- After resetting enable_case_sensitive_super_attribute, the query now returns NULL
 for ITEMS.Name (if in lax mode).

SELECT json_table.data.ITEMS.Name,
       json_table.data.price
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 name | price
 -----+-------
      | 345
(1 row)
```

解析 SUPER 的选项

当您使用 JSON_PARSE 函数将 JSON 字符串解析为 SUPER 值时，某些限制适用：

- 相同的属性名称不能出现在同一个对象中，但可以出现在嵌套对象中。`json_parse_dedup_attributes` 配置选项允许 JSON_PARSE 仅保留最后一次出现的重复属性，而不是返回错误。
- 字符串值不能超过 65535 个字节的系统最大 varchar 大小。`json_parse_truncate_strings` 配置选项允许 JSON_PARSE() 自动截断超过此限制的字符串，而不会返回错误。此行为仅影响字符串值，而不影响属性名称。

有关 JSON_PARSE 函数的更多信息，请参阅 JSON_PARSE 函数。

以下示例显示如何将 json_parse_dedup_attributes 配置选项设置为默认行为，即针对重复属性返回错误。

```
SET json_parse_dedup_attributes=OFF;  --default behavior of returning error instead of
 de-duplicating attributes
```

下面的示例显示如何设置 json_parse_truncate_strings 配置选项以实现默认行为，即针对长度超过此限制的字符串返回错误。

```
SET json_parse_truncate_strings=OFF;  --default behavior of returning error instead of
 truncating strings
```

限制

使用 SUPER 数据类型时，请考虑以下限制：

- 不能将 SUPER 列定义为分配键或排序键。
- 单个 SUPER 对象最多可保存 16 MB 的数据。
- SUPER 对象中的单个值被限制为对应 Amazon Redshift 类型的最大长度。例如，加载到 SUPER 的单个字符串值限制为 65535 个字节的最大 VARCHAR 长度。
- 不能对 SUPER 列执行部分更新或转换操作。
- 您不能在右联接或完全外连接中使用 SUPER 数据类型及其别名。
- SUPER 数据类型不支持 XML 作为入站或出站序列化格式。
- 在引用用于取消嵌套的表变量的子查询（相关或不相关）的 FROM 子句中，查询只能引用其父表而不能引用其他表。
- 转换限制

  SUPER 值可以转换为其他数据类型或从其他数据类型进行转换，但存在一些例外情况：

  - Amazon Redshift 不区分小数位数为 0 的整数和小数。
  - 如果小数位数不为零，则 SUPER 数据类型的行为与其他 Amazon Redshift 数据类型相同，但 Amazon Redshift 会将与 SUPER 相关的错误转换为 null，如以下示例所示。

    ```
    SELECT 5::bool;
     bool
    -------
     True
    (1 row)
    ```

```
SELECT 5::decimal::bool;
ERROR:  cannot cast type numeric to boolean

SELECT 5::super::bool;
 bool
-------
 True
(1 row)

SELECT 5.0::bool;
ERROR:  cannot cast type numeric to boolean

SELECT 5.0::super::bool;
 bool
-------
(1 row)
```

- Amazon Redshift 不会将日期和时间类型转换为 SUPER 数据类型。Amazon Redshift 只能从超级数据类型转换日期和时间数据类型，如以下示例所示。

```
SELECT o.o_orderdate FROM customer_orders_lineitem c,c.c_orders o;
  order_date
------------------
 "2001-09-08"
(1 row)


SELECT JSON_TYPEOF(o.o_orderdate) FROM customer_orders_lineitem c,c.c_orders o;
 json_typeof
-----------------
 string
(1 row)


SELECT o.o_orderdate::date FROM customer_orders_lineitem c,c.c_orders o;
 order_date
----------------
 2001-09-08
(1 row)


--date/time cannot be cast to super
```

```
SELECT '2019-09-09'::date::super;
ERROR:  cannot cast type date to super
```

- 从非标量值（对象和数组）转换为字符串将返回 NULL。要正确序列化这些非标量值，请不要转换它们。请改用 json_serialize 来转换非标量值。json_serialize 函数返回一个 varchar。通常，您不需要将非标量值转换为 varchar，因为 Amazon Redshift 会隐式序列化，如以下第一个示例所示。

```
SELECT r_nations FROM region_nations WHERE r_regionkey=300;
   r_nations
----------------
 [1,"abc",null]
(1 row)

SELECT r_nations::varchar FROM region_nations WHERE r_regionkey=300;
 r_nations
-------------
(1 row)

SELECT JSON_SERIALIZE(r_nations) FROM region_nations WHERE r_regionkey=300;
 json_serialize
-----------------
 [1,"abc",null]
(1 row)
```

  - 对于不区分大小写的数据库，Amazon Redshift 不支持 SUPER 数据类型。对于不区分大小写的列，Amazon Redshift 不会将它们转换为 SUPER 类型。因此，Amazon Redshift 不支持 SUPER 列与触发转换的不区分大小写的列交互。
- Amazon Redshift 在子查询中不支持易失性函数，例如 RANDOM ( ) 或 TIMEOFDAY ( )，这些子查询使用此类子查询取消嵌套外部表或 IN 函数的左侧 (LHS)。

将 SUPER 数据类型用于具体化视图

Amazon Redshift 扩展了实体化视图的功能，以便在实体化视图中使用 SUPER 数据类型和 PartiQL。SQL 和 ParPartiQL 查询可以使用递增实体化视图进行预计算。有关实体化视图的更多信息，请参阅在 Amazon Redshift 中创建实体化视图。

将无 schema 和半结构化数据存储到 SUPER 中后，您可以使用 PartiQL 实体化视图对数据进行自检并将其分解为实体化视图。

加快 PartiQL 查询

您可以使用实体化视图加快在 SUPER 列中导航和/或取消嵌套分层数据的 PartiQL 查询。创建一个或多个实体化视图，将 SUPER 值分解为多个列，并利用 Amazon Redshift 分析查询的列式组织。因此，查询使用实体化视图。

实体化视图实质上会提取并标准化嵌套数据。标准化程度取决于您在将 SUPER 数据转换为传统的柱状数据方面投入了多少精力。

使用实体化视图分解成 SUPER 列

以下示例显示了一个实体化视图，该视图将嵌套数据分解，其结果列仍为 SUPER 数据类型。

```
SELECT c.c_name, o.o_orderstatus
FROM customer_orders_lineitem c, c.c_orders o;
```

以下示例显示了一个实体化视图，该视图通过分解的数据创建传统的 Amazon Redshift 标量列。

```
SELECT c.c_name, c.c_orders[0].o_totalprice
FROM customer_orders_lineitem c;
```

您可以创建单个实体化视图 super_mv 来加速这两个查询。

要应答第一个查询，必须具体化属性 o_orderstatus。您可以省略属性 c_name，因为它不涉及嵌套导航或取消嵌套。您还必须在实体化视图中包括 customer_orders_lineitem 的属性 c_custkey，以便能够将基表与实体化视图联接起来。

要应答第二个查询，您还必须具体化属性 o_totalprice 和数组 index o_idx of c_orders。因此，您可以访问 c_orders 的索引 0。

```
CREATE MATERIALIZED VIEW super_mv distkey(c_custkey) sortkey(c_custkey) AS (
  SELECT c_custkey, o.o_orderstatus, o.o_totalprice, o_idx
  FROM customer_orders_lineitem c, c.c_orders o AT o_idx
);
```

实体化视图 super_mv are SUPER 的属性 o_orderstatus 和 o_totalprice。

对基表 customer_orders_lineitem 进行更改时，将以增量方式刷新实体化视图 super_mv。

```
REFRESH MATERIALIZED VIEW super_mv;
```

```
INFO: Materialized view super_mv was incrementally updated successfully.
```

若要将第一个 PartiQL 查询重写为常规 SQL 查询，请如下所示将 customer_orders_lineitem 与 super_mv as 联接。

```
SELECT c.c_name, v.o_orderstatus
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey;
```

同样，您可以重写第二个 PartiQL 查询。以下示例对 o_idx = 0 使用筛选条件。

```
SELECT c.c_name, v.o_totalprice
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey
WHERE v.o_idx = 0;
```

在 CREATE MATERIALIZED VIEW 命令中，将 c_custkey 指定为分配键，并为排序键指定 super_mv。Amazon Redshift 会执行高效的合并联接，假设 c_custkey 也是 customer_orders_lineitem 的分配键和排序键。如果不是这种情况，您可以指定 c_custkey 作为 customer_orders_lineitem 的排序键和分配键，如下所示。

```
ALTER TABLE customer_orders_lineitem
ALTER DISTKEY c_custkey, ALTER SORTKEY (c_custkey);
```

使用 EXPLAIN 语句验证 Amazon Redshift 是否对重写的查询执行合并联接。

```
EXPLAIN
      SELECT c.c_name, v.o_orderstatus
      FROM customer_orders_lineitem c JOIN super_mv v ON c.c_custkey = v.c_custkey;

      QUERY PLAN

 ------------------------------------------------------------------------------------------------
      XN Merge Join DS_DIST_NONE  (cost=0.00..34701.82 rows=1470776 width=27)
      Merge Cond: ("outer".c_custkey = "inner".c_custkey)
      ->  XN Seq Scan on mv_tbl__super_mv__0 derived_table2  (cost=0.00..14999.86
 rows=1499986 width=13)
      ->  XN Seq Scan on customer_orders_lineitem c  (cost=0.00..999.96 rows=99996
 width=30)
      (4 rows)
```

使用分解数据创建 Amazon Redshift 标量列

了解存储到 SUPER 中的无 schema 数据会影响 Amazon Redshift 的性能。例如，筛选条件谓词或联接条件作为范围限制扫描无法有效地使用区域映射。用户和 BI 工具可以使用实体化视图作为数据的传统表示形式，并提高分析查询的性能。

以下查询扫描实体化视图 super_mv 并筛选 o_orderstatus。

```
SELECT c.c_name, v.o_totalprice
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey
WHERE v.o_orderstatus = 'F';
```

检查 stl_scan 以验证 Amazon Redshift 无法通过 o_orderstatus 在范围限制扫描中有效地使用区域映射。

```
SELECT slice, is_rrscan FROM stl_scan
WHERE query = pg_last_query_id() AND perm_table_name LIKE '%super_mv%';

 slice | is_rrscan
-------+-----------
     0 | f
     1 | f
     5 | f
     4 | f
     2 | f
     3 | f
(6 rows)
```

以下示例调整实体化视图 super_mv 以从分解的数据中创建标量列。在这种情况下，Amazon Redshift 会将 o_orderstatus 从 SUPER 转换为 VARCHAR。此外，指定 o_orderstatus 作为 super_mv 的排序键。

```
CREATE MATERIALIZED VIEW super_mv distkey(c_custkey) sortkey(c_custkey, o_orderstatus)
 AS (
  SELECT c_custkey, o.o_orderstatus::VARCHAR AS o_orderstatus, o.o_totalprice, o_idx
  FROM customer_orders_lineitem c, c.c_orders o AT o_idx
);
```

重新运行查询后，验证 Amazon Redshift 现在可以使用区域映射。

```
SELECT v.o_totalprice
FROM super_mv v
WHERE v.o_orderstatus = 'F';
```

您可以验证范围限制扫描现在是否使用区域映射，如下所示。

```
SELECT slice, is_rrscan FROM stl_scan
WHERE query = pg_last_query_id() AND perm_table_name LIKE '%super_mv%';

 slice | is_rrscan
-------+-----------
     0 | t
     1 | t
     2 | t
     3 | t
     4 | t
     5 | t
(6 rows)
```

将 SUPER 数据类型用于实体化视图的限制

将 SUPER 数据类型与实体化视图结合使用时，请遵守以下限制。

Amazon Redshift 中的实体化视图没有任何与 PartiQL 或 SUPER 有关的特定限制。

有关创建实体化视图时的一般 SQL 限制的信息，请参阅限制。

有关实体化视图递增刷新的一般 SQL 限制的信息，请参阅增量刷新限制。

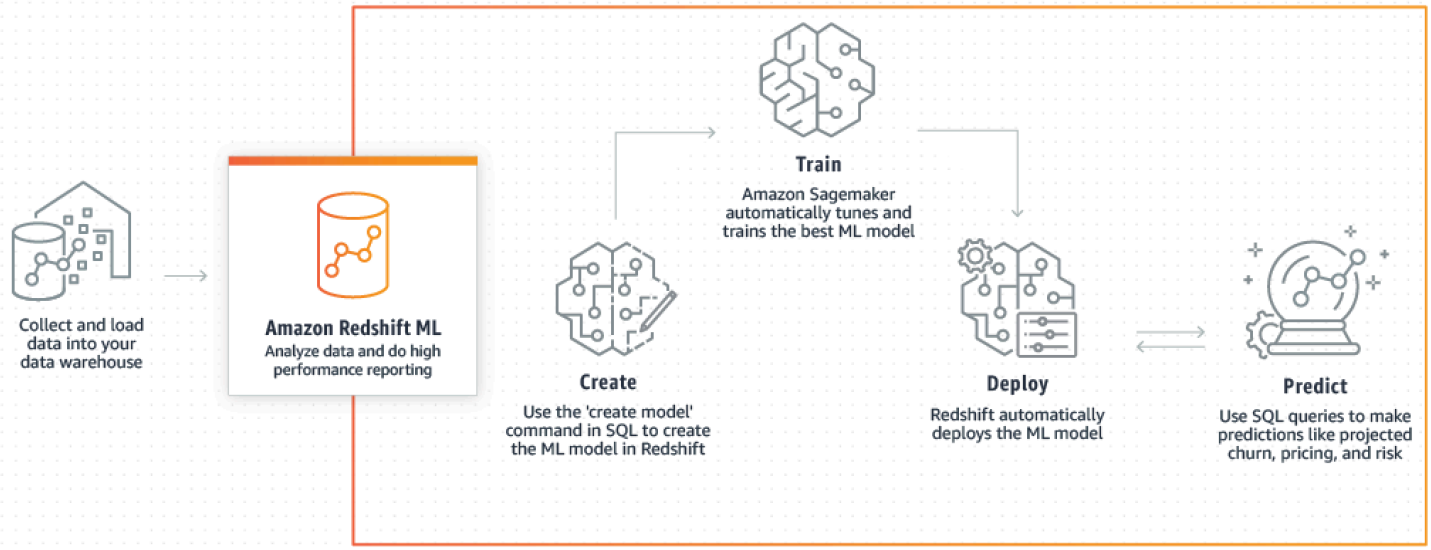
在 Amazon Redshift 中使用机器学习

Amazon Redshift 机器学习 (Amazon Redshift ML) 是一种基于云的稳健服务，能够让所有技能水平的分析人员和数据科学家都能更轻松使用机器学习技术。您将要训练模型的数据以及与数据输入相关的元数据提供给 Amazon Redshift。然后，Amazon Redshift ML 将创建模型来捕获输入数据中的模式。接着，您可以使用这些模型为新输入数据生成预测，而不会产生额外的成本。

Amazon Redshift ML 如何与 Amazon SageMaker 结合使用

Amazon Redshift 与 Amazon SageMaker Autopilot 结合使用，以自动获取最佳模型并使预测函数在 Amazon Redshift 中可用。

下图说明了 Amazon Redshift ML 的工作原理。



常见工作流程如下：

1. Amazon Redshift 将训练数据导出到 Simple Storage Service（Amazon S3）中。
2. Amazon SageMaker Autopilot 预处理训练数据。预处理执行重要功能，例如插入缺失值。它认识到有些列是分类的（如邮政编码），正确设置它们的格式以进行训练，并执行许多其他任务。选择要应用于训练数据集的最佳预处理器本身就是一个问题，Amazon SageMaker Autopilot 可自动执行其解决方案。
3. Amazon SageMaker Autopilot 查找算法和算法超参数，从而为模型提供最准确的预测结果。
4. Amazon Redshift 会在您的 Amazon Redshift 集群中将预测函数注册为 SQL 函数。
5. 当您运行 CREATE MODEL 语句时，Amazon Redshift 使用 Amazon SageMaker 进行训练。因此，训练模型会产生相关的成本。这是 Amazon SageMaker 在您的 AWS 账单中的单独行项目。您

还需要为 Simple Storage Service（Amazon S3）中用于存储训练数据的存储支付费用。使用可在 Redshift 集群上编译和运行的 CREATE MODEL 创建的模型进行推断不会产生费用。使用 Amazon Redshift ML 不会产生额外的 Amazon Redshift 费用。

主题

机器学习概览

通过使用 Amazon Redshift ML，您可以使用 SQL 语句训练机器学习模型，并在 SQL 查询中调用它们以进行预测。

为了帮助您了解如何使用 Amazon Redshift ML，您可以观看以下视频：Amazon Redshift ML。

有关设置 Redshift 集群的先决条件、使用 Amazon Redshift ML 的权限和所有权的信息，请阅读以下部分。这些部分还介绍了简单的训练和预测在 Amazon Redshift ML 中的工作原理。

机器学习如何解决问题

机器学习模型通过在训练数据中查找模式，然后将这些模式应用于新数据来生成预测。在机器学习中，您可以通过学习最能解释您的数据数据的模式来训练这些模型。然后，您可以使用这些模型对新数据做出预测（也称为推理）。机器学习通常是一个迭代过程，您可以通过更改参数和改进训练数据来继续提高预测的准确性。如果数据发生变化，则会使用新数据集重新训练新模型。

为了实现各种业务目标，可以使用不同的基础机器学习方法。

Amazon Redshift ML 中的有监督学习

Amazon Redshift 支持有监督学习，这是最常见的高级企业分析方法。当您拥有一组已建立的数据并了解特定输入数据如何预测各种业务成果时，监督学习是首选的机器学习方法。这些结果有时也称为标签。特别是，数据集是一个表，其中具有包含特征（输入）和目标（输出）的属性。例如，假设您有一个提供过去和现在客户的年龄和邮政编码的表。假设您还有一个“活动”字段，对于现有客户来说值为 true，对于已暂停其成员资格的客户来说值为 false。有监督机器学习的目标是发现导致客户流失的年龄和邮政编码模式，正如目标为“False”的客户所代表的那样。您可以使用此模型预测可能流失的客户，例如暂停其会员资格，或者提供保留激励。

Amazon Redshift 支持有监督学习，包括回归、二进制分类和多类分类。回归是指预测连续值的问题，例如客户的总支出。二进制分类是指预测两种结果之一的问题，例如预测客户是否流失。多类分类是指预测许多结果之一的问题，例如预测客户可能感兴趣的项目。数据分析师和数据科学家可以使用它来执行有监督学习，以解决范围包括预测、个性化或客户流失的各种问题。您还可以在预测哪些销售将关闭、收入预测、欺诈侦测和客户生命周期价值预测等问题中使用有监督学习。

Amazon Redshift ML 中的无监督学习

无监督学习使用机器学习算法对未标记的训练数据进行分析和分组。算法会发现隐藏的模式或分组。目标是对数据中的底层结构或分布进行建模，以了解有关数据的更多信息。

Amazon Redshift 支持使用 K-Means 集群算法来解决无监督学习问题。此算法可解决需要在数据中发现分组的集群问题。K-Means 算法尝试在数据中寻找离散分组。根据未分类数据的相似与不同之处进行分组和分区。通过分组，K-Means 算法以迭代方式确定最佳质心，并将每个成员分配给最近的质心。离同一质心最近的成员属于同一组。一个组的成员尽可能与同一组中的其它成员相似，并与其它组的成员尽可能不同。例如，K-Means 集群算法可用于对受疫情影响的城市进行分类，或根据消费品的受欢迎程度对城市进行分类。

使用 K-Means 算法时，您可以指定输入 k，该值指定要在数据中寻找的集群数量。此算法的输出是一组 k 个质心。每个数据点属于离其最近的 k 个集群之一。每个集群通过其质心进行描述。质心可以视为集群的多维平均值。K-Means 算法对距离进行比较，以了解集群之间的差异。距离越大通常表示集群之间的差异越大。

预处理数据对 K-Means 很重要，因为它可以确保模型的特征保持在相同的尺度上并生成可靠的结果。Amazon Redshift 支持一些用于 CREATE MODEL 语句的 K-Means 预处理器，例如 StandardScaler、MinMax 和 NumericPassthrough。如果您不想对 K-Means 应用任何预处理，请明确选择 NumericPassthrough 作为转换器。有关 K-Means 参数的更多信息，请参阅带有 K-MANES 参数的 CREATE MODEL。

为了帮助您了解如何使用 K-Means 集群执行无监督训练，您可以观看以下视频：使用 K-Means 集群进行无监督训练。

Amazon Redshift ML 的术语和概念

以下术语用于描述一些 Amazon Redshift ML 概念：

- Amazon Redshift 中的机器学习使用一个 SQL 命令训练模型。Amazon Redshift ML 和 Amazon SageMaker 管理所有数据转换、权限、资源使用情况以及适当模型的发现。
- 训练是 Amazon Redshift 通过将指定的数据子集运行到模型中来创建机器学习模型的阶段。Amazon Redshift 会自动启动 Amazon SageMaker 中的训练任务，并生成一个模型。

- 预测（也称为推理）是在 Amazon Redshift SQL 查询中使用模型来预测结果。推理时，Amazon Redshift 使用基于模型的预测函数作为大型查询的一部分来生成预测。预测在 Redshift 集群本地计算，从而实现高吞吐量、低延迟和零额外费用。
- 使用自带模型 (BYOM)，您可以将在 Amazon Redshift 之外训练的模型与 Amazon SageMaker 结合使用，以用于 Amazon Redshift 本地的数据库内推理。Amazon Redshift ML 支持在本地推理中使用 BYOM。
- 本地推理在模型在 Amazon SageMaker 中预训练、由 Amazon SageMaker Neo 编译并在 Amazon Redshift ML 中本地化时使用。要将本地推理支持的模型导入到 Amazon Redshift，请使用 CREATE MODEL 命令。Amazon Redshift 通过调用 Amazon SageMaker Neo 导入预训练的 SageMaker 模型。您可以在此编译模型并将已编译的模型导入 Amazon Redshift。使用本地推理提高速度并降低成本。
- 远程推理在 Amazon Redshift 调用部署在 SageMaker 中的模型端点时使用。远程推理提供了调用所有类型的自定义模型和深度学习模型的灵活度，例如您在 Amazon SageMaker 中构建和部署的 TensorFlow 模型。

同样重要的是以下几点：

- Amazon SageMaker 是一项完全托管的机器学习服务。借助 Amazon SageMaker，数据科学家和开发人员可以轻松地构建、训练模型，然后直接将模型部署到生产就绪托管环境中。有关 Amazon SageMaker 的信息，请参阅 Amazon SageMaker 开发人员指南中的什么是 Amazon SageMaker。
- Amazon SageMaker Autopilot 是一个功能集，可根据您的数据自动训练和调整用于分类或回归的最佳机器学习模型。您可以保持完全的控制和可见性。Amazon SageMaker Autopilot 支持以表格格式输入数据。Amazon SageMaker Autopilot 提供自动数据清理和预处理、线性回归的自动算法选择、二进制分类和多类分类功能。它还支持自动超参数优化 (HPO)、分布式训练、自动实例和集群大小选择。有关 Amazon SageMaker Autopilot 的信息，请参阅 Amazon SageMaker 开发人员指南中的使用 Amazon SageMaker Autopilot 自动化模型开发。

面向新手和专家的机器学习

通过 Amazon Redshift ML，您可以使用一个 SQL CREATE MODEL 命令来训练模型。CREATE MODEL 命令将创建一个模型，Amazon Redshift 可以使用该模型来生成具有熟悉 SQL 结构的基于模型的预测。

当您不具备机器学习、工具、语言、算法和 API 方面的专业知识时，Amazon Redshift ML 特别有用。借助 Amazon Redshift ML，您无需执行与外部机器学习服务集成所需的千篇一律的繁重工作。Amazon Redshift 为您节省了格式化和移动数据、管理权限控制或构建自定义集成、工作流和

脚本的时间。您可以轻松使用常用的机器学习算法，并简化从训练到预测需要频繁迭代的训练需求。Amazon Redshift 会自动发现最佳算法并针对您的问题调整最佳模型。您可以从 Amazon Redshift 集群中进行预测，而无需将数据移出 Amazon Redshift，也无需与其它服务接口并支付其它服务费用。

Amazon Redshift ML 支持数据分析师和数据科学家使用机器学习。此外，机器学习专家还能够利用自己的知识来指导 CREATE MODEL 语句仅使用他们指定的方面。通过这样做，您可以加快 CREATE MODEL 找到最佳候选项所需的时间和/或提高模型的准确性。

CREATE MODEL 语句在如何指定训练任务的参数方面提供了灵活度。借助此灵活度，机器学习新手或专家用户都能够选择他们首选的预处理器、算法、问题类型或超参数。例如，对客户流失感兴趣的用户可能会在 CREATE MODEL 语句中指定问题类型是一种可以很好地适用于客户流失的二进制分类。然后，CREATE MODEL 语句将其对最佳模型的搜索范围缩小到二进制分类模型中。即使用户选择了问题类型，CREATE MODEL 语句仍然有很多选项可以使用。例如，CREATE MODEL 发现并应用最佳的预处理转换，并发现最佳的超参数设置。

Amazon Redshift ML 通过使用 Amazon SageMaker Autopilot 自动查找最佳模型，使训练变得更轻松。在后台，Amazon SageMaker Autopilot 会根据您提供的数据自动训练和调整最佳的机器学习模型。然后，Amazon SageMaker Neo 将编译训练模型，并使其可在您的 Redshift 集群中进行预测。当您使用训练好的模型运行机器学习推理查询时，查询可以使用所有的 Amazon Redshift 大规模并行处理功能。同时，查询可以使用基于机器学习的预测。

- 作为机器学习新手，利用有关机器学习的不同方面（如预处理器、算法和超参数）的一般知识，只将 CREATE MODEL 语句用于指定的方面。然后，您可以缩短 CREATE MODEL 找到最佳候选项所需的时间或提高模型的准确性。此外，您还可以通过引入其他领域知识（如问题类型或目标）来提高预测的商业价值。例如，在客户流失情况下，如果结果“客户不活跃”很少，则 F1 目标通常优先于准确性目标。由于高精度模型可能会始终预测“客户处于活动状态”，因此可以实现高精度，但业务价值却很少。有关 F1 目标的信息，请参阅 Amazon SageMaker API 参考中的 AutoMLJobObjective。

  有关 CREATE MODEL 语句的基本选项的更多信息，请参阅简单 CREATE MODEL。
- 作为机器学习高级从业人员，您可以为某些（但不是全部）功能指定问题类型和预处理器。然后，CREATE MODEL 将遵循您对指定方面的建议。同时，CREATE MODEL 仍然会发现剩余功能的最佳预处理器和最佳超参数。有关如何约束训练管道的一个或多个方面的更多信息，请参阅根据用户指导创建模型。
- 作为机器学习专家，您可以完全控制训练和超参数调整。然后，CREATE MODEL 语句不会尝试发现最佳预处理器、算法和超参数，因为您做出了所有选择。有关如何将 CREATE MODEL 语句与 AUTO OFF 结合使用的更多信息，请参阅带有 AUTO OFF 的 CREATE XGBoost 模型。
- 作为数据工程师，您可以在 Amazon SageMaker 中引入预训练的 XGBoost 模型，并将其导入 Amazon Redshift 进行本地推理。使用自带模型 (BYOM)，您可以将在 Amazon Redshift 之外训练

的模型与 Amazon SageMaker 结合使用，以用于 Amazon Redshift 本地的数据库内推理。Amazon Redshift ML 支持在本地或远程推理中使用 BYOM。

有关如何将 CREATE MODEL 语句用于本地或远程推理的更多信息，请参阅自带模型 (BYOM) – 本地推理。

作为 Amazon Redshift ML 用户，您可以选择以下任何选项来训练和部署模型：

- 问题类型，请参阅根据用户指导创建模型。
- 目标，请参阅根据用户指导创建模型或者带有 AUTO OFF 的 CREATE XGBoost 模型。
- 模型类型，请参阅带有 AUTO OFF 的 CREATE XGBoost 模型。
- 预处理器，请参阅根据用户指导创建模型。
- 超参数，请参阅带有 AUTO OFF 的 CREATE XGBoost 模型。
- 自带模型 (BYOM)，请参阅自带模型 (BYOM) – 本地推理。

使用 Amazon Redshift ML 的成本

Amazon Redshift ML 使用您现有的集群资源进行预测，因此您可以避免额外的 Amazon Redshift 费用。创建或使用模型不会产生额外的 Amazon Redshift 费用。预测在 Amazon Redshift 集群本地进行，因此，除非您需要调整集群大小，否则您无需支付额外费用。Amazon Redshift ML 使用 Amazon SageMaker 来训练您的模型，这确实会产生额外的相关费用。

Amazon Redshift 集群中运行的预测函数不收取额外费用。CREATE MODEL 语句使用 Amazon SageMaker，并产生额外的费用。成本随训练数据中的单元格的数量而增加。单元格数量是记录数（在训练查询或表时间中）乘以列数的乘积。例如，当 CREATE MODEL 语句的 SELECT 查询创建 10,000 条记录和 5 列时，它创建的单元格数为 50,000。

在某些情况下，CREATE MODEL 的 SELECT 查询生成的训练数据超过了您提供的 MAX_CELLS 限制（如果您没有提供限制值，则为原定设置 100 万）。在这些情况下，CREATE MODEL 会随机选择大约 MAX_CELLS（即训练数据集中的“列数”记录），然后使用这些随机选择的元组执行训练。随机采样可确保减少的训练数据集不会有任何偏差。因此，通过设置 MAX_CELLS，您可以控制您的训练成本。

使用 CREATE MODEL 命令语句时，可以使用 MAX_CELLS 和 MAX_RUNTIME 选项来控制成本、时间和潜在模型精度。

MAX_RUNTIME 指定使用 AUTO ON 或 OFF 选项时，训练在 SageMaker 中可能花费的最长时间。根据数据集的大小，训练任务通常比 MAX_RUNTIME 早完成。训练模型后，Amazon Redshift 会在后台执行额外的工作，以便在集群中编译和安装您的模型。因此，CREATE MODEL 可能需要比 MAX_RUNTIME 更长的时间才能完成。但是，MAX_RUNTIME 会限制 SageMaker 中用于训练模型的计算量和时间。您可以使用 SHOW MODEL 随时检查模型的状态。

当您用 AUTO ON 运行 CREATE MODEL 时，Amazon Redshift ML 使用 SageMaker Autopilot 自动智能地探索不同型号（或候选型号），以找到最佳型号。MAX_RUNTIME 限制花费的时间和计算量。如果 MAX_RUNTIME 设置过低，则可能没有足够的时间来探索一个候选项。如果您看到错误 “Autopilot 候选项没有模型”，请使用较大的 MAX_RUNTIME 值重新运行 CREATE MODEL。有关该参数的更多信息，请参阅 Amazon SageMaker API 参考中的 MaxAutoMLJobRuntimeInSeconds。

当您用 AUTO OFF 运行 CREATE MODEL 时，MAX_RUNTIME 对应于在 SageMaker 中运行训练任务的时间限制。根据数据集的大小和使用的其他参数（例如 MODEL_TYPE XGBOOST 中的 num_rounds），训练任务通常会更快完成。

您还可以通过在运行 CREATE MODEL 时指定较小的 MAX_CELLS 值来控制成本或减少训练时间。单元格是数据库中的一个条目。每行对应的单元格数量与列数相同，这些单元格可以是固定的，也可以有不同的宽度。MAX_CELLS 限制单元格的数量，从而限制用于训练模型的训练示例数量。预设情况下，MAX_CELLS 设置为 100 万个单元格。减少 MAX_CELLS 会减少 CREATE MODEL 中的 SELECT 查询结果中的行数，Amazon Redshift 会导出该结果并发送到 SageMaker 来训练模型。在 AUTO ON 和 AUTO OFF 下，减少 MAX_CELLS 均可减小用于训练模型的数据集的大小。这种方法有助于降低训练模型的成本和时间。要查看有关特定训练作业的训练和计费时间的信息，请在 Amazon SageMaker 中选择 Training jobs（训练作业）。

增加 MAX_RUNTIME 和 MAX_CELLS 通常会允许 SageMaker 探索更多候选项，从而提高模型质量。这样一来，SageMaker 可能需要更多的时间来训练每个候选项，并使用更多的数据来训练更好的模型。如果希望更快地迭代或浏览数据集，请使用较低的 MAX_RUNTIME 和 MAX_CELLS。如果您希望提高模型的精度，请使用更高的 MAX_RUNTIME 和 MAX_CELLS。

有关与各种单元格数量相关的成本免费试用详细信息，请参阅 Amazon Redshift 定价。

Amazon Redshift ML 入门

Amazon Redshift ML 使 SQL 用户可以轻松地使用熟悉的 SQL 命令创建、训练和部署机器学习模型。通过使用 Amazon Redshift ML，您可以使用 Redshift 集群中的数据来通过 Amazon SageMaker 训练模型。随后，模型将会本地化，并可在 Amazon Redshift 数据库中进行预测。Amazon Redshift ML 目前支持机器学习算法 XGBoost（AUTO ON 和 OFF）和多层感知（AUTO ON）、K-Means（AUTO OFF）和线性学习器。

主题

为 Amazon Redshift ML 配置集群和管理设置

在使用 Amazon Redshift ML 之前，请完成集群设置并配置使用 Amazon Redshift ML 的权限。

用于使用 Amazon Redshift ML 的集群设置

在使用 Amazon Redshift ML 之前，请先完成以下先决条件。

作为 Amazon Redshift 管理员，请执行以下一次性设置。

为 Amazon Redshift ML 执行一次性集群设置：

1. 使用AWS Management Console或 AWS Command Line Interface（AWS CLI）创建 Amazon Redshift 集群。确保在创建集群时附上 AWS Identity and Access Management（IAM）策略。有关将 Amazon Redshift ML 与 Amazon SageMaker 结合使用所需的权限的更多信息，请参阅将 Amazon Redshift 机器学习（ML）与 Amazon SageMaker 结合使用所需的权限。
2. 通过下列方式之一创建使用 Amazon Redshift ML 所需的 IAM 角色：

   - 一个简单的方法是使用 `AmazonS3FullAccess` 和 `AmazonSageMakerFullAccess` 策略创建 IAM 角色，以与 Amazon Redshift ML 一起使用。如果您计划还要创建预测模型，请将 `AmazonForecastFullAccess` 策略也附加到您的角色。
   - 我们建议您通过 Amazon Redshift 控制台创建一个 IAM 角色，该角色拥有 `AmazonRedshiftAllCommandsFullAccess` 策略，具备运行 SQL 命令（例如 CREATE MODEL）的权限。Amazon Redshift 使用基于 API 的无缝机制以编程方式在您的 AWS 账户中代表您创建 IAM 角色。Amazon Redshift 会自动将现有的 AWS 托管式策略附加到该 IAM 角色。这种方法表示您可以停留在 Amazon Redshift 控制台中，无需切换到 IAM 控制台进行角色创建。有关更多信息，请参阅为 Amazon Redshift 创建默认 IAM 角色。

     创建 IAM 角色作为集群的默认角色时，将 `redshift` 添加为资源名称的一部分，或使用 RedShift 特定标签来标记这些资源。

如果您的集群启用了增强型 Amazon VPC 路由，则可以使用通过 Amazon Redshift 控制台创建的 IAM 角色。此 IAM 角色附加 AmazonRedshiftAllCommandsFullAccess 策略并向策略添加以下权限。这些额外权限允许 Amazon Redshift 在您的账户中创建和删除弹性网络接口（ENI），并将其附加到 Amazon EC2 或 Amazon ECS 上运行的编译任务。这样做使您的 Simple Storage Service（Amazon S3）桶中的对象只能通过互联网访问被阻止的 Virtual Private Cloud（VPC）进行访问。

```
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:CreateNetworkInterface",
        "ec2:ModifyNetworkInterfaceAttribute"
    ],
    "Resource": "*"
}
```

- 如果您要使用更具限制性的策略创建 IAM 角色，您可以使用以下策略。您还可以修改此策略以满足您的需求。

  Amazon S3 桶 redshift-downloads/redshift-ml/ 是存储用于其他步骤和示例的示例数据的位置。如果您不需要从 Simple Storage Service（Amazon S3）加载数据，您可以将其删除。或者，将其替换为您用于将数据加载到 Amazon Redshift 的其他 Simple Storage Service（Amazon S3）桶。

  your-account-id、*your-role* 和 *your-s3-bucket* 的值是您作为 CREATE MODEL 命令的一部分指定的值。

  （可选）如果您在使用 Amazon Redshift ML 时指定了一个 AWS KMS 键，请使用示例策略的 AWS KMS 键部分。*your-kms-key* 值是作为 CREATE MODEL 命令一部分使用的密钥。

  ```
  {
  ```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "sagemaker:*Job*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "s3:AbortMultipartUpload",
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:iam::<your-account-id>:role/<your-role>",
                "arn:aws:s3:::<your-s3-bucket>/*",
                "arn:aws:s3:::redshift-downloads/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::<your-s3-bucket>",
                "arn:aws:s3:::redshift-downloads"
            ]
        }
```

```
        // Optional section needed if you use AWS KMS keys.
        ,{
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:Encrypt",
                "kms:GenerateDataKey*"
             ],
             "Resource": [
                "arn:aws:kms:<your-region>:<your-account-id>:key/<your-kms-key>"
             ]
        }
    ]
}
```

3. 要允许 Amazon Redshift 和 SageMaker 担任角色以与其他服务交互，请将以下信任策略添加到 IAM 角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "redshift.amazonaws.com",
          "sagemaker.amazonaws.com",
          "forecast.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

4. （可选）创建 Simple Storage Service（Amazon S3）桶和 AWS KMS 密钥。供 Amazon Redshift 用来存储发送到 Amazon SageMaker 的训练数据以及从 Amazon SageMaker 接收训练模型。

5. （可选）创建 IAM 角色和 Simple Storage Service（Amazon S3）桶的不同组合，以控制对不同用户组的访问。

6. （可选）当您在 Redshift 集群上启用 VPC 路由时，请为您的 Redshift 集群所在的 VPC 创建 Simple Storage Service（Amazon S3）端点和 SageMaker 端点。这样做使得流量可以在 CREATE MODEL 期间通过您的 VPC 在服务之间运行。有关 VPC 路由的更多信息，请参阅 Amazon Redshift 中的增强型 VPC 路由。

   有关为超参数优化任务指定私有 VPC 所需的权限的更多信息，请参阅将 Amazon Redshift ML 与 Amazon SageMaker 结合使用所需的权限。

有关如何使用 CREATE MODEL 语句开始为不同的使用案例创建模型的信息，请参阅CREATE MODEL。

管理权限和所有权

正如其它数据库对象（例如表或函数）一样，Amazon Redshift 将创建和使用 ML 模型绑定到访问控制机制。这些是创建运行预测函数的模型所需的单独权限。

以下示例使用两个用户组：`retention_analyst_grp`（模型创建者）和 `marketing_analyst_grp`（模型用户）来说明 Amazon Redshift 如何管理访问控制。保留分析师创建机器学习模型，其它用户可以通过获得的权限使用这些模型。

超级用户可以使用以下语句授予 USER 或 GROUP 创建机器学习模型的权限。

```
GRANT CREATE MODEL TO GROUP retention_analyst_grp;
```

具有此权限的用户或组可以在集群中的任何架构中创建模型，前提是用户对 SCHEMA 具有通常的 CREATE 权限。机器学习模型是架构层次结构的一部分，方式类似于表、视图、过程和用户定义的函数。

假设架构 `demo_ml` 已存在，请按如下方式授予两个用户组对架构的权限。

```
GRANT CREATE, USAGE ON SCHEMA demo_ml TO GROUP retention_analyst_grp;
```

```
GRANT USAGE ON SCHEMA demo_ml TO GROUP marketing_analyst_grp;
```

要让其它用户使用机器学习推理函数，请授予 EXECUTE 权限。以下示例使用 EXECUTE 权限授予 marketing_analyst_grp GROUP 使用该模型的权限。

```
GRANT EXECUTE ON MODEL demo_ml.customer_churn_auto_model TO GROUP
 marketing_analyst_grp;
```

将 REVOKE 语句与 CREATE MODEL 和 EXECUTE 一起使用，取消用户或组的这些权限。有关权限控制命令的更多信息，请参阅GRANT和REVOKE。

将模型可解释性与 Amazon Redshift ML 结合使用

借助 Amazon Redshift ML 中的模型可解释性，您可以使用功能重要性值来帮助了解训练数据中的每个属性对预测结果的贡献。

模型可解释性解释您的模型所做的预测，以此帮助改善您的机器学习（ML）模型。模型可解释性有助于解释这些模型如何使用功能归因方法进行预测。

Amazon Redshift ML 融入了模型可解释性，为 Amazon Redshift ML 用户提供模型解释功能。有关模型可解释性的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的什么是机器学习预测的公平性和模型可解释性？。

模型可解释性还监控模型在生产过程中针对功能归因偏移所做的推断。它还提供了一些工具来帮助您生成模型监管报告，供您用来告知风险和合规团队以及外部监管机构。

当您在使用 CREATE MODEL 语句时指定 AUTO ON 或 AUTO OFF 选项时，在模型训练任务完成后，SageMaker 将创建解释输出。您可以使用 EXPLAIN_MODEL 函数查询 JSON 格式的可解释性报告。有关更多信息，请参阅机器学习函数。。

Amazon Redshift ML 概率指标

在有监督学习问题中，分类标签是使用输入数据进行预测的结果。例如，如果您使用模型来预测客户是否会重新订阅串流服务，那么可能的标签是“可能”和“不太可能”。Redshift ML 提供了概率指标的功能，该功能为每个标签分配一个概率以指明其可能性。这可以帮助您根据预测结果作出更明智的决策。在 Amazon Redshift ML 中，使用二进制分类或多分类器问题类型创建 AUTO ON 模型时提供了概率指标。如果您省略 AUTO ON 参数，Redshift ML 会假定模型应具有 AUTO ON。

创建模型

创建模型时，Amazon Redshift 会自动检测模型类型和问题类型。如果是分类问题，Redshift 会自动创建第二个推理函数，您可以使用该函数输出与每个标签相关的概率。第二个推理函数的名称是您指定的推理函数名称，后面再加上字符串 `_probabilities`。例如，如果您将推理函数命名为 `customer_churn_predict`，则第二个推理函数的名称为 `customer_churn_predict_probabilities`。然后，您可以查询此函数以获取每个标签的概率。

```
CREATE MODEL customer_churn_model
FROM customer_activity
```

```
    PROBLEM_TYPE BINARY_CLASSIFICATION
TARGET churn
FUNCTION customer_churn_predict
IAM_ROLE {default}
AUTO ON
SETTINGS ( S3_BUCKET '<DOC-EXAMPLE-BUCKET>'
```

获取概率

概率函数准备就绪后，运行该命令会返回一个 SUPER 类型，其中包含返回概率的数组及其关联标签。例如，结果 `"probabilities" : [0.7, 0.3], "labels" : ["False.", "True."]` 意味着 False 标签的概率为 0.7，True 标签的概率为 0.3。

```
SELECT customer_churn_predict_probabilities(Account_length, Area_code,
            VMail_message, Day_mins, Day_calls, Day_charge,Eve_mins, Eve_calls,
            Eve_charge, Night_mins, Night_calls, Night_charge,Intl_mins, Intl_calls,
            Intl_charge, Cust_serv_calls)
FROM customer_activity;

customer_churn_predict_probabilities
 --------------------
 {"probabilities" : [0.7, 0.3], "labels" : ["False.", "True."]}
 {"probabilities" : [0.8, 0.2], "labels" : ["False.", "True."]}
 {"probabilities" : [0.75, 0.25], "labels" : ["True.", "False"]}
```

概率和标签数组总是按其概率降序排序。您可以编写查询，通过取消嵌套概率函数的 SUPER 返回结果，仅返回概率最高的预测标签。

```
SELECT prediction.labels[0], prediction.probabilities[0]
            FROM (SELECT customer_churn_predict_probabilities(Account_length,
 Area_code,
            VMail_message, Day_mins, Day_calls, Day_charge,Eve_mins, Eve_calls,
            Eve_charge, Night_mins, Night_calls, Night_charge,Intl_mins, Intl_calls,
            Intl_charge, Cust_serv_calls) AS prediction
FROM customer_activity);

  labels   | probabilities
-----------+--------------
 "False."  | 0.7
 "False."  | 0.8
 "True."   | 0.75
```

为了简化查询，可以将预测函数的结果存储在表中。

```
CREATE TABLE churn_auto_predict_probabilities AS
             (SELECT customer_churn_predict_probabilities(Account_length, Area_code,
             VMail_message, Day_mins, Day_calls, Day_charge,Eve_mins, Eve_calls,
             Eve_charge, Night_mins, Night_calls, Night_charge,Intl_mins,
             Intl_calls, Intl_charge, Cust_serv_calls) AS prediction
FROM customer_activity);
```

您可以查询带有结果的表，以便仅返回概率高于 0.7 的预测。

```
SELECT prediction.labels[0], prediction.probabilities[0]
FROM churn_auto_predict_probabilities
WHERE prediction.probabilities[0] > 0.7;

  labels   | probabilities
-----------+--------------
 "False."  | 0.8
 "True."   | 0.75
```

使用索引表示法，可以获得特定标签的概率。以下示例返回所有 `True.` 标签的概率。

```
SELECT label, index, p.prediction.probabilities[index]
FROM churn_auto_predict_probabilities p, p.prediction.labels AS label AT index
WHERE label='True.';

  label  | index | probabilities
---------+-------+---------------
 "True." |     0 | 0.3
 "True." |     0 | 0.2
 "True." |     0 | 0.75
```

以下示例返回具有 `True.` 标签且概率大于 0.7 的所有行，这表明客户很可能会流失。

```
SELECT prediction.labels[0], prediction.probabilities[0]
FROM churn_auto_predict_probabilities
WHERE prediction.probabilities[0] > 0.7 AND prediction.labels[0] = "True.";

labels     | probabilities
-----------+--------------
 "True."   | 0.75
```

Amazon Redshift ML 的教程

可以使用 Amazon Redshift ML 通过 SQL 语句训练机器学习模型，并在 SQL 查询中调用模型以进行预测。Amazon Redshift 中的机器学习使用一个 SQL 命令训练模型。Amazon Redshift 会自动启动 Amazon SageMaker 中的训练任务，并生成一个模型。创建模型后，您就可以使用模型的预测函数在 Amazon Redshift 中执行预测。

按照这些教程中的步骤了解 Amazon Redshift ML 功能：

- 教程：构建客户流失模型
- 教程：构建远程推理模型
- 教程：构建 K 均值聚类模型
- 教程：构建多类别分类模型
- 教程：构建 XGBoost 模型
- 教程：构建回归模型
- 教程：使用线性学习器构建回归模型
- 教程：使用线性学习器构建多类别分类模型

教程：构建客户流失模型

在本教程中，您使用 Amazon Redshift ML 通过 CREATE MODEL 命令创建客户流失模型，并针对用户场景运行预测查询。然后，您可以使用 CREATE MODEL 命令生成的 SQL 函数来实施查询。

您可以使用简单的 CREATE MODEL 命令导出训练数据、训练模型、导入模型以及准备 Amazon Redshift 预测函数。使用 CREATE MODEL 语句将训练数据指定为表或 SELECT 语句。

该示例使用历史信息构建一个有关移动运营商客户流失的机器学习模型。首先，SageMaker 训练您的机器学习模型，然后使用任意客户的资料信息测试您的模型。模型经过验证后，Amazon SageMaker 会将模型和预测函数部署到 Amazon Redshift。您可以使用预测函数来预测客户是否会流失。

使用案例示例

您可以使用 Amazon Redshift ML 解决其他二进制分类问题，例如预测销售机会是否会关闭。您还可以预测金融交易是否具有欺诈性。

任务

- 先决条件

- 步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift
- 步骤 2：创建机器学习模型
- 步骤 3：使用模型执行预测

先决条件

要完成本教程，您需要满足以下先决条件：

- 您必须为 Amazon Redshift ML 设置 Amazon Redshift 集群。为此，请使用适用于 Amazon Redshift ML 管理的集群和配置设置文档。
- 您用于创建模型的 Amazon Redshift 集群和用于暂存训练数据和存储模型构件的 Amazon S3 桶必须位于同一个 AWS 区域中。
- 要下载本文档中使用的 SQL 命令和示例数据集，请执行以下操作之一：
  - 下载 SQL 命令、客户活动文件和鲍鱼文件。
  - 将 AWS CLI 用于 Simple Storage Service（Amazon S3），运行以下命令。您可以使用自己的目标路径。

```
aws s3 cp s3://redshift-downloads/redshift-ml/tutorial-scripts/redshift-ml-
tutorial.sql </target/path>
aws s3 cp s3://redshift-downloads/redshift-ml/customer_activity/
customer_activity.csv </target/path>
aws s3 cp s3://redshift-downloads/redshift-ml/abalone_xgb/abalone_xgb.csv </target/
path>
```

步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift

使用 Amazon Redshift 查询器 v2 编辑和运行查询并可视化结果。

运行以下查询将创建一个名为 `customer_activity` 的表，并从 Amazon S3 获取示例数据集。

```
DROP TABLE IF EXISTS customer_activity;

CREATE TABLE customer_activity (
state varchar(2),
account_length int,
area_code int,
phone varchar(8),
intl_plan varchar(3),
```

```
vMail_plan varchar(3),
vMail_message int,
day_mins float,
day_calls int,
day_charge float,
total_charge float,
eve_mins float,
eve_calls int,
eve_charge float,
night_mins float,
night_calls int,
night_charge float,
intl_mins float,
intl_calls int,
intl_charge float,
cust_serv_calls int,
churn varchar(6),
record_date date
);

COPY customer_activity
FROM 's3://redshift-downloads/redshift-ml/customer_activity/'
REGION 'us-east-1' IAM_ROLE default
FORMAT AS CSV IGNOREHEADER 1;
```

步骤 2：创建机器学习模型

在这个模型中，流失是我们的目标输入。模型的所有其他输入是一些属性，它们可帮助创建用于预测流失的函数。

下面的示例使用 CREATE MODEL 操作来提供一个模型，该模型使用诸如客户的年龄、邮政编码、支出和案例等输入来预测客户是否将成为活跃客户。在以下示例中，将 *DOC-EXAMPLE-BUCKET* 替换为您自己的 Amazon S3 桶。

```
CREATE MODEL customer_churn_auto_model
FROM
    (
      SELECT state,
             account_length,
             area_code,
             total_charge/account_length AS average_daily_spend,
             cust_serv_calls/account_length AS average_daily_cases,
             churn
```

```
      FROM customer_activity
      WHERE  record_date < '2020-01-01'
     )
TARGET churn FUNCTION ml_fn_customer_churn_auto
IAM_ROLE default SETTINGS (
  S3_BUCKET '<DOC-EXAMPLE-BUCKET>'
);
```

前一个示例中的 SELECT 查询将创建训练数据。TARGET 子句指定哪一列是 CREATE MODEL 操作用于学习如何进行预测的机器学习标签。目标列“流失”表示客户是否仍具有有效会员资格或已暂停会员资格。S3_BUCKET 字段是您之前创建的 Amazon S3 桶的名称。Amazon S3 桶用于在 Amazon Redshift 和 Amazon SageMaker 之间共享训练数据和构件。其余列是用于预测的功能。

有关 CREATE MODEL 命令的基本使用案例的语法和特性的摘要，请参阅简单 CREATE MODEL。

添加服务器端加密的权限（可选）

原定设置情况下，Amazon Redshift 使用 Amazon SageMaker Autopilot 进行训练。特别是，Amazon Redshift 将训练数据安全地导出到客户指定的 Amazon S3 桶。如果不指定 KMS_KEY_ID，则原定设置情况下，会使用服务器端加密 SSE-S3 对数据进行加密。

如果您使用服务器端加密及 AWS KMS 托管式密钥 (SSE-KMS) 对您的输入进行加密，则添加以下权限：

```
{
    "Effect": "Allow",
    "Action": [
    "kms:Encrypt"
    "kms:Decrypt"
    ]
}
```

有关 Amazon SageMaker 角色的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的 Amazon SageMaker 角色。

检查模型训练的状态（可选）

您可以使用 SHOW MODEL 命令来了解模型何时准备就绪。

使用以下操作检查模型的状态。

```
SHOW MODEL customer_churn_auto_model;
```

以下是上一个操作的输出示例。

```
+--------------------------
+-----------------------------------------------------------------------------------------------------------------
+
|           Key            |
                 Value
             |
+--------------------------
+-----------------------------------------------------------------------------------------------------------------
+
|        Model Name        |
        customer_churn_auto_model
             |
|       Schema Name        |
                 public
             |
|          Owner           |
                 awsuser
             |
|      Creation Time       |
        Tue, 14.06.2022 17:15:52
             |
|       Model State        |
                TRAINING
             |
|                          |

             |
|      TRAINING DATA:      |

             |
|          Query           | SELECT STATE, ACCOUNT_LENGTH, AREA_CODE, TOTAL_CHARGE /
 ACCOUNT_LENGTH AS AVERAGE_DAILY_SPEND, CUST_SERV_CALLS / ACCOUNT_LENGTH AS
 AVERAGE_DAILY_CASES, CHURN |
|                          |
         FROM CUSTOMER_ACTIVITY
             |
|                          |
    WHERE RECORD_DATE < '2020-01-01'
             |
|      Target Column       |
                 CHURN
             |
```

```
|                          |

             |
|       PARAMETERS:        |

             |
|        Model Type        |
                  auto
             |
|       Problem Type       |

             |
|        Objective         |

             |
|     AutoML Job Name      |
     redshiftml-20220614171552640901
             |
|      Function Name       |
        ml_fn_customer_churn_auto
             |
|   Function Parameters    |                                                  state
 account_length area_code average_daily_spend average_daily_cases
                       |
| Function Parameter Types |
      varchar int4 int4 float8 int4
             |
|         IAM Role         |
          default-aws-iam-role
             |
|        S3 Bucket         |
             DOC-EXAMPLE-BUCKET
             |
|       Max Runtime        |
                  5400
             |
+--------------------------
+-----------------------------------------------------------------------------------------------------------
+
```

模型训练完成后，model_state 变量变为 Model is Ready，预测函数变为可用。

步骤 3：使用模型执行预测

您可以使用 SQL 语句来查看预测模型所做的预测。在此示例中，由 CREATE MODEL 操作创建的预测函数名为 `ml_fn_customer_churn_auto`。预测函数的输入参数对应于功能的类型，如 varchar 对应于 `state`，而 integer 对应于 `account_length`。预测函数的输出类型与 CREATE MODEL 语句的 TARGET 列相同。

1. 您根据 2020 年 1 月 1 日之前的数据对模型进行了训练，因此，现在您可以对测试集使用预测函数。以下查询显示了 2020 年 1 月 1 日之后注册的客户是否会经历流失的预测。

```
SELECT
    phone,
    ml_fn_customer_churn_auto(
        state,
        account_length,
        area_code,
        total_charge / account_length,
        cust_serv_calls / account_length
    ) AS active
FROM
    customer_activity
WHERE
    record_date > '2020-01-01';
```

2. 以下示例将相同的预测函数用于不同的使用案例。在此案例中，Amazon Redshift 预测记录日期晚于 2020 年 1 月 1 日的各州客户的流失者和非流失者的比例。

```
WITH predicted AS (
    SELECT
        state,
        ml_fn_customer_churn_auto(
            state,
            account_length,
            area_code,
            total_charge / account_length,
            cust_serv_calls / account_length
        ) :: varchar(6) AS active
    FROM
        customer_activity
    WHERE
        record_date > '2020-01-01'
)
```

```
SELECT
    state,
    SUM(
        CASE
            WHEN active = 'True.' THEN 1
            ELSE 0
        END
    ) AS churners,
    SUM(
        CASE
            WHEN active = 'False.' THEN 1
            ELSE 0
        END
    ) AS nonchurners,
    COUNT(*) AS total_per_state
FROM
    predicted
GROUP BY
    state
ORDER BY
    state;
```

3. 以下示例将预测函数用于预测某州客户流失百分比的使用案例。在此案例中，Amazon Redshift 预测记录日期晚于 2020 年 1 月 1 日的流失百分比。

```
WITH predicted AS (
    SELECT
        state,
        ml_fn_customer_churn_auto(
            state,
            account_length,
            area_code,
            total_charge / account_length,
            cust_serv_calls / account_length
        ) :: varchar(6) AS active
    FROM
        customer_activity
    WHERE
        record_date > '2020-01-01'
)
SELECT
    state,
    CAST((CAST((SUM(
```

```
        CASE
            WHEN active = 'True.' THEN 1
            ELSE 0
        END
    )) AS FLOAT) / CAST(COUNT(*) AS FLOAT)) AS DECIMAL (3, 2)) AS pct_churn,
    COUNT(*) AS total_customers_per_state
FROM
    predicted
GROUP BY
    state
ORDER BY
    3 DESC;
```

相关主题

有关 Amazon Redshift ML 的更多信息，请参阅以下文档：

- 使用 Amazon Redshift ML 的成本
- CREATE MODEL 命令
- EXPLAIN_MODEL 函数

有关机器学习的更多信息，请参阅以下文档：

- 机器学习概览
- 面向新手和专家的机器学习
- 机器学习预测的公平性和模型可解释性是什么？

教程：构建远程推理模型

下面的教程介绍有关如何创建随机森林砍伐模型的步骤，该模型以前在 Amazon Redshift 之外的 Amazon SageMaker 中进行过训练和部署。随机森林砍伐算法检测数据集中的异常数据点。通过创建远程推理模型，您可以将随机森林砍伐 SageMaker 模型引入到 Amazon Redshift 中。然后，在 Amazon Redshift 中，您可以使用 SQL 对远程 SageMaker 端点执行预测。

您可以使用 CREATE MODEL 命令从 Amazon SageMaker 端点导入机器学习模型，并准备 Amazon Redshift 预测函数。使用 CREATE MODEL 操作时，您需要提供 SageMaker 机器学习模型的端点名称。

在本教程中，您将使用 SageMaker 模型端点创建 Amazon Redshift 机器学习模型。机器学习模型准备就绪后，您可以使用它在 Amazon Redshift 中执行预测。首先，在 Amazon SageMaker 中训练并创建端点，然后获得端点名称。然后，您可以使用 CREATE MODEL 命令通过 Amazon Redshift ML 创建模型。最后，使用 CREATE MODEL 命令生成的预测函数对模型执行预测。

使用案例示例

您可以使用随机森林砍伐模型和远程推理来检测其他数据集中的异常，例如预测电子商务交易的快速增长或减少。还可以预测天气或地震活动的重大变化。

任务

- 先决条件
- 步骤 1：部署 Amazon SageMaker 模型
- 步骤 2：获取 SageMaker 模型端点
- 步骤 3：将数据从 Amazon S3 加载到 Amazon Redshift
- 步骤 4：使用 Amazon Redshift ML 创建模型
- 步骤 5：使用模型执行预测

先决条件

要完成本教程，您需要满足以下先决条件：

- 您已完成 Amazon Redshift ML 的管理设置。
- 您已经下载了纽约出租车数据集，创建了 Amazon S3 桶，并已将数据上载到 Amazon S3 桶中。
- 您必须训练、部署 SageMaker 模型和端点，然后获取 SageMaker 端点的名称。使用此 AWS CloudFormation 模板自动预调配您的 AWS 账户中的所有 SageMaker 资源。

步骤 1：部署 Amazon SageMaker 模型

1. 要部署模型，请转到 Amazon SageMaker 控制台，在导航窗格中选择 Notebook（笔记本）下的 Notebook instances（笔记本实例）。
2. 对于 CloudFormation 模板创建的 Jupyter 笔记本，选择 Open Jupyter（打开 Jupyter）。
3. 选择`bring-your-own-model-remote-inference.ipynb`。
4. 通过将以下各行替换为您的 Amazon S3 桶和前缀，设置参数以在 Amazon S3 中存储训练输入和输出。

```
data_location=f"s3://{bucket}/{prefix}/",
output_path=f"s3://{bucket}/{prefix}/output",
```

5. 选择 fast-forward（快进）按钮以运行所有单元格。

步骤 2：获取 SageMaker 模型端点

在 Amazon SageMaker 控制台上，在导航窗格中的 Inference（推理）下方，选择 Endpoints（端点）并找到您的模型名称。在 Amazon Redshift 中创建远程推理模型时，必须复制模型的端点名称。

步骤 3：将数据从 Amazon S3 加载到 Amazon Redshift

使用 Amazon Redshift 查询编辑器 v2 在 Amazon Redshift 中运行以下 SQL 命令。这些命令会剔除 `rcf_taxi_data` 表（如果存在），创建一个同名的表，然后将示例数据集加载到该表中。

```
DROP TABLE IF EXISTS public.rcf_taxi_data CASCADE;

CREATE TABLE public.rcf_taxi_data (ride_timestamp timestamp, nbr_passengers int);

COPY public.rcf_taxi_data
FROM
    's3://sagemaker-sample-files/datasets/tabular/anomaly_benchmark_taxi/
NAB_nyc_taxi.csv'
    IAM_ROLE default
    IGNOREHEADER 1
    FORMAT AS CSV;
```

步骤 4：使用 Amazon Redshift ML 创建模型

运行以下查询，使用您在上一步中获得的 SageMaker 模型端点在 Amazon Redshift ML 中创建模型。将 *`randomcutforest-xxxxxxxxx`* 替换为您自己的 SageMaker 端点的名称。

```
CREATE MODEL public.remote_random_cut_forest
FUNCTION remote_fn_rcf(int)
RETURNS decimal(10, 6) SAGEMAKER '<randomcutforest-xxxxxxxxx>' IAM_ROLE default;
```

检查模型状态（可选）

您可以使用 SHOW MODEL 命令来了解模型何时准备就绪。

要检查模型状态，请使用以下 SHOW MODEL 操作。

```
SHOW MODEL public.remote_random_cut_forest
```

输出显示 SageMaker 端点和函数名称。

```
+--------------------------+----------------------------------------+
|        Model Name        |        remote_random_cut_forest        |
+--------------------------+----------------------------------------+
|       Schema Name        |                public                  |
|          Owner           |                awsuser                 |
|      Creation Time       |        Wed, 15.06.2022 17:58:21        |
|       Model State        |                 READY                  |
|                          |                                        |
|       PARAMETERS:        |                                        |
|         Endpoint         |      <randomcutforest-xxxxxxxxx>       |
|      Function Name       |              remote_fn_rcf             |
|      Inference Type      |                Remote                  |
| Function Parameter Types |                 int4                   |
|         IAM Role         |          default-aws-iam-role          |
+--------------------------+----------------------------------------+
```

步骤 5：使用模型执行预测

Amazon SageMaker 随机森林砍伐算法设计为检测数据集中的异常数据点。在此示例中，您的模型设计为检测由于重要事件而导致的出租车乘坐高峰。您可以使用该模型，通过为每个数据点生成异常分数来预测异常事件。

使用以下查询计算整个出租车数据集的异常分数。请注意，您引用您上一步在 CREATE MEL 语句中使用的函数。

```
SELECT
    ride_timestamp,
    nbr_passengers,
    public.remote_fn_rcf(nbr_passengers) AS score
FROM
    public.rcf_taxi_data;
```

检查高低异常（可选）

运行以下查询，以查找分数大于平均分数三个标准差的任何数据点。

```
WITH score_cutoff AS (
    SELECT
        STDDEV(public.remote_fn_rcf(nbr_passengers)) AS std,
        AVG(public.remote_fn_rcf(nbr_passengers)) AS mean,
        (mean + 3 * std) AS score_cutoff_value
    FROM
        public.rcf_taxi_data
)
SELECT
    ride_timestamp,
    nbr_passengers,
    public.remote_fn_rcf(nbr_passengers) AS score
FROM
    public.rcf_taxi_data
WHERE
    score > (
        SELECT
            score_cutoff_value
        FROM
            score_cutoff
    )
ORDER BY
    2 DESC;
```

运行以下查询，以查找分数大于平均分数三个标准差的任何数据点。

```
WITH score_cutoff AS (
    SELECT
        STDDEV(public.remote_fn_rcf(nbr_passengers)) AS std,
        AVG(public.remote_fn_rcf(nbr_passengers)) AS mean,
        (mean - 3 * std) AS score_cutoff_value
    FROM
        public.rcf_taxi_data
)
SELECT
    ride_timestamp,
    nbr_passengers,
    public.remote_fn_rcf(nbr_passengers) AS score
FROM
    public.rcf_taxi_data
WHERE
    score < (
        SELECT
```

```
            score_cutoff_value
        FROM
            score_cutoff
    )
ORDER BY
    2 DESC;
```

相关主题

有关 Amazon Redshift ML 的更多信息，请参阅以下文档：

- 使用 Amazon Redshift ML 的成本
- CREATE MODEL 操作
- EXPLAIN_MODEL 函数

有关机器学习的更多信息，请参阅以下文档：

- 机器学习概览
- 面向新手和专家的机器学习
- 机器学习预测的公平性和模型可解释性是什么？

教程：构建 K 均值聚类模型

在本教程中，您使用 Amazon Redshift ML 基于 K 均值算法，创建、训练和部署一个机器学习模型。此算法可解决需要在数据中发现分组的集群问题。K 均值有助于对尚未标注的数据进行分组。要了解有关 K 均值聚类的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的 K 均值聚类的工作原理。

您将使用 CREATE MODEL 操作从 Amazon Redshift 集群创建 K 均值模型。您可以使用 CREATE MODEL 命令导出训练数据、训练模型、导入模型以及准备 Amazon Redshift 预测函数。使用 CREATE MODEL 操作将训练数据指定为表或 SELECT 语句。

在本教程中，您将对全球事件、语言和语调数据库（GDELT）数据集使用 K 均值，该数据集监控世界各地的全球新闻，并且每天每秒都存储数据。K 均值将对语调、参与者或位置相似的事件进行分组。数据以多个文件的形式存储在 Amazon Simple Storage Service 的两个不同文件夹中。这些文件夹是历史文件夹（涵盖 1979 年 - 2013 年）和每日更新（涵盖 2013 年及以后的年份）。在此示例中，我们使用历史格式并引入 1979 年的数据。

使用案例示例

您可以使用 Amazon Redshift ML 解决其他聚类问题，例如对在流媒体服务上具有相似观看习惯的客户进行分组。您还可以使用 Redshift ML 来预测配送服务的最佳发货中心数量。

任务

- 先决条件
- 步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift
- 步骤 2：创建机器学习模型
- 步骤 3：使用模型执行预测

先决条件

要完成此教程，必须完成 Amazon Redshift ML 的管理设置。

步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift

1. 使用 Amazon Redshift 查询器 v2 运行以下查询。查询将剔除公有架构中的 gdelt_data 表（如果存在），并在公有架构中创建同名的表。

```
DROP TABLE IF EXISTS gdelt_data CASCADE;

CREATE TABLE gdelt_data (
    GlobalEventId bigint,
    SqlDate bigint,
    MonthYear bigint,
    Year bigint,
    FractionDate double precision,
    Actor1Code varchar(256),
    Actor1Name varchar(256),
    Actor1CountryCode varchar(256),
    Actor1KnownGroupCode varchar(256),
    Actor1EthnicCode varchar(256),
    Actor1Religion1Code varchar(256),
    Actor1Religion2Code varchar(256),
    Actor1Type1Code varchar(256),
    Actor1Type2Code varchar(256),
    Actor1Type3Code varchar(256),
    Actor2Code varchar(256),
    Actor2Name varchar(256),
    Actor2CountryCode varchar(256),
```

```
    Actor2KnownGroupCode varchar(256),
    Actor2EthnicCode varchar(256),
    Actor2Religion1Code varchar(256),
    Actor2Religion2Code varchar(256),
    Actor2Type1Code varchar(256),
    Actor2Type2Code varchar(256),
    Actor2Type3Code varchar(256),
    IsRootEvent bigint,
    EventCode bigint,
    EventBaseCode bigint,
    EventRootCode bigint,
    QuadClass bigint,
    GoldsteinScale double precision,
    NumMentions bigint,
    NumSources bigint,
    NumArticles bigint,
    AvgTone double precision,
    Actor1Geo_Type bigint,
    Actor1Geo_FullName varchar(256),
    Actor1Geo_CountryCode varchar(256),
    Actor1Geo_ADM1Code varchar(256),
    Actor1Geo_Lat double precision,
    Actor1Geo_Long double precision,
    Actor1Geo_FeatureID bigint,
    Actor2Geo_Type bigint,
    Actor2Geo_FullName varchar(256),
    Actor2Geo_CountryCode varchar(256),
    Actor2Geo_ADM1Code varchar(256),
    Actor2Geo_Lat double precision,
    Actor2Geo_Long double precision,
    Actor2Geo_FeatureID bigint,
    ActionGeo_Type bigint,
    ActionGeo_FullName varchar(256),
    ActionGeo_CountryCode varchar(256),
    ActionGeo_ADM1Code varchar(256),
    ActionGeo_Lat double precision,
    ActionGeo_Long double precision,
    ActionGeo_FeatureID bigint,
    DATEADDED bigint
);
```

2. 以下查询将示例数据加载到 gdelt_data 表。

```
COPY gdelt_data
```

```
FROM 's3://gdelt-open-data/events/1979.csv'
REGION 'us-east-1'
IAM_ROLE default
CSV
DELIMITER '\t';
```

检查训练数据（可选）

要查看将根据哪些数据训练模型，请使用以下查询。

```
SELECT
    AvgTone,
    EventCode,
    NumArticles,
    Actor1Geo_Lat,
    Actor1Geo_Long,
    Actor2Geo_Lat,
    Actor2Geo_Long
FROM
    gdelt_data LIMIT 100;
```

步骤 2：创建机器学习模型

以下示例使用 CREATE MODEL 命令创建一个模型，该模型将数据分组为七个聚类。K 值是数据点划分到的聚类的数量。该模型将数据点分类为数据点彼此更相似的聚类。通过将数据点聚类为组，K 均值算法以迭代方式确定最佳聚类中心。然后，算法将每个数据点分配给最近的聚类中心。离同一聚类中心最近的成员属于同一组。一个组的成员尽可能与同一组中的其它成员相似，并与其它组的成员尽可能不同。K 值是主观的，取决于测量数据点之间相似性的方法。如果聚类分布不均匀，则可以更改 K 值以平滑聚类大小。

在以下示例中，将 *DOC-EXAMPLE-BUCKET* 替换为您自己的 Amazon S3 桶。

```
CREATE MODEL news_data_clusters
FROM
    (
        SELECT
            AvgTone,
            EventCode,
            NumArticles,
            Actor1Geo_Lat,
            Actor1Geo_Long,
```

```
            Actor2Geo_Lat,
            Actor2Geo_Long
        FROM
            gdelt_data
    ) FUNCTION news_monitoring_cluster
    IAM_ROLE default
    AUTO OFF
    MODEL_TYPE KMEANS
    PREPROCESSORS 'none'
    HYPERPARAMETERS DEFAULT
    EXCEPT
    (K '7')
    SETTINGS (S3_BUCKET '<DOC-EXAMPLE-BUCKET>');
```

检查模型训练的状态（可选）

您可以使用 SHOW MODEL 命令来了解模型何时准备就绪。

要检查模型状态，请使用以下 SHOW MODEL 操作，并查明 Model State 是否为 Ready。

```
SHOW MODEL NEWS_DATA_CLUSTERS;
```

当模型准备就绪时，上一个操作的输出应显示 Model State 为 Ready。以下是 SHOW MODEL 操作的输出示例。

```
+--------------------------
+-------------------------------------------------------------------------------------------------
+
|        Model Name        |
 news_data_clusters                                    |
+--------------------------
+-------------------------------------------------------------------------------------------------
+
|       Schema Name        |                                                    public
                                          |
|          Owner           |                                                  awsuser
                                          |
|      Creation Time       |                                      Fri, 17.06.2022
 16:32:19                                      |
|       Model State        |                                                   READY
                                          |
|        train:msd         |                                          2973.822754
                                          |
```

```
|      train:progress      |                                                  100.000000
                                        |
|     train:throughput     |                                               237114.875000
                                        |
|      Estimated Cost      |                                                  0.004983
                                        |
|                          |
                                        |
|      TRAINING DATA:      |
                                        |
|          Query           | SELECT AVGTONE, EVENTCODE, NUMARTICLES, ACTOR1GEO_LAT,
 ACTOR1GEO_LONG, ACTOR2GEO_LAT, ACTOR2GEO_LONG |
|                          |                                                FROM GDELT_DATA
                                        |
|                          |
                                        |
|       PARAMETERS:        |
                                        |
|        Model Type        |                                                     kmeans
                                        |
|    Training Job Name     |
 redshiftml-20220617163219978978-kmeans                                  |
|      Function Name       |
 news_monitoring_cluster                                      |
|   Function Parameters    |       avgtone eventcode numarticles actor1geo_lat
 actor1geo_long actor2geo_lat actor2geo_long        |
| Function Parameter Types |                              float8 int8 int8 float8 float8
 float8 float8                          |
|         IAM Role         |                                        default-aws-iam-
role                                      |
|        S3 Bucket         |                                           DOC-EXAMPLE-
BUCKET                                     |
|       Max Runtime        |                                                5400
                                        |
|                          |
                                        |
|     HYPERPARAMETERS:     |
                                        |
|       feature_dim        |                                                  7
                                        |
|            k             |                                                  7
                                        |
```

```
+--------------------------
+-------------------------------------------------------------------------------------------------------------
+
```

步骤 3：使用模型执行预测

标识聚类

您可以找到模型在数据中标识的离散分组，也称为聚类。聚类是指一组数据点，它们离其聚类中心的距离比离任何其他聚类中心都更近。由于 K 值表示模型中的聚类数，因此它也表示聚类中心的数量。以下查询通过显示与每个 globaleventid 关联的聚类来标识聚类。

```
SELECT
    globaleventid,
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS cluster
FROM
    gdelt_data;
```

检查数据的分布

您可以检查数据跨聚类的分布，以查看所选的 K 值是否导致数据在某种程度上均匀分布。使用以下查询来确定数据是否在聚类间均匀分布。

```
SELECT
    events_cluster,
    COUNT(*) AS nbr_events
FROM
    (
        SELECT
            globaleventid,
            news_monitoring_cluster(
                AvgTone,
                EventCode,
                NumArticles,
```

```
                Actor1Geo_Lat,
                Actor1Geo_Long,
                Actor2Geo_Lat,
                Actor2Geo_Long
            ) AS events_cluster
        FROM
            gdelt_data
    )
GROUP BY
    1;
```

请注意，如果聚类分布不均匀，则可以更改 K 值以平滑聚类大小。

确定聚类中心

数据点离其聚类中心的距离比它离任何其他聚类中心都更近。因此，找到聚类中心有助于定义聚类。

运行以下查询，以根据事件代码的文章数量确定聚类的中心。

```
SELECT
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS events_cluster,
    eventcode,
    SUM(numArticles) AS numArticles
FROM
    gdelt_data
GROUP BY
    1,
    2;
```

显示有关聚类中的数据点的信息

使用以下查询返回分配给第五个聚类的点的数据。选定的文章必须有两个参与者。

```
SELECT
    news_monitoring_cluster (
```

```
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS events_cluster,
    eventcode,
    actor1name,
    actor2name,
    SUM(numarticles) AS totalarticles
FROM
    gdelt_data
WHERE
    events_cluster = 5
    AND actor1name <> ' '
    AND actor2name <> ' '
GROUP BY
    1,
    2,
    3,
    4
ORDER BY
    5 desc;
```

显示具有相同种族代码的参与者的事件数据

下面的查询统计了以积极的语调撰写的有关事件的文章数。该查询还要求两个参与者具有相同的种族代码，并返回每个事件分配到哪个聚类。

```
SELECT
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS events_cluster,
    SUM(numarticles) AS total_articles,
    eventcode AS event_code,
```

```
    Actor1EthnicCode AS ethnic_code
FROM
    gdelt_data
WHERE
    Actor1EthnicCode = Actor2EthnicCode
    AND Actor1EthnicCode <> ' '
    AND Actor2EthnicCode <> ' '
    AND AvgTone > 0
GROUP BY
    1,
    3,
    4
HAVING
    (total_articles) > 4
ORDER BY
    1,
    2 ASC;
```

相关主题

有关 Amazon Redshift ML 的更多信息，请参阅以下文档：

- 使用 Amazon Redshift ML 的成本
- CREATE MODEL 操作
- EXPLAIN_MODEL 函数

有关机器学习的更多信息，请参阅以下文档：

- 机器学习概览
- 面向新手和专家的机器学习
- 机器学习预测的公平性和模型可解释性是什么？

教程：构建多类别分类模型

在本教程中，您使用 Amazon Redshift ML 创建机器学习模型来解决多类别分类问题。多类别分类算法将数据点划分为三个或更多类别中的一个类别。然后，您可以使用 CREATE MODEL 命令生成的 SQL 函数来实施查询。

您可以使用 CREATE MODEL 命令导出训练数据、训练模型、导入模型以及准备 Amazon Redshift 预测函数。使用 CREATE MODEL 操作将训练数据指定为表或 SELECT 语句。

要继续学习本教程，您可以使用公用数据集电子商务销售额预测，其中包括一家英国在线零售商的销售数据。您生成的模型将为一个特殊的客户忠诚度计划找出最活跃的客户。通过多类别分类，您可以使用该模型预测客户在 13 个月期间内将有多少个月处于活跃状态。预测函数会指定据预测将活跃 7 个月或更多月数的客户，以便将这些客户加入该计划。

使用案例示例

您可以使用 Amazon Redshift ML 解决其他多类别分类问题，例如预测某个产品系列中最畅销的商品。您还可以预测图像中包含哪些水果，例如选择苹果、梨或橘子。

任务

- 先决条件
- 步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift
- 步骤 2：创建机器学习模型
- 步骤 3：使用模型执行预测

先决条件

要完成此教程，必须完成 Amazon Redshift ML 的管理设置。

步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift

使用 Amazon Redshift 查询器 v2 运行以下查询。这些查询会将示例数据加载到 Amazon Redshift 中。

1. 下面的查询创建一个名为 ecommerce_sales 的表。

```
CREATE TABLE IF NOT EXISTS ecommerce_sales (
    invoiceno VARCHAR(30),
    stockcode VARCHAR(30),
    description VARCHAR(60),
    quantity DOUBLE PRECISION,
    invoicedate VARCHAR(30),
    unitprice DOUBLE PRECISION,
    customerid BIGINT,
    country VARCHAR(25)
);
```

2. 以下查询将电子商务销售预测数据集中的示例数据复制到 ecommerce_sales 表中。

```
COPY ecommerce_sales
```

```
FROM
    's3://redshift-ml-multiclass/ecommerce_data.txt'
IAM_ROLE default
DELIMITER '\t'
IGNOREHEADER 1
REGION 'us-east-1'
MAXERROR 100;
```

拆分数据

当您在 Amazon Redshift ML 中创建模型时，SageMaker 会自动将您的数据拆分为训练集和测试集，以便 SageMaker 可以确定模型的准确性。通过在此步骤中手动拆分数据，您将能够通过分配额外的预测集来验证模型的准确性。

使用以下 SQL 语句将数据拆分为三个用于训练、验证和预测的集。

```
--creates table with all data
CREATE TABLE ecommerce_sales_data AS (
    SELECT
        t1.stockcode,
        t1.description,
        t1.invoicedate,
        t1.customerid,
        t1.country,
        t1.sales_amt,
        CAST(RANDOM() * 100 AS INT) AS data_group_id
    FROM
        (
            SELECT
                stockcode,
                description,
                invoicedate,
                customerid,
                country,
                SUM(quantity * unitprice) AS sales_amt
            FROM
                ecommerce_sales
            GROUP BY
                1,
                2,
                3,
                4,
```

```
                5
        ) t1
);

--creates training set
CREATE TABLE ecommerce_sales_training AS (
    SELECT
        a.customerid,
        a.country,
        a.stockcode,
        a.description,
        a.invoicedate,
        a.sales_amt,
        (b.nbr_months_active) AS nbr_months_active
    FROM
        ecommerce_sales_data a
        INNER JOIN (
            SELECT
                customerid,
                COUNT(
                    DISTINCT(
                        DATE_PART(y, CAST(invoicedate AS DATE)) || '-' || LPAD(
                            DATE_PART(mon, CAST(invoicedate AS DATE)),
                            2,
                            '00'
                        )
                    )
                ) AS nbr_months_active
            FROM
                ecommerce_sales_data
            GROUP BY
                1
        ) b ON a.customerid = b.customerid
    WHERE
        a.data_group_id < 80
);

--creates validation set
CREATE TABLE ecommerce_sales_validation AS (
    SELECT
        a.customerid,
        a.country,
        a.stockcode,
        a.description,
```

```
        a.invoicedate,
        a.sales_amt,
        (b.nbr_months_active) AS nbr_months_active
    FROM
        ecommerce_sales_data a
        INNER JOIN (
            SELECT
                customerid,
                COUNT(
                    DISTINCT(
                        DATE_PART(y, CAST(invoicedate AS DATE)) || '-' || LPAD(
                            DATE_PART(mon, CAST(invoicedate AS DATE)),
                            2,
                            '00'
                        )
                    )
                ) AS nbr_months_active
            FROM
                ecommerce_sales_data
            GROUP BY
                1
        ) b ON a.customerid = b.customerid
    WHERE
        a.data_group_id BETWEEN 80
        AND 90
);

--creates prediction set
CREATE TABLE ecommerce_sales_prediction AS (
    SELECT
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    FROM
        ecommerce_sales_data
    WHERE
        data_group_id > 90);
```

步骤 2：创建机器学习模型

在此步骤中，您使用 CREATE MODEL 语句通过多类别分类来创建机器学习模型。

以下查询使用 CREATE MODEL 操作通过训练集来创建多类别分类模型。将 *DOC-EXAMPLE-BUCKET* 替换为您自己的 Amazon S3 桶。

```
CREATE MODEL ecommerce_customer_activity
FROM
    (
        SELECT
            customerid,
            country,
            stockcode,
            description,
            invoicedate,
            sales_amt,
            nbr_months_active
        FROM
            ecommerce_sales_training
    ) TARGET nbr_months_active FUNCTION predict_customer_activity IAM_ROLE default
 PROBLEM_TYPE MULTICLASS_CLASSIFICATION SETTINGS (
        S3_BUCKET '<DOC-EXAMPLE-BUCKET>',
        S3_GARBAGE_COLLECT OFF
    );
```

在此查询中，您可以将问题类型指定为 `Multiclass_Classification`。您为模型预测的目标是 `nbr_months_active`。当 SageMaker 完成模型训练后，它会创建 `predict_customer_activity` 函数，您将使用该函数在 Amazon Redshift 中进行预测。

显示模型训练的状态（可选）

您可以使用 SHOW MODEL 命令来了解模型何时准备就绪。

使用以下查询返回模型的各种指标，包括模型状态和准确性。

```
SHOW MODEL ecommerce_customer_activity;
```

当模型准备就绪时，上一个操作的输出应显示 `Model State` 为 `Ready`。以下是 SHOW MODEL 操作的输出示例。

```
+--------------------------
+-----------------------------------------------------------------------------------------------
+
|        Model Name        |
 ecommerce_customer_activity                                    |
```

```
+--------------------------
+-----------------------------------------------------------------------------------------------------------
+
|       Schema Name        |                                                 public
                                    |
|          Owner           |                                                 awsuser
                                    |
|      Creation Time       |                                    Fri, 17.06.2022 19:02:15
                                    |
|       Model State        |                                                  READY
                                    |
|   Training Job Status    |
 MaxAutoMLJobRuntimeReached                                    |
|   validation:accuracy    |                                                 0.991280
                                    |
|      Estimated Cost      |                                                 7.897689
                                    |
|                          |
                                    |
|      TRAINING DATA:      |
                                    |
|          Query           | SELECT CUSTOMERID, COUNTRY, STOCKCODE, DESCRIPTION,
 INVOICEDATE, SALES_AMT, NBR_MONTHS_ACTIVE |
|                          |                                       FROM
 ECOMMERCE_SALES_TRAINING                                  |
|      Target Column       |                                           NBR_MONTHS_ACTIVE
                                    |
|                          |
                                    |
|       PARAMETERS:        |
                                    |
|        Model Type        |                                                  xgboost
                                    |
|       Problem Type       |                                    MulticlassClassification
                                    |
|        Objective         |                                            Accuracy
                                    |
|     AutoML Job Name      |
 redshiftml-20220617190215268770                                  |
|      Function Name       |
 predict_customer_activity                                    |
|   Function Parameters    |               customerid country stockcode description
 invoicedate sales_amt                |
```

```
| Function Parameter Types |                          int8 varchar varchar varchar
 varchar float8                          |
|         IAM Role         |                                    default-aws-iam-role
                                    |
|        S3 Bucket         |                                        DOC-EXAMPLE-BUCKET
                                    |
|       Max Runtime        |                                            5400
                                    |
+--------------------------
+-----------------------------------------------------------------------------------------------------------
+
```

步骤 3：使用模型执行预测

以下查询显示哪些客户有资格加入您的客户忠诚度计划。如果模型预测客户将活跃至少七个月，则模型将选择该客户加入忠诚度计划。

```
SELECT
    customerid,
    predict_customer_activity(
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    ) AS predicted_months_active
FROM
    ecommerce_sales_prediction
WHERE
    predicted_months_active >= 7
GROUP BY
    1,
    2
LIMIT
    10;
```

对验证数据运行预测查询（可选）

针对验证数据运行以下预测查询，以查看模型的准确性级别。

```
SELECT
    CAST(SUM(t1.match) AS decimal(7, 2)) AS predicted_matches,
```

```
    CAST(SUM(t1.nonmatch) AS decimal(7, 2)) AS predicted_non_matches,
    CAST(SUM(t1.match + t1.nonmatch) AS decimal(7, 2)) AS total_predictions,
    predicted_matches / total_predictions AS pct_accuracy
FROM
    (
        SELECT
            customerid,
            country,
            stockcode,
            description,
            invoicedate,
            sales_amt,
            nbr_months_active,
            predict_customer_activity(
                customerid,
                country,
                stockcode,
                description,
                invoicedate,
                sales_amt
            ) AS predicted_months_active,
            CASE
                WHEN nbr_months_active = predicted_months_active THEN 1
                ELSE 0
            END AS match,
            CASE
                WHEN nbr_months_active <> predicted_months_active THEN 1
                ELSE 0
            END AS nonmatch
        FROM
            ecommerce_sales_validation
    )t1;
```

预测有多少客户错过了加入机会（可选）

以下查询比较据预测仅活跃 5 个月或 6 个月的客户数量。该模型预测这些客户将错失忠诚度计划。然后，此查询将几乎可以不错过该计划的数量与预测有资格加入忠诚度计划的数量进行比较。此查询可用于决定是否降低忠诚度计划的门槛。您还可以确定是否有大量客户据预测几乎可以不错过该计划。然后，您可以鼓励这些客户增加其活动以获得忠诚度计划会员资格。

```
SELECT
    predict_customer_activity(
        customerid,
```

```
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    ) AS predicted_months_active,
    COUNT(customerid)
FROM
    ecommerce_sales_prediction
WHERE
    predicted_months_active BETWEEN 5 AND 6
GROUP BY
    1
ORDER BY
    1 ASC
LIMIT
    10)
UNION
(SELECT
      NULL AS predicted_months_active,
    COUNT (customerid)
FROM
    ecommerce_sales_prediction
WHERE
    predict_customer_activity(
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    ) >=7);
```

相关主题

有关 Amazon Redshift ML 的更多信息，请参阅以下文档：

- 使用 Amazon Redshift ML 的成本
- CREATE MODEL 操作
- EXPLAIN_MODEL 函数

有关机器学习的更多信息，请参阅以下文档：

- 机器学习概览
- 面向新手和专家的机器学习
- 机器学习预测的公平性和模型可解释性是什么？

教程：构建 XGBoost 模型

在本教程中，您使用来自 Amazon S3 的数据创建模型，并使用 Amazon Redshift ML 对模型运行预测查询。XGBoost 算法是梯度提升树算法的一种优化实施。与其他梯度提升树算法相比，XGBoost 处理的数据类型和关系更多，数据分布更广泛。您可以使用 XGBoost 来处理回归、二进制分类、多类别分类以及排名问题。有关 XGBoost 算法的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的 XGBoost 算法。

带有 `AUTO OFF` 选项的 Amazon Redshift ML `CREATE MODEL` 操作当前支持将 XGBoost 作为 `MODEL_TYPE`。根据您的使用案例，您可以在 `CREATE MODEL` 命令中提供相关信息，例如目标和超参数。

在本教程中，您将使用钞票验证数据集，这是一个二进制分类问题，用于预测给定的钞票是真钞还是假钞。

使用案例示例

您可以使用 Amazon Redshift ML 解决其他二进制分类问题，例如预测接受治疗者是健康的还是患有疾病。您还可以预测电子邮件是否为垃圾邮件。

任务

- 先决条件
- 步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift
- 步骤 2：创建机器学习模型
- 步骤 3：使用模型执行预测

先决条件

要完成此教程，必须完成 Amazon Redshift ML 的管理设置。

步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift

使用 Amazon Redshift 查询器 v2 运行以下查询。

以下查询创建两个表，从 Amazon S3 加载数据，然后将数据拆分为训练集和测试集。您将使用训练集来训练模型并创建预测函数。然后，您将在测试集上测试预测函数。

```
--create training set table
CREATE TABLE banknoteauthentication_train(
    variance FLOAT,
    skewness FLOAT,
    curtosis FLOAT,
    entropy FLOAT,
    class INT
);

--Load into training table
COPY banknoteauthentication_train
FROM
    's3://redshiftbucket-ml-sagemaker/banknote_authentication/train_data/' IAM_ROLE
 default REGION 'us-west-2' IGNOREHEADER 1 CSV;

--create testing set table
CREATE TABLE banknoteauthentication_test(
    variance FLOAT,
    skewness FLOAT,
    curtosis FLOAT,
    entropy FLOAT,
    class INT
);

--Load data into testing table
COPY banknoteauthentication_test
FROM
    's3://redshiftbucket-ml-sagemaker/banknote_authentication/test_data/'
    IAM_ROLE default
    REGION 'us-west-2'
    IGNOREHEADER 1
    CSV;
```

步骤 2：创建机器学习模型

以下查询根据您在上一步中创建的训练集，在 Amazon Redshift ML 中创建 XGBoost 模型。将 DOC-EXAMPLE-BUCKET 替换为您自己的 S3_BUCKET，它将存储您的输入数据集和其他 Redshift ML 构件。

```
CREATE MODEL model_banknoteauthentication_xgboost_binary
```

```
FROM
    banknoteauthentication_train
    TARGET class
    FUNCTION func_model_banknoteauthentication_xgboost_binary
    IAM_ROLE default
    AUTO OFF
    MODEL_TYPE xgboost
    OBJECTIVE 'binary:logistic'
    PREPROCESSORS 'none'
    HYPERPARAMETERS DEFAULT
EXCEPT(NUM_ROUND '100')
SETTINGS(S3_BUCKET '<DOC-EXAMPLE-BUCKET>');
```

显示模型训练的状态（可选）

您可以使用 SHOW MODEL 命令来了解模型何时准备就绪。

使用以下查询监控模型训练的进度。

```
SHOW MODEL model_banknoteauthentication_xgboost_binary;
```

如果模型为 READY，则 SHOW MODEL 操作还提供 `train:error` 指标，如以下输出示例所示。`train:error` 指标用于衡量模型的准确性，精确到小数点后六位。值为 0 表示最准确，值为 1 表示最不准确。

```
+--------------------------+-------------------------------------------------+
|        Model Name        |   model_banknoteauthentication_xgboost_binary   |
+--------------------------+-------------------------------------------------+
| Schema Name              | public                                          |
| Owner                    | awsuser                                         |
| Creation Time            | Tue, 21.06.2022 19:07:35                        |
| Model State              | READY                                           |
| train:error              |                                        0.000000 |
| Estimated Cost           |                                        0.006197 |
|                          |                                                 |
| TRAINING DATA:           |                                                 |
| Query                    | SELECT *                                        |
|                          | FROM "BANKNOTEAUTHENTICATION_TRAIN"             |
| Target Column            | CLASS                                           |
|                          |                                                 |
| PARAMETERS:              |                                                 |
| Model Type               | xgboost                                         |
```

```
| Training Job Name         | redshiftml-20220621190735686935-xgboost           |
| Function Name             | func_model_banknoteauthentication_xgboost_binary |
| Function Parameters       | variance skewness curtosis entropy                |
| Function Parameter Types | float8 float8 float8 float8                       |
| IAM Role                  | default-aws-iam-role                              |
| S3 Bucket                 | DOC-EXAMPLE-BUCKET    |
| Max Runtime               |                                                5400 |
|                           |                                                     |
| HYPERPARAMETERS:          |                                                     |
| num_round                 |                                                 100 |
| objective                 | binary:logistic                                     |
+---------------------------+-----------------------------------------------------+
```

步骤 3：使用模型执行预测

检查模型的准确性

以下预测查询使用在上一步中创建的预测函数来检查模型的准确性。对测试集运行此查询，以确保模型与训练集的对应关系不会过于紧密。这种紧密的对应关系也称为过拟合，而过拟合可能导致模型做出不可靠的预测。

```
WITH predict_data AS (
    SELECT
        class AS label,
        func_model_banknoteauthentication_xgboost_binary (variance, skewness, curtosis,
 entropy) AS predicted,
        CASE
            WHEN label IS NULL THEN 0
            ELSE label
        END AS actual,
        CASE
            WHEN actual = predicted THEN 1 :: INT
            ELSE 0 :: INT
        END AS correct
    FROM
        banknoteauthentication_test
),
aggr_data AS (
    SELECT
        SUM(correct) AS num_correct,
        COUNT(*) AS total
```

```
    FROM
        predict_data
)
SELECT
    (num_correct :: FLOAT / total :: FLOAT) AS accuracy
FROM
    aggr_data;
```

预测真钞和假钞的数量

以下预测查询返回测试集中预测的真钞和假钞的数量。

```
WITH predict_data AS (
    SELECT
        func_model_banknoteauthentication_xgboost_binary(variance, skewness, curtosis,
 entropy) AS predicted
    FROM
        banknoteauthentication_test
)
SELECT
    CASE
        WHEN predicted = '0' THEN 'Original banknote'
        WHEN predicted = '1' THEN 'Counterfeit banknote'
        ELSE 'NA'
    END AS banknote_authentication,
    COUNT(1) AS count
FROM
    predict_data
GROUP BY
    1;
```

找出真钞和假钞的平均观察值

以下预测查询返回测试集中预测为真钞和假钞的钞票的每个特征的平均值。

```
WITH predict_data AS (
    SELECT
        func_model_banknoteauthentication_xgboost_binary(variance, skewness, curtosis,
 entropy) AS predicted,
          variance,
          skewness,
          curtosis,
          entropy
```

```
    FROM
        banknoteauthentication_test
)
SELECT
    CASE
        WHEN predicted = '0' THEN 'Original banknote'
        WHEN predicted = '1' THEN 'Counterfeit banknote'
        ELSE 'NA'
    END AS banknote_authentication,
    TRUNC(AVG(variance), 2) AS avg_variance,
    TRUNC(AVG(skewness), 2) AS avg_skewness,
    TRUNC(AVG(curtosis), 2) AS avg_curtosis,
    TRUNC(AVG(entropy), 2) AS avg_entropy
FROM
    predict_data
GROUP BY
    1
ORDER BY
    2;
```

相关主题

有关 Amazon Redshift ML 的更多信息，请参阅以下文档：

- 使用 Amazon Redshift ML 的成本
- CREATE MODEL 操作
- EXPLAIN_MODEL 函数

有关机器学习的更多信息，请参阅以下文档：

- 机器学习概览
- 面向新手和专家的机器学习
- 机器学习预测的公平性和模型可解释性是什么？

教程：构建回归模型

在本教程中，您使用 Amazon Redshift ML 创建机器学习回归模型，并对该模型运行预测查询。回归模型允许您预测数值结果，例如房屋价格，或有多少人将使用城市的自行车租赁服务。您可以在 Amazon Redshift 中使用 CREATE MODEL 命令来处理您的训练数据。然后，Amazon Redshift ML 编译模型，

将经过训练的模型导入到 Redshift 中，并准备一个 SQL 预测函数。您可以在 Amazon Redshift 的 SQL 查询中使用预测函数。

在本教程中，您将使用 Amazon Redshift ML 构建回归模型，该模型可预测在一天中任何给定小时内使用多伦多市自行车共享服务的人数。模型的输入包括节假日和天气状况。您将使用回归模型，因为您需要此问题的数值结果。

您可以使用 CREATE MODEL 命令导出训练数据，训练模型，并使模型在 Amazon Redshift 中可用作 SQL 函数。使用 CREATE MODEL 操作将训练数据指定为表或 SELECT 语句。

使用案例示例

您可以使用 Amazon Redshift ML 解决其他回归问题，例如预测客户的生命周期价值。还可以使用 Redshift ML 来预测商品的最能赢利的价格和相应的收入。

任务

- 先决条件
- 步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift
- 步骤 2：创建机器学习模型
- 步骤 3：验证模型

先决条件

要完成此教程，必须完成 Amazon Redshift ML 的管理设置。

步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift

使用 Amazon Redshift 查询器 v2 运行以下查询。

1. 您必须创建三个表才能将三个公有数据集加载到 Amazon Redshift 中。这些数据集是多伦多自行车骑行数据、历史天气数据和历史节假日数据。在 Amazon Redshift 查询编辑器中运行以下查询，以便创建名为 `ridership`、`weather` 和 `holiday` 的表。

```
CREATE TABLE IF NOT EXISTS ridership (
    trip_id INT,
    trip_duration_seconds INT,
    trip_start_time timestamp,
    trip_stop_time timestamp,
    from_station_name VARCHAR(50),
    to_station_name VARCHAR(50),
```

```
    from_station_id SMALLINT,
    to_station_id SMALLINT,
    user_type VARCHAR(20)
);

CREATE TABLE IF NOT EXISTS weather (
    longitude_x DECIMAL(5, 2),
    latitude_y DECIMAL(5, 2),
    station_name VARCHAR(20),
    climate_id BIGINT,
    datetime_utc TIMESTAMP,
    weather_year SMALLINT,
    weather_month SMALLINT,
    weather_day SMALLINT,
    time_utc VARCHAR(5),
    temp_c DECIMAL(5, 2),
    temp_flag VARCHAR(1),
    dew_point_temp_c DECIMAL(5, 2),
    dew_point_temp_flag VARCHAR(1),
    rel_hum SMALLINT,
    rel_hum_flag VARCHAR(1),
    precip_amount_mm DECIMAL(5, 2),
    precip_amount_flag VARCHAR(1),
    wind_dir_10s_deg VARCHAR(10),
    wind_dir_flag VARCHAR(1),
    wind_spd_kmh VARCHAR(10),
    wind_spd_flag VARCHAR(1),
    visibility_km VARCHAR(10),
    visibility_flag VARCHAR(1),
    stn_press_kpa DECIMAL(5, 2),
    stn_press_flag VARCHAR(1),
    hmdx SMALLINT,
    hmdx_flag VARCHAR(1),
    wind_chill VARCHAR(10),
    wind_chill_flag VARCHAR(1),
    weather VARCHAR(10)
);

CREATE TABLE IF NOT EXISTS holiday (holiday_date DATE, description VARCHAR(100));
```

2. 以下查询将示例数据加载到您在上一步中创建的各表中。

```
COPY ridership
FROM
```

```
    's3://redshift-ml-bikesharing-data/bike-sharing-data/ridership/'
    IAM_ROLE default
    FORMAT CSV
    IGNOREHEADER 1
    DATEFORMAT 'auto'
    TIMEFORMAT 'auto'
    REGION 'us-west-2'
    gzip;

COPY weather
FROM
    's3://redshift-ml-bikesharing-data/bike-sharing-data/weather/'
    IAM_ROLE default
    FORMAT csv
    IGNOREHEADER 1
    DATEFORMAT 'auto'
    TIMEFORMAT 'auto'
    REGION 'us-west-2'
    gzip;

COPY holiday
FROM
    's3://redshift-ml-bikesharing-data/bike-sharing-data/holiday/'
    IAM_ROLE default
    FORMAT csv
    IGNOREHEADER 1
    DATEFORMAT 'auto'
    TIMEFORMAT 'auto'
    REGION 'us-west-2'
    gzip;
```

3. 以下查询对 ridership 和 weather 数据集执行转换以消除偏差或异常。消除偏差和异常可提高模型准确性。查询通过创建两个名为 ridership_view 和 weather_view 的新视图来简化表。

```
CREATE
OR REPLACE VIEW ridership_view AS
SELECT
    trip_time,
    trip_count,
    TO_CHAR(trip_time, 'hh24') :: INT trip_hour,
    TO_CHAR(trip_time, 'dd') :: INT trip_day,
    TO_CHAR(trip_time, 'mm') :: INT trip_month,
    TO_CHAR(trip_time, 'yy') :: INT trip_year,
```

```
    TO_CHAR(trip_time, 'q') :: INT trip_quarter,
    TO_CHAR(trip_time, 'w') :: INT trip_month_week,
    TO_CHAR(trip_time, 'd') :: INT trip_week_day
FROM
    (
        SELECT
            CASE
                WHEN TRUNC(r.trip_start_time) < '2017-07-01' :: DATE THEN
 CONVERT_TIMEZONE(
                    'US/Eastern',
                    DATE_TRUNC('hour', r.trip_start_time)
                )
                ELSE DATE_TRUNC('hour', r.trip_start_time)
            END trip_time,
            COUNT(1) trip_count
        FROM
            ridership r
        WHERE
            r.trip_duration_seconds BETWEEN 60
            AND 60 * 60 * 24
        GROUP BY
            1
    );

CREATE
OR REPLACE VIEW weather_view AS
SELECT
    CONVERT_TIMEZONE(
        'US/Eastern',
        DATE_TRUNC('hour', datetime_utc)
    ) daytime,
    ROUND(AVG(temp_c)) temp_c,
    ROUND(AVG(precip_amount_mm)) precip_amount_mm
FROM
    weather
GROUP BY
    1;
```

4. 以下查询创建了一个表，该表将来自 ridership_view 和 weather_view 的所有相关输入属性组合到 trip_data 表中。

```
CREATE TABLE trip_data AS
SELECT
```

```
    r.trip_time,
    r.trip_count,
    r.trip_hour,
    r.trip_day,
    r.trip_month,
    r.trip_year,
    r.trip_quarter,
    r.trip_month_week,
    r.trip_week_day,
    w.temp_c,
    w.precip_amount_mm,CASE
        WHEN h.holiday_date IS NOT NULL THEN 1
        WHEN TO_CHAR(r.trip_time, 'D') :: INT IN (1, 7) THEN 1
        ELSE 0
    END is_holiday,
    ROW_NUMBER() OVER (
        ORDER BY
            RANDOM()
    ) serial_number
FROM
    ridership_view r
    JOIN weather_view w ON (r.trip_time = w.daytime)
    LEFT OUTER JOIN holiday h ON (TRUNC(r.trip_time) = h.holiday_date);
```

查看示例数据（可选）

以下查询显示表中的条目。您可以运行此操作以确保正确创建了此表。

```
SELECT *
FROM trip_data
LIMIT 5;
```

以下是上一个操作的输出示例。

```
+---------------------+------------+-----------+----------+------------+-----------
+--------------+-----------------+---------------+--------+------------------
+------------+---------------+
|      trip_time      | trip_count | trip_hour | trip_day | trip_month | trip_year
 | trip_quarter | trip_month_week | trip_week_day | temp_c | precip_amount_mm |
 is_holiday | serial_number |
```

```
+---------------------+------------+-----------+----------+------------+-----------
+--------------+-----------------+---------------+--------+------------------
+------------+---------------+
| 2017-03-21 22:00:00 |         47 |        22 |       21 |          3 |        17 |
         1 |               3 |             3 |      1 |                0 |          0 |
             1 |
| 2018-05-04 01:00:00 |         19 |         1 |        4 |          5 |        18 |
         2 |               1 |             6 |     12 |                0 |          0 |
             3 |
| 2018-01-11 10:00:00 |         93 |        10 |       11 |          1 |        18 |
         1 |               2 |             5 |      9 |                0 |          0 |
             5 |
| 2017-10-28 04:00:00 |         20 |         4 |       28 |         10 |        17 |
         4 |               4 |             7 |     11 |                0 |          1 |
             7 |
| 2017-12-31 21:00:00 |         11 |        21 |       31 |         12 |        17 |
         4 |               5 |             1 |    -15 |                0 |          1 |
             9 |
+---------------------+------------+-----------+----------+------------+-----------
+--------------+-----------------+---------------+--------+------------------
+------------+---------------+
```

显示属性之间的相关性（可选）

确定相关性有助于衡量属性之间的关联强度。关联级别可以帮助您确定影响目标输出的因素。在本教程中，目标输出为 trip_count。

以下查询将创建或替换 sp_correlation 过程。您可以使用名为 sp_correlation 的存储过程，以显示 Amazon Redshift 的表中某个属性与其他属性之间的相关性。

```
CREATE OR REPLACE PROCEDURE sp_correlation(source_schema_name in varchar(255),
 source_table_name in varchar(255), target_column_name in varchar(255),
 output_temp_table_name inout varchar(255)) AS $$
DECLARE
  v_sql varchar(max);
  v_generated_sql varchar(max);
  v_source_schema_name varchar(255)=lower(source_schema_name);
  v_source_table_name varchar(255)=lower(source_table_name);
  v_target_column_name varchar(255)=lower(target_column_name);
BEGIN
  EXECUTE 'DROP TABLE IF EXISTS ' || output_temp_table_name;
  v_sql = '
SELECT
```

```
  ''CREATE temp table '|| output_temp_table_name||' AS SELECT ''|| outer_calculation||
  '' FROM (SELECT COUNT(1) number_of_items, SUM('||v_target_column_name||')
 sum_target, SUM(POW('||v_target_column_name||',2)) sum_square_target, POW(SUM('||
v_target_column_name||'),2) square_sum_target,''||
  inner_calculation||
  '' FROM (SELECT ''||
  column_name||
  '' FROM '||v_source_table_name||'))''
FROM
  (
  SELECT
    DISTINCT
    LISTAGG(outer_calculation,'','') OVER () outer_calculation
    ,LISTAGG(inner_calculation,'','') OVER () inner_calculation
    ,LISTAGG(column_name,'','') OVER () column_name
  FROM
    (
    SELECT
      CASE WHEN atttypid=16 THEN ''DECODE(''||column_name||'',true,1,0)'' ELSE
 column_name END column_name
      ,atttypid
      ,''CAST(DECODE(number_of_items * sum_square_''||rn||'' - square_sum_''||
rn||'',0,null,(number_of_items*sum_target_''||rn||'' - sum_target * sum_''||rn||
        '')/SQRT((number_of_items * sum_square_target - square_sum_target) *
 (number_of_items * sum_square_''||rn||
        '' - square_sum_''||rn||'')) AS numeric(5,2)) ''||column_name
 outer_calculation
      ,''sum(''||column_name||'') sum_''||rn||'',''||
            ''SUM(trip_count*''||column_name||'') sum_target_''||rn||'',''||
            ''SUM(POW(''||column_name||'',2)) sum_square_''||rn||'',''||
            ''POW(SUM(''||column_name||''),2) square_sum_''||rn inner_calculation
    FROM
      (
      SELECT
        row_number() OVER (order by a.attnum) rn
        ,a.attname::VARCHAR column_name
        ,a.atttypid
      FROM pg_namespace AS n
        INNER JOIN pg_class AS c ON n.oid = c.relnamespace
        INNER JOIN pg_attribute AS a ON c.oid = a.attrelid
      WHERE a.attnum > 0
        AND n.nspname = '''||v_source_schema_name||'''
        AND c.relname = '''||v_source_table_name||'''
        AND a.atttypid IN (16,20,21,23,700,701,1700)
```

```
      )
    )
)';
  EXECUTE v_sql INTO v_generated_sql;
  EXECUTE  v_generated_sql;
END;
$$ LANGUAGE plpgsql;
```

下面的查询显示数据集中的目标列、trip_count 和其他数值属性之间的相关性。

```
call sp_correlation(
    'public',
    'trip_data',
    'trip_count',
    'tmp_corr_table'
);

SELECT
    *
FROM
    tmp_corr_table;
```

下例显示了上一个 sp_correlation 操作的输出。

```
+------------+-----------+----------+------------+-----------+--------------
+-----------------+---------------+--------+------------------+------------
+---------------+
| trip_count | trip_hour | trip_day | trip_month | trip_year | trip_quarter
 | trip_month_week | trip_week_day | temp_c | precip_amount_mm | is_holiday |
 serial_number |
+------------+-----------+----------+------------+-----------+--------------
+-----------------+---------------+--------+------------------+------------
+---------------+
|          1 |      0.32 |     0.01 |       0.18 |      0.12 |         0.18 |
     0 |          0.02 |   0.53 |            -0.07 |      -0.13 |             0 |
+------------+-----------+----------+------------+-----------+--------------
+-----------------+---------------+--------+------------------+------------
+---------------+
```

步骤 2：创建机器学习模型

1. 以下查询通过指定 80% 的数据集用于训练和 20% 的数据集用于验证，将数据拆分为训练集和验证集。训练集是 ML 模型的输入，用于确定模型的最佳可能算法。创建模型后，您可以使用验证集来验证模型的准确性。

```
CREATE TABLE training_data AS
SELECT
    trip_count,
    trip_hour,
    trip_day,
    trip_month,
    trip_year,
    trip_quarter,
    trip_month_week,
    trip_week_day,
    temp_c,
    precip_amount_mm,
    is_holiday
FROM
    trip_data
WHERE
    serial_number > (
        SELECT
            COUNT(1) * 0.2
        FROM
            trip_data
    );

CREATE TABLE validation_data AS
SELECT
    trip_count,
    trip_hour,
    trip_day,
    trip_month,
    trip_year,
    trip_quarter,
    trip_month_week,
    trip_week_day,
    temp_c,
    precip_amount_mm,
    is_holiday,
    trip_time
```

```
FROM
    trip_data
WHERE
    serial_number <= (
        SELECT
            COUNT(1) * 0.2
        FROM
            trip_data
    );
```

2. 以下查询创建一个回归模型来预测任何输入日期和时间的 trip_count 值。在以下示例中，将 *DOC-EXAMPLE-BUCKET* 替换为您自己的 S3 桶。

```
CREATE MODEL predict_rental_count
FROM
    training_data TARGET trip_count FUNCTION predict_rental_count
    IAM_ROLE default
    PROBLEM_TYPE regression
    OBJECTIVE 'mse'
    SETTINGS (
        s3_bucket '<DOC-EXAMPLE-BUCKET>',
        s3_garbage_collect off,
        max_runtime 5000
    );
```

步骤 3：验证模型

1. 使用以下查询输出模型的多个方面，并在输出中找出均方误差指标。均方误差是回归问题的典型准确性指标。

```
show model predict_rental_count;
```

2. 根据验证数据运行以下预测查询，以将预测的行程计数与实际行程计数进行比较。

```
SELECT
    trip_time,
    actual_count,
    predicted_count,
    (actual_count - predicted_count) difference
FROM
    (
```

```
        SELECT
            trip_time,
            trip_count AS actual_count,
            PREDICT_RENTAL_COUNT (
                trip_hour,
                trip_day,
                trip_month,
                trip_year,
                trip_quarter,
                trip_month_week,
                trip_week_day,
                temp_c,
                precip_amount_mm,
                is_holiday
            ) predicted_count
        FROM
            validation_data
    )
LIMIT
    5;
```

3. 以下查询根据您的验证数据计算均方误差和均方根误差。您可以使用均方误差和均方根误差，来测量预测的数值目标与实际数值答案之间的差距。一个好的模型在这两个指标中的分数都很低。下面的查询返回这两个指标的值。

```
SELECT
    ROUND(
        AVG(POWER((actual_count - predicted_count), 2)),
        2
    ) mse,
    ROUND(
        SQRT(AVG(POWER((actual_count - predicted_count), 2))),
        2
    ) rmse
FROM
    (
        SELECT
            trip_time,
            trip_count AS actual_count,
            PREDICT_RENTAL_COUNT (
                trip_hour,
                trip_day,
                trip_month,
```

```
                trip_year,
                trip_quarter,
                trip_month_week,
                trip_week_day,
                temp_c,
                precip_amount_mm,
                is_holiday
            ) predicted_count
        FROM
            validation_data
    );
```

4. 以下查询计算 2017 年 1 月 1 日每次行程时间的行程计数误差百分比。该查询对行程时间进行排序，采用的顺序为从误差百分比最低的时间到误差百分比最高的时间。

```
SELECT
    trip_time,
    CAST(ABS(((actual_count - predicted_count) / actual_count)) * 100 AS DECIMAL
 (7,2)) AS pct_error
FROM
    (
        SELECT
            trip_time,
            trip_count AS actual_count,
            PREDICT_RENTAL_COUNT (
                trip_hour,
                trip_day,
                trip_month,
                trip_year,
                trip_quarter,
                trip_month_week,
                trip_week_day,
                temp_c,
                precip_amount_mm,
                is_holiday
            ) predicted_count
        FROM
            validation_data
    )
WHERE
   trip_time LIKE '2017-01-01 %%:%%:%%'
ORDER BY
   2 ASC;
```

相关主题

有关 Amazon Redshift ML 的更多信息，请参阅以下文档：

- 使用 Amazon Redshift ML 的成本
- CREATE MODEL 操作
- EXPLAIN_MODEL 函数

有关机器学习的更多信息，请参阅以下文档：

- 机器学习概览
- 面向新手和专家的机器学习
- 机器学习预测的公平性和模型可解释性是什么？

教程：使用线性学习器构建回归模型

在本教程中，您使用来自 Amazon S3 的数据创建线性学习器模型，并使用 Amazon Redshift ML 对模型运行预测查询。SageMaker 线性学习器算法可解决回归或多类别分类问题。要了解有关回归和多类别分类问题的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的机器学习范式的问题类型。在本教程中，您将解决一个回归问题。线性学习器算法并行训练许多模型，并自动确定最优化的模型。您可以在 Amazon Redshift 中使用 CREATE MODEL 操作，该操作使用 SageMaker 创建线性学习器模型，并将预测函数发送到 Amazon Redshift。有关线性学习器算法的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的线性学习器算法。

您可以使用 CREATE MODEL 命令导出训练数据、训练模型、导入模型以及准备 Amazon Redshift 预测函数。使用 CREATE MODEL 操作将训练数据指定为表或 SELECT 语句。

线性学习器模型可以优化连续目标或离散目标。连续目标用于回归，而离散变量用于分类。一些方法仅为连续目标提供解决方案，例如回归方法。线性学习器算法提供了比朴素超参数优化技术（如朴素贝叶斯技术）更快的速度。朴素优化技术假定每个输入变量都是独立的。要使用线性学习器算法，必须提供表示输入维度的列和表示观察值的行。有关线性学习器算法的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的线性学习器算法。

在本教程中，您将构建预测鲍鱼年龄的线性学习器模型。您可以对鲍鱼数据集使用 CREATE MODEL 命令，以确定鲍鱼的物理测量值之间的关系。然后，您可以使用该模型来确定鲍鱼的年龄。

使用案例示例

您可以使用线性学习器和 Amazon Redshift ML 解决其他回归问题，例如预测房屋价格。也可以使用 Redshift ML 来预测将使用城市自行车租赁服务的人数。

任务

- 先决条件
- 步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift
- 步骤 2：创建机器学习模型
- 步骤 3：验证模型

先决条件

要完成此教程，必须完成 Amazon Redshift ML 的管理设置。

步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift

使用 Amazon Redshift 查询器 v2 运行以下查询。这些查询将示例数据加载到 Redshift 中，然后将数据划分为训练集和验证集。

1. 以下查询将创建 abalone_dataset 表。

```
CREATE TABLE abalone_dataset (
    id INT IDENTITY(1, 1),
    Sex CHAR(1),
    Length float,
    Diameter float,
    Height float,
    Whole float,
    Shucked float,
    Viscera float,
    Shell float,
    Rings integer
);
```

2. 下面的查询将 Amazon S3 的鲍鱼数据集中的示例数据复制到您之前在 Amazon Redshift 中创建的 abalone_dataset 表中。

```
COPY abalone_dataset
FROM
```

```
    's3://redshift-ml-multiclass/abalone.csv' REGION 'us-east-1' IAM_ROLE default CSV
 IGNOREHEADER 1 NULL AS 'NULL';
```

3. 通过手动拆分数据，您将能够通过分配额外的预测集来验证模型的准确性。以下查询将数据拆分为两个集。abalone_training 表用于训练，abalone_validation 表用于验证。

```
CREATE TABLE abalone_training as
SELECT
    *
FROM
    abalone_dataset
WHERE
    mod(id, 10) < 8;

CREATE TABLE abalone_validation as
SELECT
    *
FROM
    abalone_dataset
WHERE
    mod(id, 10) >= 8;
```

步骤 2：创建机器学习模型

在此步骤中，您将使用 CREATE MODEL 语句，通过线性学习器算法创建机器学习模型。

以下查询使用您的 S3 桶通过 CREATE MODEL 操作创建线性学习器模型。将 *DOC-EXAMPLE-BUCKET* 替换为您自己的 S3 桶。

```
CREATE MODEL model_abalone_ring_prediction
FROM
    (
        SELECT
            Sex,
            Length,
            Diameter,
            Height,
            Whole,
            Shucked,
            Viscera,
            Shell,
            Rings AS target_label
```

```
        FROM
            abalone_training
    ) TARGET target_label FUNCTION f_abalone_ring_prediction IAM_ROLE default
 MODEL_TYPE LINEAR_LEARNER PROBLEM_TYPE REGRESSION OBJECTIVE 'MSE' SETTINGS (
        S3_BUCKET 'DOC-EXAMPLE-BUCKET',
        MAX_RUNTIME 15000
    );
```

显示模型训练的状态（可选）

您可以使用 SHOW MODEL 命令来了解模型何时准备就绪。

使用以下查询监控模型训练的进度。

```
SHOW MODEL model_abalone_ring_prediction;
```

模型准备就绪后，上一个操作的输出内容应类似于以下示例。请注意，输出提供了 validation:mse 指标，这是均方误差。在下一个步骤中，您将使用均方误差验证模型的准确性。

```
+--------------------------
+-----------------------------------------------------------------------------------------------------
+
|        Model Name        |
 model_abalone_ring_prediction                                    |
+--------------------------
+-----------------------------------------------------------------------------------------------------
+
| Schema Name              | public
                                        |
| Owner                    | awsuser
                                        |
| Creation Time            | Thu, 30.06.2022 18:00:10
                                        |
| Model State              | READY
                                        |
| validation:mse           |
                                   4.168633 |
| Estimated Cost           |
                                   4.291608 |
|                          |
                                        |
| TRAINING DATA:           |
                                        |
```

```
| Query                     | SELECT SEX , LENGTH , DIAMETER , HEIGHT , WHOLE ,
 SHUCKED , VISCERA , SHELL, RINGS AS TARGET_LABEL |
|                           | FROM ABALONE_TRAINING
                                          |
| Target Column             | TARGET_LABEL
                                          |
|                           |
                                          |
| PARAMETERS:               |
                                          |
| Model Type                | linear_learner
                                          |
| Problem Type              | Regression
                                          |
| Objective                 | MSE
                                          |
| AutoML Job Name           | redshiftml-20220630180010947843
                                          |
| Function Name             | f_abalone_ring_prediction
                                          |
| Function Parameters       | sex length diameter height whole shucked viscera shell
                                          |
| Function Parameter Types | bpchar float8 float8 float8 float8 float8 float8 float8
                                          |
| IAM Role                  | default-aws-iam-role
                                          |
| S3 Bucket                 | DOC-EXAMPLE-BUCKET
                                          |
| Max Runtime               |
                                 15000 |
+--------------------------
+-----------------------------------------------------------------------------------------------
+
```

步骤 3：验证模型

1. 以下预测查询通过计算均方误差和均方根误差来验证模型对于 abalone_validation 数据集的准确性。

```
SELECT
    ROUND(AVG(POWER((tgt_label - predicted), 2)), 2) mse,
    ROUND(SQRT(AVG(POWER((tgt_label - predicted), 2))), 2) rmse
FROM
```

```
    (
        SELECT
            Sex,
            Length,
            Diameter,
            Height,
            Whole,
            Shucked,
            Viscera,
            Shell,
            Rings AS tgt_label,
            f_abalone_ring_prediction(
                Sex,
                Length,
                Diameter,
                Height,
                Whole,
                Shucked,
                Viscera,
                Shell
            ) AS predicted,
            CASE
                WHEN tgt_label = predicted then 1
                ELSE 0
            END AS match,
            CASE
                WHEN tgt_label <> predicted then 1
                ELSE 0
            END AS nonmatch
        FROM
            abalone_validation
    ) t1;
```

上一个查询的输出应类似于以下示例。均方误差指标的值应类似于由 SHOW MODEL 操作的输出所显示的 validation:mse 指标。

```
+-----+--------------------+
| mse |        rmse        |
+-----+--------------------+
| 5.1 | 2.2600000000000002 |
+-----+--------------------+
```

2. 使用以下查询对预测函数运行 EXPLAIN_MODEL 操作。该操作将返回模型可解释性报告。有关 EXPLAIN_MODEL 操作的更多信息，请参阅《Amazon Redshift 数据库开发人员指南》中的 EXPLAIN_MODEL 函数。

```
SELECT
    EXPLAIN_MODEL ('model_abalone_ring_prediction');
```

以下信息是之前的 EXPLAIN_MODEL 操作生成的模型可解释性报告的示例。每个输入的值都是 Shapley 值。Shapley 值表示每个输入对模型预测的影响，值较高的输入对预测的影响更大。在此示例中，值较高的输入对预测鲍鱼年龄的影响更大。

```
{
    "explanations": {
        "kernel_shap": {
            "label0": {
                "expected_value" :10.290688514709473,
                "global_shap_values": {
                    "diameter" :0.6856910187882492,
                    "height" :0.4415323937124035,
                    "length" :0.21507476107609084,
                    "sex" :0.448611774505744,
                    "shell" :1.70426496893776,
                    "shucked" :2.1181392924386994,
                    "viscera" :0.342220754059912,
                    "whole" :0.6711906974084011
                }
            }
        }
    },
    "version" :"1.0"
};
```

3. 使用以下查询计算模型对尚未成熟的鲍鱼进行正确预测的百分比。未成熟的鲍鱼有 10 个或更少的环，正确的预测精确到实际环数的一个环内。

```
SELECT
    TRUNC(
        SUM(
            CASE
                WHEN ROUND(
                    f_abalone_ring_prediction(
```

```
                        Sex,
                        Length,
                        Diameter,
                        Height,
                        Whole,
                        Shucked,
                        Viscera,
                        Shell
                    ),
                    0
                ) BETWEEN Rings - 1
                AND Rings + 1 THEN 1
                ELSE 0
            END
        ) / CAST(COUNT(SHELL) AS FLOAT),
        4
    ) AS prediction_pct
FROM
    abalone_validation
WHERE
    Rings <= 10;
```

相关主题

有关 Amazon Redshift ML 的更多信息，请参阅以下文档：

- 使用 Amazon Redshift ML 的成本
- CREATE MODEL 操作
- EXPLAIN_MODEL 函数

有关机器学习的更多信息，请参阅以下文档：

- 机器学习概览
- 面向新手和专家的机器学习
- 机器学习预测的公平性和模型可解释性是什么？

教程：使用线性学习器构建多类别分类模型

在本教程中，您使用来自 Amazon S3 的数据创建线性学习器模型，然后使用 Amazon Redshift ML 对模型运行预测查询。SageMaker 线性学习器算法可解决回归或分类问题。要了解有关回归和多类别分类问题的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的机器学习范式的问题类型。在本教程中，您将解决一个多类别分类问题。线性学习器算法并行训练许多模型，并自动确定最优化的模型。您可以在 Amazon Redshift 中使用 CREATE MODEL 操作，该操作使用 SageMaker 创建线性学习器模型，并将预测函数发送到 Amazon Redshift。有关线性学习器算法的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的线性学习器算法。

您可以使用 CREATE MODEL 命令导出训练数据、训练模型、导入模型以及准备 Amazon Redshift 预测函数。使用 CREATE MODEL 操作将训练数据指定为表或 SELECT 语句。

线性学习器模型可以优化连续目标或离散目标。连续目标用于回归，而离散变量用于分类。一些方法仅为连续目标提供解决方案，例如回归方法。线性学习器算法提供了比朴素超参数优化技术（如朴素贝叶斯技术）更快的速度。朴素优化技术假定每个输入变量都是独立的。线性学习器算法并行训练许多模型，并选择最优化的模型。一种类似的算法是 XGBoost，它将来自一组更简单和更弱模型的估计值结合起来进行预测。要了解有关 XGBoost 的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的 XGBoost 算法。

要使用线性学习器算法，必须提供表示输入维度的列和表示观察值的行。有关线性学习器算法的更多信息，请参阅《Amazon SageMaker 开发人员指南》中的线性学习器算法。

在本教程中，您将构建一个线性学习器模型，用于预测给定区域的覆盖类型。对 UCI 机器学习存储库中的覆盖类型数据集使用 CREATE MODEL 命令。然后，您可以使用由该命令创建的预测函数来确定荒野区域的覆盖类型。森林覆盖类型通常是一种树木。Redshift ML 将用于创建模型的输入包括土壤类型、到道路的距离和荒野区域指定范围。有关数据集的更多信息，请参阅 UCI 机器学习存储库中的覆盖类型数据集。

使用案例示例

您可以使用线性学习器及 Amazon Redshift ML 解决其他多类别分类问题，例如从图像中预测植物的种类。您还可以预测客户将要购买的商品的数量。

任务

- 先决条件
- 步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift
- 步骤 2：创建机器学习模型
- 步骤 3：验证模型

先决条件

要完成此教程，必须完成 Amazon Redshift ML 的管理设置。

步骤 1：将数据从 Amazon S3 加载到 Amazon Redshift

使用 Amazon Redshift 查询器 v2 运行以下查询。这些查询将示例数据加载到 Redshift 中，然后将数据划分为训练集和验证集。

1. 以下查询将创建 covertype_data 表。

```
CREATE TABLE public.covertype_data (
    elevation bigint ENCODE az64,
    aspect bigint ENCODE az64,
    slope bigint ENCODE az64,
    horizontal_distance_to_hydrology bigint ENCODE az64,
    vertical_distance_to_hydrology bigint ENCODE az64,
    horizontal_distance_to_roadways bigint ENCODE az64,
    hillshade_9am bigint ENCODE az64,
    hillshade_noon bigint ENCODE az64,
    hillshade_3pm bigint ENCODE az64,
    horizontal_distance_to_fire_points bigint ENCODE az64,
    wilderness_area1 bigint ENCODE az64,
    wilderness_area2 bigint ENCODE az64,
    wilderness_area3 bigint ENCODE az64,
    wilderness_area4 bigint ENCODE az64,
    soil_type1 bigint ENCODE az64,
    soil_type2 bigint ENCODE az64,
    soil_type3 bigint ENCODE az64,
    soil_type4 bigint ENCODE az64,
    soil_type5 bigint ENCODE az64,
    soil_type6 bigint ENCODE az64,
    soil_type7 bigint ENCODE az64,
    soil_type8 bigint ENCODE az64,
    soil_type9 bigint ENCODE az64,
    soil_type10 bigint ENCODE az64,
    soil_type11 bigint ENCODE az64,
    soil_type12 bigint ENCODE az64,
    soil_type13 bigint ENCODE az64,
    soil_type14 bigint ENCODE az64,
    soil_type15 bigint ENCODE az64,
    soil_type16 bigint ENCODE az64,
    soil_type17 bigint ENCODE az64,
    soil_type18 bigint ENCODE az64,
```

```
    soil_type19 bigint ENCODE az64,
    soil_type20 bigint ENCODE az64,
    soil_type21 bigint ENCODE az64,
    soil_type22 bigint ENCODE az64,
    soil_type23 bigint ENCODE az64,
    soil_type24 bigint ENCODE az64,
    soil_type25 bigint ENCODE az64,
    soil_type26 bigint ENCODE az64,
    soil_type27 bigint ENCODE az64,
    soil_type28 bigint ENCODE az64,
    soil_type29 bigint ENCODE az64,
    soil_type30 bigint ENCODE az64,
    soil_type31 bigint ENCODE az64,
    soil_type32 bigint ENCODE az64,
    soil_type33 bigint ENCODE az64,
    soil_type34 bigint ENCODE az64,
    soil_type35 bigint ENCODE az64,
    soil_type36 bigint ENCODE az64,
    soil_type37 bigint ENCODE az64,
    soil_type38 bigint ENCODE az64,
    soil_type39 bigint ENCODE az64,
    soil_type40 bigint ENCODE az64,
    cover_type bigint ENCODE az64
) DISTSTYLE AUTO;
```

2. 下面的查询将 Amazon S3 的覆盖类型数据集中的示例数据复制到您之前在 Amazon Redshift 中创建的 covertype_data 表中。

```
COPY public.covertype_data
FROM
    's3://redshift-ml-multiclass/covtype.data.gz' IAM_ROLE DEFAULT gzip DELIMITER ','
 REGION 'us-east-1';
```

3. 通过手动拆分数据，您将能够通过分配额外的测试集来验证模型的准确性。以下查询将数据拆分为三个集。covertype_training 表用于训练，covertype_validation 表用于验证，而 covertype_test 表用于测试模型。您将使用训练集来训练模型，并使用验证集来验证模型的开发。然后，您可以使用测试集来测试模型的性能，并查看模型对数据集是过拟合还是欠拟合。

```
CREATE TABLE public.covertype_data_prep AS
SELECT
    a.*,
    CAST (random() * 100 AS int) AS data_group_id
FROM
```

```
    public.covertype_data a;

--training dataset
 CREATE TABLE public.covertype_training as
SELECT
    *
FROM
    public.covertype_data_prep
WHERE
    data_group_id < 80;

--validation dataset
 CREATE TABLE public.covertype_validation AS
SELECT
    *
FROM
    public.covertype_data_prep
WHERE
    data_group_id BETWEEN 80
    AND 89;

--test dataset
 CREATE TABLE public.covertype_test AS
SELECT
    *
FROM
    public.covertype_data_prep
WHERE
    data_group_id > 89;
```

步骤 2：创建机器学习模型

在此步骤中，您将使用 CREATE MODEL 语句，通过线性学习器算法创建机器学习模型。

以下查询使用您的 S3 桶通过 CREATE MODEL 操作创建线性学习器模型。将 *DOC-EXAMPLE-BUCKET* 替换为您自己的 S3 桶。

```
CREATE MODEL forest_cover_type_model
FROM
    (
        SELECT
            Elevation,
            Aspect,
```

```
            Slope,
            Horizontal_distance_to_hydrology,
            Vertical_distance_to_hydrology,
            Horizontal_distance_to_roadways,
            HIllshade_9am,
            Hillshade_noon,
            Hillshade_3pm,
            Horizontal_Distance_To_Fire_Points,
            Wilderness_Area1,
            Wilderness_Area2,
            Wilderness_Area3,
            Wilderness_Area4,
            soil_type1,
            Soil_Type2,
            Soil_Type3,
            Soil_Type4,
            Soil_Type5,
            Soil_Type6,
            Soil_Type7,
            Soil_Type8,
            Soil_Type9,
            Soil_Type10,
            Soil_Type11,
            Soil_Type12,
            Soil_Type13,
            Soil_Type14,
            Soil_Type15,
            Soil_Type16,
            Soil_Type17,
            Soil_Type18,
            Soil_Type19,
            Soil_Type20,
            Soil_Type21,
            Soil_Type22,
            Soil_Type23,
            Soil_Type24,
            Soil_Type25,
            Soil_Type26,
            Soil_Type27,
            Soil_Type28,
            Soil_Type29,
            Soil_Type30,
            Soil_Type31,
            Soil_Type32,
```

```
            Soil_Type33,
            Soil_Type34,
            Soil_Type36,
            Soil_Type37,
            Soil_Type38,
            Soil_Type39,
            Soil_Type40,
            Cover_type
        from
            public.covertype_training
    ) TARGET cover_type FUNCTION predict_cover_type IAM_ROLE default MODEL_TYPE
 LINEAR_LEARNER PROBLEM_TYPE MULTICLASS_CLASSIFICATION OBJECTIVE 'Accuracy' SETTINGS (
        S3_BUCKET '<DOC-EXAMPLE-BUCKET>',
        S3_GARBAGE_COLLECT OFF,
        MAX_RUNTIME 15000
    );
```

显示模型训练的状态（可选）

您可以使用 SHOW MODEL 命令来了解模型何时准备就绪。

使用以下查询监控模型训练的进度。

```
SHOW MODEL forest_cover_type_model;
```

模型准备就绪后，上一个操作的输出内容应类似于以下示例。请注意，输出提供了 validation:multiclass_accuracy 指标，您可以在以下示例的右侧查看该指标。多类别准确性用于衡量由模型正确分类的数据点的百分比。在下一步中，您将使用多类别准确性来验证模型的准确性。

```
+--------------------------------
+-------------------------------------------------------------------------------------------------------------
+
|              Key               |




 Value


                                                          |
```

```
+--------------------------------
+-------------------------------------------------------------------------------------------------------------------------------
+
| Model Name                     | forest_cover_type_model

                                                                                                                                |
| Schema Name                    | public

                                                                                                                                |
| Owner                          | awsuser

                                                                                                                                |
| Creation Time                  | Tue, 12.07.2022 20:24:32

                                                                                                                                |
```

```
| Model State                     | READY

                                                                |
| validation:multiclass_accuracy |

                                                                0.724952 |
| Estimated Cost                  |

                                                                5.341750 |
|                                 |

                                                                |
| TRAINING DATA:                  |
```

```
                                                    |
| Query                          | SELECT ELEVATION, ASPECT, SLOPE,
 HORIZONTAL_DISTANCE_TO_HYDROLOGY, VERTICAL_DISTANCE_TO_HYDROLOGY,
 HORIZONTAL_DISTANCE_TO_ROADWAYS, HILLSHADE_9AM, HILLSHADE_NOON, HILLSHADE_3PM ,
 HORIZONTAL_DISTANCE_TO_FIRE_POINTS, WILDERNESS_AREA1, WILDERNESS_AREA2,
 WILDERNESS_AREA3, WILDERNESS_AREA4, SOIL_TYPE1, SOIL_TYPE2, SOIL_TYPE3, SOIL_TYPE4,
 SOIL_TYPE5, SOIL_TYPE6, SOIL_TYPE7, SOIL_TYPE8, SOIL_TYPE9, SOIL_TYPE10 , SOIL_TYPE11,
 SOIL_TYPE12 , SOIL_TYPE13 , SOIL_TYPE14, SOIL_TYPE15, SOIL_TYPE16, SOIL_TYPE17,
 SOIL_TYPE18, SOIL_TYPE19, SOIL_TYPE20, SOIL_TYPE21, SOIL_TYPE22, SOIL_TYPE23,
 SOIL_TYPE24, SOIL_TYPE25, SOIL_TYPE26, SOIL_TYPE27, SOIL_TYPE28, SOIL_TYPE29,
 SOIL_TYPE30, SOIL_TYPE31, SOIL_TYPE32, SOIL_TYPE33, SOIL_TYPE34, SOIL_TYPE36,
 SOIL_TYPE37, SOIL_TYPE38, SOIL_TYPE39, SOIL_TYPE40, COVER_TYPE |
|                                | FROM PUBLIC.COVERTYPE_TRAINING
```

```
                                                    |
| Target Column                  | COVER_TYPE
```

```
                                                    |
|                                |
```

```
                                                    |
| PARAMETERS:                    |

                                                    |
| Model Type                     | linear_learner

                                                    |
| Problem Type                   | MulticlassClassification

                                                    |
| Objective                      | Accuracy

                                                    |
```

```
| AutoML Job Name                | redshiftml-20220712202432187659
                                                                  |
| Function Name                  | predict_cover_type
                                                                  |
| Function Parameters            | elevation aspect slope
 horizontal_distance_to_hydrology vertical_distance_to_hydrology
 horizontal_distance_to_roadways hillshade_9am hillshade_noon hillshade_3pm
 horizontal_distance_to_fire_points wilderness_area1 wilderness_area2 wilderness_area3
 wilderness_area4 soil_type1 soil_type2 soil_type3 soil_type4 soil_type5 soil_type6
 soil_type7 soil_type8 soil_type9 soil_type10 soil_type11 soil_type12 soil_type13
 soil_type14 soil_type15 soil_type16 soil_type17 soil_type18 soil_type19 soil_type20
 soil_type21 soil_type22 soil_type23 soil_type24 soil_type25 soil_type26 soil_type27
 soil_type28 soil_type29 soil_type30 soil_type31 soil_type32 soil_type33 soil_type34
 soil_type36 soil_type37 soil_type38 soil_type39 soil_type40
                                                                 |
| Function Parameter Types       | int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8
                                                                       |
| IAM Role                       | default-aws-iam-role
```

```
                                                    |
| S3 Bucket                        | DOC-EXAMPLE-BUCKET

                                                    |
| Max Runtime                      |

                                                15000 |
+---------------------------------
+-----------------------------------------------------------------------------------------------------------
+
```

步骤 3：验证模型

1. 以下预测查询通过计算多类别准确性来验证模型对于 covertype_validation 数据集的准确性。多类别准确性是模型的预测中属于正确预测的百分比。

```
SELECT
    CAST(sum(t1.match) AS decimal(7, 2)) AS predicted_matches,
    CAST(sum(t1.nonmatch) AS decimal(7, 2)) AS predicted_non_matches,
    CAST(sum(t1.match + t1.nonmatch) AS decimal(7, 2)) AS total_predictions,
    predicted_matches / total_predictions AS pct_accuracy
FROM
    (
```

```
        SELECT
            Elevation,
            Aspect,
            Slope,
            Horizontal_distance_to_hydrology,
            Vertical_distance_to_hydrology,
            Horizontal_distance_to_roadways,
            HIllshade_9am,
            Hillshade_noon,
            Hillshade_3pm,
            Horizontal_Distance_To_Fire_Points,
            Wilderness_Area1,
            Wilderness_Area2,
            Wilderness_Area3,
            Wilderness_Area4,
            soil_type1,
            Soil_Type2,
            Soil_Type3,
            Soil_Type4,
            Soil_Type5,
            Soil_Type6,
            Soil_Type7,
            Soil_Type8,
            Soil_Type9,
            Soil_Type10,
            Soil_Type11,
            Soil_Type12,
            Soil_Type13,
            Soil_Type14,
            Soil_Type15,
            Soil_Type16,
            Soil_Type17,
            Soil_Type18,
            Soil_Type19,
            Soil_Type20,
            Soil_Type21,
            Soil_Type22,
            Soil_Type23,
            Soil_Type24,
            Soil_Type25,
            Soil_Type26,
            Soil_Type27,
            Soil_Type28,
            Soil_Type29,
```

```
            Soil_Type30,
            Soil_Type31,
            Soil_Type32,
            Soil_Type33,
            Soil_Type34,
            Soil_Type36,
            Soil_Type37,
            Soil_Type38,
            Soil_Type39,
            Soil_Type40,
            Cover_type AS actual_cover_type,
            predict_cover_type(
                Elevation,
                Aspect,
                Slope,
                Horizontal_distance_to_hydrology,
                Vertical_distance_to_hydrology,
                Horizontal_distance_to_roadways,
                HIllshade_9am,
                Hillshade_noon,
                Hillshade_3pm,
                Horizontal_Distance_To_Fire_Points,
                Wilderness_Area1,
                Wilderness_Area2,
                Wilderness_Area3,
                Wilderness_Area4,
                soil_type1,
                Soil_Type2,
                Soil_Type3,
                Soil_Type4,
                Soil_Type5,
                Soil_Type6,
                Soil_Type7,
                Soil_Type8,
                Soil_Type9,
                Soil_Type10,
                Soil_Type11,
                Soil_Type12,
                Soil_Type13,
                Soil_Type14,
                Soil_Type15,
                Soil_Type16,
                Soil_Type17,
                Soil_Type18,
```

```
                Soil_Type19,
                Soil_Type20,
                Soil_Type21,
                Soil_Type22,
                Soil_Type23,
                Soil_Type24,
                Soil_Type25,
                Soil_Type26,
                Soil_Type27,
                Soil_Type28,
                Soil_Type29,
                Soil_Type30,
                Soil_Type31,
                Soil_Type32,
                Soil_Type33,
                Soil_Type34,
                Soil_Type36,
                Soil_Type37,
                Soil_Type38,
                Soil_Type39,
                Soil_Type40
            ) AS predicted_cover_type,
            CASE
                WHEN actual_cover_type = predicted_cover_type THEN 1
                ELSE 0
            END AS match,
            CASE
                WHEN actual_cover_type <> predicted_cover_type THEN 1
                ELSE 0
            END AS nonmatch
        FROM
            public.covertype_validation
    ) t1;
```

上一个查询的输出应类似于以下示例。多类别准确性指标的值应类似于由 SHOW MODEL 操作的输出所显示的 validation:multiclass_accuracy 指标。

```
+-------------------+-----------------------+-------------------+--------------+
| predicted_matches | predicted_non_matches | total_predictions | pct_accuracy |
+-------------------+-----------------------+-------------------+--------------+
|             41211 |                 16324 |             57535 |   0.71627704 |
+-------------------+-----------------------+-------------------+--------------+
```

2. 下面的查询预测了 wilderness_area2 的最常见覆盖类型。该数据集包括四个荒野区域和七种覆盖类型。荒野区域可以有多种覆盖类型。

```
SELECT t1. predicted_cover_type, COUNT(*)
FROM
(
SELECT
   Elevation,
   Aspect,
   Slope,
   Horizontal_distance_to_hydrology,
   Vertical_distance_to_hydrology,
   Horizontal_distance_to_roadways,
   HIllshade_9am,
   Hillshade_noon,
   Hillshade_3pm ,
   Horizontal_Distance_To_Fire_Points,
   Wilderness_Area1,
   Wilderness_Area2,
   Wilderness_Area3,
   Wilderness_Area4,
   soil_type1,
   Soil_Type2,
   Soil_Type3,
   Soil_Type4,
   Soil_Type5,
   Soil_Type6,
   Soil_Type7,
   Soil_Type8,
   Soil_Type9,
   Soil_Type10 ,
   Soil_Type11,
   Soil_Type12 ,
   Soil_Type13 ,
   Soil_Type14,
   Soil_Type15,
   Soil_Type16,
   Soil_Type17,
   Soil_Type18,
   Soil_Type19,
   Soil_Type20,
   Soil_Type21,
   Soil_Type22,
```

```
    Soil_Type23,
    Soil_Type24,
    Soil_Type25,
    Soil_Type26,
    Soil_Type27,
    Soil_Type28,
    Soil_Type29,
    Soil_Type30,
    Soil_Type31,
    Soil_Type32,
    Soil_Type33,
    Soil_Type34,
    Soil_Type36,
    Soil_Type37,
    Soil_Type38,
    Soil_Type39,
    Soil_Type40,
    predict_cover_type( Elevation,
    Aspect,
    Slope,
    Horizontal_distance_to_hydrology,
    Vertical_distance_to_hydrology,
    Horizontal_distance_to_roadways,
    HIllshade_9am,
    Hillshade_noon,
    Hillshade_3pm ,
    Horizontal_Distance_To_Fire_Points,
    Wilderness_Area1,
    Wilderness_Area2,
    Wilderness_Area3,
    Wilderness_Area4,
    soil_type1,
    Soil_Type2,
    Soil_Type3,
    Soil_Type4,
    Soil_Type5,
    Soil_Type6,
    Soil_Type7,
    Soil_Type8,
    Soil_Type9,
    Soil_Type10,
    Soil_Type11,
    Soil_Type12,
    Soil_Type13,
```

```
   Soil_Type14,
   Soil_Type15,
   Soil_Type16,
   Soil_Type17,
   Soil_Type18,
   Soil_Type19,
   Soil_Type20,
   Soil_Type21,
   Soil_Type22,
   Soil_Type23,
   Soil_Type24,
   Soil_Type25,
   Soil_Type26,
   Soil_Type27,
   Soil_Type28,
   Soil_Type29,
   Soil_Type30,
   Soil_Type31,
   Soil_Type32,
   Soil_Type33,
   Soil_Type34,
   Soil_Type36,
   Soil_Type37,
   Soil_Type38,
   Soil_Type39,
   Soil_Type40) AS predicted_cover_type

FROM public.covertype_test
WHERE wilderness_area2 = 1)
t1
GROUP BY 1;
```

前一个操作的输出内容应类似如下示例。这一输出意味着，该模型预测大部分覆盖为覆盖类型 1，并且存在一些覆盖类型为 2 和 7 的覆盖。

```
+----------------------+-------+
| predicted_cover_type | count |
+----------------------+-------+
|                    2 |   564 |
|                    7 |    97 |
|                    1 |  2309 |
+----------------------+-------+
```

3. 以下查询显示了单个荒野区域中最常见的覆盖类型。该查询将显示该覆盖类型的数量和覆盖类型的荒野区域。

```
SELECT t1. predicted_cover_type, COUNT(*), wilderness_area
FROM
(
SELECT
   Elevation,
   Aspect,
   Slope,
   Horizontal_distance_to_hydrology,
   Vertical_distance_to_hydrology,
   Horizontal_distance_to_roadways,
   HIllshade_9am,
   Hillshade_noon,
   Hillshade_3pm ,
   Horizontal_Distance_To_Fire_Points,
   Wilderness_Area1,
   Wilderness_Area2,
   Wilderness_Area3,
   Wilderness_Area4,
   soil_type1,
   Soil_Type2,
   Soil_Type3,
   Soil_Type4,
   Soil_Type5,
   Soil_Type6,
   Soil_Type7,
   Soil_Type8,
   Soil_Type9,
   Soil_Type10 ,
   Soil_Type11,
   Soil_Type12 ,
   Soil_Type13 ,
   Soil_Type14,
   Soil_Type15,
   Soil_Type16,
   Soil_Type17,
   Soil_Type18,
   Soil_Type19,
   Soil_Type20,
   Soil_Type21,
   Soil_Type22,
```

```
    Soil_Type23,
    Soil_Type24,
    Soil_Type25,
    Soil_Type26,
    Soil_Type27,
    Soil_Type28,
    Soil_Type29,
    Soil_Type30,
    Soil_Type31,
    Soil_Type32,
    Soil_Type33,
    Soil_Type34,
    Soil_Type36,
    Soil_Type37,
    Soil_Type38,
    Soil_Type39,
    Soil_Type40,
    predict_cover_type( Elevation,
    Aspect,
    Slope,
    Horizontal_distance_to_hydrology,
    Vertical_distance_to_hydrology,
    Horizontal_distance_to_roadways,
    HIllshade_9am,
    Hillshade_noon,
    Hillshade_3pm ,
    Horizontal_Distance_To_Fire_Points,
    Wilderness_Area1,
    Wilderness_Area2,
    Wilderness_Area3,
    Wilderness_Area4,
    soil_type1,
    Soil_Type2,
    Soil_Type3,
    Soil_Type4,
    Soil_Type5,
    Soil_Type6,
    Soil_Type7,
    Soil_Type8,
    Soil_Type9,
    Soil_Type10,
    Soil_Type11,
    Soil_Type12,
    Soil_Type13,
```

```
   Soil_Type14,
   Soil_Type15,
   Soil_Type16,
   Soil_Type17,
   Soil_Type18,
   Soil_Type19,
   Soil_Type20,
   Soil_Type21,
   Soil_Type22,
   Soil_Type23,
   Soil_Type24,
   Soil_Type25,
   Soil_Type26,
   Soil_Type27,
   Soil_Type28,
   Soil_Type29,
   Soil_Type30,
   Soil_Type31,
   Soil_Type32,
   Soil_Type33,
   Soil_Type34,
   Soil_Type36,
   Soil_Type37,
   Soil_Type38,
   Soil_Type39,
   Soil_Type40) AS predicted_cover_type,
   CASE WHEN Wilderness_Area1 = 1 THEN 1
        WHEN Wilderness_Area2 = 1 THEN 2
        WHEN Wilderness_Area3 = 1 THEN 3
        WHEN Wilderness_Area4 = 1 THEN 4
        ELSE 0
   END AS wilderness_area

FROM public.covertype_test)
t1
GROUP BY 1, 3
ORDER BY 2 DESC
LIMIT 1;
```

前一个操作的输出内容应类似如下示例。

```
+----------------------+-------+-----------------+
| predicted_cover_type | count | wilderness_area |
```

```
+----------------------+-------+-----------------+
|                    2 | 15738 |               1 |
+----------------------+-------+-----------------+
```

相关主题

有关 Amazon Redshift ML 的更多信息，请参阅以下文档：

- 使用 Amazon Redshift ML 的成本
- CREATE MODEL 操作
- EXPLAIN_MODEL 函数

有关机器学习的更多信息，请参阅以下文档：

- 机器学习概览
- 面向新手和专家的机器学习
- 机器学习预测的公平性和模型可解释性是什么？

优化查询性能

Amazon Redshift 使用基于结构化查询语言 (SQL) 的查询与系统中的数据和对象进行交互。数据操作语言 (DML) 是您用于查看、添加、更改和删除数据的 SQL 子集。数据定义语言 (DDL) 是您用于添加、更改和删除数据库对象（如表和视图）的 SQL 子集。

系统设置完毕后，您通常使用最多的是 DML，特别是用于检索和查看数据的 SELECT 命令。要在 Amazon Redshift 中编写高效的数据检索查询，请熟悉 SELECT 并应用设计表的 Amazon Redshift 最佳实践中概述的提示，以尽量提高查询效率。

要了解 Amazon Redshift 如何处理查询，请参阅查询处理和分析和改进查询两节。然后，您可以将此信息与诊断工具结合应用，以确定并消除查询性能方面的问题。

要确定和解决一些在使用 Amazon Redshift 查询时可能遇到的最常见问题和最严重问题，请参阅查询故障排除一节。

主题

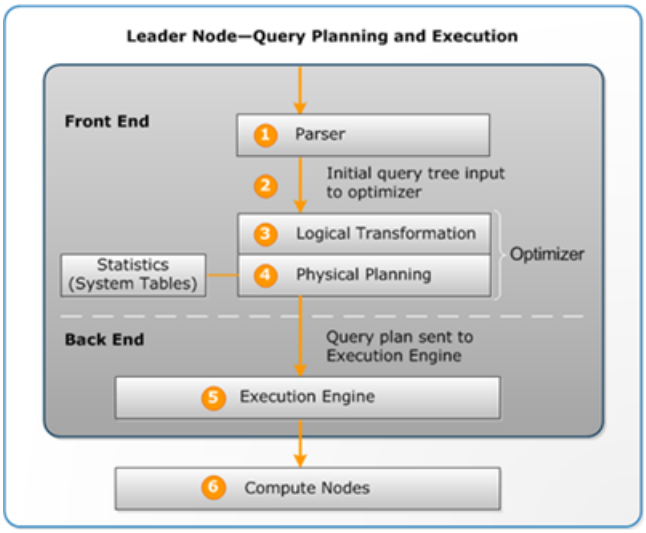
查询处理

Amazon Redshift 通过分析程序和优化程序路由提交的 SQL 查询，以制订查询计划。然后，执行引擎将查询计划转换为代码并将代码发送到计算节点执行。

主题

查询计划和执行工作流程

下图概要介绍查询计划和执行工作流程。



查询计划和执行工作流程遵循以下步骤：

1. 领导节点接收查询并解析 SQL。
2. 分析程序生成初步查询树，后者是原始查询的逻辑表示。然后，Amazon Redshift 将该查询树输入到查询优化程序中。
3. 优化程序评估并在必要时重写查询以最大限度地提高查询的效率。此过程有时会导致创建多个相关查询来替换单个查询。
4. 优化程序会生成一个查询计划以最佳性能进行执行（如果上一步导致多个查询，则生成多个查询计划）。查询计划指定执行选项，如联接类型、联接顺序、聚合选项和数据分配要求。

   您可以使用 EXPLAIN 命令查看查询计划。查询计划是分析和优化复杂查询的基本工具。有关更多信息，请参阅查询计划。
5. 执行引擎将查询计划转换为步骤、分段和流：

   步骤

   每个步骤都是查询执行过程中需要的单独操作。可以组合步骤以允许计算节点执行查询、联接或其他数据库操作。

   Segment

   可以通过单个进程完成的几个步骤的组合，也是由计算节点切片执行的最小编译单元。切片是 Amazon Redshift 中并行处理的单元。流中的分段并行运行。

   流

   要分配到可用计算节点切片上的分段的集合。

执行引擎基于步骤、段和流生成编译后的代码。编译的代码运行速度比解释的代码快，并且使用的计算容量更少。然后，将此编译的代码广播到计算节点。

> **Note**
>
> 在对查询进行基准测试时，您应始终比较查询第二次执行的时间，因为第一次执行时间包括编译代码的开销。有关更多信息，请参阅影响查询性能的因素。

6. 计算节点切片并行运行查询分段。在该流程中，Amazon Redshift 利用优化的网络通信、内存和磁盘管理，将中间结果从一个查询计划步骤传递到下一个。这还有助于加快查询执行速度。

步骤 5 和 6 针对每个流执行一次。引擎为一个流创建可执行分段并将其发送到计算节点。当该流的分段完成时，引擎会生成下一个流的分段。通过这种方式，引擎可以分析先前流中发生的情况（例如，操作是否基于磁盘），以影响下一个流中分段的生成。

计算节点完成后，它们会将查询结果返回到领导节点以进行最终处理。领导节点将数据合并到单个结果集中，并解决任何需要的排序或聚合。然后，领导节点将结果返回至客户端。

> **Note**
>
> 如有必要，计算节点可能会在查询执行期间将某些数据返回到领导节点。例如，如果您有一个包含 LIMIT 子句的子查询，则在数据在集群间重新分配以进一步处理前会在领导节点上应用该限制。

查询计划

您可以借助查询计划获取有关运行查询所需的各个操作的信息。在处理查询计划前，建议您先了解 Amazon Redshift 如何处理查询和如何创建查询计划。有关更多信息，请参阅查询计划和执行工作流程。

要创建查询计划，请运行 EXPLAIN 命令，后跟实际查询文本。查询计划提供以下信息：

- 执行引擎将执行的操作，自下而上地阅读结果。
- 每个操作执行的步骤的类型。
- 每个操作中使用的表和列。
- 每个操作中处理的数据量（以字节为单位），以行数和数据宽度计。

- 操作的相对成本。成本是比较计划内的步骤的相对执行时间的度量。成本不提供有关实际执行时间或内存消耗的任何精确信息，也没有提供执行计划之间的有意义的比较。它可以指示查询中的哪些操作消耗最多的资源。

EXPLAIN 命令不实际运行查询。它只显示当查询在当前操作条件下运行时 Amazon Redshift 将执行的计划。如果您更改表的 schema 或数据后再次运行 ANALYZE 以更新统计元数据，则查询计划可能会不同。

EXPLANE 的查询计划输出是查询执行的简化高级视图。它不描述并行查询处理的详细信息。要查看详细信息，请运行查询本身，然后从 SVL_QUERY_SUMMARY 或 SVL_QUERY_REPORT 视图获取查询摘要信息。有关使用这些视图的更多信息，请参阅分析查询摘要。

以下示例显示 EVENT 表上的简单 GROUP BY 查询的 EXPLAIN 输出：

```
explain select eventname, count(*) from event group by eventname;

                            QUERY PLAN
------------------------------------------------------------------------
XN HashAggregate  (cost=131.97..133.41 rows=576 width=17)
  ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=17)
```

EXPLAIN 为每个操作返回以下指标：

费用

对于比较计划内的操作非常有用的相对值。成本由被两个圆点分隔的十进制值组成，例如 `cost=131.97..133.41`。第一个值（在本例中为 131.97）提供返回此操作的第一行的相对成本。第二个值（在本例中为 133.41）提供完成操作的相对成本。查询计划的成本在读取计划时进行累积，因此本示例中的 HashAggregate 成本 (131.97..133.41) 包括其下面的序列扫描的成本 (0.00..87.98)。

行数

要返回的估计行数。在此示例中，扫描预计将返回 8798 行。HashAggregate 运算符本身应返回 576 行（在从结果集中丢弃重复的事件名称之后）。

> **Note**
>
> 行数估算基于 ANALYZE 命令生成的可用统计数据。如果最近未运行过 ANALYZE，则估算的可靠性会降低。

宽度

平均行的估计宽度（以字节为单位）。在此示例中，平均行的宽度应为 17 个字节。

EXPLAIN 运算符

本节简要介绍了在 EXPLAIN 输出中最常见的运算符。有关运算符的完整列表，请参阅 SQL 命令部分中的EXPLAIN。

顺序扫描运算符

顺序扫描运算符 (Seq Scan) 指示表扫描。Seq Scan 扫描从开始到结束按顺序扫描表中的每一列，并计算每一行的查询约束（在 WHERE 子句中）。

联接运算符

Amazon Redshift 根据要联接的表的物理设计、联接所需的数据的位置以及查询本身的特定要求来选择联接运算符。

- 嵌套循环

  最优化程度最差的联接，即嵌套循环，主要用于交叉联接（笛卡尔积）和一些不等式联接。

- 哈希联接和哈希

  哈希联接和哈希的运行速度通常比签到循环快，可用于内部联接以及左和右外部联接。这些运算符在联接列不是分配键和排序键的情况下用于联接表。哈希运算符为联接中的内部表创建哈希表；哈希联接运算符读取外部表，对联接列进行哈希处理，然后在内部哈希表中查找匹配项。

- 合并联接

  合并联接通常是最快的连接，用于内联接和外联接。合并联接不用于完全联接。此运算符在联接列都是分配键和排序键的情况下，以及未排序的联接表少于 20% 时用于联接表。它按顺序读取两个排序表并查找匹配的行。要查看未排序行的百分比，请查询 SVV_TABLE_INFO 系统表。

- 空间联接

  通常是基于空间数据邻近度的快速联接，用于 GEOMETRY 和 GEOGRAPHY 数据类型。

聚合运算符

查询计划在涉及聚合函数和 GROUP BY 操作的查询中使用以下运算符。

- 聚合

  标量聚合函数（如 AVG 和 SUM）的运算符。
- HashAggregate

  未排序分组聚合函数的运算符。
- GroupAggregate

  已排序分组聚合函数的运算符。

排序运算符

当查询必须对结果集进行排序或合并时，查询计划使用以下运算符。

- 排序

  评估 ORDER BY 子句和其他排序操作，例如 UNION 查询和联接所需的排序、SELECT DISTINCT 查询和窗口函数。
- 合并

  根据从并行操作得到的临时排序结果，来生成最终排序结果。

UNION、INTERSECT 和 EXCEPT 运算符

查询计划将以下运算符用于涉及使用 UNION、INTERSECT 和 EXCEPT 进行集合操作的查询。

- Subquery

  用于运行 UNION 查询。
- Hash Intersect Distinct

  用于运行 INTERSECT 查询。
- SetOp Except

  用于运行 EXCEPT（或 MINUS）查询。

其他运算符

以下运算符也经常出现在例行查询的 EXPLAIN 输出中。

- 唯一

  消除 SELECT DISTINCT 查询和 UNION 查询的重复项。
- 限制

  处理 LIMIT 子句。
- 窗口

  运行窗口函数。
- 结果

  运行不涉及任何表访问的标量函数。
- 子计划

  用于特定的子查询。
- Network

  将临时结果发送到领导节点，以待进一步处理。
- 实体化

  保存嵌套循环联接和某些合并联接的输入中的行。

EXPLAIN 中的联接

查询优化程序使用不同的联接类型来检索表数据，具体取决于查询和基础表的结构。EXPLAIN 输出引用了联接类型、使用的表以及在集群中分布表数据的方式，以描述查询的处理方式。

联接类型示例

下面的示例显示了查询优化程序可以使用的不同联接类型。查询计划中使用的联接类型取决于所涉表的物理设计。

示例：对两个表进行哈希联接

以下查询在 CATID 列上将 EVENT 和 CATEGORY 联接起来。CATID 是 CATEGORY 的分配和排序键，但不适用于 EVENT。使用 EVENT 作为外部表并使用 CATEGORY 作为内部表来执行哈希联接。由于 CATEGORY 是较小的表，因此计划程序在查询处理过程中使用 DS_BCAST_INNER 将其副本广播到计算节点。此示例中的联接成本占计划累计成本的大部分。

```
explain select * from category, event where category.catid=event.catid;
```

```
                                QUERY PLAN
------------------------------------------------------------------------------
 XN Hash Join DS_BCAST_INNER  (cost=0.14..6600286.07 rows=8798 width=84)
   Hash Cond: ("outer".catid = "inner".catid)
   ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=35)
   ->  XN Hash  (cost=0.11..0.11 rows=11 width=49)
         ->  XN Seq Scan on category  (cost=0.00..0.11 rows=11 width=49)
```

> **Note**
>
> EXPLAIN 输出中运算符的缩进对齐有时表示这些操作不相互依赖，并且可以并行开始。在前面的示例中，尽管 EVENT 表上的扫描和哈希操作已对齐，但 EVENT 扫描必须等到哈希操作完全完成。

示例：对两个表进行合并联接

以下查询还使用 SELECT *，但它在 LISTID 列上联接 SALES 和 LISTING，其中 LISTID 已设置为两个表的分配和排序键。选择合并联接，并且不需要对联接重新分配数据 (DS_DIST_NONE)。

```
explain select * from sales, listing where sales.listid = listing.listid;
QUERY PLAN
------------------------------------------------------------------------------
XN Merge Join DS_DIST_NONE  (cost=0.00..6285.93 rows=172456 width=97)
  Merge Cond: ("outer".listid = "inner".listid)
  ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=44)
  ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=53)
```

以下示例演示了同一查询中的不同类型的联接。与前面的示例一样，SALES 和 LISTING 是合并联接的，但第三个表 EVENT 必须与合并联接的结果进行哈希联接。同样，哈希联接会产生广播成本。

```
explain select * from sales, listing, event
where sales.listid = listing.listid and sales.eventid = event.eventid;
                                QUERY PLAN
------------------------------------------------------------------------------
XN Hash Join DS_BCAST_INNER  (cost=109.98..3871130276.17 rows=172456 width=132)
  Hash Cond: ("outer".eventid = "inner".eventid)
  ->  XN Merge Join DS_DIST_NONE  (cost=0.00..6285.93 rows=172456 width=97)
        Merge Cond: ("outer".listid = "inner".listid)
        ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=44)
```

```
       ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=53)
  ->  XN Hash  (cost=87.98..87.98 rows=8798 width=35)
       ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=35)
```

示例：联接、聚合和排序

以下查询运行 SALES 和 EVENT 表的哈希联接，然后执行聚合和排序操作，以考虑分组 SUM 函数和 ORDER BY 子句。初始排序运算符在计算节点上并行运行。然后，Network 运算符将结果发送到领导节点，其中合并运算符生成最终的排序结果。

```
explain select eventname, sum(pricepaid) from sales, event
where sales.eventid=event.eventid group by eventname
order by 2 desc;
                                            QUERY PLAN
------------------------------------------------------------------------------
 XN Merge  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
  Merge Key: sum(sales.pricepaid)
  ->  XN Network  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
       Send to leader
       ->  XN Sort  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
             Sort Key: sum(sales.pricepaid)
             ->  XN HashAggregate  (cost=2815366577.07..2815366578.51 rows=576
 width=27)
                   ->  XN Hash Join DS_BCAST_INNER  (cost=109.98..2815365714.80
 rows=172456 width=27)
                         Hash Cond: ("outer".eventid = "inner".eventid)
                         ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456
 width=14)
                         ->  XN Hash  (cost=87.98..87.98 rows=8798 width=21)
                               ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798
 width=21)
```

数据重新分配

联接的 EXPLAIN 输出还指定在集群上移动数据以便进行联接的方法。数据移动可以采用广播方法或重新分配。在广播中，联接一侧的数据值将从每个计算节点复制到每个其他计算节点，以便每个计算节点最终得到数据的完整副本。在重新分配中，参与的数据值从其当前切片发送到新切片（可能位于不同节点上）。如果该分配键是联接列之一，则通常会重新分配数据以匹配参与联接的其他表的分配键。如果两个表在其中一个联接列上都没有分配键，则两个表都会被分配，或者内部表将广播到每个节点。

EXPLAIN 输出还引用内表和外表。首先扫描内部表，并显示在查询计划底部附近。内部表是用来探测匹配项的表。它通常保存在内存中，通常是哈希的源表，如果可能的话，是两者中较小的表。外部表是

要与内部表匹配的行的源。它通常是从磁盘读取的。查询优化程序根据最新运行的 ANALYZE 命令中的数据库统计信息选择内表和外表。查询的 FROM 子句中的表顺序并不区分内部表和外部表。

您可以通过查询计划中的以下属性了解数据的移动方式，以便执行查询：

- DS_BCAST_INNER

  将整个内部表的副本广播到所有计算节点。
- DS_DIST_ALL_NONE

  无需重新分配，因为内部表已经使用 DISTSTYLE ALL 被分配到每个节点。
- DS_DIST_NONE

  两个表都未重新分配。可以使用并置连接，因为相应的切片联接时不会在节点之间移动数据。
- DS_DIST_INNER

  内部表重新分配。
- DS_DIST_OUTER

  外部表重新分配。
- DS_DIST_ALL_INNER

  整个内部表重新分配到单个切片，因为外部表使用 DISTSTYLE ALL。
- DS_DIST_BOTH

  两个表都重新分配。

检查查询计划步骤

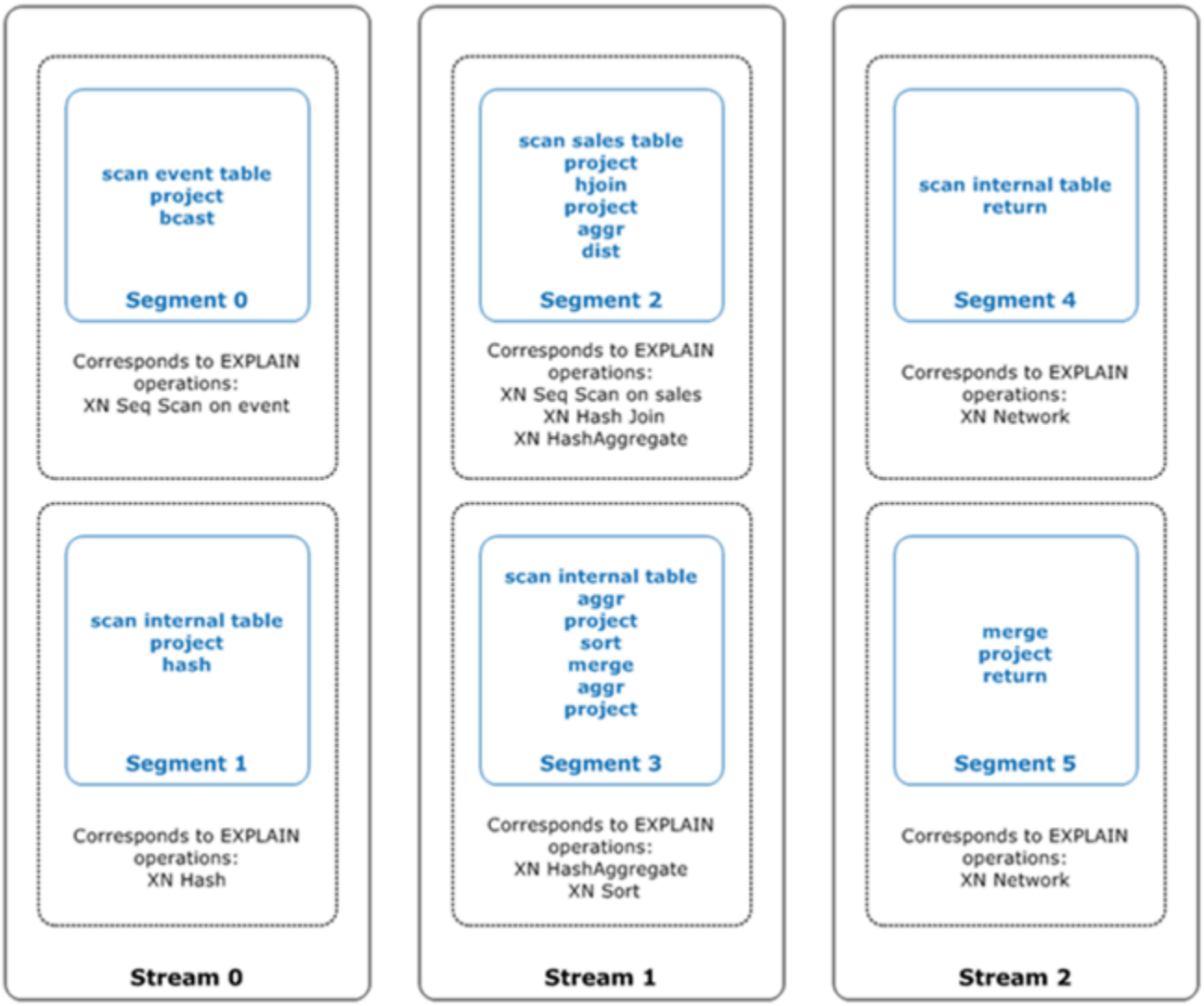
您可以通过运行 EXPLAIN 命令来查看查询计划中的步骤。以下示例显示了 SQL 查询并解释了输出。自下而上阅读该查询计划，您可以了解执行该查询所用的每个逻辑操作。有关更多信息，请参阅查询计划。

```
explain
select eventname, sum(pricepaid) from sales, event
where sales.eventid = event.eventid
group by eventname
order by 2 desc;
```

```
XN Merge  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
  Merge Key: sum(sales.pricepaid)
  ->  XN Network  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
        Send to leader
        ->  XN Sort  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
              Sort Key: sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=2815366577.07..2815366578.51 rows=576
 width=27)
                    ->  XN Hash Join DS_BCAST_INNER  (cost=109.98..2815365714.80
 rows=172456 width=27)
                          Hash Cond: ("outer".eventid = "inner".eventid)
                          ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456
 width=14)
                          ->  XN Hash  (cost=87.98..87.98 rows=8798 width=21)
                                ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798
 width=21)
```

作为生成查询计划的一部分，查询优化程序将计划分解为流、分段和步骤。查询优化程序将分解计划，以准备将数据和查询工作负载分配到计算节点。有关流、分段和步骤的更多信息，请参阅查询计划和执行工作流程。

下图显示了前面的查询和关联的查询计划。它显示涉及的查询操作如何映射到 Amazon Redshift 用于为计算节点切片生成编译代码的步骤。每个查询计划操作映射到段中的多个步骤，有时映射到流中的多个段。



在此图中，查询优化程序运行查询计划，如下所示：

1. 在 Stream 0 中，查询通过顺序扫描操作运行 Segment 0，以扫描 events 表。查询继续通过哈希操作运行 Segment 1，以为联接中的内部表创建哈希表。
2. 在 Stream 1 中，查询通过顺序扫描操作运行 Segment 2，以扫描 sales 表。它继续使用哈希联接运行 Segment 2 以联接表，其中联接列不是分配键和排序键。它再次继续通过哈希聚合来运行 Segment 2，以聚合结果。然后，查询使用哈希聚合操作运行 Segment 3 来执行未排序的分组聚合函数和排序操作，以评估 ORDER BY 子句和其他排序操作。
3. 在 Stream 2 中，查询在 Segment 4 和 Segment 5 中运行网络操作，以将中间结果发送到领导节点以待进一步处理。

查询的最后一个分段返回数据。如果聚合或对返回集进行排序，则计算节点将各自的中间结果段发送到领导节点。然后，领导节点合并数据，以便将最终结果发送回请求客户端。

有关 EXPLAIN 运算符的更多信息，请参阅EXPLAIN。

影响查询性能的因素

有很多因素会影响查询性能。数据、集群和数据库操作的以下方面都在查询处理速度方面发挥作用。

- 节点、处理器或切片的数量 – 一个计算节点分为多个切片。节点越多意味着处理器和切片越多，通过跨各个切片并发运行查询的多个部分，可加快查询的处理速度。但是，节点越多也意味着花费越高，因此，您需要为自己的系统找到成本和性能之间的适当平衡点。有关 Amazon Redshift 集群架构的更多信息，请参阅数据仓库系统架构。
- 节点类型 – Amazon Redshift 集群有多种节点类型可以选择。每种节点类型都提供不同的大小和限制，以帮助您适当地扩展集群。节点大小决定集群中每个节点的存储容量、内存、CPU 和价格。有关节点类型的更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 集群概述。
- 数据分配 – Amazon Redshift 根据表的分配方式在计算节点上存储表数据。在执行查询时，查询优化程序根据执行联接和聚合的需要将数据重新分配到计算节点。为表选择正确的分配方式有助于通过在执行联接前将数据放在需要的位置来最大程度地减小重新分配步骤的影响。有关更多信息，请参阅使用数据分配样式。
- 数据排序顺序 – Amazon Redshift 根据表的排序键将表数据按照排序顺序存储在磁盘中。查询优化程序和查询处理器使用有关数据所在位置的信息来减少需要扫描的数据块数，从而提高查询速度。有关更多信息，请参阅使用排序键。
- 数据集大小 – 集群中的数据量越大，则需要扫描和重新分配的行数也越多，这会降低查询性能。您可以通过定期对数据进行 vacuum 操作和归档以及使用谓词来限制查询数据集来减轻这种影响。
- 并发操作 – 同时运行多个操作会影响查询性能。每个操作在可用查询队列中都会占用一个或多个插槽，并使用与这些插槽关联的内存。如果其他操作正在运行，则可能没有充足的查询队列槽可用。在这种情况下，查询必须等待有槽回收后才能开始处理。有关创建和配置查询队列的更多信息，请参阅实施工作负载管理。
- 查询结构 – 查询的编写也会影响其性能。在满足需求的前提下，请尽量将查询编写为处理和返回尽量少的数据。有关更多信息，请参阅设计查询的 Amazon Redshift 最佳实践。
- 代码编译 – Amazon Redshift 为每个查询执行计划生成和编译代码。

  编译代码执行更快，因为它消除了使用解释器的开销。但首次生成和编译代码通常会产生一些开销。因此，首次运行查询时的性能可能会造成误导。运行一次性查询时，这一开销可能会特别明显。再运行一次查询来确定其典型性能。Amazon Redshift 使用无服务器编译服务将查询编译扩展到 Amazon

Redshift 集群的计算资源之外。编译的代码段本地缓存于集群中，且是在几乎无限的缓存中。集群重新启动后，此缓存将保持不变。相同查询的后续执行可以更快地运行，因为它们可以跳过编译阶段。

缓存在 Amazon Redshift 版本之间不兼容，因此，在版本升级后运行查询时会刷新编译缓存并重新编译代码。如果您的查询具有严格的 SLA，我们建议您预先运行查询分段，以扫描集群表中的数据。这样，Amazon Redshift 就可以缓存基表数据，从而缩短版本升级后查询的规划时间。通过使用可扩展的编译服务，Amazon Redshift 能够并行编译代码，以提供始终如一的快速性能。工作负载加速的幅度取决于查询的复杂性和并发性。

分析和改进查询

从 Amazon Redshift 数据仓库检索信息需要对海量数据运行复杂的查询，这可能需要很长的时间进行处理。为确保尽快处理查询，您可以使用多种工具来确定潜在的性能问题。

主题

查询分析工作流程

如果查询所花费的时间比预期的时间长，请使用以下步骤来确定并纠正可能对查询性能产生负面影响的问题。如果您不确定系统中的哪些查询可能会受益于性能调整，请先运行 确定最适合优化的查询 中的诊断查询。

1. 确保您的表格按照最佳实践进行设计。有关更多信息，请参阅设计表的 Amazon Redshift 最佳实践。
2. 查看是否可以删除或归档表中的任何不需要的数据。例如，假设您的查询始终以过去 6 个月的数据为目标，但您的表中有最近 18 个月的数据。在这种情况下，您可以删除或归档较旧的数据，以减少必须扫描和分配的记录数。
3. 对查询中的表运行 VACUUM 命令以回收空间并对行进行重新排序。如果未排序的区域很大，并且查询在联接或谓词中使用排序键，则运行 VACUUM 会有所帮助。

4. 对查询中的表运行 ANALYZE 命令，以确保统计信息是最新的。如果查询中的任何表最近在大小上发生了很大变化，则运行 ANALYZE 会有所帮助。如果运行完整的 ANALYZE 命令需要过长时间，则对某一列运行 ANALYZE，以减少处理时间。这种方法仍会更新表大小的统计数据；表大小在查询计划中是一个很重要的因素。
5. 确保查询在每个类型的客户端（基于客户端使用的连接协议的类型）上都运行一次，以便查询得到编译和缓存。该种方法可加快查询的后续运行速度。有关更多信息，请参阅影响查询性能的因素。
6. 检查 STL_ALERT_EVENT_LOG 表来识别和纠正查询中可能存在的问题。有关更多信息，请参阅查看查询警报。
7. 运行 EXPLAIN 命令以获取查询计划并使用它优化查询。有关更多信息，请参阅分析查询计划。
8. 使用 SVL_QUERY_SUMMARY 和 SVL_QUERY_REPORT 视图来获取摘要信息并使用它来优化查询。有关更多信息，请参阅分析查询摘要。

有时，应该快速运行的查询要被迫等到另一个运行时间较长的查询完成。在这种情况下，查询本身可能无需改进，但您可以通过为不同类型的查询创建和使用查询队列来提高整体系统性能。要了解查询的队列等待时间，请参阅查看查询的队列等待时间。有关配置查询队列的更多信息，请参阅实施工作负载管理。

查看查询警报

要使用 STL_ALERT_EVENT_LOG 系统表来识别和纠正查询的潜在性能问题，请按照下列步骤操作：

1. 运行以下命令以确定查询 ID：

```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

   检查 substring 字段中的截断查询文本来确定选择哪些 query 值。如果您已多次运行查询，请使用具有较低 elapsed 值的行中的 query 值。这是已编译的行。如果运行多个查询，则可以增大 LIMIT 子句使用的值，以确保将查询包含在内。

2. 从查询的 STL_ALERT_EVENT_LOG 中选择行：

```
Select * from stl_alert_event_log where query = MyQueryID;
```

userid	query	slice	segment	step	pid	xid	event	solution	event_time
100	32359	4	0	0	8780	71195	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-10 17:40:50
100	32359	5	0	0	8781	71195	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-10 17:40:50
100	109142	4	0	0	8780	302411	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-24 20:32:28
100	109142	5	0	0	8781	302411	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-24 20:32:28
100	109828	4	1	0	8746	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:27:52
100	109828	5	1	0	8747	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:27:52
100	109829	4	1	0	8760	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:28:01
100	109829	5	1	0	8761	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:28:01
100	113910	4	1	0	8774	316848	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-25 17:14:58

3. 评估查询的结果。使用下面的表找到已发现的任何问题的可能解决方案。

> **Note**
>
> 并非所有查询都在 STL_ALERT_EVENT_LOG 中拥有行（仅限存在已发现的问题的查询）。

问题	事件值	解决方案值	建议的解决方案
查询中表的统计信息缺失或过期。	缺少查询计划程序统计信息	运行 ANALYZE 命令	请参阅 表统计数据缺失或过时。
查询计划中存在嵌套循环联接（最不优化的联接）。	查询计划中的嵌套循环联接	查看联接谓词以避免笛卡尔积	请参阅 嵌套循环。
扫描跳过了相对大量的标记为已删除但未抽空的行或已插入但未提交的行。	已扫描大量已删除的行	运行 VACUUM 命令以回收已删除的空间	请参阅 虚影行或未提交的行。
为进行哈希联接或聚合重新分配了超过 1000000 的行。	在整个网络中分布了大量行：分配了 RowCount 行以处理聚合	查看选择的分配键以并置联接或聚合	请参阅 非最优数据分配。
为进行哈希联接广播了超过了 1000000 个行。	已在网络中广播大量行	查看分配键的选择以并置联接，并考虑使用分配表	请参阅 非最优数据分配。
DS_DIST_ALL_INNER 重新分配方式已在查询计划中指明，此方式强制	查询计划中哈希联接的	查看分配策略的选择，以分配内	请参阅 非最优数据分配。

问题	事件值	解决方案值	建议的解决方案
实施序列执行，因为整个内部表已重新分配到单个节点。	DS_DIST_ALL_INNER	部表而不是外部表	

分析查询计划

在分析查询计划前，应该熟悉如何阅读该计划。如果您不熟悉如何阅读查询计划，建议阅读 查询计划 后再继续。

运行 EXPLAIN 命令以获取查询计划。若要分析查询计划提供的数据，请按照下列步骤操作：

1. 确定成本最高的步骤。继续完成其余步骤时，专注于优化这些步骤。
2. 查看联接类型：
   - 嵌套循环：此类联接通常因此联接条件被忽略而发生。有关建议的解决方案，请参阅嵌套循环。
   - 哈希和哈希联接：当联接列不为分配键，也不为排序键时，使用哈希联接来联接表。有关建议的解决方案，请参阅哈希联接。
   - 合并联接：无需更改。
3. 注意哪个表用于内部连接，哪个表用于外部连接。查询引擎通常会为内部连接选择较小的表，为外部连接选择较大的表。如果没有进行此类选择，则您的统计数据可能已过时。有关建议的解决方案，请参阅表统计数据缺失或过时。
4. 查看是否有任何高成本的排序操作。如果有，请参阅 未排序或排序错乱的行 了解建议的解决方案。
5. 查找具有高成本操作的以下广播运算符：
   - DS_BCAST_INNER：指示将表广播到所有计算节点。这对于小表而言很好，但对于大表来说并不理想。
   - DS_DIST_ALL_INNER：表示所有工作负载都位于单个切片上。
   - DS_DIST_BOTH：表示大量重新分配。

   有关这些情况的建议解决方案，请参阅非最优数据分配。

分析查询摘要

要获取比 EXPLAIN 生成的查询计划更详细的执行步骤和统计数据，请使用 SVL_QUERY_SUMMARY 和 SVL_QUERY_REPORT 系统视图。

SVL_QUERY_SUMMARY 按流提供查询统计数据。您可以利用它提供的信息发现与高昂成本步骤、耗时步骤及写入磁盘步骤有关的问题。

SVL_QUERY_REPORT 系统视图为您提供与 SVL_QUERY_SUMMARY 相似的信息，但其按计算节点切片而非按流提供信息。您可以利用切片级信息检测跨集群的不均匀数据分配（也称作数据分配偏斜），后者会导致某些节点执行更多的工作，从而影响查询的性能。

主题

使用 SVL_QUERY_SUMMARY 视图

要按流分析查询摘要信息，请执行以下操作：

1. 运行以下查询以确定查询 ID：

```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

   检查 substring 字段中的截断查询文本来确定哪些 query 值代表您的查询。如果您已多次运行查询，请使用具有较低 elapsed 值的行中的 query 值。这是已编译的行。如果运行多个查询，则可以增大 LIMIT 子句使用的值，以确保将查询包含在内。

2. 从查询的 SVL_QUERY_SUMMARY 中选择。按流、分段和步骤对结果进行排序：

```
select * from svl_query_summary where query = MyQueryID order by stm, seg, step;
```

Result 1 | Messages

userid	query	stm	seg	step	maxtime	avgtime	rows	bytes	rate_row	rate_byte	label	is_diskbased	workmem	is_rrscan	is_delayed_scan	rows_pre_filter
1	249059	0	0	0	58	27	4	192			scan tbl=246 name=Internal Worktable	f	0	f	f	0
1	249059	0	0	1	58	27	4	0			project	f	0	f	f	0
1	249059	0	0	2	58	27	4	64			save tbl=249	f	481296384	f	f	0
1	249059	1	1	0	20	20	1	48			scan tbl=250 name=Internal Worktable	f	0	f	f	0
1	249059	1	1	1	20	20	1	0			dist	f	0	f	f	0
1	249059	1	2	0	2275	1350	1	48			scan tbl=19221 name=Internal Worktable	f	0	f	f	0
1	249059	1	2	1	2275	1350	1	0			project	f	0	f	f	0
1	249059	1	2	2	2275	1350	1	16			save tbl=249	f	475004928	f	f	0
1	249059	2	3	0	1640	792	5	80			scan tbl=249 name=Internal Worktable	f	0	f	f	0
1	249059	2	3	1	1640	792	5	80			sort tbl=248	f	468713472	f	f	0
1	249059	3	4	0	26	9	5	80			scan tbl=248 name=Internal Worktable	f	0	f	f	0
1	249059	3	4	1	26	9	5	0			return	f	0	f	f	0
1	249059	3	5	0	49	49	0	0			merge	f	0	f	f	0
1	249059	3	5	1	49	49	5	0			project	f	0	f	f	0
1	249059	3	5	2	49	49	0	0			return	f	0	f	f	0

3. 使用 将查询计划映射到查询摘要 中的信息将步骤映射到查询计划中的操作。它们应具有大致相同的行和字节值（查询计划中的行 * 宽度）。如果没有，请参阅表统计数据缺失或过时了解建议的解决方案。
4. 查看在任何步骤中 is_diskbased 字段的值是否都为 t（真）。哈希、聚合和排序是指在系统没有足够的内存用于查询处理的情况下可能会将数据写入磁盘的运算符。

   如果 is_diskbased 为真，请参阅 分配给查询的内存不足 了解建议的解决方案。
5. 查看 label 字段值，并查看步骤中是否存在 AGG-DIST-AGG 序列。它的存在表明聚合分两步，其成本高昂。要解决此问题，请将 GROUP BY 子句更改为使用分配键（如果有多个键，则为第一个键）。
6. 查看每个分段的 maxtime 值（该值在分段中的所有步骤中相同）。标识具有最高 maxtime 值的分段，并查看此分段中的以下运算符的步骤。

   > **Note**
   >
   > 较高的 maxtime 值并不一定表示分段出现问题。尽管值很高，但该分段可能不需要很长时间来处理。流中的所有分段都开始统一计时。但是，从上游段获得数据前，某些下游段可能无法运行。这可能导致它们看起来需要长时间执行，因为它们的 maxtime 值包含等待时间和处理时间。

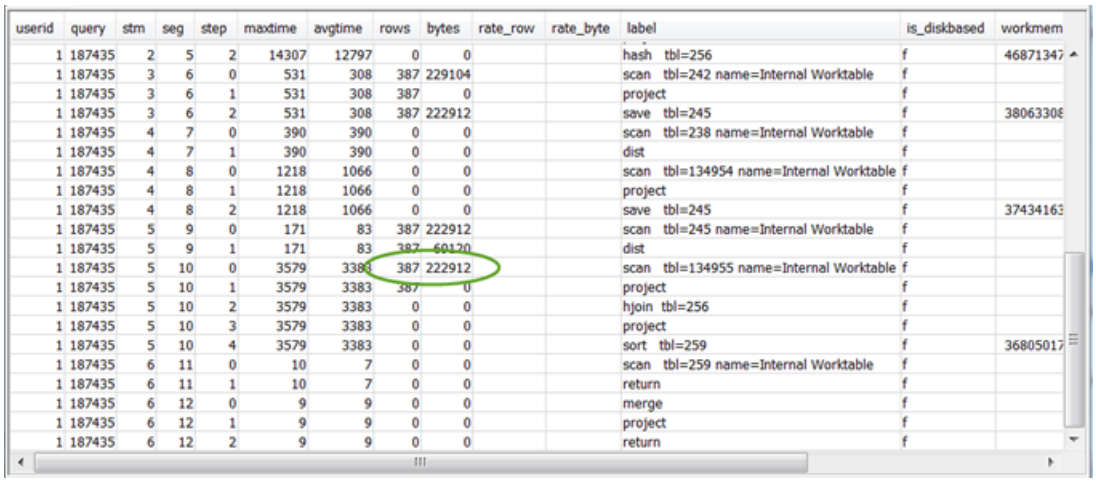
   - BCAST 或 DIST：在这些情况下，maxtime 值较大可能是由重新分配大量的行造成的。有关建议的解决方案，请参阅非最优数据分配。
   - HJOIN（哈希联接）：如果所涉及的步骤的 rows 字段值比查询的最终 RETURN 步骤的 rows 值高很多，请参阅 哈希联接 了解建议的解决方案。
   - SCAN/SORT：查找联接步骤之前的 SCAN、SORT、SCAN、MERGE 步骤序列。此模式表示正在扫描、排序未排序的数据，然后将其与表的排序区域合并。

     查看 SCAN 步骤的行值是否比查询的最终 RETURN 步骤中的行值高很多。此模式表示执行引擎正在扫描稍后将丢弃的行，这样做效率低下。有关建议的解决方案，请参阅谓词限制性不足。

     如果 SCAN 步骤的 maxtime 值较高，请参阅 非最优 WHERE 子句 了解建议的解决方案。

     如果 SORT 步骤的 rows 值较高，请参阅未排序或排序错乱的行了解建议的解决方案。
7. 查看最终 RETURN 步骤前 5-10 步的 rows 和 bytes 值，以了解返回到客户端的数据量。这个过程可以说是一门艺术。

例如，在下面的查询摘要中，您可以看到第三个 PROJECT 步骤提供了 rows 值，但未提供 bytes 值。通过查看前面的步骤，查找具有相同 rows 值的步骤，您可以找到同时提供行和字节信息的 SCAN 步骤：



userid	query	stm	seg	step	maxtime	avgtime	rows	bytes	rate_row	rate_byte	label	is_diskbased	workmem
1	187435	2	5	2	14307	12797	0	0			hash tbl=256	f	46871347
1	187435	3	6	0	531	308	387	229104			scan tbl=242 name=Internal Worktable	f	
1	187435	3	6	1	531	308	387	0			project	f	
1	187435	3	6	2	531	308	387	222912			save tbl=245	f	38063308
1	187435	4	7	0	390	390	0	0			scan tbl=238 name=Internal Worktable	f	
1	187435	4	7	1	390	390	0	0			dist	f	
1	187435	4	8	0	1218	1066	0	0			scan tbl=134954 name=Internal Worktable	f	
1	187435	4	8	1	1218	1066	0	0			project	f	
1	187435	4	8	2	1218	1066	0	0			save tbl=245	f	37434163
1	187435	5	9	0	171	83	387	222912			scan tbl=245 name=Internal Worktable	f	
1	187435	5	9	1	171	83	387	60120			dist	f	
1	187435	5	10	0	3579	3383	387	222912			scan tbl=134955 name=Internal Worktable	f	
1	187435	5	10	1	3579	3383	387	0			project	f	
1	187435	5	10	2	3579	3383	0	0			hjoin tbl=256	f	
1	187435	5	10	3	3579	3383	0	0			project	f	
1	187435	5	10	4	3579	3383	0	0			sort tbl=259	f	36805017
1	187435	6	11	0	10	7	0	0			scan tbl=259 name=Internal Worktable	f	
1	187435	6	11	1	10	7	0	0			return	f	
1	187435	6	12	0	9	9	0	0			merge	f	
1	187435	6	12	1	9	9	0	0			project	f	
1	187435	6	12	2	9	9	0	0			return	f	

如果要返回量异常大的数据，请参阅 极大结果集 了解建议的解决方案。

8. 查看任何步骤的 bytes 与其他步骤相比是否相对 rows 值较高。此模式说明您选择了大量列。有关建议的解决方案，请参阅大型 SELECT 列表。

使用 SVL_QUERY_REPORT 视图

要按切片分析查询摘要信息，请执行以下操作：

1. 运行以下命令以确定查询 ID：

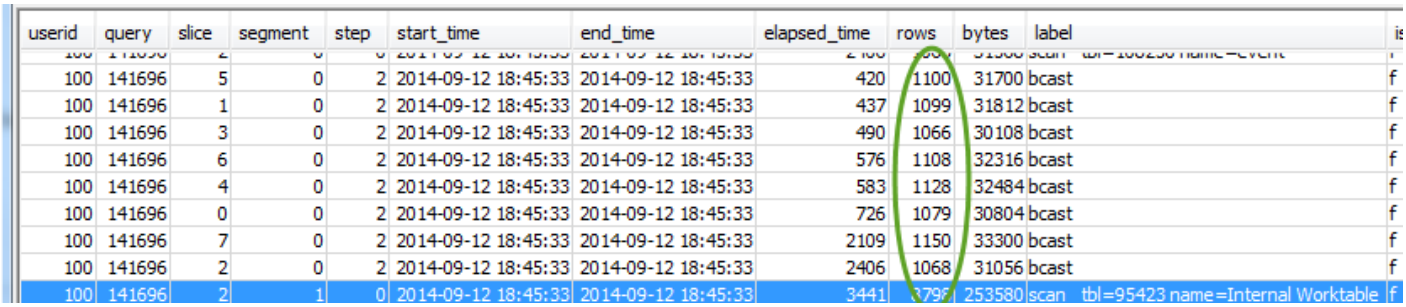
```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

检查 substring 字段中的截断查询文本来确定哪些 query 值代表您的查询。如果您已多次运行查询，请使用具有较低 elapsed 值的行中的 query 值。这是已编译的行。如果运行多个查询，则可以增大 LIMIT 子句使用的值，以确保将查询包含在内。

2. 从查询的 SVL_QUERY_REPORT 中选择。按分段、步骤、elapsed_time 和行对结果进行排序：

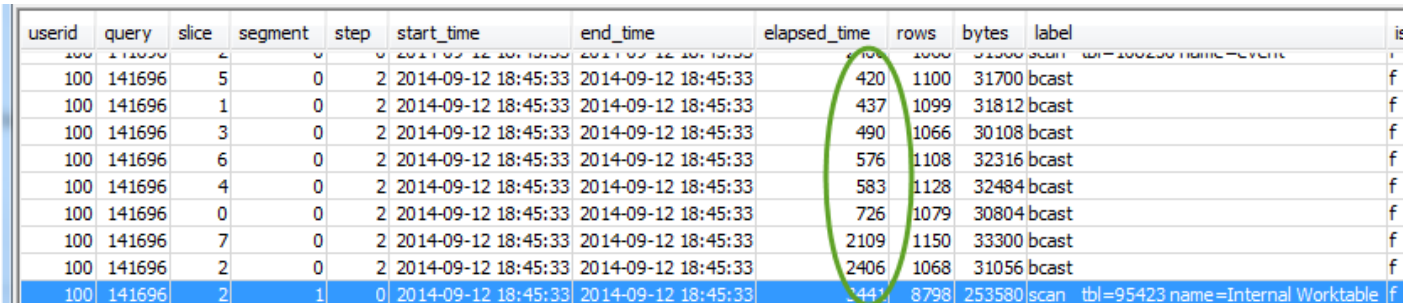
```
select * from svl_query_report where query = MyQueryID order by segment, step,
 elapsed_time, rows;
```

3. 对于每个步骤，检查以查看处理的所有切片数是否与行数大致相同：



userid	query	slice	segment	step	start_time	end_time	elapsed_time	rows	bytes	label
100	141696	5	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	420	1100	31700	bcast
100	141696	1	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	437	1099	31812	bcast
100	141696	3	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	490	1066	30108	bcast
100	141696	6	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	576	1108	32316	bcast
100	141696	4	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	583	1128	32484	bcast
100	141696	0	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	726	1079	30804	bcast
100	141696	7	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2109	1150	33300	bcast
100	141696	2	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2406	1068	31056	bcast
100	141696	2	1	0	2014-09-12 18:45:33	2014-09-12 18:45:33	3441	3798	253580	scan tbl=95423 name=Internal Worktable

此外，检查以查看所有切片所花费的时间是否大致相同：



userid	query	slice	segment	step	start_time	end_time	elapsed_time	rows	bytes	label
100	141696	5	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	420	1100	31700	bcast
100	141696	1	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	437	1099	31812	bcast
100	141696	3	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	490	1066	30108	bcast
100	141696	6	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	576	1108	32316	bcast
100	141696	4	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	583	1128	32484	bcast
100	141696	0	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	726	1079	30804	bcast
100	141696	7	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2109	1150	33300	bcast
100	141696	2	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2406	1068	31056	bcast
100	141696	2	1	0	2014-09-12 18:45:33	2014-09-12 18:45:33	3441	8798	253580	scan tbl=95423 name=Internal Worktable

这些值存在较大的差异可能表明由于此特定查询的分配方式不佳而导致数据分布偏差。有关建议的解决方案，请参阅非最优数据分配。

将查询计划映射到查询摘要

它有助于将查询计划中的操作映射到查询摘要中的步骤（由标签字段值标识），以获取更多详细信息：

查询计划操作	标签字段值	描述
聚合 HashAggregate GroupAggregate	AGGR	评估聚合函数和 GROUP BY 条件。
DS_BCAST_INNER	BCAST（广播）	向所有节点广播整个表或某些行集（例如表中的筛选行集）。
未出现在查询计划中	删除	从表中删除行。

查询计划操作	标签字段值	描述
DS_DIST_NONE DS_DIST_ALL_NONE DS_DIST_INNER DS_DIST_ALL_INNER DS_DIST_ALL_BOTH	DIST（分配）	将行分配到节点，用于并行联接或其他并行处理。
HASH	HASH	构建用于哈希联接的联接表。
哈希联接	HJOIN（哈希联接）	执行两个表或中间结果集的哈希联接。
未出现在查询计划中	INSERT	将行插入到表中。
限制	LIMIT	将 LIMIT 子句应用于结果集。
合并	MERGE	合并来自并行排序或联接操作的行。
合并联接	MJOIN（合并联接）	执行两个表或中间结果集的合并联接。
嵌套循环	NLOOP（嵌套循环）	执行两个表或中间结果集的嵌套循环联接。
未出现在查询计划中	PARSE	将字符串解析为二进制值以进行加载。
项目	PROJECT	评估表达式。
Network	RETURN	将行返回到领导节点或客户端。
未出现在查询计划中	SAVE	具体化行，以便在下一处理步骤中使用。

查询计划操作	标签字段值	描述
Seq Scan	SCAN	扫描表或中间结果集。
排序	SORT	根据其他后续操作（如联接或聚合）的要求或为了满足 ORDER BY 子句的要求而对行或中间结果集进行排序。
唯一	UNIQUE	根据其他操作的要求应用 SELECT DISTINCT 子句或删除重复项。
窗口	WINDOW	计算聚合并对窗口函数进行排名。

提高查询性能

以下是影响查询性能的一些常见问题，以及有关诊断和解决这些问题的方法的说明。

主题

- 表统计数据缺失或过时
- 嵌套循环
- 哈希联接
- 虚影行或未提交的行
- 未排序或排序错乱的行
- 非最优数据分配
- 分配给查询的内存不足
- 非最优 WHERE 子句
- 谓词限制性不足
- 极大结果集
- 大型 SELECT 列表

表统计数据缺失或过时

如果表统计信息缺失或过期，您可能会看到以下内容：

- EXPLAIN 命令结果中存在警告消息。
- STL_ALERT_EVENT_LOG 中存在缺失统计数据提醒事件。有关更多信息，请参阅查看查询警报。

要修复此问题，请运行 ANALYZE。

嵌套循环

如果存在嵌套循环，您可能会在 STL_ALERT_EVENT_LOG 中看到嵌套循环提示事件。此外，您还可以通过运行确定具有嵌套循环的查询中的查询来识别此类事件。有关更多信息，请参阅查看查询警报。

要修复这一问题，请检查查询是否存在交叉联接，尽量将其删除。交叉联接是无联接条件的联接，它会导致对两个表执行笛卡尔积操作。它们通常作为嵌套循环联接运行，这是最慢的可能联接类型。

哈希联接

如果存在哈希联接，您可能会看到以下内容：

- 查询计划中的哈希和哈希联接操作。有关更多信息，请参阅分析查询计划。
- SVL_QUERY_SUMMARY 中 maxtime 值最大的段中的 HJOIN 步骤。有关更多信息，请参阅使用 SVL_QUERY_SUMMARY 视图。

要修复这一问题，您可以采取以下几种方法：

- 重写查询，尽可能使用合并联接。为此，您可以指定既是分配键又是排序键的联接列。
- 如果 SVL_QUERY_SUMMARY 中 HJOIN 步骤的行数字段值较查询中最终 RETURN 步骤中的行数值大得多，请确认您能否重写查询以基于唯一的列进行联接。当查询未联接唯一列（如主键）时，该列会增加联接中涉及的行数。

虚影行或未提交的行

如果存在虚影行或未提交的行，您可能会在 STL_ALERT_EVENT_LOG 中看到一个提示事件，指示虚影行过多。有关更多信息，请参阅查看查询警报。

要修复这一问题，您可以采取以下几种方法：

- 检查 Amazon Redshift 控制台的加载选项卡，以便对任何查询表进行活动加载操作。如果您发现有处于活动状态的加载操作，请等待这些操作完成，然后再执行操作。
- 如果没有活动加载操作，请对查询表运行 VACUUM 以移除已删除的行。

未排序或排序错乱的行

如果存在未排序或排序错误的行，您可能会在 STL_ALERT_EVENT_LOG 中看到一个非常有选择性的筛选提示事件。有关更多信息，请参阅查看查询警报。

您也可以运行确定具有数据偏斜或未排序行的表中的查询以检查查询中的任意表是否包含大片未排序的区域。

要修复这一问题，您可以采取以下几种方法：

- 对查询表运行 VACUUM 以重新排序行。
- 检查查询表的排序键，看看有无可以改进之处。在进行任何更改之前，请务必权衡查询的性能及其他重要查询和系统的整体性能。有关更多信息，请参阅使用排序键。

非最优数据分配

如果数据分配不佳，您可能会看到以下内容：

- STL_ALERT_EVENT_LOG 中存在顺序执行、大型广播或大型分配提醒事件。有关更多信息，请参阅查看查询警报。
- 对于给定步骤，处理的切片数与行数相差较大。有关更多信息，请参阅使用 SVL_QUERY_REPORT 视图。
- 对于给定步骤，切片的处理时间相差较大。有关更多信息，请参阅使用 SVL_QUERY_REPORT 视图。

如果上述条件均未满足，您还可以查看查询中是否有任意表存在数据偏斜（运行确定具有数据偏斜或未排序行的表中的查询）。

要修复这一问题，请查看查询中表的分布方式，看看有无任何可以改进之处。在进行任何更改之前，请务必权衡查询的性能及其他重要查询和系统的整体性能。有关更多信息，请参阅使用数据分配样式。

分配给查询的内存不足

如果为查询分配到的内存不足，您可能会看到 SVL_QUERY_SUMMARY 中存在一个 `is_diskbased` 值为真的步骤。有关更多信息，请参阅使用 SVL_QUERY_SUMMARY 视图。

要修复这一问题，请临时增加查询使用的查询槽的数目，以向其分配更多的内存。工作负载管理 (WLM) 在查询队列中预留与为查询设置的并发级别相等的槽数。例如，并发级别为 5 的队列拥有 5 个槽。分配给队列的内存平均分配到每个槽。将多个槽分配给一个查询可使该查询访问所有这些槽的内存。有关如何临时增加查询的插槽的更多信息，请参阅wlm_query_slot_count。

非最优 WHERE 子句

如果 WHERE 子句导致表扫描过多，您可能会在分段中看到在 SVL_QUERY_SUMMARY 中具有最高 `maxtime` 值的 SCAN 步骤。有关更多信息，请参阅使用 SVL_QUERY_SUMMARY 视图。

要修复这一问题，请根据最大的表的主排序列向查询添加 WHERE 子句。这种方法有助于尽量减少扫描时间。有关更多信息，请参阅设计表的 Amazon Redshift 最佳实践。

谓词限制性不足

如果您的查询具有限制性不足的谓词，您可能会在分段中看到在 SVL_QUERY_SUMMARY 中具有最高 `maxtime` SCAN 步骤，且它的 `rows` 值比查询的最终 RETURN 步骤中的 `rows` 值高很多。有关更多信息，请参阅使用 SVL_QUERY_SUMMARY 视图。

若要解决此问题，请尝试向查询添加谓词或使现有谓词更具限制性，以缩小输出范围。

极大结果集

如果查询返回极大的结果集，请考虑重写查询，使用 UNLOAD 将结果写入 Amazon S3。这种方法可充分利用并行处理的优势，从而提高 RETURN 步骤的性能。有关检查极大结果集的更多信息，请参阅使用 SVL_QUERY_SUMMARY 视图。

大型 SELECT 列表

如果您的查询有一个异常大的 SELECT 列表，您可能会看到 SVL_QUERY_SUMMARY 中的任何步骤的 `bytes` 值（与其他步骤相比）相对于 `rows` 值较高。`bytes` 值较大说明您选择了大量列。有关更多信息，请参阅使用 SVL_QUERY_SUMMARY 视图。

要解决此问题，请查看您正在选择的列，并查看是否可以删除任何列。

用于优化查询的诊断查询

使用以下查询发现与可能影响查询性能的查询或基础表有关的问题。我们建议将这些查询与分析和改进查询中讨论的查询优化过程结合使用。

主题

确定最适合优化的查询

以下查询标识了过去 7 天内运行的前 50 个最耗时的语句。您可以使用结果来识别需要非常长时间的查询。您也可以确定经常运行的查询（在结果集中多次出现的查询）。通常可以优化此类查询以提高系统性能。

此外，此查询还提供与每个所发现的查询关联的提醒事件计数。这些提醒提供详细信息，供您用于提高查询的性能。有关更多信息，请参阅查看查询警报。

```
select trim(database) as db, count(query) as n_qry,
max(substring (qrytext,1,80)) as qrytext,
min(run_minutes) as "min" ,
max(run_minutes) as "max",
avg(run_minutes) as "avg", sum(run_minutes) as total,
max(query) as max_query_id,
max(starttime)::date as last_run,
sum(alerts) as alerts, aborted
from (select userid, label, stl_query.query,
trim(database) as database,
trim(querytxt) as qrytext,
md5(trim(querytxt)) as qry_md5,
starttime, endtime,
(datediff(seconds, starttime,endtime)::numeric(12,2))/60 as run_minutes,
alrt.num_events as alerts, aborted
from stl_query
left outer join
```

```
(select query, 1 as num_events from stl_alert_event_log group by query ) as alrt
on alrt.query = stl_query.query
where userid <> 1 and starttime >=  dateadd(day, -7, current_date))
group by database, label, qry_md5, aborted
order by total desc limit 50;
```

确定具有数据偏斜或未排序行的表

以下查询标识数据分配不均匀（数据偏斜）或未排序行百分比较高的表。

skew 值较低表明表数据分配适当。如果表的 skew 值达到 4.00 或以上，可以考虑修改其数据分配方式。有关更多信息，请参阅非最优数据分配。

如果表的 pct_unsorted 值大于 20%，可以考虑运行 VACUUM 命令。有关更多信息，请参阅未排序或排序错乱的行。

还应检查每个表的 mbytes 和 pct_of_total 值。这些列标识表的大小及表占用的原始磁盘空间比例。原始磁盘空间包括 Amazon Redshift 保留供内部使用的空间，因此它大于名义磁盘容量（可供用户使用的磁盘空间量）。使用这些信息可验证可用磁盘空间等于最大表大小的 2.5 倍或以上。如果使用此空间，系统可以在处理复杂查询时将临时结果写入磁盘。

```
select trim(pgn.nspname) as schema,
trim(a.name) as table, id as tableid,
decode(pgc.reldiststyle,0, 'even',1,det.distkey ,8,'all') as distkey,
 dist_ratio.ratio::decimal(10,4) as skew,
det.head_sort as "sortkey",
det.n_sortkeys as "#sks", b.mbytes,
decode(b.mbytes,0,0,((b.mbytes/part.total::decimal)*100)::decimal(5,2)) as
 pct_of_total,
decode(det.max_enc,0,'n','y') as enc, a.rows,
decode( det.n_sortkeys, 0, null, a.unsorted_rows ) as unsorted_rows ,
decode( det.n_sortkeys, 0, null, decode( a.rows,0,0, (a.unsorted_rows::decimal(32)/
a.rows)*100) )::decimal(5,2) as pct_unsorted
from (select db_id, id, name, sum(rows) as rows,
sum(rows)-sum(sorted_rows) as unsorted_rows
from stv_tbl_perm a
group by db_id, id, name) as a
join pg_class as pgc on pgc.oid = a.id
join pg_namespace as pgn on pgn.oid = pgc.relnamespace
left outer join (select tbl, count(*) as mbytes
from stv_blocklist group by tbl) b on a.id=b.tbl
inner join (select attrelid,
min(case attisdistkey when 't' then attname else null end) as "distkey",
```

```
min(case attsortkeyord when 1 then attname  else null end ) as head_sort ,
max(attsortkeyord) as n_sortkeys,
max(attencodingtype) as max_enc
from pg_attribute group by 1) as det
on det.attrelid = a.id
inner join ( select tbl, max(mbytes)::decimal(32)/min(mbytes) as ratio
from (select tbl, trim(name) as name, slice, count(*) as mbytes
from svv_diskusage group by tbl, name, slice )
group by tbl, name ) as dist_ratio on a.id = dist_ratio.tbl
join ( select sum(capacity) as  total
from stv_partitions where part_begin=0 ) as part on 1=1
where mbytes is not null
order by  mbytes desc;
```

确定具有嵌套循环的查询

以下查询标识已为嵌套循环记录提示事件的查询。有关如何修复嵌套循环条件的信息，请参阅嵌套循环。

```
select query, trim(querytxt) as SQL, starttime
from stl_query
where query in (
select distinct query
from stl_alert_event_log
where event like 'Nested Loop Join in the query plan%')
order by starttime desc;
```

查看查询的队列等待时间

以下查询显示了最近的查询在运行之前等待查询队列中可用槽的时间。如果等待时间较长，可能需要修改查询队列配置，以获得更高的吞吐量。有关更多信息，请参阅实施手动 WLM。

```
select trim(database) as DB , w.query,
substring(q.querytxt, 1, 100) as querytxt,  w.queue_start_time,
w.service_class as class, w.slot_count as slots,
w.total_queue_time/1000000 as queue_seconds,
w.total_exec_time/1000000 exec_seconds, (w.total_queue_time+w.total_Exec_time)/1000000
 as total_seconds
from stl_wlm_query w
left join stl_query q on q.query = w.query and q.userid = w.userid
where w.queue_start_Time >= dateadd(day, -7, current_Date)
and w.total_queue_Time > 0  and w.userid >1
```

```
and q.starttime >= dateadd(day, -7, current_Date)
order by w.total_queue_time desc, w.queue_start_time desc limit 35;
```

按表查看查询警报

以下查询标识记录了提示事件的表，并标识了最频繁引发的提示类型。

如果所确定的表的行的 minutes 值较大，请检查该表以查看是否需要对其执行日常维护（如对其运行 ANALYZE 或 VACUUM）。

如果某一行的 count 值高但 table 值为 null，则对 STL_ALERT_EVENT_LOG 运行查询以获得相关的 event 值，从而调查如此频繁引发提示的原因。

```
select trim(s.perm_table_name) as table,
(sum(abs(datediff(seconds, s.starttime, s.endtime)))/60)::numeric(24,0) as minutes,
 trim(split_part(l.event,':',1)) as event,  trim(l.solution) as solution,
max(l.query) as sample_query, count(*)
from stl_alert_event_log as l
left join stl_scan as s on s.query = l.query and s.slice = l.slice
and s.segment = l.segment and s.step = l.step
where l.event_time >=  dateadd(day, -7, current_Date)
group by 1,3,4
order by 2 desc,6 desc;
```

确定统计数据缺失的表

以下查询提供了对缺少统计信息的表运行的查询的计数。如果此查询返回任何行，请查看 plannode 值来确定受影响的表，然后对其运行 ANALYZE。

```
select substring(trim(plannode),1,100) as plannode, count(*)
from stl_explain
where plannode like '%missing statistics%'
group by plannode
order by 2 desc;
```

查询故障排除

本节提供快速参考，帮助您识别和解决一些在使用 Amazon Redshift 查询时可能遇到的最常见问题和最严重问题。

主题

- 连接失败
- 查询挂起
- 查询耗时过长
- 加载失败
- 加载耗时过长
- 加载数据不正确
- 设置 JDBC 提取大小参数

这些建议为您提供进行故障排除的起点。您还可以参阅以下资源以获取更多详细信息。

- 访问 Amazon Redshift 集群和数据库
- 使用自动表优化
- 加载数据
- 教程：从 Amazon S3 加载数据

连接失败

由于以下原因，您的查询连接可能会失败；我们建议您使用以下故障排除方法。

客户端无法连接到服务器

如果您使用了 SSL 或服务器证书，请在开始排查连接问题之前将其删除以降低复杂性。待您找到解决方案后，再重新添加 SSL 或服务器证书。有关更多信息，请转至《Amazon Redshift 管理指南》中的配置连接的安全选项。

连接被拒绝

通常情况下，当您收到提示连接建立失败的错误消息时，意味着您没有访问集群的权限。有关更多信息，请转至《Amazon Redshift 管理指南》中的连接被拒绝或失败。

查询挂起

由于以下原因，您的查询可能会挂起或停止响应；我们建议您使用以下故障排除方法。

到数据库的连接中断

减小最大传输单元 (MTU) 的大小。MTU 大小确定可通过网络连接在单个以太网帧中传输的数据包的最大大小（以字节为单位）。有关更多信息，请转至《Amazon Redshift 管理指南》中的至数据库的连接被删除。

到数据库的连接超时

在运行 COPY 命令等较长的查询时，客户端到数据库的连接会挂起或超时。此时，您可能会发现，Amazon Redshift 控制台显示查询已完成，而客户端工具仍然显示正在运行查询。查询结果可能会丢失或不完整，具体取决于连接停止的时间。如果中间网络组件终止空闲连接，则会出现这种情况。有关更多信息，请转至《Amazon Redshift 管理指南》中的防火墙超时问题。

ODBC 出现客户端内存不足错误

如果您的客户端应用程序使用 ODBC 连接，并且您的查询创建的结果集太大，无法存储到内存中，则可使用光标将结果集流式传输到客户端应用程序。有关更多信息，请参阅 DECLARE和 使用游标时的性能注意事项。

JDBC 出现客户端内存不足错误

因此，尝试通过 JDBC 连接检索大型结果集时，可能会遇到客户端内存不足错误。有关更多信息，请参阅设置 JDBC 提取大小参数。

可能存在死锁

如果存在潜在死锁，请尝试以下操作：

- 查看 STV_LOCKS 和 STL_TR_CONFLICT 系统表，查找涉及对多个表的更新的冲突。
- 使用 PG_CANCEL_BACKEND 函数可取消一个或多个冲突查询。
- 使用 PG_TERMINATE_BACKEND 函数终止会话，这会强制使已终止会话中的任意当前正在运行的查询释放所有死锁并回滚事务。
- 仔细计划并发写入操作。有关更多信息，请参阅管理并发写入操作。

查询耗时过长

由于以下原因，您的查询可能需要太长时间；我们建议您使用以下故障排除方法。

表未优化

设置表的排序键、分配方式和压缩编码，以充分利用并行处理。有关更多信息，请参阅使用自动表优化。

查询正在写入到磁盘

您的查询可能至少在部分查询执行中写入磁盘。有关更多信息，请参阅提高查询性能。

查询必须等待其他查询完成

您可以通过创建查询队列并将不同类型的查询分配给适当的队列来提高整体系统性能。有关更多信息，请参阅实施工作负载管理。

查询未优化

分析解释计划，找到重写查询或优化数据库的机会。有关更多信息，请参阅查询计划。

查询需要更多内存才能运行

如果特定查询需要更多内容，您可以通过增大 wlm_query_slot_count 来增加可用内存。

数据库需要运行 VACUUM 命令

当添加、删除或修改大量行时，运行 VACUUM 命令，除非按排序键顺序加载数据。VACUUM 命令会重新组织您的数据，以维持排序顺序和还原性能。有关更多信息，请参阅对表执行 vacuum 操作。

对长时间运行的查询进行故障排除的其他资源

以下是有助于优化查询的系统视图主题和其他文档部分：

- STV_INFLIGHT 系统视图显示集群上正在运行哪些查询。将它与 STV_RECENTS 一起使用有助于确定哪些查询当前正在运行或最近已完成。
- SYS_QUERY_HISTORY 对故障排除很有用。它可显示 DDL 和 DML 查询的相关属性，例如它们的当前状态（如 `running` 或 `failed`）、每个查询运行所花的时间，以及查询是否在并发扩展集群上运行。
- STL_QUERYTEXT 捕获 SQL 命令的查询文本。此外，SVV_QUERY_INFLIGHT（可将 STL_QUERYTEXT 联接到 STV_INFLIGHT）显示了更多的查询元数据。
- 事务锁定冲突可能是查询性能问题的根源。有关当前持有表锁的事务的信息，请参阅SVV_TRANSACTIONS。
- 确定最适合优化的查询提供了一个故障排除查询，可帮助您确定最近运行的哪些查询最耗时。这可以帮助您将精力集中在需要改进的查询上。

- 如果您想进一步探索查询管理并了解如何管理查询队列，实施工作负载管理显示了如何实现。工作负载管理是一项高级功能，在大多数情况下，我们建议使用自动工作负载管理。

加载失败

您的数据加载可能会因以下原因而失败；我们建议您使用以下故障排除方法。

数据源在不同的 AWS 区域

预设情况下，COPY 命令中指定的 Amazon S3 桶或 Amazon DynamoDB 表必须位于集群所在的 AWS 区域。如果您的数据和集群位于不同的区域，您将收到如下所示的错误消息：

```
The bucket you are attempting to access must be addressed using the specified endpoint.
```

尽量确保集群和数据源位于同一区域。您可以通过在 COPY 命令中使用 REGION 选项来指定其他区域。

> **Note**
>
> 如果集群和数据源位于不同的 AWS 区域，则会产生数据传输费用。此外，您还会经历更高的延迟。

COPY 命令失败

查询 STL_LOAD_ERRORS 以发现在特定加载期间发生的错误。有关更多信息，请参阅STL_LOAD_ERRORS。

加载耗时过长

由于以下原因，加载操作可能需要太长时间；我们建议采用以下故障排除方法。

COPY 从单个文件中加载数据

将加载数据拆分为多个文件。如果从一个大型文件加载所有数据，Amazon Redshift 必须执行序列化加载，这样速度很慢。文件数应为集群中切片数量的倍数，且文件大小应大致相同，压缩后介于 1 MB 和 1 GB 之间。有关更多信息，请参阅设计查询的 Amazon Redshift 最佳实践。

加载操作使用多个 COPY 命令

如果您使用多个并发 COPY 命令从多个文件加载一个表，会强制 Amazon Redshift 执行序列化加载，这样速度慢得多。在这种情况下，请使用单个 COPY 命令。

加载数据不正确

您的 COPY 操作可以通过以下方式加载不正确的数据；我们建议使用以下故障排除方法。

加载错误的文件

使用对象前缀指定数据文件可能会导致读取不需要的文件。因此，请使用清单文件准确指定要加载的文件。有关更多信息，请参阅 COPY 命令中的 copy_from_s3_manifest_file 选项和 COPY 示例中的 Example: COPY from Amazon S3 using a manifest。

设置 JDBC 提取大小参数

预设情况下，JDBC 驱动程序一次收集查询的所有结果。因此，尝试通过 JDBC 连接检索大型结果集时，可能遇到客户端内存不足错误。为使您的客户端按批检索结果集，而不是在单个“要么全部检索，要么失败”提取中检索结果集，请在客户端应用程序中设置 JDBC 提取大小参数。

> **Note**
>
> ODBC 不支持提取大小。

为获得最佳性能，请将提取大小设置为不会导致内存不足错误的最大值。较小的提取大小值会导致更多的服务器通信，从而延长执行时间。服务器会预留资源，包括 WLM 查询槽和关联内存，直到客户端检索到整个结果集或查询取消为止。如果适当优化提取大小，则可以更快释放这些资源，使其能够供其他查询使用。

> **Note**
>
> 如果需要提取大型数据集，建议使用 UNLOAD 语句将数据传输到 Amazon S3。使用 UNLOAD 时，计算节点并行工作，以加快数据的传输。

有关设置 JDBC 提取大小参数的更多信息，请参阅 PostgreSQL 文档中的基于光标获取结果。

实施工作负载管理

您可以使用工作负载管理 (WLM) 定义多个查询队列并在运行时将查询路由到适当的队列。

在一些情况下，您可能有多个会话或用户同时运行查询。在这些情况下，某些查询可能会长时间占用集群资源，从而影响其他查询的性能。例如，假设一组用户时不时提交复杂、耗时的查询（从多个大型表中选择和排序行）。另一组用户经常提交短查询（仅从一个或两个表中选择少量行，运行时长只有数秒）。这种情况下，短时查询可能不得不在队列中等待耗时查询完成。WLM 帮助应对这种情况。

您可以配置 Amazon Redshift WLM 以通过自动 WLM 或手动 WLM 运行。

自动 WLM

为了最大限度地提高系统吞吐量和高效地使用资源，您可以启用 Amazon Redshift 来管理如何划分资源以使用自动 WLM 运行并行查询。自动 WLM 管理运行查询所需的资源。Amazon Redshift 确定有多少个查询并发运行以及向每个分派的查询分配多少内存。您可以使用 Amazon Redshift 控制台，依次选择切换 WLM 模式和自动 WLM，从而启用自动 WLM。使用此选项，最多八个队列用于管理查询，并且内存和主要并发字段均设置为自动。您可以指定优先级来反映映射到每个队列的工作负载或用户的业务优先级。默认的查询优先级设置为正常。有关如何更改队列中查询的优先级的信息，请参阅查询优先级。有关更多信息，请参阅 实施自动 WLM。

在运行时，您可以根据用户组或查询组将查询路由到这些队列。您还可以配置查询监控规则 (QMR) 来限制长时间运行的查询。

使用并发扩展和自动 WLM，您可以支持几乎无限的并发用户和并发查询，同时提供始终如一的快速查询性能。有关更多信息，请参阅 使用并发扩展。

> **Note**
>
> 我们建议您创建参数组并选择自动 WLM 来管理查询资源。有关如何从手动 WLM 迁移到自动 WLM 的详细信息，请参阅 从手动 WLM 迁移到自动 WLM。

手动 WLM

此外，您可以通过修改 WLM 配置，为耗时查询和短时查询分别创建队列，来管理系统性能和用户体验。在运行时，您可以根据用户组或查询组将查询路由到这些队列。使用 Amazon Redshift 控制台切换到手动 WLM即可启用此手动配置。使用此选项，您可以指定用于管理查询的队列，以及内存和主要

并发字段值。使用手动配置，您可以配置多达八个查询队列，并设置可在每个队列中同时运行的查询数。

您可以设置规则以根据运行查询的用户或指定的标签将查询路由到特定的队列。您还可以配置分配到每个队列的内存量，使大型查询在内存更多的队列中运行。您还可以配置查询监控规则 (QMR) 来限制长时间运行的查询。有关更多信息，请参阅 实施手动 WLM。

> **Note**
>
> 我们建议您为手动 WLM 查询队列配置不超过 15 个查询槽。有关更多信息，请参阅 并发级别。

WLM 排队限制

请注意，对于手动 WLM 配置，您可以分配给队列的最大插槽数为 50。但是，这并不意味着在自动 WLM 配置中，Amazon Redshift 集群总是会同时运行 50 个查询。根据集群上的内存需求或其他类型的资源分配，这一数字会发生改变。

自动 WLM 和手动 WLM 的使用案例

当您希望 Amazon Redshift 管理如何划分资源以运行并发查询时，请使用自动 WLM。相比手动 WLM，使用自动 WLM 通常会导致更高的吞吐量。使用自动 WLM，您可以为队列中的工作负载定义查询优先级。有关查询优先级的更多信息，请参阅查询优先级。

如果您想对并发性进行更多控制，请使用手动 WLM。

主题

- 修改 WLM 配置
- 实施自动 WLM
- 实施手动 WLM
- 使用并发扩展
- 使用短查询加速
- WLM 队列分配规则
- 为队列分配查询
- WLM 动态和静态配置属性
- WLM 查询监控规则
- WLM 系统表和视图

修改 WLM 配置

修改 WLM 配置的最简单方法是使用 Amazon Redshift 控制台。您也可以使用 AWS CLI 或 Amazon Redshift API。

在自动和手动 WLM 之间切换集群时，集群将进入 `pending reboot` 状态。在下次集群重新启动之前，更改不会生效。

有关修改 WLM 配置的详细信息，请参阅《Amazon Redshift 管理指南》中的配置工作负载管理。

从手动 WLM 迁移到自动 WLM

为了最大限度地提高系统吞吐量和高效地使用资源，我们建议您为队列设置自动 WLM。考虑采用以下方法设置从手动 WLM 到自动 WLM 的平滑过渡。

要从手动 WLM 迁移到自动 WLM 并使用查询优先级，我们建议您创建新参数组，然后将该参数组附加到集群。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 参数组。

> **Important**
>
> 要更改参数组或从手动切换到自动 WLM，需要重新启动集群。有关更多信息，请参阅 WLM 动态和静态配置属性。

我们举一个例子，其中有三个手动 WLM 队列。每个队列分别对应于 ETL 工作负载、分析工作负载和数据科学工作负载。ETL 工作负载每 6 小时运行一次，分析工作负载全天运行，而数据科学工作负载可能随时激增。使用手动 WLM，您可以根据您对每个工作负载对业务重要性的理解来指定每个工作负载队列所获得的内存和并发性。指定内存和并发性不仅难以弄清楚，而且还会导致对集群资源静态分区，使得只有一部分工作负载在运行，造成了资源浪费。

您可以将自动 WLM 与查询优先级一起使用，以指示工作负载的相对优先级，从而避免前面的问题。对于此示例，请执行以下步骤：

- 创建一个新参数组并切换到自动 WLM模式。
- 为三个工作负载中的每一个添加队列：ETL 工作负载、分析工作负载和数据科学工作负载。为与手动 WLM模式一起使用的每个工作负载使用相同的用户组。
- 将 ETL 工作负载的优先级设置为 `High`，将分析工作负载的优先级设置为 `Normal`，并将数据科学工作负载的优先级设置为 `Low`。这些优先级反映不同工作负载或用户组的业务优先级。

- （可选）为分析或数据科学队列启用并发扩展，以便即使 ETL 工作负载每 6 小时运行一次，这些队列中的查询也能获得一致的性能。

使用查询优先级，当只有分析工作负载在集群上运行时，它会获取整个系统的资源来运行。这样可以在更好的系统利用率下实现高吞吐量。但是，当 ETL 工作负载启动时，它会获得先行权，因为它具有更高的优先级。除了在被接纳之后的优先资源分配之外，作为 ETL 工作负载的一部分运行的查询在被接纳期间也会获得优先级。因此，无论系统上运行的是什么其他内容，ETL 工作负载都可以按预测的方式执行。高优先级工作负载的可预测性能是以其他较低优先级工作负载为代价的，这些工作负载运行时间更长，因为它们的查询在等待更重要的查询完成。或者，因为当它们与更高优先级的查询同时运行时，它们获得的资源比例较小。Amazon Redshift 使用的调度算法有助于让较低优先级的查询不会遇到没有资源的情况，而是继续执行，尽管速度较慢。

> **Note**
>
> - 超时字段在自动 WLM 中不可用。请改用 QMR 规则 `query_execution_time`。有关更多信息，请参阅 WLM 查询监控规则。
> - QMR 操作 HOP 不适用于自动 WLM。请改用 `change priority` 操作。有关更多信息，请参阅 WLM 查询监控规则。
> - 集群使用自动 WLM 和手动 WLM 队列的方式不同，这可能会导致混淆配置。例如，您可以在自动 WLM 队列中配置优先级属性，但不能在手动 WLM 队列中配置该属性。因此，在一个参数组中，应避免混用自动 WLM 队列和手动 WLM 队列。而是在迁移到自动 WLM 时创建新参数组。

实施自动 WLM

使用自动工作负载管理 (WLM)，Amazon Redshift 管理查询并发性和内存分配。您可以使用服务类标识符 100-107 创建最多八个队列。每个队列都有一个优先级。有关更多信息，请参阅 查询优先级。

自动 WLM 确定查询所需的资源量，并根据工作负载调整并发性。当系统中有查询需要大量资源（例如，较大表之间的哈希联接）时，并发性较低。当提交了较轻量的查询（例如，插入、删除、扫描或简单聚合）时，并发性较高。

自动 WLM 与短查询加速 (SQA) 分开，它以不同方式评估查询。自动 WLM 和 SQA 协同工作，即使长时间运行、资源密集型的查询处于活动状态，也可以完成短期运行和轻量级查询。有关 SQA 的更多信息，请参阅使用短查询加速。

Amazon Redshift 通过参数组启用自动 WLM：

- 如果您的集群使用默认参数组，则 Amazon Redshift 为它们启用自动 WLM。
- 如果您的集群使用自定义参数组，可以配置您的集群以启用自动 WLM。我们建议您为自动 WLM 配置创建单独的参数组。

要配置 WLM，请编辑参数组中可与一个或多个集群关联的 `wlm_json_configuration` 参数。有关更多信息，请参阅 修改 WLM 配置。

您可以在 WLM 配置中定义查询队列。您可以向默认 WLM 配置添加其他查询队列，最多可添加八个用户队列。您可以为每个查询队列配置以下内容：

- 优先级
- 并发扩展模式
- 用户组
- 查询组
- 查询监控规则

优先级

您可以通过设置优先级值来定义查询在工作负载中的相对重要性。为队列指定优先级，并由与队列关联的所有查询继承。有关更多信息，请参阅 查询优先级。

并发扩展模式

启用并发扩展后，Amazon Redshift 会在需要时自动增加额外的集群容量来处理增多的并发读取查询。不管查询在主集群上运行还是在并发扩展集群上运行，用户都将看到最新的数据。

您可以通过配置 WLM 队列来管理将哪些查询发送到并发扩展集群。为队列启用并发扩展后，符合条件的查询将发送到并发扩展集群，而不是排队等待。有关更多信息，请参阅 使用并发扩展。

用户组

您可以通过指定每个用户组的名称或使用通配符将一组用户组分配给某个队列。当所列用户组的成员运行某个查询时，该查询将在相应的队列中运行。您可以向队列分配任意数量的用户组。有关更多信息，请参阅 根据用户组为队列分配查询。

查询组

您可以通过指定每个队列组的名称或使用通配符将一组查询组分配给某个队列。查询组 只是一种标签。在运行时，您可以将查询组标签分配给一系列查询。分配给所列查询组的任意查询都将在相应的队列中运行。您可以向队列分配任意数量的查询组。有关更多信息，请参阅 为查询组分配查询。

通配符

如果在 WLM 队列配置中启用了通配符，则可以单独地或使用 Unix shell 样式的通配符向队列分配用户组和查询组。模式匹配不区分大小写。

例如，“*”通配符字符匹配任意数量的字符。因此，如果您将 dba_* 添加到某个队列的用户组列表中，则属于名称以开头的 dba_ 组的所有用户查询都将分配到该队列。示例包括 dba_admin 或 DBA_primary。“?”通配符匹配任意单个字符。因此，如果队列包括用户组 dba?1，则名为 dba11 和 dba21 的用户组匹配，但 dba12 不匹配。

默认情况下，未启用通配符。

查询监控规则

查询监控规则为 WLM 查询定义基于指标的性能界限，并指定在查询超出这些界限时需要采取的操作。例如，对于短时间运行查询专用的队列，您可创建取消运行超过 60 秒的查询的规则。要跟踪设计不佳的查询，您可创建记录包含嵌套循环的查询的其他规则。有关更多信息，请参阅 WLM 查询监控规则。

检查自动 WLM

要检查是否启用了自动 WLM，请运行以下查询。如果查询返回至少一行内容，则说明自动 WLM 已启用。

```
select * from stv_wlm_service_class_config
where service_class >= 100;
```

以下查询显示遍历每个查询队列（服务类）的查询数量。它还显示平均执行时间、等待时间排在第九十百分位数的查询数量以及平均等待时间。自动 WLM 查询使用服务类 100 到 107。

```
select final_state, service_class, count(*), avg(total_exec_time),
percentile_cont(0.9) within group (order by total_queue_time), avg(total_queue_time)
from stl_wlm_query where userid >= 100 group by 1,2 order by 2,1;
```

要查明自动 WLM 运行并成功完成哪些查询，请运行以下查询。

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class >= 100 and a.final_state = 'Completed'
order by b.query desc limit 5;
```

查询优先级

并非所有查询都具有同等重要性，并且通常一个工作负载或一组用户的性能可能更重要。如果您启用了自动 WLM，可以通过设置优先级值来定义查询在工作负载中的相对重要性。为队列指定优先级，并由与队列关联的所有查询继承。通过将用户组和查询组映射到队列，可以将查询与队列相关联。您可以设置以下优先级（从最高优先级到最低优先级列出）：

1. HIGHEST
2. HIGH
3. NORMAL
4. LOW
5. LOWEST

当具有不同优先级的查询争用相同的资源时，管理员使用这些优先级来显示其工作负载的相对重要性。Amazon Redshift 在将查询放入系统时使用优先级，并确定分配给查询的资源量。默认情况下，查询运行时的优先级设置为 NORMAL。

额外的优先级 CRITICAL（优先级高于 HIGHEST）适用于超级用户。要设置此优先级，您可以使用函数 CHANGE_QUERY_PRIORITY、CHANGE_SESSION_PRIORITY 和 CHANGE_USER_PRIORITY。要授予数据库用户使用这些函数的权限，您可以创建存储过程并向用户授予权限。有关示例，请参阅 CHANGE_SESSION_PRIORITY。

> **Note**
>
> 一次只能运行一个 CRITICAL 查询。

我们来看个例子，其中，提取、转换、加载 (ETL) 工作负载的优先级高于分析工作负载的优先级。ETL 工作负载每六个小时运行一次，分析工作负载全天运行。当只有分析工作负载在集群上运行时，它会使整个系统自身产生高吞吐量和最佳系统利用率。但是，当 ETL 工作负载启动时，它会获得先行权，因为它具有更高的优先级。除了在被接纳之后的优先资源分配之外，作为 ETL 工作负载的一部分运行的

查询在被接纳期间也会获得先行权。因此，无论系统上运行的是什么其他内容，ETL 工作负载都可以按预测的方式执行。因此，它提供了可预测的性能，并为管理员提供了为其业务用户提供服务级别协议 (SLA) 的能力。

在给定的集群中，高优先级工作负载的可预测性能是以其他优先级较低的工作负载为代价的。较低优先级的工作负载可能运行时间更长，因为其查询正在等待更重要的查询完成。或者，因为当它们与更高优先级的查询同时运行时，它们获得的资源比例较小，因此运行时间可能较长。较低优先级的查询不会遭受资源匮乏，而是继续以较慢的速度取得进展。

在前面的示例中，管理员可以为分析工作负载启用并发扩展。即使 ETL 工作负载以高优先级运行，执行此操作也可以使分析工作负载保持其吞吐量。

配置队列优先级

如果已启用自动 WLM，则每个队列都具有优先级值。查询基于用户组和查询组路由至队列 先将队列优先级设置为 NORMAL。根据与队列的用户组和查询组关联的工作负载，将优先级设置为更高或更低。

您可以在 Amazon Redshift 控制台上更改队列的优先级。在 Amazon Redshift 控制台上，工作负载管理页面显示队列并支持编辑队列属性（如优先级）。要使用 CLI 或 API 操作设置优先级，请使用 wlm_json_configuration 参数。有关更多信息，请参阅《Amazon Redshift 管理指南》中的配置工作负载管理。

以下 wlm_json_configuration 示例定义了三个用户组（ingest、reporting 和 analytics）。来自其中一个组的用户提交的查询分别以优先级 highest、normal 和 low 运行。

```
[
    {
        "user_group": [
            "ingest"
        ],
        "priority": "highest",
        "queue_type": "auto"
    },
    {
        "user_group": [
            "reporting"
        ],
        "priority": "normal",
        "queue_type": "auto"
    },
    {
        "user_group": [
```

```
            "analytics"
        ],
        "priority": "low",
        "queue_type": "auto",
        "auto_wlm": true
    }
]
```

使用查询监控规则更改查询优先级

查询监视规则 (QMR) 使您可以根据查询运行时的行为更改查询的优先级。您可以通过在 QMR 谓词中指定优先级属性以及某项操作来达成上述目的。有关更多信息，请参阅 WLM 查询监控规则。

例如，您可以定义一条规则以取消任何分类为 high 优先级且运行超过 10 分钟的查询。

```
"rules" :[
  {
    "rule_name":"rule_abort",
    "predicate":[
      {
        "metric_name":"query_cpu_time",
        "operator":">",
        "value":600
      },
      {
        "metric_name":"query_priority",
        "operator":"=",
        "value":"high"
      }
    ],
    "action":"abort"
  }
]
```

另一个示例是定义一条规则，以将当前优先级为 lowest 且磁盘溢出超过 1 TB 的任何查询的查询优先级更改为 normal。

```
"rules":[
  {
    "rule_name":"rule_change_priority",
    "predicate":[
      {
```

```
        "metric_name":"query_temp_blocks_to_disk",
        "operator":">",
        "value":1000000
      },
      {
        "metric_name":"query_priority",
        "operator":"=",
        "value":"normal"
      }
    ],
    "action":"change_query_priority",
    "value":"lowest"
  }
]
```

监控查询优先级

要显示正在等待和运行的查询的优先级，请查看 stv_wlm_query_state 系统表中的 query_priority 列。

```
query    | service_cl | wlm_start_time             | state            | queue_time |
 query_priority
---------+------------+----------------------------+------------------+------------
+----------------
2673299  | 102        | 2019-06-24 17:35:38.866356 | QueuedWaiting    | 265116     |
 Highest
2673236  | 101        | 2019-06-24 17:35:33.313854 | Running          | 0          |
 Highest
2673265  | 102        | 2019-06-24 17:35:33.523332 | Running          | 0          |
 High
2673284  | 102        | 2019-06-24 17:35:38.477366 | Running          | 0          |
 Highest
2673288  | 102        | 2019-06-24 17:35:38.621819 | Running          | 0          |
 Highest
2673310  | 103        | 2019-06-24 17:35:39.068513 | QueuedWaiting    | 62970      |
 High
2673303  | 102        | 2019-06-24 17:35:38.968921 | QueuedWaiting    | 162560     |
 Normal
2673306  | 104        | 2019-06-24 17:35:39.002733 | QueuedWaiting    | 128691     |
 Lowest
```

要列出已完成的查询的查询优先级，请参阅 stl_wlm_query 系统表中的 query_priority 列。

```
select query, service_class as svclass, service_class_start_time as starttime,
 query_priority
from stl_wlm_query order by 3 desc limit 10;
```

```
  query   | svclass |         starttime          |    query_priority
---------+---------+----------------------------+----------------------
 2723254 |     100 | 2019-06-24 18:14:50.780094 | Normal
 2723251 |     102 | 2019-06-24 18:14:50.749961 | Highest
 2723246 |     102 | 2019-06-24 18:14:50.725275 | Highest
 2723244 |     103 | 2019-06-24 18:14:50.719241 | High
 2723243 |     101 | 2019-06-24 18:14:50.699325 | Low
 2723242 |     102 | 2019-06-24 18:14:50.692573 | Highest
 2723239 |     101 | 2019-06-24 18:14:50.668535 | Low
 2723237 |     102 | 2019-06-24 18:14:50.661918 | Highest
 2723236 |     102 | 2019-06-24 18:14:50.643636 | Highest
```

为了优化工作负载的吞吐量，Amazon Redshift 可能会修改用户提交查询的优先级。Amazon Redshift 使用高级机器学习算法来确定此优化何时有利于您的工作负载，并在满足以下所有条件时自动应用此优化。

- 自动 WLM 启用。
- 只定义了一个 WLM 队列。
- 您尚未定义用于设置查询优先级的查询监控规则 (QMR)。这些规则包括 QMR 指标 `query_priority` 或 QMR 操作 `change_query_priority`。有关更多信息，请参阅 WLM 查询监控规则。

实施手动 WLM

通过手动 WLM，您可以通过修改 WLM 配置，为耗时查询和短时查询分别创建队列，来管理系统性能和用户体验。

当用户在 Amazon Redshift 中运行查询时，查询会路由到查询队列。每个查询队列都包含很多查询槽。每个队列都分配有一部分集群可用内存。队列的内存在队列的查询槽间分配。您可以允许 Amazon Redshift 通过自动 WLM 管理查询并发性。有关更多信息，请参阅 实施自动 WLM。

或者，您可以为每个查询队列配置 WLM 属性。这样可以指定如何在槽间分配内存，以及如何在运行时将查询路由到特定的队列。您还可以配置 WLM 属性以取消长时间运行的查询。

预设情况下，Amazon Redshift 配置以下查询队列：

- 一个超级用户队列

  超级用户队列是专为超级用户预留的队列，无法进行配置。仅在需要运行影响系统的查询或用于故障排除目的时，才应使用该队列。例如，如果需要取消用户的耗时查询或向数据库添加用户，则可使用该队列。请不要使用它来执行常规查询。该队列不显示在控制台中，但在数据库的系统表中显示为第五队列。要在超级用户队列中运行查询，用户必须以超级用户身份登录并使用预定义的 `superuser` 查询组运行查询。

- 一个默认用户队列

  默认队列初始配置为并发运行五个查询。在使用手动 WLM 时，您可以更改默认队列的并发、超时和内存分配属性，但不能指定用户组或查询组。默认队列必须是 WLM 配置中最后一个队列。未路由到其他队列的查询在默认队列中运行。

查询队列在 WLM 配置中定义。WLM 配置是参数组中的可编辑参数 (`wlm_json_configuration`)，可与一个或多个集群关联。有关更多信息，请参阅《Amazon Redshift 管理指南》中的配置工作负载管理。

您可以向默认 WLM 配置添加其他查询队列，最多可添加八个用户队列。您可以为每个查询队列配置以下内容：

- 并发扩展模式
- 并发级别
- 用户组
- 查询组
- 要使用的 WLM 内存百分比
- WLM 超时
- WLM 查询队列跳过
- 查询监控规则

并发扩展模式

启用并发扩展后，Amazon Redshift 会在需要时自动增加额外的集群容量来处理增多的并发读取查询。不管查询在主集群上运行还是在并发扩展集群上运行，用户都将看到最新的数据。

您可以通过配置 WLM 队列来管理将哪些查询发送到并发扩展集群。为队列启用并发扩展后，符合条件的查询将发送到并发扩展集群，而不是排队等待。有关更多信息，请参阅 使用并发扩展。

并发级别

队列中的查询以并发方式运行，直到它们达到为该队列定义的 WLM 队列插槽计数或并发 级别。达到并发级别后，后续查询将在队列中等待。

> **Note**
>
> WLM 并发级别不同于一个集群可以拥有的并行用户连接数。有关更多信息，请参阅《Amazon Redshift 管理指南》中的连接到集群。

在自动 WLM 配置（推荐）中，并发级别设置为自动。Amazon Redshift 为查询动态分配内存，这会随之确定同时运行多少个查询。此数量基于运行查询和排队查询所需的资源。自动 WLM 不可配置。有关更多信息，请参阅 实施自动 WLM。

在手动 WLM 配置中，Amazon Redshift 向每个队列静态分配固定数量的内存。队列的内存在查询槽之间均匀分配。举例来说，如果向队列分配了 20% 的集群内存并且有 10 个插槽，则每个查询将分配 2% 的集群内存。内存分配保持固定，不论并行运行的查询数量是多少。由于此固定内存分配，对于插槽计数为 5 时可完全在内存中运行的查询，如果将其插槽计数提高到 20，则可能需要将中间结果写入磁盘。在这个例子中，每个查询的队列内存份额从 1/5 减少到 1/20。额外的磁盘 I/O 会降低性能。

所有用户定义的队列的最大插槽计数为 50。这限制了所有队列的总插槽数，包括默认队列。唯一不受此限制的队列是预留的超级用户队列。

默认情况下，手动 WLM 队列的并发级别为 5。在某些情况下，更高的并发级别可能对您的工作负载有用，如下所示：

- 如果有许多小型查询被迫等待耗时的查询，则可创建一个具有更高插槽计数的独立队列，并将小型查询分配给该队列。并发级别较高的队列，分配给每个查询槽的内存较少，不过较小的查询需要的内存较少。

  > **Note**
  >
  > 如果您启用短查询加速 (SQA)，WLM 会自动让短时查询优先于长时查询，因此对于大多数工作流程，不需要单独的队列来进行短查询。有关更多信息，请参阅 使用短查询加速。

- 如果您有多个查询，每个查询访问单个切片上的数据，则可设置独立的 WLM 队列，以并发执行这些查询。Amazon Redshift 将并发查询分配给多个独立的切片，使多个查询能够在多个切片上并行执行。例如，如果某个查询是简单的聚合操作，且使用基于分配键的谓词，则该查询的数据将位于单个切片上。

手动 WLM 示例

此示例是一个简单的手动 WLM 场景，显示了如何分配插槽和内存。您对以下三个队列实施手动 WLM：

- data-ingestion 队列 – 此队列设置用于提取数据。为该队列分配了 20% 的集群内存，并且该队列有 5 个插槽。因此，队列中可以同时运行 5 个查询，每个查询分配了 4% 的集群内存。
- data-scientist 队列 – 专为内存密集型查询设计。为该队列分配了 40% 的集群内存，并且该队列有 5 个插槽。因此可以同时运行 5 个查询，每个查询分配了 8% 的集群内存。
- default 队列 – 这是为组织中大多数用户设计的队列。包括销售和会计团队，他们会运行短时间或中等时间长度且并不复杂的查询。为该队列分配了 40% 的集群内存，并且该队列有 40 个插槽。此队列中可以同时运行 40 个查询，每个查询分配了 1% 的集群内存。这是可以为此队列分配的最大插槽数，因为所有队列的总数限制为 50。

如果您正在运行 WLM，并且工作负载需要并行运行 15 个以上的查询，我们建议您启用并发扩展。这是因为将查询槽数增加到 15 以上可能会导致系统资源争用，限制了单个集群的总吞吐量。利用并发扩展，您可以并行运行数百个查询，最多可达到配置的并发扩展集群数。并发扩展集群的数量由 max_concurrency_scaling_clusters 控制。有关并发扩展的更多信息，请参阅使用并发扩展。

有关更多信息，请参阅 提高查询性能。

用户组

您可以通过指定每个用户组的名称或使用通配符将一组用户组分配给某个队列。当所列用户组的成员运行某个查询时，该查询将在相应的队列中运行。您可以向队列分配任意数量的用户组。有关更多信息，请参阅 根据用户组为队列分配查询。

查询组

您可以通过指定每个队列组的名称或使用通配符将一组查询组分配给某个队列。查询组只是一种标签。在运行时，您可以将查询组标签分配给一系列查询。分配给所列查询组的任意查询都将在相应的队列中运行。您可以向队列分配任意数量的查询组。有关更多信息，请参阅 为查询组分配查询。

通配符

如果在 WLM 队列配置中启用了通配符，则可以单独地或使用 Unix shell 样式的通配符向队列分配用户组和查询组。模式匹配不区分大小写。

例如，“*”通配符字符匹配任意数量的字符。因此，如果您将 dba_* 添加到某个队列的用户组列表中，则属于名称以开头的 dba_ 组的所有用户查询都将分配到该队列。示例包括 dba_admin 或 DBA_primary。“?”通配符匹配任意单个字符。因此，如果队列包括用户组 dba?1，则名为 dba11 和 dba21 的用户组匹配，但 dba12 不匹配。

默认情况下，通配符处于关闭状态。

要使用的 WLM 内存百分比

在自动 WLM 配置中，内存百分比设置为 **auto**。有关更多信息，请参阅 实施自动 WLM。

在手动 WLM 配置中，要指定分配给查询的可用内存量，您可以设置 WLM Memory Percent to Use 参数。默认情况下，会向每个用户定义队列分配相等的用户定义查询可用内存。例如，如果有四个用户定义队列，则会向每个队列分配 25% 的可用内存。超级用户队列有自己的分配内存，无法进行修改。要更改分配，您可以向每个队列分配整数比例的内存，总计最高为 100%。任意未分配的内存将由 Amazon Redshift 进行管理，如果有队列请求更多内存以进行处理，可以临时分配给该队列。

例如，如果配置了四个队列，则可以按以下方式分配内存：20%、30%、15%、15%。其余的 20% 未分配，由该服务管理。

WLM 超时

WLM 超时 (max_execution_time) 已弃用。相反，使用 query_execution_time 创建查询监控规则 (QMR) 来限制经过的查询执行时间。有关更多信息，请参阅 WLM 查询监控规则。

要限制允许查询在给定 WLM 队列中停留的时间，您可以为每个队列设置 WLM 超时。超时参数指定 Amazon Redshift 在取消或跳过查询前等待查询执行的时间量（单位为毫秒）。超时基于查询执行时间，不包括在队列中等待的时间。

WLM 尝试跳过 CREATE TABLE AS (CTAS) 语句和只读查询，例如 SELECT 语句。无法跳过的查询将被取消。有关更多信息，请参阅 WLM 查询队列跳过。

查询已进入“returning”状态时，WLM 超时不适用。要查看查询的状态，请参阅 STV_WLM_QUERY_STATE 系统表。COPY 语句和维护操作（例如 ANALYZE 和 VACUUM）不受 WLM 超时约束。

WLM 超时功能类似于 statement_timeout 配置参数。区别在于，statement_timeout 配置参数应用于整个集群，WLM 超时特定于 WLM 配置中的单个队列。

如果还指定了 statement_timeout，则会使用 statement_timeout 和 WLM timeout (max_execution_time) 中的较小者。

查询监控规则

查询监控规则为 WLM 查询定义基于指标的性能界限，并指定在查询超出这些界限时需要采取的操作。例如，对于短时间运行查询专用的队列，您可创建取消运行超过 60 秒的查询的规则。要跟踪设计不佳的查询，您可创建记录包含嵌套循环的查询的其他规则。有关更多信息，请参阅 WLM 查询监控规则。

WLM 查询队列跳过

由于 WLM 超时或查询监控规则 (QMR) 跃点操作，可能会跳过查询。您只能在手动 WLM 配置中跳转查询。

当跳过查询时，WLM 会尝试根据 WLM 队列分配规则将该查询路由到下一个匹配的队列。如果查询不与任何其他队列定义匹配，则查询会被取消。它不会分配给默认队列。

WLM 超时操作

下表总结了具有 WLM 超时的不同类型查询的行为。

查询类型	操作
INSERT、UPDATE 和 DELETE	Cancel
用户定义的函数 (UDF)	Cancel
UNLOAD	Cancel
COPY	继续执行
维护操作	继续执行
处于 `returning` 状态的只读查询	继续执行
处于 `running` 状态的只读查询	重新分配或重新启动
CREATE TABLE AS (CTAS)、SELECT INTO	重新分配或重新启动

WLM 超时队列跳过

如果下列类型的查询超时，WLM 会跳过它们：

- 只读查询，例如 WLM 状态为 `running` 的 SELECT 语句。要了解查询的 WLM 状态，请查看 STV_WLM_QUERY_STATE 系统表中的 STATE 列。
- CREATE TABLE AS (CTAS) 语句。WLM 队列跳跃支持用户定义的和系统生成的 CTAS 语句。
- SELECT INTO 语句。

不受 WLM 超时约束的查询将继续在原始队列中运行，直到完成为止。以下类型的查询不受 WLM 超时约束：

- COPY 语句
- 维护操作，例如 ANALYZE 和 VACUUM
- 只读查询，例如 WLM 状态达到 `returning` 的 SELECT 语句。要了解查询的 WLM 状态，请查看 STV_WLM_QUERY_STATE 系统表中的 STATE 列。

没有资格因 WLM 超时跳过的查询将在超时时被取消。下列类型的查询没有资格因 WLM 超时跳过：

- INSERT、UPDATE 和 DELETE 语句
- UNLOAD 语句
- 用户定义的函数 (UDF)

WLM 超时重新分配和重新启动的查询

当跳过查询并且找不到匹配的队列时，该查询将被取消。

当跳过查询并且找到匹配的队列时，WLM 会尝试将该查询重新分配给新队列。如果无法重新分配查询，则它会在新队列中重新启动，如下所述。

只有在满足以下所有条件时，才会重新分配查询：

- 找到匹配的队列。
- 新队列有足够多的空闲插槽来运行查询。如果 wlm_query_slot_count 参数设置为大于 1 的值，则查询可能需要多个插槽。
- 新队列具有至少与查询当前使用的一样多的可用内存。

如果重新分配查询，则查询将继续在新队列中执行。将会保留中间结果，这样对总体执行时间造成的影响可降至最低。

如果无法重新分配查询，则该查询会被取消并在新队列中重新启动。中间结果会被删除。查询会在队列中等待，然后在有足够多的插槽可用时开始运行。

QMR 跳过操作

下表总结了具有 QMR 跃点操作的不同类型查询的行为。

查询类型	操作
COPY	继续执行
维护操作	继续执行
用户定义的函数 (UDF)	继续执行
UNLOAD	重新分配或继续执行
INSERT、UPDATE 和 DELETE	重新分配或继续执行
处于 `returning` 状态的只读查询	重新分配或继续执行
处于 `running` 状态的只读查询	重新分配或重新启动
CREATE TABLE AS (CTAS)、SELECT INTO	重新分配或重新启动

要查明由 QMR 跳过的查询是否已重新分配、重新启动或取消，请查询 STL_WLM_RULE_ACTION 系统日志表。

QMR 跳过操作重新分配和重新启动的查询

当跳过查询并且找不到匹配的队列时，该查询将被取消。

当跳过查询并且找到匹配的队列时，WLM 会尝试将该查询重新分配给新队列。如果无法重新分配查询，则它会在新队列中重新启动或在原始队列中继续执行，如下所述。

只有在满足以下所有条件时，才会重新分配查询：

- 找到匹配的队列。

- 新队列有足够多的空闲插槽来运行查询。如果 wlm_query_slot_count 参数设置为大于 1 的值，则查询可能需要多个插槽。
- 新队列具有至少与查询当前使用的一样多的可用内存。

如果重新分配查询，则查询将继续在新队列中执行。将会保留中间结果，这样对总体执行时间造成的影响可降至最低。

如果无法重新分配查询，则该查询会重新启动或在原始队列中继续执行。如果无法重新启动查询，则该查询会被取消并在新队列中重新启动。中间结果会被删除。查询会在队列中等待，然后在有足够多的插槽可用时开始执行。

教程：配置手动工作负载管理 (WLM) 队列

概述

我们建议您在 Amazon Redshift 中配置自动工作负载管理 (WLM)。有关自动 WLM 的更多信息，请参阅实施工作负载管理。但是，如果您需要多个 WLM 队列，则本教程将逐步指导您完成在 Amazon Redshift 中配置手动工作负载管理 (WLM) 的过程。通过配置手动 WLM，您可以改进集群中的查询性能和资源分配。

Amazon Redshift 将用户查询路由至队列以进行处理。WLM 将定义这些查询路由至队列的方式。预设情况下，Amazon Redshift 具有两个可用于查询的队列：一个面向超级用户，一个面向用户。超级用户队列无法进行配置且一次只能处理一个查询。您应保留此队列以仅作故障排除之用。用户队列可一次处理多达 5 个查询，但您可以在需要时通过更改队列的并发级别来配置此能力。

若您有多个用户正在对数据库运行查询，您可能会发现另一种配置将更高效。例如，如果一些用户运行资源密集型操作（如 VACUUM），则这些操作可能会对资源不太密集型查询（如报告）产生负面影响。您可考虑添加其他队列并针对不同的工作负载配置它们。

估计时间：75 分钟

估算费用：50 美分

先决条件

您将需要一个 Amazon Redshift 集群、示例 TICKIT 数据库和 Amazon Redshift RSQL 客户端工具。如果您还没有设置上述这些项，请访问 Amazon Redshift 入门指南和 Amazon Redshift RSQL。

Sections

- 第 1 节：了解默认队列处理行为

- 第 2 节：修改 WLM 查询队列配置
- 第 3 节：根据用户组和查询组将查询路由到队列
- 第 4 节：使用 wlm_query_slot_count 临时覆盖队列中的并发级别
- 第 5 节：清理资源

第 1 节：了解默认队列处理行为

在您开始配置手动 WLM 前，了解 Amazon Redshift 中队列处理的默认行为是很有用的。在此节中，您将创建两个从多个系统表中返回信息的数据库视图。然后，您将运行一些测试查询以了解预设情况下路由查询的方式。有关系统表的更多信息，请参阅系统表和视图参考。

步骤 1：创建 WLM_QUEUE_STATE_VW 视图

在此步骤中，您将创建一个名为 WLM_QUEUE_STATE_VW 的视图。此视图返回以下系统表中的信息。

- STV_WLM_CLASSIFICATION_CONFIG
- STV_WLM_SERVICE_CLASS_CONFIG
- STV_WLM_SERVICE_CLASS_STATE

您将在整个教程中使用此视图来监控在更改 WLM 配置后队列将发生的情况。下表描述了 WLM_QUEUE_STATE_VW 视图返回的数据。

列	描述
queue	与表示一个队列的行关联的编号。队列编号确定了队列在数据库中的顺序。
description	一个值，用于描述队列是仅适用于某些用户组、某些查询组还是所有类型的查询。
slots	分配给队列的槽位数量。
mem	分配给队列的内存量（以每个槽的 MB 数为单位）。
max_execution_time	在查询终止之前允许其运行的时间量。
user_*	一个值，用于指示是否允许在 WLM 配置中使用通配符来匹配用户组。

列	描述
query_*	一个值，用于指示是否允许在 WLM 配置中使用通配符来匹配查询组。
queued	正在队列中等待进行处理的查询的数量。
executing	当前正在运行的查询数。
executed	已运行的查询数。

创建 WLM_QUEUE_STATE_VW 视图

1. 打开 Amazon Redshift RSQL 并连接到您的 TICKIT 示例数据库。如果您没有此数据库，请参阅先决条件。
2. 运行以下查询以创建 WLM_QUEUE_STATE_VW 视图。

```
create view WLM_QUEUE_STATE_VW as
select (config.service_class-5) as queue
, trim (class.condition) as description
, config.num_query_tasks as slots
, config.query_working_mem as mem
, config.max_execution_time as max_time
, config.user_group_wild_card as "user_*"
, config.query_group_wild_card as "query_*"
, state.num_queued_queries queued
, state.num_executing_queries executing
, state.num_executed_queries executed
from
STV_WLM_CLASSIFICATION_CONFIG class,
STV_WLM_SERVICE_CLASS_CONFIG config,
STV_WLM_SERVICE_CLASS_STATE state
where
class.action_service_class = config.service_class
and class.action_service_class = state.service_class
and config.service_class > 4
order by config.service_class;
```

3. 运行以下查询以查看该视图包含的信息。

```
select * from wlm_queue_state_vw;
```

下面是示例结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(querytype: any)	5	836	0	false	false	0	1	160

步骤 2：创建 WLM_QUERY_STATE_VW 视图

在此步骤中，您将创建一个名为 WLM_QUERY_STATE_VW 的视图。此视图返回 STV_WLM_QUERY_STATE 系统表中的信息。

您将在整个教程中使用此视图来监控正在运行的查询。下表描述了 WLM_QUERY_STATE_VW 视图返回的数据。

列	描述
query	查询 ID。
queue	队列编号。
slot_count	分配给查询的槽位数量。
start_time	启动查询的时间。
state	查询的状态，如“正在执行”。
queue_time	查询已在队列中等待的微秒数。
exec_time	查询处于运行中的微秒数。

创建 WLM_QUERY_STATE_VW 视图

1. 在 RSQL 中，运行以下查询以创建 WLM_QUERY_STATE_VW 视图。

```
create view WLM_QUERY_STATE_VW as
select query, (service_class-5) as queue, slot_count, trim(wlm_start_time) as
 start_time, trim(state) as state, trim(queue_time) as queue_time, trim(exec_time) as
 exec_time
from stv_wlm_query_state;
```

2. 运行以下查询以查看该视图包含的信息。

```
select * from wlm_query_state_vw;
```

下面是示例结果。

query	queue	slot_count	start_time	state	queue_time	exec_time
1249	1	1	2014-09-24 22:19:16	Executing	0	516

步骤 3：运行测试查询

在此步骤中，您将在 RSQL 中通过多个连接运行查询，并查看系统表以确定路由查询以进行处理的方式。

在此步骤中，您将需要打开两个 RSQL 窗口：

- 在 RSQL 窗口 1 中，您将运行用于监控队列状态的查询和使用已在本教程中创建的视图的查询。
- 在 RSQL 窗口 2 中，您将运行长时间运行的查询以更改在 RSQL 窗口 1 中找到的结果。

运行测试查询

1. 打开两个 RSQL 窗口。如果您已打开一个窗口，则只需打开另一个窗口。您可针对这两个连接使用同一用户账户。
2. 在 RSQL 窗口 1 中，运行以下查询。

```
select * from wlm_query_state_vw;
```

下面是示例结果。

query	queue	slot_count	start_time	state	queue_time	exec_time
1258	1	1	2014-09-24 22:21:03	Executing	0	549

此查询返回一个自引用结果。当前正在运行的查询是来自此视图的 SELECT 语句。对此视图的查询将始终至少返回一个结果。将此结果与在下一步中启动长时间运行的查询后产生的结果进行比较。

3. 在 RSQL 窗口 2 中，从 TICKIT 示例数据库运行以下查询。此查询应会运行约一分钟时间，因此您有时间来探究之前创建的 WLM_QUEUE_STATE_VW 视图和 WLM_QUERY_STATE_VW 视图的结果。在某些情况下，您可能会发现查询运行的时长不足以查询这两个视图。在这些情况下，您可以增加 `l.listid` 上的筛选条件值使其运行时间更长。

> **Note**
>
> 为了缩短查询执行时间并改进系统性能，Amazon Redshift 在领导节点的内存中缓存特定查询类型的结果。当启用结果缓存时，后续查询的运行速度会快得多。要阻止查询快速运行，请对当前会话禁用结果缓存。

要为当前会话禁用结果缓存，请将 enable_result_cache_for_session 参数设置为 off，如下所示。

```
set enable_result_cache_for_session to off;
```

在 RSQL 窗口 2 中，运行以下查询。

```
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <
 100000;
```

4. 在 RSQL 窗口 1 中，查询 WLM_QUEUE_STATE_VW 和 WLM_QUERY_STATE_VW 并将结果与之前的结果进行比较。

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

以下是示例结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(querytype: any)	5	836	0	false	false	0	2	163

query	queue	slot_count	start_time	state	queue_time	exec_time
1267	1	1	2014-09-24 22:22:30	Executing	0	684
1265	1	1	2014-09-24 22:22:26	Executing	0	4080859

请注意，之前的查询结果与本步骤中的结果之间存在以下区别：

- WLM_QUERY_STATE_VW 中现在有两行。一个结果是用于对此视图运行 SELECT 操作的自引用查询。第二个结果是上一步中的长时间运行的查询。
- WLM_QUEUE_STATE_VW 中的 executing 列已从 1 增加到 2。此列条目表示队列中有两个正在运行的查询。

- executed 列在您每次在队列中运行一个查询时将会递增。

WLM_QUEUE_STATE_VW 视图可用于全面了解队列以及每个队列中正在处理的查询的数量。WLM_QUERY_STATE_VW 视图可用于更详细地了解当前正在运行的单个查询。

第 2 节：修改 WLM 查询队列配置

现在您了解了队列的默认工作方式，您将了解如何使用手动 WLM 配置查询队列。在此节中，您将为集群创建并配置新的参数组。您将创建另外两个用户查询并配置它们以基于查询的用户组或查询组标签接受查询。未路由至这两个队列之一的任何查询在运行时将会路由至默认队列。

要在参数组中创建手动 WLM 配置

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择 Configurations（配置），然后选择 Workload management（工作负载管理），以显示 Workload management（工作负载管理）页面。
3. 选择 Create (创建) 显示 Create parameter group (创建参数组) 窗口。
4. 为 Parameter group name（参数组名称）和 Description（描述）输入 **WLMTutorial**，然后选择 Create（创建）以创建参数组。

   > **Note**
   >
   > 参数组名称会在创建时转换为全部小写格式。

5. 在 Workload management（工作负载管理）页面上，选择参数组 **wlmtutorial** 以显示详细信息页面，其中包含 Parameters（参数）和 Workload management（工作负载管理）选项卡。
6. 确认您正在 Workload management（工作负载管理）选项卡，然后选择 Switch WLM mode（切换 WLM 模式）以显示 Concurrency settings（并发设置）窗口。
7. 选择 Manual WLM（手动 WLM），然后选择 Save（保存）以切换到手动 WLM。
8. 选择 Edit workload queues（编辑工作负载队列）。
9. 选择 Add queue（添加队列）两次以添加两个队列。现在有三个队列：队列 1、队列 2 和默认队列。
10. 按如下方式输入每个队列的信息：

    - 对于队列 1，为内存 (%) 输入 **30**，为主集群上的并发输入 **2**，并为查询组输入 **test**。将其他设置保留为默认值。

- 对于队列 2，为内存 (%) 输入 **40**，为主集群上的并发输入 **3**，并为用户组输入 **admin**。将其他设置保留为默认值。
- 不要对默认队列做出任何更改。WLM 将未分配的内存分配给默认队列。

11. 选择 Save (保存) 以保存您的设置。

接下来，将具有手动 WLM 配置的参数组与集群相关联。

要将具有手动 WLM 配置的参数组与集群关联

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择 Clusters（集群），然后选择 Clusters（集群）以显示集群的列表。
3. 选择您的集群，例如 `examplecluster`，以显示集群详细信息。然后选择 Properties（属性）选项卡，以显示该集群的属性。
4. 在 Database configurations（数据库配置）部分中，选择 Edit（编辑）、Edit parameter group（编辑参数组）以显示参数组窗口。
5. 对于参数组，选择您之前创建的 **wlmtutorial** 参数组。
6. 选择 Save changes（保存更改）以关联参数组。

   使用更改的参数组修改集群。但是，您需要重启集群才能将更改应用于数据库。
7. 选择您的集群，然后为 Actions（操作）选择 Reboot（重启）。

在重启集群后，状态将返回为 Available（可用）。

第 3 节：根据用户组和查询组将查询路由到队列

现在，您的集群已与一个新的参数组相关联，并且您已配置 WLM。接下来，运行一些查询，以了解 Amazon Redshift 如何将查询路由到队列进行处理。

步骤 1：在数据库中查看查询队列配置

首先，验证数据库具有您希望的 WLM 配置。

查看查询队列配置

1. 打开 RSQL 并运行以下查询。此查询使用您在步骤 1：创建 WLM_QUEUE_STATE_VW 视图中创建的 WLM_QUEUE_STATE_VW 视图。如果您在集群重启之前已将会话连接到数据库，则将需要进行重新连接。

```
select * from wlm_queue_state_vw;
```

下面是示例结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(query group: test)	2	627	0	false	false	0	0	0
2	(user group: admin)	3	557	0	false	false	0	0	0
3	(querytype: any)	5	250	0	false	false	0	1	0

将这些结果您与步骤 1：创建 WLM_QUEUE_STATE_VW 视图中获得的结果进行比较。请注意，现在有另外两个队列。队列 1 现在是面向 test 查询组的队列，而队列 2 则是面向 admin 用户组的队列。

队列 3 现在是默认队列。列表中的最后一个队列始终是默认队列。这是在查询中未指定任何用户组或查询组的情况下默认将查询路由至的队列。

2. 运行以下查询以确认您的查询现在在队列 3 中运行。

```
select * from wlm_query_state_vw;
```

下面是示例结果。

query	queue	slot_count	start_time	state	queue_time	exec_time
2144	3	1	2014-09-24 23:49:59	Executing	0	550430

步骤 2：使用查询组队列运行查询

使用查询组队列运行查询

1. 运行以下查询以将其路由至 `test` 查询组。

```
set query_group to test;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. 在另一个 RSQL 窗口中，运行以下查询。

```
select * from wlm_query_state_vw;
```

下面是示例结果。

query	queue	slot_count	start_time	state	queue_time	exec_time
2168	1	1	2014-09-24 23:54:18	Executing	0	6343309
2170	3	1	2014-09-24 23:54:24	Executing	0	847

查询已路由至 test 查询组，现在是队列 1。

3. 从队列状态视图中选择全部。

```
select * from wlm_queue_state_vw;
```

您将看到类似以下内容的结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(query group: test)	2	627	0	false	false	0	1	0
2	(user group: admin)	3	557	0	false	false	0	0	0
3	(querytype: any)	5	250	0	false	false	0	1	3

4. 现有，重置查询组并再次运行长时间运行的查询：

```
reset query_group;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

5. 针对视图运行查询以查看结果。

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

以下是示例结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(query group: test)	2	627	0	false	false	0	0	1
2	(user group: admin)	3	557	0	false	false	0	0	0
3	(querytype: any)	5	250	0	false	false	0	2	5

query	queue	slot_count	start_time	state	queue_time	exec_time
2186	3	1	2014-09-24 23:57:52	Executing	0	649
2184	3	1	2014-09-24 23:57:48	Executing	0	4137349

结果应是查询现在正在队列 3 中再次运行。

步骤 3：创建数据库用户和组

在您可以在此队列中运行任何查询之前，需要在数据库中创建用户组并向该组添加用户。然后，您将使用新用户的凭证通过 RSQL 进行登录，然后运行查询。您需要作为超级用户（如管理员用户）运行查询来创建数据库用户。

创建新的数据库用户和用户组

1. 在数据库中，通过在 RSQL 窗口中运行以下命令来创建名为 adminwlm 的新数据库用户。

```
create user adminwlm createuser password '123Admin';
```

2. 然后，运行以下命令来创建新用户组并将您的新 adminwlm 用户添加到该组。

```
create group admin;
alter group admin add user adminwlm;
```

步骤 4：使用用户组队列运行查询

接下来，您将运行一个查询并将该查询路由至用户组队列。当您想要将查询路由至配置为处理您要运行的查询类型的队列时，可执行此操作。

使用用户组队列运行查询

1. 在 RSQL 窗口 2 中，运行以下查询以切换至 adminwlm 账户并以该用户的身份运行查询。

```
set session authorization 'adminwlm';
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. 在 RSQL 窗口 1 中，运行以下查询以查看查询路由至的查询队列。

```
select * from wlm_query_state_vw;
select * from wlm_queue_state_vw;
```

以下是示例结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(query group: test)	2	627	0	false	false	0	0	1
2	(user group: admin)	3	557	0	false	false	0	1	0
3	(querytype: any)	5	250	0	false	false	0	1	8

query	queue	slot_count	start_time	state	queue_time	exec_time
2202	2	1	2014-09-25 00:01:38	Executing	0	4885796
2204	3	1	2014-09-25 00:01:43	Executing	0	650

此查询是在队列 2（admin 用户队列）中运行的。无论何时您以此用户的身份登录来运行查询，这些查询均将在队列 2 中运行，除非您指定要使用另一个查询组。所选队列取决于队列分配规则。有关更多信息，请参阅 WLM 队列分配规则。

3. 现在，从 RSQL 窗口 2 中运行以下查询。

```
set query_group to test;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

4. 在 RSQL 窗口 1 中，运行以下查询以查看查询路由至的查询队列。

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

以下是示例结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(query group: test)	2	627	0	false	false	0	1	1
2	(user group: admin)	3	557	0	false	false	0	0	1
3	(querytype: any)	5	250	0	false	false	0	1	10

query	queue	slot_count	start_time	state	queue_time	exec_time
2218	1	1	2014-09-25 00:04:30	Executing	0	4819666
2220	3	1	2014-09-25 00:04:35	Executing	0	685

5. 当您完成后，请重置查询组。

```
reset query_group;
```

第 4 节：使用 wlm_query_slot_count 临时覆盖队列中的并发级别

有时，用户可能会为某个特定查询临时需要更多资源。如果是这样的话，他们可以使用 wlm_query_slot_count 配置设置来临时覆盖查询队列中分配槽位的方式。槽位 是用于处理查询的内存

和 CPU 的单位。您可在偶尔使用消耗集群中大量资源的查询时（例如，在数据库中执行 VACUUM 操作时）覆盖槽位计数。

您可能会发现用户经常需要针对特定类型的查询设置 wlm_query_slot_count。如果是，请考虑调整 WLM 配置并为用户提供更适合其查询需求的队列。有关通过使用槽位计数临时覆盖并发级别的更多信息，请参阅wlm_query_slot_count。

步骤 1：使用 wlm_query_slot_count 覆盖并发级别

在本教程中，我们将运行同一个长时间运行的 SELECT 查询。我们将以 adminwlm 用户的身份运行查询，并使用 wlm_query_slot_count 增加查询的可用槽位数。

使用 wlm_query_slot_count 覆盖并发级别

1. 提高对查询的限制以确保您有足够的时间来查询 WLM_QUERY_STATE_VW 视图并查看结果。

```
set wlm_query_slot_count to 3;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. 现在，可使用管理员用户查询 WLM_QUERY_STATE_VW，以查看查询的运行情况。

```
select * from wlm_query_state_vw;
```

下面是示例结果。

query	queue	slot_count	start_time	state	queue_time	exec_time
2240	2	3	2014-09-25 00:08:45	Executing	0	3731414
2242	3	1	2014-09-25 00:08:49	Executing	0	596

请注意，查询的槽位计数是 3。此计数表示查询正在使用所有三个槽位来处理查询，并将队列中的所有资源分配给查询。

3. 现在，运行以下查询。

```
select * from WLM_QUEUE_STATE_VW;
```

下面是示例结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(query group: test)	2	627	0	false	false	0	0	4
2	(user group: admin)	3	557	0	false	false	0	1	3
3	(querytype: any)	5	250	0	false	false	0	1	25

wlm_query_slot_count 配置设置仅对当前会话有效。如果会话过期或者另一用户运行查询，则使用 WLM 配置。

4. 重置槽位计数并重新运行测试。

```
reset wlm_query_slot_count;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

以下是示例结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(query group: test)	2	627	0	false	false	0	0	2
2	(user group: admin)	3	557	0	false	false	0	1	2
3	(querytype: any)	5	250	0	false	false	0	1	14

query	queue	slot_count	start_time	state	queue_time	exec_time
2260	2	1	2014-09-25 00:12:11	Executing	0	4042618
2262	3	1	2014-09-25 00:12:15	Executing	0	680

步骤 2：从不同的会话中运行查询

接下来，从不同会话中运行查询。

从不同的会话中运行查询

1. 在 RSQL 窗口 1 和 2 中，运行以下查询以使用测试查询组。

```
set query_group to test;
```

2. 在 RSQL 窗口 1 中，运行以下长时间运行的查询。

```
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

3. 由于长时间运行的查询仍在 RSQL 窗口 1 中运行，请运行以下查询。这些命令将增加槽位计数，以使用队列的所有槽位，然后开始运行长时间运行的查询。

```
set wlm_query_slot_count to 2;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

4. 打开第三个 RSQL 窗口并查询视图以查看结果。

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

以下是示例结果。

queue	description	slots	mem	max_time	user_*	query_*	queued	executing	executed
0	(super user) and (query group: superuser)	1	357	0	false	false	0	0	0
1	(query group: test)	2	627	0	false	false	1	1	2
2	(user group: admin)	3	557	0	false	false	0	0	3
3	(querytype: any)	5	250	0	false	false	0	1	18

query	queue	slot_count	start_time	state	queue_time	exec_time
2286	1	2	2014-09-25 00:16:48	QueuedWaiting	3758950	0
2282	1	1	2014-09-25 00:16:33	Executing	0	19335850
2288	3	1	2014-09-25 00:16:52	Executing	0	666

请注意，第一个查询正在使用分配给队列 1 的槽位之一来运行查询。此外，请注意，队列中有一个查询正在等待（其中 queued 为 1，state 为 QueuedWaiting）。在第一个查询完成之后，第二个查询即会开始运行。此执行发生的原因在于：两个查询均已路由至 test 查询组，而且第二个查询必须等待有足够多的槽位才能开始进行处理。

第 5 节：清理资源

只要您的集群正在运行，就将继续产生费用。完成本教程后，请按照《Amazon Redshift 入门指南》中的查找其它资源并重置环境中的步骤操作，使您的环境返回上一个状态。

有关 WLM 的更多信息，请参阅 实施工作负载管理。

使用并发扩展

使用并发扩展功能，您可以支持成千上万的并发用户和并发查询，同时提供始终如一的快速查询性能。开启并发扩展后，Amazon Redshift 会自动增加额外的集群容量来处理增多的读取查询和写入查询。不管查询在主集群上运行还是在并发扩展集群上运行，用户都将看到最新的数据。

您可以通过配置 WLM 队列来管理将哪些查询发送到并发扩展集群。开启并发扩展后，符合条件的查询将发送到并发扩展集群，而不是排队等待。

仅当并发扩展集群正在主动运行查询时，您才需要为其付费。有关定价的更多信息，包括费用如何累积和最低费用，请参阅并发扩展定价。

并发扩展功能

为 WLM 队列开启并发扩展时，它将适用于读取操作，如控制面板查询。它还适用于常用的写操作，例如用于数据摄入和处理的语句。

写操作的并发扩展功能

并发扩展支持经常使用的写操作，例如提取、转换和加载 (ETL) 语句。当您希望在集群收到大量请求时保持一致的响应时间时，写操作的并发扩展特别有用。它提高了在主集群上争夺资源的写操作的吞吐量。

并发扩展支持 COPY、INSERT、DELETE、UPDATE 和 CREATE TABLE AS (CTAS) 语句。此外，并发扩展支持不使用聚合的 MV 的实体化视图刷新。不支持其他数据处理语言 (DML) 语句和数据定义语言 (DDL) 语句。如果不支持的写入语句（如 CREATE without TABLE AS）包含在支持的写入语句之前的显式事务中，则所有写入语句都不会在并发扩展集群上运行。

当您为并发扩展累计积分时，此积分应计适用于读取和写操作。

并发扩展的限制

以下是使用 Amazon Redshift 并发扩展的限制：

- 它不支持查询使用交错排序键的表。
- 它不支持查询临时表。
- 它不支持访问受限制性网络或 Virtual Private Cloud (VPC) 配置保护的外部资源的查询。
- 它不支持包含 Python 用户定义函数（UDF）和 Lambda UDF 的查询。
- 它不支持访问系统表、PostgreSQL 目录表或非备份表的查询。
- 在实施限制性 IAM 策略权限时，它不支持访问外部资源的 COPY 或 UNLOAD 查询。这包括应用于资源（例如 Amazon S3 存储桶或 DynamoDB 表）或源的权限。IAM 源可以包括：
  - `aws:sourceVpc` – 一个 VPC 源。
  - `aws:sourceVpce` – 源 VPC 端点。
  - `aws:sourceIp` – 源 IP 地址。

  在某些情况下，您可能需要删除限制资源或源的权限，以便将访问资源的 COPY 和 UNLOAD 查询发送到并发扩展集群。

  有关资源策略的更多信息，请参阅《AWS Identity and Access Management 用户指南》中的策略类型，以及使用存储桶策略控制从 VPC 端点的访问。

- DDL 操作（如 CREATE TABLE 或 ALTER TABLE）不支持 Amazon Redshift 并发扩展用于写操作。
- 它不支持 COPY 命令的 ANALYZE。
- 它不支持对 DISTSTYLE 设置为 ALL 的目标表进行写操作。
- 它不支持以下文件格式的 COPY：
  - Parquet
  - ORC
- 它不支持对具有身份列的表进行写操作。
- Amazon Redshift 支持仅在 Amazon Redshift RA3 节点上执行写操作的并发扩展，特别是 ra3.16xlarge、ra3.4xlarge 和 ra3.xlplus。其他节点类型不支持写操作的并发扩展。

并发扩展的AWS 区域

并发扩展在以下 AWS 区域可用：

- 美国东部（弗吉尼亚北部）区域 (us-east-1)
- 美国东部（俄亥俄）区域 (us-east-2)
- AWS GovCloud（美国东部）
- 美国西部（加利福尼亚北部）区域 (us-west-1)
- 美国西部（俄勒冈州）区域 (us-west-2)
- 亚太地区（孟买）区域 (ap-south-1)
- 亚太地区（首尔）区域 (ap-northeast-2)
- 亚太地区（新加坡）区域 (ap-southeast-1)
- 亚太地区（悉尼）区域 (ap-southeast-2)
- 亚太地区（东京）区域 (ap-northeast-1)
- 加拿大（中部）区域 (ca-central-1)
- 欧洲（法兰克福）区域 (eu-central-1)
- 欧洲（爱尔兰）区域 (eu-west-1)
- 欧洲（伦敦）区域 (eu-west-2)
- 欧洲（巴黎）区域 (eu-west-3)
- 欧洲（斯德哥尔摩）区域 (eu-north-1)

- 南美洲（圣保罗）区域 (sa-east-1)

并发扩展候选项

仅当主集群满足以下要求时，才将查询路由到并发扩展集群：

- EC2-VPC 平台。
- 节点类型必须为 dc2.8xlarge、dc2.large、ra3.xlplus、ra3.4xlarge 或 ra3.16xlarge。仅在以下 Amazon Redshift RA3 节点上支持执行写操作的并发扩展：ra3.16xlarge、ra3.4xlarge 和 ra3.xlplus。
- 对于具有 ra3.xlplus、ra3.4xlarge 或 ra3.16xlarge 节点类型的集群，最多 32 个计算节点。此外，在最初创建集群时，主集群的节点数不能大于 32 个节点。例如，即使集群当前有 20 个节点，但最初创建时具有 40 个节点，它也不符合并发扩展的要求。相反，如果 DC2 集群当前有 40 个节点，但最初创建时具有 20 个节点，则它确实满足并发扩展的要求。
- 非单节点集群。

配置并发扩展队列

通过将工作负载管理器 (WLM) 队列启用为并发扩展队列来将查询路由到并发扩展集群。要在队列上开启并发扩展，请将并发扩展模式值设置为自动。

当路由到并发扩展队列的查询数超过队列配置的并发数时，符合条件的查询将发送到并发扩展集群。当有槽位可用时，将在主集群上运行查询。队列数仅受每集群允许的队列数限制。与任何 WLM 队列一样，您可以根据用户组或通过使用查询组标签标记查询来将查询路由到并发扩展队列。您还可以通过定义 WLM 查询监控规则来路由查询。例如，您可以将所有耗时超过 5 秒的查询路由到并发扩展队列。

并发扩展集群的默认数量为 1。可以使用的并发扩展集群的数量由 max_concurrency_scaling_clusters 控制。

监控并发扩展

您可以通过以下方式查看查询运行在主集群还是并发扩展集群上：在 Amazon Redshift 控制台中，导航到集群，选择一个集群。然后选择查询监控选项卡和工作负载并发以查看有关正在运行的查询和排队查询的信息。

要查找执行时间，请查询 STL_QUERY 表并筛选 `concurrency_scaling_status` 列。以下查询比较在并发扩展集群上运行的查询与在主集群上运行的查询的队列时间和执行时间。

```
SELECT w.service_class AS queue
, CASE WHEN q.concurrency_scaling_status = 1 THEN 'concurrency scaling cluster' ELSE
 'main cluster' END as concurrency_scaling_status
, COUNT( * ) AS queries
, SUM( q.aborted ) AS aborted
, SUM( ROUND( total_queue_time::NUMERIC / 1000000,2) ) AS queue_secs
, SUM( ROUND( total_exec_time::NUMERIC / 1000000,2) ) AS exec_secs
FROM stl_query q
JOIN stl_wlm_query w
USING (userid,query)
WHERE q.userid > 1
AND q.starttime > '2019-01-04 16:38:00'
AND q.endtime < '2019-01-04 17:40:00'
GROUP BY 1,2
ORDER BY 1,2;
```

根据您的要求调整 `starttime` 和 `endtime` 值。

并发扩展系统视图

一组带有前缀 SVCS 的系统视图提供了系统日志表中有关主集群和并发扩展集群上的查询的详细信息。

以下视图具有与相应的 STL 视图或 SVL 视图类似的信息：

- SVCS_ALERT_EVENT_LOG
- SVCS_COMPILE
- SVCS_EXPLAIN
- SVCS_PLAN_INFO
- SVCS_QUERY_SUMMARY
- SVCS_STREAM_SEGS

以下视图专门用于并发扩展。

- SVCS_CONCURRENCY_SCALING_USAGE

有关并发扩展的更多信息，请参阅《Amazon Redshift 管理指南》中的以下主题。

- 查看并发扩展数据

- 查看查询执行期间的集群性能
- 查看查询详细信息

使用短查询加速

短查询加速 (SQA) 让选定的短时查询优先于长时查询。SQA 在专用空间中运行短时查询，因此 SQA 查询不会被迫排在队列中的长时查询后面等待。SQA 仅优先处理用户定义的队列中的短时查询。使用 SQA，短时查询会更快地开始运行，用户会更快地看到结果。

如果您启用 SQA，则可以减少专用于运行短查询的工作负载管理 (WLM) 队列。此外，长时查询无需与短查询竞争队列中的插槽，因此您可以将 WLM 队列配置为使用较少的查询插槽。当您使用较低的并发度时，查询吞吐量会增加，而且大多数工作负载的总体系统性能会得到提高。

CREATE TABLE AS (CTAS) 语句和只读查询 (例如 SELECT 语句) 符合 SQA 资格。

Amazon Redshift 使用机器学习算法分析每个有资格的查询，并预测查询的执行时间。默认情况下，WLM 根据集群的工作负载分析为 SQA 最大运行时动态分配值。或者，您也可以指定一个介于 1-20 秒之间的固定值。如果预测的查询运行时间小于定义或动态分配的 SQA 最大运行时间，并且查询在 WLM 队列中等待，则 SQA 会将查询与 WLM 队列分开并安排优先执行。如果查询运行的时间长于 SQA 最大运行时间，WLM 会根据 WLM 队列分配规则将查询移动到第一个匹配 WLM 队列。随着时间的推移，预测会随着 SQA 从您的查询模式中学习而提高。

默认情况下，为默认参数组和所有新参数组启用 SQA。要在 Amazon Redshift 控制台中禁用 SQA，请编辑参数组的 WLM 配置并取消选择启用短查询加速。作为最佳实践，我们建议您使用 WLM 查询插槽计数 15 或更少，以保持最佳整体系统性能。有关修改 WLM 配置的信息，请参阅《Amazon Redshift 管理指南》中的配置工作负载管理。

短查询的最大运行时间

当您启用 SQA 时，默认情况下 WLM 会将短查询的最大运行时间设置为动态的。我们建议保留 SQA 最大运行时间的动态设置。您可以通过指定一个介于 1-20 秒之间的固定值来覆盖默认设置。

在某些情况下，您可能会考虑对 SQA 最大运行时间值使用不同的值，以提高系统性能。在这些情况下，可分析您的工作负载以查找您的大部分短时查询的最大执行时间。以下查询返回位于大约第七十个百分位数的查询的最大运行时间。

```
select least(greatest(percentile_cont(0.7)
within group (order by total_exec_time / 1000000) + 2, 2), 20)
from stl_wlm_query
```

```
where userid >= 100
and final_state = 'Completed';
```

在确定适合您的工作负载的最大运行时间值后，不需要更改它，除非工作负载发生重大变化。

监控 SQA

要检查是否启用了 SQA，请运行以下查询。如果查询返回一行内容，则说明 SQA 已启用。

```
select * from stv_wlm_service_class_config
where service_class = 14;
```

以下查询显示遍历每个查询队列（服务类）的查询数量。它还显示平均执行时间、等待时间排在第九十百分位数的查询数量以及平均等待时间。SQA 查询使用服务类 14。

```
select final_state, service_class, count(*), avg(total_exec_time),
percentile_cont(0.9) within group (order by total_queue_time), avg(total_queue_time)
from stl_wlm_query where userid >= 100 group by 1,2 order by 2,1;
```

要查明哪些查询由 SQA 选取并成功完成，请运行以下查询。

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class = 14 and a.final_state = 'Completed'
order by b.query desc limit 5;
```

要查找由 SQA 选取但超时的查询，请运行以下查询。

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class = 14 and a.final_state = 'Evicted'
order by b.query desc limit 5;
```

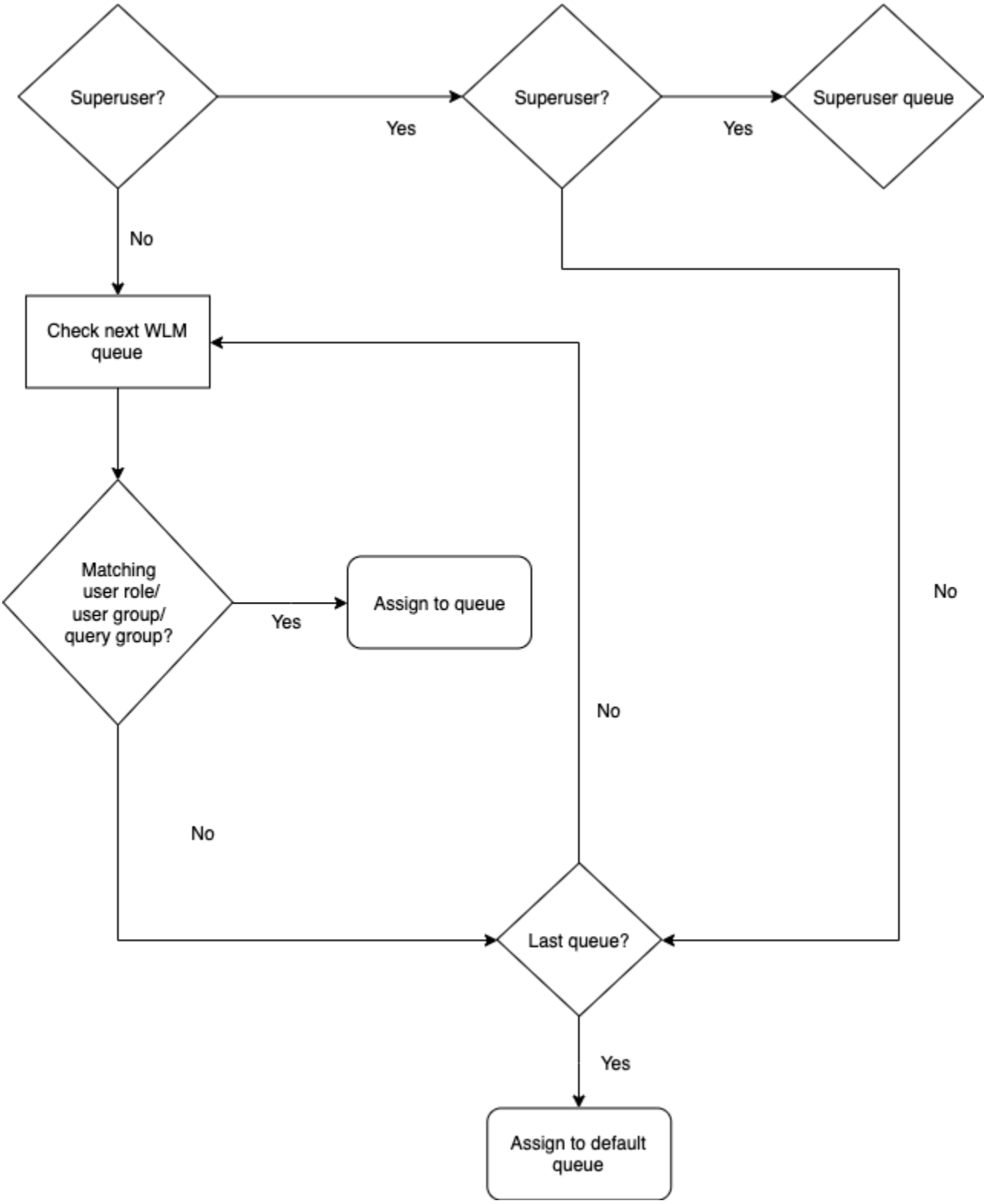
有关已移出的查询的更多信息，更笼统地说，有关可以对查询采取的基于规则的操作的更多信息，请参阅WLM 查询监控规则。

WLM 队列分配规则

当用户运行查询时，WLM 基于 WLM 队列分配规则将查询分配给第一个匹配的队列。

1. 如果用户以超级用户身份登录并在带有超级用户标签的查询组中运行某个查询，则该查询会分配到超级用户队列。
2. 如果用户属于某个角色、属于所列的用户组或在所列查询组中运行某个查询，则该查询会分配给第一个匹配队列。
3. 如果某个查询不满足任何条件，则该查询会分配给默认队列，即 WLM 配置中定义的最后一个队列。

下图说明这些规则的工作方式。

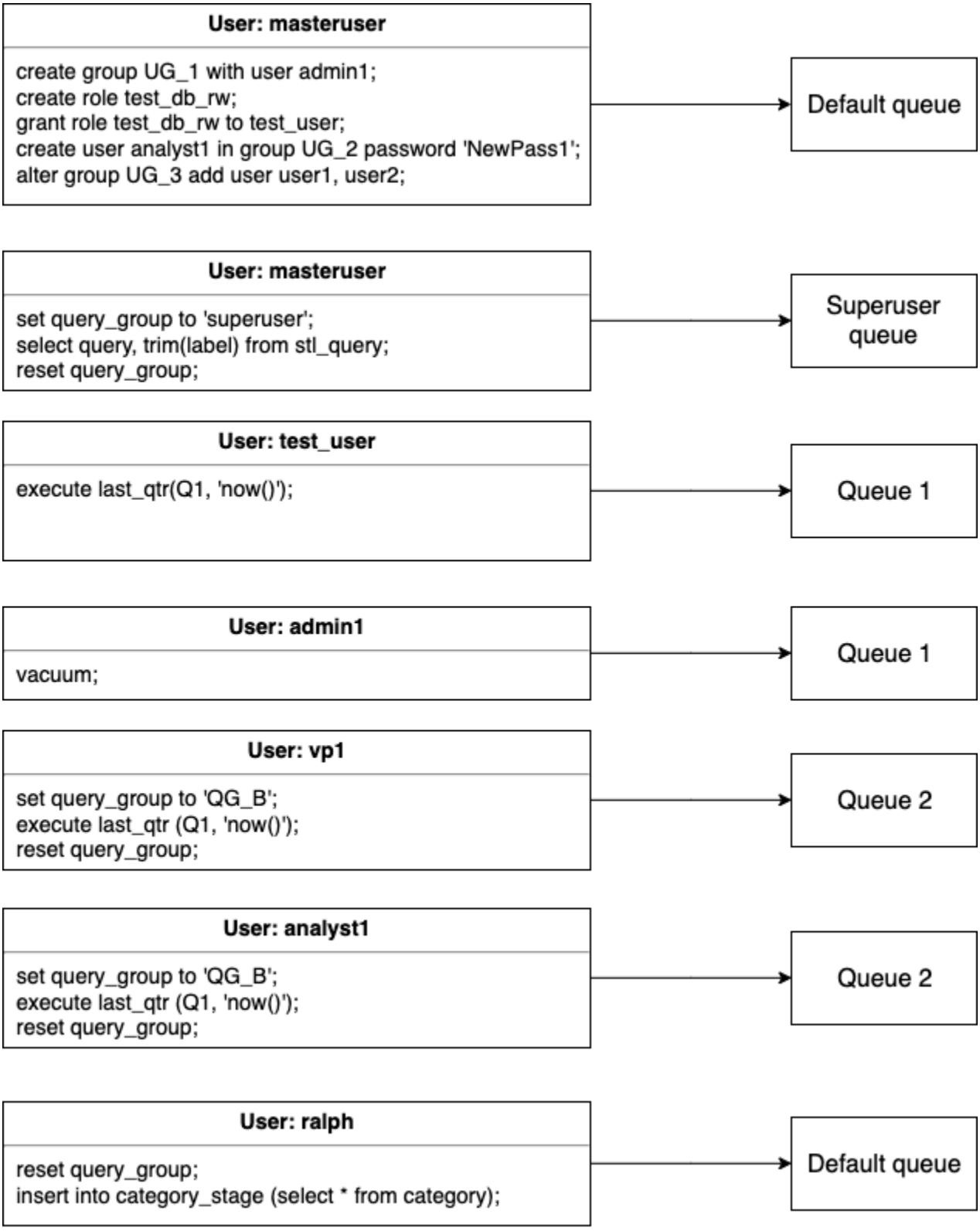


队列分配示例

下表列出了具有超级用户队列和四个用户定义队列的 WLM 配置。

Queue	并发	用户角色	用户组	查询组
Superuser	1			超级用户
1	5	test_db_rw	UG_1	
2	5			QG_B
3	5		UG_2	QG_C
默认	5			

下面的插图介绍系统如何根据用户组和查询组将查询分配给前一个表中的队列。有关如何在运行时将查询分配给用户组和查询组的信息，请参阅本节后面的为队列分配查询。



在本示例中，WLM 进行以下分配：

1. 第一组语句介绍将用户分配给用户组的三种方式。这些语句由用户 adminuser 运行，该用户不是任何 WLM 队列中所列用户组的成员。未设置任何查询组，因此，系统将这些语句路由到默认队列。
2. 用户 adminuser 是超级用户，查询组设置为 'superuser'，因此，该查询分配给超级用户队列。
3. 为用户 test_user 分配了队列 1 中所列的角色 test_db_rw，因此，该查询分配给队列 1。
4. 用户 admin1 是队列 1 中所列用户组的成员，因此，该查询分配给队列 1。
5. 用户 vp1 不是任何所列用户组的成员。查询组设置为 'QG_B'，因此，该查询分配给队列 2。
6. 用户 analyst1 是队列 3 中所列用户组的成员，但 'QG_B' 匹配队列 2，因此，该查询分配给队列 2。
7. 用户 ralph 不是任何所列用户组的成员，且查询组被重置，因此，不存在任何匹配队列。该查询分配给默认队列。

为队列分配查询

以下示例根据用户角色、用户组和查询组向队列分配查询。

根据用户角色为队列分配查询

如果为用户分配某个角色，并且该角色附加到队列，则该用户运行的查询将分配给该队列。以下示例创建了一个名为 sales_rw 的用户角色并将用户 test_user 分配给该角色。

```
create role sales_rw;
grant role sales_rw to test_user;
```

您还可以通过显式将一个角色授予另一个角色来合并两个角色的权限。为用户分配嵌套角色会向该用户授予这两个角色的权限。

```
create role sales_rw;
create role sales_ro;
grant role sales_ro to role sales_rw;
grant role sales_rw to test_user;
```

要查看集群中已授予角色的用户列表，请查询 SVV_USER_GRANTS 表。要查看集群中已授予角色的角色列表，请查询 SVV_ROLE_GRANTS 表。

```
select * from svv_user_grants;
```

```
select * from svv_role_grants;
```

根据用户组为队列分配查询

如果用户组名称列在队列定义中，则由该用户组的成员运行的查询会分配到相应的队列。下面的示例使用 SQL 命令 CREATE USER、CREATE GROUP 和 ALTER GROUP 创建用户组并将用户添加到组。

```
create group admin_group with user admin246, admin135, sec555;
create user vp1234 in group ad_hoc_group password 'vpPass1234';
alter group admin_group add user analyst44, analyst45, analyst46;
```

为查询组分配查询

您可以通过将查询分配到相应的查询组在运行时向队列分配查询。使用 SET 命令开始使用查询组。

```
SET query_group TO group_label
```

其中，*group_label* 是 WLM 配置中列出的查询组标签。

在 SET query_group 命令之后运行的所有查询都以指定查询组的成员身份运行，直到您重置查询组或结束当前登录会话为止。有关设置和重置 Amazon Redshift 对象的信息，请参阅“SQL 命令参考”中的 SET 和 RESET。

您指定的查询组标签必须包含在当前 WLM 配置中；否则，SET query_group 命令对查询队列不起作用。

查询日志会捕获在 TO 子句中定义的标签，因此，您可以利用标签进行故障排除。有关 query_group 配置参数的信息，请参阅“配置参考”中的 query_group。

下面的示例将两个查询作为查询组“priority”的一部分运行，然后重置查询组。

```
set query_group to 'priority';
select count(*)from stv_blocklist;
select query, elapsed, substring from svl_qlog order by query desc limit 5;
reset query_group;
```

为超级用户队列分配查询

要向超级用户队列分配查询，请以超级用户身份登录 Amazon Redshift，然后在超级用户组中运行查询。完成后，重置查询组，使后续查询不会在超级用户队列中运行。

下面的示例分配两个命令以在超级用户队列中运行。

```
set query_group to 'superuser';

analyze;
vacuum;
reset query_group;
```

要查看超级用户的列表，请查询 PG_USER 系统目录表。

```
select * from pg_user where usesuper = 'true';
```

WLM 动态和静态配置属性

WLM 配置属性可以是动态的，也可以是静态的。您可以将动态属性应用于数据库而无需重新启动集群，但静态属性需要重新启动集群才能够使更改生效。不过，如果您同时更改动态属性和静态属性，则必须重新启动集群才能使所有属性更改生效。无论更改的属性是动态还是静态的，都是如此。

在应用动态属性时，您的集群状态是 `modifying`。在自动 WLM 与手动 WLM 之间切换是一项静态更改，需要重启集群才能生效。

下表显示使用自动 WLM 或手动 WLM 时哪些 WLM 属性是动态的或静态的。

WLM 属性	自动 WLM	手动 WLM
查询组	动态	静态
查询组通配符	动态	静态
用户组	动态	静态
用户组通配符	动态	静态
用户角色	动态	静态
用户角色通配符	动态	静态
主要并发	不适用	动态
并发扩展模式	动态	动态

WLM 属性	自动 WLM	手动 WLM
启用短查询加速	不适用	动态
短查询的最大运行时间	动态	动态
要使用的内存的百分比	不适用	动态
超时	不适用	动态
优先级	动态	不适用
添加或移除队列	动态	静态

如果修改查询监控规则 (QMR)，无需修改集群即可自动进行更改。

> **Note**
>
> 使用手动 WLM 时，如果更改了超时值，则新值将应用于更改值之后开始执行的所有查询。如果更改了要使用的并发性或内存百分比，则 Amazon Redshift 将动态更改为新的配置。这样，当前运行的查询不会受到更改的影响。有关更多信息，请参阅 WLM 动态内存分配。

主题

WLM 动态内存分配

在每个队列中，WLM 创建与队列的并发级别相等的查询槽数目。分配到查询槽的内存量等于分配到队列的内存百分比除以槽数。如果您更改内存分配或并发性，Amazon Redshift 会动态管理转变到新 WLM 配置的过程。因此，活动的查询可以使用当前分配的内存量运行直至完成。与此同时，Amazon Redshift 确保总内存使用率不超过可用内存的 100%。

工作负载管理器使用下面的流程管理过渡：

1. WLM 重新计算针对每个新查询槽的内存分配。
2. 如果正在运行的查询未主动使用某个查询槽，则 WLM 删除该槽，使这些内存可供新的槽使用。

3. 如果主动使用某个查询槽，则 WLM 等待查询完成。
4. 活动查询完成后，会删除空槽，释放关联的内存。
5. 因为有充足的内存可供添加一个或多个槽，所以会添加新的槽。
6. 发生更改时正在运行的所有查询完成之后，槽数等于新的并发级别，完成到新的 WLM 配置的过渡。

实际上，发生更改时正在运行的查询将继续使用原始内存分配。对于发生更改时排队中的查询，当有新的槽可用时将路由到这些槽。

如果 WLM 动态属性在过渡期间发生更改，则 WLM 立即开始过渡到新的配置（从当前状态开始）。要查看过渡的状态，请查询 STV_WLM_SERVICE_CLASS_CONFIG 系统表。

动态 WLM 示例

假定您的集群 WLM 使用了以下动态属性配置了两个队列。

Queue	并发	使用的内存百分比
1	4	50%
2	4	50%

现在，假定集群有 200 GB 内存可用于查询处理。（此数字是随机的，只是为了演示目的。）如下面的等式所示，每个槽分配 25 GB 内存。

```
(200 GB * 50% ) / 4 slots  = 25 GB
```

接下来，将 WLM 更改为使用以下动态属性。

Queue	并发	使用的内存百分比
1	3	75%
2	4	25%

如下面的等式所示，队列 1 中每个槽的新内存分配为 50 GB。

```
(200 GB * 75% ) / 3 slots = 50 GB
```

假定在应用新配置时，查询 A1、A2、A3、A4 正在运行，查询 B1、B2、B3、B4 正在排队。WLM 动态重新配置查询槽，如下所示。

步骤	正在运行的查询	当前槽数	目标槽数	分配的内存	可用内存
1	A1、A2、A3、A4	4	0	100 GB	50 GB
2	A2、A3、A4	3	0	75 GB	75 GB
3	A3、A4	2	0	50 GB	100 GB
4	A3、A4、B1	2	1	100 GB	50 GB
5	A4、B1	1	1	75 GB	75 GB
6	A4、B1、B2	1	2	125 GB	25 GB
7	B1、B2	0	2	100 GB	50 GB
8	B1、B2、B3	0	3	150 GB	0 GB

1. WLM 重新计算每个查询槽的内存分配。最初，队列 1 分配 100 GB 内存。新的队列总共分配 150 GB 内存，因此，新队列立即有 50 GB 内存可用。队列 1 现在使用四个槽，新的并发级别为三个槽，因此，不添加任何新槽。
2. 当某个查询完成时，该槽被删除并释放 25 GB 内存。队列 1 现有三个槽和 75 GB 可用内存。新配置需要为每个槽分配 50 GB 内存，但新的并发级别为三个槽，因此不添加任何新的槽。
3. 当第二个查询完成时，其槽被删除并释放 25 GB 内存。队列 1 现有两个槽和 100 GB 可用内存。
4. 使用 50 GB 可用内存添加新槽。队列 1 现有三个槽和 50 GB 可用内存。排队的查询现在可路由到新的槽。
5. 在第三个查询完成时，其槽被删除并释放 25 GB 内存。队列 1 现有两个槽和 75 GB 可用内存。
6. 使用 50 GB 可用内存添加新槽。队列 1 现有三个槽和 25 GB 内存。排队的查询现在可路由到新的槽。

7. 当第四个查询完成时，其槽被删除并释放 25 GB 内存。队列 1 现有两个槽和 50 GB 可用内存。
8. 使用 50 GB 可用内存添加新槽。队列 1 现有三个槽，每个槽有 50 GB 内存，所有可用内存均已分配。

过渡完成，所有查询槽都可用于排队的查询。

WLM 查询监控规则

在 Amazon Redshift 工作负载管理 (WLM) 中，查询监控规则为 WLM 查询定义基于指标的性能界限，并指定在查询超出这些界限时需要采取的操作。例如，对于短时间运行查询专用的队列，您可创建取消运行超过 60 秒的查询的规则。要跟踪设计不佳的查询，您可创建记录包含嵌套循环的查询的其他规则。

您将在工作负载管理 (WLM) 配置中定义查询监控规则。您最多可以为每个队列定义 25 个规则，而且所有队列的限制为 25 个规则。每个规则最多包括三个条件 (即，谓词) 和一个操作。谓词 包含指标、比较条件 (=、< 或 >) 和值。如果满足任何规则的所有谓词，则会触发该规则的操作。可能的规则操作有 log、hop 和 abort，如以下所讨论的。

某个给定队列中的规则只能应用于在该队列中运行的查询。各个规则彼此独立。

WLM 每 10 秒评估一次指标。如果同一期间内触发了多个规则，WLM 将启动最严厉的操作 — abort、hop、然后是 log。如果操作为 hop 或 abort，则将记录操作且从队列中移出查询。如果操作为 log，则查询将继续在队列中运行。WLM 针对每个规则的每个查询仅启用一个 log 操作。如果队列中包含其他规则，这些规则将保持有效。如果操作为 hop 且将查询路由到其他队列，则将应用新队列的规则。有关对特定查询执行的查询监控和跟踪操作的更多信息，请参阅使用短查询加速中的示例集。

当满足规则的所有谓词时，WLM 会在 STL_WLM_RULE_ACTION 系统表中写入一行。此外，Amazon Redshift 会将当前运行的查询的查询指标记录到 STV_QUERY_METRICS。已完成的查询的指标存储在 STL_QUERY_METRICS 中。

定义查询监控规则

您创建作为您的 WLM 配置的一部分的查询监控规则，您可将该规则定义为您集群的参数组定义的一部分。

您可使用 AWS Management Console或以编程方式使用 JSON 来创建规则。

> **Note**
>
> 如果您选择以编程方式创建规则，强烈建议您使用控制台生成包含在参数组定义中的 JSON。有关更多信息，请参阅《Amazon Redshift 管理指南》中的使用控制台创建或修改查询监控规则和使用 AWS CLI 配置参数值。

要定义查询监控规则，您要指定以下元素：

- 一个规则名称 – 规则名称必须在 WLM 配置内是唯一的。规则名称最多可包含 32 个字母数字字符或下划线，且不能包含空格或引号。您可以让每个队列有最多 25 个规则，并且所有队列的总限制为 25 个规则。
- 一个或多个谓词 – 您最多可以为每个规则设置三个谓词。如果满足任何规则的所有谓词，则会触发关联操作。谓词由指标名称、运算符 (=、< 或 >) 和值定义。示例是 `query_cpu_time > 100000`。有关指标列表以及不同指标值的示例，请参阅本部分中后面的预置的 Amazon Redshift 的查询监控指标。
- 操作 – 如果触发了多个操作，则 WLM 会选择具有最严重操作的规则。可能的操作 (按严重性的升序顺序) 包括：
  - Log – 记录有关 STL_WLM_RULE_ACTION 系统表中查询的信息。在您想要仅写入日志记录时使用 Log 操作。WLM 针对每个规则的每个查询最多创建一个日志。log 操作后，其他规则将保持有效且 WLM 将继续监控查询。
  - Hop (仅适用于手动 WLM) – 记录操作，并让查询跳到下一个匹配的队列中。如果没有其他匹配的队列，则会取消查询。QMR 仅跳过 CREATE TABLE AS (CTAS)语句和只读查询，例如 SELECT 语句。有关更多信息，请参阅 WLM 查询队列跳过。
  - 中止 – 记录操作并取消查询。QMR 不会中止 COPY 语句和维护操作，例如 ANALYZE 和 VACUUM。
  - 更改优先级 (仅适用于自动 WLM) – 更改查询的优先级。

要限制查询的运行时，我们建议您创建查询监控规则，而不是使用 WLM 超时。例如，您可以将 `max_execution_time` 设置为 50,000 毫秒，如以下 JSON 代码段所示。

```
"max_execution_time": 50000
```

但我们建议您定义一个等效的查询监控规则，此规则将 `query_execution_time` 设置为 50 秒，如下面的 JSON 代码段所示。

```
"rules":
[
    {
        "rule_name": "rule_query_execution",
        "predicate": [
            {
                "metric_name": "query_execution_time",
                "operator": ">",
                "value": 50
            }
        ],
        "action": "abort"
    }
]
```

有关创建或修改查询监控规则的步骤，请参阅《Amazon Redshift 管理指南》中的使用控制台创建或修改查询监控规则和 wlm_json_configuration 参数中的属性。

您可以在以下主题中找到有关查询监控规则的更多信息：

- 预置的 Amazon Redshift 的查询监控指标
- 查询监控规则模板
- 使用控制台创建规则
- 配置工作负载管理
- 查询监控规则的系统表和视图

预置的 Amazon Redshift 的查询监控指标

下表描述了查询监控规则中使用的指标。(这些指标与存储在 STV_QUERY_METRICS 和 STL_QUERY_METRICS 系统表中的指标不同。)

对于某个给定指标，将在查询级别或段级别跟踪性能阈值。有关段和步骤的更多信息，请参阅查询计划和执行工作流程。

> **Note**
>
> WLM 超时 参数与查询监控规则不同。

指标	名称	描述
查询 CPU 时间	query_cpu_time	查询使用的 CPU 时间（以秒为单位）。CPU time 与 Query execution time 不同。 有效值为 0–999,999。
数据块读取	query_blocks_read	查询读取的 1 MB 数据块的数量。 有效值为 0–1,048,575。
扫描行数	scan_row_count	扫描步骤中行的数量。行计数是在筛选标记为删除的行（虚影行）之前和应用用户定义的查询筛选之前发出的行的总数。 有效值为 0–999,999,999,999,999。
查询执行时间	query_execution_time	执行查询所用的时间（以秒为单位）。执行时间不包括在队列中等待的时间。 有效值为 0–86,399。
查询队列时间	query_queue_time	在队列中等待所花的时间（以秒为单位）。 有效值为 0–86,399。
CPU 使用率	query_cpu_usage_percent	查询使用的 CPU 容量的百分比。 有效值为 0–6,399。
内存到磁盘	query_temp_blocks_to_disk	用于写入中间结果的临时磁盘空间（单位为 1 MB 数据块）。 有效值为 0–319,815,679。
CPU 偏斜	cpu_skew	任何切片的最大 CPU 使用率与所有切片的平均 CPU 使用率的比率。此指标在段级别进行定义。 有效值为 0–99。

指标	名称	描述
I/O 偏斜	io_skew	任何切片的最大数据块读取 (I/O) 与所有切片的平均数据块读取的比率。此指标在段级别进行定义。 有效值为 0–99。
联接的行	join_row_count	联接步骤中处理的行数。 有效值为 0–999,999,999,999,999。
嵌套循环联接行数	nested_loop_join_row_count	嵌套循环联接中行的数量。 有效值为 0–999,999,999,999,999。
返回行数	return_row_count	查询返回的行数。 有效值为 0–999,999,999,999,999。
段执行时间	segment_execution_time	执行单个段所用的时间（以秒为单位）。为避免或减少采样错误，请在规则中包括 segment_execution_time > 10 。 有效值为 0–86,388。
Spectrum 扫描行数	spectrum_scan_row_count	Amazon S3 中由 Amazon Redshift Spectrum 查询扫描的数据的行数。 有效值为 0–999,999,999,999,999。
Spectrum 扫描大小	spectrum_scan_size_mb	Amazon S3 中由 Amazon Redshift Spectrum 查询扫描的数据的大小 (MB)。 有效值为 0–999,999,999,999,999。

指标	名称	描述
查询优先级	query_priority	查询的优先级。 有效值为 HIGHEST、HIGH、NORMAL、LOW 和 LOWEST。使用大于 (>) 和小于 (<) 运算符比较 query_priority 时，HIGHEST 大于 HIGH，HIGH 大于 NORMAL，依此类推。

> **Note**
>
> - query_queue_time 谓词不支持跳转操作。也就是说，在满足 query_queue_time 谓词时，将忽略定义了跳转的规则。
> - 较短的段执行时间可能会导致某些指标（例如 io_skew 和 query_cpu_usage_percent）出现采样错误。为避免或减少采样错误，请在规则中包括段执行时间。segment_execution_time > 10 是一个好起点。

SVL_QUERY_METRICS 视图显示已完成查询的指标。SVL_QUERY_METRICS_SUMMARY 视图显示已完成查询的指标的最大值。使用这些视图中的值帮助确定用于定义查询监控规则的阈值。

查询 Amazon Redshift Serverless 的监控指标

下表描述了为 Amazon Redshift Serverless 查询监控规则中使用的指标。

指标	名称	描述
查询 CPU 时间	max_query_cpu_time	查询使用的 CPU 时间（以秒为单位）。CPU time 与 Query execution time 不同。 有效值为 0–999,999。
数据块读取	max_query_blocks_read	查询读取的 1 MB 数据块的数量。 有效值为 0–1,048,575。

指标	名称	描述
扫描行数	max_scan_row_count	扫描步骤中行的数量。行计数是在筛选标记为删除的行（虚影行）之前和应用用户定义的查询筛选之前发出的行的总数。 有效值为 0–999,999,999,999,999。
查询执行时间	max_query_execution_time	执行查询所用的时间（以秒为单位）。执行时间不包括在队列中等待的时间。如果查询超出设置的执行时间，Amazon Redshift Serverless 将停止查询。 有效值为 0–86,399。
查询队列时间	max_query_queue_time	在队列中等待所花的时间（以秒为单位）。 有效值为 0–86,399。
CPU 使用率	max_query_cpu_usage_percent	查询使用的 CPU 容量的百分比。 有效值为 0–6,399。
内存到磁盘	max_query_temp_blocks_to_disk	用于写入中间结果的临时磁盘空间（单位为 1 MB 数据块）。 有效值为 0–319,815,679。
联接的行	max_join_row_count	联接步骤中处理的行数。 有效值为 0–999,999,999,999,999。
嵌套循环联接行数	max_nested_loop_join_row_count	嵌套循环联接中行的数量。 有效值为 0–999,999,999,999,999。

> **Note**
>
> - max_query_queue_time 谓词不支持跳转操作。也就是说，在满足 max_query_queue_time 谓词时，将忽略定义了跳转的规则。
> - 较短的段执行时间可能会导致某些指标（例如 max_io_skew 和 max_query_cpu_usage_percent）出现采样错误。

查询监控规则模板

当您使用 Amazon Redshift 控制台添加规则时，可以选择从预定义的模板中创建规则。Amazon Redshift 将创建具有一组谓词的新规则，并使用默认值填充这些谓词。默认操作是 log。您可以修改这些谓词和操作以满足您的使用案例。

下表列出了可用的模板。

模板名称	Predicates	描述
嵌套循环联接	nested_loop_join_row_count > 100	嵌套循环联接可能表示未完成的联接谓词，这通常会产生一个非常大的返回集（笛卡尔积）。请使用一个较小的行数以及早找到潜在的失控查询。
查询返回了大量的行	return_row_count > 1000000	如果您将队列指定为简单的短时间运行查询，则可包含查找返回较大行数的查询的规则。该模板使用 100 万个行的默认值。对于某些系统，您可能认为 100 万个行过大，或者在较大型系统中，100 万或更多个行可能过大。
与大量的行联接	join_row_count > 1000000000	涉及异常过大行数的联接步骤可能表示需要更严格的筛选条件。该模板使用 10 亿个行的默认值。对于旨在快速简单查询的临时（一次性）队列，您可使用较小的行数。
写入中间结果时占用大量的磁盘空间	query_temp_blocks_	如果当前正在执行的查询使用多个可用的系统 RAM，则查询执行引擎会将中间结果写入磁盘（溢出的内存）。通常，此条件是恶意查询的结

模板名称	Predicates	描述
	`to_disk > 100000`	果，通常也是使用大部分磁盘空间的查询。磁盘使用率的可接受阈值因集群节点类型和节点数而异。该模板使用 100000 个数据块或 100 GB 的默认值。对于小型集群，您可使用较小的数字。
I/O 偏斜高的查询运行时间长	`segment_execution_time > 120`和`io_skew > 1.30`	I/O 偏斜发生在一个节点切片具有的 I/O 比率要比其他切片的比率高得多的时候。作为起点，1.30 的偏斜（平均 1.3 倍）被认为较高。高 I/O 偏斜并不总是一个问题，但在与长时间运行的查询结合时，它可能表示分配方式或排序键存在问题。

查询监控规则的系统表和视图

当满足规则的所有谓词时，WLM 会在 STL_WLM_RULE_ACTION 系统表中写入一行。此行包含规则所触发查询的详细信息以及所导致的操作。

此外，Amazon Redshift 还将查询指标记录到以下系统表和视图中。

- STV_QUERY_METRICS 表显示当前正在运行的查询的指标。
- STL_QUERY_METRICS 表记录已完成的查询的指标。
- SVL_QUERY_METRICS 视图显示已完成查询的指标。
- SVL_QUERY_METRICS_SUMMARY 视图显示已完成查询的指标的最大值。

WLM 系统表和视图

WLM 根据内部定义的 WLM 服务类配置查询队列。Amazon Redshift 根据这些服务类以及在 WLM 配置中定义的队列创建多个内部队列。在系统表中，术语队列 和服务类 通常可互换使用。超级用户队列使用服务类 5。用户定义的队列使用服务类 6 及更高的服务类。

您可以使用特定于 WLM 的系统表查看查询、队列和服务类的状态。查询以下系统表并注意以下事项：

- 查看正在跟踪哪些查询，工作负载管理器分配哪些资源。
- 查看查询被分配到哪个队列。

- 查看工作负载管理器当前跟踪的查询的状态。

表名称	描述
STL_WLM_ERROR	包含 WLM 相关错误事件的日志。
STL_WLM_QUERY	列出 WLM 跟踪的查询。
STV_WLM_CLASSIFICATION_CONFIG	显示 WLM 的当前分类规则。
STV_WLM_QUERY_QUEUE_STATE	记录查询队列的当前状态。
STV_WLM_QUERY_STATE	提供 WLM 跟踪的查询的当前状态快照。
STV_WLM_QUERY_TASK_STATE	包含查询任务的当前状态。
STV_WLM_SERVICE_CLASS_CONFIG	记录 WLM 的服务类配置。
STV_WLM_SERVICE_CLASS_STATE	包含服务类的当前状态。
STL_WLM_RULE_ACTION	记录有关从 WLM 查询监控规则生成的与用户定义的队列关联的操作的详细信息。
STV_WLM_QMR_CONFIG	记录 WLM 查询监控规则 (QMR) 的配置。

您使用任务 ID 来跟踪系统表中的查询。下面的示例介绍如何获取最近提交的用户查询的任务 ID：

```
select task from stl_wlm_query where exec_start_time =(select max(exec_start_time) from
 stl_wlm_query);

task
------
137
(1 row)
```

下面的示例介绍当前正在各个服务类（队列）中执行或等待的查询。该查询在跟踪 Amazon Redshift 的整体并发工作负载时很有用：

```
select * from stv_wlm_query_state order by query;


xid |task|query|service_| wlm_start_  |  state  |queue_ | exec_
    |    |     |class   | time        |         |time   | time
----+----+-----+--------+-------------+---------+-------+--------
2645| 84 | 98  | 3      | 2010-10-... |Returning|   0   | 3438369
2650| 85 | 100 | 3      | 2010-10-... |Waiting  |   0   | 1645879
2660| 87 | 101 | 2      | 2010-10-... |Executing|   0   | 916046
2661| 88 | 102 | 1      | 2010-10-... |Executing|   0   | 13291
(4 rows)
```

WLM 服务类 ID

下表列出了分配到服务类的 ID 的列表。

ID	服务类
1–4	保留供系统使用。
5	由超级用户队列使用。
6–13	由在 WLM 配置中定义的手动 WLM 队列使用。
14	由短查询加速使用。
15	预留供 Amazon Redshift 运行的维护活动使用。
100–107	在 auto_wlm 为 true 时由自动 WLM 队列使用。

管理数据库安全性

主题

您可以通过控制哪些用户可以访问哪些数据库对象来管理数据库安全。

对数据库对象的访问权限取决于您向用户或组授予的权限。以下指南概括了数据库安全的工作方式：

- 默认情况下，仅向对象拥有者授予权限。
- Amazon Redshift 数据库用户是可连接到数据库的指定用户。可通过两种方式向用户授予权限：显式（直接将这些权限分配给账户），或隐式（成为被授予权限的组的成员）。
- 组是可集体分配权限的用户集合，更简化安全维护。
- Schemas 是数据库表和其他数据库对象的集合。架构类似于文件系统目录，但架构不能嵌套。可以向用户授予对单个 schema 或多个 schemas 的访问权限。

此外，Amazon Redshift 还采用了以下功能，让您可以更好地控制哪些用户可以访问哪些数据库对象：

- 通过基于角色的访问控制 (RBAC)，您可以向角色分配权限，然后将角色应用到用户，这样就可以控制大批用户的权限。与组不同，角色可以继承其他角色的权限。

  通过行级别安全性 (RLS)，您可以定义策略来限制对所选行的访问，然后将这些策略应用于用户或组。

动态数据掩蔽 (DDM) 会在查询运行时转换数据，这样您就可以允许用户访问数据而不暴露敏感的详细信息，从而进一步保护您的数据。

有关安全实现的示例，请参阅控制用户和组访问的示例。

有关保护数据的更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 中的安全性。

Amazon Redshift 安全性概览

Amazon Redshift 数据库安全不同于其他类型的 Amazon Redshift 安全。除此部分介绍的数据库安全之外，Amazon Redshift 还提供下面的功能来管理安全：

- 登录凭证 – 对 Amazon Redshift AWS 管理控制台的访问 AWS 受账户权限控制。有关更多信息，请参阅登录凭证。
- 访问管理 — 要控制对特定 Amazon Redshift 资源的访问权限，您可以定义 AWS Identity and Access Management (IAM) 账户。有关更多信息，请参阅控制对 Amazon Redshift 资源的访问。
- 集群安全组 — 要向其他用户授予对 Amazon Redshift 集群的入站访问权限，可定义一个集群安全组并将其与集群相关联。有关更多信息，请参阅 Amazon Redshift 集群安全组。
- VPC — 要通过使用虚拟联网环境保护对集群的访问，可在 Amazon Virtual Private Cloud (VPC) 中启动集群。有关更多信息，请参阅在 Virtual Private Cloud (VPC) 中管理集群。
- 集群加密 – 要对用户创建的所有表中的数据进行加密，可以在启动集群时开启集群加密。有关更多信息，请参阅 Amazon Redshift 集群。
- SSL 连接 — 要对 SQL 客户端与集群之间的连接进行加密，可以使用安全套接字层 (SSL) 加密。有关更多信息，请参阅使用 SSL 连接到集群。
- 加载数据加密 — 要在将表加载数据文件上载到 Amazon S3 时对这些文件进行加密，可以使用服务器端加密或客户端加密。从服务器端加密的数据进行加载时，Amazon S3 将以透明方式处理解密。从客户端加密的数据进行加载时，Amazon Redshift COPY 命令将在加载表时解密数据。有关更多信息，请参阅 将加密的数据上载到 Amazon S3。
- 传输中的数据 – 为保护 AWS 云中的传输中数据，Amazon Redshift 使用硬件加速的 SSL 与 Amazon S3 或 Amazon DynamoDB 通信以执行复制、卸载、备份和还原操作。
- 列级访问控制 – 要在 Amazon Redshift 中对数据进行列级访问控制，请使用列级授予和撤销语句，而无需实施基于视图的访问控制或使用其他系统。

- 行级安全控制 — 要对 Amazon Redshift 中的数据实施行级安全控制，请创建在策略中定义了限制对行的访问权限的策略，并将其附加到角色或用户。

默认数据库用户权限

当您创建数据库对象时，您便是该对象的所有者。默认情况下，只有超级用户或对象的拥有者可以查询、修改或授予该对象上的权限。为使用户能够使用对象，您必须向该用户或包含该用户的组授予必要的权限。数据库超级用户具有与数据库拥有者相同的权限。

Amazon Redshift 支持以下权限：
SELECT、INSERT、UPDATE、DELETE、REFERENCES、CREATE、TEMPORARY 和 USAGE。不同的权限与不同的对象类型关联。有关 Amazon Redshift 支持的数据库对象权限的信息，请参阅 GRANT 命令。

仅所有者有权修改或销毁对象。

默认情况下，所有用户都对数据库的 PUBLIC Schema 具有 CREATE 和 USAGE 权限。要禁止用户在数据库的 PUBLIC 架构中创建对象，请使用 REVOKE 命令删除该权限。

要撤消以前授予的权限，可以使用 REVOKE 命令。对象拥有者的权限（例如 DROP、GRANT 和 REVOKE 权限）是隐式的，无法授予或撤消。对象拥有者可以撤消自己的普通权限，例如，使表对自己以及其他人只读。超级用户将保留所有权限，而不考虑 GRANT 和 REVOKE 命令。

超级用户

数据库超级用户具有与所有数据库的数据库拥有者相同的权限。

您在启动集群时创建的用户 admin 即是超级用户。

您必须是超级用户才能创建超级用户。

Amazon Redshift 系统表和系统视图要么只对超级用户可见，要么对所有用户可见。只有超级用户可以查询指定为“对超级用户可见”的系统表和系统视图。有关信息，请参阅系统表和视图。

超级用户可以查看所有目录表。有关信息，请参阅系统目录表。

数据库超级用户会绕过所有权限检查。超级用户将保留所有权限，而不考虑 GRANT 和 REVOKE 命令。请谨慎使用超级用户角色。建议您使用不是超级用户的角色执行多数工作。您可以创建具有更多限制权限的管理员角色。有关创建角色的更多信息，请参阅基于角色的访问控制 (RBAC)。

要创建新的数据库超级用户，请以超级用户身份登录到数据库，并使用 CREATEUSER 权限发送 CREATE USER 命令或 ALTER USER 命令。

```
CREATE USER adminuser CREATEUSER PASSWORD '1234Admin';
ALTER USER adminuser CREATEUSER;
```

要创建、更改或删除超级用户，请使用相同的命令来管理用户。有关更多信息，请参阅 创建、修改和删除用户。

用户

您可以使用 Amazon Redshift SQL 的 CREATE USER 和 ALTER USER 命令来创建和管理数据库用户。您还可以使用 Amazon Redshift JDBC 或 ODBC 驱动程序配置 SQL 客户端。这些驱动程序在数据库登录过程中管理创建数据库用户和临时密码的过程。

这些驱动程序将基于 AWS Identity and Access Management (IAM) 身份验证来验证数据库用户的身份。如果您已在 AWS 外部管理用户身份，则可以使用符合 SAML 2.0 标准的身份提供者 (IdP) 来管理对 Amazon Redshift 资源的访问。您使用 IAM 角色将 IdP 和 AWS 配置为允许联合身份用户生成临时数据库凭证并登录 Amazon Redshift 数据库。有关更多信息，请参阅使用 IAM 身份验证生成数据库用户凭证。

Amazon Redshift 用户只能由数据库超级用户创建和删除。在用户登录 Amazon Redshift 时会对其进行身份验证。用户可以拥有数据库和数据库对象（例如，表）。他们还可以向用户、组和 Schema 授予对这些对象的权限，以控制谁可以访问哪个对象。具有 CREATE DATABASE 权限的用户可以创建数据库并授予对这些数据库的权限。超级用户具有所有数据库的数据库所有权。

创建、修改和删除用户

数据库用户在数据仓库集群中具有全局性（不针对单个数据库）。

- 要创建用户，请使用 CREATE USER 命令。
- 要创建超级用户，请使用 CREATE USER 命令和 CREATEUSER 选项。
- 要删除现有用户，请使用 DROP USER 命令。
- 要更改用户（例如更改密码），请使用 ALTER USER 命令。
- 要查看用户列表，请查询 PG_USER 目录表。

```
select * from pg_user;
```

```
  usename   | usesysid | usecreatedb | usesuper | usecatupd |  passwd  | valuntil |
 useconfig
------------+----------+-------------+----------+-----------+----------+----------
+-----------
 rdsdb      |        1 | t           | t        | t         | ******** |          |
 masteruser |      100 | t           | t        | f         | ******** |          |
 dwuser     |      101 | f           | f        | f         | ******** |          |
 simpleuser |      102 | f           | f        | f         | ******** |          |
 poweruser  |      103 | f           | t        | f         | ******** |          |
 dbuser     |      104 | t           | f        | f         | ******** |          |
(6 rows)
```

组

组是一些用户的集合，这些用户全部被授予与组关联的所有权限。您可以使用组分配权限。例如，您可以为销售、管理和支持创建不同的组，并向每组中的用户授予对其工作所需数据的适当访问权限。您可以在组级别授予或撤消权限，而这些更改将应用于组的所有成员，但超级用户除外。

要查看所有用户组，请查询 PG_GROUP 系统目录表：

```
select * from pg_group;
```

例如，要按组列出所有数据库用户，请运行以下 SQL。

```
SELECT u.usesysid
,g.groname
,u.usename
FROM pg_user u
LEFT JOIN pg_group g ON u.usesysid = ANY (g.grolist)
```

创建、修改和删除组

只有超级用户才能创建、修改或删除组。

您可以执行以下操作：

- 要创建组，请使用 CREATE GROUP 命令。
- 要在现有组中添加或删除用户，请使用 ALTER GROUP 命令。

- 要删除组，请使用 DROP GROUP 命令。此命令仅删除组，而不删除组成员用户。

控制用户和组访问的示例

此示例创建了用户组和用户，然后向其授予对连接到 Web 应用程序客户端的 Amazon Redshift 数据库的各种权限。此示例假设三组用户：Web 应用程序的常规用户、Web 应用程序的超级用户和 Web 开发人员。

1. 创建将在其中分配用户的组。下面一组命令创建三个不同的用户组：

```
create group webappusers;

create group webpowerusers;

create group webdevusers;
```

2. 创建多个具有不同权限的数据库用户并将其添加到组。
   a. 创建两个用户并将其添加到 WEBAPPUSERS 组：

```
create user webappuser1 password 'webAppuser1pass'
in group webappusers;

create user webappuser2 password 'webAppuser2pass'
in group webappusers;
```

   b. 创建 Web 开发人员用户并将其添加到 WEBDEVUSERS 组：

```
create user webdevuser1 password 'webDevuser2pass'
in group webdevusers;
```

   c. 创建超级用户。此用户将具有管理权限，可以创建其他用户：

```
create user webappadmin  password 'webAppadminpass1'
createuser;
```

3. 创建要与 Web 应用程序所使用的数据库表关联的 schema，并授予各种对此 schema 的用户组访问权限：
   a. 创建 WEBAPP schema：

```
create schema webapp;
```

b. 向 WEBAPPUSERS 组授予 USAGE 权限：

```
grant usage on schema webapp to group webappusers;
```

c. 向 WEBPOWERUSERS 组授予 USAGE 权限：

```
grant usage on schema webapp to group webpowerusers;
```

d. 向 WEBDEVUSERS 组授予 ALL 权限：

```
grant all on schema webapp to group webdevusers;
```

基本用户和组现已设置。您现在可以修改用户和组。

4. 例如，以下命令会修改 WEBAPPUSER1 的 search_path 参数。

```
alter user webappuser1 set search_path to webapp, public;
```

SEARCH_PATH 指定在通过未指定 schema 的简单名称引用数据库对象（例如，表和函数）时，对象的 schema 搜索顺序。

5. 您还可以在创建组之后向组中添加用户，例如，将 WEBAPPUSER2 添加到 WEBPOWERUSERS 组：

```
alter group webpowerusers add user webappuser2;
```

架构

数据库包含一个或多个命名 schemas。数据库中的每个 schema 包含表和其他类型的命名对象。默认情况下，数据库具有单个 schema（名为 PUBLIC）。您可以通过 schemas 使用公用名称分组数据库对象。架构类似于文件系统目录，但架构不能嵌套。

相同的数据库对象名称可以用在同一数据库的不同 schemas 中，没有冲突。例如，MY_SCHEMA 和 YOUR_SCHEMA 都可以包含名为 MYTABLE 的表。具有所需权限的用户可以访问数据库的多个 Schemas 中的对象。

默认情况下，在数据库搜索路径中的第一个 schema 内创建对象。有关信息，请参阅此部分后面的搜索路径。

Schema 可帮助解决多用户环境中的组织和并发问题，方式如下：

- 让多名开发人员使用同一数据库且互不干扰。
- 将数据库对象组织到逻辑组以使其更易于管理。
- 使应用程序能够将其对象放入到单独的 schemas 中，以使其对象名称不与其他应用程序所使用的对象名称冲突。

创建、修改和删除架构

任何用户都可以创建 schema 和修改或删除其拥有的 schema。

您可以执行以下操作：

- 要创建 schema，请使用 CREATE SCHEMA 命令。
- 要更改 schema 所有者，请使用 ALTER SCHEMA 命令。
- 要删除 schema 及其对象，请使用 DROP SCHEMA 命令。
- 要在 schema 内创建表，请以 schema_name.table_name 格式创建表。

要查看所有 Schemas 的列表，请查询 PG_NAMESPACE 系统目录表：

```
select * from pg_namespace;
```

要查看属于某 schema 的表列表，请查询 PG_TABLE_DEF 系统目录表。例如，以下查询会返回 PG_CATALOG schema 中的表列表。

```
select distinct(tablename) from pg_table_def
where schemaname = 'pg_catalog';
```

搜索路径

搜索路径定义在 search_path 参数中，采用逗号分隔的 schema 名称列表形式。搜索路径指定在通过不包含 schemas 限定词的简单名称引用对象（例如，表或函数）时，搜索 schema 的顺序。

如果创建了对象而未指定目标 schema，则将该对象添加到搜索路径中列出的第一个 schema。当不同 schemas 中存在同名的对象时，未指定 schema 的对象名称将引用搜索路径中包含具有该名称的对象的第一个 schema。

要更改当前会话的默认 schema，请使用 SET 命令。

有关更多信息，请参阅“配置引用”中的 search_path 描述。

基于 Schema 的权限

基于 Schema 的权限由 Schema 的拥有者确定：

- 默认情况下，所有用户都对数据库的 PUBLIC Schema 具有 CREATE 和 USAGE 权限。要禁止用户在数据库的 PUBLIC Schema 中创建对象，请使用 REVOKE 命令删除该权限。
- 除非对象拥有者向用户授予了 USAGE 权限，否则用户无法访问其不拥有的 Schemas 中的任何对象。
- 如果已向用户授予对其他用户所创建的 Schema 的 CREATE 权限，则这些用户可以在该 Schema 中创建对象。

基于角色的访问控制 (RBAC)

借助基于角色的访问控制 (RBAC) 在 Amazon Redshift 中管理数据库权限，可以简化 Amazon Redshift 中的安全权限管理。您可以通过总体或精细控制用户可以执行的操作来保护对敏感数据的访问。您还可以控制用户对通常仅限超级用户执行的任务的访问权限。通过将不同的权限分配给不同的角色，然后将角色分配给不同的用户，您可以更精细地控制用户访问权限。

用户获得分配的角色后，将只能执行由所分配角色指定的被授权执行的任务。例如，假设用户所分配的角色具有 CREATE TABLE 和 DROP TABLE 权限，则仅有权执行这些任务。您可以通过向不同用户授予不同级别的安全权限来访问其工作所需的数据，从而控制用户访问权限。

RBAC 根据用户的角色要求对用户执行最低权限原则，而不论所涉及的是何种对象类型。授予和撤消权限在角色级别执行，无需更新单个数据库对象的权限。

使用 RBAC，您可以创建具有相应权限的角色，以运行通常需要超级用户权限的命令。只要用户通过包含这些权限的角色获得授权，即可以运行这些命令。同样，您还可以创建角色来限制对某些命令的访问权限，并将角色分配给超级用户或已获得该角色授权的用户。

要了解 Amazon Redshift RBAC 的工作原理，请观看以下视频：在 Amazon Redshift 中引入基于角色的访问控制 (RBAC)。

角色层次结构

角色是可以分配给用户或其他角色的权限集合。您可以为角色分配系统或数据库权限。用户从所分配的角色继承权限。

在 RBAC 中，用户可以拥有嵌套角色。您可以向用户和角色授予角色。将一个角色授予某个用户时，您将向该用户授予此角色包括的所有权限。将角色 r1 授予某个用户时，您向该用户授予了来自 r1 的权限。用户现在拥有来自 r1 的权限以及他们已经拥有的任何现有权限。

将角色 (r1) 授予另一个角色 (r2) 时，您向 r2 授予了来自 r1 的权限。此外，在将 r2 授予另一个角色 (r3) 时，r3 的权限是来自 r1 和 r2 的权限的组合。由于角色层次结构的原因，r2 继承了 r1 的权限。Amazon Redshift 通过执行每个角色授权来传播权限。通过将 r1 授予 r2，然后将 r2 授予 r3，从而将这三个角色的所有权限授予了 r3。因此，将 r3 授予某个用户后，该用户将拥有这三个角色的所有权限。

Amazon Redshift 不允许创建角色循环授权。如果将嵌套角色分配回角色层次结构中的上级角色，将会出现角色循环授权，例如将 r3 分配回 r1。有关如何创建角色以及管理角色分配的更多信息，请参阅管理 RBAC 中的角色。

角色分配

超级用户和拥有 CREATE ROLE 权限的普通用户都可以使用 CREATE ROLE 语句创建角色。超级用户和角色管理员可以使用 GRANT ROLE 语句向其他人授予角色。他们可以使用 REVOKE ROLE 语句撤消授予其他人的角色，也可以使用 DROP ROLE 语句来删除角色。管理员角色包括所有者角色和已授予 ADMIN OPTION 权限角色的用户角色。

只有超级用户或角色管理员才能授予和撤销角色。您可以向一个或多个角色或用户授予或撤消一个或多个角色。使用 GRANT ROLE 语句和 WITH ADMIN OPTION 选项，可以向所有被授予者提供所有授予的角色的管理选项。

Amazon Redshift 支持不同的角色分配组合，例如授予多个角色或有多个被授予者。WITH ADMIN OPTION 选项仅适用于用户，而不适用于角色。同样，在 REVOKE ROLE 语句中使用 WITH ADMIN OPTION 选项将会删除相关角色并撤销被授予者的管理授权。使用 ADMIN OPTION 选项时，将仅撤销该角色的管理授权。

下面的示例将撤销授予 `user2` 的 `sample_role2` 角色的管理授权。

```
REVOKE ADMIN OPTION FOR sample_role2 FROM user2;
```

有关如何创建角色以及管理角色分配的更多信息，请参阅管理 RBAC 中的角色。

Amazon Redshift 系统定义的角色

Amazon Redshift 提供了使用特定权限定义的一些系统定义角色。系统特定角色以 sys: 前缀开头。仅具有适当访问权限的用户才能更改系统定义角色或创建自定义的系统定义角色。您将 sys: 前缀用于自定义的系统定义角色。

下表总结了各种角色及其权限。

角色名称	描述		
sys:monitor	此角色拥有访问目录或系统表的权限。		
sys:operator	此角色拥有访问目录或系统表，以及分析、Vacuum 或取消查询的权限。		
sys:dba	此角色拥有创建 Schema、创建表、删除 Schema、删除表和截断表的权限。它拥有创建或替换存储的过程、删除过程、创建或替换函数、创建或替换外部函数、创建视图和删除视图的权限。此外，此角色还继承了 sys:operator 角色的所有权限。		
sys:superuser	此角色具有 RBAC 的系统权限 中定义的所有支持的系统权限。		
sys:secadmin	• 此角色拥有创建用户、更改用户、删除用户、创建角色、删除角色和授予角色的权限。 • 此角色拥有对关系开启或关闭 RLS 的权限以及管理 RLS 和 DDM 策略的权限（CREATE、DROP、ATTACH、DETACH 和 ALTER）。另请注意，默认情况下，将向此角色授予		

角色名称	描述		
	EXPLAIN RLS、IGNORE RLS 和 EXPLAIN MASKING 权限。 • 只有在向角色显式授予权限时，此角色才能访问用户表。		

用于数据共享的系统定义角色和用户

Amazon Redshift 创建的角色和用户供内部使用，这些角色和用户对应于数据共享和数据共享使用者。每个内部角色名称和用户名都有保留的命名空间前缀 ds:。它们具有以下格式：

名称	描述		
ds:*sharei*	与数据共享对应的系统角色。		
ds:*sharei* *_consume:*	与数据共享使用者对应的系统用户。		

为每个数据共享创建一个数据共享角色。它拥有当前授予数据共享的所有权限。将为数据共享的每个使用者创建一个数据共享用户。它被授予对单个数据共享角色的权限。添加到多个数据共享的使用者将为每个数据共享创建一个数据共享用户。

这些用户和角色是数据共享正常运行所必需的。不能修改或删除它们，也不能访问它们或将它们用于客户运行的任何任务。您可以放心地忽略它们。有关数据共享的更多信息，请参阅[在 Amazon Redshift 中跨集群共享数据](#)。

> **Note**
>
> 您不能使用 ds: 前缀来创建用户定义的角色或用户。

RBAC 的系统权限

以下是可向角色授予或撤消的系统权限列表。

命令	您必须通过以下方式之一获得运行该命令的权限		
CREATE ROLE	• 超级用户。 • 具有 CREATE ROLE 权限的用户。		
DROP ROLE	• 超级用户。 • 角色拥有者是创建角色的用户，或者已被授予具有 WITH ADMIN OPTION 权限的角色的用户。		
CREATE USER	• 超级用户。 • 具有 CREATE USER 权限的用户。这些用户不能创建超级用户。		
DROP USER	• 超级用户。 • 具有 DROP USER 权限的用户。		
ALTER USER	• 超级用户。 • 具有 ALTER USER 权限的用户。这些用户不能将用户更改为超级用户，也不能将超级用户更改为用户。 • 想更改自己密码的当前用户。		
CREATE SCHEMA	• 超级用户。 • 具有 CREATE SCHEMA 权限的用户。		
DROP SCHEMA	• 超级用户。 • 具有 DROP SCHEMA 权限的用户。 • Schema 拥有者。		
ALTER DEFAULT PRIVILEGES	• 超级用户。 • 具有 ALTER DEFAULT PRIVILEGES 权限的用户。 • 更改自己的默认访问权限的用户。 • 为其拥有访问权限的 Schema 设置权限的用户。		
CREATE TABLE	• 超级用户。 • 具有 CREATE TABLE 权限的用户。		

命令	您必须通过以下方式之一获得运行该命令的权限		
	• 具有 Schema 的 CREATE 权限的用户。		
DROP TABLE	• 超级用户。 • 具有 DROP TABLE 权限的用户。 • 对模式拥有 USAGE 权限的表拥有者。		
ALTER TABLE	• 超级用户。 • 具有 ALTER TABLE 权限的用户。 • 对模式拥有 USAGE 权限的表拥有者。		

CREATE OR REPLAC FUNCTIC	• CREATE FUNCTION：
	  • 超级用户。
	  • 具有 CREATE OR REPLACE FUNCTION 权限的用户。
	  • 具有语言 USAGE 权限的用户。
	• REPLACE FUNCTION：
	  • 超级用户。
	  • 具有 CREATE OR REPLACE FUNCTION 权限的用户。
	  • 函数拥有者。		

CREATE OR REPLAC EXTERN FUNCTIC	• 超级用户。 • 具有 CREATE OR REPLACE EXTERNAL FUNCTION 权限的用户。		
DROP FUNCTIC	• 超级用户。 • 具有 DROP FUNCTION 权限的用户。 • 函数拥有者。		

命令	您必须通过以下方式之一获得运行该命令的权限		

CREATE OR REPLAC PROCED	• CREATE PROCEDURE：
	  • 超级用户。
	  • 具有 CREATE OR REPLACE PROCEDURE 权限的用户。
	  • 具有语言 USAGE 权限的用户。
	• REPLACE PROCEDURE：
	  • 超级用户。
	  • 具有 CREATE OR REPLACE PROCEDURE 权限的用户。
	  • 过程拥有者。		

DROP PROCED	• 超级用户。 • 具有 DROP PROCEDURE 权限的用户。 • 过程拥有者。		

CREATE OR REPLAC VIEW	• CREATE VIEW：
	  • 超级用户。
	  • 具有 CREATE OR REPLACE VIEW 权限的用户。
	  • 具有 Schema 的 CREATE 权限的用户。
	• REPLACE VIEW：
	  • 超级用户。
	  • 具有 CREATE OR REPLACE VIEW 权限的用户。
	  • 视图拥有者。		

DROP VIEW	• 超级用户。 • 具有 DROP VIEW 权限的用户。 • 视图拥有者。		
CREATE MODEL	• 超级用户。 • 具有 CREATE MODEL 系统权限的用户，应能够读取 CREATE MODEL 的关系。 • 具有 CREATE MODEL 权限的用户。		

命令	您必须通过以下方式之一获得运行该命令的权限
DROP MODEL	• 超级用户。 • 具有 DROP MODEL 权限的用户。 • 模型拥有者。 • Schema 拥有者。
CREATE DATASH	• 超级用户。 • 具有 CREATE DATASHARE 权限的用户。 • 数据库拥有者。
ALTER DATASH	• 超级用户。 • 具有 ALTER DATASHARE 权限的用户。 • 具有数据共享的 ALTER 或 ALL 权限的用户。 • 要将特定对象添加到数据共享中，这些用户必须具有对象的权限。用户应是对象的拥有者，或者具有这些对象的 SELECT、USAGE 或 ALL 权限。
DROP DATASH	• 超级用户。 • 具有 DROP DATASHARE 权限的用户。 • 数据库拥有者。
CREATE LIBRARY	• 超级用户。 • 具有 CREATE LIBRARY 权限或具有指定语言权限的用户。
DROP LIBRARY	• 超级用户。 • 具有 DROP LIBRARY 权限的用户。 • 库拥有者。
ANALYZE	• 超级用户。 • 具有 ANALYZE 权限的用户。 • 关系的拥有者。 • 向其共享了表的数据库拥有者。

命令	您必须通过以下方式之一获得运行该命令的权限		
CANCEL	• 取消自己的查询的超级用户。 • 取消用户的查询的超级用户。 • 取消用户的查询并且具有 CANCEL 权限的用户。 • 取消自己的查询的用户。		
TRUNCATE TABLE	• 超级用户。 • 具有 TRUNCATE TABLE 权限的用户。 • 表拥有者。		
VACUUM	• 超级用户。 • 具有 VACUUM 权限的用户。 • 表拥有者。 • 向其共享了表的数据库拥有者。		
IGNORE RLS	• 超级用户。 • `sys:secadmin` 角色中的用户。		
EXPLAIN RLS	• 超级用户。 • `sys:secadmin` 角色中的用户。		
EXPLAIN MASKING	• 超级用户。 • `sys:secadmin` 角色中的用户。		

数据库对象权限

除了系统权限外，Amazon Redshift 还包括定义访问选项的数据库对象权限。这些权限包括各种选项，例如读取表和视图中的数据、写入数据、创建表和删除表等。有关更多信息，请参阅 GRANT 命令。

与系统权限的使用类似，您可以通过使用 RBAC 将数据库对象权限分配给角色。然后，您可以为用户分配角色、向用户授予系统权限以及向用户授予数据库权限。

适用于 RBAC 的 ALTER DEFAULT PRIVILEGES

使用 ALTER DEFAULT PRIVILEGES 来定义默认访问权限集，这些权限将应用于指定的用户未来创建的对象。默认情况下，用户只能更改他们自己的原定设置访问权限。使用 RBAC，您可以设置角色的默认访问权限。有关更多信息，请参阅 ALTER DEFAULT PRIVILEGES 命令。

与系统权限类似，您可以通过 RBAC 将数据库对象权限分配给角色。然后，您可以为用户分配角色、向用户授予系统权限和/或数据库权限。

RBAC 中的角色使用注意事项

在使用 RBAC 角色时，请注意以下事项：

- Amazon Redshift 不允许角色循环授权。您不能将 r1 授予 r2，然后将 r2 授予 r1。
- RBAC 支持原生 Amazon Redshift 对象和 Amazon Redshift Spectrum 表。
- Amazon Redshift 管理员可以通过将集群升级到最新的维护补丁，从而开始使用 RBAC。
- 只有超级用户和具有 CREATE ROLE 系统权限的用户才能创建角色。
- 只有超级用户和角色管理员才能修改或删除角色。
- 角色名称不能与用户名称相同。
- 角色名称不能包含无效字符，例如“:/\n”。
- 角色名称不能是保留字，例如 PUBLIC。
- 角色名称不能以默认角色的保留前缀 `sys:` 开头。
- 将具有 RESTRICT 参数的角色授予其他角色时，不能删除该角色。原定设置为 RESTRICT。当您尝试删除继承了其他角色的角色时，Amazon Redshift 会返回错误。
- 无角色的管理员权限的用户不能授予或撤销角色。

管理 RBAC 中的角色

要执行以下操作，请使用以下命令：

- 要创建角色，请使用 CREATE ROLE 命令。
- 要重命名角色或更改角色的拥有者，请使用 ALTER ROLE 命令。
- 要删除角色，请使用 DROP ROLE 命令。
- 要向用户授予角色，请使用 GRANT 命令。

- 要撤消用户的角色，请使用 REVOKE 命令。
- 要向角色授予系统权限，请使用 GRANT 命令。
- 要撤销角色的系统权限，请使用 REVOKE 命令。

要查看集群或工作组中的角色列表，请参阅SVV_ROLES。

教程：使用 RBAC 创建角色和进行查询

使用 RBAC，您可以创建具有相应权限的角色，以运行通常需要超级用户权限的命令。只要用户通过包含这些权限的角色获得授权，即可以运行这些命令。

在本教程中，您将使用基于角色的访问控制（RBAC）来管理所创建数据库中的权限。然后，您将连接到数据库并从两个不同的角色查询数据库，以测试 RBAC 的功能。

您创建并用于查询数据库的两个角色是 `sales_ro` 和 `sales_rw`。您将创建 `sales_ro` 角色并以具有 `sales_ro` 角色的用户身份查询数据。`sales_ro` 用户只能使用 SELECT 命令，而不能使用 UPDATE 命令。然后，您将创建 `sales_rw` 角色并以具有 `sales_rw` 角色的用户身份查询数据。`sales_rw` 用户可以使用 SELECT 命令和 UPDATE 命令。

此外，您还可以创建角色来限制对某些命令的访问权限，并将角色分配给超级用户或普通用户。

任务

- 先决条件
- 步骤 1：创建管理员用户
- 步骤 2：设置架构
- 步骤 3：创建只读用户
- 步骤 4：以只读用户身份查询数据
- 步骤 5：创建读写用户
- 步骤 6：以继承了只读角色的用户身份查询数据
- 步骤 7：向读写角色授予更新和插入权限
- 步骤 8：以读写用户身份查询数据
- 步骤 9：以管理员用户身份分析和清理数据库中的表
- 步骤 10：以读写用户身份截断表

先决条件

- 创建装有 TICKIT 示例数据库的 Amazon Redshift 集群或 Serverless 工作组。要创建 Serverless 工作组，请参阅 Amazon Redshift Serverless。要创建集群，请参阅创建示例 Amazon Redshift 集群。有关 TICKIT 示例数据库的更多信息，请参阅示例数据库。
- 可以访问具有超级用户或角色管理员权限的用户。只有超级用户或角色管理员才能授予或撤销角色。有关 RBAC 所需权限的更多信息，请参阅RBAC 的系统权限。
- 查看RBAC 中的角色使用注意事项。

步骤 1：创建管理员用户

要为本教程进行设置，请在此步骤中创建数据库管理员角色并将其附加到数据库管理员用户。必须将数据库管理员创建为超级用户或角色管理员。

在 Amazon Redshift https://docs.aws.amazon.com/redshift/latest/mgmt/query-editor-v2-using.html 中运行所有查询。

1. 要创建管理员角色 db_admin，请使用以下示例。

```
CREATE ROLE db_admin;
```

2. 要创建名为 dbadmin 的数据库用户，请使用以下示例。

```
CREATE USER dbadmin PASSWORD 'Test12345';
```

3. 要将名为 sys:dba 的系统定义角色授予 db_admin 角色，请使用以下示例。当被授予 sys:dba 角色后，dbadmin 用户就可以创建架构和表。有关更多信息，请参阅 Amazon Redshift 系统定义的角色。

步骤 2：设置架构

在此步骤中，您将以数据库管理员的身份连接到您的数据库。然后，您将创建两个架构并向它们添加数据。

1. 使用查询编辑器 v2，以 dbadmin 用户身份连接到 dev 数据库。有关连接到数据库的更多信息，请参阅使用查询编辑器 v2。
2. 要创建销售和营销数据库架构，请使用以下示例。

```
CREATE SCHEMA sales;
CREATE SCHEMA marketing;
```

3. 要在销售架构的表中创建和插入值，请使用以下示例。

```
CREATE TABLE sales.cat(
catid smallint,
catgroup varchar(10),
catname varchar(10),
catdesc varchar(50)
);
INSERT INTO sales.cat(SELECT * FROM category);

CREATE TABLE sales.dates(
dateid smallint,
caldate date,
day char(3),
week smallint,
month char(5),
qtr char(5),
year smallint,
holiday boolean
);
INSERT INTO sales.dates(SELECT * FROM date);

CREATE TABLE sales.events(
eventid integer,
venueid smallint,
catid smallint,
dateid smallint,
eventname varchar(200),
starttime timestamp
);
INSERT INTO sales.events(SELECT * FROM event);

 CREATE TABLE sales.sale(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
```

```
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp
);
INSERT INTO sales.sale(SELECT * FROM sales);
```

4. 要在营销架构的表中创建和插入值，请使用以下示例。

```
CREATE TABLE marketing.cat(
catid smallint,
catgroup varchar(10),
catname varchar(10),
catdesc varchar(50)
);
INSERT INTO marketing.cat(SELECT * FROM category);

CREATE TABLE marketing.dates(
dateid smallint,
caldate date,
day char(3),
week smallint,
month char(5),
qtr char(5),
year smallint,
holiday boolean
);
INSERT INTO marketing.dates(SELECT * FROM date);

CREATE TABLE marketing.events(
eventid integer,
venueid smallint,
catid smallint,
dateid smallint,
eventname varchar(200),
starttime timestamp
);
INSERT INTO marketing.events(SELECT * FROM event);

CREATE TABLE marketing.sale(
marketingid integer,
listid integer,
sellerid integer,
buyerid integer,
```

```
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp
);
INSERT INTO marketing.sale(SELECT * FROM marketing);
```

步骤 3：创建只读用户

在此步骤中，您将创建一个只读角色，并为该只读角色创建一个 salesanalyst 用户。销售分析师只需对销售架构中的表进行只读访问，即可完成分配给他们的任务，即查找产生最大佣金的事件。

1. 以 dbadmin 用户身份连接到数据库。
2. 要创建 sales_ro 角色，请使用以下示例。

```
CREATE ROLE sales_ro;
```

3. 要创建 salesanalyst 用户，请使用以下示例。

```
CREATE USER salesanalyst PASSWORD 'Test12345';
```

4. 要授予 sales_ro 角色使用权限并选择对销售架构对象的访问权限，请使用以下示例。

```
GRANT USAGE ON SCHEMA sales TO ROLE sales_ro;
GRANT SELECT ON ALL TABLES IN SCHEMA sales TO ROLE sales_ro;
```

5. 要向 salesanalyst 用户授予 sales_ro 角色，请使用以下示例。

```
GRANT ROLE sales_ro TO salesanalyst;
```

步骤 4：以只读用户身份查询数据

在此步骤中，salesanalyst 用户从销售架构中查询数据。然后，salesanalyst 用户尝试更新一个表并读取营销架构中的表。

1. 以 salesanalyst 用户身份连接到数据库。
2. 要查找佣金最高的 10 笔销售，请使用以下示例。

```
SET SEARCH_PATH TO sales;
SELECT DISTINCT events.dateid, sale.commission, cat.catname
FROM sale, events, dates, cat
WHERE events.dateid=dates.dateid AND events.dateid=sale.dateid AND events.catid =
 cat.catid
ORDER BY 2 DESC LIMIT 10;

+--------+------------+----------+
| dateid | commission | catname  |
+--------+------------+----------+
|   1880 |     1893.6 | Pop      |
|   1880 |     1893.6 | Opera    |
|   1880 |     1893.6 | Plays    |
|   1880 |     1893.6 | Musicals |
|   1861 |       1500 | Plays    |
|   2003 |       1500 | Pop      |
|   1861 |       1500 | Opera    |
|   2003 |       1500 | Plays    |
|   1861 |       1500 | Musicals |
|   1861 |       1500 | Pop      |
+--------+------------+----------+
```

3. 要从销售架构的事件表中选择 10 个事件，请使用以下示例。

```
SELECT * FROM sales.events LIMIT 10;

+---------+---------+-------+--------+--------------------+---------------------+
| eventid | venueid | catid | dateid |     eventname      |      starttime      |
+---------+---------+-------+--------+--------------------+---------------------+
|    4836 |      73 |     9 |   1871 | Soulfest           | 2008-02-14 19:30:00 |
|    5739 |      41 |     9 |   1871 | Fab Faux           | 2008-02-14 19:30:00 |
|     627 |     229 |     6 |   1872 | High Society       | 2008-02-15 14:00:00 |
|    2563 |     246 |     7 |   1872 | Hamlet             | 2008-02-15 20:00:00 |
|    7703 |      78 |     9 |   1872 | Feist              | 2008-02-15 14:00:00 |
|    7903 |      90 |     9 |   1872 | Little Big Town    | 2008-02-15 19:30:00 |
|    7925 |     101 |     9 |   1872 | Spoon              | 2008-02-15 19:00:00 |
|    8113 |      17 |     9 |   1872 | Santana            | 2008-02-15 15:00:00 |
|     463 |     303 |     8 |   1873 | Tristan und Isolde | 2008-02-16 19:00:00 |
|     613 |     236 |     6 |   1873 | Pal Joey           | 2008-02-16 15:00:00 |
+---------+---------+-------+--------+--------------------+---------------------+
```

4. 要尝试更新 eventid 1 的事件名称，请运行以下示例。此示例将导致权限被拒绝错误，因为 salesanalyst 用户仅对销售架构中的事件表具有 SELECT 权限。要更新事件表，必须向 sales_ro 角色授予 UPDATE 权限。有关如何授予权限以更新表的更多信息，请参阅 GRANT 的 UPDATE 参数。有关 UPDATE 命令的更多信息，请参阅UPDATE。

```
UPDATE sales.events
SET eventname = 'Comment event'
WHERE eventid = 1;

ERROR: permission denied for relation events
```

5. 要尝试从营销架构的事件表中选择所有事件，请使用以下示例。此示例将导致权限被拒绝错误，因为 salesanalyst 用户仅对销售架构中的事件表具有 SELECT 权限。要从营销架构的事件表中选择数据，必须授予 sales_ro 角色对营销架构中事件表的 SELECT 权限。

```
SELECT * FROM marketing.events;

               ERROR: permission denied for schema marketing
```

步骤 5：创建读写用户

在此步骤中，负责为销售架构中的数据处理构建提取、转换、加载（ETL）管道的销售工程师将获得只读访问权限，但稍后将获得执行任务的读写权限。

1. 以 dbadmin 用户身份连接到数据库。
2. 要在销售架构中创建 sales_rw 角色，请使用以下示例。

```
CREATE ROLE sales_rw;
```

3. 要创建 salesengineer 用户，请使用以下示例。

```
CREATE USER salesengineer PASSWORD 'Test12345';
```

4. 要授予 sales_rw 角色使用权限，并通过向其分配 sales_ro 角色来选择对销售架构对象的访问权限，请使用以下示例。有关 Amazon Redshift 中角色如何继承权限的更多信息，请参阅角色层次结构。

```
GRANT ROLE sales_ro TO ROLE sales_rw;
```

5. 要将 sales_rw 角色分配给 salesengineer 用户，请使用以下示例。

```
GRANT ROLE sales_rw TO salesengineer;
```

步骤 6：以继承了只读角色的用户身份查询数据

在此步骤中，salesengineer 用户在被授予读取权限之前尝试更新事件表。

1. 以 salesengineer 用户身份连接到数据库。
2. salesengineer 用户可以从销售架构的事件表中成功读取数据。要从销售架构的事件表中选择 eventid 为 1 的事件，请使用以下示例。

```
SELECT * FROM sales.events where eventid=1;

+---------+---------+-------+--------+-----------------+---------------------+
| eventid | venueid | catid | dateid |    eventname    |      starttime      |
+---------+---------+-------+--------+-----------------+---------------------+
|       1 |     305 |     8 |   1851 | Gotterdammerung | 2008-01-25 14:30:00 |
+---------+---------+-------+--------+-----------------+---------------------+
```

3. 要尝试从营销架构的事件表中选择所有事件，请使用以下示例。salesengineer 用户对营销架构中的表没有权限，因此，该查询将导致权限被拒绝错误。要从营销架构的事件表中选择数据，必须授予 sales_rw 角色对营销架构中事件表的 SELECT 权限。

```
SELECT * FROM marketing.events;

ERROR: permission denied for schema marketing
```

4. 要尝试更新 eventid 1 的事件名称，请运行以下示例。此示例将导致权限被拒绝错误，因为 salesengineer 用户仅对销售架构中的事件表具有 SELECT 权限。要更新事件表，必须向 sales_rw 角色授予 UPDATE 权限。

```
UPDATE sales.events
SET eventname = 'Comment event'
WHERE eventid = 1;

ERROR: permission denied for relation events
```

步骤 7：向读写角色授予更新和插入权限

在此步骤中，您向 sales_rw 角色授予 UPDATE 和 INSERT 权限。

1. 以 dbadmin 用户身份连接到数据库。
2. 要向 sales_rw 角色授予 UPDATE、INSERT 和 DELETE 权限，请使用以下示例。

```
GRANT UPDATE, INSERT, ON ALL TABLES IN SCHEMA sales TO role sales_rw;
```

步骤 8：以读写用户身份查询数据

在此步骤中，salesengineer 在其角色被授予 INSERT 和 UPDATE 权限后成功更新表。接下来，salesengineer 尝试分析和清理事件表，但未能成功。

1. 以 salesengineer 用户身份连接到数据库。
2. 要更新 eventid 1 的事件名称，请运行以下示例。

```
UPDATE sales.events
SET eventname = 'Comment event'
WHERE eventid = 1;
```

3. 要查看在上一个查询中进行的更改，请使用以下示例从销售架构的事件表中选择 eventid 为 1 的事件。

```
SELECT * FROM sales.events WHERE eventid=1;

+---------+---------+-------+--------+---------------+---------------------+
| eventid | venueid | catid | dateid |   eventname   |      starttime      |
+---------+---------+-------+--------+---------------+---------------------+
|       1 |     305 |     8 |   1851 | Comment event | 2008-01-25 14:30:00 |
+---------+---------+-------+--------+---------------+---------------------+
```

4. 要分析销售架构中更新的事件表，请使用以下示例。此示例将导致权限被拒绝错误，因为 salesengineer 用户没有必要的权限，也不是销售架构中事件表的所有者。要分析事件表，必须使用 GRANT 命令向 sales_rw 角色授予 ANALYZE 权限。有关 ANALYZE 命令的更多信息，请参阅 ANALYZE。

```
ANALYZE sales.events;
```

```
ERROR: skipping "events" --- only table or database owner can analyze
```

5. 要清理更新的事件表，请使用以下示例。此示例将导致权限被拒绝错误，因为 salesengineer 用户没有必要的权限，也不是销售架构中事件表的所有者。要清理事件表，必须使用 GRANT 命令向 sales_rw 角色授予 VACUUM 权限。有关 VACUUM 命令的更多信息，请参阅 VACUUM。

```
VACUUM sales.events;

ERROR: skipping "events" --- only table or database owner can vacuum it
```

步骤 9：以管理员用户身份分析和清理数据库中的表

在此步骤中，dbadmin 用户分析并清理所有表。用户对此数据库具有管理员权限，因此他们能够运行这些命令。

1. 以 dbadmin 用户身份连接到数据库。
2. 要分析销售架构中的事件表，请使用以下示例。

```
ANALYZE sales.events;
```

3. 要清理销售架构中的事件表，请使用以下示例。

```
VACUUM sales.events;
```

4. 要分析营销架构中的事件表，请使用以下示例。

```
ANALYZE marketing.events;
```

5. 要清理营销架构中的事件表，请使用以下示例。

```
VACUUM marketing.events;
```

步骤 10：以读写用户身份截断表

在此步骤中，salesengineer 用户尝试截断销售架构中的事件表，但只有在 dbadmin 用户授予 TRUNCATE 权限时才会成功。

1. 以 salesengineer 用户身份连接到数据库。

2. 要尝试删除销售架构中事件表的所有行，请使用以下示例。此示例将导致错误，因为 salesengineer 用户没有必要的权限，也不是销售架构中事件表的所有者。要截断事件表，必须使用 GRANT 命令向 sales_rw 角色授予 TRUNCATE 权限。有关 TRUNCATE 命令的更多信息，请参阅 TRUNCATE。

```
TRUNCATE sales.events;

ERROR: must be owner of relation events
```

3. 以 dbadmin 用户身份连接到数据库。
4. 要向 sales_rw 角色授予截断表权限，请使用以下示例。

```
GRANT TRUNCATE TABLE TO role sales_rw;
```

5. 使用查询编辑器 v2，以 salesengineer 用户身份连接到数据库。
6. 要读取销售架构事件表中的前 10 个事件，请使用以下示例。

```
SELECT * FROM sales.events ORDER BY eventid LIMIT 10;

+---------+---------+-------+--------+-----------------------------
+---------------------+
| eventid | venueid | catid | dateid |          eventname          |      starttime
    |
+---------+---------+-------+--------+-----------------------------
+---------------------+
|       1 |     305 |     8 |   1851 | Comment event               | 2008-01-25
 14:30:00 |
|       2 |     306 |     8 |   2114 | Boris Godunov               | 2008-10-15
 20:00:00 |
|       3 |     302 |     8 |   1935 | Salome                      | 2008-04-19
 14:30:00 |
|       4 |     309 |     8 |   2090 | La Cenerentola (Cinderella) | 2008-09-21
 14:30:00 |
|       5 |     302 |     8 |   1982 | Il Trovatore                | 2008-06-05
 19:00:00 |
|       6 |     308 |     8 |   2109 | L Elisir d Amore            | 2008-10-10
 19:30:00 |
|       7 |     309 |     8 |   1891 | Doctor Atomic               | 2008-03-06
 14:00:00 |
|       8 |     302 |     8 |   1832 | The Magic Flute             | 2008-01-06
 20:00:00 |
```

```
|       9 |     308 |     8 |   2087 | The Fly                       | 2008-09-18
 19:30:00 |
|      10 |     305 |     8 |   2079 | Rigoletto                     | 2008-09-10
 15:00:00 |
+---------+---------+-------+--------+-------------------------------
+---------------------+
```

7. 要截断销售架构中的事件表，请使用以下示例。

```
TRUNCATE sales.events;
```

8. 要读取销售架构中更新的事件表中的数据，请使用以下示例。

```
SELECT * FROM sales.events ORDER BY eventid LIMIT 10;

+---------+---------+-------+--------+-------------------------------
+---------------------+
| eventid | venueid | catid | dateid |          eventname          |      starttime
    |
+---------+---------+-------+--------+-------------------------------
+---------------------+
```

为营销架构创建只读和读写角色（可选）

在此步骤中，您将为营销架构创建只读和读写角色。

1. 以 dbadmin 用户身份连接到数据库。
2. 要为营销架构创建只读和读写角色，请使用以下示例。

```
CREATE ROLE marketing_ro;

CREATE ROLE marketing_rw;

GRANT USAGE ON SCHEMA marketing TO ROLE marketing_ro, ROLE marketing_rw;

GRANT SELECT ON ALL TABLES IN SCHEMA marketing TO ROLE marketing_ro;

GRANT ROLE marketing_ro TO ROLE marketing_rw;

GRANT INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA marketing TO ROLE marketing_rw;

CREATE USER marketinganalyst PASSWORD 'Test12345';
```

```
CREATE USER marketingengineer PASSWORD 'Test12345';

GRANT ROLE marketing_ro TO marketinganalyst;

GRANT ROLE marketing_rw TO marketingengineer;
```

RBAC 的系统功能（可选）

Amazon Redshift 有两个函数可提供有关用户成员资格和其他组或角色的角色成员资格的系统信息：role_is_member_of 和 user_is_member_of。这些函数可供超级用户和普通用户使用。超级用户可以查看所有角色成员资格。普通用户只能查看已被授予访问权限的角色的成员资格。

使用 role_is_member_of 函数

1. 以 salesengineer 用户身份连接到数据库。
2. 要查看 sales_rw 角色是否是 sales_ro 角色的成员，请使用以下示例。

```
SELECT role_is_member_of('sales_rw', 'sales_ro');

+-------------------+
| role_is_member_of |
+-------------------+
| true              |
+-------------------+
```

3. 要查看 sales_ro 角色是否是 sales_rw 角色的成员，请使用以下示例。

```
SELECT role_is_member_of('sales_ro', 'sales_rw');

+-------------------+
| role_is_member_of |
+-------------------+
| false             |
+-------------------+
```

使用 user_is_member_of 函数

1. 以 salesengineer 用户身份连接到数据库。

2. 以下示例尝试查看 salesanalyst 用户的用户成员资格。此查询会导致错误，因为 salesengineer 无权访问 salesanalyst。要成功运行此命令，请以 salesanalyst 用户身份连接到数据库并使用示例。

```
SELECT user_is_member_of('salesanalyst', 'sales_ro');

ERROR
```

3. 以超级用户身份连接到数据库。
4. 以超级用户身份连接后，要查看 salesanalyst 用户的成员资格，请使用以下示例。

```
SELECT user_is_member_of('salesanalyst', 'sales_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| true              |
+-------------------+
```

5. 以 dbadmin 用户身份连接到数据库。
6. 要查看 salesengineer 用户的成员资格，请使用以下示例。

```
SELECT user_is_member_of('salesengineer', 'sales_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| true              |
+-------------------+

SELECT user_is_member_of('salesengineer', 'marketing_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| false             |
+-------------------+

SELECT user_is_member_of('marketinganalyst', 'sales_ro');

+-------------------+
| user_is_member_of |
+-------------------+
```

```
| false             |
+-------------------+
```

RBAC 的系统视图（可选）

要查看角色、用户角色分配、角色层次结构以及通过角色对数据库对象的权限，请使用 Amazon Redshift 的系统视图。超级用户和普通用户均可查看这些视图。超级用户可以查看所有角色详细信息。普通用户只能查看他们已被授予访问权限的角色的详细信息。

1. 要查看在集群中被显式授予了角色的用户列表，请使用以下示例。

```
SELECT * FROM svv_user_grants;
```

2. 要查看在集群中被显式授予了角色的角色列表，请使用以下示例。

```
SELECT * FROM svv_role_grants;
```

有关系统视图的完整列表，请参阅SVV 元数据视图。

在 RBAC 中使用行级别安全性（可选）

要对敏感数据进行精细访问控制，请使用行级别安全性（RLS）。有关 RLS 的更多信息，请参阅 行级别安全性。

在本节中，您将创建一个 RLS 策略，该策略仅允许 salesengineer 用户查看 cat 表中具有 Major League Baseball catdesc 值的行。然后，您以 salesengineer 用户身份查询数据库。

1. 以 salesengineer 用户身份连接到数据库。
2. 要查看 cat 表中的前 5 个条目，请使用以下示例。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
|     1 | Sports   | MLB     | Major League Baseball           |
```

```
|     2 | Sports   | NHL     | National Hockey League          |
|     3 | Sports   | NFL     | National Football League        |
|     4 | Sports   | NBA     | National Basketball Association |
|     5 | Sports   | MLS     | Major League Soccer             |
+-------+----------+---------+---------------------------------+
```

3. 以 dbadmin 用户身份连接到数据库。
4. 要为 cat 表中的 catdesc 列创建 RLS 策略，请使用以下示例。

```
CREATE RLS POLICY policy_mlb_engineer
WITH (catdesc VARCHAR(50))
USING (catdesc = 'Major League Baseball');
```

5. 要将 RLS 策略附加到 sales_rw 角色，请使用以下示例。

```
ATTACH RLS POLICY policy_mlb_engineer ON sales.cat TO ROLE sales_rw;
```

6. 要更改表以打开 RLS，请使用以下示例。

```
ALTER TABLE sales.cat ROW LEVEL SECURITY ON;
```

7. 以 salesengineer 用户身份连接到数据库。
8. 要尝试查看 cat 表中的前 5 个条目，请使用以下示例。请注意，仅当 catdesc 列为 Major League Baseball 时，才会显示条目。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+----------------------+
| catid | catgroup | catname |        catdesc       |
+-------+----------+---------+----------------------+
|     1 | Sports   | MLB     | Major League Baseball |
+-------+----------+---------+----------------------+
```

9. 以 salesanalyst 用户身份连接到数据库。
10. 要尝试查看 cat 表中的前 5 个条目，请使用以下示例。请注意，由于应用了默认的“全部拒绝”策略，因此不会显示任何条目。

```
SELECT *
```

```
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+----------------------+
| catid | catgroup | catname |        catdesc       |
+-------+----------+---------+----------------------+
```

11以 dbadmin 用户身份连接到数据库。

12要向 sales_ro 角色授予 IGNORE RLS 权限，请使用以下示例。这将授予 salesanalyst 用户忽略 RLS 策略的权限，因为他们是 sales_ro 角色的成员。

```
GRANT IGNORE RLS TO ROLE sales_ro;
```

13以 salesanalyst 用户身份连接到数据库。

14要查看 cat 表中的前 5 个条目，请使用以下示例。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
|     1 | Sports   | MLB     | Major League Baseball           |
|     2 | Sports   | NHL     | National Hockey League          |
|     3 | Sports   | NFL     | National Football League        |
|     4 | Sports   | NBA     | National Basketball Association |
|     5 | Sports   | MLS     | Major League Soccer             |
+-------+----------+---------+---------------------------------+
```

15以 dbadmin 用户身份连接到数据库。

16要撤销 sales_ro 角色的 IGNORE RLS 权限，请使用以下示例。

```
REVOKE IGNORE RLS FROM ROLE sales_ro;
```

17以 salesanalyst 用户身份连接到数据库。

18要尝试查看 cat 表中的前 5 个条目，请使用以下示例。请注意，由于应用了默认的“全部拒绝”策略，因此不会显示任何条目。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+----------------------+
| catid | catgroup | catname |        catdesc       |
+-------+----------+---------+----------------------+
```

19以 dbadmin 用户身份连接到数据库。

20要将 RLS 策略从 cat 表中分离，请使用以下示例。

```
DETACH RLS POLICY policy_mlb_engineer ON cat FROM ROLE sales_rw;
```

21以 salesanalyst 用户身份连接到数据库。

22要尝试查看 cat 表中的前 5 个条目，请使用以下示例。请注意，由于应用了默认的“全部拒绝”策略，因此不会显示任何条目。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
|     1 | Sports   | MLB     | Major League Baseball           |
|     2 | Sports   | NHL     | National Hockey League          |
|     3 | Sports   | NFL     | National Football League        |
|     4 | Sports   | NBA     | National Basketball Association |
|     5 | Sports   | MLS     | Major League Soccer             |
+-------+----------+---------+---------------------------------+
```

23以 dbadmin 用户身份连接到数据库。

24要删除 RLS 策略，请使用以下示例。

```
DROP RLS POLICY policy_mlb_engineer;
```

25要删除 RLS，请使用以下示例。

```
ALTER TABLE cat ROW LEVEL SECURITY OFF;
```

相关 主题

有关 RBAC 的更多信息，请参阅以下文档：

- 角色层次结构
- 角色分配
- 数据库对象权限
- 适用于 RBAC 的 ALTER DEFAULT PRIVILEGES

行级别安全性

使用 Amazon Redshift 中的行级别安全性 (RLS)，您可以对敏感数据进行精细访问控制。您可以根据在数据库对象级别定义的安全策略，来决定哪些用户或角色可以访问架构或表中数据的特定记录。除了列级别安全性（您可以向用户授予对列的子集的权限）之外，还可使用 RLS 策略进一步限制对可见列中特定行的访问。有关列级别安全性的更多信息，请参阅 列级访问控制的使用说明。

在对表强制实施 RLS 策略时，您可以限制用户运行查询时返回的结果集。

在创建 RLS 策略时，您可以指定表达式，规定 Amazon Redshift 是否在查询中返回表中的任何现有行。通过创建 RLS 策略来限制访问，您不必在查询中添加或外部化其他条件。

在创建 RLS 策略时，我们建议您创建简单策略，并且避免在策略中使用复杂语句。在定义 RLS 策略时，不要在基于策略的策略定义中使用过多表联接。

当某个策略引用某个查找表时，Amazon Redshift 除了扫描该策略所在的表以外，还会扫描其他表。对于附加了 RLS 策略的用户和未附加任何策略的用户，同一查询之间将存在性能差异。

在 SQL 语句中使用 RLS 策略

在 SQL 语句中使用 RLS 策略时，Amazon Redshift 将应用以下规则：

- 默认情况下，Amazon Redshift 会将 RLS 策略应用于 SELECT、UPDATE 和 DELETE 语句。
- 对于 SELECT 和 UNLOAD，Amazon Redshift 将根据您定义的策略筛选行。
- 对于 UPDATE，Amazon Redshift 将仅更新对您可见的行。如果策略限制了表中行的子集，则您将无法更新这些行。

- 对于 DELETE，您只能删除对您可见的行。如果策略限制了表中行的子集，则您将无法删除这些行。对于 TRUNCATE，您仍然可以截断表。
- 对于 CREATE TABLE LIKE，使用 LIKE 选项创建的表不会继承源表的权限设置。同样，目标表也不会继承源表的 RLS 策略。

为每个用户组合多个策略

Amazon Redshift 中的 RLS 支持为每个用户和对象附加多个策略。当为一个用户定义了多个策略时，Amazon Redshift 会根据表的 RLS CONJUNCTION TYPE 设置，使用 AND 或 OR 语法应用所有策略。有关联接类型的更多信息，请参阅ALTER TABLE。

一个表上的多个策略可以与您关联。将多个策略直接附加到您，或是您属于多个角色，并且这些角色附加了不同策略。

当多个策略应限制给定关系中的行访问权限时，可以将该关系的 RLS CONJUNCTION TYPE 设置为 AND。考虑以下示例。Alice 只能看到策略指定的“catname”为 NBA 的体育赛事。

```
-- Create an analyst role and grant it to a user named Alice.
CREATE ROLE analyst;
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
GRANT ROLE analyst TO alice;

-- Create an RLS policy that only lets the user see sports.
CREATE RLS POLICY policy_sports
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Sports');

-- Create an RLS policy that only lets the user see NBA.
CREATE RLS POLICY policy_nba
WITH (catname VARCHAR(10))
USING (catname = 'NBA');

-- Attach both to the analyst role.
ATTACH RLS POLICY policy_sports ON category TO ROLE analyst;
ATTACH RLS POLICY policy_nba ON category TO ROLE analyst;

-- Activate RLS on the category table with AND CONJUNCTION TYPE.
ALTER TABLE category ROW LEVEL SECURITY ON CONJUNCTION TYPE AND;

-- Change session to Alice.
SET SESSION AUTHORIZATION alice;
```

```
-- Select all from the category table.
SELECT catgroup, catname
FROM category;

 catgroup | catname
---------+---------
 Sports   | NBA
(1 row)
```

当多个策略应允许用户查看给定关系中的更多行时，用户可以将该关系的 RLS CONJUNCTION TYPE 设置为 OR。考虑以下示例。Alice 只能看到策略指定的“Concerts”和“Sports”。

```
-- Create an analyst role and grant it to a user named Alice.
CREATE ROLE analyst;
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
GRANT ROLE analyst TO alice;

-- Create an RLS policy that only lets the user see concerts.
CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');

-- Create an RLS policy that only lets the user see sports.
CREATE RLS POLICY policy_sports
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Sports');

-- Attach both to the analyst role.
ATTACH RLS POLICY policy_concerts ON category TO ROLE analyst;
ATTACH RLS POLICY policy_sports ON category TO ROLE analyst;

-- Activate RLS on the category table with OR CONJUNCTION TYPE.
ALTER TABLE category ROW LEVEL SECURITY ON CONJUNCTION TYPE OR;

-- Change session to Alice.
SET SESSION AUTHORIZATION alice;

-- Select all from the category table.
SELECT catgroup, count(*)
FROM category
GROUP BY catgroup ORDER BY catgroup;
```

```
 catgroup | count
---------+-------
 Concerts |  3
 Sports   |  5
(2 rows)
```

RLS 策略所有权和管理

作为超级用户、安全管理员或具有 sys:secadmin 角色的用户，您可以创建、修改或管理表的所有 RLS 策略。在对象级别，您可以打开或关闭行级别安全性，而无需修改表的架构定义。

要开始使用行级别安全性，以下是您可以使用的 SQL 语句：

- 使用 ALTER TABLE 语句可以打开或关闭表上的 RLS。有关更多信息，请参阅 ALTER TABLE。
- 使用 CREATE RLS POLICY 语句可为一个或多个表创建安全策略，并在该策略中指定一个或多个用户或角色。

  有关更多信息，请参阅 CREATE RLS POLICY。
- 使用 ALTER RLS POLICY 语句更改策略，例如更改策略定义。您可以对多个表或视图使用相同的策略。

  有关更多信息，请参阅 ALTER RLS POLICY。
- 使用 ATTACH RLS POLICY 语句可将策略附加到一个或多个关系、一个或多个用户，或者角色。

  有关更多信息，请参阅 ATTACH RLS POLICY 。
- 使用 DETACH RLS POLICY 语句可将策略与一个或多个关系、一个或多个用户或者角色分离。

  有关更多信息，请参阅 DETACH RLS POLICY 。
- 使用 DROP RLS POLICY 语句可以删除策略。

  有关更多信息，请参阅 DROP RLS POLICY 。
- 使用 GRANT 和 REVOKE 语句可以明确授予和撤消对引用查找表的 RLS 策略的 SELECT 权限。有关更多信息，请参阅GRANT 和REVOKE。

要监控创建的策略，sys:secadmin 可以查看 SVV_RLS_POLICY 和 SVV_RLS_ATTACHED_POLICY。

要列出受 RLS 保护的关系，sys:secadmin 可以查看 SVV_RLS_RELATION。

要跟踪 RLS 策略在引用受 RLS 保护的关系的查询中的应用情况，超级用户、sys:operator 或任何拥有系统权限 ACCESS SYSTEM TABLE 的用户均可查看 SVV_RLS_APPLIED_POLICY 。请注意，默认情况下不会向 sys:secadmin 授予这些权限。

要查询附加了 RLS 策略的表，但看不到这些表，可以向任何用户授予 IGNORE RLS 权限。作为超级用户或 sys:secadmin 的用户将被自动授予 IGNORE RLS 权限。有关更多信息，请参阅 GRANT。

要解释 EXPLAIN 计划中查询的 RLS 策略筛选器以解决与 RLS 相关的查询，您可以向任何用户授予 EXPLAIN RLS 权限。有关更多信息，请参阅GRANT 和EXPLAIN。

依赖于策略的对象和原则

为了为应用程序提供安全性并防止策略对象过时或失效，Amazon Redshift 不允许删除或更改 RLS 策略引用的对象。

以下示例说明了如何跟踪架构依赖关系。

```
-- The CREATE and ATTACH policy statements for `policy_events` references some
-- target and lookup tables.
-- Target tables are tickit_event_redshift and target_schema.target_event_table.
-- Lookup table is tickit_sales_redshift.
-- Policy `policy_events` has following dependencies:
--   table tickit_sales_redshift column eventid, qtysold
--   table tickit_event_redshift column eventid
--   table target_event_table column eventid
--   schema public and target_schema
CREATE RLS POLICY policy_events
WITH (eventid INTEGER)
USING (
    eventid IN (SELECT eventid FROM tickit_sales_redshift WHERE qtysold <3)
);

ATTACH RLS POLICY policy_events ON tickit_event_redshift TO ROLE analyst;

ATTACH RLS POLICY policy_events ON target_schema.target_event_table TO ROLE consumer;
```

下面列出了 Amazon Redshift 针对 RLS 策略跟踪的架构对象依赖关系。

- 在跟踪目标表的架构对象依赖关系时，Amazon Redshift 将遵循以下规则：
  - 当您删除目标表时，Amazon Redshift 会将策略与关系、用户、角色或公共分离。
  - 当您重命名目标表名称时，对附加的策略没有任何影响。

  - 如果您首先删除或分离策略，您只能删除策略定义中引用的目标表的列。在指定 CASCADE 选项时，这也将适用。您可以删除目标表中的其他列。
  - 您无法重命名目标表的引用列。要重命名引用列，请先分离策略。在指定 CASCADE 选项时，这也将适用。
  - 即便指定了 CASCADE 选项，您也无法更改引用列的类型。
- 在跟踪查找表的架构对象依赖关系时，Amazon Redshift 将遵循以下规则：
  - 您无法删除查找表。要删除查找表，请先删除引用该查找表的策略。
  - 您无法重命名查找表。要重命名查找表，请先删除引用该查找表的策略。在指定 CASCADE 选项时，这也将适用。
  - 您无法删除策略定义中使用的查找表列。要删除策略定义中使用的查找表列，请先删除引用该查找表的策略。当在 ALTER TABLE DROP COLUMN 语句中指定了 CASCADE 选项时，此规则也适用。您可以删除查找表中的其他列。
  - 您无法重命名查找表的引用列。要重命名引用列，请先删除引用该查找表的策略。在指定 CASCADE 选项时，这也将适用。
  - 您无法更改引用列的类型。
- 在删除用户或角色时，Amazon Redshift 将自动分离附加到该用户或角色的所有策略。
- 当您在 DROP SCHEMA 语句中使用 CASCADE 选项时，Amazon Redshift 还会删除架构中的关系。它还会删除依赖于已删除架构中的关系的任何其他架构中的关系。对于作为策略中查找表的关系，Amazon Redshift 将使 DROP SCHEMA DDL 失败。对于 DROP SCHEMA 语句删除的任何关系，Amazon Redshift 将分离附加到这些关系的所有策略。
- 当您同时删除策略时，您只能删除查找函数（在策略定义中引用的函数）。在指定 CASCADE 选项时，这也将适用。
- 在将策略附加到表时，Amazon Redshift 会检查此表是否是不同策略中的查找表。如果属于这种情况，Amazon Redshift 将不允许将策略附加到此表。
- 在创建 RLS 策略时，Amazon Redshift 会检查此表是否是任何其他 RLS 策略的目标表。如果属于这种情况，Amazon Redshift 将不允许在此表上创建策略。

使用 RLS 策略的注意事项

以下是使用 RLS 策略时的注意事项：

- Amazon Redshift 可将 RLS 策略应用于 SELECT、UPDATE 或 DELETE 语句。
- Amazon Redshift 不会将 RLS 策略应用于 INSERT、COPY、ALTER TABLE APPEND 语句。

- 行级别安全性与列级别安全性配合使用，以保护您的数据。
- 如果您的 Amazon Redshift 集群先前位于支持 RLS 的最新正式发布版本上，但现在已降级到早期版本，则当您对附加了 RLS 策略的基表运行查询时，Amazon Redshift 将返回错误。sys:secadmin 可以撤消被授予受限策略的用户的访问权限、关闭针对表的 RLS，以及删除策略。
- 在为源关系启用 RLS 后，Amazon Redshift 将为超级用户、已被明确授予系统权限 IGNORE RLS 或 sys:secadmin 角色的用户支持 ALTER TABLE APPEND 语句。在这种情况下，您可以通过从现有源表移动数据，运行 ALTER TABLE APPEND 语句，以将行附加到目标表。Amazon Redshift 会将所有元组从源关系移动到目标关系中。目标关系的 RLS 状态不会影响 ALTER TABLE APPEND 语句。
- 为了便于从其他数据仓库系统迁移，您可以通过指定变量名称和值，以设置和检索连接的自定义会话上下文变量。

  以下示例为行级别安全性 (RLS) 策略设置了会话上下文变量。

```
-- Set a customized context variable.
SELECT set_config('app.category', 'Concerts', FALSE);

-- Create a RLS policy using current_setting() to get the value of a customized
 context variable.
CREATE RLS POLICY policy_categories
WITH (catgroup VARCHAR(10))
USING (catgroup = current_setting('app.category', FALSE));

-- Set correct roles and attach the policy on the target table to one or more roles.
ATTACH RLS POLICY policy_categories ON tickit_category_redshift TO ROLE analyst, ROLE
 dbadmin;
```

  有关如何设置和检索自定义会话上下文变量的详细信息，请参阅 SET、SET_CONFIG、SHOW、CURRENT_SETTING 和 RESET。
- 在 DECLARE 和 FETCH 之间或后续 FETCH 语句之间使用 SET SESSION AUTHORIZATION 更改会话用户不会刷新根据 DECLARE 时的用户策略已经准备好的计划。将游标与受 RLS 保护的表一起使用时，请避免更改会话用户。
- 当视图对象内的基础对象受 RLS 保护时，附加到运行查询的用户的策略将应用于相应的基本对象。这与对象级权限检查不同，在对象级权限检查中，根据视图基本对象检查视图所有者的权限。您可以在查询的 EXPLAIN 计划输出中查看受 RLS 保护的关系。
- 当在附加到用户的关系的 RLS 策略中引用用户定义函数 (UDF) 时，用户必须拥有对该 UDF 的 EXECUTE 权限才能查询该关系。

- 行级安全性可能会限制查询优化。我们建议在大型数据集上部署受 RLS 保护的视图之前，仔细评估查询性能。
- 应用于后期绑定视图的行级安全策略可能会推送到联合表中。这些 RLS 策略可能在外部处理引擎日志中可见。

限制

以下是使用 RLS 策略时的限制：

- Amazon Redshift 可以针对某些包含具有复杂联接的查询的 RLS 策略支持 SELECT 语句，但不支持 UPDATE 或 DELETE 语句。在使用 UPDATE 或 DELETE 语句的情况下，Amazon Redshift 将返回以下错误：

```
ERROR: One of the RLS policies on target relation is not supported in UPDATE/DELETE.
```

- 每当在附加到用户的关系的 RLS 策略中引用用户定义函数 (UDF) 时，用户都必须拥有对该 UDF 的 EXECUTE 权限才能查询该关系。
- 不支持关联的子查询。Amazon Redshift 将返回以下错误：

```
ERROR: RLS policy could not be rewritten.
```

- RLS 策略不能附加到外部表和实体化视图。
- Amazon Redshift 不支持与 RLS 的数据共享。如果关系没有对数据共享关闭 RLS，则使用者集群上的查询将失败，并出现以下错误：

```
RLS-protected relation "rls_protected_table" cannot be accessed via datasharing
 query.
```

- 在跨数据库查询中，Amazon Redshift 会阻止读取受 RLS 保护的关系。拥有 IGNORE RLS 权限的用户可以使用跨数据库查询访问受保护的关系。当没有 IGNORE RLS 权限的用户通过跨数据库查询访问受 RLS 保护的关系时，会出现以下错误：

```
RLS-protected relation "rls_protected_table" cannot be accessed via cross-database
 query.
```

- ALTER RLS POLICY 仅支持使用 USING（using_predicate_exp）子句修改 RLS 策略。运行 ALTER RLS POLICY 时，您无法使用 WITH 子句修改 RLS 策略。
- 如果以下任一配置选项的值与会话的默认值不匹配，则无法查询已开启行级安全性的关系：

- enable_case_sensitive_super_attribute
- enable_case_sensitive_identifier
- downcase_delimited_identifier

如果您试图查询具有行级安全性的关系，并看到消息“RLS 保护的关系不支持会话级别配置，因为区分大小写不同于其默认值”，请考虑重置会话的配置选项。

- 当您的预调配集群或无服务器命名空间具有任何行级安全策略时，普通用户将无法使用以下命令：

```
ALTER <current_user> SET enable_case_sensitive_super_attribute/
enable_case_sensitive_identifier/downcase_delimited_identifier
```

创建 RLS 策略时，我们建议您更改普通用户的默认配置选项设置，使其与创建策略时会话的配置选项设置相匹配。超级用户和具有 ALTER USER 权限的用户可以使用参数组设置或 ALTER USER 命令来执行此操作。有关参数组的信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 参数组。有关 ALTER USER 命令的信息，请参阅 ALTER USER。

- 普通用户无法使用 CREATE VIEW 命令替换具有行级安全策略的视图和后期绑定视图。要替换具有 RLS 策略的视图或 LBV，请先分离附加到这些视图的所有 RLS 策略，替换视图或 LBV，然后重新附加策略。具有 sys:secadmin permission 的超级用户和用户可以在具有 RLS 策略的视图或 LBV 上使用 CREATE VIEW，而无需分离策略。
- 具有行级安全策略的视图不能引用系统表和系统视图。
- 常规视图引用的后期绑定视图无法受到 RLS 保护。
- 无法在同一个查询中访问受 RLS 保护的关系和来自数据湖的嵌套数据。

RLS 性能最佳实践

以下是确保 Amazon Redshift 在受 RLS 保护的表上能够获得更高性能的最佳实践。

运算符和函数的安全性

当查询受 RLS 保护的表时，使用某些运算符或函数可能会导致性能下降。对于查询受 RLS 保护的表，Amazon Redshift 会将运算符和函数分类为安全或不安全。如果根据输入，函数或运算符没有任何可观察到的副作用，则该函数或运算符将被分类为 RLS 安全。特别是，RLS 安全的函数或运算符不能是以下情况之一：

- 输出输入值，或任何依赖于输入值的值，无论是否显示错误消息。
- 失败或返回依赖于输入值的错误。

RLS 不安全的运算符包括：

- 算术运算符：+、-、/、*、%。
- 文本运算符：LIKE 和 SIMILAR TO。
- 强制转换运算符。
- UDF。

使用以下 SELECT 语句以检查运算符和函数的安全性。

```
SELECT proname, proc_is_rls_safe(oid) FROM pg_proc;
```

在规划对受 RLS 保护的表进行查询时，Amazon Redshift 将对包含 RLS 不安全运算符和函数的用户谓词的评估顺序施加限制。在查询受 RLS 保护的表时，引用 RLS 不安全运算符或函数的查询可能会导致性能下降。当 Amazon Redshift 无法将 RLS 不安全谓词向下推送到基表扫描以利用排序键时，性能可能会显著下降。为了获得更高性能，请避免使用利用排序键的 RLS 不安全谓词进行的查询。要验证 Amazon Redshift 是否能够向下推送运算符和函数，您可以将 EXPLAIN 语句与系统权限 EXPLAIN RLS 结合使用。

结果缓存

为了缩短查询运行时间并提高系统性能，Amazon Redshift 在领导节点的内存中缓存特定查询类型的结果。

在满足未受保护的表的所有条件并且满足以下所有条件时，Amazon Redshift 会将缓存的结果用于新查询来扫描受 RLS 保护的表：

- 未曾修改策略中的表或视图。
- 策略不使用在每次运行时必须求值的函数，如 GETDATE 或 CURRENT_USER。

为了获得更高性能，请避免使用不满足前述条件的策略谓词。

有关 Amazon Redshift 中的结果缓存的更多信息，请参阅 结果缓存 。

复杂策略

为了获得更高性能，请避免将复杂策略与联接多个表的子查询配合使用。

创建、附加、分离和删除 RLS 策略

您可以执行以下操作：

- 要创建 RLS 策略，请使用 CREATE RLS POLICY 命令。
- 要将表上的 RLS 策略附加到一个或多个用户或角色，请使用 ATTACH RLS POLICY 命令。
- 要将表上的行级别安全性策略与一个或多个用户或角色分离，请使用 DETACH RLS POLICY 命令。
- 要删除所有数据库中所有表的 RLS 策略，请使用 DROP RLS POLICY 命令。

下面是一个端到端的示例，用于说明超级用户如何创建某些用户和角色。然后，具有 secadmin 角色的用户将创建、附加、分离和删除 RLS 策略。此示例使用 tickit 示例数据库。有关更多信息，请参阅《Amazon Redshift 入门指南》中的将数据从 Amazon S3 加载到 Amazon Redshift。

```
-- Create users and roles referenced in the policy statements.
CREATE ROLE analyst;
CREATE ROLE consumer;
CREATE ROLE dbadmin;
CREATE ROLE auditor;
CREATE USER bob WITH PASSWORD 'Name_is_bob_1';
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
CREATE USER joe WITH PASSWORD 'Name_is_joe_1';
CREATE USER molly WITH PASSWORD 'Name_is_molly_1';
CREATE USER bruce WITH PASSWORD 'Name_is_bruce_1';
GRANT ROLE sys:secadmin TO bob;
GRANT ROLE analyst TO alice;
GRANT ROLE consumer TO joe;
GRANT ROLE dbadmin TO molly;
GRANT ROLE auditor TO bruce;
GRANT ALL ON TABLE tickit_category_redshift TO PUBLIC;
GRANT ALL ON TABLE tickit_sales_redshift TO PUBLIC;
GRANT ALL ON TABLE tickit_event_redshift TO PUBLIC;

-- Create table and schema referenced in the policy statements.
CREATE SCHEMA target_schema;
GRANT ALL ON SCHEMA target_schema TO PUBLIC;
CREATE TABLE target_schema.target_event_table (LIKE tickit_event_redshift);
GRANT ALL ON TABLE target_schema.target_event_table TO PUBLIC;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;
```

```
-- Check the tuples visible to analyst alice.
-- Should contain all 3 categories.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');

SELECT poldb, polname, polalias, polatts, polqual, polenabled, polmodifiedby FROM
 svv_rls_policy WHERE poldb = CURRENT_DATABASE();

ATTACH RLS POLICY policy_concerts ON tickit_category_redshift TO ROLE analyst, ROLE
 dbadmin;

ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;

SELECT * FROM svv_rls_attached_policy;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check that tuples with only `Concert` category will be visible to analyst alice.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to consumer joe.
SET SESSION AUTHORIZATION joe;

-- Although the policy is attached to a different role, no tuples will be
-- visible to consumer joe because the default deny all policy is applied.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to dbadmin molly.
SET SESSION AUTHORIZATION molly;
```

```
-- Check that tuples with only `Concert` category will be visible to dbadmin molly.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Check that EXPLAIN output contains RLS SecureScan to prevent disclosure of
-- sensitive information such as RLS filters.
EXPLAIN SELECT catgroup, count(*) FROM tickit_category_redshift GROUP BY catgroup ORDER
 BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

-- Grant IGNORE RLS permission so that RLS policies do not get applicable to role
 dbadmin.
GRANT IGNORE RLS TO ROLE dbadmin;

-- Grant EXPLAIN RLS permission so that anyone in role auditor can view complete
 EXPLAIN output.
GRANT EXPLAIN RLS TO ROLE auditor;

-- Change session to dbadmin molly.
SET SESSION AUTHORIZATION molly;

-- Check that all tuples are visible to dbadmin molly because `IGNORE RLS` is granted
 to role dbadmin.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to auditor bruce.
SET SESSION AUTHORIZATION bruce;

-- Check explain plan is visible to auditor bruce because `EXPLAIN RLS` is granted to
 role auditor.
EXPLAIN SELECT catgroup, count(*) FROM tickit_category_redshift GROUP BY catgroup ORDER
 BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

DETACH RLS POLICY policy_concerts ON tickit_category_redshift FROM ROLE analyst, ROLE
 dbadmin;
```

```
-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check that no tuples are visible to analyst alice.
-- Although the policy is detached, no tuples will be visible to analyst alice
-- because of default deny all policy is applied if the table has RLS on.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

CREATE RLS POLICY policy_events
WITH (eventid INTEGER) AS ev
USING (
    ev.eventid IN (SELECT eventid FROM tickit_sales_redshift WHERE qtysold <3)
);

ATTACH RLS POLICY policy_events ON tickit_event_redshift TO ROLE analyst;
ATTACH RLS POLICY policy_events ON target_schema.target_event_table TO ROLE consumer;

RESET SESSION AUTHORIZATION;

-- Can not cannot alter type of dependent column.
ALTER TABLE target_schema.target_event_table ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_event_redshift ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_sales_redshift ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_sales_redshift ALTER COLUMN qtysold TYPE float;

-- Can not cannot rename dependent column.
ALTER TABLE target_schema.target_event_table RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_event_redshift RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_sales_redshift RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_sales_redshift RENAME COLUMN qtysold TO renamed_qtysold;

-- Can not drop dependent column.
ALTER TABLE target_schema.target_event_table DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_event_redshift DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_sales_redshift DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_sales_redshift DROP COLUMN qtysold CASCADE;

-- Can not drop lookup table.
DROP TABLE tickit_sales_redshift CASCADE;
```

```
-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

DROP RLS POLICY policy_concerts;
DROP RLS POLICY IF EXISTS policy_events;

ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY OFF;

RESET SESSION AUTHORIZATION;

-- Drop users and roles.
DROP USER bob;
DROP USER alice;
DROP USER joe;
DROP USER molly;
DROP USER bruce;
DROP ROLE analyst;
DROP ROLE consumer;
DROP ROLE auditor FORCE;
DROP ROLE dbadmin FORCE;
```

元数据安全性

与 Amazon Redshift 的行级安全性一样，元数据安全性可让您对元数据进行更精细的控制。如果为预置集群或无服务器工作组启用了元数据安全性，则用户可以查看他们有权查看的对象的元数据。元数据安全性可让您根据自己的需求来区分可见性。例如，您可以使用单个数据仓库来集中所有数据存储。但是，如果您存储多个扇区的数据，则管理安全性可能会变得很麻烦。启用元数据安全性后，您可以配置自己的可见性。一个扇区的用户可以对其对象有更多的可见性，而您则限制另一个扇区用户的查看权限。元数据安全性支持所有对象类型，例如架构、表、视图、实体化视图、存储过程、用户定义的函数和机器学习模型。

用户可以在以下情况下查看对象的元数据：

- 如果向用户授予对象访问权限。
- 如果向用户所属的组或角色授予对象访问权限。
- 对象是公开的。
- 用户是数据库对象的拥有者。

要启用元数据安全性，请使用 ALTER SYSTEM 命令。以下是如何使用具有元数据安全性的 ALTER SYSTEM 命令的语法。

```
ALTER SYSTEM SET metadata_security=[true|t|on|false|f|off];
```

启用元数据安全性后，所有拥有必要权限的用户都可以查看他们有权访问的对象的相关元数据。如果您只希望某些用户能够查看元数据安全性，请向角色授予 ACCESS CATALOG 权限，然后将该角色分配给用户。有关使用角色更好地控制安全性的更多信息，请参阅基于角色的访问控制。

以下示例演示如何向角色授予 ACCESS CATALOG 权限，然后将该角色分配给用户。有关授予权限的更多信息，请参阅 GRANT 命令。

```
CREATE ROLE sample_metadata_viewer;

GRANT ACCESS CATALOG TO ROLE sample_metadata_viewer;

GRANT ROLE sample_metadata_viewer to salesadmin;
```

如果您希望使用已定义的角色，则系统定义的角色 operator、secadmin、dba 和 superuser 都具有查看对象元数据的必要权限。默认情况下，超级用户可以看到完整的目录。

```
GRANT ROLE operator to sample_user;
```

如果您使用角色来控制元数据安全性，则可以访问基于角色的访问控制附带的所有系统视图和函数。例如，您可以查询 SVV_ROLES 视图以查看所有角色。要查看用户是否是角色或组的成员，请使用 USER_IS_MEMBER_OF 函数。有关 SVV 视图的完整列表，请参阅 SVV 元数据视图。有关系统信息函数的列表，请参阅系统信息函数。

动态数据掩蔽

概述

使用 Amazon Redshift 中的动态数据掩蔽 (DDM)，您可以保护数据仓库中的敏感数据。您可以操纵 Amazon Redshift 在查询时如何向用户显示敏感数据，而无需在数据库中对其进行转换。您可以通过将自定义模糊处理规则应用于给定用户或角色的屏蔽策略来控制对数据的访问。这样，您无需更改底层数据或编辑 SQL 查询即可响应不断变化的隐私要求。

动态数据掩蔽策略会隐藏、模糊处理或假名化与给定格式匹配的数据。附加到表时，对表的一个或多个列应用屏蔽表达式。您可以进一步修改屏蔽策略，使其仅应用于某些用户，或者应用于您可以使用 基

于角色的访问控制 (RBAC) 创建的用户定义角色。此外，在创建屏蔽策略时，您可以使用条件列在单元格级别应用 DDM。有关条件屏蔽的更多信息，请参阅条件动态数据掩蔽。

您可以将具有不同模糊处理级别的多个屏蔽策略应用于表中的同一列，并将它们分配给不同的角色。为避免在不同角色对一列应用不同策略时发生冲突，可以为每个应用程序设置优先级。通过这种方式，您可以控制给定用户或角色可以访问哪些数据。DDM 策略可以部分或完全编辑数据，也可以使用以 SQL、Python 或 AWS Lambda 编写的用户定义函数对其进行哈希处理。通过使用哈希屏蔽数据，您可以对这些数据应用联接，而无需访问潜在的敏感信息。

端到端示例

下面是一个端到端的示例，显示了如何创建屏蔽策略并将其附加到列。这些策略让用户可以访问列并查看不同的值，具体取决于与其角色相关的策略的混淆程度。您必须是超级用户或具有 `sys:secadmin` 角色才能运行此示例。

创建屏蔽策略

首先，创建一个表并填入信用卡值。

```
--create the table
CREATE TABLE credit_cards (
  customer_id INT,
  credit_card TEXT
);

--populate the table with sample values
INSERT INTO credit_cards
VALUES
  (100, '4532993817514842'),
  (100, '4716002041425888'),
  (102, '5243112427642649'),
  (102, '6011720771834675'),
  (102, '6011378662059710'),
  (103, '373611968625635')
;

--run GRANT to grant permission to use the SELECT statement on the table
GRANT SELECT ON credit_cards TO PUBLIC;

--create two users
CREATE USER regular_user WITH PASSWORD '1234Test!';
```

```
CREATE USER analytics_user WITH PASSWORD '1234Test!';

--create the analytics_role role and grant it to analytics_user
--regular_user does not have a role
CREATE ROLE analytics_role;

GRANT ROLE analytics_role TO analytics_user;
```

接下来，创建应用于分析角色的屏蔽策略。

```
--create a masking policy that fully masks the credit card number
CREATE MASKING POLICY mask_credit_card_full
WITH (credit_card VARCHAR(256))
USING ('000000XXXX0000'::TEXT);

--create a user-defined function that partially obfuscates credit card data
CREATE FUNCTION REDACT_CREDIT_CARD (credit_card TEXT)
RETURNS TEXT IMMUTABLE
AS $$
    import re
    regexp = re.compile("^([0-9]{6})[0-9]{5,6}([0-9]{4})")

    match = regexp.search(credit_card)
    if match != None:
        first = match.group(1)
        last = match.group(2)
    else:
        first = "000000"
        last = "0000"

    return "{}XXXXX{}".format(first, last)
$$ LANGUAGE plpythonu;

--create a masking policy that applies the REDACT_CREDIT_CARD function
CREATE MASKING POLICY mask_credit_card_partial
WITH (credit_card VARCHAR(256))
USING (REDACT_CREDIT_CARD(credit_card));

--confirm the masking policies using the associated system views
SELECT * FROM svv_masking_policy;

SELECT * FROM svv_attached_masking_policy;
```

附加屏蔽政策

将屏蔽政策附加到信用卡表中。

```
--attach mask_credit_card_full to the credit card table as the default policy
--all users will see this masking policy unless a higher priority masking policy is
 attached to them or their role
ATTACH MASKING POLICY mask_credit_card_full
ON credit_cards(credit_card)
TO PUBLIC;

--attach mask_credit_card_partial to the analytics role
--users with the analytics role can see partial credit card information
ATTACH MASKING POLICY mask_credit_card_partial
ON credit_cards(credit_card)
TO ROLE analytics_role
PRIORITY 10;

--confirm the masking policies are applied to the table and role in the associated
 system view
SELECT * FROM svv_attached_masking_policy;

--confirm the full masking policy is in place for normal users by selecting from the
 credit card table as regular_user
SET SESSION AUTHORIZATION regular_user;

SELECT * FROM credit_cards;

--confirm the partial masking policy is in place for users with the analytics role by
 selecting from the credit card table as analytics_user
SET SESSION AUTHORIZATION analytics_user;

SELECT * FROM credit_cards;
```

修改掩蔽政策

以下部分介绍了如何修改动态数据掩蔽政策。

```
--reset session authorization to the default
RESET SESSION AUTHORIZATION;

--alter the mask_credit_card_full policy
ALTER MASKING POLICY mask_credit_card_full
```

```
USING ('00000000000000'::TEXT);

--confirm the full masking policy is in place after altering the policy, and that
 results are altered from '000000XXXX0000' to '00000000000000'
SELECT * FROM credit_cards;
```

分离和删除屏蔽策略

以下部分显示如何通过从表中删除所有动态数据掩蔽策略来分离和删除屏蔽策略。

```
--reset session authorization to the default
RESET SESSION AUTHORIZATION;

--detach both masking policies from the credit_cards table
DETACH MASKING POLICY mask_credit_card_full
ON credit_cards(credit_card)
FROM PUBLIC;

DETACH MASKING POLICY mask_credit_card_partial
ON credit_cards(credit_card)
FROM ROLE analytics_role;

--drop both masking policies
DROP MASKING POLICY mask_credit_card_full;

DROP MASKING POLICY mask_credit_card_partial;
```

使用动态数据掩蔽时的注意事项

使用动态数据掩蔽时，请考虑以下事项：

- 在查询通过表创建的对象（例如视图）时，用户将看到基于他们自己的屏蔽策略的结果，而不是创建对象的用户的策略。例如，具有分析师角色的用户查询 secadmin 创建的视图时，他将看到附加到分析师角色的屏蔽策略的结果。
- 为防止 EXPLAIN 命令可能暴露敏感的屏蔽策略筛选条件，只有拥有 SYS_EXPLAIN_DDM 权限的用户才能看到在 EXPLAIN 输出中应用的屏蔽策略。默认情况下，用户没有 SYS_EXPLAIN_DDM 权限。

  向角色授予权限的语法如下。

  ```
  GRANT EXPLAIN MASKING TO ROLE rolename
  ```

有关 EXPLAIN 命令的更多信息，请参阅 EXPLAIN。

- 根据所使用的筛选条件或联接条件，具有不同角色的用户会看到不同的结果。例如，如果运行命令的用户应用了会模糊处理特定列的屏蔽策略，则在使用该列值的表上运行 SELECT 命令将失败。
- 必须在任何谓词操作或预测之前应用 DDM 策略。掩蔽策略可以包括：
  - 低成本常量操作，例如将值转换为 null
  - 中等成本操作，例如 HMAC 哈希
  - 高成本操作，例如调用外部 Lambda 用户定义函数

  因此，如果可能，我们建议您使用简单屏蔽表达式。
- 您可以对具有行级安全策略的角色使用 DDM 策略，但请注意，RLS 策略在 DDM 之前应用。动态数据掩蔽表达式将无法读取受 RLS 保护的行。有关 RLS 的更多信息，请参阅 行级别安全性。
- 使用 COPY 命令从 parquet 复制到受保护的目标表时，您应在 COPY 语句中明确指定列。有关使用 COPY 映射列的更多信息，请参阅列映射选项。
- DDM 策略不能附加到以下关系：
  - 系统表和目录
  - 外部表
  - 数据共享表
  - 实体化视图
  - 跨数据库关系
  - 临时表
  - 关联的查询
- DDM 策略可以包括查找表。查找表可以存在于 USING 子句中。以下关系类型不能用作查找表：
  - 系统表和目录
  - 外部表
  - 数据共享表
  - 视图、实体化视图和后期绑定视图
  - 跨数据库关系
  - 临时表
  - 关联的查询

以下是将掩蔽政策附加到查找表的示例。

```
--Create a masking policy referencing a lookup table
CREATE MASKING POLICY lookup_mask_credit_card WITH (credit_card TEXT) USING (
  CASE
    WHEN
      credit_card IN (SELECT credit_card_lookup FROM credit_cards_lookup)
    THEN '000000XXXX0000'
    ELSE REDACT_CREDIT_CARD(credit_card)
    END
  );

--Provides access to the lookup table via a policy attached to a role
GRANT SELECT ON TABLE credit_cards_lookup TO MASKING POLICY lookup_mask_credit_card;
```

- 您不能附加所产生的输出与目标列的类型和大小不兼容的屏蔽策略。例如，您不能附加将 12 个字符长的字符串输出到 VARCHAR(10) 列的屏蔽策略。Amazon Redshift 支持以下例外情况：
  - 只要 M < N，输入类型为 INTN 的屏蔽策略就可以附加到大小为 INTM 的策略上。例如，BIGINT (INT8) 输入策略可以附加到 smallint (INT4) 列。
  - 输入类型为 NUMERIC 或 DECIMAL 的屏蔽策略始终可以附加到 FLOAT 列。
- DDM 策略不能用于数据共享。如果数据共享的数据创建者将 DDM 策略附加到数据共享中的表，则尝试查询该表的数据使用者用户将无法访问该表。无法将附加 DDM 策略的表添加到数据共享中。
- 如果以下任一配置选项的值与会话的默认值不匹配，则无法查询附加了 DDM 策略的关系：
  - `enable_case_sensitive_super_attribute`
  - `enable_case_sensitive_identifier`
  - `downcase_delimited_identifier`

  如果您试图查询附加了 DDM 策略的关系，并看到消息“DDM 保护的关系不支持会话级别配置，因为区分大小写不同于其默认值”，请考虑重置会话的配置选项。
- 当您的预调配集群或无服务器命名空间具有任何动态数据掩蔽策略时，普通用户将无法使用以下命令：

  ```
  ALTER <current_user> SET enable_case_sensitive_super_attribute/
  enable_case_sensitive_identifier/downcase_delimited_identifier
  ```

  创建 DDM 策略时，我们建议您更改普通用户的默认配置选项设置，使其与创建策略时会话的配置选项设置相匹配。超级用户和具有 ALTER USER 权限的用户可以使用参数组设置或 ALTER USER 命令来执行此操作。有关参数组的信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 参数组。有关 ALTER USER 命令的信息，请参阅 ALTER USER。

- 普通用户无法使用 CREATE VIEW 命令替换已附加 DDM 策略的视图和后期绑定视图。要替换具有 DDM 策略的视图或 LBV，请先分离附加到这些视图的所有 DDM 策略，替换视图或 LBV，然后重新附加策略。具有 sys:secadmin 权限的超级用户和用户可以在具有 DDM 策略的视图或 LBV 上使用 CREATE VIEW，而无需分离策略。
- 附加了 DDM 策略的视图无法引用系统表和视图。后期绑定视图可以引用系统表和视图。
- 附加了 DDM 策略的后期绑定视图无法引用数据湖中的嵌套数据，例如 JSON 文档。
- 如果任何视图引用了后期绑定视图，则后期绑定视图不能附加 DDM 策略。
- 附加到后期绑定视图的 DDM 策略按列名附加。在查询时，Amazon Redshift 会验证是否已成功应用附加到后期绑定视图的所有掩蔽策略，以及后期绑定视图的输出列类型是否与附加掩蔽策略中的类型相匹配。如果验证失败，Amazon Redshift 将返回查询错误。

管理动态数据掩蔽策略

您可以执行以下操作：

- 要创建 DDM 策略，请使用 CREATE MASKING POLICY 命令。

  以下是使用 SHA-2 哈希函数创建屏蔽策略的示例。

```
CREATE MASKING POLICY hash_credit
WITH (credit_card varchar(256))
USING (sha2(credit_card + 'testSalt', 256));
```

- 要更改现有 DDM 政策，请使用 ALTER MASKING POLICY 命令。

  以下是更改现有掩蔽政策的示例。

```
ALTER MASKING POLICY hash_credit
USING (sha2(credit_card + 'otherTestSalt', 256));
```

- 要将表上的 DDM 策略附加到一个或多个用户或角色，请使用 ATTACH MASKING POLICY 命令。

  下面是将屏蔽策略附加到列/角色对的示例。

```
 ATTACH MASKING POLICY hash_credit
ON credit_cards (credit_card)
TO ROLE science_role
PRIORITY 30;
```

PRIORITY 子句确定当多个策略附加到同一列时，哪个屏蔽策略适用于用户会话。例如，如果前面示例中的用户在同一信用卡列附加了另一个屏蔽策略，该策略的优先级为 20，则 science_role 的策略是适用的策略，因为它具有更高的优先级，即 30。

- 要将表上的 DDM 策略与一个或多个用户或角色分离，请使用 DETACH MASKING POLICY 命令。

  下面是从列/角色对分离屏蔽策略的示例。

```
DETACH MASKING POLICY hash_credit
ON credit_cards(credit_card)
FROM ROLE science_role;
```

- 要从所有数据库删除 DDM 策略，请使用 DROP MASKING POLICY 命令。

  以下是从所有数据库删除屏蔽策略的示例。

```
DROP MASKING POLICY hash_credit;
```

屏蔽策略层次结构

在附加多个掩蔽策略时，请注意以下事项：

- 您可以将多个屏蔽策略附加到单个列。
- 当多个屏蔽策略适用于一个查询时，附加到每个相应列的最高优先级策略适用。考虑以下示例。

```
ATTACH MASKING POLICY partial_hash
ON credit_cards(address, credit_card)
TO ROLE analytics_role
PRIORITY 20;

ATTACH MASKING POLICY full_hash
ON credit_cards(credit_card, ssn)
TO ROLE auditor_role
PRIORITY 30;

SELECT address, credit_card, ssn
FROM credit_cards;
```

运行 SELECT 语句时，同时具有分析和审计角色的用户会看到应用了 partial_hash 屏蔽策略的地址列。他们会看到应用了 full_hash 掩蔽策略的信用卡和 SSN 列，因为 full_hash 策略在信用卡列上的优先级更高。

- 如果在附加屏蔽策略时未指定优先级，则默认为优先级为 0。
- 您不能将两个具有相同优先级的策略附加到同一列。
- 您不能将两个策略附加到用户和列或角色和列的相同组合。
- 当多个掩蔽策略适用于同一 SUPER 路径，同时附加到同一用户或角色时，只有优先级最高的掩蔽策略才会生效。考虑以下示例。

  第一个示例显示两个掩蔽策略附加在同一路径上，优先级较高的策略生效。

```
ATTACH MASKING POLICY hide_name
ON employees(col_person.name)
TO PUBLIC
PRIORITY 20;

ATTACH MASKING POLICY hide_last_name
ON employees(col_person.name.last)
TO PUBLIC
PRIORITY 30;

--Only the hide_last_name policy takes effect.
SELECT employees.col_person.name FROM employees;
```

  第二个示例显示两个掩蔽策略附加到同一 SUPER 对象中的不同路径，这两个策略之间没有冲突。这两个掩蔽策略将同时适用。

```
ATTACH MASKING POLICY hide_first_name
ON employees(col_person.name.first)
TO PUBLIC
PRIORITY 20;

ATTACH MASKING POLICY hide_last_name
ON employees(col_person.name.last)
TO PUBLIC
PRIORITY 20;

--Both col_person.name.first and col_person.name.last are masked.
SELECT employees.col_person.name FROM employees;
```

要确认哪个掩蔽策略适用于给定的用户和列或角色和列组合，具有 sys:secadmin 角色的用户可以在 SVV_ATTACHED_MASKING_POLICY 系统视图中查找列/角色或列/用户对。有关更多信息，请参阅用于动态数据掩蔽的系统视图。

对 SUPER 数据类型路径使用动态数据掩蔽

Amazon Redshift 支持将动态数据掩蔽策略附加到 SUPER 类型列的路径。有关 SUPER 数据类型的更多信息，请参阅在 Amazon Redshift 中摄取和查询半结构化数据。

将掩蔽策略附加到 SUPER 类型列的路径时，请考虑以下几点。

- 将掩蔽策略附加到列上的路径时，必须将该列定义为 SUPER 数据类型。只能对 SUPER 路径上的标量值应用掩蔽策略。不能将掩蔽策略应用于复杂的结构或数组。
- 只要 SUPER 路径不冲突，就可以对单个 SUPER 列上的多个标量值应用不同的掩蔽策略。例如，SUPER 路径 `a.b` 和 `a.b.c` 冲突，因为它们在同一路径上，`a.b` 是 `a.b.c` 的父路径。SUPER 路径 `a.b.c` 和 `a.b.d` 不冲突。
- 在用户查询运行时应用策略之前，Amazon Redshift 无法检查数据中是否存在掩蔽策略所附加的路径，也无法检查这些路径是否属于预期类型。例如，当您将掩蔽 TEXT 值的掩蔽策略附加到包含 INT 值的 SUPER 路径时，Amazon Redshift 会尝试在路径上转换值的类型。

  在这种情况下，Amazon Redshift 在运行时的行为取决于您用于查询 SUPER 对象的配置设置。默认情况下，Amazon Redshift 采用宽松模式，对于给定的 SUPER 路径，会将缺失的路径和无效的转换解析为 NULL。有关 SUPER 相关配置设置的更多信息，请参阅SUPER 配置。
- SUPER 是一种无架构类型，这意味着 Amazon Redshift 无法确认给定 SUPER 路径上的值是否存在。如果您将掩蔽策略附加到不存在的 SUPER 路径并且 Amazon Redshift 采用宽松模式，则 Amazon Redshift 会将该路径解析为 NULL 值。我们建议您在将掩蔽策略附加到 SUPER 列的路径时，考虑 SUPER 对象的预期格式以及它们具有意外属性的可能性。如果您认为 SUPER 列中可能存在意外架构，请考虑将掩蔽策略直接附加到 SUPER 列。您可以使用 SUPER 类型信息函数来检查属性和类型，并使用 OBJECT_TRANSFORM 来掩蔽这些值。有关 SUPER 类型信息函数的更多信息，请参阅SUPER 类型信息函数。

示例

将掩蔽策略附加到 SUPER 路径

以下示例在一列中将多个掩蔽策略附加到多个 SUPER 类型路径上。

```
CREATE TABLE employees (
```

```
    col_person SUPER
);

INSERT INTO employees
VALUES
    (
        json_parse('
            {
                "name": {
                    "first": "John",
                    "last": "Doe"
                },
                "age": 25,
                "ssn": "111-22-3333",
                "company": "Company Inc."
            }
        ')
    ),
    (
        json_parse('
            {
                "name": {
                    "first": "Jane",
                    "last": "Appleseed"
                },
                "age": 34,
                "ssn": "444-55-7777",
                "company": "Organization Org."
            }
        ')
    )
;
GRANT ALL ON ALL TABLES IN SCHEMA "public" TO PUBLIC;

-- Create the masking policies.

-- This policy converts the given name to all uppercase letters.
CREATE MASKING POLICY mask_first_name
WITH(first_name TEXT)
USING ( UPPER(first_name) );

-- This policy replaces the given name with the fixed string 'XXXX'.
CREATE MASKING POLICY mask_last_name
WITH(last_name TEXT)
```

```
USING ( 'XXXX'::TEXT );

-- This policy rounds down the given age to the nearest 10.
CREATE MASKING POLICY mask_age
WITH(age INT)
USING ( (FLOOR(age::FLOAT / 10) * 10)::INT );

-- This policy converts the first five digits of the given SSN to 'XXX-XX'.
CREATE MASKING POLICY mask_ssn
WITH(ssn TEXT)
USING ( 'XXX-XX-'::TEXT || SUBSTRING(ssn::TEXT FROM 8 FOR 4) );

-- Attach the masking policies to the employees table.
ATTACH MASKING POLICY mask_first_name
ON employees(col_person.name.first)
TO PUBLIC;

ATTACH MASKING POLICY mask_last_name
ON employees(col_person.name.last)
TO PUBLIC;

ATTACH MASKING POLICY mask_age
ON employees(col_person.age)
TO PUBLIC;

ATTACH MASKING POLICY mask_ssn
ON employees(col_person.ssn)
TO PUBLIC;

-- Verify that your masking policies are attached.
SELECT
    policy_name,
    TABLE_NAME,
    priority,
    input_columns,
    output_columns
FROM
    svv_attached_masking_policy;

   policy_name   | table_name | priority |           input_columns           |
 output_columns
-----------------+------------+----------+-----------------------------------
+-----------------------------------
```

```
 mask_age        | employees  |        0 | ["col_person.\"age\""]            |
 ["col_person.\"age\""]
 mask_first_name | employees  |        0 | ["col_person.\"name\".\"first\""] |
 ["col_person.\"name\".\"first\""]
 mask_last_name  | employees  |        0 | ["col_person.\"name\".\"last\""]  |
 ["col_person.\"name\".\"last\""]
 mask_ssn        | employees  |        0 | ["col_person.\"ssn\""]            |
 ["col_person.\"ssn\""]
(4 rows)

-- Observe the masking policies taking effect.
SELECT col_person FROM employees ORDER BY col_person.age;

-- This result is formatted for ease of reading.
         col_person
--------------------------------
{
    "name": {
        "first": "JOHN",
        "last": "XXXX"
    },
    "age": 20,
    "ssn": "XXX-XX-3333",
    "company": "Company Inc."
}
{
    "name": {
        "first": "JANE",
        "last": "XXXX"
    },
    "age": 30,
    "ssn": "XXX-XX-7777",
    "company": "Organization Org."
}
```

以下是附加到 SUPER 路径的无效掩蔽策略的一些示例。

```
-- This attachment fails because there is already a policy
-- with equal priority attached to employees.name.last, which is
-- on the same SUPER path as employees.name.
ATTACH MASKING POLICY mask_ssn
ON employees(col_person.name)
TO PUBLIC;
```

```
ERROR:  DDM policy "mask_last_name" is already attached on relation "employees" column
 "col_person."name"."last"" with same priority

-- Create a masking policy that masks DATETIME objects.
CREATE MASKING POLICY mask_date
WITH(INPUT DATETIME)
USING ( INPUT );

-- This attachment fails because SUPER type columns can't contain DATETIME objects.
ATTACH MASKING POLICY mask_date
ON employees(col_person.company)
TO PUBLIC;
ERROR:  cannot attach masking policy for output of type "timestamp without time zone"
 to column "col_person."company"" of type "super
```

以下是将掩蔽策略附加到不存在的 SUPER 路径的示例。默认情况下，Amazon Redshift 将路径解析为 NULL。

```
ATTACH MASKING POLICY mask_first_name
ON employees(col_person.not_exists)
TO PUBLIC;

SELECT col_person FROM employees LIMIT 1;

-- This result is formatted for ease of reading.
         col_person
-----------------------------------
{
    "name": {
        "first": "JOHN",
        "last": "XXXX"
    },
    "age": 20,
    "ssn": "XXX-XX-3333",
    "company": "Company Inc.",
    "not_exists": null
}
```

条件动态数据掩蔽

通过在屏蔽表达式中使用条件表达式创建掩蔽策略，可以在单元格级别掩蔽数据。例如，您可以创建一个掩蔽策略，根据该行中另一列的值对值应用不同的掩蔽。

以下是使用条件数据掩蔽来创建和附加掩蔽策略的示例，该策略会部分编辑涉及欺诈的信用卡号，同时完全隐藏所有其他信用卡号。您必须是超级用户或具有 sys:secadmin 角色才能运行此示例。

```
--Create an analyst role.
CREATE ROLE analyst;

--Create a credit card table. The table contains an is_fraud boolean column,
--which is TRUE if the credit card number in that row was involved in a fraudulent
 transaction.
CREATE TABLE credit_cards (id INT, is_fraud BOOLEAN, credit_card_number VARCHAR(16));

--Create a function that partially redacts credit card numbers.
CREATE FUNCTION REDACT_CREDIT_CARD (credit_card VARCHAR(16))
RETURNS VARCHAR(16) IMMUTABLE
AS $$
    import re
    regexp = re.compile("^([0-9]{6})[0-9]{5,6}([0-9]{4})")

    match = regexp.search(credit_card)
    if match != None:
        first = match.group(1)
        last = match.group(2)
    else:
        first = "000000"
        last = "0000"

    return "{}XXXXX{}".format(first, last)
$$ LANGUAGE plpythonu;

--Create a masking policy that partially redacts credit card numbers if the is_fraud
 value for that row is TRUE,
--and otherwise blanks out the credit card number completely.
CREATE MASKING POLICY card_number_conditional_mask
    WITH (fraudulent BOOLEAN, pan varchar(16))
    USING (CASE WHEN fraudulent THEN REDACT_CREDIT_CARD(pan)
                ELSE Null
           END);

--Attach the masking policy to the credit_cards/analyst table/role pair.
ATTACH MASKING POLICY card_number_conditional_mask ON credit_cards (credit_card_number)
 USING (is_fraud, credit_card_number)
 TO ROLE analyst PRIORITY 100;
```

用于动态数据掩蔽的系统视图

超级用户、具有 sys:operator 角色的用户和具有 ACCESS SYSTEM TABLE 权限的用户可以访问以下与 DDM 相关的系统视图。

- SVV_MASKING_POLICY

  使用 SVV_MASKING_POLICY 可查看在集群或工作组上创建的所有屏蔽策略。
- SVV_ATTACHED_MASKING_POLICY

  使用 SVV_ATTACHED_MASKING_POLICY 可查看已在当前连接的数据库上附加了策略的所有关系和角色或用户。
- SYS_APPLIED_MASKING_POLICY_LOG

  使用 SYS_APPLIED_MASKING_POLICY_LOG 可跟踪屏蔽策略在引用受 DDM 保护关系的查询上的应用情况。

下面是使用系统视图可以找到的一些信息示例。

```
--Select all policies associated with specific users, as opposed to roles
SELECT policy_name,
       schema_name,
       table_name,
       grantee
FROM svv_attached_masking_policy
WHERE grantee_type = 'user';

--Select all policies attached to a specific user
SELECT policy_name,
       schema_name,
       table_name,
       grantee
FROM svv_attached_masking_policy
WHERE grantee = 'target_grantee_name'

--Select all policies attached to a given table
SELECT policy_name,
       schema_name,
       table_name,
       grantee
FROM svv_attached_masking_policy
```

```
WHERE table_name = 'target_table_name'
      AND schema_name = 'target_schema_name';

--Select the highest priority policy attachment for a given role
SELECT samp.policy_name,
       samp.priority,
       samp.grantee,
       smp.policy_expression
FROM svv_masking_policy AS smp
JOIN svv_attached_masking_policy AS samp
    ON samp.policy_name = smp.policy_name
WHERE
    samp.grantee_type = 'role' AND
    samp.policy_name = mask_get_policy_for_role_on_column(
        'target_schema_name',
        'target_table_name',
        'target_column_name',
        'target_role_name')
ORDER BY samp.priority desc
LIMIT 1;

--See which policy a specific user will see on a specific column in a given relation
SELECT samp.policy_name,
       samp.priority,
       samp.grantee,
       smp.policy_expression
FROM svv_masking_policy AS smp
JOIN svv_attached_masking_policy AS samp
    ON samp.policy_name = smp.policy_name
WHERE
    samp.grantee_type = 'role' AND
    samp.policy_name = mask_get_policy_for_user_on_column(
        'target_schema_name',
        'target_table_name',
        'target_column_name',
        'target_user_name')
ORDER BY samp.priority desc;

 --Select all policies attached to a given relation.
SELECT policy_name,
schema_name,
relation_name,
database_name
FROM sys_applied_masking_policy_log
```

```
WHERE relation_name = 'relation_name'
AND schema_name = 'schema_name';
```

限定范围权限

限定范围权限允许您向用户或角色授予对数据库或架构中某一类型的所有对象的访问权限。具有限定范围权限的用户和角色对数据库或架构中的所有当前和未来对象具有指定权限。

有关应用限定范围权限的更多信息，请参阅GRANT和REVOKE。

使用限定范围权限的注意事项

使用限定范围权限时，请注意以下事项：

- 您可以使用限定范围权限将数据库或架构范围的权限授予指定用户或角色，或者撤销这些权限。
- 您不能将限定范围权限授予用户组。
- 授予或撤销限定范围权限会更改范围中所有当前和未来对象的权限。
- 限定范围权限和对象级权限相互独立地运作。例如，在以下两种情况下，用户都将保留对表的权限。
  - 授予用户对表 schema1.table1 的 SELECT 权限和对 schema1 的 SELECT 限定范围权限。然后，用户撤消了对架构 schema1 中所有表的 SELECT 权限。用户在 schema1.table1 上保留 SELECT 权限。
  - 授予用户对表 schema1.table1 的 SELECT 权限和对 schema1 的 SELECT 限定范围权限。然后，用户撤销了对 schema1.table1 的 SELECT 权限。用户在 schema1.table1 上保留 SELECT 权限。
- 要授予或撤销限定范围权限，您必须满足以下条件之一：
  - 超级用户。
  - 拥有该权限授权选项的用户。有关授权选项的更多信息，请参阅 GRANT 中的 WITH GRANT OPTION 参数。
- 只能针对已连接数据库或从数据共享导入的数据库的对象授予或撤销限定范围权限。
- 您可以使用限定范围权限，为通过数据共享创建的数据库设置默认权限。被授予对共享数据库的限定范围权限的使用者端数据共享用户将自动获得对生产者端添加到数据共享的任何新对象的这些权限。
- 生产者可以向数据共享（预览版）授予对架构内对象的限定范围权限。

SQL 参考

主题

Amazon Redshift SQL

主题

Amazon Redshift 是围绕行业标准 SQL 构建的，新增了管理非常大的数据库并支持针对这些数据进行高性能分析和报告的功能。

> **Note**
>
> 单个 Amazon Redshift SQL 语句的最大大小为 16MB。

在领导节点上支持的 SQL 函数

一些 Amazon Redshift 查询是在计算节点上分发和运行的；而另一些查询仅在领导节点上运行。

每当查询引用用户创建的表或系统表（具有 STL 或 STV 前缀的表和具有 SVL 或 SVV 前缀的系统视图）时，领导节点就会将 SQL 分发到计算节点。仅引用目录表（驻留在领导节点上的、具有 PG 前缀的表，如 PG_TABLE_DEF）或不引用任何表的查询在领导节点上以独占方式运行。

部分 Amazon Redshift SQL 函数仅在领导节点上受支持，在计算节点上不受支持。使用领导节点函数的查询必须在领导节点上而不是计算节点上以独占方式执行，否则它将返回错误。

必须在领导节点上以独占方式运行的每个函数的文档包含一个注释，指示该函数将在引用用户定义的表或 Amazon Redshift 系统表时返回错误。有关在领导节点上以独占方式运行的函数的列表，请参阅仅领导节点函数。

示例

以下示例使用示例 TICKIT 数据库。有关示例数据库的更多信息，请转到示例数据库。

CURRENT_SCHEMA

CURRENT_SCHEMA 函数是仅适用于领导节点的函数。在此示例中，查询不会引用表，因此它将在领导节点上以独占方式运行。

```
select current_schema();

current_schema
---------------
public
```

在下一个示例中，查询引用系统目录表，因此它将在领导节点上以独占方式运行。

```
select * from pg_table_def
where schemaname = current_schema() limit 1;

 schemaname | tablename | column |   type    | encoding | distkey | sortkey | notnull
------------+-----------+--------+-----------+----------+---------+---------+---------
 public     | category  | catid  | smallint | none     | t       |       1 | t
```

在下一个示例中，查询引用驻留在计算节点上的 Amazon Redshift 系统表，因此它将返回错误。

```
select current_schema(), userid from users;

INFO:  Function "current_schema()" not supported.
ERROR:  Specified types or functions (one per INFO message) not supported on Amazon
 Redshift tables.
```

SUBSTR

SUBSTR 也是一个仅适用于领导节点的函数。在下面的示例中，由于查询没有引用表，因此在领导节点上以独占方式运行。

```
SELECT SUBSTR('amazon', 5);

+--------+
| substr |
+--------+
| on     |
+--------+
```

在下面的示例中，查询引用驻留在计算节点上的表。这会导致错误。

```
SELECT SUBSTR(catdesc, 1) FROM category LIMIT 1;

ERROR: SUBSTR() function is not supported (Hint: use SUBSTRING instead)
```

要成功运行前一个查询，请使用 SUBSTRING。

```
SELECT SUBSTRING(catdesc, 1) FROM category LIMIT 1;

+---------------------------------+
|            substring            |
+---------------------------------+
| National Basketball Association |
+---------------------------------+
```

FACTORIAL()

FACTORIAL() 是仅适用于主节点的函数。在下面的示例中，由于查询没有引用表，因此在领导节点上以独占方式运行。

```
SELECT FACTORIAL(5);

 factorial
-------------
 120
```

在下面的示例中，查询引用驻留在计算节点上的表。使用查询编辑器 v2 运行时会导致错误。

```
create table t(a int);
insert into t values (5);
select factorial(a) from t;
```

```
ERROR: Specified types or functions (one per INFO message) not supported on Redshift
 tables.
Info: Function "factorial(bigint)" not supported.
```

Amazon Redshift 和 PostgreSQL

主题

Amazon Redshift 基于 PostgreSQL。Amazon Redshift 和 PostgreSQL 之间的差别非常大，您在设计和开发数据仓库应用程序时必须注意这一点。

Amazon Redshift 是专为联机分析处理 (OLAP) 和业务智能 (BI) 应用程序设计的，这些应用程序需要针对大型数据集的复杂查询。由于它解决了迥然不同的需求，因此 Amazon Redshift 使用的专用数据存储 schema 和查询执行引擎完全不同于 PostgreSQL 实现。例如，联机事务处理 (OLTP) 应用程序通常将数据存储在行中，Amazon Redshift 将数据存储在列中，并使用专业的数据压缩编码以获得最优的内存使用和磁盘输入/输出。某些适合小型 OLTP 处理，如二级索引和高效单行数据操作运算的 PostgreSQL 功能已省略，以改进性能。

有关 Amazon Redshift 数据仓库系统架构的详细解释，请参阅系统和架构概述。

PostgreSQL 9.x 包含一些在 Amazon Redshift 中不支持的功能。此外，Amazon Redshift SQL 和 PostgreSQL 之间有一些重大差异，您必须了解。本节重点介绍了 Amazon Redshift 和 PostgreSQL 之间的差异，并提供了开发充分利用 Amazon Redshift SQL 实现的数据仓库的指南。

Amazon Redshift 及 PostgreSQL JDBC 和 ODBC

由于 Amazon Redshift 基于 PostgreSQL，因此我们之前建议使用 JDBC4 Postgresql 驱动程序版本 8.4.703 和 psqlODBC 版本 9.x 的驱动程序。如果您当前使用的是这两种驱动程序，建议您接下来转移至新的特定于 Amazon Redshift 的驱动程序。有关驱动程序和配置连接的更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 的 JDBC 和 ODBC 驱动程序。

为了避免在使用 JDBC 检索大型数据集时出现客户端内存不足错误，您可通过设置 JDBC 提取大小参数来使您的客户端能够成批提取数据。有关更多信息，请参阅 设置 JDBC 提取大小参数。

Amazon Redshift 无法识别 JDBC maxRows 参数。请改为指定 LIMIT 子句来限制结果集。您还可使用 OFFSET 子句跳至结果集中的特定起点。

以不同方式实施的功能

许多 Amazon Redshift SQL 语言元素都具有不同的性能特征和使用语法以及语义，并且等效的 PostgreSQL 实施有很大不同。

> **Important**
>
> 请勿假设 Amazon Redshift 和 PostgreSQL 共同具有的元素的语义是相同的。确保查阅《Amazon Redshift 开发人员指南》SQL 命令以了解常有的细小差异。

一个具体的示例是 VACUUM 命令，该命令用于清理和重新组织表。VACUUM 具有不同的功能，并且使用的参数组不同于 PostgreSQL 版本。有关在 Amazon Redshift 中使用 VACUUM 的更多信息，请参阅对表执行 vacuum 操作。

通常，数据库管理功能和工具也不相同。例如，Amazon Redshift 维护一组提供有关系统运行方式的信息的系统表和视图。参阅 系统表和视图 了解更多信息。

以下列表包含 Amazon Redshift 中的实现方式不同的 SQL 功能的一些示例。

- CREATE TABLE

  Amazon Redshift 不支持表空间、表分区、继承和某些约束。CREATE TABLE 的 Amazon Redshift 实现使您能够为表定义排序和分配算法以优化并行处理。

  Amazon Redshift Spectrum 支持使用 CREATE EXTERNAL TABLE 命令的表分区。

- ALTER TABLE

  只支持 ALTER COLUMN 操作的子集。

  ADD COLUMN 支持在每个 ALTER TABLE 语句中仅添加一列。

- COPY

  Amazon Redshift COPY 命令专门用于支持从 Amazon S3 桶和 Amazon DynamoDB 表加载数据以及方便自动压缩。有关详细信息，请参阅加载数据部分和 COPY 命令参考。

- VACUUM

VACUUM 的参数完全不同。例如，PostgreSQL 中的默认 VACUUM 操作只是回收空间并使空间可供重复使用；但是，Amazon Redshift 中的默认 VACUUM 操作为 VACUUM FULL，可回收磁盘空间并对所有行重新排序。

- 在比较字符串值时，会忽略 VARCHAR 值的尾部空格。有关更多信息，请参阅 尾部空格的意义。

不支持的 PostgreSQL 功能

这些功能在 Amazon Redshift 中不受支持。

> **Important**
>
> 请勿假设 Amazon Redshift 和 PostgreSQL 共同具有的元素的语义是相同的。确保查阅《Amazon Redshift 开发人员指南》SQL 命令以了解常有的细小差异。

- 不支持查询工具 RSQL。支持 Amazon Redshift RSQL 客户端。
- 表分区（范围和列表分区）
- 表空间
- 约束
  - 唯一
  - 外键
  - 主键
  - 检查约束
  - 排他性约束

  允许唯一键、主键和外键约束，但它们仅供参考。这些约束不由系统强制实施，而是由查询规划器使用。
- 数据库角色
- 继承
- PostgreSQL 系统列

  Amazon Redshift SQL 不会隐式定义系统列。但是，以下 PostgreSQL 系统列名称无法用作用户定义列的名称：`oid`、`tableoid`、`xmin`、`cmin`、`xmax`、`cmax` 和 `ctid`。有关更多信息，请参阅 https://www.postgresql.org/docs/8.0/static/ddl-system-columns.html。

- 索引
- 窗口函数中的 NULLS 子句
- 排序规则

  Amazon Redshift 不支持区域设置特定的或用户定义的排序规则序列。请参阅 排序规则序列。
- 值表达式
  - 下标表达式
  - 数组构造函数
  - 行构造函数
- 触发
- 外部数据管理 (SQL/MED)
- 表函数
- 用作常量表的 VALUES 列表
- Sequences 属性
- 全文搜索

不支持的 PostgreSQL 数据类型

通常，如果查询尝试使用不受支持的数据类型，包括显式或隐式强制转换，则将返回错误。但是，使用不支持的数据类型的某些查询将仅在领导节点运行，而不在计算节点上运行。请参阅 在领导节点上支持的 SQL 函数。

有关支持的数据类型的列表，请参阅数据类型。

这些 PostgreSQL 数据类型在 Amazon Redshift 中不受支持。

- 数组
- BIT、BIT VARYING
- BYTEA
- 复合类型
- 日期/时间类型
- 枚举类型
- 几何类型

- HSTORE
- JSON
- 网络地址类型
- 数字类型
  - SERIAL、BIGSERIAL、SMALLSERIAL
  - MONEY
- 对象标识符类型
- 伪类型
- 范围类型
- 特殊字符类型
  - “char” – 单字节内部类型（其中名为 char 的数据类型用引号括起来）。
  - name – 对象名称的内部类型。

  有关这些类型的更多信息，请参阅 PostgreSQL 文档中的特殊字符类型。
- 文本搜索类型
- TXID_SNAPSHOT
- UUID
- XML

不支持的 PostgreSQL 函数

很多未排除的函数具有不同的语义或用法。例如，一些受支持的函数将仅在领导节点上运行。此外，一些不受支持的函数在领导节点上运行时不会返回错误。这些函数在某些情况下不返回错误的情况不应视为表示这些函数受 Amazon Redshift 支持。

> **Important**
>
> 请勿假设 Amazon Redshift 和 PostgreSQL 共同具有的元素的语义是相同的。确保查阅《Amazon Redshift 数据库开发人员指南》SQL 命令以了解常有的细小差异。

有关更多信息，请参阅 在领导节点上支持的 SQL 函数。

这些 PostgreSQL 函数在 Amazon Redshift 中不受支持。

- 访问权限查询函数
- 劝告锁函数
- 聚合函数
  - STRING_AGG()
  - ARRAY_AGG()
  - EVERY()
  - XML_AGG()
  - CORR()
  - COVAR_POP()
  - COVAR_SAMP()
  - REGR_AVGX()、REGR_AVGY()
  - REGR_COUNT()
  - REGR_INTERCEPT()
  - REGR_R2()
  - REGR_SLOPE()
  - REGR_SXX()、REGR_SXY()、REGR_SYY()
- 数组函数和运算符
- 备份控制函数
- 注释信息函数
- 数据库对象位置函数
- 数据库对象大小函数
- 日期/时间函数和运算符
  - CLOCK_TIMESTAMP()
  - JUSTIFY_DAYS()、JUSTIFY_HOURS()、JUSTIFY_INTERVAL()
  - PG_SLEEP()
  - TRANSACTION_TIMESTAMP()
- ENUM 支持函数
- 几何函数和运算符
- 一般文件访问函数
- IS DISTINCT FROM

- 网络地址函数和运算符
- 数学函数
  - DIV()
  - SETSEED()
  - WIDTH_BUCKET()
- 设置返回函数
  - GENERATE_SERIES()
  - GENERATE_SUBSCRIPTS()
- 范围函数和运算符
- 恢复控制函数
- 恢复信息函数
- ROLLBACK TO SAVEPOINT 函数
- Schema 可见性查询函数
- 服务器信号函数
- 快照同步函数
- 顺序操作函数
- 字符串函数
  - BIT_LENGTH()
  - OVERLAY()
  - CONVERT()、CONVERT_FROM()、CONVERT_TO()
  - ENCODE()
  - FORMAT()
  - QUOTE_NULLABLE()
  - REGEXP_MATCHES()
  - REGEXP_SPLIT_TO_ARRAY()
  - REGEXP_SPLIT_TO_TABLE()
- 系统目录信息函数
- 系统信息函数
  - CURRENT_CATALOG CURRENT_QUERY()
  - INET_CLIENT_ADDR()

  - INET_CLIENT_PORT()
  - INET_SERVER_ADDR() INET_SERVER_PORT()
  - PG_CONF_LOAD_TIME()
  - PG_IS_OTHER_TEMP_SCHEMA()
  - PG_LISTENING_CHANNELS()
  - PG_MY_TEMP_SCHEMA()
  - PG_POSTMASTER_START_TIME()
  - PG_TRIGGER_DEPTH()
  - SHOW VERSION()
- 文本搜索函数和运算符
- 事务 ID 和快照函数
- 触发器函数
- XML 函数

使用 SQL

主题

SQL 语言由您用于处理数据库和数据库对象的命令和函数组成。该语言还会强制实施有关数据类型、表达式和文本使用的规则。

SQL 参考惯例

此部分介绍用于为 SQL 参考部分中介绍的 SQL 表达式、命令和函数编写语法的约定。

字符	描述
CAPS	采用大写字母的字样为关键字。

字符	描述
[ ]	方括号表示可选参数。方括号中的多个参数表示您可选择任意数量的参数。此外，不同行的括号中的参数表示 Amazon Redshift 分析程序需要按照参数在语法中列出的顺序获取参数。有关示例，请参阅 SELECT。
{ }	大括号表示您需要选择大括号内的参数之一。
|	管道表示您可以在不同参数之间选择。
斜体	斜体字样表示占位符。必须插入适当的值以替换斜体字样。
. . .	省略号表示可以重复前面的元素。
'	单引号中的字样表示必须键入引号。

基本元素

主题

- 名称和标识符
- 文本
- null
- 数据类型
- 排序规则序列

本部分介绍了用于处理数据库对象名称、文本、null 和数据类型的规则。

名称和标识符

名称用于标识数据库对象，包括表和列以及用户和密码。名称 和标识符 这两个术语可互换使用。有两种类型的标识符：标准标识符以及带引号的标识符或分隔标识符。标识符必须仅包含 UTF-8 可打印字符。标准标识符和分隔标识符中的 ASCII 字母是不区分大小写的，并且会在数据库中转换为小写。默认情况下，查询结果中的列名称以小写形式返回。要以大写形式返回列名称，请将 describe_field_name_in_uppercase 配置参数设置为 **`true`**。

标准标识符

标准 SQL 标识符遵循一组规则并且必须：

- 以 ASCII 单字节字母字符或下划线字符开头，或者以 UTF-8 多字节字符（长度为 2 到 4 字节）开头。
- 后续字符可以是 ASCII 单字节字母数字字符、下划线或美元符号，或者 UTF-8 多字节字符 (两到四字节长)。
- 长度介于 1 到 127 个字节之间，不包含分隔标识符的引号。
- 不包含引号和空格。
- 不是保留的 SQL 关键字。

分隔标识符

分隔标识符（也称为“带引号的标识符”）以双引号 (") 开头和结尾。如果您使用分隔标识符，则必须在每次引用对象时使用双引号。标识符可包含双引号以外的任何标准 UTF-8 可打印字符。因此，您可创建包含非法字符的列或表名称，如空格或百分比符号。

分隔标识符中的 ASCII 字母是不区分大小写的，并且会转换为小写。要在字符串中使用双引号，您必须在字符串前加上另一个双引号字符。

区分大小写的标识符

区分大小写的标识符（也称为混合大小写标识符）可以包含大写字母和小写字母。要使用区分大小写的标识符，可您可以将配置 `enable_case_sensitive_identifier` 设置为 `true`。您可以为集群或会话设置此配置。有关更多信息，请参阅《Amazon Redshift 管理指南》中的默认参数值和enable_case_sensitive_identifier。

系统列名称

以下 PostgreSQL 系统列名称无法用作用户定义列中的列名称。有关更多信息，请参阅 https://www.postgresql.org/docs/8.0/static/ddl-system-columns.html。

- `oid`
- `tableoid`
- `xmin`
- `cmin`
- `xmax`
- `cmax`
- `ctid`

示例

此表显示了分隔标识符的示例、生成的输出和讨论：

语法	结果	讨论
"group"	group	GROUP 是保留字，因此在标识符中使用它需要双引号。
"""WHERE"""	"where"	WHERE 也是一个保留字。要在字符串中包含引号，请使用其他双引号字符对每个双引号字符进行转义。
"This name"	this name	需要使用双引号来保留空格。
"This ""IS IT"""	this "is it"	围绕 IS IT 的每个引号必须在前面添加额外的引号才能成为名称的一部分。

创建一个名为 group 的表，其中一列的名称为 this "is it"：

```
create table "group" (
"This ""IS IT""" char(10));
```

下面的查询返回相同的结果：

```
select "This ""IS IT"""
from "group";

this "is it"
--------------
(0 rows)
```

```
select "this ""is it"""
from "group";

this "is it"
--------------
(0 rows)
```

以下完全限定的 `table.column` 语法也返回相同的结果：

```
select "group"."this ""is it"""
```

```
from "group";

this "is it"
--------------
(0 rows)
```

以下 CREATE TABLE 命令创建一个表，其中的一个列名包含斜杠：

```
create table if not exists city_slash_id(
                  "city/id" integer not null,
                  state char(2) not null);
```

文本

文本或常量是固定数据值，由一系列字符或数字常量组成。Amazon Redshift 支持多种文本，包括：

- 整数、小数和浮点数的数字文本。有关更多信息，请参阅 整数和浮点文本。
- 字符文本，也称为“字符串”或“字符常量”
- 与日期时间数据类型一起使用的日期时间和间隔文本。有关更多信息，请参阅日期、时间和时间戳文本 和间隔数据类型和文字。

null

如果行中有某个列缺失、未知或不适用，则该列为 null 值，或者说该列包含 null。null 可出现在不受主键或 NOT NULL 约束限制的任何数据类型的字段中。null 不等于零值或空字符串。

任何包含 null 的算术表达式的计算结果始终为 null。除串联之外的所有运算符在给定 null 参数或操作数的情况下将返回 null。

要测试是否为 null，请使用比较条件 IS NULL 和 IS NOT NULL。由于 null 表示缺少数据，因此 null 不等于任何值或其他 null。

数据类型

主题

- 多字节字符
- 数字类型
- 字符类型
- 日期时间类型

Amazon Redshift 存储或检索的每个值都具有包含一组固定关联属性的数据类型。数据类型是在创建表时声明的。数据类型约束了列或参数可包含的一组值。

下表列出了您可在 Amazon Redshift 表中使用的数据类型。

数据类型	别名	描述
SMALLINT	INT2	有符号的二字节整数
INTEGER	INT、INT4	有符号的四字节整数
BIGINT	INT8	有符号的八字节整数
DECIMAL	NUMERIC	可选精度的精确数字
REAL	FLOAT4	单精度浮点数
DOUBLE PRECISION	FLOAT8、FLOAT	双精度浮点数
CHAR	CHARACTER、NCHAR、BPCHAR	固定长度字符串
VARCHAR	CHARACTER VARYING、NVARCHAR、TEXT	具有用户定义的限制的可变长度字符串，
DATE		日历日期（年、月、日）
TIME	TIME WITHOUT TIME ZONE	Time of day
TIMETZ	带时区的时间	Time of day with time zone
TIMESTAMP	TIMESTAMP WITHOUT TIME ZONE	日期和时间（没有时区）

数据类型	别名	描述
TIMESTAMPTZ	TIMESTAMP（有时区）	日期和时间（有时区）
INTERVAL YEAR TO MONTH		按年到月顺序排列的持续时间
INTERVAL DAY TO SECOND		按日到秒顺序排列的持续时间
BOOLEAN	BOOL	逻辑布尔值 (true/false)
HLLSKETCH		用于 HyperLogLog 草图的类型。
SUPER		一种超集数据类型，包含 Amazon Redshift 的所有标量类型，包括 ARRAY 和 STRUTS 等复杂类型。
VARBYTE	VARBINARY，二进制可变	长度可变的二进制值
GEOMETRY		空间数据
GEOGRAPHY		空间数据

> **Note**
>
> 有关不支持的数据类型的信息，例如“char”（注意 char 用引号括起来），请参阅不支持的 PostgreSQL 数据类型。

多字节字符

VARCHAR 数据类型支持多达 4 个字节的 UTF-8 多字节字符。不支持 5 个字节或更长的字符。要计算包含多字节字符的 VARCHAR 列的大小，请用字符数乘以每个字符的字节数。例如，如果一个字符串包含四个中文字符，并且每个字符的长度为三个字节，则您需要一个 VARCHAR(12) 列才能存储该字符串。

VARCHAR 数据类型不支持下列无效的 UTF-8 代码点：

`0xD800 – 0xDFFF`（字节序列：`ED A0 80 – ED BF BF`）

CHAR 数据类型不支持多字节字符。

数字类型

主题

数字数据类型包括整数、小数和浮点数。

整数类型

使用 SMALLINT、INTEGER 和 BIGINT 数据类型存储各种范围的整数。您无法存储每种类型所允许范围之外的值。

名称	存储	范围
SMALLINT 或 INT2	2 字节	-32768 到 +32767
INTEGER、INT 或 INT4	4 字节	-2147483648 到 +2147483647
BIGINT 或 INT8	8 字节	-9223372036854775808 到 9223372036854775807

DECIMAL 或 NUMERIC 类型

使用 DECIMAL 或 NUMERIC 数据类型存储具有用户定义的精度 的值。DECIMAL 和 NUMERIC 关键字是可互换的。在本文档中，小数 是此数据类型的首选术语。术语数字 一般用于指整数、小数和浮点数据类型。

存储	范围
可变，对于未压缩的 DECIMAL 类型可以多达 128 位。	128 位有符号整数，精度位数可以多达 38 位。

通过指定精度 和小数位数 来定义表中的 DECIMAL 列：

```
decimal(precision, scale)
```

精度

整个值中有效位的总数：小数点两边的位数。例如，数字 48.2891 的精度为 6，小数位数为 4。如果未指定，默认精度为 18。最大精度为 38。

如果输入值中的小数点左侧的位数超出了列的精度减去其小数位数的差值，则此值无法复制到列中（或无法插入或更新）。此规则适用于列定义范围之外的任何值。例如，numeric(5,2) 列所允许的值范围为 -999.99 到 999.99。

小数位数

值的小数部分中小数点右侧的小数位数。整数的小数位数为零。在列规范中，小数位数值必须小于或等于精度值。如果未指定，默认小数位数为 0。最大小数位数为 37。

如果加载到表中的输入值的小数位数大于列的小数位数，则该值将四舍五入到指定小数位数。例如，SALES 表中的 PRICEPAID 列为 DECIMAL(8,2) 列。如果将 DECIMAL(8,4) 值插入到 PRICEPAID 列中，则该值将四舍五入到小数位数 2。

```
insert into sales
values (0, 8, 1, 1, 2000, 14, 5, 4323.8951, 11.00, null);

select pricepaid, salesid from sales where salesid=0;

pricepaid | salesid
-----------+---------
4323.90 |       0
(1 row)
```

但是，对于显示强制转换表中选定的值而得出的结果，不会进行四舍五入。

> **Note**
>
> 您可插入到 DECIMAL(19,0) 列中的最大正值为 9223372036854775807 (2^{63} -1)。最大负值为 -9223372036854775807。例如，尝试插入值 9999999999999999999（19 个 9）将导致溢出错误。无论小数点的位置如何，Amazon Redshift 可表示为 DECIMAL 数的最大字符串为 9223372036854775807。例如，您可加载到 DECIMAL(19,18) 列中的最大值为 9.223372036854775807。
> 这些规则是因为有效精度位为 19 位或更少的 DECIMAL 值在内部存储为 8 字节整数，而有效精度位为 20 到 38 位的 DECIMAL 值存储为 16 字节整数。

有关使用 128 位 DECIMAL 或 NUMERIC 列的说明

请勿为 DECIMAL 列随意分配最高精度，除非您确定您的应用程序需要此精度。128 位值使用的磁盘空间是 64 位值的两倍，并且可能会降低查询执行速度。

浮点类型

使用 REAL 和 DOUBLE PRECISION 数据类型可存储具有可变精度 的数值。这些类型是不精确的 类型，意味着一些值是作为估计值存储的，因此存储和返回某个特定值可能导致细微的差异。如果您需要精确的存储和计算（如货币金额），请使用 DECIMAL 数据类型。

根据 IEEE 二进制浮点运算标准 754，REAL 表示单精度浮点数格式。它的精度约为 6 位，范围在 1E-37 到 1E+37 之间。您也可以将此数据类型指定为 FLOAT4。

根据 IEEE 二进制浮点运算标准 754，DOUBLE PRECISION 表示双精度浮点数格式。它的精度约为 15 位，范围在 1E-307 到 1E+308 之间。您也可以将此数据类型指定为 FLOAT 或 FLOAT8。

除普通数值外，浮点型还有几个特殊值。在 SQL 中使用这些值时，请在这些值周围使用单引号：

- `NaN` – 非数字
- `Infinity` – 无穷大
- `-Infinity` – 负无穷大

例如，要在表 `customer_activity` 的列 `day_charge` 中插入非数字，请运行以下 SQL：

```
insert into customer_activity(day_charge) values('NaN');
```

数值计算

在此上下文中，计算 指二进制数学运算：加、减、乘、除。此部分介绍这些运算的预期返回类型，以及在涉及 DECIMAL 数据类型时应用于确定精度和小数位数的特定公式。

当在查询处理期间计算数值时，您可能会遇到无法计算和查询返回数字溢出错误的情况。您还可能会遇到计算值的小数位数发生变化或出乎意料的情况。对于一些运算，您可使用显式强制转换（类型提升）或 Amazon Redshift 配置参数来解决这些问题。

有关类似使用 SQL 函数的计算的结果的信息，请参阅聚合函数。

计算的返回类型

提供 Amazon Redshift 中支持的一组数字数据类型，下表中显示了加、减、乘、除运算的预期返回类型。表左侧第一列表示计算中的第一个操作数，顶部行表示第二个操作数。

	INT2	INT4	INT8	DECIMAL	FLOAT4	FLOAT8
INT2	INT2	INT4	INT8	DECIMAL	FLOAT8	FLOAT8
INT4	INT4	INT4	INT8	DECIMAL	FLOAT8	FLOAT8
INT8	INT8	INT8	INT8	DECIMAL	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT4	FLOAT8
FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8

计算的 DECIMAL 结果的精度和小数位数

下表汇总了在数学运算返回 DECIMAL 结果时用于计算生成的精度和小数位数的规则。在此表中，p1 和 s1 分别表示计算中第一个操作数的精度和小数位数，p2 和 s2 分别表示第二个操作数的精度和小数位数。（不管这些计算如何，最大的结果精度为 38，最大的结果小数位数为 38）。

运算	结果精度和小数位数
+ 或者 -	小数位数 = max(s1,s2)

运算	结果精度和小数位数
	精度 = max(p1-s1,p2-s2)+1+scale
*	小数位数 = s1+s2 精度 = p1+p2+1
/	小数位数 = max(4,s1+p2-s2+1) 精度 = p1-s1+ s2+scale

例如，SALES 表中的 PRICEPAID 和 COMMISSION 列均为 DECIMAL(8,2) 列。如果您用 PRICEPAID 除以 COMMISSION（或者反过来），采用的公式如下所示：

```
Precision = 8-2 + 2 + max(4,2+8-2+1)
= 6 + 2 + 9 = 17

Scale = max(4,2+8-2+1) = 9

Result = DECIMAL(17,9)
```

以下计算是使用集合运算符（如 UNION、INTERSECT 和 EXCEPT）或 COALESCE 和 DECODE 等函数计算对 DECIMAL 值执行的运算的最终精度和小数位数的一般规则：

```
Scale = max(s1,s2)
Precision = min(max(p1-s1,p2-s2)+scale,19)
```

例如，具有一个 DECIMAL(7,2) 列的 DEC1 表将与具有一个 DECIMAL(15,3) 列的 DEC2 表联接以创建 DEC3 表。DEC3 的 schema 表明该列变成了 NUMERIC(15,3) 列。

```
create table dec3 as select * from dec1 union select * from dec2;
```

结果

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'dec3';

column |     type      | encoding | distkey | sortkey
```

```
-------+---------------+----------+---------+---------
c1     | numeric(15,3) | none     | f       | 0
```

在上例中，采用的公式如下所示：

```
Precision = min(max(7-2,15-3) + max(2,3), 19)
= 12 + 3 = 15

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

有关除法运算的说明

对于除法运算，被零除条件将返回错误。

在计算精度和小数位数之后，将应用小数位数最多为 100 的限制。如果计算所得的结果小数位数大于 100，则除法运算结果的范围如下所示：

- 精度 = `precision - (scale - max_scale)`
- 小数位数 = `max_scale`

如果计算所得的精度大于最大精度 (38)，则精度将减少为 38，小数位数的结果将介于以下范围：`max((38 + scale - precision), min(4, 100))`

溢出条件

将检查所有数值计算是否存在溢出情况。精度为 19 或 19 以下的 DECIMAL 数据存储为 64 位整数。精度大于 19 的 DECIMAL 数据存储为 128 位整数。所有 DECIMAL 值的最大精度为 38，最大小数位数为 37。当值超出这些限制时将出现溢出错误，中间结果集和最终结果集都存在这种情况：

- 当特定数据值不符合强制转换函数指定的请求精度或小数位数时，显式强制转换将生成运行时溢出错误。例如，您无法强制转换 SALES 表中 PRICEPAID 列（DECIMAL(8,2) 列）的所有值并返回 DECIMAL(7,3) 结果：

  ```
  select pricepaid::decimal(7,3) from sales;
  ERROR:  Numeric data overflow (result precision)
  ```

  此错误出现的原因是 PRICEPAID 列中的一些 较大的值无法强制转换。

- 乘法运算的乘积结果中的小数位数为所有操作数的小数位数之和。例如，如果两个操作数的小数位数都为 4，则结果的小数位数为 8，小数点左侧仅剩下 10 位。因此，在具有有效小数位数的两个大数相乘时，遇到溢出的情况相对容易一些。

  以下代码会导致溢出错误：

```
SELECT CAST(1 AS DECIMAL(38, 20)) * CAST(10 AS DECIMAL(38, 20));
ERROR: 128 bit numeric data overflow (multiplication)
```

  可以通过使用除法而不是乘法来解决溢出错误。使用以下示例除以 1 与原始除数相除的商。

```
SELECT CAST(1 AS DECIMAL(38, 20)) / (1 / CAST(10 AS DECIMAL(38, 20)));
+-----------+
| ?column? |
+-----------+
| 10       |
+-----------+
```

INTEGER 和 DECIMAL 类型的数值计算

如果计算中的一个操作数具有 INTEGER 数据类型且另一个操作数为 DECIMAL，则 INTEGER 操作数将隐式强制转换为 DECIMAL：

- INT2 (SMALLINT) 将强制转换为 DECIMAL(5,0)
- INT4 (INTEGER) 将强制转换为 DECIMAL(10,0)
- INT8 (BIGINT) 将强制转换为 DECIMAL(19,0)

例如，如果将 SALES.COMMISSION（DECIMAL(8,2) 列）和 SALES.QTYSOLD（SMALLINT 列）相乘，则此计算将强制转换为：

```
DECIMAL(8,2) * DECIMAL(5,0)
```

整数和浮点文本

表示数字的文本或常量可以是整数或浮点。

整数文本

整数常量是一系列 0-9 的数字，并且数字前带有可选的正号 (+) 或负号 (-)。

语法

```
[ + | - ] digit ...
```

示例

有效整数包括以下值：

```
23
-555
+17
```

浮点文本

浮点文本（也称为“小数、数字或分数文本”）是一系列可包含小数点和指数标记 (e)（可选）的数字。

语法

```
[ + | - ] digit ... [ . ] [ digit ...]
[ e | E [ + | - ] digit ... ]
```

参数

e | E

e 或 E 表示使用科学记数法指定的数字。

示例

有效的浮点文本包括以下值：

```
3.14159
-37.
2.0e19
-2E-19
```

数字类型的示例

CREATE TABLE 语句

以下 CREATE TABLE 语句演示不同数字数据类型的声明：

```
create table film (
film_id integer,
language_id smallint,
original_language_id smallint,
rental_duration smallint default 3,
rental_rate numeric(4,2) default 4.99,
length smallint,
replacement_cost real default 25.00);
```

尝试插入超出范围的整数

以下示例尝试将值 33000 插入到 SMALLINT 列中。

```
insert into film(language_id) values(33000);
```

SMALLINT 的范围为 -32768 至 +32767，因此 Amazon Redshift 将返回错误。

```
An error occurred when executing the SQL command:
insert into film(language_id) values(33000)

ERROR: smallint out of range [SQL State=22003]
```

将小数值插入到整数列中

以下示例将小数值插入到 INT 列中。

```
insert into film(language_id) values(1.5);
```

插入此值，但会将其四舍五入为整数值 2。

插入由于小数位数四舍五入而成功的小数

以下示例将具有更高精度的小数值插入列中。

```
insert into film(rental_rate) values(35.512);
```

在此示例中，值 35.51 将插入列中。

尝试插入超出范围的小数值

在此示例中，值 350.10 超出范围。DECIMAL 列中值的位数等于此列的精度减去其小数位数（对于 RENTAL_RATE 列为 4 减去 2）。换言之，DECIMAL(4,2) 列所允许的范围为 -99.99 到 99.99。

```
insert into film(rental_rate) values (350.10);
ERROR:  numeric field overflow
DETAIL:  The absolute value is greater than or equal to 10^2 for field with precision
 4, scale 2.
```

将精度可变的值插入到 REAL 列中

以下示例将精度可变的值插入到 REAL 列中。

```
insert into film(replacement_cost) values(1999999.99);

insert into film(replacement_cost) values(1999.99);

select replacement_cost from film;

+------------------+
| replacement_cost |
+------------------+
| 2000000          |
| 1999.99          |
+------------------+
```

值 1999999.99 将转换为 2000000 以满足 REAL 列的精度要求。值 1999.99 将按原样加载。

字符类型

主题

- 存储和范围
- CHAR 或 CHARACTER
- VARCHAR 或 CHARACTER VARYING
- NCHAR 和 NVARCHAR 类型
- TEXT 和 BPCHAR 类型
- 尾部空格的意义
- 字符类型的示例

字符数据类型包括 CHAR（字符）和 VARCHAR（字符变体）。

存储和范围

CHAR 和 VARCHAR 数据类型是按照字节而不是字符来定义的。CHAR 列只能包含单字节字符，因此 CHAR(10) 列可包含最大长度为 10 字节的字符串。VARCHAR 可包含多字节字符，并且每个字符最多可以有 4 个字节。例如，VARCHAR(12) 列可包含 12 个单字节字符、6 个双字节字符、4 个三字节字符或 3 个四字节字符。

名称	存储	范围（列宽度）
CHAR、CHARACTER 或 NCHAR	字符串的长度，包括尾部空格（如有）	4096 字节
VARCHAR、CHARACTER VARYING 或 NVARCHAR	4 字节 + 字符的总字节，其中每个字符可为 1 至 4 个字节。	65535 字节 (64K -1)
BPCHAR	已转换为固定长度 CHAR(256)。	256 字节
TEXT	已转换为 VARCHAR(256)。	260 字节

> **Note**
>
> CREATE TABLE 语法支持对字符数据类型使用 MAX 关键字。例如：
>
> ```
> create table test(col1 varchar(max));
> ```
>
> MAX 设置将列的宽度定义为 4096 字节 (CHAR) 或 65535 字节 (VARCHAR)。

CHAR 或 CHARACTER

使用 CHAR 或 CHARACTER 列存储固定长度字符串。这些字符串将使用空格填补，因此 CHAR(10) 列始终占用 10 字节的存储。

```
char(10)
```

未指定长度的 CHAR 列将生成 CHAR(1) 列。

VARCHAR 或 CHARACTER VARYING

使用 VARCHAR 或 CHARACTER VARYING 列存储具有固定限制的可变长度字符串。这些字符串不会使用空格填补，因此 VARCHAR(120) 列最多包含 120 个单字节字符、60 个双字节字符、40 个三字节字符或 30 个四字节字符。

```
varchar(120)
```

如果在 CREATE TABLE 语句中使用不带长度说明符的 VARCHAR 数据类型，则默认长度为 256。如果在表达式中使用，则输出大小将使用输入表达式确定（最大为 65535）。

NCHAR 和 NVARCHAR 类型

您可使用 NCHAR 和 NVARCHAR 类型（也称为 NATIONAL CHARACTER 和 NATIONAL CHARACTER VARYING 类型）创建列。这些类型将分别转换为 CHAR 和 VARCHAR 类型，并且使用指定字节数量存储。

未指定长度的 NCHAR 列将转换为 CHAR(1) 列。

未指定长度的 NVARCHAR 列将转换为 VARCHAR(256) 列。

TEXT 和 BPCHAR 类型

您可创建包含 TEXT 列的 Amazon Redshift 表，但此列将转换为接受最多包含 256 个字符的可变长度值的 VARCHAR(256) 列。

您可创建包含 BPCHAR（空格填补字符）类型的 Amazon Redshift 列，Amazon Redshift 会将此列转换为固定长度的 CHAR(256) 列。

尾部空格的意义

CHAR 和 VARCHAR 数据类型存储长度最多为 n 字节的字符串。尝试将更长的字符串存储到这些类型的列中将导致错误，除非额外的字符全为空格，这样字符串将截断至最大长度。如果字符串短于最大长度，CHAR 值将使用空格填补，但 VARCHAR 值将存储不带空格的字符串。

CHAR 值中的尾部空格始终无语义意义。当比较两个 CHAR 值时将忽视尾部空格，而不将其包含在 LENGTH 计算中，在将 CHAR 值转换为其他字符串类型时将删除尾部空格。

VARCHAR 和 CHAR 值中的尾部空格将在比较值时视为无语义意义。

长度计算将返回 VARCHAR 字符串的包含尾部空格在内的长度。尾部空格不会计入固定长度字符串的长度中。

字符类型的示例

CREATE TABLE 语句

以下 CREATE TABLE 语句演示 VARCHAR 和 CHAR 数据类型的使用：

```
create table address(
address_id integer,
address1 varchar(100),
address2 varchar(50),
district varchar(20),
city_name char(20),
state char(2),
postal_code char(5)
);
```

下列示例使用此表。

可变长度字符串中的尾部空格

由于 ADDRESS1 是 VARCHAR 列，插入的第二个地址中的尾部空格始终无语义意义。换言之，插入的这两个地址匹配。

```
insert into address(address1) values('9516 Magnolia Boulevard');

insert into address(address1) values('9516 Magnolia Boulevard  ');
```

```
select count(*) from address
where address1='9516 Magnolia Boulevard';

count
-------
2
(1 row)
```

如果 ADDRESS1 列是 CHAR 列并且插入了相同的值，则 COUNT(*) 查询会将字符串识别为相同并返回 2。

LENGTH 函数的结果

LENGTH 函数识别 VARCHAR 列中的尾部空格：

```
select length(address1) from address;

length
--------
23
25
(2 rows)
```

CITY_NAME 列（是 CHAR 列）中的 Augusta 的值，不管输入字符串有任何尾部空格，始终返回 7 个字符的长度。

超出列长度的值

未截断字符串来匹配列的声明宽度：

```
insert into address(city_name) values('City of South San Francisco');
ERROR: value too long for type character(20)
```

解决此问题的一个办法是将此值强制转换为列的大小：

```
insert into address(city_name)
values('City of South San Francisco'::char(20));
```

在此示例中，字符串 (City of South San Fr) 的前 20 个字符将加载到列中。

日期时间类型

主题

- 存储和范围
- DATE
- TIME
- TIMETZ
- TIMESTAMP
- TIMESTAMPTZ
- 日期时间类型的示例

- 日期、时间和时间戳文本
- 间隔数据类型和文字

日期时间数据类型包括 DATE、TIME、TIMETZ、TIMESTAMP 和 TIMESTAMPTZ。

存储和范围

名称	存储	范围	解析
DATE	4 字节	4713 BC 到 294276 AD	1 天
TIME	8 字节	00:00:00 至 24:00:00	1 微秒
TIMETZ	8 字节	00:00:00+1459 至 00:00:00+1459	1 微秒
TIMESTAMP	8 字节	4713 BC 到 294276 AD	1 微秒
TIMESTAMPTZ	8 字节	4713 BC 到 294276 AD	1 微秒

DATE

使用 DATE 数据类型存储没有时间戳的简单日历日期。

TIME

TIME 是 TIME WITHOUT TIME ZONE 的别名。

使用 TIME 数据类型存储一天中的时间。

TIME 列存储小数秒的精度最高达到 6 位的值。

预设情况下，用户表和 Amazon Redshift 系统表中，TIME 值均为协调世界时 (UTC)。

TIMETZ

TIMETZ 是 TIME WITH TIME ZONE 的别名。

使用 TIMETZ 数据类型来存储带有时区的一天中的时间。

TIMETZ 列存储小数秒的精度最高达到 6 位的值。

预设情况下，用户表和 Amazon Redshift 系统表中的 TIME 值为 (UTC)。

TIMESTAMP

TIMESTAMP 是 TIMESTAMP WITHOUT TIME ZONE 的别名。

使用 TIMESTAMP 数据类型存储包含日期和当日时间的完整时间戳值。

TIMESTAMP 列存储小数秒的精度最高达到 6 位的值。

如果将日期插入到 TIMESTAMP 列中，或插入具有部分时间戳值的日期，则值将隐式转换为完整时间戳值。此完整时间戳值具有缺少的小时、分钟和秒的默认值 (00)。输入字符串中的时区值被忽略。

预设情况下，用户表和 Amazon Redshift 系统表中的 TIMESTAMP 值为 (UTC)。

TIMESTAMPTZ

TIMESTAMPTZ 是 TIMESTAMP WITH TIME ZONE 的别名。

使用 TIMESTAMPTZ 数据类型输入包含日期、当日时间和时区的完整时间戳值。当输入值包含一个时区时，Amazon Redshift 使用时区将值转化为协 UTC 并存储 UTC 值。

要查看支持的时区名称的列表，请执行以下命令。

```
select pg_timezone_names();
```

要查看支持的时区缩写的列表，请执行以下命令。

```
select pg_timezone_abbrevs();
```

您还可以在 IANA 时区数据库中找到有关时区的当前信息。

下表提供了时区格式的示例。

格式	示例
dd mon hh:mi:ss yyyy tz	17 Dec 07:37:16 1997 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 US/Pacific
yyyy-mm-dd hh:mi:ss+/-tz	1997-12-17 07:37:16-08

格式	示例
dd.mm.yyyy hh:mi:ss tz	17.12.1997 07:37:16.00 PST

TIMESTAMPTZ 列存储小数秒的精度最高达到 6 位的值。

如果将日期插入到 TIMESTAMPTZ 列中，或插入具有部分时间戳的日期，则值将隐式转换为完整时间戳值。此完整时间戳值具有缺少的小时、分钟和秒的默认值 (00)。

TIMESTAMPTZ 值为用户表中的 UTC。

日期时间类型的示例

下面，您可以找到处理 Amazon Redshift 支持的日期时间类型的示例。

日期示例

以下示例插入具有不同格式的日期并显示输出。

```
create table datetable (start_date date, end_date date);
```

```
insert into datetable values ('2008-06-01','2008-12-31');

insert into datetable values ('Jun 1,2008','20081231');
```

```
select * from datetable order by 1;

start_date |  end_date
-----------------------
2008-06-01 | 2008-12-31
2008-06-01 | 2008-12-31
```

如果您将时间戳值插入 DATE 列，时间部分会被忽略，只会加载日期。

时间示例

以下示例插入具有不同格式的 TIME 和 TIMETZ 值并显示输出。

```
create table timetable (start_time time, end_time timetz);
```

```
insert into timetable values ('19:11:19','20:41:19 UTC');
```

```
insert into timetable values ('191119', '204119 UTC');
```

```
select * from timetable order by 1;
start_time |  end_time
------------------------
 19:11:19  | 20:41:19+00
 19:11:19  | 20:41:19+00
```

时间戳示例

如果您将日期插入到 TIMESTAMP 或 TIMESTAMPTZ 列中，时间将默认为午夜。例如，如果您插入文本 20081231，存储的值为 2008-12-31 00:00:00。

如要更改当前会话的时区，请使用 SET 命令设置 timezone 配置参数。

下面的示例插入不同格式的时间戳并显示生成的表。

```
create table tstamp(timeofday timestamp, timeofdaytz timestamptz);

insert into tstamp values('Jun 1,2008  09:59:59', 'Jun 1,2008 09:59:59 EST' );
insert into tstamp values('Dec 31,2008 18:20','Dec 31,2008 18:20');
insert into tstamp values('Jun 1,2008  09:59:59 EST', 'Jun 1,2008 09:59:59');

SELECT * FROM tstamp;

+---------------------+-----------------------+
|      timeofday      |      timeofdaytz      |
+---------------------+-----------------------+
| 2008-06-01 09:59:59 | 2008-06-01 14:59:59+00 |
| 2008-12-31 18:20:00 | 2008-12-31 18:20:00+00 |
| 2008-06-01 09:59:59 | 2008-06-01 09:59:59+00 |
+---------------------+-----------------------+
```

日期、时间和时间戳文本

以下是用于处理 Amazon Redshift 支持的日期、时间和时间戳文本的规则。

日期

下列输入日期是您可加载到 Amazon Redshift 表中的 DATE 数据类型的文本日期值的所有有效示例。默认 MDY DateStyle 模式被认为是有效的。此模式意味着在字符串中，月份值将位于日期值之前，如 1999-01-08 和 01/02/00。

> **Note**
>
> 当您将日期或时间戳文本加载到表中时，这些文本必须用引号括起来。

输入日期	完整日期
1999 年 1 月 8 日	1999 年 1 月 8 日
1999-01-08	1999 年 1 月 8 日
1/8/1999	1999 年 1 月 8 日
01/02/00	2000 年 1 月 2 日
2000-Jan-31	2000 年 1 月 31 日
Jan-31-2000	2000 年 1 月 31 日
31-Jan-2000	2000 年 1 月 31 日
20080215	2008 年 2 月 15 日
080215	2008 年 2 月 15 日
2008.366	2008 年 12 月 31 日（三位数的日期部分必须介于 001 和 366 之间）

Times

下列输入时间是您可加载到 Amazon Redshift 表中的 TIME 和 TIMETZ 数据类型的文本时间值的所有有效示例。

输入时间	描述（时间部分）
04:05:06.789	上午 4:05 过 6.789 秒
04:05:06	上午 4:05 过 6 秒

输入时间	描述（时间部分）
04:05	恰好上午 4:05
040506	上午 4:05 过 6 秒
04:05 AM	恰好上午 4:05；AM 为可选
04:05 PM	恰好下午 4:05；小时值必须小于 12
16:05	恰好下午 4:05

时间戳

下列输入时间戳是您可加载到 Amazon Redshift 表中的 TIMESTAMP 和 TIMESTAMPTZ 数据类型的文本时间值的所有有效示例。所有有效的日期文本可与下列时间文本组合。

输入时间戳（连接在一起的日期和时间）	描述（时间部分）
20080215 04:05:06.789	上午 4:05 过 6.789 秒
20080215 04:05:06	上午 4:05 过 6 秒
20080215 04:05	恰好上午 4:05
20080215 040506	上午 4:05 过 6 秒
20080215 04:05 AM	恰好上午 4:05；AM 为可选
20080215 04:05 PM	恰好下午 4:05；小时值必须小于 12
20080215 16:05	恰好下午 4:05
20080215	午夜（默认情况）

特殊日期时间值

下列特殊值可用作日期时间文本和日期函数的参数。它们需要单引号，并在查询处理期间转换为常规时间戳值。

特殊值	描述
now	计算结果为当前事务的开始时间并返回具有微秒精度的时间戳。
today	计算结果为相应的日期并返回时间部分为零的时间戳。
tomorrow	计算结果为相应的日期并返回时间部分为零的时间戳。
yesterday	计算结果为相应的日期并返回时间部分为零的时间戳。

以下示例演示 now 和 today 如何与 DATEADD 函数结合使用。

```
select dateadd(day,1,'today');

date_add
---------------------
2009-11-17 00:00:00
(1 row)

select dateadd(day,1,'now');

date_add
----------------------------
2009-11-17 10:45:32.021394
(1 row)
```

间隔数据类型和文字

您可以使用间隔数据类型以 seconds、minutes、hours、days、months、和 years 等为单位存储时间段。间隔数据类型和文字可用于日期时间计算，例如在日期和时间戳中添加间隔、对间隔求和以及从日期或时间戳中减去间隔。间隔文字可用作表中间隔数据类型列的输入值。

间隔数据类型的语法

指定间隔数据类型以存储以年和月为单位的持续时间：

```
INTERVAL year_to_month_qualifier
```

指定间隔数据类型以存储以天、小时、分钟和秒为单位的持续时间：

```
INTERVAL day_to_second_qualifier [ (fractional_precision) ]
```

间隔文字的语法

指定间隔文字以定义以年和月为单位的持续时间：

```
INTERVAL quoted-string year_to_month_qualifier
```

指定间隔文字以定义以天、小时、分钟和秒为单位的持续时间：

```
INTERVAL quoted-string day_to_second_qualifier [ (fractional_precision) ]
```

参数

引用字符串

指定正数值或负数值，以指定数量和日期时间单位作为输入字符串。如果引用字符串仅包含数字，Amazon Redshift 将根据 year_to_month_qualifier 或 day_to_second_qualifier 确定单位。例如，'23' MONTH 表示 1 year 11 months、'-2' DAY 表示 -2 days 0 hours 0 minutes 0.0 seconds，'1-2' MONTH 表示 1 year 2 months、'13 day 1 hour 1 minute 1.123 seconds' SECOND 表示 13 days 1 hour 1 minute 1.123 seconds。有关间隔输出格式的更多信息，请参阅间隔样式。

year_to_month_qualifier

指定间隔的范围。如果您使用限定词并创建时间单位小于限定词的间隔，Amazon Redshift 会截断并丢弃间隔中较小的部分。year_to_month_qualifier 的有效值为：

- YEAR
- MONTH
- YEAR TO MONTH

day_to_second_qualifier

指定间隔的范围。如果您使用限定词并创建时间单位小于限定词的间隔，Amazon Redshift 会截断并丢弃间隔中较小的部分。day_to_second_qualifier 的有效值为：

- DAY
- HOUR
- MINUTE
- SECOND
- DAY TO HOUR
- DAY TO MINUTE
- DAY TO SECOND
- HOUR TO MINUTE
- HOUR TO SECOND
- MINUTE TO SECOND

INTERVAL 文字的输出会被截断为指定的最小 INTERVAL 分量。例如，在使用 MINUTE 限定词时，Amazon Redshift 会丢弃小于 MINUTE 的时间单位。

```
select INTERVAL '1 day 1 hour 1 minute 1.123 seconds' MINUTE
```

结果值被截断为 '1 day 01:01:00'。

fractional_precision

可选参数，用于指定间隔中允许的小数位数。仅在间隔包含 SECOND 时，才应指定 fractional_precision 参数。例如，SECOND(3) 创建的间隔仅允许三个小数位，例如 1.234 秒。最大小数位数为六位。

会话配置 interval_forbid_composite_literals 确定在同时指定 YEAR TO MONTH 和 DAY TO SECOND 部分的间隔时是否返回错误。有关更多信息，请参阅 interval_forbid_composite_literals。

间隔算术

您可以将间隔值与其他日期时间值一起使用来执行算术运算。下表描述了可用的操作以及每项操作产生的数据类型。例如，在将 interval 添加 date 时，如果是 YEAR TO MONTH 间隔，则结果为 date；如果是 DAY TO SECOND 间隔，则结果为时间戳。

		Date	Timestamp	Interval	数值
Interval	-	不适用	不适用	Interval	不适用

		Date	Timestamp	Interval	数值
	+	Date	日期/时间戳	Interval	不适用
	*	不适用	不适用	不适用	Interval
	/	不适用	不适用	不适用	Interval
日期	-	数值	Interval	日期/时间戳	Date
	+	不适用	不适用	不适用	不适用
Timestamp	-	Interval	Interval	Timestamp	Timestamp
	+	不适用	不适用	不适用	不适用

间隔样式

您可以使用 SQL the section called “SET” 命令更改间隔值的输出显示格式。在 SQL 中使用间隔数据类型时，将其转换为文本可以查看预期的间隔风格，例如 `YEAR TO MONTH::text`。用于设置 `IntervalStyle` 值的可用值包括：

- `postgres` – 遵循 PostgreSQL 风格。这是默认模式。
- `postgres_verbose` – 遵循 PostgreSQL 的详细风格。
- `sql_standard` – 遵循 SQL 标准间隔文字风格。

以下命令将间隔风格设置为 `sql_standard`。

```
SET IntervalStyle to 'sql_standard';
```

postgres 输出格式

以下是 `postgres` 间隔风格的输出格式。每个数值都可以是负数。

```
'<numeric> <unit> [, <numeric> <unit> ...]'
```

```
select INTERVAL '1-2' YEAR TO MONTH::text

varchar
```

```
---------------
1 year 2 mons
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
------------------
1 day 02:03:04.5678
```

postgres_verbose 输出格式

postgres_verbose 语法与 postgres 类似，但是 postgres_verbose 输出还包含时间单位。

```
'[@] <numeric> <unit> [, <numeric> <unit> ...] [direction]'
```

```
select INTERVAL '1-2' YEAR TO MONTH::text

varchar
-----------------
@ 1 year 2 mons
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
---------------------------
@ 1 day 2 hours 3 mins 4.56 secs
```

sql_standard 输出格式

年至月间隔值的格式如下所示。在间隔之前指定负数表示间隔为负值，适用于整个间隔。

```
'[-]yy-mm'
```

日至秒值间隔的格式如下所示。

```
'[-]dd hh:mm:ss.ffffff'
```

```
SELECT INTERVAL '1-2' YEAR TO MONTH::text

varchar
```

```
-------
1-2
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
---------------
1 2:03:04.5678
```

间隔数据类型示例

以下示例演示如何将 INTERVAL 数据类型与表结合使用。

```
create table sample_intervals (y2m interval month, h2m interval hour to minute);
insert into sample_intervals values (interval '20' month, interval '2 days
 1:1:1.123456' day to second);
select y2m::text, h2m::text from sample_intervals;


      y2m      |       h2m
---------------+-----------------
 1 year 8 mons | 2 days 01:01:00
```

```
update sample_intervals set y2m = interval '2' year where y2m = interval '1-8' year to
 month;
select * from sample_intervals;

   y2m   |       h2m
---------+-----------------
 2 years | 2 days 01:01:00
```

```
delete from sample_intervals where h2m = interval '2 1:1:0' day to second;
select * from sample_intervals;

 y2m | h2m
-----+-----
```

间隔文字示例

以下示例在间隔风格设置为 postgres 的情况下运行。

以下示例演示如何创建间隔为 1 年的 INTERVAL 文字。

```
select INTERVAL '1' YEAR

intervaly2m
---------------
1 years 0 mons
```

如果您指定的引用字符串超过限定词，则剩余的时间单位将从间隔处截断。在以下示例中，13 个月的间隔变为 1 年零 1 个月，但由于使用 YEAR 限定词，剩余的 1 个月被排除在外。

```
select INTERVAL '13 months' YEAR

intervaly2m
---------------
1 years 0 mons
```

如果您使用的限定词小于间隔字符串，则会包括排除的单位。

```
select INTERVAL '13 months' MONTH

intervaly2m
---------------
1 years 1 mons
```

在间隔中指定精度会将小数位数截断为指定的精度。

```
select INTERVAL '1.234567' SECOND (3)

intervald2s
--------------------------------
0 days 0 hours 0 mins 1.235 secs
```

如果您未指定精度，Amazon Redshift 将使用最大精度 6。

```
select INTERVAL '1.23456789' SECOND

intervald2s
-----------------------------------
0 days 0 hours 0 mins 1.234567 secs
```

以下示例演示如何创建范围间隔。

```
select INTERVAL '2:2' MINUTE TO SECOND

intervald2s
------------------------------
0 days 0 hours 2 mins 2.0 secs
```

限定词决定了您要指定的单位。例如，尽管以下示例使用与前一个示例相同的“2:2”引用字符串，但 Amazon Redshift 仍会识别出它由于限定词而使用了不同的时间单位。

```
select INTERVAL '2:2' HOUR TO MINUTE

intervald2s
------------------------------
0 days 2 hours 2 mins 0.0 secs
```

还支持每个单位的缩写和复数。例如，5s、5 second、和 5 seconds 是等效间隔。支持的单位为年、月、小时、分钟和秒。

```
select INTERVAL '5s' SECOND

intervald2s
------------------------------
0 days 0 hours 0 mins 5.0 secs
```

```
select INTERVAL '5 HOURS' HOUR

intervald2s
------------------------------
0 days 5 hours 0 mins 0.0 secs
```

```
select INTERVAL '5 h' HOUR

intervald2s
------------------------------
0 days 5 hours 0 mins 0.0 secs
```

不带限定词语法的间隔文字示例

> **Note**
>
> 以下示例演示如何使用不带 YEAR TO MONTH 或 DAY TO SECOND 限定词的间隔文字。有关将推荐的间隔文字与限定词一起使用的信息，请参阅间隔数据类型和文字。

使用间隔文本标识特定时间段（如 12 hours 或 6 months）。您可在涉及日期时间表达式的条件和计算中使用这些间隔文本。

间隔文字用 INTERVAL 关键字与数量和支持的日期部分的组合来表示；例如 INTERVAL '7 days' 或 INTERVAL '59 minutes'。您可以将许多数量和单位连接在一起以形成更精确的间隔；例如：INTERVAL '7 days, 3 hours, 59 minutes'。还支持每个单位的缩写和复数；例如：5 s、5 second 和 5 seconds 是等效的间隔。

如果您未指定日期部分，则间隔值表示秒。您可指定数量值作为小数（例如：0.5 days）。

以下示例显示了具有不同间隔值的一系列计算。

以下选项向指定日期添加 1 秒。

```
select caldate + interval '1 second' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 00:00:01
(1 row)
```

以下选项向指定日期添加 1 分钟。

```
select caldate + interval '1 minute' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 00:01:00
(1 row)
```

以下选项向指定日期添加 3 小时 35 分钟。

```
select caldate + interval '3 hours, 35 minutes' as dateplus from date
```

```
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 03:35:00
(1 row)
```

以下选项向指定日期添加 52 周。

```
select caldate + interval '52 weeks' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2009-12-30 00:00:00
(1 row)
```

以下选项向指定日期添加 1 周、1 小时、1 分钟和 1 秒：

```
select caldate + interval '1w, 1h, 1m, 1s' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2009-01-07 01:01:01
(1 row)
```

以下选项向指定日期添加 12 小时（半天）。

```
select caldate + interval '0.5 days' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 12:00:00
(1 row)
```

以下计算将从 2023 年 2 月 15 日减去 4 个月，结果为 2022 年 10 月 15 日。

```
select date '2023-02-15' - interval '4 months';

?column?
---------------------
2022-10-15 00:00:00
```

以下计算将从 2023 年 3 月 31 日减去 4 个月，结果为 2022 年 11 月 30 日。计算时会考虑一个月中的天数。

```
select date '2023-03-31' - interval '4 months';

?column?
---------------------
2022-11-30 00:00:00
```

布尔值类型

使用 BOOLEAN 数据类型在单字节列中存储 true 和 false 值。下表描述了布尔值的三种可能状态以及导致这些状态的文本值。不管输入字符串如何，Boolean 列将存储和输出“t”表示 true，“f”表示 false。

州	有效的文本值	存储
True	`TRUE 't' 'true' 'y' 'yes' '1'`	1 字节
False	`FALSE 'f' 'false' 'n' 'no' '0'`	1 字节
Unknown	`NULL`	1 字节

您可以使用 IS 比较将布尔值仅作为 WHERE 子句中的谓词进行检查。不能将 IS 比较与 SELECT 列表中的布尔值一起使用。

示例

您可使用 BOOLEAN 列将每个客户的“有效/无效”状态存储在 CUSTOMER 表中。

```
create table customer(
custid int,
active_flag boolean default true);
```

```
insert into customer values(100, default);
```

```
select * from customer;
custid | active_flag
-------+-------------
   100 | t
```

如果未在 CREATE TABLE 语句中指定默认值（`true` 或 `false`），则插入默认值意味着插入 null。

在此示例中，查询从 USERS 表中选择喜欢运动而不喜欢电影院的用户：

```
select firstname, lastname, likesports, liketheatre
from users
where likesports is true and liketheatre is false
order by userid limit 10;

firstname |  lastname  | likesports | liketheatre
----------+------------+------------+-------------
Lars      | Ratliff    | t          | f
Mufutau   | Watkins    | t          | f
Scarlett  | Mayer      | t          | f
Shafira   | Glenn      | t          | f
Winifred  | Cherry     | t          | f
Chase     | Lamb       | t          | f
Liberty   | Ellison    | t          | f
Aladdin   | Haney      | t          | f
Tashya    | Michael    | t          | f
Lucian    | Montgomery | t          | f
(10 rows)
```

以下示例从 USERS 表中选择不清楚是否喜欢摇滚音乐的用户。

```
select firstname, lastname, likerock
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
----------+----------+----------
Rafael    | Taylor   |
Vladimir  | Humphrey |
Barry     | Roy      |
Tamekah   | Juarez   |
Mufutau   | Watkins  |
```

```
Naida     | Calderon |
Anika     | Huff     |
Bruce     | Beck     |
Mallory   | Farrell  |
Scarlett  | Mayer    |
(10 rows)
```

以下示例返回错误，因为它在 SELECT 列表中使用了 IS 比较。

```
select firstname, lastname, likerock is true as "check"
from users
order by userid limit 10;

[Amazon](500310) Invalid operation: Not implemented
```

以下示例成功，因为它在 SELECT 列表中使用了等于比较 (=) 而不是 IS 比较。

```
select firstname, lastname, likerock = true as "check"
from users
order by userid limit 10;

firstname | lastname  | check
----------+-----------+------
Rafael    | Taylor    |
Vladimir  | Humphrey  |
Lars      | Ratliff   | true
Barry     | Roy       |
Reagan    | Hodge     | true
Victor    | Hernandez | true
Tamekah   | Juarez    |
Colton    | Roy       | false
Mufutau   | Watkins   |
Naida     | Calderon  |
```

HLLSKETCH 类型

将 HLLSKETCH 数据类型用于 HyperLogLog 草图。Amazon Redshift 支持稀疏或密集的 HyperLogLog 草图表示。草图从稀疏开始，并在密集格式更有效地减少所使用的内存占用时切换为密集。

在以下 JSON 格式导入、导出或打印草图时，Amazon Redshift 会自动转换稀疏的 HyperLogLog 草图。

```
{"logm":15,"sparse":{"indices":[4878,9559,14523],"values":[1,2,1]}}
```

Amazon Redshift 使用 Base64 格式的字符串表示形式来表示密集的 HyperLogLog 草图。

Amazon Redshift 使用以下 Base64 格式的字符串表示形式来表示密集的 HyperLogLog 草图。

```
"ABAABA..."
```

当在原始压缩中使用时，HLLSKETCH 对象的最大大小为 24580 个字节。

SUPER 类型

使用 SUPER 数据类型将半结构化数据或文档存储为值。

半结构化数据不符合 SQL 数据库中使用的关系数据模型的刚性和表格结构。它包含引用数据中不同实体的标签。它们可以包含复杂的值，如数组、嵌套结构和其他与序列化格式（如 JSON）相关联的复杂结构。SUPER 数据类型是一组无 schema 数组和结构值，它们包含 Amazon Redshift 的所有其他标量类型。

SUPER 数据类型最高支持 16 MB 的单个 SUPER 对象的数据。有关 SUPER 数据类型的更多信息，包括在表中实现它的示例，请参阅在 Amazon Redshift 中摄取和查询半结构化数据。

只能从以下文件格式摄取大于 1MB 的 SUPER 对象：

- Parquet
- JSON
- TEXT
- CSV

SUPER 数据类型具有以下属性：

- Amazon Redshift 标量值：
  - Null
  - 布尔值
  - 一个数字，如 smallint、整数、bigint、小数或浮点（如 float4 或 float8）
  - 字符串值，如 varchar 或 char
- 一个复杂的值：

- 一个值数组，包括标量或复数
- 一个结构，也称为元组或对象，它是属性名称和值（标量或复数）的映射

这两种类型的复数值中的任何一种都包含它们自己的标量或复数值，而对规则性没有任何限制。

SUPER 数据类型以无 schema 形式支持半结构化数据的持久性。虽然分层数据模型可以更改，但旧版本的数据可以共存在于同一个 SUPER 列中。

Amazon Redshift 使用 PartiQL 来实现数组和结构的导航。Amazon Redshift 还使用 PartiQL 语法遍历 SUPER 数组。有关更多信息，请参阅导航 和取消嵌套查询。

Amazon Redshift 使用动态键入来处理无架构的 SUPER 数据，而无需在查询中使用数据类型之前声明数据类型。有关更多信息，请参阅 动态键入。

您可以将动态数据掩蔽策略应用于 SUPER 类型列路径上的 `scalar` 值。有关动态数据掩蔽的更多信息，请参阅动态数据掩蔽。有关为 SUPER 数据类型使用动态数据掩蔽的信息，请参阅对 SUPER 数据类型路径使用动态数据掩蔽。

VARBYTE 类型

使用 VARBYTE、VARBINARY 或 BINARY VARYING 列存储具有固定限制的可变长度二进制值。

```
varbyte [ (n) ]
```

最大字节数（n）范围为 1 到 16,777,216。默认值为 64,000。

下面是一些你可能想使用 VARBYTE 数据类型的示例:

- 在 VARBYTE 列上联接表。
- 创建包含 VARBYTE 列的实体化视图。支持包含 VARBYTE 列的实体化视图的增量刷新。但是，除了 COUNT、MIN、MAX 和 GROUP BY 以外，VARBYTE 列上的聚合函数不支持增量刷新。

为确保所有字节都是可打印字符，Amazon Redshift 使用十六进制格式打印 VARBYTE 值。例如，以下 SQL 将十六进制字符串 6162 转换为二进制值。尽管返回值为二进制值，但结果将以十六进制形式 6162 打印。

```
select from_hex('6162');

 from_hex
```

```
----------
 6162
```

Amazon Redshift 支持在 VARBYTE 和以下数据类型之间进行转换：

- CHAR
- VARCHAR
- SMALLINT
- INTEGER
- BIGINT

使用 CHAR 和 VARCHAR 进行转换时，将采用 UTF-8 格式。有关 UTF-8 格式的更多信息，请参阅TO_VARBYTE。从 SMALLINT、INTEGER 和 BIGINT 进行转换时，原始数据类型的字节数将保持不变。对于 SMALLINT 是 2 个字节，INTEGER 是 4 个字节，BIGINT 是 8 个字节。

以下 SQL 语句将 VARCHAR 字符串转换为 VARBYTE。尽管返回值为二进制值，但结果将以十六进制形式 616263 打印。

```
select 'abc'::varbyte;

 varbyte
---------
 616263
```

以下 SQL 语句将列中的 CHAR 值转换为 VARBYTE。此示例创建一个包含 CHAR(10) 列（c）的表，插入长度小于 10 的字符值。生成的转换使用空格字符（hex'20'）将结果填充到定义的列大小。尽管返回值是二进制值，但结果将以十六进制形式打印。

```
create table t (c char(10));
insert into t values ('aa'), ('abc');
select c::varbyte from t;
          c
----------------------
 6161202020202020202020
 6162632020202020202020
```

以下 SQL 语句将 SMALLINT 字符串转换为 VARBYTE。尽管返回值是二进制值，但结果将以十六进制形式 0005 打印，为 2 个字节或 4 个十六进制字符。

```
select 5::smallint::varbyte;

 varbyte
---------
 0005
```

以下 SQL 语句将 INTEGER 转换为 VARBYTE。尽管返回值是二进制值，但结果将以十六进制形式 00000005 打印，为 4 个字节或 8 个十六进制字符。

```
select 5::int::varbyte;

 varbyte
----------
 00000005
```

以下 SQL 语句将 BIGINT 转换为 VARBYTE。尽管返回值是二进制值，但结果将以十六进制形式 0000000000000005 打印，为 8 个字节或 16 个十六进制字符。

```
select 5::bigint::varbyte;

     varbyte
------------------
 0000000000000005
```

支持 VARBYTE 数据类型的 Amazon Redshift 功能包括：

- VARBYTE 运算符
- CONCAT
- LEN
- LENGTH 函数
- OCTET_LENGTH
- SUBSTRING 函数
- FROM_HEX
- TO_HEX
- FROM_VARBYTE
- TO_VARBYTE

- GETBIT
- 加载 VARBYTE 数据类型的列
- 卸载 VARBYTE 数据类型的列

将空间数据与 Amazon Redshift 一起使用时的限制

以下是将 VARBYTE 数据类型与 Amazon Redshift 一起使用时的限制：

- Amazon Redshift Spectrum 仅支持 Parquet 和 ORC 文件的 VARBYTE 数据类型。
- Amazon Redshift 查询编辑器和 Amazon Redshift 查询编辑器 v2 尚未完全支持 VARBYTE 数据类型。因此，在处理 VARBYTE 表达式时，请使用不同的 SQL 客户端。

  作为使用查询编辑器的解决方法，如果您的数据长度低于 64 KB 并且内容是有效的 UTF-8，那么您可以将 VARBYTE 值转换为 VARCHAR，例如：

  ```
  select to_varbyte('6162', 'hex')::varchar;
  ```
- 您不能将 VARBYTE 数据类型与 Python 或 Lambda 用户定义的函数（UDF）结合使用。
- 您不能从 VARBYTE 列创建 HLLSKETCH 列，也不能在 VARBYTE 列上使用近似去重统计。
- 大于 1 MB 的 VARBYTE 值只能从以下文件格式中摄取：
  - Parquet
  - 文本
  - 逗号分隔值（CSV）

类型兼容性和转换

您可以在下面找到关于类型转换规则和数据类型兼容性如何在 Amazon Redshift 中工作的讨论。

兼容性

在各种数据库操作期间，将会出现数据类型匹配以及文本值和常量与数据类型的匹配，包括以下情况：

- 表中的数据操控语言 (DML) 操作
- UNION、INTERSECT 和 EXCEPT 查询
- CASE 表达式
- 谓词（如 LIKE 和 IN）的计算

- 执行数据比较或提取的 SQL 函数的计算
- 数学运算符的比较

这些运算的结果取决于类型转换规则和数据类型兼容性。兼容性 意味着并非总是需要特定值和特定数据类型的一对一匹配。由于一些数据类型是兼容的，因此可进行隐式转换或强制转换（更多信息请参阅隐式转换类型）。如果数据类型不兼容，您有时可通过使用显式转换函数将值从一种数据类型转换为另一种数据类型。

一般兼容性和转换规则

请注意下列兼容性和转换规则：

- 一般来说，同属一种类型类别的数据类型（如不同的数字数据类型）是兼容的并且可隐式转换。

  例如，通过使用隐式转换，您可以将一个小数值插入整数列。小数进位为整数。或者，您可以从日期中提取一个数字值（如 2008）并将其插入到整数列中。
- 数字数据类型强制实施将在您尝试插入超出范围的值时出现的溢出条件。例如，精度为 5 的小数值无法放入到精度定义为 4 的小数列中。绝不会截断整数或小数的整数部分；但是，小数的小数部分可以视情况进行向上或向下取整。但是，对于显示强制转换表中选定的值而得出的结果，不会进行四舍五入。
- 不同类型的字符串是兼容的；包含单字节数据的 VARCHAR 列字符串和 CHAR 列字符串是兼容且可隐式转换的。包含多字节数据的 VARCHAR 字符串是不可兼容的。此外，如果字符串是适当的文本值，则您可以将字符串转换为日期、时间、时间戳或数字值；将忽略任何前导或尾部空格。反过来，您也可以将日期、时间、时间戳或数字值转换为固定长度或可变长度的字符串。

  > **Note**
  >
  > 您要强制转换为数字类型的字符串必须包含数字的字符表示形式。例如，您可将字符串 '1.0' 或 '5.9' 强制转换为小数值，但无法将字符串 'ABC' 强制转换为任何数字类型。

- 如果您将 DECIMAL 值与字符串进行比较，Amazon Redshift 会尝试将字符串转换为 DECIMAL 值。在将所有其他数值与字符串进行比较时，数值将转换为字符串。如果要强制进行相反的转换（例如，将字符串转换为正数，或者将 DECIMAL 值转换为字符串），请使用显式函数，例如 CAST。
- 若要将 64 位 DECIMAL 或 NUMERIC 值转换为更高的精度，必须使用显式转换函数（如 CAST 或 CONVERT）。
- 将 DATE 或 TIMESTAMP 转换为 TIMESTAMPTZ 时，或者将 TIME 转换为 TIMETZ 时，时区设置为当前会话时区。会话时区默认为 UTC。有关设置会话时区的更多信息，请参阅 timezone。

- 与之类似，TIMESTAMPTZ 可根据当前会话时区转化为 DATE、TIME 或 TIMESTAMP。会话时区默认为 UTC。转换后，时区信息将被删除。
- 使用当前会话时区（默认为 UTC）将代表有指定时区的时间戳的字符串转换为 TIMESTAMPTZ。使用当前会话时区（默认为 UTC）将代表有指定时区的时间的字符串转换为 TIMETZ。

隐式转换类型

有两种隐式转换类型：

- 分配中的隐式转化，如 INSERT 或 UPDATE 命令中的设置值。
- 表达式中的隐式转化，例如在 WHERE 子句中执行比较。

下表列出了在赋值或表达式中可隐式转换的数据类型。您还可使用显式转换函数执行这些转换。

源类型	目标类型
BIGINT (INT8)	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT, INT4)
	REAL (FLOAT4)
	SMALLINT (INT2)
	VARCHAR
CHAR	VARCHAR
DATE	CHAR
	VARCHAR
	TIMESTAMP

源类型	目标类型
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT (INT8)
	CHAR
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT, INT4)
	REAL (FLOAT4)
	SMALLINT (INT2)
	VARCHAR
DOUBLE PRECISION (FLOAT8)	BIGINT (INT8)
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT, INT4)
	REAL (FLOAT4)
	SMALLINT (INT2)
	VARCHAR
INTEGER (INT, INT4)	BIGINT (INT8)
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)

源类型	目标类型
	REAL (FLOAT4)
	SMALLINT (INT2)
	VARCHAR
REAL (FLOAT4)	BIGINT (INT8)
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT, INT4)
	SMALLINT (INT2)
	VARCHAR
SMALLINT (INT2)	BIGINT (INT8)
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT, INT4)
	REAL (FLOAT4)
	VARCHAR
TIMESTAMP	CHAR
	DATE
	VARCHAR

源类型	目标类型
	TIMESTAMPTZ
	TIME
TIMESTAMPTZ	CHAR
	DATE
	VARCHAR
	TIMESTAMP
	TIMETZ
TIME	VARCHAR
	TIMETZ
	INTERVAL DAY TO SECOND
TIMETZ	VARCHAR
	TIME
GEOMETRY	GEOGRAPHY
GEOGRAPHY	GEOMETRY

Note

在 TIMESTAMPTZ、TIMESTAMP、DATE、TIME、TIMETZ 或字符串之间的隐式转换使用当前会话时区。有关设置当前时间地区的信息，请参阅 timezone。
除了彼此之外，GEOMETRY 和 GEOGRAPHY 数据类型不能隐式转换为任何其它数据类型。有关更多信息，请参阅 CAST 函数。
无法将 VARBYTE 数据类型隐式转换为任何其它数据类型。有关更多信息，请参阅 CAST 函数。

对 SUPER 数据类型使用动态键入

Amazon Redshift 使用动态键入处理无 schemal SUPER 数据，无需在查询中使用数据类型之前声明数据类型。动态键入使用导航到 SUPER 数据列的结果，而无需将其显式转换为 Amazon Redshift 类型。有关为 SUPER 数据类型使用动态键入的更多信息，请参阅动态键入。

您可以将 SUPER 值转换为其他数据类型或从其他数据类型进行转换，但存在一些例外情况。有关更多信息，请参阅 限制。

排序规则序列

Amazon Redshift 不支持区域设置特定的或用户定义的排序规则序列。一般来说，如果缺少用于对数据值进行排序和比较的区域设置特定规则，则可能会影响任何上下文中的任何谓词的结果。例如，ORDER BY 表达式和函数（如 MIN、MAX 和 RANK）将根据数据的二进制 UTF8 顺序（不考虑区域设置特定字符）返回结果。

表达式

主题

- 简单表达式
- 复合表达式
- 表达式列表
- 标量子查询
- 函数表达式

表达式是一个或多个值、运算符或计算结果为值的函数的组合。表达式的数据类型一般为其组成部分的数据类型。

简单表达式

简单表达式是以下任一值：

- 常量或文本值
- 列名称或列引用
- 标量函数
- 聚合（集合）函数

- 窗口函数
- 标量子查询

简单表达式的示例包括：

```
5+12
dateid
sales.qtysold * 100
sqrt (4)
max (qtysold)
(select max (qtysold) from sales)
```

复合表达式

复合表达式是由算术运算符联接的一系列简单表达式。复合表达式中使用的简单表达式必须返回数字值。

语法

```
expression
operator
expression | (compound_expression)
```

参数

expression

计算结果为值的简单表达式。

operator

复合算术表达式可使用下列采用此优先顺序的运算符构造：

- ( )：用于控制计算顺序的圆括号
- +、-：正号和负号/运算符
- ^、|/、||/：乘方、平方根、立方根
- *、/、%：乘、除和取模运算符
- @：绝对值
- +、-：加和减

- &、|、#、~、<<、>>：逻辑与、逻辑或、逻辑非、左移位、右移位运算符
- ||：连接

(compound_expression)

复合表达式可以使用圆括号嵌套。

示例

复合表达式的示例包括以下各项。

```
('SMITH' || 'JONES')
sum(x) / y
sqrt(256) * avg(column)
rank() over (order by qtysold) / 100
(select (pricepaid - commission) from sales where dateid = 1882) * (qtysold)
```

一些函数还可嵌套在其他函数中。例如，任何标量函数都可嵌套在另一标量函数中。以下示例返回一组数字的绝对值之和：

```
sum(abs(qtysold))
```

窗口函数无法用作聚合函数或其他窗口函数的参数。以下表达式将返回错误：

```
avg(rank() over (order by qtysold))
```

窗口函数可以包含一个嵌套的聚合函数。以下表达式对值集进行求和，然后为它们排序：

```
rank() over (order by sum(qtysold))
```

表达式列表

表达式列表是表达式的组合，可出现在成员条件和比较条件（WHERE 子句）以及 GROUP BY 子句中。

语法

```
expression , expression , ... | (expression, expression, ...)
```

参数

expression

计算结果为值的简单表达式。表达式列表可包含一个或多个逗号分隔的表达式或一组或多组逗号分隔的表达式。如果存在多组表达式，则每组表达式必须包含数量相同的表达式，并且必须用圆括号隔开。每组中的表达式数量必须与条件中的运算符前的表达式的数量一致。

示例

下面是条件中的表达式列表的示例：

```
(1, 5, 10)
('THESE', 'ARE', 'STRINGS')
(('one', 'two', 'three'), ('blue', 'yellow', 'green'))
```

每组中的表达式数量必须与语句的第一部分中的数量匹配：

```
select * from venue
where (venuecity, venuestate) in (('Miami', 'FL'), ('Tampa', 'FL'))
order by venueid;

venueid |        venuename        | venuecity | venuestate | venueseats
---------+-------------------------+-----------+------------+------------
28 | American Airlines Arena | Miami     | FL         |          0
54 | St. Pete Times Forum    | Tampa     | FL         |          0
91 | Raymond James Stadium   | Tampa     | FL         |      65647
(3 rows)
```

标量子查询

标量子查询是圆括号中的常规 SELECT 查询，仅返回一个值：带有一个列的一行。执行此查询，返回值将在外部查询中使用。如果子查询返回零行，则子查询表达式的值为 null。如果它返回多行，则 Amazon Redshift 将返回错误。子查询可引用父查询中的变量，这将在子查询的任何一次调用中充当常量。

您可在需要表达式的大部分语句中使用标量子查询。标量子查询在下列情况下是无效表达式：

- 作为表达式的默认值
- 在 GROUP BY 和 HAVING 子句中

示例

以下子查询计算 2008 年全年的每笔销售支付的平均价格，然后外部查询使用输出中的值来比较每个季度每笔销售的平均价格：

```
select qtr, avg(pricepaid) as avg_saleprice_per_qtr,
(select avg(pricepaid)
from sales join date on sales.dateid=date.dateid
where year = 2008) as avg_saleprice_yearly
from sales join date on sales.dateid=date.dateid
where year = 2008
group by qtr
order by qtr;
qtr  | avg_saleprice_per_qtr | avg_saleprice_yearly
-------+-----------------------+----------------------
1     |                647.64 |               642.28
2     |                646.86 |               642.28
3     |                636.79 |               642.28
4     |                638.26 |               642.28
(4 rows)
```

函数表达式

语法

任何内置函数均可用作表达式。函数调用的语法是函数的名称后跟用圆括号括起的参数列表。

```
function ( [expression [, expression...]] )
```

参数

函数

任何内置函数。有关一些示例函数，请参阅SQL 函数参考。

expression

任何与函数预期的数据类型和参数计数匹配的表达式。

示例

```
abs (variable)
select avg (qtysold + 3) from sales;
```

```
select dateadd (day,30,caldate) as plus30days from date;
```

条件

主题

- 语法
- 比较条件
- 逻辑条件
- 模式匹配条件
- BETWEEN 范围条件
- Null 条件
- EXISTS 条件
- IN 条件

条件是包含一个或多个表达式和逻辑运算符的语句，计算结果为 true、false 或 unknown。条件有时也称为“谓词”。

> **Note**
>
> 所有字符串比较和 LIKE 模式匹配项均区分大小写。例如，“A”和“a”不匹配。但是，您可通过使用 ILIKE 谓词执行不区分大小写的模式匹配。

语法

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

比较条件

比较条件阐明两个值之间的逻辑关系。所有比较条件都是具有布尔值返回类型的二进制运算符。Amazon Redshift 支持下表中描述的比较运算符：

操作符	语法	描述
<	`a < b`	值 a 小于值 b。
>	`a > b`	值 a 大于值 b。
<=	`a <= b`	值 a 小于或等于值 b。
>=	`a >= b`	值 a 大于或等于值 b。
=	`a = b`	值 a 等于值 b。
<> 或 !=	`a <> b or a != b`	值 a 不等于值 b。
ANY | SOME	`a = ANY(subquery)`	值 a 等于子查询返回的任何值。
ALL	`a <> ALL or != ALL (subquery))`	值 a 不等于子查询返回的任何值。
IS TRUE | FALSE | UNKNOWN	`a IS TRUE`	值 a 是布尔值 TRUE。

使用说明

= ANY | SOME

ANY 和 SOME 关键字与 IN 条件同义，当比较操作相对于可返回一个或多个值的子查询所返回的至少一个值为 true 时，将返回 true。对于 ANY 和 SOME，Amazon Redshift 仅支持 = (等于) 条件。不支持不相等条件。

> **Note**
>
> 不支持 ALL 谓词。

<> ALL

ALL 关键字与 NOT IN (请参阅 IN 条件 条件) 同义并在表达式未包含在子查询的结果中时返回 true。对于 ALL，Amazon Redshift 仅支持 <> 或 != (不等于) 条件。不支持其他比较条件。

IS TRUE/FALSE/UNKNOWN

非零值等于 TRUE，0 等于 FALSE，null 等于 UNKNOWN。请参阅布尔值类型数据类型。

示例

下面是比较条件的一些简单示例：

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

以下查询返回 VENUE 表中座位数超过 10000 的场地：

```
select venueid, venuename, venueseats from venue
where venueseats > 10000
order by venueseats desc;

venueid |           venuename            | venueseats
---------+--------------------------------+------------
83 | FedExField                     |      91704
 6 | New York Giants Stadium        |      80242
79 | Arrowhead Stadium              |      79451
78 | INVESCO Field                  |      76125
69 | Dolphin Stadium                |      74916
67 | Ralph Wilson Stadium           |      73967
76 | Jacksonville Municipal Stadium |      73800
89 | Bank of America Stadium        |      73298
72 | Cleveland Browns Stadium       |      73200
86 | Lambeau Field                  |      72922
...
(57 rows)
```

此示例从 USERS 表中选择喜欢摇滚音乐的用户 (USERID)：

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
```

```
5
6
13
16
(5 rows)
```

此示例从 USERS 表中选择不清楚是否喜欢摇滚音乐的用户 (USERID)：

```
select firstname, lastname, likerock
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
----------+----------+----------
Rafael    | Taylor   |
Vladimir  | Humphrey |
Barry     | Roy      |
Tamekah   | Juarez   |
Mufutau   | Watkins  |
Naida     | Calderon |
Anika     | Huff     |
Bruce     | Beck     |
Mallory   | Farrell  |
Scarlett  | Mayer    |
(10 rows
```

具有 TIME 列的示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

```
select time_val from time_test;

time_val
--------------------
20:00:00
00:00:00.5550
00:58:00
```

以下示例从每个 timetz_val 中提取小时数。

```
select time_val from time_test where time_val < '3:00';
```

```
   time_val
---------------
 00:00:00.5550
 00:58:00
```

以下示例比较两种时间文本。

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
 t
```

具有 TIMETZ 列的示例

下面的示例表 TIMETZ_TEST 具有一个列 TIMETZ_VAL（类型 TIMETZ），其中插入了三个值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

下面的示例仅选择小于 3:00:00 UTC 的 TIMETZ 值。将值转换为 UTC 后进行比较。

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

以下示例比较两种 TIMETZ 文本。比较时忽略时区。

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

 ?column?
----------
 t
```

逻辑条件

逻辑条件组合两个条件的结果以生成一个结果。所有逻辑条件都是具有布尔值返回类型的二进制运算符。

语法

```
expression
{ AND | OR }
expression
NOT expression
```

逻辑条件使用具有三个值的布尔逻辑，其中 null 值表示未知关系。下表描述逻辑条件的结果，其中 E1 和 E2 表示表达式：

E1	E2	E1 AND E2	E1 OR E2	NOT E2
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

NOT 运算符先于 AND 计算，而 AND 运算符先于 OR 运算符计算。使用的任何圆括号可优先于此默认计算顺序。

示例

以下示例将返回 USERS 表中用户同时喜欢拉斯维加斯和运动的 USERID 和 USERNAME：

```
select userid, username from users
where likevegas = 1 and likesports = 1
order by userid;

userid | username
--------+----------
1 | JSG99FHE
67 | TWU10MZT
87 | DUF19VXU
92 | HYP36WEQ
109 | FPL38HZK
120 | DMJ24GUZ
123 | QZR22XGQ
130 | ZQC82ALK
133 | LBN45WCH
144 | UCX04JKN
165 | TEY68OEB
169 | AYQ83HGO
184 | TVX65AZX
...
(2128 rows)
```

下一个示例将返回 USERS 表中用户喜欢拉斯维加斯或运动或同时喜欢这二者的 USERID 和 USERNAME。此查询将返回上例中的所有输出以及只喜欢拉斯维加斯或运动的用户。

```
select userid, username from users
where likevegas = 1 or likesports = 1
order by userid;

userid | username
--------+----------
1 | JSG99FHE
2 | PGL08LJI
3 | IFT66TXU
5 | AEB55QTM
6 | NDQ15VBM
9 | MSD36KVR
10 | WKW41AIW
13 | QTF33MCG
15 | OWU78MTR
16 | ZMG93CDD
22 | RHT62AGI
27 | KOY02CVE
```

```
29 | HUH27PKK
...
(18968 rows)
```

以下查询使用圆括号将 OR 条件括起来以查找纽约或加利福尼亚演出过 Macbeth 的场地：

```
select distinct venuename, venuecity
from venue join event on venue.venueid=event.venueid
where (venuestate = 'NY' or venuestate = 'CA') and eventname='Macbeth'
order by 2,1;

venuename                |   venuecity
-----------------------------------------+---------------
Geffen Playhouse                         | Los Angeles
Greek Theatre                            | Los Angeles
Royce Hall                               | Los Angeles
American Airlines Theatre                | New York City
August Wilson Theatre                    | New York City
Belasco Theatre                          | New York City
Bernard B. Jacobs Theatre                | New York City
...
```

删除此示例中的圆括号将更改逻辑和查询的结果。

以下示例使用 NOT 运算符：

```
select * from category
where not catid=1
order by 1;

catid | catgroup |  catname  |                 catdesc
-------+----------+-----------+--------------------------------------------
2 | Sports   | NHL       | National Hockey League
3 | Sports   | NFL       | National Football League
4 | Sports   | NBA       | National Basketball Association
5 | Sports   | MLS       | Major League Soccer
...
```

以下示例使用一个 NOT 条件并后跟一个 AND 条件：

```
select * from category
where (not catid=1) and catgroup='Sports'
```

```
order by catid;

catid | catgroup | catname |             catdesc
-------+----------+---------+---------------------------------
2 | Sports   | NHL     | National Hockey League
3 | Sports   | NFL     | National Football League
4 | Sports   | NBA     | National Basketball Association
5 | Sports   | MLS     | Major League Soccer
(4 rows)
```

模式匹配条件

主题

- LIKE
- SIMILAR TO
- POSIX 运算符

模式匹配运算符针对搜索条件表达式中指定的模式搜索字符串，然后根据是否找到匹配项来返回 true 或 false。Amazon Redshift 使用三种模式匹配方法：

- LIKE 表达式

  LIKE 运算符将字符串表达式（如列名称）与使用通配符 %（百分比）和 _（下划线）的模式进行比较。LIKE 模式匹配始终涵盖整个字符串。LIKE 执行区分大小写的匹配，而 ILIKE 执行不区分大小写的匹配。
- SIMILAR TO 正则表达式

  SIMILAR TO 运算符使用 SQL 标准正则表达式模式来匹配字符串表达式，该模式可包含一组模式匹配元字符，其中包括 LIKE 运算符支持的两个元字符。SIMILAR TO 匹配整个字符串并且执行区分大小写的匹配。
- POSIX 样式的正则表达式

  与 LIKE 和 SIMILAR TO 运算符相比，POSIX 正则表达式提供了更强大的模式匹配手段。POSIX 正则表达式模式可与字符串的任何部分匹配，并执行区分大小写的匹配。

使用 SIMILAR TO 或 POSIX 运算符的正则表达式匹配的计算成本高昂。我们建议尽可能使用 LIKE，尤其是在处理非常多的行时。例如，下列查询的功能相同，但使用 LIKE 的查询相比于使用正则表达式的查询的运行速度快若干倍：

```
select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die%';
```

LIKE

LIKE 运算符将字符串表达式（如列名称）与使用通配符 %（百分比）和 _（下划线）的模式进行比较。LIKE 模式匹配始终涵盖整个字符串。若要匹配字符串中任意位置的序列，模式必须以百分比符号开始和结尾。

LIKE 区分大小写；ILIKE 不区分大小写。

语法

```
expression [ NOT ] LIKE | ILIKE pattern [ ESCAPE 'escape_char' ]
```

参数

expression

有效的 UTF-8 字符表达式（如列名称）。

LIKE | ILIKE

LIKE 执行区分大小写的模式匹配。ILIKE 对单字节 UTF-8 (ASCII) 字符执行不区分大小写的模式匹配。要为多字节字符执行不区分大小写的模式匹配，请将 expression 上的 LOWER 函数和带有 LIKE 函数的 pattern 一起使用。

与比较谓词（例如 = 和 <>）相反，LIKE 和 ILIKE 谓词并不会隐式忽略尾随空格。要忽略尾随空格，请使用 RTRIM 或者将 CHAR 列显式强制转换为 VARCHAR。

~~ 运算符等同于 LIKE，而 ~~* 等同于 ILIKE。此外，!~~ 和 !~~* 运算符等同于 NOT LIKE 和 NOT ILIKE。

pattern

具有要匹配的模式的有效的 UTF-8 字符表达式。

escape_char

将对模式中的元字符进行转义的字符表达式。默认为两个反斜杠 ('\\')。

如果 pattern 不包含元字符，则模式仅表示字符串本身；在此情况下，LIKE 的行为与等于运算符相同。

其中一个字符表达式可以是 CHAR 或 VARCHAR 数据类型。如果它们不同，Amazon Redshift 会将 pattern 转换为 expression 的数据类型。

LIKE 支持下列模式匹配元字符：

操作符	描述
%	匹配任意序列的零个或多个字符。
_	匹配任何单个字符。

示例

下表显示使用 LIKE 的模式匹配的示例：

表达式	返回值
'abc' LIKE 'abc'	True
'abc' LIKE 'a%'	True
'abc' LIKE '_B_'	False
'abc' ILIKE '_B_'	True
'abc' LIKE 'c%'	False

以下示例查找名称以“E”开头的所有城市：

```
select distinct city from users
where city like 'E%' order by city;
city
---------------
East Hartford
East Lansing
East Rutherford
East St. Louis
Easthampton
Easton
```

```
Eatontown
Eau Claire
...
```

以下示例查找姓中包含“ten”的用户：

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
-------------
Christensen
Wooten
...
```

以下示例演示如何匹配多个模式。

```
select distinct lastname from tickit.users
where lastname like 'Chris%' or lastname like '%Wooten' order by lastname;
lastname
-------------
Christensen
Christian
Wooten
...
```

以下示例查找第三和第四个字符为“ea”的城市。此命令使用 ILIKE 来演示不区分大小写的匹配：

```
select distinct city from users where city ilike '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

以下示例使用默认转义字符串 (\\ ) 搜索包含“start_” (文本 `start` 后跟下划线 _)的字符串：

```
select tablename, "column" from pg_table_def
where "column" like '%start\\_%'
```

```
limit 5;

     tablename     |    column
-------------------+---------------
 stl_s3client      | start_time
 stl_tr_conflict   | xact_start_ts
 stl_undone        | undo_start_ts
 stl_unload_log    | start_time
 stl_vacuum_detail | start_row
(5 rows)
```

以下示例指定“^”作为转义字符，然后使用该转义字符搜索包含“start_”（文本 `start` 后跟下划线 _)的字符串：

```
select tablename, "column" from pg_table_def
where "column" like '%start^_%' escape '^'
limit 5;

     tablename     |    column
-------------------+---------------
 stl_s3client      | start_time
 stl_tr_conflict   | xact_start_ts
 stl_undone        | undo_start_ts
 stl_unload_log    | start_time
 stl_vacuum_detail | start_row
(5 rows)
```

以下示例使用 ~~* 运算符对以“Ag”开头的城市进行不区分大小写（ILIKE）的搜索。

```
select distinct city from users where city ~~* 'Ag%' order by city;

city
------------
Agat
Agawam
Agoura Hills
Aguadilla
```

SIMILAR TO

SIMILAR TO 运算符使用 SQL 标准正则表达式模式来匹配字符串表达式（如列名称）。SQL 正则表达式可包含一组模式匹配元字符，包括 LIKE 运算符支持的两个元字符。

仅当模式与整个字符串匹配时，SIMILAR TO 运算符才会返回 true，这与 POSIX 正则表达式的行为不同（其中的模式可与字符串的任何部分匹配）。

SIMILAR TO 执行区分大小写的匹配。

> **Note**
>
> 使用 SIMILAR TO 的正则表达式匹配的计算成本高昂。我们建议尽可能使用 LIKE，尤其是在处理非常多的行时。例如，下列查询的功能相同，但使用 LIKE 的查询相比于使用正则表达式的查询的运行速度快若干倍：
>
> ```
> select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
> select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die
> %';
> ```

语法

```
expression [ NOT ] SIMILAR TO pattern [ ESCAPE 'escape_char' ]
```

参数

expression

有效的 UTF-8 字符表达式（如列名称）。

SIMILAR TO

SIMILAR TO 对 expression 中的整个字符串执行区分大小写的模式匹配。

pattern

有效的 UTF-8 字符表达式，表示 SQL 标准正则表达式模式。

escape_char

将对模式中的元字符进行转义的字符表达式。默认为两个反斜杠 ('\\')。

如果 pattern 不包含元字符，则模式仅表示字符串本身。

其中一个字符表达式可以是 CHAR 或 VARCHAR 数据类型。如果它们不同，Amazon Redshift 会将 pattern 转换为 expression 的数据类型。

SIMILAR TO 支持下列模式匹配元字符：

操作符	描述
%	匹配任意序列的零个或多个字符。
_	匹配任何单个字符。
|	表示替换（两个替换中的一个）。
*	重复上一项目零次或更多次。
+	重复上一项目一次或更多次。
?	重复上一项目零次或一次。
{m}	重复上一项目正好 m 次。
{m,}	重复上一项目 m 次或更多次。
{m,n}	重复上一项目至少 m 次且不超过 n 次。
()	圆括号将项分组为单个逻辑项。
[...]	括号表达式指定一个字符类，正如在 POSIX 正则表达式中一样。

示例

下表显示了使用 SIMILAR TO 的模式匹配的示例：

表达式	返回值
'abc' SIMILAR TO 'abc'	True
'abc' SIMILAR TO '_b_'	True
'abc' SIMILAR TO '_A_'	False
'abc' SIMILAR TO '%(b|d)%'	True

表达式	返回值
`'abc' SIMILAR TO '(b|c)%'`	False
`'AbcAbcdefgefg12efgefg12' SIMILAR TO '((Ab)?c)+d((efg)+(12))+'`	True
`'aaaaaab11111xy' SIMILAR TO 'a{6}_[0-9]{5}(x|y){2}'`	True
`'$0.87' SIMILAR TO '$[0-9]+(.[0-9][0-9])?'`	True

以下示例查找名称包含“E”或“H”的城市：

```
SELECT DISTINCT city FROM users
WHERE city SIMILAR TO '%E%|%H%' ORDER BY city LIMIT 5;

      city
-----------------
 Agoura Hills
 Auburn Hills
 Benton Harbor
 Beverly Hills
 Chicago Heights
```

以下示例使用默认转义字符串（“\\”）搜索包含“_”的字符串：

```
SELECT tablename, "column" FROM pg_table_def
WHERE "column" SIMILAR TO '%start\\_%'
ORDER BY tablename, "column" LIMIT 5;

        tablename         |       column
--------------------------+---------------------
 stcs_abort_idle          | idle_start_time
 stcs_abort_idle          | txn_start_time
 stcs_analyze_compression | start_time
 stcs_auto_worker_levels  | start_level
 stcs_auto_worker_levels  | start_wlm_occupancy
```

以下示例指定“^”作为转义字符串，然后使用此转义字符串搜索包含“_”的字符串：

```
SELECT tablename, "column" FROM pg_table_def
WHERE "column" SIMILAR TO '%start^_%' ESCAPE '^'
ORDER BY tablename, "column" LIMIT 5;

        tablename         |       column
--------------------------+--------------------
 stcs_abort_idle          | idle_start_time
 stcs_abort_idle          | txn_start_time
 stcs_analyze_compression | start_time
 stcs_auto_worker_levels  | start_level
 stcs_auto_worker_levels  | start_wlm_occupancy
```

POSIX 运算符

POSIX 正则表达式是指定匹配模式的字符序列。如果字符串是正则表达式描述的正则集的成员，则该字符串与正则表达式匹配。

与 LIKE 和 SIMILAR TO 运算符相比，POSIX 正则表达式提供了更强大的模式匹配手段。POSIX 正则表达式模式可匹配字符串的任何部分，这与 SIMILAR TO 运算符不同，SIMILAR TO 运算符仅当其模式匹配整个字符串时才返回 true。

> **Note**
>
> 使用 POSIX 运算符的正则表达式匹配的计算成本高昂。我们建议尽可能使用 LIKE，尤其是在处理非常多的行时。例如，下列查询的功能相同，但使用 LIKE 的查询相比于使用正则表达式的查询的运行速度快若干倍：
>
> ```
> select count(*) from event where eventname ~ '.*(Ring|Die).*';
> select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die
> %';
> ```

语法

```
expression [ ! ] ~ pattern
```

参数

expression

有效的 UTF-8 字符表达式（如列名称）。

!

求反运算符。请不要与正则表达式匹配。

~

对 expression 的任何子字符串执行区分大小写的匹配。

> **Note**
>
> ~~ 是 LIKE 的同义词。

pattern

表示正则表达式模式的字符串文本。

如果 pattern 不包含通配符，则模式仅表示字符串本身。

要搜索包含元字符的字符串（如“. * | ? ”等），请使用两个反斜杠（“ \\”）对字符进行转义。与 SIMILAR TO 和 LIKE 不同，POSIX 正则表达式语法不支持用户定义的转义字符。

其中一个字符表达式可以是 CHAR 或 VARCHAR 数据类型。如果它们不同，Amazon Redshift 会将 pattern 转换为 expression 的数据类型。

所有字符表达式都可以是 CHAR 或 VARCHAR 数据类型。如果表达式的数据类型不同，Amazon Redshift 会将其转换为 expression 的数据类型。

POSIX 模式匹配支持下列元字符：

POSIX	描述
.	匹配任何单个字符。
*	匹配零或多个匹配项。
+	匹配一个或多个匹配项。

POSIX	描述
?	匹配零或一个匹配项。
|	指定替换匹配项；例如，E | H E表示 或 H。
^	匹配行首字符。
$	匹配行尾字符。
$	匹配字符串的结尾。
[]	括号指定一个匹配列表，它应与列表中的一个表达式匹配。脱字号 (^) 位于不匹配列表之前，它与列表中表示的表达式之外的任何字符匹配。
( )	圆括号将项分组为单个逻辑项。
{m}	重复上一项目正好 m 次。
{m,}	重复上一项目 m 次或更多次。
{m,n}	重复上一项目至少 m 次且不超过 n 次。
[: :]	匹配 POSIX 字符类中的任何字符。在下列字符类中，Amazon Redshift 仅支持 ASCII 字符：[:alnum:] 、[:alpha:] 、[:lower:] 、[:upper:]

Amazon Redshift 支持下列 POSIX 字符类。

字符类	描述
[[:alnum:]]	所有 ASCII 字母数字字符
[[:alpha:]]	所有 ASCII 字母字符
[[:blank:]]	所有空格字符
[[:cntrl:]]	所有控制字符（非打印）
[[:digit:]]	所有数字

字符类	描述
[[:lower:]]	所有小写 ASCII 字母字符
[[:punct:]]	所有标点字符
[[:space:]]	所有空格字符（非打印）
[[:upper:]]	所有大写 ASCII 字母字符
[[:xdigit:]]	所有有效的十六进制字符

Amazon Redshift 在正则表达式中支持下列受 Perl 影响的运算符。使用两个反斜杠（“\\”）转义此运算符。

操作符	描述	等效的字符类表达式
\\d	数字字符	[[:digit:]]
\\D	非数字字符	[^[:digit:]]
\\w	单词字符	[[:word:]]
\\W	非单词字符	[^[:word:]]
\\s	空格字符	[[:space:]]
\\S	非空格字符	[^[:space:]]
\\b	边界字	

示例

下表显示了使用 POSIX 运算符的模式匹配的示例：

表达式	返回值
'abc' ~ 'abc'	True

表达式	返回值
'abc' ~ 'a'	True
'abc' ~ 'A'	False
'abc' ~ '.*(b|d).*'	True
'abc' ~ '(b|c).*'	True
'AbcAbcdefgefg12efgefg12' ~ '((Ab)?c)+d((efg)+(12))+'	True
'aaaaaab11111xy' ~ 'a{6}.[1]{5}(x|y){2}'	True
'$0.87' ~ '\\$[0-9]+(\\.[0-9][0-9])?'	True
'ab c' ~ '[[:space:]]'	True
'ab c' ~ '\\s'	True
' ' ~ '\\S'	False

以下示例查找名称包含 E 或 H 的城市：

```
SELECT DISTINCT city FROM users
WHERE city ~ '.*E.*|.*H.*' ORDER BY city LIMIT 5;

      city
-----------------
 Agoura Hills
 Auburn Hills
 Benton Harbor
 Beverly Hills
 Chicago Heights
```

以下示例查找名称不包含 E 或 H 的城市：

```
SELECT DISTINCT city FROM users WHERE city !~ '.*E.*|.*H.*' ORDER BY city LIMIT 5;

      city
-----------------
 Aberdeen
 Abilene
 Ada
 Agat
 Agawam
```

以下示例使用转义字符串（“\\”）搜索包含句点的字符串。

```
SELECT venuename FROM venue
WHERE venuename ~ '.*\\..*'
ORDER BY venueid;

          venuename
----------------------------
 St. Pete Times Forum
 Jobing.com Arena
 Hubert H. Humphrey Metrodome
 U.S. Cellular Field
 Superpages.com Center
 E.J. Nutter Center
 Bernard B. Jacobs Theatre
 St. James Theatre
```

BETWEEN 范围条件

BETWEEN 条件使用关键字 BETWEEN 和 AND 测试表达式是否包含在某个值范围中。

语法

```
expression [ NOT ] BETWEEN expression AND expression
```

表达式可以是数字、字符或日期时间数据类型，但它们必须是可兼容的。此范围包含起始值。

示例

第一个示例计算有多少个事务登记了 2、3 或 4 票证的销售：

```
select count(*) from sales
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

范围条件包含开始和结束值。

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

范围条件中的第一个表达式必须是较小的值，第二个表达式必须是较大的值。在以下示例中，由于表达式的值，将始终返回零行：

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

但是，应用 NOT 修饰符将反转逻辑并生成所有行的计数：

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
172456
(1 row)
```

以下查询将返回拥有 20000 到 50000 个座位的场馆的列表：

```
select venueid, venuename, venueseats from venue
```

```
where venueseats between 20000 and 50000
order by venueseats desc;

venueid |       venuename               | venueseats
---------+-------------------------------+------------
116 | Busch Stadium                 |      49660
106 | Rangers BallPark in Arlington |      49115
96 | Oriole Park at Camden Yards   |      48876
...
(22 rows)
```

以下示例演示了如何为日期值使用 BETWEEN：

```
select salesid, qtysold, pricepaid, commission, saletime
from sales
where eventid between 1000 and 2000
   and saletime between '2008-01-01' and '2008-01-03'
order by saletime asc;

salesid | qtysold | pricepaid | commission |   saletime
--------+---------+-----------+------------+---------------
  65082 |       4 |       472 |       70.8 | 1/1/2008 06:06
 110917 |       1 |       337 |      50.55 | 1/1/2008 07:05
 112103 |       1 |       241 |      36.15 | 1/2/2008 03:15
 137882 |       3 |      1473 |     220.95 | 1/2/2008 05:18
  40331 |       2 |        58 |        8.7 | 1/2/2008 05:57
 110918 |       3 |      1011 |     151.65 | 1/2/2008 07:17
  96274 |       1 |       104 |       15.6 | 1/2/2008 07:18
 150499 |       3 |       135 |      20.25 | 1/2/2008 07:20
  68413 |       2 |       158 |       23.7 | 1/2/2008 08:12
```

请注意，尽管 BETWEEN 的范围包括在内，但日期默认具有 00:00:00 的时间值。示例查询中唯一有效的 1 月 3 日行是 saletime 为 `1/3/2008 00:00:00` 的行。

Null 条件

在缺少值或值未知时，null 条件测试是否存在 null。

语法

```
expression IS [ NOT ] NULL
```

参数

expression

任何表达式（如列）。

IS NULL

当表达式的值为 null 时为 true；当表达式具有一个值时，为 false。

IS NOT NULL

当表达式的值为 null 时为 false；当表达式具有一个值时，为 true。

示例

此示例指示 SALES 表的 QTYSOLD 字段中包含 null 的次数：

```
select count(*) from sales
where qtysold is null;
count
-------
0
(1 row)
```

EXISTS 条件

EXISTS 条件测试子查询中是否存在行，并在子查询返回至少一个行时返回 true。如果指定 NOT，此条件将在子查询未返回任何行时返回 true。

语法

```
[ NOT ] EXISTS (table_subquery)
```

参数

EXISTS

当 table_subquery 返回至少一行时，为 true。

NOT EXISTS

当 table_subquery 未返回任何行时，为 true。

table_subquery

计算结果为包含一个或多个列和一个或多个行的表的子查询。

示例

此示例针对具有任何类型的销售的日期返回所有日期标识符，一次返回一个日期：

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
1827
1828
1829
...
```

IN 条件

IN 条件测试一组值或一个子查询中的成员值。

语法

```
expression [ NOT ] IN (expr_list | table_subquery)
```

参数

expression

数字、字符或日期时间表达式，针对 expr_list 或 table_subquery 进行计算，必须是与列表或子查询的数据类型兼容的。

expr_list

一个或多个逗号分隔的表达式，或一组或多组逗号分隔的表达式（用括号限定）。

table_subquery

一个子查询，计算结果为具有一行或多行的表，但在其选择列表中限制为一列。

IN | NOT IN

如果表达式是表达式列表或查询的成员，则 IN 将返回 true。如果表达式不是成员，NOT IN 将返回 true。在下列情况下，IN 和 NOT IN 将返回 NULL 并且不会返回任何行：如果 expression 生成 null；或者，如果没有匹配的 expr_list 或 table_subquery 值并且至少一个比较行生成 null。

示例

下列条件仅对列出的值有效：

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

优化大型 IN 列表

为了优化查询性能，包含 10 个以上的值的 IN 列表将在内部作为标量数组计算。少于 10 个值的 IN 列表将作为一系列 OR 谓词计算。SMALLINT、INTEGER、BIGINT、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP 和 TIMESTAMPTZ 数据类型均支持此优化。

查看查询的 EXPLAIN 输出以查看此优化的效果。例如：

```
explain select * from sales
QUERY PLAN
------------------------------------------------------------------------
XN Seq Scan on sales  (cost=0.00..6035.96 rows=86228 width=53)
Filter: (salesid = ANY ('{1,2,3,4,5,6,7,8,9,10,11}'::integer[]))
(2 rows)
```

SQL 命令

SQL 语言包括各种命令，您可以使用这些命令来创建和处理数据库对象、运行查询、加载表和修改表中的数据。

Amazon Redshift 基于 PostgreSQL。Amazon Redshift 和 PostgreSQL 之间的差别非常大，您在设计和开发数据仓库应用程序时必须注意这一点。有关 Amazon Redshift SQL 与 PostgreSQL 之间的差异的更多信息，请参阅Amazon Redshift 和 PostgreSQL。

> **Note**
>
> 单个 SQL 语句的最大大小为 16MB。

主题

- ABORT
- ALTER DATABASE
- ALTER DATASHARE
- ALTER DEFAULT PRIVILEGES
- ALTER EXTERNAL VIEW（预览版）
- ALTER FUNCTION
- ALTER GROUP
- ALTER IDENTITY PROVIDER
- ALTER MASKING POLICY
- ALTER MATERIALIZED VIEW
- ALTER RLS POLICY
- ALTER ROLE
- ALTER PROCEDURE
- ALTER SCHEMA
- ALTER SYSTEM
- ALTER TABLE
- ALTER TABLE APPEND
- ALTER USER
- ANALYZE
- ANALYZE COMPRESSION
- ATTACH MASKING POLICY
- ATTACH RLS POLICY
- BEGIN
- CALL
- CANCEL

- CLOSE
- COMMENT
- COMMIT
- COPY
- CREATE DATABASE
- CREATE DATASHARE
- CREATE EXTERNAL FUNCTION
- CREATE EXTERNAL SCHEMA
- CREATE EXTERNAL TABLE
- CREATE EXTERNAL VIEW（预览版）
- CREATE FUNCTION
- CREATE GROUP
- CREATE IDENTITY PROVIDER
- CREATE LIBRARY
- CREATE MASKING POLICY
- CREATE MATERIALIZED VIEW
- CREATE MODEL
- CREATE PROCEDURE
- CREATE RLS POLICY
- CREATE ROLE
- CREATE SCHEMA
- CREATE TABLE
- CREATE TABLE AS
- CREATE USER
- CREATE VIEW
- DEALLOCATE
- DECLARE
- 删除
- DESC DATASHARE

- DESC IDENTITY PROVIDER
- DETACH MASKING POLICY
- DETACH RLS POLICY
- DROP DATABASE
- DROP DATASHARE
- DROP EXTERNAL VIEW（预览版）
- DROP FUNCTION
- DROP GROUP
- DROP IDENTITY PROVIDER
- DROP LIBRARY
- DROP MASKING POLICY
- DROP MODEL
- DROP MATERIALIZED VIEW
- DROP PROCEDURE
- DROP RLS POLICY
- DROP ROLE
- DROP SCHEMA
- DROP TABLE
- DROP USER
- DROP VIEW
- END
- EXECUTE
- EXPLAIN
- FETCH
- GRANT
- INSERT
- INSERT（外部表）
- LOCK
- MERGE

- PREPARE
- REFRESH MATERIALIZED VIEW
- RESET
- REVOKE
- ROLLBACK
- SELECT
- SELECT INTO
- SET
- SET SESSION AUTHORIZATION
- SET SESSION CHARACTERISTICS
- SHOW
- SHOW COLUMNS
- SHOW EXTERNAL TABLE
- SHOW DATABASES
- SHOW MODEL
- SHOW DATASHARES
- SHOW PROCEDURE
- SHOW SCHEMAS
- SHOW TABLE
- SHOW TABLES
- SHOW VIEW
- START TRANSACTION
- TRUNCATE
- UNLOAD
- UPDATE
- VACUUM

ABORT

停止当前正在运行的事务，并放弃该事务执行的所有更新。ABORT 对已完成的事务没有任何效果。

此命令执行与 ROLLBACK 命令相同的功能。有关信息，请参阅 ROLLBACK。

语法

```
ABORT [ WORK | TRANSACTION ]
```

参数

WORK

可选关键字。

TRANSACTION

可选关键字；WORK 和 TRANSACTION 同义。

示例

以下示例创建一个表，然后启动将数据插入到表的事务。然后，ABORT 命令将回滚数据插入操作，以便将表保持为空表。

以下命令创建一个名为 MOVIE_GROSS 的示例表：

```
create table movie_gross( name varchar(30), gross bigint );
```

下一组命令启动将两个数据行插入到表中的事务：

```
begin;

insert into movie_gross values ( 'Raiders of the Lost Ark', 23400000);

insert into movie_gross values ( 'Star Wars', 10000000 );
```

接下来，以下命令从表中选择数据，表明已插入成功：

```
select * from movie_gross;
```

命令输出表明已成功插入两行：

```
        name           |  gross
```

```
------------------------+----------
Raiders of the Lost Ark | 23400000
Star Wars               | 10000000
(2 rows)
```

现在，此命令将数据更改回滚到事务开始的位置：

```
abort;
```

现在，从表中选择数据时，显示这是一个空表：

```
select * from movie_gross;

 name | gross
------+-------
(0 rows)
```

ALTER DATABASE

更改数据库的属性。

所需的权限

要使用 ALTER DATABASE，需要以下权限之一。

- Superuser
- 具有 ALTER DATABASE 权限的用户
- 数据库拥有者

语法

```
ALTER DATABASE database_name
{ RENAME TO new_name
| OWNER TO new_owner
| CONNECTION LIMIT { limit | UNLIMITED }
| COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }
| ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }
| INTEGRATION REFRESH {{ ALL | INERROR } TABLES [IN SCHEMA schema [, ...]] |
 TABLE schema.table [, ...]}
```

```
}
```

参数

database_name

要更改的数据库的名称。通常，您将更改当前未连接到的数据库；在任何情况下，更改只在后续的会话中才会生效。您可以更改当前数据库的所有者，但不能重命名该数据库：

```
alter database tickit rename to newtickit;
ERROR:  current database may not be renamed
```

RENAME TO

重命名指定的数据库。有关有效名称的更多信息，请参阅名称和标识符。不能重命名 dev、padb_harvest、template0、template1 或 sys:internal 数据库，也不能重命名当前数据库。只有数据库所有者或 superuser (p. 756) 可以重命名数据库；非超级用户所有者还必须具有 CREATEDB 权限。

new_name

新数据库名称。

OWNER TO

更改指定数据库的所有者。您可以更改当前数据库或其他某个数据库的所有者。只有超级用户可以更改所有者。

new_owner

新数据库所有者。新所有者必须是具有写入权限的现有数据库用户。有关用户权限的更多信息，请参阅GRANT。

CONNECTION LIMIT { limit | UNLIMITED }

允许用户同时打开的数据库连接的最大数量。此限制不适用于超级用户。使用 UNLIMITED 关键字设置允许的并行连接的最大数量。可能对每个用户的连接数量也会施加限制。有关更多信息，请参阅 CREATE USER。默认为 UNLIMITED。要查看当前连接，请查询 STV_SESSIONS 系统视图。

> **Note**
>
> 如果用户及数据库连接限制均适用，当用户尝试连接时，必须有一个同时低于这两个限制的未使用的连接槽可用。

COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }

指定字符串搜索或比较是区分大小写还是不区分大小写的子句。

您可以更改当前的空数据库的区分大小写。

您必须具有对当前数据库的权限才能更改区分大小写。具有 CREATE DATABASE 权限的超级用户或数据库拥有者也可以更改数据库的区分大小写设置。

ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }

指定针对数据库运行查询时使用的隔离级别的子句。

- SERIALIZABLE 隔离–为并发事务提供完全可序列性。有关更多信息，请参阅 可序列化的隔离。
- SNAPSHOT 隔离 – 提供隔离级别，来防止出现更新和删除冲突。

有关隔离级别的更多信息，请参阅 CREATE DATABASE。

更改数据库的隔离级别时，请考虑以下项目：

- 您必须对当前数据库具有超级用户或 CREATE DATABASE 权限，才能更改数据库隔离级别。
- 您不能更改 dev 数据库的隔离级别。
- 您不能更改事务块中的隔离级别。
- 如果有其他用户连接到数据库，则更改隔离级别命令将失败。
- 更改隔离级别命令可以更改当前会话的隔离级别设置。

INTEGRATION REFRESH {{ ALL | INERROR } TABLES [IN SCHEMA schema [, ...]] | TABLE schema.table [, ...]}

指定 Amazon Redshift 是否刷新指定架构或表中的所有表或有错误的表的子句。刷新将触发从源数据库完全复制指定架构或表中的表。

有关更多信息，请参阅《Amazon Redshift 管理指南》中的使用零 ETL 集成。有关集成状态的更多信息，请参阅SVV_INTEGRATION_TABLE_STATE和SVV_INTEGRATION。

使用说明

ALTER DATABASE 命令应用于后续会话，而不应用于当前会话。您必须重新连接到更改后的数据库，以查看更改结果。

示例

以下示例将一个名为 TICKIT_SANDBOX 的数据库重命名为 TICKIT_TEST：

```
alter database tickit_sandbox rename to tickit_test;
```

以下示例将 TICKIT 数据库（当前数据库）的所有者更改为 DWUSER：

```
alter database tickit owner to dwuser;
```

以下示例更改了 sampledb 数据库的数据库区分大小写：

```
ALTER DATABASE sampledb COLLATE CASE_INSENSITIVE;
```

以下示例使用 SNAPSHOT 隔离级别更改名为 **sampledb** 的数据库。

```
ALTER DATABASE sampledb ISOLATION LEVEL SNAPSHOT;
```

以下示例刷新零 ETL 集成的数据库 **sample_integration_db** 中的表 **sample_table1** 和 **sample_table2**。

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH TABLES sample_table1,
 sample_table2;
```

以下示例刷新零 ETL 集成中所有已同步和失败的表。

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH ALL tables;
```

以下示例刷新架构 **sample_schema** 的 ErrorState 中的所有表。

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH INERROR TABLES in SCHEMA
 sample_schema;
```

ALTER DATASHARE

更改数据共享的定义。您可以使用 ALTER DATASHARE 添加对象或删除对象。您只能更改当前数据库中的数据共享。将对象从关联数据库添加到数据共享中或者从数据共享中删除对象。对要添加或删除的数据共享具有所需权限的数据共享拥有者可以更改数据共享。

所需的权限

以下是 ALTER DATASHARE 所需的权限：

- 超级用户。
- 具有 ALTER DATASHARE 权限的用户。
- 对数据共享具有 ALTER 或 ALL 权限的用户。
- 要将特定对象添加到数据共享中，用户必须具有对象的权限。在此例中，用户应是对象的拥有者，或者具有这些对象的 SELECT、USAGE 或 ALL 权限。

语法

以下语法展示了如何向数据共享添加或删除对象。

```
ALTER DATASHARE datashare_name { ADD | REMOVE } {
TABLE schema.table [, ...]
| SCHEMA schema [, ...]
| FUNCTION schema.sql_udf (argtype,...) [, ...]
| ALL TABLES IN SCHEMA schema [, ...]
| ALL FUNCTIONS IN SCHEMA schema [, ...] }
```

以下语法展示了如何配置数据共享的属性。

```
ALTER DATASHARE datashare_name {
[ SET PUBLICACCESSIBLE [=] TRUE | FALSE ]
[ SET INCLUDENEW [=] TRUE | FALSE FOR SCHEMA schema ] }
```

参数

datashare_name

要更改的数据共享的名称。

ADD | REMOVE

指定是向数据集中添加对象还是从中删除对象的子句。

TABLE schema.table [, ...]

要添加到数据共享的指定 schema 中的表或视图的名称。

SCHEMA schema [, ...]

要添加到数据共享中的 schema 的名称。

FUNCTION schema.sql_udf (argtype,...) [, ...]

要添加到数据共享的用户定义的 SQL 函数的名称和参数类型。

ALL TABLES IN SCHEMA schema [, ...]

指定是否将指定 schema 中的所有表和视图添加到数据共享中的子句。

ALL FUNCTIONS IN SCHEMA schema [, ...] }

指定将指定 schema 中的所有函数添加到数据共享中的子句。

[SET PUBLICACCESSIBLE [=] TRUE | FALSE]

指定是否可以将数据共享共享给可公开访问的集群的子句。

[SET INCLUDENEW [=] TRUE | FALSE FOR SCHEMA schema]

指定是否将在指定 schema 中创建的任何未来表、视图或 SQL 用户定义函数 (UDF) 添加到数据共享中的子句。指定 schema 中的当前表、视图或 SQL UDF 不会添加到数据共享中。只有超级用户才可以更改每个数据共享-schema 对的此属性。预设情况下，INCLUDENEW 子句为 false。

ALTER DATASHARE 使用说明

- 以下用户可以更改数据共享：
  - 超级用户
  - 数据共享的拥有者
  - 对数据共享具有 ALTER 或 ALL 权限的用户
- 要将特定对象添加到数据共享中，用户必须具有对象的正确权限。用户应该是对象的拥有者，或者对这些对象具有 SELECT、USAGE 或 ALL 权限。
- 您可以共享 schema、表、常规视图、后期绑定视图、实体化视图和 SQL 用户定义函数 (UDF)。在架构中添加对象之前，首先将架构添加到数据共享中。

  当您添加 schema 时，Amazon Redshift 不会在其下添加所有对象。您必须显式添加它们。
- 我们建议您在可公开访问的设置处于开启状态下创建 AWS Data Exchange 数据共享。
- 通常，我们不建议您使用 ALTER DATASHARE 语句更改 AWS Data Exchange 数据共享，以关闭公开可访问性。如果您这样做的话，如果其集群可以公开访问，有权访问数据共享的 AWS 账户 将失去访问权限。执行这种类型的更改可能会违反 AWS Data Exchange 中的数据产品条款。对于此建议的例外情况，请参阅以下内容。

  以下示例显示了设置处于关闭状态下创建 AWS Data Exchange 数据共享时发生的错误。

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
ERROR:  Alter of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value 'c670ba4db22f4b'
```

要允许更改 AWS Data Exchange 数据共享，以禁用可公开访问的设置，请设置以下变量，然后再次运行 ALTER DATASHARE 语句。

```
SET datashare_break_glass_session_var to 'c670ba4db22f4b';
```

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
```

在这种情况下，Amazon Redshift 会生成一个随机的一次性值来设置会话变量，以允许对 AWS Data Exchange 数据共享执行 ALTER DATASHARE SET PUBLICACCESSIBLE FALSE。

示例

以下示例将 public 架构添加到数据共享 salesshare 中。

```
ALTER DATASHARE salesshare ADD SCHEMA public;
```

以下示例将 public.tickit_sales_redshift 表添加到了数据共享 salesshare 中。

```
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
```

以下示例将所有表添加到了数据共享 salesshare 中。

```
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

以下示例删除了数据共享 salesshare 中的 public.tickit_sales_redshift 表。

```
ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
```

ALTER DEFAULT PRIVILEGES

定义默认访问权限集，这些权限将应用于指定的用户在以后创建的对象。默认情况下，用户只能更改他们自己的默认访问权限。只有超级用户能够为其他用户指定默认权限。

您可以将默认权限应用到角色、用户或用户组。您可以为在当前数据库中创建的所有对象全局设置默认权限，也可以仅为在指定的架构中创建的对象进行此设置。

默认权限仅应用于新对象。运行 ALTER DEFAULT PRIVILEGES 时不会更改现有对象的权限。要授予对数据库或架构中任何用户创建的所有当前和将来对象的权限，请参阅限定范围权限。

要查看有关数据库用户的默认权限的信息，请查询 PG_DEFAULT_ACL 系统目录表。

有关权限的更多信息，请参阅 GRANT。

所需的权限

以下是 ALTER DEFAULT PRIVILEGES 所需的权限：

- Superuser
- 具有 ALTER DEFAULT PRIVILEGES 权限的用户
- 更改自己的默认访问权限的用户
- 为其具有访问权限的 Schema 设置权限的用户

语法

```
ALTER DEFAULT PRIVILEGES
    [ FOR USER target_user [, ...] ]
    [ IN SCHEMA schema_name [, ...] ]
    grant_or_revoke_clause

where grant_or_revoke_clause is one of:

GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES | TRUNCATE } [,...] |
 ALL [ PRIVILEGES ] }
 ON TABLES
 TO { user_name [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
 ON FUNCTIONS
 TO { user_name [ WITH GRANT OPTION ] |  ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
```

```
 ON PROCEDURES
 TO { user_name [ WITH GRANT OPTION ] |  ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

REVOKE [ GRANT OPTION FOR ] { { SELECT | INSERT | UPDATE | DELETE | REFERENCES |
 TRUNCATE } [,...] | ALL [ PRIVILEGES ] }
 ON TABLES
 FROM user_name [, ...] [ RESTRICT ]

REVOKE  { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRUNCATE } [,...] | ALL
 [ PRIVILEGES ] }
 ON TABLES
 FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [ RESTRICT ]

REVOKE [ GRANT OPTION FOR ] { EXECUTE | ALL [ PRIVILEGES ] }
 ON FUNCTIONS
 FROM user_name [, ...] [ RESTRICT ]

REVOKE { EXECUTE | ALL [ PRIVILEGES ] }
 ON FUNCTIONS
 FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [ RESTRICT ]

REVOKE [ GRANT OPTION FOR ] { EXECUTE | ALL [ PRIVILEGES ] }
 ON PROCEDURES
 FROM user_name [, ...] [ RESTRICT ]

REVOKE { EXECUTE | ALL [ PRIVILEGES ] }
 ON PROCEDURES
 FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [ RESTRICT ]
```

参数

FOR USER target_user

可选。为其定义默认权限的用户的名称。只有超级用户能够为其他用户指定默认权限。默认值为当前用户。

IN SCHEMA schema_name

可选。如果出现 IN SCHEMA 子句，则指定默认权限将应用于在指定 schema_name 中创建的新对象。在这种情况下，作为 ALTER DEFAULT PRIVILEGES 目标的用户或用户组必须对指定 schema 拥有 CREATE 权限。特定于某个 schema 的默认权限将添加到现有的全局默认权限中。默认情况下，默认权限全局应用于整个数据库。

GRANT

针对指定用户创建的所有新表和视图、函数或存储过程，向指定的用户或组授予的权限集。与使用 GRANT 命令一样，您可以使用 GRANT 子句来设置相同的权限和选项。

WITH GRANT OPTION

一个子句，指示接收权限的用户又可以将相同权限授予其他用户。您无法将 WITH GRANT OPTION 授予组或 PUBLIC。

TO user_name | ROLE role_name | GROUP group_name

将指定的默认权限应用于的用户、角色或用户组的名称。

REVOKE

针对指定用户创建的所有新表、函数或存储过程，从指定的用户或组撤销权限集。与使用 REVOKE 命令一样，您可以使用 REVOKE 子句来设置相同的权限和选项。

GRANT OPTION FOR

一个子句，仅撤消将指定的权限授予其他用户的选项，而不撤消权限本身。您无法从组或 PUBLIC 撤消 GRANT OPTION。

FROM user_name | ROLE role_name | GROUP group_name

默认情况下从其撤消指定权限的用户、角色或用户组的名称。

RESTRICT

RESTRICT 选项仅会撤消用户直接授予的权限。这是默认模式。

示例

假设您希望允许用户组 `report_readers` 中的所有用户查看用户 `report_admin` 创建的所有表和视图。在这种情况下，以超级用户身份执行以下命令。

```
alter default privileges for user report_admin grant select on tables to group
 report_readers;
```

在以下示例中，第一个命令授予对您创建的所有新表和视图的 SELECT 权限。

```
alter default privileges grant select on tables to public;
```

以下示例针对您在 sales_admin schema 中创建的所有新表和视图，将 INSERT 权限授予 sales 用户组。

```
alter default privileges in schema sales grant insert on tables to group sales_admin;
```

以下示例撤消上述示例中 ALTER DEFAULT PRIVILEGES 命令的执行效果。

```
alter default privileges in schema sales revoke insert on tables from group
 sales_admin;
```

默认情况下，PUBLIC 用户组对所有新的用户定义的函数具有执行权限。要撤消对您的新函数的 public 执行权限，然后只将执行权限授予 dev_test 用户组，请运行以下命令。

```
alter default privileges revoke execute on functions from public;
alter default privileges grant execute on functions to group dev_test;
```

ALTER EXTERNAL VIEW（预览版）

以下是预览版 Data Catalog for Amazon Redshift 中的预发行文档视图。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

您可以在预览版中创建 Amazon Redshift 集群，以便测试 Amazon Redshift 的新功能。您无法在生产环境中使用这些功能，也无法将预览版集群移动到生产集群或另一个跟踪上的集群。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

在预览版中创建集群

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择预置集群控制面板，然后选择集群。列出您的账户在当前 AWS 区域 区域中的集群。列表中的各个列中显示了每个集群的一部分属性。
3. 集群列表页面上会显示一个横幅，其中介绍了预览版。选择创建预览版集群按钮以打开创建集群页面。
4. 输入集群的属性。选择包含要测试的功能的预览版跟踪。我们建议输入的集群名称指明要对该集群进行预览版跟踪。为您的集群选择选项，包括标记为 -preview 的选项，用于要测试的功能。有关创建集群的一般信息，请参阅《Amazon Redshift 管理指南》中的创建集群。

5. 选择创建集群以在预览模式下创建集群。

> **Note**
>
> preview_2023 跟踪是最新可用的预览版跟踪。此版本仅支持创建具有 RA3 节点类型的集群。不支持节点类型 DC2 以及任何更早的节点类型。

6. 当您的预览集群可用时，使用 SQL 客户端加载和查询数据。

Data Catalog 视图预览功能仅在以下区域中可用。

- 美国东部（俄亥俄州）(us-east-2)
- 美国东部（弗吉尼亚州北部）(us-east-1)
- 美国西部（北加利福尼亚）(us-west-1)
- 亚太地区（东京）(ap-northeast-1)
- 欧洲地区（爱尔兰）(eu-west-1)
- 欧洲地区（斯德哥尔摩）(eu-north-1)

您也可以创建预览工作组来测试 Data Catalog 视图。您无法在生产中使用这些功能，也无法将您的工作组移至其他工作组。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。有关如何创建预览工作组的说明，请参阅创建预览工作组。

使用 ALTER EXTERNAL VIEW 命令更新您的外部视图。根据您使用的参数，也可以引用此视图的其他 SQL 引擎（例如 Amazon Athena 和 Amazon EMR Spark）可能会受到影响。有关 Data Catalog 视图的更多信息，请参阅创建 Data Catalog 视图（预览版）。

语法

```
ALTER EXTERNAL VIEW schema_name.view_name
{catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name |
 external_schema_name.view_name}
[FORCE] { AS (query_definition) | REMOVE DEFINITION }
```

参数

schema_name.view_name

附加到 AWS Glue 数据库的架构，后面是视图的名称。

catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name | external_schema_name.view_name

更改视图时要使用的架构符号。可以指定使用您创建的 Glue 数据库 AWS Glue Data Catalog 或您创建的外部架构。有关更多信息，请参阅 CREATE DATABASE 和 CREATE EXTERNAL SCHEMA。

FORCE

即使表中引用的对象与其他 SQL 引擎不一致，AWS Lake Formation 是否仍应更新视图的定义。如果 Lake Formation 更新了视图，其他 SQL 引擎就会认为该视图是过时的，直到这些引擎也更新为止。

AS query_definition

Amazon Redshift 为更改视图而运行的 SQL 查询的定义。

REMOVE DEFINITION

是否删除并重新创建视图。必须删除并重新创建视图才能将其标记为 PROTECTED。

示例

以下示例更改了名为 sample_schema.glue_data_catalog_view 的 Data Catalog 视图。

```
ALTER EXTERNAL VIEW sample_schema.glue_data_catalog_view
FORCE
REMOVE DEFINITION
```

ALTER FUNCTION

重命名函数或者更改拥有者。函数名称和数据类型均为必需项。只有拥有者或超级用户可以重命名函数。只有超级用户可以更改函数的拥有者。

语法

```
ALTER FUNCTION function_name ( { [ py_arg_name py_arg_data_type | sql_arg_data_type }
 [ , ... ] ] )
    RENAME TO new_name
```

```
ALTER FUNCTION function_name ( { [ py_arg_name py_arg_data_type | sql_arg_data_type }
 [ , ... ] ] )
```

```
    OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

参数

function_name

要更改的函数的名称。指定当前搜索路径中函数的名称，或者通过格式 `schema_name.function_name` 使用特定架构。

py_arg_name py_arg_data_type | sql_arg_data_type

可选。Python 用户定义函数的输入参数名称和数据类型的列表，或 SQL 用户定义函数的输入参数数据类型的列表。

new_name

用户定义函数的新名称。

new_owner | CURRENT_USER | SESSION_USER

用户定义函数的新拥有者。

示例

以下示例将函数的名称从 `first_quarter_revenue` 更改为 `quarterly_revenue`。

```
ALTER FUNCTION first_quarter_revenue(bigint, numeric, int)
         RENAME TO quarterly_revenue;
```

以下示例将 `quarterly_revenue` 函数的拥有者更改为 `etl_user`。

```
ALTER FUNCTION quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

ALTER GROUP

更改用户组。使用此命令可以将用户添加到组、从组中删除用户或重命名组。

语法

```
ALTER GROUP group_name
{
ADD USER username [, ... ] |
```

```
DROP USER username [, ... ] |
RENAME TO new_name
}
```

参数

group_name

要修改的用户组的名称。

ADD

将用户添加到用户组。

DROP

从用户组中删除用户。

username

要添加到组或从组中删除的用户的名称。

RENAME TO

重命名用户组。以两个下划线开头的组名保留供 Amazon Redshift 内部使用。有关有效名称的更多信息，请参阅名称和标识符。

new_name

用户组的新名称。

示例

以下示例将名为 DWUSER 的用户添加到 ADMIN_GROUP 组。

```
ALTER GROUP admin_group
ADD USER dwuser;
```

以下示例将组 ADMIN_GROUP 重命名为 ADMINISTRATORS。

```
ALTER GROUP admin_group
RENAME TO administrators;
```

以下示例将两个用户添加到组 ADMIN_GROUP。

```
ALTER GROUP admin_group
ADD USER u1, u2;
```

以下示例从 ADMIN_GROUP 组删除两个用户。

```
ALTER GROUP admin_group
DROP USER u1, u2;
```

ALTER IDENTITY PROVIDER

更改身份提供者以分配新的参数和值。运行此命令时，先前设置的所有参数值都将在分配新值之前删除。只有超级用户可以更改身份提供者。

语法

```
ALTER IDENTITY PROVIDER identity_provider_name
[PARAMETERS parameter_string]
[NAMESPACE namespace]
[IAM_ROLE iam_role]
[DISABLE | ENABLE]
```

参数

identity_provider_name

新身份提供者的名称。有关有效名称的更多信息，请参阅名称和标识符。

parameter_string

一个包含格式正确的 JSON 对象的字符串，其中包含特定身份提供者所需的参数和值。

命名空间

组织命名空间。

iam_role

提供连接到 IAM Identity Center 的权限的 IAM 角色。仅当身份提供者类型为 AWSIDC 时，此参数才适用。

DISABLE 或 ENABLE

开启或关闭身份提供者。默认值为 ENABLE

示例

以下示例更改名为 oauth_standard 的身份提供者。它特别适用于 Microsoft Azure AD 作为身份提供者的情况。

```
ALTER IDENTITY PROVIDER oauth_standard
PARAMETERS '{"issuer":"https://sts.windows.net/2sdfdsf-d475-420d-b5ac-667adad7c702/",
"client_id":"87f4aa26-78b7-410e-bf29-57b39929ef9a",
"client_secret":"BUAH~ewrqewrqwerUUY^%tHe1oNZShoiU7",
"audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift"]
}'
```

以下示例显示了如何设置身份提供者命名空间。这可能适用于 Microsoft Azure AD（如果它遵循与上一个示例类似的语句），也可能适用于其它身份提供者。如果您通过托管应用程序设置了连接，则它也可能适用于将现有的 Amazon Redshift 预调配集群或 Amazon Redshift Serverless 工作组连接到 IAM Identity Center 的情况。

```
ALTER IDENTITY PROVIDER "my-redshift-idc-application"
NAMESPACE 'MYCO';
```

以下示例设置了 IAM 角色，并适用于配置 Redshift 与 IAM Identity Center 集成的使用案例。

```
ALTER IDENTITY PROVIDER "my-redshift-idc-application"
IAM_ROLE 'arn:aws:iam::123456789012:role/myadministratorrole';
```

有关设置从 Redshift 到 IAM Identity Center 的连接的更多信息，请参阅[将 Redshift 与 IAM Identity Center 连接，为用户提供单点登录体验](#)。

禁用身份提供者

以下示例语句显示了如何禁用身份提供者。禁用身份提供者后，身份提供者中的联合用户将无法登录集群，直至再次启用它。

```
ALTER IDENTITY PROVIDER "redshift-idc-app" DISABLE;
```

ALTER MASKING POLICY

更改现有的动态数据掩蔽政策。有关动态数据掩蔽的更多信息，请参阅 [动态数据掩蔽](#)。

超级用户和具有 sys:secadmin 角色的用户或角色可以更改掩蔽政策。

语法

```
ALTER MASKING POLICY policy_name
   USING (masking_expression);
```

参数

policy_name

屏蔽策略的名称。此名称必须是数据库中已存在的掩蔽政策的名称。

masking_expression

用于转换目标列的 SQL 表达式。可以使用诸如字符串操作函数之类的数据操作函数来编写该表达式，也可以与使用 SQL、Python 或 AWS Lambda 编写的用户定义函数结合使用。

表达式必须与原始表达式的输入列和数据类型相匹配。例如，如果原始掩蔽政策的输入列是 `sample_1 FLOAT` 和 `sample_2 VARCHAR(10)`，则您将无法更改掩蔽政策来使用第三列，也无法使该政策采用一个浮点数和一个布尔值。如果您使用常量作为掩蔽表达式，则必须将其显式转换为与输入类型匹配的类型。

您必须对在屏蔽表达式中使用的任何用户定义函数具有 USAGE 权限。

ALTER MATERIALIZED VIEW

启用对实体化视图的自动刷新。

语法

```
ALTER MATERIALIZED VIEW mv_name
[ AUTO REFRESH { YES | NO } ]
[ ROW LEVEL SECURITY { ON | OFF } [ CONJUNCTION TYPE { AND | OR } ] [FOR DATASHARES] ];
```

参数

mv_name

要更改的实体化视图的名称。

AUTO REFRESH { YES | NO }

开启或关闭实体化视图的自动刷新的子句。有关自动刷新实体化视图的更多信息，请参阅刷新实体化视图。

ROW LEVEL SECURITY { ON | OFF } [CONJUNCTION TYPE { AND | OR }] [FOR DATASHARES]

一个对关系开启或关闭行级安全性的子句。

在为关系开启行级安全性后，您只能读取行级安全策略允许您访问的行。如果没有策略向您授予对关系的访问权限，您将看不到关系中的任何行。只有超级用户和拥有 `sys:secadmin` 角色的用户或角色才能设置 ROW LEVEL SECURITY 子句。有关更多信息，请参阅 行级别安全性。

- [CONJUNCTION TYPE { AND | OR }]

  一个允许您为关系选择行级安全策略的联接类型的子句。将多个行级安全策略附加到关系时，可以将这些策略与 AND 或 OR 子句合并。默认情况下，Amazon Redshift 将 RLS 策略与 AND 子句合并。具有 `sys:secadmin` 角色的超级用户、用户或角色可以使用此子句为关系定义行级安全策略的联接类型。有关更多信息，请参阅 为每个用户组合多个策略。

- FOR DATASHARES

  一个子句，用于确定是否可以通过数据共享访问受 RLS 保护的关系。默认情况下，无法通过数据共享访问受 RLS 保护的关系。使用此子句运行的 ALTER MATERIALIZED VIEW ROW LEVEL SECURITY 命令只会影响关系的数据共享可访问性属性。ROW LEVEL SECURITY 属性未更改。

  如果您允许通过数据共享访问受 RLS 保护的关系，则该关系在使用者端数据共享数据库中没有行级安全性。该关系在生产者端保留其 RLS 属性。

示例

以下示例启用要自动刷新的 `tickets_mv` 实体化视图。

```
ALTER MATERIALIZED VIEW tickets_mv AUTO REFRESH YES
```

DISTYLE 和 SORTKEY 示例

本主题中的示例向您展示了如何使用 ALTER MATERIAD VIEW 对 DISTYLE 和 SORTKEY 进行更改。

以下示例查询显示了如何使用示例基表更改 DISTSTYLE KEY DISTKEY 列：

```
CREATE TABLE base_inventory(
inv_date_sk int4 not null,
inv_item_sk int4 not null,
inv_warehouse_sk int4 not null,
inv_quantity_on_hand int4
);

INSERT INTO base_inventory VALUES(1,1,1,1);

CREATE MATERIALIZED VIEW inventory DISTSTYLE EVEN
as SELECT * FROM base_inventory;
SELECT "table", DISTSTYLE FROM svv_table_info WHERE "table" = 'inventory';

ALTER MATERIALIZED VIEW inventory ALTER DISTSTYLE KEY DISTKEY inv_warehouse_sk;
SELECT "table", DISTSTYLE FROM svv_table_info where "table" = 'inventory';

ALTER MATERIALIZED VIEW inventory ALTER DISTKEY inv_item_sk;
SELECT "table", diststyle from svv_table_info where "table" = 'inventory';
```

将实体化视图更改为 DISTSTYLE ALL：

```
CREATE TABLE base_inventory(
inv_date_sk int4 not null,
inv_item_sk int4 not null,
inv_warehouse_sk int4 not null,
inv_quantity_on_hand int4
);

INSERT INTO base_inventory values(1,1,1,1);

CREATE MATERIALIZED VIEW inventory DISTSTYLE EVEN
as SELECT * FROM base_inventory;

SELECT "table", DISTSTYLE FROM svv_table_info WHERE "table" = 'inventory';
```

以下命令显示了使用示例基表的 ALTER MATERIALIZED VIEW SORTKEY 示例：

```
CREATE MATERIALIZED VIEW base_inventory (c0 int, c1 int);

CREATE MATERIALIZED VIEW inventory
interleaved sortkey(c0, c1)
as SELECT * FROM base_inventory;
```

```
SELECT "table", sortkey1 FROM svv_table_info WHERE "table" = 'inventory';

ALTER MATERIALIZED VIEW t1 alter sortkey(c0, c1);
SELECT "table", diststyle, sortkey_num FROM svv_table_info WHERE "table" = 'inventory';

ALTER MATERIALIZED VIEW t1 alter sortkey none;
SELECT "table", diststyle, sortkey_num FROM svv_table_info WHERE "table" = 'inventory';

ALTER MATERIALIZED VIEW t1 alter sortkey(c0);
SELECT "table", diststyle, sortkey_num FROM svv_table_info WHERE "table" = 'inventory';
```

ALTER RLS POLICY

更改表上现有的行级别安全性策略。

超级用户和具有 sys:secadmin 角色的用户或角色可以更改策略。

语法

```
ALTER RLS POLICY policy_name
USING ( using_predicate_exp );
```

参数

policy_name

策略的名称。

USING (using_predicate_exp)

指定应用于查询的 WHERE 子句的筛选器。Amazon Redshift 会在查询级别的用户谓词之前应用策略谓词。例如，**`current_user = 'joe' and price > 10`** 限制 Joe 只能查看价格高于 10 美元的记录。

该表达式可以访问在 CREATE RLS POLICY 语句的 WITH 子句中声明的变量，该语句用于创建名为 policy_name 的策略。

示例

以下示例介绍了更改 RLS 策略。

```
-- First create an RLS policy that limits access to rows where catgroup is 'concerts'.
CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'concerts');

-- Then, alter the RLS policy to only show rows where catgroup is 'piano concerts'.
ALTER RLS POLICY policy_concerts
USING (catgroup = 'piano concerts');
```

ALTER ROLE

重命名角色或者更改拥有者。有关 Amazon Redshift 系统定义的角色列表，请参阅the section called “Amazon Redshift 系统定义的角色”。

所需的权限

以下是 ALTER ROLE 所需的权限：

- Superuser
- 具有 ALTER ROLE 权限的用户

语法

```
ALTER ROLE role [ WITH ]
  { { RENAME TO role } | { OWNER TO user_name } }[, ...]
  [ EXTERNALID TO external_id ]
```

参数

role

要更改的角色的名称。

RENAME TO

角色的新名称。

OWNER TO user_name

角色的新拥有者。

EXTERNALID TO external_id

角色的新外部 ID，与身份提供者关联。有关更多信息，请参阅 Amazon Redshift 的原生身份提供者 (IdP) 联合身份验证。

示例

以下示例将角色的名称从 sample_role1 更改为 sample_role2。

```
ALTER ROLE sample_role1 WITH RENAME TO sample_role2;
```

以下示例将更改角色的拥有者。

```
ALTER ROLE sample_role1 WITH OWNER TO user1
```

ALTER ROLE 的语法与以下 ALTER PROCEDURE 类似。

```
ALTER PROCEDURE first_quarter_revenue(bigint, numeric) RENAME TO quarterly_revenue;
```

以下示例将过程的拥有者更改为 etl_user。

```
ALTER PROCEDURE quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

以下示例使用与身份提供者关联的新外部 ID 更新了角色 sample_role1。

```
ALTER ROLE sample_role1 EXTERNALID TO "XYZ456";
```

ALTER PROCEDURE

重命名过程或者更改拥有者。需要过程名称和数据类型（或签名）。只有拥有者或超级用户可以重命名过程。只有超级用户可以更改过程的拥有者。

语法

```
ALTER PROCEDURE sp_name [ ( [ [ argname ] [ argmode ] argtype [, ...] ] ) ]
    RENAME TO new_name
```

```
ALTER PROCEDURE sp_name [ ( [ [ argname ] [ argmode ] argtype [, ...] ] ) ]
    OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

参数

sp_name

要变更的过程的名称。只指定当前搜索路径中过程的名称，或者通过格式 `schema_name.sp_procedure_name` 使用特定 schema。

[argname] [argmode] argtype

参数名称、参数模式和数据类型的列表。只需要输入数据类型，这些数据类型用于标识存储过程。另外，您可以提供用于创建过程的完整签名，包括输入和输出参数及其模式。

new_name

存储过程的新名称。

new_owner | CURRENT_USER | SESSION_USER

存储过程的新拥有者。

示例

以下示例将过程的名称从 `first_quarter_revenue` 更改为 `quarterly_revenue`。

```
ALTER PROCEDURE first_quarter_revenue(volume INOUT bigint, at_price IN numeric,
 result OUT int) RENAME TO quarterly_revenue;
```

此示例等效于以下内容：

```
ALTER PROCEDURE first_quarter_revenue(bigint, numeric) RENAME TO quarterly_revenue;
```

以下示例将过程的拥有者更改为 `etl_user`。

```
ALTER PROCEDURE quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

ALTER SCHEMA

更改现有 schema 的定义。使用此命令可重命名 schema 或更改 schema 的所有者。例如，当您计划创建现有 schema 的新版本时，将现有 schema 重命名可保留该 schema 的备份副本。有关 schema 的更多信息，请参阅 CREATE SCHEMA。

要查看已配置的 schema 配额，请参阅SVV_SCHEMA_QUOTA_STATE。

要查看已超出 schema 配额的记录，请参阅STL_SCHEMA_QUOTA_VIOLATIONS。

所需的权限

以下是 ALTER SCHEMA 所需的权限：

- Superuser
- 具有 ALTER SCHEMA 权限的用户
- Schema 拥有者

更改架构名称时，请注意，使用了旧名称的对象，例如存储过程或实体化视图，必须对其进行更新以使用新名称。

语法

```
ALTER SCHEMA schema_name
{
RENAME TO new_name |
OWNER TO new_owner |
QUOTA { quota [MB | GB | TB] | UNLIMITED }
}
```

参数

schema_name

要修改的数据库 schema 的名称。

RENAME TO

用于重命名 schema 的子句。

new_name

schema 的新名称。有关有效名称的更多信息，请参阅名称和标识符。

OWNER TO

用于更改 schema 所有者的子句。

new_owner

schema 的新所有者。

QUOTA

指定的 schema 可以使用的最大磁盘空间量。此空间是指定 schema 下所有表的整体大小。Amazon Redshift 将选定值转换为 MB。如果您未指定值，则 GB 是默认的测量单位。

有关配置 schema 配额的更多信息，请参阅CREATE SCHEMA。

示例

以下示例将 SALES schema 重命名为 US_SALES。

```
alter schema sales
rename to us_sales;
```

以下示例将 US_SALES schema 的所有权授予用户 DWUSER。

```
alter schema us_sales
owner to dwuser;
```

以下示例将配额更改为 300 GB 并删除此配额。

```
alter schema us_sales QUOTA 300 GB;
alter schema us_sales QUOTA UNLIMITED;
```

ALTER SYSTEM

更改 Amazon Redshift 集群或 Redshift Serverless 工作组的系统级配置选项。

所需的权限

以下用户类型之一可以运行 ALTER SYSTEM 命令：

- Superuser
- 管理员用户

语法

```
ALTER SYSTEM SET system-level-configuration = {true| t | on | false | f | off}
```

参数

system-level-configuration

系统级配置。有效值：data_catalog_auto_mount 和 metadata_security。

{true| t | on | false | f | off}

用于激活或停用系统级配置的值。true、t 或 on 表示要激活配置。false、f 或 off 表示要停用配置。

使用说明

对于预置集群，对 data_catalog_auto_mount 的更改将在集群下次重新引导时生效。有关更多信息，请参阅《Amazon Redshift 管理指南》中的重新引导集群。

对于无服务器工作组，对 data_catalog_auto_mount 的更改不会立即生效。

示例

以下示例开启了自动挂载 AWS Glue Data Catalog 的功能。

```
ALTER SYSTEM SET data_catalog_auto_mount = true;
```

以下示例开启了元数据安全性。

```
ALTER SYSTEM SET metadata_security = true;
```

设置默认身份命名空间

此示例特定于使用身份提供者。您可以将 Redshift 与 IAM Identity Center 和身份提供者集成，来集中管理 Redshift 和其它 AWS 服务的身份。

以下示例显示了如何为系统设置默认身份命名空间。这样做后，将可以更轻松地运行 GRANT 和 CREATE 语句，因为您不必包括此命名空间来作为每个身份的前缀。

```
ALTER SYSTEM SET default_identity_namespace = 'MYCO';
```

运行命令后，您可以运行如下语句：

```
GRANT SELECT ON TABLE mytable TO alice;
```

```
GRANT UPDATE ON TABLE mytable TO salesrole;

CREATE USER bob password 'md50c983d1a624280812631c5389e60d48c';
```

设置默认身份命名空间的效果是，每个身份都不需要将其作为前缀。在本例中，`alice` 替换为 `MYCO:alice`。如果包含任何身份，就会发生这种情况。有关将身份提供者与 Redshift 结合使用的更多信息，请参阅将 Redshift 与 IAM Identity Center 连接，为用户提供单点登录体验。

有关与 IAM Identity Center 的 Redshift 配置相关的设置的更多信息，请参阅 SET 和 ALTER IDENTITY PROVIDER。

ALTER TABLE

此命令更改 Amazon Redshift 表或 Amazon Redshift Spectrum 外部表的定义。此命令更新 CREATE TABLE 或 CREATE EXTERNAL TABLE 设置的值和属性。

您不能在以下事务块内的外部表上运行 ALTER TABLE (BEGIN ... END)。有关事务的更多信息，请参阅 可序列化的隔离。

除非文档中明确规定可以在表更改时查询表或执行其他操作，否则 ALTER TABLE 会锁定表的读写操作，直到包含 ALTER TABLE 操作的事务完成。

所需的权限

修改表的用户需要适当的权限才能成功执行命令。根据具体的 ALTER TABLE 命令，需要以下权限之一。

- Superuser
- 具有 ALTER TABLE 权限的用户
- 对模式拥有 USAGE 权限的表拥有者

语法

```
ALTER TABLE table_name
{
ADD table_constraint
| DROP CONSTRAINT constraint_name [ RESTRICT | CASCADE ]
| OWNER TO new_owner
| RENAME TO new_name
| RENAME COLUMN column_name TO new_name
| ALTER COLUMN column_name TYPE updated_varchar_data_type_size
```

```
| ALTER COLUMN column_name ENCODE new_encode_type
| ALTER COLUMN column_name ENCODE encode_type,
| ALTER COLUMN column_name ENCODE encode_type, .....;
| ALTER DISTKEY column_name
| ALTER DISTSTYLE ALL
| ALTER DISTSTYLE EVEN
| ALTER DISTSTYLE KEY DISTKEY column_name
| ALTER DISTSTYLE AUTO
| ALTER [COMPOUND] SORTKEY ( column_name [,...] )
| ALTER SORTKEY AUTO
| ALTER SORTKEY NONE
| ALTER ENCODE AUTO
| ADD [ COLUMN ] column_name column_type
  [ DEFAULT default_expr ]
  [ ENCODE encoding ]
  [ NOT NULL | NULL ]
  [ COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE } ] |
| DROP [ COLUMN ] column_name [ RESTRICT | CASCADE ]
| ROW LEVEL SECURITY { ON | OFF } [ CONJUNCTION TYPE { AND | OR } ] [ FOR DATASHARES ]}

where table_constraint is:

[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] )
| PRIMARY KEY ( column_name [, ... ] )
| FOREIGN KEY (column_name [, ... ] )
   REFERENCES  reftable [ ( refcolumn ) ]}

The following options apply only to external tables:

SET LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }
| SET FILE FORMAT format |
| SET TABLE PROPERTIES ('property_name'='property_value')
| PARTITION ( partition_column=partition_value [, ...] )
  SET LOCATION { 's3://bucket/folder' |'s3://bucket/manifest_file' }
| ADD [IF NOT EXISTS]
    PARTITION ( partition_column=partition_value [, ...] ) LOCATION
 { 's3://bucket/folder' |'s3://bucket/manifest_file' }
    [, ... ]
| DROP PARTITION ( partition_column=partition_value [, ...] )
```

要减少运行 ALTER TABLE 命令的时间，可以结合使用 ALTER TABLE 命令的一些子句。

Amazon Redshift 支持以下 ALTER TABLE 子句组合：

```
ALTER TABLE tablename ALTER SORTKEY (column_list), ALTER DISTKEY column_Id;
ALTER TABLE tablename ALTER DISTKEY column_Id, ALTER SORTKEY (column_list);
ALTER TABLE tablename ALTER SORTKEY (column_list), ALTER DISTSTYLE ALL;
ALTER TABLE tablename ALTER DISTSTYLE ALL, ALTER SORTKEY (column_list);
```

参数

table_name

要修改的表的名称。只指定表的名称，或者通过格式 schema_name.table_name 使用特定 schema。外部表必须通过一个外部 schema 名称进行限定。如果您使用 ALTER TABLE 语句重命名视图或更改其拥有者，您还可以指定视图名称。表名称的最大长度为 127 个字节；更长的名称将被截断为 127 个字节。您可以使用 UTF-8 多字节字符，每个字符最多为四个字节。有关有效名称的更多信息，请参阅名称和标识符。

ADD table_constraint

用于将指定约束添加到表的子句。有关有效 table_constraint 值的描述，请参阅 CREATE TABLE。

> **Note**
>
> 您不能将主键约束添加到可为空的列。如果列最初是使用 NOT NULL 约束创建的，您可以添加主键约束。

DROP CONSTRAINT constraint_name

用于从表中删除指定约束的子句。要删除约束，请指定约束名称而不是约束类型。要查看表约束名称，请运行以下查询。

```
select constraint_name, constraint_type
from information_schema.table_constraints;
```

RESTRICT

用于仅删除指定约束的子句。RESTRICT 是 DROP CONSTRAINT 的一个选项。RESTRICT 不能与 CASCADE 一起使用。

CASCADE

用于删除指定约束和依赖于该约束的任何内容的子句。CASCADE 是 DROP CONSTRAINT 的选项。CASCADE 不能与 RESTRICT 一起使用。

OWNER TO new_owner

用于将表（或视图）的所有者更改为 new_owner 值的子句。

RENAME TO new_name

用于将表（或视图）重命名为 new_name 中指定的值的子句。表名称的最大长度为 127 个字节；更长的名称将被截断为 127 个字节。

您不能将永久表重命名为以“#”开头的表。名称以“#”开头的表是临时表。

您无法重命名外部表。

ALTER COLUMN column_name TYPE updated_varchar_data_type_size

这是一个更改定义为 VARCHAR 数据类型的列大小的子句。此子句仅支持修改 VARCHAR 数据类型的大小。请考虑以下限制：

- 您无法修改具有 BYTEDICT、RUNLENGTH、TEXT255 或 TEXT32K 压缩编码的列。
- 您无法将大小减少到小于现有数据的最大大小。
- 您无法更改具有默认值的列。
- 您无法更改具有 UNIQUE、PRIMARY KEY 或 FOREIGN KEY 的列。
- 您不能更改以下事务块中的列：(BEGIN ... END)。有关事务的更多信息，请参阅 可序列化的隔离。

ALTER COLUMN column_name ENCODE new_encode_type

更改列的压缩编码的子句。如果为列指定压缩编码，则表不再设置为 ENCODE AUTO。有关压缩编码的信息，请参阅使用列压缩。

在更改列的压缩编码时，表仍可供查询。

请考虑以下限制：

- 您不能将列更改为与当前为该列定义的相同的编码。
- 不能使用交错排序键更改表中列的编码。

ALTER COLUMN column_name ENCODE encode_type, ALTER COLUMN column_name ENCODE encode_type, ...;

更改单个命令中多个列的压缩编码的子句。有关压缩编码的信息，请参阅使用列压缩。

在更改列的压缩编码时，表仍可供查询。

请考虑以下限制：

- 您不能在单个命令中多次将列更改为相同或不同的编码类型。
- 您不能将列更改为与当前为该列定义的相同的编码。
- 不能使用交错排序键更改表中列的编码。

ALTER DISTSTYLE ALL

用于将表的现有分配样式更改为 ALL 的子句。请考虑以下事项：

- ALTER DISTSTYLE、ALTER SORTKEY 和 VACUUM 不能同时对同一个表运行。
  - 如果 VACUUM 当前正在运行，则运行 ALTER DISTSTYLE ALL 将返回错误。
  - 如果 ALTER DISTSTYLE ALL 正在运行，则不在表上启动后台 vacuum。
- 对于具有交错排序键和临时表的表，不支持 ALTER DISTSTYLE ALL 命令。
- 如果分配方式以前被定义为 AUTO，则表不再是自动表优化的候选项。

有关 DISTSTYLE ALL 的更多信息，请参阅 CREATE TABLE。

ALTER DISTSTYLE EVEN

用于将表的现有分配样式更改为 EVEN 的子句。请考虑以下事项：

- ALTER DISTSYTLE、ALTER SORTKEY 和 VACUUM 不能同时对同一个表运行。
  - 如果 VACUUM 当前正在运行，则运行 ALTER DISTSTYLE EVEN 将返回错误。
  - 如果 ALTER DISTSTYLE EVEN 正在运行，则不在表上启动后台 Vacuum。
- 对于具有交错排序键和临时表的表，不支持 ALTER DISTSTYLE EVEN 命令。
- 如果分配方式以前被定义为 AUTO，则表不再是自动表优化的候选项。

有关 DISTSTYLE EVEN 的更多信息，请参阅 CREATE TABLE。

ALTER DISTKEY column_name 或 ALTER DISTSTYLE KEY DISTKEY column_name

一个子句，可更改用作表的分配键的列。请考虑以下事项：

- 不能在同一个表上并发运行 VACUUM 和 ALTER DISTKEY。
  - 如果 VACUUM 已经运行，则 ALTER DISTKEY 返回错误。
  - 如果 ALTER DISTKEY 正在运行，则不在表上启动后台 Vacuum。
  - 如果 ALTER DISTKEY 正在运行，则前台 vacuum 会返回错误。
- 您一次只能在一个表上运行一个 ALTER DISTKEY 命令。
- 具有交错排序键的表不支持 ALTER DISTKEY 命令。

- 如果分配方式以前被定义为 AUTO，则表不再是自动表优化的候选项。

指定 DISTSTYLE KEY 时，按 DISTKEY 列中的值分配数据。有关 DISTSTYLE 的更多信息，请参阅 CREATE TABLE。

ALTER DISTSTYLE AUTO

用于将表的现有分配方式更改为 AUTO 的子句。

将分配方式更改为 AUTO 时，表的分配模式设置为以下内容：

- 带有 DISTSTYLE ALL 的小型表被转换为 AUTO(ALL)。
- 带有 DISTSTYLE EVEN 的小型表被转换为 AUTO(ALL)。
- 带有 DISTSTYLE KEY 的小型表被转换为 AUTO(ALL)。
- 带有 DISTSTYLE ALL 的大型表被转换为 AUTO(EVEN)。
- 带有 DISTSTYLE EVEN 的大型表被转换为 AUTO(EVEN)。
- 带有 DISTSTYLE KEY 的大型表被转换为 AUTO(KEY)，且 DISTKEY 被保留。在这种情况下，Amazon Redshift 不对表进行任何更改。

如果 Amazon Redshift 确定新的分配方式或密钥将提高查询的性能，那么 Amazon Redshift 将来可能会更改您的表格的分配方式或键。例如，Amazon Redshift 可能会将 DISTSTYLE 为 AUTO(KEY) 的表转换为 AUTO(EVEN)，反之亦然。有关分发密钥被更改时行为的更多信息（包括数据重新分发和锁定），请参阅 Amazon Redshift Advisor 建议。

有关 DISTSTYLE AUTO 的更多信息，请参阅CREATE TABLE。

要查看表的分配方式，请查询 SVV_TABLE_INFO 系统目录视图。有关更多信息，请参阅 SVV_TABLE_INFO。要查看 Amazon Redshift Advisor 对表的建议，请查询 SVV_ALTER_TABLE_RECOMMENDATIONS 系统目录视图。有关更多信息，请参阅 SVV_ALTER_TABLE_RECOMMENDATIONS。要查看 Amazon Redshift 所采取的操作，请查询 SVL_AUTO_WORKER_ACTION 系统目录视图。有关更多信息，请参阅 SVL_AUTO_WORKER_ACTION。

ALTER [COMPOUND] SORTKEY (column_name [,...])

一个旨在更改或添加用于表的排序键的子句。

当您更改排序键时，新排序键或原始排序键中列的压缩编码可能会更改。如果没有为表显式定义编码，则 Amazon Redshift 按如下方式自动分配压缩编码：

- 为定义为排序键的列分配 RAW 压缩。

- 为定义为 BOOLEAN、REAL 或 DOUBLE PRECISION 数据类型的列分配 RAW 压缩。
- 定义为 SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZT、IMESTAMP 或 TIMESTAMPTZ 的列分配了 AZ64 压缩。
- 定义为 CHAR 或 VARCHAR 的列分配了 LZO 压缩。

请考虑以下事项：

- 最多可以为每个表的排序键定义 400 列。
- 您可以将交错排序键更改为复合排序键或没有排序键。但是，您不能将复合排序键更改为交错排序键。
- 如果排序键以前被定义为 AUTO，则表不再是自动表优化的候选项。
- Amazon Redshift 建议对定义为排序键的列使用 RAW 编码（不压缩）。当您更改列以将其选择为排序键时，列的压缩将更改为 RAW 压缩（无压缩）。这将增加表所需的存储空间。表大小的增加程度取决于特定的表定义和表格内容。有关压缩的更多信息，请参阅压缩编码

将数据加载到表中时，将按照排序键的顺序加载数据。更改排序键时，Amazon Redshift 对数据重新排序。有关 SORTKEY 的更多信息，请参阅 CREATE TABLE。

ALTER SORTKEY AUTO

一个可更改目标表的排序键或将其添加到 AUTO 的子句。

当您将排序键更改为 AUTO 时，Amazon Redshift 会保留表的现有排序键。

如果 Amazon Redshift 确定新的排序键将提高查询的性能，那么 Amazon Redshift 将来可能会更改您的表的排序键。

有关 SORTKEY AUTO 的更多信息，请参阅CREATE TABLE。

要查看表的排序键，请查询 SVV_TABLE_INFO 系统目录视图。有关更多信息，请参阅 SVV_TABLE_INFO。要查看 Amazon Redshift Advisor 对表的建议，请查询 SVV_ALTER_TABLE_RECOMMENDATIONS 系统目录视图。有关更多信息，请参阅 SVV_ALTER_TABLE_RECOMMENDATIONS。要查看 Amazon Redshift 所采取的操作，请查询 SVL_AUTO_WORKER_ACTION 系统目录视图。有关更多信息，请参阅 SVL_AUTO_WORKER_ACTION。

ALTER SORTKEY NONE

删除目标表的排序键的子句。

如果排序键以前被定义为 AUTO，则表不再是自动表优化的候选项。

ALTER ENCODE AUTO

将目标表列的编码类型更改为 AUTO 的子句。当您将编码更改为 AUTO 时，Amazon Redshift 会保留表中列的现有编码类型。然后，如果 Amazon Redshift 确定新的编码类型可以提高查询性能，则 Amazon Redshift 可以更改表列的编码类型。

如果您更改一个或多个列以指定编码，Amazon Redshift 将不再自动调整表中所有列的编码。这些列保留当前的编码设置。

以下操作不会影响表的 ENCODE AUTO 设置：

- 重命名表。
- 更改表的 DISTSTYLE 或 SORTKEY 设置。
- 使用 ENCODE 设置添加或删除列。
- 使用 COPY 命令的 COMPUPDATE 选项。有关更多信息，请参阅 数据加载操作。

要查看表的编码，请查询 SVV_TABLE_INFO 系统目录视图。有关更多信息，请参阅 SVV_TABLE_INFO。

RENAME COLUMN column_name TO new_name

用于将列重命名为 new_name 中指定的值的子句。列名称的最大长度为 127 个字节；更长的名称将被截断为 127 个字节。有关有效名称的更多信息，请参阅名称和标识符。

ADD [COLUMN] column_name

用于将具有指定名称的列添加到表的子句。您在每个 ALTER TABLE 语句中只能修改一列。

您无法添加用作表的分配键 (DISTKEY) 或排序键 (SORTKEY) 的列。

您不能使用 ALTER TABLE ADD COLUMN 命令修改以下表和列属性：

- UNIQUE
- PRIMARY KEY
- REFERENCES（外键）
- IDENTITY 或 GENERATED BY DEFAULT AS IDENTITY

列名称的最大长度为 127 个字节；更长的名称将被截断为 127 个字节。可在单个表中定义的列的最大数目为 1,600。

在将列添加到外部表时，会应用以下限制：

- 您无法向列约束为 DEFAULT、ENCODE、NOT NULL 或 NULL 的外部表中添加列。

- 您无法向使用 AVRO 文件格式定义的外部表中添加列。
- 如果启用 pseudocolumns，则可在单个表中定义的最大列数为 1,598。如果未启用 pseudocolumns，则可在单个表中定义的最大列数为 1600。

有关更多信息，请参阅 CREATE EXTERNAL TABLE。

column_type

要添加的列的数据类型。对于 CHAR 和 VARCHAR 列，您可以使用 MAX 关键字而不是声明最大长度。MAX 将最大长度设置为 4096 字节（对于 CHAR）或 65535 字节（对于 VARCHAR）。GEOMETRY 对象的最大大小为 1048447 字节。

有关 Amazon Redshift 支持的数据类型的信息，请参阅数据类型。

DEFAULT default_expr

用于为列分配默认数据值的子句。default_expr 的数据类型必须匹配列的数据类型。DEFAULT 值必须是无变量的表达式。不允许子查询、对当前表中其他列的交叉引用和用户定义的函数。

default_expr 在未为列指定值的任何 INSERT 操作中使用。如果未指定默认值，则列的默认值为 null。

如果 COPY 操作在具有 DEFAULT 值和 NOT NULL 约束的列上遇到空字段，则 COPY 命令将插入 default_expr 值。

外部表不支持 DEFAULT。

ENCODE encoding

列的压缩编码。预设情况下，如果您没有为表中的任何列指定压缩编码，或者您为表指定了 ENCODE AUTO 选项，Amazon Redshift 会自动管理表中所有列的压缩编码。

如果您为表中的任何列指定压缩编码，或者您没有为表指定 ENCODE AUTO 选项，Amazon Redshift 会自动将压缩编码分配给您未指定压缩编码的列，如下所示：

- 默认情况下，会为临时表中的所有列分配 RAW 压缩。
- 为定义为排序键的列分配 RAW 压缩。
- 定义为 BOOLEAN、REAL、DOUBLE PRECISION、GEOMETRY 或 GEOGRAPHY 数据类型的列分配了 RAW 压缩。
- 定义为 SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP 或 TIMESTAMPTZ 的列分配了 AZ64 压缩。
- 定义为 CHAR、VARCHAR 或 VARBYTE 的列分配了 LZO 压缩。

> **Note**
>
> 如果您不希望压缩某个列，请显式指定 RAW 编码。

支持以下compression encodings (p. 55)：

- AZ64
- BYTEDICT
- DELTA
- DELTA32K
- LZO
- MOSTLY8
- MOSTLY16
- MOSTLY32
- RAW（无压缩）
- RUNLENGTH
- TEXT255
- TEXT32K
- ZSTD

外部表不支持 ENCODE。

NOT NULL | NULL

NOT NULL 指定列中不允许包含 null 值。NULL（默认值）指定列接受 null 值。

外部表不支持 NOT NULL 和 NULL。

COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }

指定列中的字符串搜索或比较是 CASE_SENSITIVE 还是 CASE_INSENSITIVE 的子句。默认值与数据库的当前区分大小写的配置相同。

要查找数据库排序规则信息，请使用以下命令：

```
SELECT db_collation();

db_collation
```

```
----------------
 case_sensitive
(1 row)
```

DROP [COLUMN] column_name

要从表中删除的列的名称。

您无法删除表中的最后一列。表必须有至少一列。

您无法删除用作表的分配键 (DISTKEY) 或排序键 (SORTKEY) 的列。如果列具有任何从属对象，例如视图、主键、外键或 UNIQUE 限制，则 DROP COLUMN 的默认行为是 RESTRICT。

对外部表中的列执行 DROP 时，会应用以下限制：

- 如果列用作分区，则您无法哦那个外部表中删除列。
- 您无法从使用 AVRO 文件格式定义的外部表中删除列。
- 对于外部表，将会忽略 RESTRICT 和 CASCADE。
- 除非删除或分离策略，否则您无法删除策略定义中引用的策略表中的列。在指定 CASCADE 选项时，这也将适用。您可以删除策略表中的其他列。

有关更多信息，请参阅 CREATE EXTERNAL TABLE。

RESTRICT

在下面这些情况下，与 DROP COLUMN 一起使用时，RESTRICT 表示不删除要删除的列：

- 定义的视图引用了要删除的列
- 外键引用了该列
- 该列参与了多部分键

RESTRICT 不能与 CASCADE 一起使用。

对于外部表，将会忽略 RESTRICT 和 CASCADE。

CASCADE

在与 DROP COLUMN 配合使用时，删除指定的列以及依赖该列的任何内容。CASCADE 不能与 RESTRICT 一起使用。

对于外部表，将会忽略 RESTRICT 和 CASCADE。

以下选项仅适用于外部表。

SET LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }

包含数据文件的 Amazon S3 文件夹的路径或包含 Amazon S3 对象路径列表的清单文件。桶必须与 Amazon Redshift 集群位于同一 AWS 区域。有关受支持的 AWS 区域的列表，请参阅Amazon Redshift Spectrum 注意事项。有关使用清单文件的更多信息，请参阅 CREATE EXTERNAL TABLE 参数 参考中的 LOCATION。

SET FILE FORMAT format

外部数据文件的文件格式。

有效格式如下所示：

- AVRO
- PARQUET
- RCFILE
- SEQUENCEFILE
- TEXTFILE

SET TABLE PROPERTIES ('property_name'='property_value')

一个子句，用于设置外部表的表属性的表定义。

> **Note**
>
> 表属性区分大小写。

'numRows'='row_count'

用于为表定义设置 numRows 值的属性。若要显式更新外部表的统计数据，请设置 numRows 属性来指示表的大小。Amazon Redshift 不分析外部表来生成表统计数据，查询优化程序会使用这些统计数据来生成查询计划。如果没有为外部表设置表统计数据，则 Amazon Redshift 会生成查询执行计划。此计划是基于“外部表较大，本地表较小”的假设生成的。

'skip.header.line.count'='line_count'

用于设置在每个源文件开头要跳过的行数的属性。

PARTITION (partition_column=partition_value [, ...] SET LOCATION { 's3://bucket/folder' | 's3://bucket/manifest_file' }

用于为一个或多个分区列设置新位置的子句。

ADD [IF NOT EXISTS] PARTITION (partition_column=partition_value [, ...]) LOCATION { 's3://bucket/folder' | 's3://bucket/manifest_file' } [, ...]

用于添加一个或多个分区的子句。您可以使用单个 ALTER TABLE … ADD 语句指定多个 PARTITION 子句。

> **Note**
>
> 如果使用 AWS Glue 目录，则可以使用单个 ALTER TABLE 语句最多添加 100 个分区。

IF NOT EXISTS 子句指示，如果指定的分区已存在，命令应不进行任何更改。它还指示该命令应返回分区存在的消息，而不是以错误终止。此子句在编写脚本时很有用，可使脚本在 ALTER TABLE 尝试添加已存在的分区时不会失败。

DROP PARTITION (partition_column=partition_value [, ...])

用于删除指定分区的子句。删除分区只会更改外部表元数据。Amazon S3 上的数据不受影响。

ROW LEVEL SECURITY { ON | OFF } [CONJUNCTION TYPE { AND | OR }] [FOR DATASHARES]

一个对关系开启或关闭行级安全性的子句。

在为关系开启行级安全性后，您只能读取行级安全策略允许您访问的行。如果没有策略向您授予对关系的访问权限，您将看不到关系中的任何行。只有超级用户和拥有 `sys:secadmin` 角色的用户或角色才能设置 ROW LEVEL SECURITY 子句。有关更多信息，请参阅 行级别安全性。

- [CONJUNCTION TYPE { AND | OR }]

  一个允许您为关系选择行级安全策略的联接类型的子句。将多个行级安全策略附加到关系时，可以将这些策略与 AND 或 OR 子句合并。默认情况下，Amazon Redshift 将 RLS 策略与 AND 子句合并。具有 `sys:secadmin` 角色的超级用户、用户或角色可以使用此子句为关系定义行级安全策略的联接类型。有关更多信息，请参阅 为每个用户组合多个策略。

- FOR DATASHARES

  一个子句，用于确定是否可以通过数据共享访问受 RLS 保护的关系。默认情况下，无法通过数据共享访问受 RLS 保护的关系。使用此子句运行的 ALTER TABLE ROW LEVEL SECURITY 命令只会影响关系的数据共享可访问性属性。ROW LEVEL SECURITY 属性未更改。

  如果您允许通过数据共享访问受 RLS 保护的关系，则该关系在使用者端数据共享数据库中没有行级安全性。该关系在生产者端保留其 RLS 属性。

示例

有关说明 ALTER TABLE 命令用法的示例，请参阅以下内容。

- ALTER TABLE 示例
- ALTER EXTERNAL TABLE 示例
- ALTER TABLE ADD 和 DROP COLUMN 示例

ALTER TABLE 示例

以下示例演示了 ALTER TABLE 命令的基本用法。

重命名表或视图

以下命令将 USERS 表重命名为 USERS_BKUP：

```
alter table users
rename to users_bkup;
```

您还可以使用此类型的命令来重命名视图。

更改表或视图的所有者

以下命令将 VENUE 表所有者更改为用户 DWUSER：

```
alter table venue
owner to dwuser;
```

以下命令创建一个视图，然后更改其所有者：

```
create view vdate as select * from date;
alter table vdate owner to vuser;
```

重命名列

以下命令将 VENUE 表中的 VENUESEATS 列重命名为 VENUESIZE：

```
alter table venue
rename column venueseats to venuesize;
```

删除表约束

要删除表约束（例如主键、外键或唯一约束），请先查找约束的内部名称。然后在 ALTER TABLE 命令中指定约束名称。以下示例查找 CATEGORY 表的约束，然后删除名为 category_pkey 的主键。

```
select constraint_name, constraint_type
from information_schema.table_constraints
where constraint_schema ='public'
and table_name = 'category';

constraint_name | constraint_type
----------------+----------------
category_pkey   | PRIMARY KEY

alter table category
drop constraint category_pkey;
```

更改 VARCHAR 列

为了节省存储空间，您最初可以使用 VARCHAR 列定义一个表，这些列具有满足当前数据要求所需的最小大小。以后，要容纳更长的字符串，您可以更改表以增加列大小。

以下示例将 EVENTNAME 列大小增加到 VARCHAR(300)。

```
alter table event alter column eventname type varchar(300);
```

更改列的压缩编码

您可以更改列的压缩编码。您可以在下面找到一组演示此方法的示例。这些示例的表定义如下。

```
create table t1(c0 int encode lzo, c1 bigint encode zstd, c2 varchar(16) encode lzo, c3
 varchar(32) encode zstd);
```

以下语句将列 c0 的压缩编码从 LZO 编码更改为 AZ64 编码。

```
alter table t1 alter column c0 encode az64;
```

以下语句将列 c1 的压缩编码从 Zstandard 编码更改为 AZ64 编码。

```
alter table t1 alter column c1 encode az64;
```

以下语句将列 c2 的压缩编码从 LZO 编码更改为 Byte-dictionary 编码。

```
alter table t1 alter column c2 encode bytedict;
```

以下语句将列 c3 的压缩编码从 Zstandard 编码更改为 Runlength 编码。

```
alter table t1 alter column c3 encode runlength;
```

修改 DISTSTYLE KEY DISTKEY 列

以下示例显示如何更改表的 DISTSTYLE 和 DISTKEY。

使用 EVEN 分配样式创建表 SVV_TABLE_INFO 视图显示 DISTSTYLE 为 EVEN。

```
create table inventory(
  inv_date_sk int4 not null ,
  inv_item_sk int4 not null ,
  inv_warehouse_sk int4 not null ,
  inv_quantity_on_hand int4
) diststyle even;

Insert into inventory values(1,1,1,1);

select "table", "diststyle" from svv_table_info;

   table   |   diststyle
-----------+----------------
 inventory |     EVEN
```

将表 DISTKEY 更改为 inv_warehouse_sk。SVV_TABLE_INFO 视图将 inv_warehouse_sk 列显示为结果分配密钥。

```
alter table inventory alter diststyle key distkey inv_warehouse_sk;

select "table", "diststyle" from svv_table_info;

   table   |       diststyle
-----------+----------------------
 inventory | KEY(inv_warehouse_sk)
```

将表 DISTKEY 更改为 inv_item_sk。SVV_TABLE_INFO 视图将 inv_item_sk 列显示为结果分配密钥。

```
alter table inventory alter distkey inv_item_sk;

select "table", "diststyle" from svv_table_info;

   table   |       diststyle
-----------+----------------------
 inventory | KEY(inv_item_sk)
```

将表更改为 DISTSTYLE ALL

以下示例演示如何将表更改为 DISTSTYLE ALL。

使用 EVEN 分配样式创建表 SVV_TABLE_INFO 视图显示 DISTSTYLE 为 EVEN。

```
create table inventory(
  inv_date_sk int4 not null ,
  inv_item_sk int4 not null ,
  inv_warehouse_sk int4 not null ,
  inv_quantity_on_hand int4
) diststyle even;

Insert into inventory values(1,1,1,1);

select "table", "diststyle" from svv_table_info;

   table   |   diststyle
-----------+---------------
 inventory |     EVEN
```

将表 DISTSTYLE 更改为 ALL。SVV_TABLE_INFO 视图显示已更改的 DISTSYTLE。

```
alter table inventory alter diststyle all;

select "table", "diststyle" from svv_table_info;

   table   |   diststyle
-----------+---------------
 inventory |     ALL
```

更改表 SORTKEY

您可以将表更改为具有复合排序键或没有排序键。

在以下表定义中，表 t1 是使用交错排序键定义的。

```
create table t1 (c0 int, c1 int) interleaved sortkey(c0, c1);
```

以下命令将表从交错排序键更改为复合排序键。

```
alter table t1 alter sortkey(c0, c1);
```

以下命令对表进行了修改，删除了交错排序键。

```
alter table t1 alter sortkey none;
```

在以下表定义中，表 t1 将列 c0 定义为排序键。

```
create table t1 (c0 int, c1 int) sortkey(c0);
```

以下命令将表 t1 更改为复合排序键。

```
alter table t1 alter sortkey(c0, c1);
```

将表更改为 ENCODE AUTO

以下示例说明如何将表更改为 ENCODE AUTO。

此示例的表定义如下。列 c0 使用编码类型 AZ64 进行定义，列 c1 使用编码类型 LZO 定义。

```
create table t1(c0 int encode AZ64, c1 varchar encode LZO);
```

对于此表，以下语句将编码更改为 AUTO。

```
alter table t1 alter encode auto;
```

以下示例说明如何将表更改为删除 ENCODE AUTO 设置。

此示例的表定义如下。表列没有定义编码。在这种情况下，编码默认为 ENCODE AUTO。

```
create table t2(c0 int, c1 varchar);
```

对于此表，以下语句将 c0 列的编码更改为 LZO。表编码不再设置为 ENCODE AUTO。

```
alter table t2 alter column c0 encode lzo;;
```

修改行级别安全性控制

以下命令将对表关闭 RLS：

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY OFF;
```

以下命令将对表开启 RLS：

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;
```

以下命令将对表开启 RLS，使其可通过数据共享进行访问：

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY FOR DATASHARES OFF;
```

以下命令将对表开启 RLS，使其无法通过数据共享进行访问：

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY FOR DATASHARES ON;
```

以下命令将对表开启 RLS，并将表的 RLS 联接类型设置为 OR：

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON CONJUNCTION TYPE OR;
```

以下命令将对表开启 RLS，并将表的 RLS 联接类型设置为 AND：

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON CONJUNCTION TYPE AND;
```

ALTER EXTERNAL TABLE 示例

以下示例使用位于美国东部（弗吉尼亚州北部）区域（`us-east-1`）AWS 区域的 Amazon S3 存储桶，以及在 示例 中为 CREATE TABLE 创建的示例表。有关如何将分区与外部表一起使用的更多信息，请参阅对 Redshift Spectrum 外部表进行分区。

以下示例将 SPECTRUM.SALES 外部表的 numRows 表属性设置为 170000 行。

```
alter table spectrum.sales
```

```
set table properties ('numRows'='170000');
```

以下示例更改 SPECTRUM.SALES 外部表的位置。

```
alter table spectrum.sales
set location 's3://redshift-downloads/tickit/spectrum/sales/';
```

以下示例将 SPECTRUM.SALES 外部表的格式更改为 Parquet。

```
alter table spectrum.sales
set file format parquet;
```

以下示例为表 SPECTRUM.SALES_PART 添加一个分区。

```
alter table spectrum.sales_part
add if not exists partition(saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/';
```

以下示例为表 SPECTRUM.SALES_PART 添加三个分区。

```
alter table spectrum.sales_part add if not exists
partition(saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/'
partition(saledate='2008-02-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/'
partition(saledate='2008-03-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/';
```

以下示例修改 SPECTRUM.SALES_PART 以删除包含 `saledate='2008-01-01''` 的分区。

```
alter table spectrum.sales_part
drop partition(saledate='2008-01-01');
```

以下示例为包含 `saledate='2008-01-01'` 的分区设置新的 Amazon S3 路径。

```
alter table spectrum.sales_part
partition(saledate='2008-01-01')
set location 's3://redshift-downloads/tickit/spectrum/sales_partition/
saledate=2008-01-01/';
```

以下示例将 `sales_date` 的名称更改为 `transaction_date`。

```
alter table spectrum.sales rename column sales_date to transaction_date;
```

以下示例将列映射设置为使用优化行列式 (ORC) 格式的外部表的位置映射。

```
alter table spectrum.orc_example
set table properties('orc.schema.resolution'='position');
```

以下示例将列映射设置为使用 ORC 格式的外部表的名称映射。

```
alter table spectrum.orc_example
set table properties('orc.schema.resolution'='name');
```

ALTER TABLE ADD 和 DROP COLUMN 示例

以下示例演示如何使用 ALTER TABLE 添加基本表列，然后删除该列；另外还演示如何删除具有从属对象的列。

添加基本列，然后删除该列

以下示例将独立 FEEDBACK_SCORE 列添加到 USERS 表。该列只包含一个整数，并且该列的默认值为 NULL（无反馈分数）。

首先，查询 PG_TABLE_DEF 目录表以查看 USERS 表的架构：

```
column        | type                    | encoding | distkey | sortkey
--------------+-------------------------+----------+---------+--------
userid        | integer                 | delta    | true    |       1
username      | character(8)            | lzo      | false   |       0
firstname     | character varying(30)   | text32k  | false   |       0
lastname      | character varying(30)   | text32k  | false   |       0
city          | character varying(30)   | text32k  | false   |       0
state         | character(2)            | bytedict | false   |       0
email         | character varying(100)  | lzo      | false   |       0
phone         | character(14)           | lzo      | false   |       0
likesports    | boolean                 | none     | false   |       0
liketheatre   | boolean                 | none     | false   |       0
likeconcerts  | boolean                 | none     | false   |       0
likejazz      | boolean                 | none     | false   |       0
likeclassical | boolean                 | none     | false   |       0
likeopera     | boolean                 | none     | false   |       0
likerock      | boolean                 | none     | false   |       0
likevegas     | boolean                 | none     | false   |       0
```

```
likebroadway  | boolean               | none     | false   |       0
likemusicals  | boolean               | none     | false   |       0
```

现在添加 feedback_score 列：

```
alter table users
add column feedback_score int
default NULL;
```

从 USERS 中选择 FEEDBACK_SCORE 列以验证该列已添加：

```
select feedback_score from users limit 5;

feedback_score
----------------
NULL
NULL
NULL
NULL
NULL
```

删除该列以恢复原始 DDL：

```
alter table users drop column feedback_score;
```

删除具有从属对象的列

以下示例删除具有从属对象的列。结果是，同时删除从属对象。

开始时，将 FEEDBACK_SCORE 列重新添加到 USERS 表：

```
alter table users
add column feedback_score int
default NULL;
```

接下来，从名为 USERS_VIEW 的 USERS 表创建一个视图：

```
create view users_view as select * from users;
```

现在，尝试从 USERS 表中删除 FEEDBACK_SCORE 列。此 DROP 语句使用默认行为 (RESTRICT)：

```
alter table users drop column feedback_score;
```

Amazon Redshift 显示一条错误消息，说明由于另一个对象依赖该列而无法删除该列。

重新尝试删除 FEEDBACK_SCORE 列，这次指定 CASCADE 以删除所有从属对象：

```
alter table users
drop column feedback_score cascade;
```

ALTER TABLE APPEND

通过从现有的源表移动数据，将行附加到目标表。源表中的数据将移到目标表中的匹配列。列的顺序不重要。在成功将数据附加到目标表后，源表变成空表。由于是移动数据而不是复制数据，因此相比类似的 CREATE TABLE AS 或 INSERT INTO 操作，ALTER TABLE APPEND 通常要快得多。

> **Note**
>
> ALTER TABLE APPEND 在源表和目标表之间移动数据块。为了提高性能，ALTER TABLE APPEND 不作为 append 操作的一部分压缩存储。因此，存储使用率会临时增加。要回收空间，请运行 VACUUM 操作。

同名的列还必须具有相同的列属性。如果源表包含目标表中不存在的列（反之亦然），请使用 IGNOREEXTRA 或 FILLTARGET 参数来指定如何管理额外的列。

您不能附加身份列。如果两个表中均包括身份列，则命令会失败。如果只有一个表具有标识列，请包含 FILLTARGET 或 IGNOREEXTRA 参数。有关更多信息，请参阅 ALTER TABLE APPEND 使用说明。

您可以附加 GENERATED BY DEFAULT AS IDENTITY 列。您可以使用您提供的值来更新定义为 GENERATED BY DEFAULT AS IDENTITY 的列。有关更多信息，请参阅 ALTER TABLE APPEND 使用说明。

目标表必须是永久表。但是，源可以是永久表，也可以是配置为流式摄取的实体化视图。如果一个对象定义了分配方式和分配键，则两个对象必须使用相同的分配方式和分配键。如果对对象进行排序，则两个对象必须使用相同的排序方式并定义相同的列作为排序键。

在操作完成之后，ALTER TABLE APPEND 会立即自动提交。该命令不能回滚。您不能在事务数据块 (BEGIN ... END) 中运行 ALTER TABLE APPEND。有关事务的更多信息，请参阅 可序列化的隔离。

所需的权限

根据具体的 ALTER TABLE APPEND 命令，需要以下权限之一：

- Superuser
- 具有 ALTER TABLE 系统权限的用户
- 具有源表的 DELETE 和 SELECT 权限以及目标表的 INSERT 权限的用户

语法

```
ALTER TABLE target_table_name APPEND FROM [ source_table_name
 | source_materialized_view_name ]
[ IGNOREEXTRA | FILLTARGET ]
```

从实体化视图进行追加仅在实体化视图配置为串流摄取的情形下才有效。

参数

target_table_name

要将行附加到的表的名称。只指定表的名称，或者通过格式 schema_name.table_name 使用特定 schema。目标表必须是现有的永久表。

FROM source_table_name

提供要附加的行的表的名称。只指定表的名称，或者通过格式 schema_name.table_name 使用特定 schema。源表必须是现有的永久表。

FROM source_materialized_view_name

提供要附加的行的实体化视图的名称。从实体化视图进行追加仅在实体化视图配置为串流摄取的情形下才有效。源实体化视图必须已经存在。

IGNOREEXTRA

这个关键字指定，如果源表中包含目标表中不存在的列，则应该放弃额外列中的数据。您不能将 IGNOREEXTRA 与 FILLTARGET 配合使用。

FILLTARGET

这个关键字指定，如果目标表中包含源表中不存在的列，则应该使用 DEFAULT 列值（如果已定义列值；否则使用 NULL 值）填充这些列。您不能将 IGNOREEXTRA 与 FILLTARGET 配合使用。

ALTER TABLE APPEND 使用说明

ALTER TABLE APPEND 仅将相同列从源表移到目标表。列的顺序不重要。

如果源表或目标表包含额外的列，则根据以下规则使用 FILLTARGET 或 IGNOREEXTRA：

- 如果源表包含目标表中不存在的列，则包含 IGNOREEXTRA。该命令会忽略源表中额外的列。
- 如果目标表包含源表中不存在的列，则包含 FILLTARGET。该命令使用默认列值或 IDENTITY 值（如果已定义列值；否则使用 NULL 值）填充目标表中的额外列。
- 如果源表和目标表均包含额外的列，则该命令失败。您不能同时使用 FILLTARGET 和 IGNOREEXTRA。

如果两个表中的某个列具有相同名称，但具有不同的属性，则该命令失败。名称类似的列必须具有以下共同的属性：

- 数据类型
- 列大小
- 压缩编码
- 非 Null
- 排序方式
- 排序键列
- 分配方式
- 分配键列

您不能附加身份列。如果源表和目标表中均具有身份列，则该命令失败。如果只有源表具有身份列，则包含 IGNOREEXTRA 参数以忽略身份列。如果只有目标表具有身份列，则包含 FILLTARGET 参数，以便根据为该表定义的 IDENTITY 子句来填充身份列。有关更多信息，请参阅 DEFAULT。

您可以使用 ALTER TABLE APPEND 语句附加默认身份列。有关更多信息，请参阅 CREATE TABLE。

ALTER TABLE APPEND 示例

假设您的组织维护表 SALES_MONTHLY 来获取当前销售交易。您希望每个月将数据从交易表移动到 SALES 表。

您可以使用下面的 INSERT INTO 和 TRUNCATE 命令来完成任务。

```
insert into sales (select * from sales_monthly);
truncate sales_monthly;
```

不过，您可以使用 ALTER TABLE APPEND 命令执行相同的操作，而且效率要高得多。

首先，查询 PG_TABLE_DEF 系统目录表，验证两个表中包含具有相同列属性的相同列。

```
select trim(tablename) as table, "column", trim(type) as type,
encoding, distkey, sortkey, "notnull"
from pg_table_def where tablename like 'sales%';

table      | column     | type                        | encoding | distkey | sortkey |
 notnull
-----------+------------+-----------------------------+----------+---------+---------
+--------
sales      | salesid    | integer                     | lzo      | false   |       0 |
 true
sales      | listid     | integer                     | none     | true    |       1 |
 true
sales      | sellerid   | integer                     | none     | false   |       2 |
 true
sales      | buyerid    | integer                     | lzo      | false   |       0 |
 true
sales      | eventid    | integer                     | mostly16 | false   |       0 |
 true
sales      | dateid     | smallint                    | lzo      | false   |       0 |
 true
sales      | qtysold    | smallint                    | mostly8  | false   |       0 |
 true
sales      | pricepaid  | numeric(8,2)                | delta32k | false   |       0 |
 false
sales      | commission | numeric(8,2)                | delta32k | false   |       0 |
 false
sales      | saletime   | timestamp without time zone | lzo      | false   |       0 |
 false
salesmonth | salesid    | integer                     | lzo      | false   |       0 |
 true
salesmonth | listid     | integer                     | none     | true    |       1 |
 true
salesmonth | sellerid   | integer                     | none     | false   |       2 |
 true
```

```
salesmonth | buyerid    | integer                     | lzo      | false   |       0 |
 true
salesmonth | eventid    | integer                     | mostly16 | false   |       0 |
 true
salesmonth | dateid     | smallint                    | lzo      | false   |       0 |
 true
salesmonth | qtysold    | smallint                    | mostly8  | false   |       0 |
 true
salesmonth | pricepaid  | numeric(8,2)                | delta32k | false   |       0 |
 false
salesmonth | commission | numeric(8,2)                | delta32k | false   |       0 |
 false
salesmonth | saletime   | timestamp without time zone | lzo      | false   |       0 |
 false
```

接下来，查看每个表的大小。

```
select count(*) from sales_monthly;
 count
-------
  2000
(1 row)

select count(*) from sales;
 count
-------
 412,214
(1 row)
```

现在运行以下 ALTER TABLE APPEND 命令。

```
alter table sales append from sales_monthly;
```

再次查看每个表的大小。SALES_MONTHLY 表现在包含 0 行；而 SALES 表增长了 2000 行。

```
select count(*) from sales_monthly;
 count
-------
     0
(1 row)

select count(*) from sales;
```

```
 count
-------
 414214
(1 row)
```

如果源表中的列多于目标表，请指定 IGNOREEXTRA 参数。以下示例使用 IGNOREEXTRA 参数，这样在附加到 SALES 表时，会忽略 SALES_LISTING 表中的额外列。

```
alter table sales append from sales_listing ignoreextra;
```

如果目标表中的列多于源表，请指定 FILLTARGET 参数。以下示例使用 FILLTARGET 参数，以填充 SALES_REPORT 表中存在而 SALES_MONTH 表中不存在的列。

```
alter table sales_report append from sales_month filltarget;
```

以下示例显示了在实体化视图作为源的情况下如何使用 ALTER TABLE APPEND 的示例。

```
ALTER TABLE target_tbl APPEND FROM my_streaming_materialized_view;
```

此示例中的表和实体化视图名称是示例。从实体化视图进行追加仅在实体化视图配置为串流摄取的情形下才有效。它将源实体化视图中的所有记录移动到与实体化视图具有相同架构的目标表中，并保持实体化视图不变。这与数据来源为表时的行为相同。

ALTER USER

更改数据库用户。

所需的权限

以下是 ALTER USER 所需的权限：

- Superuser
- 具有 ALTER USER 权限的用户
- 想更改自己密码的当前用户

语法

```
ALTER USER username [ WITH ] option [, ... ]
```

```
where option is

CREATEDB | NOCREATEDB
| CREATEUSER | NOCREATEUSER
| SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }
| PASSWORD { 'password' | 'md5hash' | DISABLE }
[ VALID UNTIL 'expiration_date' ]
| RENAME TO new_name |
| CONNECTION LIMIT { limit | UNLIMITED }
| SESSION TIMEOUT limit | RESET SESSION TIMEOUT
| SET parameter { TO | = } { value | DEFAULT }
| RESET parameter
| EXTERNALID external_id
```

参数

username

用户的名称。

WITH

可选关键字。

CREATEDB | NOCREATEDB

CREATEDB 选项允许用户创建新数据库。NOCREATEDB 是默认值。

CREATEUSER | NOCREATEUSER

使用 CREATEUSER 选项可以创建具备所有数据库权限的超级用户，包括 CREATE USER。默认值为 NOCREATEUSER。有关更多信息，请参阅 superuser。

SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }

一个子句，它指定用户必须对 Amazon Redshift 系统表和视图具有的访问级别。

拥有 SYSLOG ACCESS RESTRICTED 权限的普通用户在用户可见的系统表和视图中只能查看该用户生成的行。默认值为 RESTRICTED。

拥有 SYSLOG ACCESS UNRESTRICTED 权限的普通用户可以查看用户可见的系统表和视图中的所有行，包括其他用户生成的行。UNRESTRICTED 不向普通用户授予对超级用户可见的表的访问权限。只有超级用户可以查看超级用户可见的表。

> **Note**
>
> 如果向用户授予对系统表的无限制访问权限，用户便可以看到由其他用户生成的数据。例如，STL_QUERY 和 STL_QUERYTEXT 包含 INSERT、UPDATE 和 DELETE 语句的完整文本（其中可能包含敏感的用户生成数据）。

SVV_TRANSACTIONS 中的所有行都对所有用户可见。

有关更多信息，请参阅 系统表和视图中的数据可见性。

PASSWORD { 'password' | 'md5hash' | DISABLE }

设置用户的密码。

默认情况下，用户可以更改自己的密码，除非密码被禁用。要禁用用户的密码，请指定 DISABLE。禁用某个用户的密码后，将从系统中删除该密码，而此用户只能使用临时 AWS Identity and Access Management (IAM) 用户凭证进行登录。有关更多信息，请参阅使用 IAM 身份验证生成数据库用户凭证。只有超级用户才能启用或禁用密码。您不能禁用超级用户的密码。要启用密码，请运行 ALTER USER 并指定密码。

有关使用 PASSWORD 参数的详细信息，请参阅CREATE USER。

VALID UNTIL 'expiration_date'

指定密码具有过期日期。使用值 `'infinity'` 可避免密码具有过期日期。此参数的有效数据类型为时间戳。

只有超级用户才能使用此参数。

RENAME TO

重命名用户。

new_name

用户的新名称。有关有效名称的更多信息，请参阅名称和标识符。

> **Important**
>
> 在重命名用户时，还必须重置用户的密码。重置密码不必与之前的密码不同。用户名用作密码加密的一部分，因此在重命名用户时，将会清除密码。用户将无法登录，直至重置密码。例如：

```
alter user newuser password 'EXAMPLENewPassword11';
```

CONNECTION LIMIT { limit | UNLIMITED }

允许用户同时打开的数据库连接的最大数量。此限制不适用于超级用户。使用 UNLIMITED 关键字设置允许的并行连接的最大数量。对每个数据库的连接数量可能也会施加限制。有关更多信息，请参阅 CREATE DATABASE。默认为 UNLIMITED。要查看当前连接，请查询 STV_SESSIONS 系统视图。

> **Note**
>
> 如果用户及数据库连接限制均适用，当用户尝试连接时，必须有一个同时低于这两个限制的未使用的连接槽可用。

SESSION TIMEOUT limit | RESET SESSION TIMEOUT

会话保持非活动或空闲状态的最长时间（秒）。范围在 60 秒（1 分钟）到 1,728,000 秒（20 天）之间。如果没有为用户设置会话超时，则应用集群设置。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 中的配额和限制。

设置会话超时时，它仅应用于新会话。

要查看有关活动用户会话的信息，包括开始时间、用户名和会话超时，请查询 STV_SESSIONS 系统视图。要查看有关用户会话历史记录的信息，请查询 STL_SESSIONS 视图。要检索有关数据库用户的信息（包括会话超时值），请查询 SVL_USER_INFO 视图。

SET

针对指定用户运行的所有会话，将某个配置参数设置为新的默认值。

RESET

为指定用户将某个配置参数重置为原始默认值。

参数

要设置或重置的参数的名称。

值

参数的新值。

DEFAULT

针对指定用户运行的所有会话，将配置参数设置为默认值。

EXTERNALID external_id

用户的标识符，与身份提供者关联。用户必须已禁用密码。有关更多信息，请参阅 Amazon Redshift 的原生身份提供者 (IdP) 联合身份验证。

使用说明

- 尝试更改 rdsdb - 您无法更改名为 `rdsdb` 的用户。
- 创建未知密码 – 使用 AWS Identity and Access Management (IAM) 身份验证创建数据库用户凭证时，您可能希望创建仅使用临时凭证便可以登录的超级用户。您不能禁用超级用户的密码，但可以使用随机生成的 MD5 哈希字符串创建一个未知密码。

```
alter user iam_superuser password 'md5123456789012345678012345678901 2';
```

- 设置 search_path – 当您使用 ALTER USER 命令设置 search_path 参数时，所做修改将在指定的用户下次登录时生效。如果您希望更改当前用户和会话的 search_path 值，请使用 SET 命令。
- 设置时区 – 当您将 SET TIMEZONE 与 ALTER USER 命令一起使用时，所做修改将在指定的用户下次登录时生效。
- 使用动态数据掩蔽和行级安全策略 – 当您的预调配集群或无服务器命名空间具有任何动态数据掩蔽或行级安全策略时，普通用户将无法使用以下命令：

```
ALTER <current_user> SET enable_case_sensitive_super_attribute/
enable_case_sensitive_identifier/downcase_delimited_identifier
```

只有超级用户和具有 ALTER USER 权限的用户才能设置这些配置选项。有关行级别安全性的信息，请参阅行级别安全性。有关动态数据掩蔽的信息，请参阅动态数据掩蔽。

示例

以下示例为用户 ADMIN 授予创建数据库的权限：

```
alter user admin createdb;
```

以下示例将用户 ADMIN 的密码设置为 `adminPass9`，并为密码设置一个到期日期和时间：

```
alter user admin password 'adminPass9'
valid until '2017-12-31 23:59';
```

以下示例将用户 ADMIN 重命名为 SYSADMIN：

```
alter user admin rename to sysadmin;
```

以下示例将用户的空闲会话超时更新为 300 秒。

```
ALTER USER dbuser SESSION TIMEOUT 300;
```

重置用户的空闲会话超时。重置它时，将应用集群设置。您必须是数据库超级用户才能执行此命令。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 中的配额和限制。

```
ALTER USER dbuser RESET SESSION TIMEOUT;
```

以下示例更新名为 bob 的用户的外部 ID。命名空间为 myco_aad。如果命名空间与注册的身份提供者没有关联，则会导致错误。

```
ALTER USER myco_aad:bob EXTERNALID "ABC123" PASSWORD DISABLE;
```

以下示例为某个特定数据库用户运行的所有会话设置时区。该示例更改后续会话的时区，而不是更改当前会话的时区。

```
ALTER USER odie SET TIMEZONE TO 'Europe/Zurich';
```

以下示例设置了允许用户 bob 打开的最大数据库连接数。

```
ALTER USER bob CONNECTION LIMIT 10;
```

ANALYZE

更新表统计数据以供查询计划程序使用。

所需的权限

以下是 ANALYZE 所需的权限：

- Superuser

- 具有 ANALYZE 权限的用户
- 关系的拥有者
- 向其共享表的数据库拥有者

语法

```
ANALYZE [ VERBOSE ]
[ [ table_name [ ( column_name [, ...] ) ] ]
[ PREDICATE COLUMNS | ALL  COLUMNS ]
```

参数

VERBOSE

返回有关 ANALYZE 操作的进度信息消息的子句。在不指定表时，此选项会非常有用。

table_name

您可以分析特定表，包括临时表。您可以通过其 schema 名称来限定表。您可以视需要指定 table_name 来分析单个表。您不能在单个 ANALYZE table_name 语句中指定多个 table_name。如果您不指定 table_name 值，则将分析当前连接的数据库中的所有表，包括系统目录中的持久性表。如果自上次执行 ANALYZE 以来更改的行数百分比低于分析阈值，Amazon Redshift 将跳过对表的分析。有关更多信息，请参阅 分析阈值。

您不需要分析 Amazon Redshift 系统表（STL 和 STV 表）。

column_name

如果您指定 table_name，您还可以指定表中的一个或多个列（以两旁加括号的逗号分隔列表的形式）。如果指定了列列表，则只分析列出的列。

PREDICATE COLUMNS | ALL COLUMNS

用于指示 ANALYZE 是否应仅包含谓词列的子句。指定 PREDICATE COLUMNS 以仅分析在以前查询中用作谓词的列或可能用作谓词的候选列。指定 ALL COLUMNS 将分析所有列。默认值为 ALL COLUMNS。

如果以下任意一项为 true，则列包括在一组谓词列中。

- 该列已在查询中用作筛选条件、联接条件或 group by 子句的一部分。
- 该列为分配键。
- 该列为排序键的一部分。

如果未将任何列标记为谓词列（例如，由于尚未查询表），则即使指定了 PREDICATE COLUMNS，也仍将分析所有列。有关谓词列的更多信息，请参阅分析表。

使用说明

Amazon Redshift 自动对使用以下命令创建的表运行 ANALYZE：

- CREATE TABLE AS
- CREATE TEMP TABLE AS
- SELECT INTO

您无法分析外部表。

在最初创建这些表时，您不需要对这些表运行 ANALYZE 命令。如果修改了这些表，则应通过用来分析其他表的方式来分析这些表。

分析阈值

为了减少处理时间并提高整体系统性能，在自上次运行 ANALYZE 命令以来更改的行数百分比低于 analyze_threshold_percent 参数所指定分析阈值的情况下，Amazon Redshift 将跳过对表执行的 ANALYZE 操作。默认情况下，analyze_threshold_percent 为 10。要更改当前会话的 analyze_threshold_percent，请执行 SET 命令。以下示例将 analyze_threshold_percent 更改为 20%。

```
set analyze_threshold_percent to 20;
```

要在仅有少量行发生更改时对表进行分析，请将 analyze_threshold_percent 设置为任意较小的数字。例如，如果您将 analyze_threshold_percent 设置为 0.01，则在含有 100000000 行的表中至少有 10000 行发生更改时，不会跳过该表。

```
set analyze_threshold_percent to 0.01;
```

如果 ANALYZE 因未达到分析阈值而跳过表，Amazon Redshift 将返回以下消息。

```
ANALYZE SKIP
```

要在即使没有任何行发生更改时仍对所有表进行分析，请将 analyze_threshold_percent 设置为 0。

要查看 ANALYZE 操作的结果，请查询 STL_ANALYZE 系统表。

有关分析表的更多信息，请参阅分析表。

示例

分析 TICKIT 数据库中的所有表，并返回进度信息。

```
analyze verbose;
```

只分析 LISTING 表。

```
analyze listing;
```

分析 VENUE 表中的 VENUEID 和 VENUENAME 列。

```
analyze venue(venueid, venuename);
```

仅分析 VENUE 表中的谓词列。

```
analyze venue predicate columns;
```

ANALYZE COMPRESSION

执行压缩分析，并针对所分析的表使用建议的压缩编码生成报告。报告会针对每一列估计与原始编码相比磁盘空间的潜在压缩量。

语法

```
ANALYZE COMPRESSION
[ [ table_name ]
[ ( column_name [, ...] ) ] ]
[COMPROWS numrows]
```

参数

table_name

您可以分析特定表的压缩，包括临时表。您可以通过其 schema 名称来限定表。您可以视需要指定 table_name 来分析单个表。如果您不指定 table_name，则将分析当前连接的数据库中的所有表。您不能在单个 ANALYZE COMPRESSION 语句中指定多个 table_name。

column_name

如果您指定 table_name，您还可以指定表中的一个或多个列（以两旁加括号的逗号分隔列表的形式）。

COMPROWS

要在压缩分析中用作采样大小的行数。将对来自每个数据切片的行运行分析。例如，如果您指定 COMPROWS 1000000 (1000000)，而系统共包含 4 个切片，则每个切片将读取和分析 250000 行。如果未指定 COMPROWS，则采样大小默认为每个切片 100000 行。如果 COMPROWS 值小于每个切片 100000 行的默认值，则会自动将该值升级到默认值。但是，如果表中的数据量不足，无法得到有意义的样本，则压缩分析将不会生成建议。如果 COMPROWS 数字大于表中的行数，则 ANALYZE COMPRESSION 命令仍会针对所有可用行继续运行压缩分析。

numrows

要在压缩分析中用作采样大小的行数。numrows 的可接受范围为 1000 到 1000000000 (1,000,000,000) 之间的数字。

使用说明

ANALYZE COMPRESSION 获取独占的表锁定，从而阻止对表的并发读写。只在表空闲时运行 ANALYZE COMPRESSION 命令。

运行 ANALYZE COMPRESSION 以根据表内容的采样来获取列编码方案的建议。ANALYZE COMPRESSION 是一个建议工具，不会修改表的列编码。可以通过重新创建表或使用相同 schema 创建新表来应用推荐的编码。使用适当的编码方案重新创建未压缩的表可以显著降低其磁盘上的大小。此方法可节省磁盘空间并改善 I/O 密集型工作负载的查询性能。

ANALYZE COMPRESSION 会跳过实际分析阶段，并直接返回指定为 SORTKEY 的任何列上的原始编码类型。之所以会执行此操作，是因为当 SORTKEY 列的压缩率远高于其他列时，范围受限扫描的执行效果可能会很差。

示例

以下示例说明了仅 LISTING 表中列的编码和预计减少的百分比：

```
analyze compression listing;

  Table  |     Column     | Encoding | Est_reduction_pct
```

```
---------+----------------+----------+-------------------
 listing | listid         | az64     | 40.96
 listing | sellerid       | az64     | 46.92
 listing | eventid        | az64     | 53.37
 listing | dateid         | raw      | 0.00
 listing | numtickets     | az64     | 65.66
 listing | priceperticket | az64     | 72.94
 listing | totalprice     | az64     | 68.05
 listing | listtime       | az64     | 49.74
```

以下示例分析了 SALES 表中的 QTYSOLD、COMMISSION 和 SALETIME 列。

```
analyze compression sales(qtysold, commission, saletime);

 Table |   Column   | Encoding | Est_reduction_pct
-------+------------+----------+-------------------
 sales | salesid    | N/A      | 0.00
 sales | listid     | N/A      | 0.00
 sales | sellerid   | N/A      | 0.00
 sales | buyerid    | N/A      | 0.00
 sales | eventid    | N/A      | 0.00
 sales | dateid     | N/A      | 0.00
 sales | qtysold    | az64     | 83.06
 sales | pricepaid  | N/A      | 0.00
 sales | commission | az64     | 71.85
 sales | saletime   | az64     | 49.63
```

ATTACH MASKING POLICY

将现有动态数据掩蔽策略附加到列。有关动态数据掩蔽的更多信息，请参阅 动态数据掩蔽。

超级用户和具有 sys:secadmin 角色的用户或角色可以附加屏蔽策略。

语法

```
ATTACH MASKING POLICY policy_name
   ON { relation_name }
   ( {output_columns_names | output_path} ) [ USING ( {input_column_names | input_path
 )} ]
   TO { user_name | ROLE role_name | PUBLIC }
   [ PRIORITY priority ];
```

参数

policy_name

要附加的屏蔽策略的名称。

relation_name

要将掩蔽策略附加到的关系的名称。

output_column_names

将要应用屏蔽策略的列的名称。

output_paths

掩蔽策略将应用于的 SUPER 对象的完整路径，包括列名。例如，对于具有名为 `person` 的 SUPER 类型列的关系，output_path 可能为 `person.name.first_name`。

input_column_names

将用作屏蔽策略输入的列的名称。此参数为可选的。如果未指定，则掩蔽策略使用 output_column_names 作为输入。

input_paths

将用作掩蔽策略输入的 SUPER 对象的完整路径。此参数为可选的。如果未指定，则掩蔽策略使用 output_path 作为输入。

user_name

要附加屏蔽策略的用户的名称。您不能将两个策略附加到用户和列或角色和列的相同组合。您可以将一个策略附加到用户，将另一个策略附加到该用户的角色。在这种情况下，优先级较高的策略适用。

在单个 ATTACH MASKING POLICY 命令中，您只能设置 user_name、role_name 和 PUBLIC 中的一项。

role_name

要附加屏蔽策略的角色的名称。您不能将两个策略附加到同一个列/角色对。您可以将一个策略附加到用户，将另一个策略附加到该用户的角色。在这种情况下，优先级较高的策略适用。

在单个 ATTACH MASKING POLICY 命令中，您只能设置 user_name、role_name 和 PUBLIC 中的一项。

PUBLIC

将屏蔽策略附加到访问表的所有用户。您必须为附加到特定列/用户或列/角色对的其他屏蔽策略分配比 PUBLIC 策略更高的优先级，这样才能让这些策略适用。

在单个 ATTACH MASKING POLICY 命令中，您只能设置 user_name、role_name 和 PUBLIC 中的一项。

priority

屏蔽策略的优先级。当多个屏蔽策略应用于给定用户的查询时，具有最高优先级的策略适用。

不能将两个不同的策略以相同的优先级附加到同一列上，即使这两个策略附加到了不同的用户或角色。您可以多次将相同的策略附加到同一组表、输出列、输入列和优先级参数，只要每次附加策略的用户或角色不同即可。

即使两个策略适用于不同的角色，您也无法将一个策略应用于与该列中附加的另一个策略具有相同优先级的列。该字段是可选的。如果未指定优先级，则屏蔽策略默认为优先级为 0。

ATTACH RLS POLICY

将表上的行级别安全性策略附加到一个或多个用户或角色。

超级用户和具有 `sys:secadmin` 角色的用户或角色可以附加策略。

语法

```
ATTACH RLS POLICY policy_name ON [TABLE] table_name [, ...]
TO { user_name | ROLE role_name | PUBLIC } [, ...]
```

参数

policy_name

策略的名称。

ON [TABLE]table_name [, ...]

行级安全策略所附加到的关系。

TO { user_name | ROLE role_name | PUBLIC} [, ...]

指定是否将策略附加到一个或多个指定用户或角色。

使用说明

在使用 ATTACH RLS POLICY 语句时，请遵守以下规则：

- 附加的表应该包含策略创建语句的 WITH 子句中列出的所有列。
- Amazon Redshift RLS 不支持将 RLS 策略附加到以下对象：
  - 目录表
  - 跨数据库关系
  - 外部表
  - 实体化视图
  - 临时表
  - 查找表
- 不能将 RLS 策略附加到超级用户或具有 `sys:secadmin` 权限的用户。

示例

以下示例将表上的策略附加到了角色。

```
ATTACH RLS POLICY policy_concerts ON tickit_category_redshift TO ROLE analyst, ROLE
 dbadmin;
```

BEGIN

开始事务。与 START TRANSACTION 同义。

事务是单一的逻辑工作单元，而无论是包含一个命令还是多个命令。一般来说，事务中的所有命令都是在数据库的快照上执行，其开始时间由为 `transaction_snapshot_begin` 系统配置参数设置的值决定。

预设情况下，单独的 Amazon Redshift 操作（查询、DDL 语句、加载）自动提交到数据库。如果要暂停操作提交直到后续工作完成，您需要使用 BEGIN 语句打开一个事务，然后运行所需的命令，最后使用 COMMIT 或 END 语句关闭事务。如果需要，您可以使用 ROLLBACK 语句停止正在进行的事务。此行为的一个例外是 TRUNCATE 命令，它在所运行的事务中提交事务，并且无法回滚。

语法

```
BEGIN [ WORK | TRANSACTION ] [ ISOLATION LEVEL option ] [ READ WRITE | READ ONLY ]
```

```
START TRANSACTION [ ISOLATION LEVEL option ] [ READ WRITE | READ ONLY ]

Where option is

SERIALIZABLE
| READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ

Note: READ UNCOMMITTED, READ COMMITTED, and REPEATABLE READ have no
operational impact and map to SERIALIZABLE in Amazon Redshift. You can see database
 isolation levels on your cluster
by querying the stv_db_isolation_level table.
```

参数

WORK

可选关键字。

TRANSACTION

可选关键字；WORK 和 TRANSACTION 同义。

ISOLATION LEVEL SERIALIZABLE

默认情况下支持可序列化隔离，因此无论此语法是否包含在语句中，事务的行为都是相同的。有关更多信息，请参阅 管理并发写入操作。不支持任何其他隔离级别。

> **Note**
>
> SQL 标准定义了四个事务隔离级别，可防止脏读（事务读取并发未提交事务写入的数据）、不可重复读（事务重新读取之前已读取的数据，并发现自初始读取以后，另一个已提交的事务已更改了该数据）以及幻读（事务重新运行查询，返回满足一组搜索条件的行，然后发现这组行由于另一个最近提交的事务而发生了更改）：
>
> - 读取未提交：脏读、不可重复读以及幻读是可能的。
> - 读取已提交：不可重复读和幻读是可能的。
> - 可重复读：幻读是可能的。
> - 可序列化：阻止脏读、不可重复读以及幻读。
>
> 虽然您可以使用这四个事务隔离级别中的任意一个，不过 Amazon Redshift 会将所有隔离级别作为可序列化级别处理。

READ WRITE

向事务提供读取和写入权限。

READ ONLY

向事务提供只读权限。

示例

以下示例开始一个可序列化事务块：

```
begin;
```

以下示例开始一个具有可序列化隔离级别和读写权限的事务块：

```
begin read write;
```

CALL

运行存储过程。CALL 命令必须包括过程名称和输入参数值。您必须使用 CALL 语句调用存储过程。

> **Note**
>
> CALL 不能成为任何常规查询的一部分。

语法

```
CALL sp_name ( [ argument ] [, ...] )
```

参数

sp_name

要运行的过程的名称。

argument

输入参数的值。此参数也可以是函数名称，例如 `pg_last_query_id()`。您不能将查询用作 CALL 参数。

使用说明

Amazon Redshift 存储过程支持嵌套和递归调用，如下所述。此外，请确保您的驱动程序支持是最新的，如下所述

主题

嵌套调用

Amazon Redshift 存储过程支持嵌套和递归调用。允许的最大嵌套级别数为 16。嵌套调用可以将业务逻辑封装到较小的过程中，这些过程可以由多个调用者共享。

如果调用了具有输出参数的嵌套过程，则该内部过程必须定义 INOUT 参数。在这种情况下，在非常量变量中传递该内部过程。不允许使用 OUT 参数。出现此问题的原因是需要一个变量来保存内部调用的输出。

内部过程和外部过程之间的关系记录在 `from_sp_call` 的 SVL_STORED_PROC_CALL 列中。

以下示例演示通过 INOUT 参数将变量传递给嵌套过程调用。

```
CREATE OR REPLACE PROCEDURE inner_proc(INOUT a int, b int, INOUT c int) LANGUAGE
 plpgsql
AS $$
BEGIN
  a := b * a;
  c := b * c;
END;
$$;

CREATE OR REPLACE PROCEDURE outer_proc(multiplier int) LANGUAGE plpgsql
AS $$
DECLARE
  x int := 3;
  y int := 4;
BEGIN
  DROP TABLE IF EXISTS test_tbl;
  CREATE TEMP TABLE test_tbl(a int, b varchar(256));
  CALL inner_proc(x, multiplier, y);
  insert into test_tbl values (x, y::varchar);
```

```
END;
$$;

CALL outer_proc(5);

SELECT * from test_tbl;
 a  | b
----+----
 15 | 20
(1 row)
```

驱动程序支持

我们建议您将 Java 数据库连接 (JDBC) 和开放式数据库连接 (ODBC) 驱动程序升级到支持 Amazon Redshift 存储过程的最新版本。

如果客户端工具使用通过 CALL 语句传递给服务器的驱动程序 API 操作，则您可以使用现有驱动程序。输出参数（如果有）将作为包含一行的结果集返回。

最新版本的 Amazon Redshift JDBC 和 ODBC 驱动程序对存储过程发现提供元数据支持。这些最新版本对于自定义 Java 应用程序还支持 `CallableStatement`。有关驱动程序的更多信息，请参阅《Amazon Redshift 管理指南》中的使用 SQL 客户端工具连接到 Amazon Redshift 集群。

以下示例说明如何对存储过程调用使用 JDBC 驱动程序的不同 API 操作。

```
void statement_example(Connection conn) throws SQLException {
  statement.execute("CALL sp_statement_example(1)");
}

void prepared_statement_example(Connection conn) throws SQLException {
  String sql = "CALL sp_prepared_statement_example(42, 84)";
  PreparedStatement pstmt = conn.prepareStatement(sql);
  pstmt.execute();
}

void callable_statement_example(Connection conn) throws SQLException {
  CallableStatement cstmt = conn.prepareCall("CALL sp_create_out_in(?,?)");
  cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
  cstmt.setInt(2, 42);
  cstmt.executeQuery();
  Integer out_value = cstmt.getInt(1);
}
```

示例

以下示例调用过程名称 test_spl。

```
call test_sp1(3,'book');
INFO:  Table "tmp_tbl" does not exist and will be skipped
INFO:  min_val = 3, f2 = book
```

以下示例调用过程名称 test_spl2。

```
call test_sp2(2,'2019');

         f2          | column2
---------------------+---------
 2019+2019+2019+2019 | 2
(1 row)
```

CANCEL

取消当前正在运行的数据库查询。

CANCEL 命令需要正在运行的查询的进程 ID 或会话 ID，并显示一条确认消息来验证查询已取消。

所需的权限

以下是 CANCEL 所需的权限：

- 取消自己的查询的超级用户
- 取消用户的查询的超级用户
- 取消用户的查询并且具有 CANCEL 权限的用户
- 取消自己的查询的用户

语法

```
CANCEL process_id [ 'message' ]
```

参数

process_id

要取消在 Amazon Redshift 集群中运行的查询，请使用 STV_RECENTS 中与要取消的查询对应的 pid（进程 ID）。

要取消在 Amazon Redshift Serverless 工作组中运行的查询，请使用 SYS_QUERY_HISTORY 中与要取消的查询对应的 session_id。

'消息'

一条可选的确认消息，该消息在查询取消操作完成时显示。如果您不指定消息，Amazon Redshift 将显示默认的确认消息。您必须将消息放在单引号内。

使用说明

您不能通过指定查询 ID 来取消查询；您必须指定查询的进程 ID（PID）或会话 ID。您只能取消当前由您的用户运行的查询。超级用户可以取消所有查询。

如果多个会话中的查询在同一个表上保留锁定，则可以使用 PG_TERMINATE_BACKEND 函数终止其中一个会话。进行此操作会强制使已终止会话中的任意当前正在运行的查询释放所有死锁并回滚事务。查询 STV_LOCKS 系统表可查看当前保留的锁定。

在特定的内部事件之后，Amazon Redshift 可能会重新启动一个活动会话并分配新的 PID。如果 PID 已发生更改，您可能会收到以下错误消息。

```
Session <PID> does not exist. The session PID might have changed. Check the
 stl_restarted_sessions system table for details.
```

要查找新的 PID，请查询 STL_RESTARTED_SESSIONS 系统表并针对 oldpid 列进行筛选。

```
select oldpid, newpid from stl_restarted_sessions where oldpid = 1234;
```

示例

要取消 Amazon Redshift 集群中当前正在运行的查询，请先检索要取消的查询的进程 ID。要确定所有当前正在运行的查询的处理 ID，请键入以下命令：

```
select pid, starttime, duration,
trim(user_name) as user,
```

```
trim (query) as querytxt
from stv_recents
where status = 'Running';

pid |         starttime          | duration |   user   |    querytxt
-----+----------------------------+----------+----------+-----------------
802 | 2008-10-14 09:19:03.550885 |      132 | dwuser | select
venuename from venue where venuestate='FL', where venuecity not in
('Miami' , 'Orlando');
834 | 2008-10-14 08:33:49.473585 |  1250414 | dwuser | select *
from listing;
964 | 2008-10-14 08:30:43.290527 |   326179 | dwuser | select
sellerid from sales where qtysold in (8, 10);
```

检查查询文本，确定哪个进程 ID (PID) 对应于您要取消的查询。

键入以下命令以使用 PID 802 来取消该查询：

```
cancel 802;
```

运行查询的会话会显示如下消息：

```
ERROR:  Query (168) cancelled on user's request
```

其中 168 是查询 ID (而不是用于取消查询的进程 ID)。

或者，您可以指定显示一条自定义消息，而不是默认消息。要指定自定义消息，请使用单引号将消息包含在 CANCEL 命令的结尾：

```
cancel 802 'Long-running query';
```

运行查询的会话会显示如下消息：

```
ERROR:  Long-running query
```

CLOSE

（可选）关闭与已打开游标关联的所有空闲资源。COMMIT、END 和 ROLLBACK 会自动关闭游标，因此不必使用 CLOSE 命令显式关闭游标。

有关更多信息，请参阅DECLARE、FETCH。

语法

```
CLOSE cursor
```

参数

cursor

要关闭的游标的名称。

CLOSE 示例

以下命令关闭游标并执行提交，这将结束事务：

```
close movie_cursor;
commit;
```

COMMENT

创建或更改有关数据库对象的注释。

语法

```
COMMENT ON
{
TABLE object_name |
COLUMN object_name.column_name |
CONSTRAINT constraint_name ON table_name |
DATABASE object_name |
VIEW object_name
}
IS 'text' | NULL
```

参数

object_name

要添加注释的数据库对象的名称。您可以将注释添加到以下对象：

- TABLE

- COLUMN（还接受 column_name）。
- CONSTRAINT（还接受 constraint_name 和 table_name）。
- DATABASE
- VIEW
- SCHEMA

IS 'text' | NULL

要为指定对象添加或替换的注释文本。文本字符串的数据类型是 TEXT。将注释放在单引号内。将值设置为 NULL 可删除注释文本。

column_name

要添加注释的列的名称。COLUMN 的参数。跟随在 object_name 中指定的表后面。

constraint_name

要添加注释的约束的名称。CONSTRAINT 的参数。

table_name

包含约束的表的名称。CONSTRAINT 的参数。

使用说明

要添加或更新注释，您必须是超级用户或数据库对象的所有者。

数据库的注释只能应用于当前数据库。如果您尝试对不同的数据库添加注释，则会显示警告消息。对不存在的数据库添加注释时，会显示同一警告。

不支持对外部表、外部列和后期绑定视图的列进行评论。

示例

以下示例将注释添加到 SALES 表中。

```
COMMENT ON TABLE sales IS 'This table stores tickets sales data';
```

以下示例在 SALES 表中显示该注释。

```
select obj_description('public.sales'::regclass);

obj_description
```

```
-------------------------------------
This table stores tickets sales data
```

以下示例从 SALES 表中删除注释。

```
COMMENT ON TABLE sales IS NULL;
```

以下示例将注释添加到 SALES 表的 EVENTID 列中。

```
COMMENT ON COLUMN sales.eventid IS 'Foreign-key reference to the EVENT table.';
```

以下示例在 SALES 表的 EVENTID 列（列号 5）中显示注释。

```
select col_description( 'public.sales'::regclass, 5::integer );

col_description
----------------------------------------
Foreign-key reference to the EVENT table.
```

以下示例将描述性注释添加到 EVENT 表。

```
comment on table event is 'Contains listings of individual events.';
```

要查看注释，请查询 PG_DESCRIPTION 系统目录。以下示例返回 EVENT 表的描述。

```
select * from pg_catalog.pg_description
where objoid =
(select oid from pg_class where relname = 'event'
and relnamespace =
(select oid from pg_catalog.pg_namespace where nspname = 'public') );

objoid | classoid | objsubid | description
-------+----------+----------+----------------------------------------
116658 |     1259 |        0 | Contains listings of individual events.
```

COMMIT

将当前事务提交到数据库。此命令使事务中的数据库更新永久生效。

语法

```
COMMIT [ WORK | TRANSACTION ]
```

参数

WORK

可选关键字。存储过程中不支持此关键字。

TRANSACTION

可选关键字。WORK 和 TRANSACTION 是同义词。两者在存储过程中均不受支持。

有关在存储过程中使用 COMMIT 的信息，请参阅管理事务。

示例

下面的每个示例都将当前事务提交到数据库：

```
commit;
```

```
commit work;
```

```
commit transaction;
```

COPY

将数据从数据文件或 Amazon DynamoDB 表加载到表中。这些文件可以位于 Amazon Simple Storage Service (Amazon S3) 桶、Amazon EMR 集群或可使用 Secure Shell (SSH) 连接访问的远程主机中。

> **Note**
>
> Amazon Redshift Spectrum 外部表为只读。您无法对外部表进行 COPY。

COPY 命令会将输入数据作为额外的行附加到表中。

来自任何源的单个输入行的最大大小为 4 MB。

主题

所需的权限

要使用 COPY 命令，您必须对 Amazon Redshift 表拥有 INSERT 权限。

COPY 语法

```
COPY table-name
[ column-list ]
FROM data_source
authorization
[ [ FORMAT ] [ AS ] data_format ]
[ parameter [ argument ] [, ... ] ]
```

只需 3 个参数即可执行 COPY 操作：表名称、数据来源和对数据的访问的授权。

Amazon Redshift 扩展了 COPY 命令的功能，使您可以从多个数据来源加载多种数据格式的数据、控制对加载数据的访问权限、管理数据转换和管理加载操作。

以下各节介绍所需的 COPY 命令参数，并按功能对可选参数进行分组。其中还会介绍每个参数并说明各个选项如何配合使用。您可以通过使用按字母顺序排列的参数列表直接转到相应的参数说明。

必需参数

COPY 命令需要三个元素：

- Table Name
- Data Source
- Authorization

最简单的 COPY 命令使用以下格式。

```
COPY table-name
FROM data-source
authorization;
```

以下示例创建一个名为 CATDEMO 的表，然后从 Amazon S3 中名为 `category_pipe.txt` 的数据文件加载包含样本数据的表。

```
create table catdemo(catid smallint, catgroup varchar(10), catname varchar(10), catdesc
 varchar(50));
```

在以下示例中，COPY 命令的数据来源是一个数据文件，名为 `category_pipe.txt`，位于名为 `redshift-downloads` 的 Amazon S3 桶的 `tickit` 文件夹中。COPY 命令有权通过 AWS Identity and Access Management (IAM) 角色访问 Amazon S3 桶。如果您的集群具有有权访问附加的 Amazon S3 的现有 IAM 角色，您可以在以下 COPY 命令中替换您的角色的 Amazon 资源名称 (ARN) 并执行该角色。

```
copy catdemo
from 's3://redshift-downloads/tickit/category_pipe.txt'
iam_role 'arn:aws:iam::<aws-account-id>:role/<role-name>'
region 'us-east-1';
```

有关如何使用 COPY 命令加载示例数据的完整说明，包括从其他 AWS 区域加载数据的说明，请参阅《Amazon Redshift 入门指南》中的从 Amazon S3 中加载示例数据。

table-name

COPY 命令的目标表的名称。该表必须已存在于数据库中。该表可以是临时的或永久的。COPY 命令会将新输入数据追加到该表中的任何现有行。

FROM data-source

要加载到目标表中的源数据的位置。可使用某些数据来源指定清单文件。

最常用的数据存储库是 Amazon S3 桶。您还可以从位于 Amazon EMR 集群中、位于 Amazon EC2 实例中或位于您的集群可使用 SSH 连接访问的远程主机中的数据文件加载，或者也可以直接从 DynamoDB 表加载。

- 从 Amazon S3 执行 COPY 操作
- 从 Amazon EMR 执行 COPY 操作
- 从远程主机中执行 COPY 操作 (SSH)
- 从 Amazon DynamoDB 执行 COPY 操作

授权

一个子句，指示您的集群用于访问其他 AWS 资源的身份验证和授权的方法。COPY 命令需要授权才能访问其他 AWS 资源（包括 Amazon S3、Amazon EMR、Amazon DynamoDB 和 Amazon EC2）中的数据。您可通过引用附加到您的集群的 IAM 角色或通过为 IAM 用户提供访问密钥 ID 和秘密访问密钥来提供该授权。

- 授权参数
- 基于角色的访问控制
- 基于密钥的访问控制

可选参数

您可以选择性地指定 COPY 命令如何将字段数据映射到目标表中的列，定义源数据属性以便让 COPY 命令正确读取和分析源数据，以及管理 COPY 命令在加载过程中执行的操作。

- 列映射选项
- 数据格式参数
- 数据转换参数
- 数据加载操作

列映射

默认情况下，COPY 会按字段在数据文件中出现的相同顺序将字段值插入到目标表的列中。如果默认列顺序不起作用，则可以指定一个列列表或使用 JSONPath 表达式将源数据字段映射到目标列。

- Column List
- JSONPaths File

数据格式参数

您可以从固定宽度、字符分隔、逗号分隔值 (CSV)、JSON 格式的文本文件加载数据，也可从 Avro 文件加载数据。

默认情况下，COPY 命令要求源数据位于字符分隔的 UTF-8 文本文件中。默认分隔符是竖线字符 (|)。如果源数据采用的是其他格式，请使用以下参数指定数据格式。

- FORMAT
- CSV
- DELIMITER
- FIXEDWIDTH
- SHAPEFILE
- AVRO
- JSON
- ENCRYPTED
- BZIP2
- GZIP
- LZOP
- PARQUET
- ORC
- ZSTD

数据转换参数

在加载表时，COPY 会尝试将源数据中的字符串隐式转换为目标列的数据类型。如果您需要指定不同于默认行为的转换，或者默认转换会产生错误，则可以通过指定以下参数来管理数据转换。

- ACCEPTANYDATE
- ACCEPTINVCHARS
- BLANKSASNULL
- DATEFORMAT
- EMPTYASNULL
- ENCODING

- ESCAPE
- EXPLICIT_IDS
- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL AS
- REMOVEQUOTES
- ROUNDEC
- TIMEFORMAT
- TRIMBLANKS
- TRUNCATECOLUMNS

数据加载操作

通过指定以下参数来管理加载操作的默认行为，以进行故障排除或缩短加载时间。

- COMPROWS
- COMPUPDATE
- IGNOREALLERRORS
- MAXERROR
- NOLOAD
- STATUPDATE

COPY 命令的使用说明和其它资源

有关如何使用 COPY 命令的更多信息，请参阅以下主题：

- 使用说明
- 教程：从 Amazon S3 加载数据
- Amazon Redshift 加载数据的最佳实践
- 使用 COPY 命令加载数据
  - 从 Amazon S3 加载数据
  - 从 Amazon EMR 中加载数据

- 从远程主机中加载数据
- 从 Amazon DynamoDB 表中加载数据
- 解决数据加载问题

COPY 命令示例

有关展示如何使用不同来源、不同格式和不同的 COPY 选项执行 COPY 操作的更多示例，请参阅COPY 示例。

COPY JOB（预览版）

这是面向预览版中的自动复制（SQL COPY JOB）的预发行文档。文档和特征都可能会更改。我们建议您仅在测试环境中使用此特征，不要在生产环境中使用。公开预览将于 2024 年 7 月 31 日结束。在预览结束两周后，将自动删除预览集群。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

有关在预览版中使用此命令的信息，请参阅 从 Amazon S3 持续摄取文件（预览版）。

管理将数据加载到表中的 COPY 命令。COPY JOB 命令是 COPY 命令的扩展，可自动从 Amazon S3 桶加载数据。当您创建 COPY 作业时，Amazon Redshift 会检测何时在指定路径中创建新的 Amazon S3 文件，然后自动加载这些文件，无需您的干预。加载数据时使用的参数与原始 COPY 命令中使用的参数相同。Amazon Redshift 会跟踪加载的文件，以确认它们只加载一次。

> **Note**
>
> 有关 COPY 命令的信息，包括用法、参数和权限，请参阅 COPY。

所需的权限

要运行 COPY JOB 的 COPY 命令，必须对要加载的表具有 INSERT 权限。

使用 COPY 命令指定的 IAM 角色必须具有访问待加载数据的权限。有关更多信息，请参阅 COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限。

语法

创建复制作业。COPY 命令的参数与复制作业一起保存。

```
COPY copy-command JOB CREATE job-name
[AUTO ON | OFF]
```

更改复制作业的配置。

```
COPY JOB ALTER job-name
[AUTO ON | OFF]
```

运行复制作业。使用存储的 COPY 命令参数。

```
COPY JOB RUN job-name
```

列出所有复制作业。

```
COPY JOB LIST
```

显示复制作业的详细信息。

```
COPY JOB SHOW job-name
```

删除复制作业。

```
COPY JOB DROP job-name
```

参数

copy-command

COPY 命令将数据从 Amazon S3 加载到 Amazon Redshift。该子句包含用于定义 Amazon S3 桶、目标表、IAM 角色的 COPY 参数，以及加载数据时使用的其他参数。支持用于 Amazon S3 数据加载的所有 COPY 命令参数，但以下参数除外：

- COPY JOB 不会摄取 COPY 命令指向的文件夹中已有的文件。只有在 COPY JOB 创建时间戳之后创建的文件才会被摄取。
- 不能使用 MAXERROR 或 IGNOREALLERRORS 选项指定 COPY 命令。
- 不能指定清单文件。COPY JOB 需要指定的 Amazon S3 位置来监控新创建的文件。
- 不能使用访问密钥和私有密钥等授权类型指定 COPY 命令。仅支持使用 IAM_ROLE 参数进行授权的 COPY 命令。有关更多信息，请参阅 授权参数。
- COPY JOB 不支持与集群关联的默认 IAM 角色。必须在 COPY 命令中指定 IAM_ROLE。

有关更多信息，请参阅 从 Amazon S3 执行 COPY 操作。

job-name

用于引用 COPY JOB 的作业的名称。

[AUTO ON | OFF]

该子句指示 Amazon S3 数据是否自动加载到 Amazon Redshift 表中。

- 选项为 ON 时，Amazon Redshift 会监控源 Amazon S3 路径中新创建的文件，如果找到新创建的文件，则使用作业定义中的 COPY 参数运行 COPY 命令。这是默认模式。
- 选项为 OFF 时，Amazon Redshift 不会自动运行 COPY JOB。

使用说明

COPY 命令的选项要等到运行时才会验证。例如，在 COPY JOB 启动时，无效的 IAM_ROLE 或 Amazon S3 数据来源会导致出现运行时错误。

如果暂停集群，则不运行 COPY JOB。

要查询已加载的 COPY 命令文件和加载错误，请参见 STL_LOAD_COMMITS、STL_LOAD_ERRORS 和 STL_LOADERROR_DETAIL。有关更多信息，请参阅 验证是否正确加载了数据。

示例

以下示例显示创建 COPY JOB 以从 Amazon S3 桶加载数据。

```
COPY public.target_table
FROM 's3://mybucket-bucket/staging-folder'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyLoadRoleName'
JOB CREATE my_copy_job_name
AUTO ON;
```

COPY 参数参考

COPY 有许多可以在多种情况下使用的参数。但是，并不是所有参数在每种情况下都受支持。例如，要从 ORC 或 PARQUET 文件加载，支持的参数数量有限。有关更多信息，请参阅 从列式数据格式中执行 COPY 操作。

主题

- 数据来源
- 授权参数

- 列映射选项
- 数据格式参数
- 文件压缩参数
- 数据转换参数
- 数据加载操作
- 按字母顺序排列的参数列表

数据来源

您可以从位于 Amazon S3 桶中、位于 Amazon EMR 集群中或位于您的集群可使用 SSH 连接访问的远程主机上的文本文件加载数据。您也可以直接从 DynamoDB 表加载数据。

来自任何源的单个输入行的最大大小为 4 MB。

要将表中的数据导出到 Amazon S3 中的一组文件，请使用 UNLOAD 命令。

主题

- 从 Amazon S3 执行 COPY 操作
- 从 Amazon EMR 执行 COPY 操作
- 从远程主机中执行 COPY 操作 (SSH)
- 从 Amazon DynamoDB 执行 COPY 操作

从 Amazon S3 执行 COPY 操作

要从位于一个或多个 S3 桶中的文件加载数据，请使用 FROM 子句指示 COPY 在 Amazon S3 中查找文件的方式。您可以提供数据文件的对象路径作为 FROM 子句的一部分，也可以提供包含了 Amazon S3 对象路径列表的清单文件的位置。从 Amazon S3 执行 COPY 操作将使用 HTTPS 连接。确保将 S3 IP 范围添加到您的允许列表中。要了解有关所需 S3 IP 范围的更多信息，请参阅网络隔离。

> **Important**
>
> 如果包含数据文件的 Amazon S3 桶未驻留在您的集群所在的 AWS 区域内，则必须使用 REGION 参数指定数据所在的区域。

主题

- 语法

- 示例
- 可选参数
- 不支持的参数

语法

```
FROM { 's3://objectpath' | 's3://manifest_file' }
authorization
| MANIFEST
| ENCRYPTED
| REGION [AS] 'aws-region'
| optional-parameters
```

示例

以下示例使用对象路径从 Amazon S3 加载数据。

```
copy customer
from 's3://mybucket/customer'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

以下示例使用清单文件从 Amazon S3 加载数据。

```
copy customer
from 's3://mybucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

参数

FROM

要加载的数据的源。有关 Amazon S3 文件编码的更多信息，请参阅数据转换参数。

's3://copy_from_s3_objectpath'

指定包含数据的 Amazon S3 对象的路径，例如 `'s3://mybucket/custdata.txt'`。s3://copy_from_s3_objectpath 参数可引用单个文件或者具有相同键前缀的一组对象或文件夹。例如，名称 `custdata.txt` 是引用很多物理文件（`custdata.txt`、`custdata.txt.1`，等等）的键前缀。`custdata.txt.2custdata.txt.bak`键前缀还可以引用很多文件夹。

例如，'s3://mybucket/custfolder' 引用文件夹 custfolder、custfolder_1，等等。custfolder_2如果键前缀引用多个文件夹，则加载这些文件夹中的所有文件。如果键前缀与一个文件以及一个文件夹匹配，如 custfolder.log，COPY 还将尝试加载该文件。如果键前缀可能导致 COPY 尝试加载不需要的文件，请使用清单文件。有关更多信息，请参阅以下内容：copy_from_s3_manifest_file。

> **Important**
>
> 如果包含数据文件的 S3 桶未驻留在您的集群所在的 AWS 区域内，则必须使用 REGION 参数指定数据所在的区域。

有关更多信息，请参阅 从 Amazon S3 加载数据。

's3://copy_from_s3_manifest_file'

为列出了要加载的数据文件的清单文件指定 Amazon S3 对象键。's3://copy_from_s3_manifest_file' 参数必须显式引用单个文件，例如's3://mybucket/manifest.txt'。它不能引用键前缀。

清单是 JSON 格式的文本文件，其中列出了要从 Amazon S3 加载的每个文件的 URL。URL 包含文件的桶名称和完整对象路径。在清单中指定的文件可以位于不同的桶中，但所有桶都必须位于 Amazon Redshift 集群所在的同一 AWS 区域。如果某个文件被列出两次，那么该文件也会被加载两次。以下示例显示了加载三个文件的清单的 JSON。

```
{
  "entries": [
    {"url":"s3://mybucket-alpha/custdata.1","mandatory":true},
    {"url":"s3://mybucket-alpha/custdata.2","mandatory":true},
    {"url":"s3://mybucket-beta/custdata.1","mandatory":false}
  ]
}
```

需要双引号字符，并且必须是简单引号 (0x22)，而不能是斜引号或“智能”引号。清单中的每个条目都可以选择性地包含 mandatory 标记。如果 mandatory 设置为 true，则当 COPY 未找到该条目对应的文件时，该命令将会终止；否则，该命令将继续。mandatory 的默认值为 false。

在从采用 ORC 或 Parquet 格式的数据文件中加载时，需要 meta 字段，如以下示例所示。

```
{
   "entries":[
      {
```

```
         "url":"s3://mybucket-alpha/orc/2013-10-04-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      },
      {
         "url":"s3://mybucket-beta/orc/2013-10-05-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      }
   ]
}
```

不能对清单文件进行加密或压缩，即使指定了 ENCRYPTED、GZIP、LZOP、BZIP2 或 ZSTD 选项。如果未找到指定的清单文件或清单文件的格式不正确，COPY 命令将返回错误。

如果使用了清单文件，则必须使用 COPY 命令指定 MANIFEST 参数。如果未指定 MANIFEST 参数，COPY 命令将假定使用 FROM 指定的文件是数据文件。

有关更多信息，请参阅 从 Amazon S3 加载数据。

授权

COPY 命令需要授权才能访问其他 AWS 资源（包括 Amazon S3 、Amazon EMR、Amazon DynamoDB 和 Amazon EC2）中的数据。您可通过引用附加到您的集群的 AWS Identity and Access Management (IAM) 角色（基于角色的访问控制）或者通过为用户提供访问凭证（基于密钥的访问控制）来提供授权。为了提高安全性和灵活性，我们建议使用基于 IAM 角色的访问控制。有关更多信息，请参阅 授权参数。

MANIFEST

指定使用一个清单来标识要从 Amazon S3 加载的数据文件。如果使用了 MANIFEST 参数，则 COPY 将从 's3://copy_from_s3_manifest_file' 引用的清单中列出的文件加载数据。如果未找到清单文件或清单文件的格式不正确，COPY 将失败。有关更多信息，请参阅 使用清单指定数据文件。

ENCRYPTED

一个子句，指定 Amazon S3 上输入文件的加密方法为：利用客户管理的密钥进行客户端加密。有关更多信息，请参阅 从 Amazon S3 中加载加密的数据文件。如果输入文件的加密方法为 Amazon S3 服务器端加密（SSE-KMS 或 SSE-S3），请不要指定 ENCRYPTED。COPY 会自动读取服务器端加密的文件。

如果您要指定 ENCRYPTED 参数，还必须指定 MASTER_SYMMETRIC_KEY 参数，或在 **master_symmetric_key** 字符串中包括 CREDENTIALS 值。

如果加密文件采用了压缩格式，请添加 GZIP、LZOP、BZIP2 或 ZSTD 参数。

即使指定了 ENCRYPTED 选项，也不得加密清单文件和 JSONPaths 文件。

MASTER_SYMMETRIC_KEY 'root_key'

用于在 Amazon S3 上加密数据文件的根对称密钥。如果指定了 MASTER_SYMMETRIC_KEY，还须指定 ENCRYPTED 参数。MASTER_SYMMETRIC_KEY 不能与 CREDENTIALS 参数配合使用。有关更多信息，请参阅 从 Amazon S3 中加载加密的数据文件。

如果加密文件采用了压缩格式，请添加 GZIP、LZOP、BZIP2 或 ZSTD 参数。

REGION [AS] 'aws-region'

指定源数据所在的 AWS 区域。当包含该数据的 AWS 资源与 Amazon Redshift 集群不在同一区域时，从 Amazon S3 桶或 DynamoDB 表执行 COPY 的操作需要 REGION。

aws_region 的值必须与 Amazon Redshift 区域和端点表中所列的区域匹配。

如果指定了 REGION 参数，则所有资源（包括清单文件或多个 Amazon S3 桶）都必须位于指定区域内。

> **Note**
>
> 对于包含数据的 Amazon S3 桶或 DynamoDB 表，跨区域传输数据将会产生额外费用。有关定价的更多信息，请参阅 Amazon S3 定价页面上的将数据从 Amazon S3 移出到另一个 AWS 区域和 Amazon DynamoDB 定价页面上的移出数据。

预设情况下，COPY 假定数据位于 Amazon Redshift 集群所在的相同区域。

可选参数

对于从 Amazon S3 执行 COPY 的操作，还可以指定以下参数：

- 列映射选项
- 数据格式参数
- 数据转换参数
- 数据加载操作

不支持的参数

对于从 Amazon S3 执行 COPY 的操作，不能使用以下参数：

- SSH
- READRATIO

从 Amazon EMR 执行 COPY 操作

您可以使用 COPY 命令从一个具有如下配置的 Amazon EMR 集群并行加载数据：将文本文件以固定宽度文件、字符分隔文件、CSV 文件、JSON 格式文件或 Avro 文件的形式写入到集群的 Hadoop Distributed File System (HDFS)。

主题

语法

```
FROM 'emr://emr_cluster_id/hdfs_filepath'
authorization
[ optional_parameters ]
```

示例

以下示例从一个 Amazon EMR 集群加载数据。

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/part-*'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

参数

FROM

要加载的数据的源。

'emr://emr_cluster_id/hdfs_file_path'

Amazon EMR 集群的唯一标识符和引用 COPY 命令的数据文件的 HDFS 文件路径。HDFS 数据文件名不能包含通配符星号 (*) 和问号 (?)。

> **Note**
>
> 在 COPY 操作完成前，Amazon EMR 集群必须持续运行。如果在 COPY 操作完成前更改或删除了任何 HDFS 数据文件，则您可能得到意外的结果，或者 COPY 操作可能会失败。

您可以使用通配符星号 (*) 和问号 (?) 作为 hdfs_file_path 参数的一部分来指定要加载的多个文件。例如，`'emr://j-SAMPLE2B500FC/myoutput/part*'` 标识文件 `part-0000`、`part-0001`，等等。如果文件路径不包含通配符，则将其视为文字字符串。如果您仅指定一个文件夹名称，则 COPY 将尝试加载该文件夹中的所有文件。

> **Important**
>
> 如果您使用通配符或仅使用文件夹名称，请确认将不会加载不需要的文件。例如，某些流程可能会将日志文件写入到输出文件夹。

有关更多信息，请参阅 从 Amazon EMR 中加载数据。

授权

COPY 命令需要授权才能访问其他 AWS 资源（包括 Amazon S3、Amazon EMR、Amazon DynamoDB 和 Amazon EC2）中的数据。您可通过引用附加到您的集群的 AWS Identity and Access Management (IAM) 角色（基于角色的访问控制）或者通过为用户提供访问凭证（基于密钥的访问控制）来提供授权。为了提高安全性和灵活性，我们建议使用基于 IAM 角色的访问控制。有关更多信息，请参阅 授权参数。

支持的参数

对于从 Amazon EMR 执行 COPY 的操作，还可以指定以下参数：

- 列映射选项
- 数据格式参数
- 数据转换参数

- 数据加载操作

不支持的参数

对于从 Amazon EMR 执行 COPY 的操作，不能使用以下参数：

- ENCRYPTED
- MANIFEST
- REGION
- READRATIO
- SSH

从远程主机中执行 COPY 操作 (SSH)

您可使用 COPY 命令从一台或多台远程主机并行加载数据，例如 Amazon Elastic Compute Cloud (Amazon EC2) 实例或其他计算机。COPY 使用 Secure Shell (SSH) 连接到远程主机并在远程主机上运行命令以生成文本输出。远程主机可以是 EC2 Linux 实例或配置为接受 SSH 连接的另一台 Unix 或 Linux 计算机。Amazon Redshift 可连接到多台主机，并可以打开到每台主机的多个 SSH 连接。Amazon Redshift 会通过每个连接发送一个唯一命令来生成到主机标准输出的文本输出，然后 Amazon Redshift 会像读取文本文件一样读取它。

使用 FROM 子句指定一个清单文件的 Amazon S3 对象键，该清单文件提供 COPY 用于建立 SSH 连接并执行远程命令的信息。

主题

- 语法
- 示例
- 参数
- 可选参数
- 不支持的参数

> **Important**
>
> 如果包含清单文件的 S3 桶未驻留在您的集群所在的 AWS 区域内，则必须使用 REGION 参数指定该桶所在的区域。

语法

```
FROM 's3://'ssh_manifest_file' }
authorization
SSH
| optional-parameters
```

示例

以下示例使用清单文件从使用 SSH 的远程主机加载数据。

```
copy sales
from 's3://mybucket/ssh_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
ssh;
```

参数

FROM

要加载的数据的源。

's3://copy_from_ssh_manifest_file'

COPY 命令可连接到使用 SSH 的多台主机，并可以与每台主机建立多个 SSH 连接。COPY 通过每个主机连接运行一个命令，然后将来自这些命令的输出并行加载到表中。s3://copy_from_ssh_manifest_file 参数指定一个清单文件的 Amazon S3 对象键，该清单文件提供 COPY 将用于建立 SSH 连接并执行远程命令的信息。

s3://copy_from_ssh_manifest_file 参数必须显式引用单个文件；它不能是键前缀。下面是一个示例：

```
's3://mybucket/ssh_manifest.txt'
```

清单文件是 Amazon Redshift 用于连接主机的文本文件，采用 JSON 格式。清单文件指定 SSH 主机端点以及将在主机上运行的用于将数据返回到 Amazon Redshift 的命令。另外，您还可以包含主机公有密钥、登录用户名和每个条目的 mandatory 标志。以下示例显示了用于创建两个 SSH 连接的清单文件：

```
{
    "entries": [
```

```
    {"endpoint":"<ssh_endpoint_or_IP>",
          "command": "<remote_command>",
          "mandatory":true,
          "publickey": "<public_key>",
          "username": "<host_user_name>"},
    {"endpoint":"<ssh_endpoint_or_IP>",
          "command": "<remote_command>",
          "mandatory":true,
          "publickey": "<public_key>",
          "username": "<host_user_name>"}
    ]
}
```

该清单文件为每个 SSH 连接包含一个 "entries" 结构。您可以与单台主机建立多个连接或与多台主机建立多个连接。如上所示，字段名称和值均需要使用双引号字符。引号字符必须是简单引号 (0x22)，而不能是倾斜引号或“智能”引号。唯一一个不需要双引号字符的值是 "mandatory" 字段的布尔值 true 或 false。

以下列表介绍了清单文件中的字段。

endpoint

主机的 URL 地址或 IP 地址，例如 "ec2-111-222-333.compute-1.amazonaws.com" 或 "198.51.100.0"。

命令

命令通过主机运行，用以产生 gzip、lzop、bzip2 或 zstd 格式的文本输出或二进制输出。该命令可以是用户 "host_user_name" 有权运行的任何命令。该命令可以是像打印文件这样简单的命令，也可以查询数据库或启动脚本。输出（文本文件、gzip 二进制文件、lzop 二进制文件或 bzip2 二进制文件）必须采用 Amazon Redshift COPY 命令可摄取的形式。有关更多信息，请参阅 准备输入数据。

publickey

（可选）主机的公有密钥。如果提供了公有密钥，Amazon Redshift 将使用它来标识主机。如果未提供公有密钥，Amazon Redshift 将不会尝试主机标识。例如，如果远程主机的公有密钥是 ssh-rsa AbcCbaxxx…Example root@amazon.com，请在公有密钥字段中键入以下文本："AbcCbaxxx…Example"

mandatory

（可选）一个子句，指示在连接尝试失败时 COPY 命令是否应失败。默认为 false。如果 Amazon Redshift 未成功建立至少一个连接，COPY 命令将失败。

username

（可选）将用于登录到主机系统并执行远程命令的用户名。用户登录名必须与用于将 Amazon Redshift 集群的公有密钥添加到主机的授权密钥文件的登录名相同。默认用户名为 `redshift`。

有关创建清单文件的更多信息，请参阅加载数据的过程。

要从远程主机执行 COPY 操作，则必须在 COPY 命令中指定 SSH 参数。如果未指定 SSH 参数，COPY 命令将假定使用 FROM 指定的文件是数据文件，操作将会失败。

如果使用自动压缩，COPY 命令将执行两个数据读取操作，这意味着它将执行远程命令两次。第一个读取操作用于提供压缩分析的数据样本，第二个读取操作实际加载数据。如果执行远程命令两次可能会导致问题，则应禁用自动压缩。要禁用自动压缩，请在运行 COPY 命令时将 COMPUPDATE 参数设置为 OFF。有关更多信息，请参阅 使用自动压缩加载表。

有关从 SSH 执行 COPY 操作的详细过程，请参阅从远程主机中加载数据。

授权

COPY 命令需要授权才能访问其他 AWS 资源（包括 Amazon S3 、Amazon EMR、Amazon DynamoDB 和 Amazon EC2）中的数据。您可通过引用附加到您的集群的 AWS Identity and Access Management (IAM) 角色（基于角色的访问控制）或者通过为用户提供访问凭证（基于密钥的访问控制）来提供授权。为了提高安全性和灵活性，我们建议使用基于 IAM 角色的访问控制。有关更多信息，请参阅 授权参数。

SSH

一个子句，指定要从使用 SSH 协议的远程主机加载数据。如果指定 SSH，则必须使用 s3://copy_from_ssh_manifest_file 参数提供清单文件。

> **Note**
>
> 如果您通过 SSH 在远程 VPC 中使用私有 IP 地址从主机进行复制，则 VPC 必须启用增强型 VPC 路由。有关增强型 VPC 路由的更多信息，请参阅 Amazon Redshift 增强型 VPC 路由。

可选参数

对于从 SSH 执行 COPY 的操作，还可以选择指定以下参数：

- 列映射选项
- 数据格式参数
- 数据转换参数
- 数据加载操作

不支持的参数

对于从 SSH 执行 COPY 的操作，不能使用以下参数：

- ENCRYPTED
- MANIFEST
- READRATIO

从 Amazon DynamoDB 执行 COPY 操作

要从现有 DynamoDB 表加载数据，请使用 FROM 子句指定 DynamoDB 表名称。

主题

- 语法
- 示例
- 可选参数
- 不支持的参数

> **Important**
>
> 如果 DynamoDB 表未驻留在您的 Amazon Redshift 集群所在的区域内，则必须使用 REGION 参数指定该数据所在的区域。

语法

```
FROM 'dynamodb://table-name'
authorization
READRATIO ratio
| REGION [AS] 'aws_region'
```

```
| optional-parameters
```

示例

以下示例从 DynamoDB 表加载数据。

```
copy favoritemovies from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

参数

FROM

要加载的数据的源。

'dynamodb://table-name'

包含数据的 DynamoDB 表的名称，例如 'dynamodb://ProductCatalog'。有关 DynamoDB 属性如何映射到 Amazon Redshift 列的详细信息，请参阅从 Amazon DynamoDB 表中加载数据。

DynamoDB 表名称对于由 AWS 访问凭证标识的 AWS 账户是唯一的。

授权

COPY 命令需要授权才能访问其他 AWS 资源（包括 Amazon S3 、Amazon EMR、DynamoDB 和 Amazon EC2）中的数据。您可通过引用附加到您的集群的 AWS Identity and Access Management (IAM) 角色（基于角色的访问控制）或者通过为用户提供访问凭证（基于密钥的访问控制）来提供授权。为了提高安全性和灵活性，我们建议使用基于 IAM 角色的访问控制。有关更多信息，请参阅授权参数。

READRATIO [AS] ratio

DynamoDB 表的预配置吞吐量中要用于数据加载的部分所占的百分比。从 DynamoDB 执行 COPY 的操作需要 READRATIO。它不能在从 Amazon S3 执行 COPY 的操作中使用。我们强烈建议您将此比率设置为一个低于平均的未使用预配置吞吐量的值。有效值为整数 1–200。

> **Important**
>
> 将 READRATIO 设置为 100 或更大值将使 Amazon Redshift 消耗 DynamoDB 表的全部预配置吞吐量，从而严重降低 COPY 会话期间对同一个表进行的并行读取操作的性能。写入流量不受影响。允许使用大于 100 的值来应对 Amazon Redshift 无法满足表的预配置吞吐

量的罕见情况。如果将 DynamoDB 中的数据持续加载到 Amazon Redshift，请考虑按时间序列组织 DynamoDB 表以将实时流量与 COPY 操作分离。

可选参数

对于从 Amazon DynamoDB 执行 COPY 的操作，还可以指定以下参数：

- 列映射选项
- 支持以下数据转换参数：
  - ACCEPTANYDATE
  - BLANKSASNULL
  - DATEFORMAT
  - EMPTYASNULL
  - ROUNDEC
  - TIMEFORMAT
  - TRIMBLANKS
  - TRUNCATECOLUMNS
- 数据加载操作

不支持的参数

对于从 DynamoDB 执行 COPY 的操作，不能使用以下参数：

- 所有数据格式参数
- ESCAPE
- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL
- REMOVEQUOTES
- ACCEPTINVCHARS
- MANIFEST

- ENCRYPTED

授权参数

COPY 命令需要授权才能访问其他 AWS 资源（包括 Amazon S3、Amazon EMR、Amazon DynamoDB 和 Amazon EC2）中的数据。您通过引用附加到集群的 AWS Identity and Access Management (IAM) 角色来提供授权（基于角色的访问控制）。您可以在 Amazon S3 上加密您的加载数据。

以下主题将提供有关身份验证选项的更多详细信息和示例：

- COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限
- 基于角色的访问控制
- 基于密钥的访问控制

使用以下参数之一为 COPY 命令提供授权：

- IAM_ROLE parameter
- ACCESS_KEY_ID and SECRET_ACCESS_KEY 参数
- CREDENTIALS 子句

IAM_ROLE { default | 'arn:aws:iam::*<AWS ##-id>*:role/*<role-name>*' }

使用默认关键字让 Amazon Redshift 使用设置为默认值并在 COPY 命令运行时与集群关联的 IAM 角色。

使用 IAM 角色的 Amazon 资源名称 (ARN)，您的集群使用该角色进行身份验证和授权。如果您指定 IAM_ROLE，则无法使用 ACCESS_KEY_ID 和 SECRET_ACCESS_KEY、SESSION_TOKEN 或 CREDENTIALS。

以下显示 IAM_ROLE 参数的语法。

```
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' }
```

有关更多信息，请参阅 基于角色的访问控制。

ACCESS_KEY_ID 'access-key-id ' SECRET_ACCESS_KEY 'secret-access-key'

不建议您使用此授权方法。

> **Note**
>
> 我们强烈建议通过指定 IAM_ROLE 参数使用基于角色的身份验证，而不是提供纯文本形式的访问凭证。有关更多信息，请参阅 基于角色的访问控制。

SESSION_TOKEN 'temporary-token'

与临时访问凭证配合使用的会话令牌。如果指定 SESSION_TOKEN，还必须使用 ACCESS_KEY_ID 和 SECRET_ACCESS_KEY 提供临时访问密钥凭证。如果指定 SESSION_TOKEN，则不能使用 IAM_ROLE 或 CREDENTIALS。有关更多信息，请参阅《IAM 用户指南》中的临时安全凭证。

> **Note**
>
> 我们强烈建议使用基于角色的身份验证，而不是创建临时安全凭证。如果您授权使用 IAM 角色，Amazon Redshift 会自动为每个会话创建临时用户凭证。有关更多信息，请参阅 基于角色的访问控制。

以下显示 SESSION_TOKEN 参数与 ACCESS_KEY_ID 和 SECRET_ACCESS_KEY 参数配合使用时的语法。

```
ACCESS_KEY_ID '<access-key-id>'
SECRET_ACCESS_KEY '<secret-access-key>'
SESSION_TOKEN '<temporary-token>';
```

如果指定 SESSION_TOKEN，则不能使用 CREDENTIALS 或 IAM_ROLE。

[WITH] CREDENTIALS [AS] 'credentials-args'

一个子句，指示您的集群在访问包含数据文件或清单文件的其他 AWS 资源时将使用的方法。CREDENTIALS 参数不能与 IAM_ROLE 或 ACCESS_KEY_ID 和 SECRET_ACCESS_KEY 配合使用。

> **Note**
>
> 要获得更高的灵活性，我们建议使用 IAM_ROLE 参数，而不是 CREDENTIALS 参数。

（可选）如果使用了 ENCRYPTED 参数，credentials-args 字符串还将提供加密密钥。

credentials-args 字符串区分大小写且不得包含空格。

关键字 WITH 和 AS 是可选的，将被忽略。

您可指定 role-based access control 或 key-based access control。在任一情况下，IAM 角色或用户都必须具有访问指定 AWS 资源所需的权限。有关更多信息，请参阅 COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限。

> **Note**
>
> 为了保护您的 AWS 凭证和敏感数据，我们强烈建议使用基于角色的访问控制。

要指定基于角色的访问控制，请按以下格式提供 credentials-args 字符串。

```
'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
```

要使用临时令牌凭证，您必须提供临时访问密钥 ID、临时秘密访问密钥和临时令牌。credentials-args 字符串采用以下格式。

```
CREDENTIALS
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;token=<temporary-token>'
```

有关更多信息，请参阅 临时安全凭证。

如果使用了 ENCRYPTED 参数，credentials-args 字符串将采用以下格式，其中 *<root-key>* 是用于对文件进行加密的根密钥的值。

```
CREDENTIALS
'<credentials-args>;master_symmetric_key=<root-key>'
```

例如，以下 COPY 命令使用带加密密钥的基于角色的访问控制。

```
copy customer from 's3://mybucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::<account-id>:role/<role-name>;master_symmetric_key=<root-key>'
```

以下 COPY 命令显示了带加密密钥的基于角色的访问控制。

```
copy customer from 's3://mybucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-
name>;master_symmetric_key=<root-key>'
```

列映射选项

默认情况下，COPY 会按字段在数据文件中出现的相同顺序将值插入到目标表的列中。如果默认列顺序不起作用，则可以指定一个列列表或使用 JSONPath 表达式将源数据字段映射到目标列。

- Column List
- JSONPaths File

列列表

您可以指定列名称的逗号分隔列表以将源数据字段加载到特定目标列中。这些列在 COPY 语句中可以采用任何顺序，但是，当从平面文件加载时（如在 Amazon S3 桶中），它们的顺序必须与源数据的顺序一致。

从 Amazon DynamoDB 表加载时，顺序并不重要。COPY 命令将从 DynamoDB 表中检索到的项中的属性名称与 Amazon Redshift 表中的列名进行匹配。有关更多信息，请参阅从 Amazon DynamoDB 表中加载数据

列列表的格式如下所示。

```
COPY tablename (column1 [,column2, ...])
```

如果列列表省略了目标表中的列，则 COPY 将加载目标列的 DEFAULT 表达式。

如果目标列没有默认值，则 COPY 将尝试加载 NULL。

如果 COPY 尝试将 NULL 分配到一个定义为 NOT NULL 的列，COPY 命令将失败。

如果 IDENTITY 列包含在列列表中，则还必须指定 EXPLICIT_IDS；如果省略了 IDENTITY 列，则无法指定 EXPLICIT_IDS。如果未指定任何列列表，则该命令将如同指定了一个完整、有序的列列表一样来执行，如果也未指定 EXPLICIT_IDS，则会省略 IDENTITY 列。

如果某个列使用 GENERATED BY DEFAULT AS IDENTITY 进行定义，则可以复制该列。使用您提供的值生成或更新值。EXPLICIT_IDS 选项不是必需项。COPY 不会更新身份高级别水印。有关更多信息，请参阅 GENERATED BY DEFAULT AS IDENTITY。

JSONPaths 文件

当从 JSON 或 Avro 格式的数据文件加载时，COPY 会将 JSON 或 Avro 源数据中的数据元素自动映射到目标表中的列。它的执行方式是通过将 Avro schema 中的字段名称与目标表或列列表中的列名称相匹配。

在某些情况下，您的列名称与字段名称不匹配，或者您需要映射到数据层次结构中的更深层次。在这些情况下，您可以使用 JSONPaths 文件将 JSON 或 Avro 数据元素显式映射到列。

有关更多信息，请参阅 JSONPaths 文件。

数据格式参数

默认情况下，COPY 命令要求源数据是字符分隔的 UTF-8 文本。默认分隔符是竖线字符 (|)。如果源数据采用的是其他格式，请使用以下参数指定数据格式：

- FORMAT
- CSV
- DELIMITER
- FIXEDWIDTH
- SHAPEFILE
- AVRO
- JSON
- PARQUET
- ORC

除标准数据格式以外，COPY 支持 Amazon S3 中有关 COPY 的以下列式数据格式：

- ORC
- PARQUET

支持列式中的 COPY，其中带有特定限制。有关更多信息，请参阅 从列式数据格式中执行 COPY 操作。

数据格式参数

FORMAT [AS]

（可选）标识数据格式关键字。FORMAT 参数如下所述。

CSV [QUOTE [AS] 'quote_character']

支持在输入数据中使用 CSV 格式。要自动对分隔符、换行符和回车符进行转义，可用 QUOTE 参数指定的字符将字段括起来。默认引号字符是双引号 (")。当在字段中使用了引号字符时，应使用另一个引号字符对其进行转义。例如，如果引号字符为双引号，那么要插入字符串 `A "quoted" word`，输入文件应包含字符串 `"A ""quoted"" word"`。当使用了 CSV 参数时，默认分隔符为逗号 (,)。您可使用 DELIMITER 参数指定一个不同的分隔符。

当某个字段用引号括起来时，分隔符和引号字符之间的空格将被忽略。如果分隔符为空格字符（如制表符），则分隔符不会被视为空格。

CSV 不能与 FIXEDWIDTH、REMOVEQUOTES 或 ESCAPE 一起使用。

QUOTE [AS] 'quote_character'

可选。指定在使用 CSV 参数时要用作引号字符的字符。默认值为双引号 (")。如果您使用 QUOTE 参数定义双引号以外的引号字符，则不需要对字段中的双引号进行转义。QUOTE 参数只能与 CSV 参数一起使用。AS 关键字是可选的。

DELIMITER [AS] ['delimiter_char']

指定用于在输入文件中分隔字段的单个 ASCII 字符，如竖线字符 (|)、逗号 (,) 或制表符 (\t)。支持非打印 ASCII 字符。ASCII 字符还可以用八进制形式表示，使用格式“\ddd”，其中“d”是八进制数字 (0–7)。默认分隔符是竖线字符 (|)，除非使用了 CSV 参数，在这种情况下，默认分隔符是逗号 (,)。AS 关键字是可选的。DELIMITER 不能与 FIXEDWIDTH 一起使用。

FIXEDWIDTH 'fixedwidth_spec'

从一个文件中加载数据，该文件中的每个列是宽度固定的列，而不是由分隔符分隔的列。fixedwidth_spec 是用于指定用户定义的列标签和列宽度的字符串。列标签可以是文本字符串或整数，具体取决于用户的选择。列标签与列名称没有关联。标签/宽度对的顺序必须与表列的顺序完全一致。FIXEDWIDTH 不能与 CSV 或 DELIMITER 一起使用。在 Amazon Redshift 中，CHAR 和 VARCHAR 列的长度以字节表示，因此在准备要加载的文件时，请确保您指定的列宽度可容纳多字节字符的二进制长度。有关更多信息，请参阅 字符类型。

fixedwidth_spec 的格式如下所示：

```
'colLabel1:colWidth1,colLabel:colWidth2, ...'
```

SHAPEFILE [SIMPLIFY [AUTO] ['tolerance']]

支持在输入数据中使用 SHAPEFILE 格式。预设情况下，shapefile 的第一列是 GEOMETRY 或 IDENTITY 列。所有后续列都遵循 shapefile 中指定的顺序。

您不能将 SHAPEFILE 与 FIXEDWIDTH、REMOVEQUOTES 或 ESCAPE 一起使用。

要将 GEOGRAPHY 对象与 COPY FROM SHAPEFILE 一起使用，请首先提取到 GEOMETRY 列，然后将对象强制转换为 GEOGRAPHY 对象。

SIMPLIFY [tolerance]

（可选）使用 Ramer-Douglas-Peucker 算法和给定的容差简化摄入过程中的所有几何体。

SIMPLIFY AUTO [tolerance]

（可选）仅简化大于最大几何大小的几何体。这种简化使用 Ramer-Douglas-Peucker 算法和自动计算的容差（如果不超过指定容差）。此算法计算在指定容差范围内存储对象的大小。公差值是可选的。

有关加载 shapefile 的示例，请参阅将 shapefile 加载到 Amazon Redshift。

AVRO [AS] 'avro_option'

指定源数据采用 Avro 格式。

从以下服务和协议执行 COPY 的操作支持 Avro 格式：

- Amazon S3
- Amazon EMR
- 远程主机 (SSH)

从 DynamoDB 执行 COPY 的操作不支持 Avro。

Avro 是一个数据序列化协议。Avro 源文件包含一个定义数据结构的 schema。Avro schema 类型必须为 record。COPY 接受使用默认的非压缩编解码器及 deflate 和 snappy 压缩编解码器创建的 Avro 文件。有关 Avro 的更多信息，请转到 Apache Avro。

avro_option 的有效值如下：

- 'auto'
- 'auto ignorecase'
- 's3://*jsonpaths_file*'

默认为 'auto'。

COPY 会将 Avro 源数据中的数据元素自动映射到目标表中的列。它的执行方式是通过将 Avro schema 中的字段名称与目标表中的列名称相匹配。'auto' 的匹配区分大小写，'auto ignorecase' 的匹配不区分大小写。

Amazon Redshift 表中的列名称始终小写，因此，当您使用 'auto' 选项时，匹配的字段名称也必须为小写。如果字段名称不是全部小写，则可以使用 'auto ignorecase' 选项。使用默认的 'auto' 参数时，COPY 仅识别结构中的第一层字段，或外部字段。

要将列名称显式映射到 Avro 字段名称，您可以使用 JSONPaths 文件。

默认情况下，COPY 会尝试将目标表中的所有列与 Avro 字段名称匹配。要加载列的子集，您可以选择性地指定包含列的列表。如果列列表中省略了目标表中的列，则 COPY 将加载目标列的 DEFAULT 表达式。如果目标列没有默认值，则 COPY 将尝试加载 NULL。如果某个列包含在列列表中，并且 COPY 在 Avro 数据中找不到匹配的字段，则 COPY 会尝试将 NULL 加载到该列中。

如果 COPY 尝试将 NULL 分配到一个定义为 NOT NULL 的列，COPY 命令将失败。

Avro Schema

Avro 源数据文件包含一个定义数据结构的 Schema。COPY 将读取作为 Avro 源数据文件的一部分的 schema 以将数据元素映射到目标表列。以下示例显示了一个 Avro schema。

```
{
    "name": "person",
    "type": "record",
    "fields": [
        {"name": "id", "type": "int"},
        {"name": "guid", "type": "string"},
        {"name": "name", "type": "string"},
        {"name": "address", "type": "string"}]
}
```

Avro schema 是使用 JSON 格式定义的。顶级 JSON 对象包含三个名称-值对，这三个名称（即键）分别为 "name"、"type" 和 "fields"。

"fields" 键与定义数据结构中每个字段的名称和数据类型的对象数组配对。默认情况下，COPY 会自动将字段名称与列名称匹配。列名称始终为小写形式，因此匹配的字段名称也必须为小写形式，除非您指定了 'auto ignorecase' 选项。与列名称不匹配的任何字段名称都将被忽略。顺序无关紧要。在上述示例中，COPY 将映射到列名称 id、guid、name 和 address。

由于存在默认的 'auto' 参数，COPY 只会将第一层对象映射到列。若要映射到 schema 中的更深层次，或者如果字段名称与列名称不匹配，请使用 JSONPaths 文件定义映射。有关更多信息，请参阅 JSONPaths 文件。

如果与键关联的值是一个复杂的 Avro 数据类型（如字节、数组、记录、映射或链接），COPY 会将该值作为一个字符串加载。这里的字符串是数据的 JSON 表示形式。COPY 会将 Avro 枚举数据类型作为字符串加载，其中的内容是类型的名称。有关示例，请参阅 从 JSON 格式数据执行的 COPY 操作。

Avro 文件标头（包括 schema 和文件元数据）的最大大小为 1 MB。

单个 Avro 数据块的最大大小为 4 MB。这与最大行大小不同。如果超过了单个 Avro 数据块的最大大小，则即使生成的行大小未达到 4 MB 的行大小限制，COPY 命令也会失败。

在计算行大小时，Amazon Redshift 在内部对竖线字符 (|) 计为两个字符。如果您的输入数据中包含大量竖线字符，则即使数据块小于 4 MB，行大小也可能超过 4 MB。

JSON [AS] 'json_option'

源数据采用 JSON 格式。

从以下服务和协议执行 COPY 的操作支持 JSON 格式：

- Amazon S3
- 从 Amazon EMR 执行 COPY 操作
- 从 SSH 执行 COPY 的操作

从 DynamoDB 执行 COPY 的操作不支持 JSON。

json_option 的有效值如下：

- 'auto'
- 'auto ignorecase'
- 's3://*jsonpaths_file*'
- 'noshred'

默认为 'auto'。在加载 JSON 文档时，Amazon Redshift 不会将 JSON 结构的属性分解为多个列。

默认情况下，COPY 会尝试将目标表中的所有列与 JSON 字段名称键匹配。要加载列的子集，您可以选择性地指定包含列的列表。如果 JSON 字段名称键包含大写字符，则您可以使用 'auto ignorecase' 选项或 JSONPaths 文件 将列名称显式地映射到 JSON 字段名称键。

如果列列表省略了目标表中的列，则 COPY 将加载目标列的 DEFAULT 表达式。如果目标列没有默认值，则 COPY 将尝试加载 NULL。如果某个列包含在列列表中，并且 COPY 在 JSON 数据中找不到匹配的字段，则 COPY 会尝试将 NULL 加载到该列。

如果 COPY 尝试将 NULL 分配到一个定义为 NOT NULL 的列，COPY 命令将失败。

COPY 会将 JSON 源数据中的数据元素映射到目标表中的列。它的操作方式是通过将源名称-值对中的对象键（即名称）与目标表中的列名称匹配。

请参阅以下有关每个 json_option 值的详细信息：

'auto'

使用此选项时，匹配区分大小写。Amazon Redshift 表中的列名称始终小写，因此，当您使用 `'auto'` 选项时，匹配的 JSON 字段名称也必须为小写。

“auto ignorecase”

使用此选项时，匹配不区分大小写。Amazon Redshift 表中的列名称始终小写，因此，当您使用 `'auto ignorecase'` 选项时，相应的 JSON 字段名称可以是小写、大写或大小写混合。

's3://jsonpaths_file'

通过此选项，COPY 使用命名的 JSONPaths 文件将 JSON 源数据中的数据元素映射到目标表中的列。*`s3://jsonpaths_file`* 参数必须是显式引用单个文件的 Amazon S3 对象键。示例是 `'s3://mybucket/jsonpaths.txt'`。参数不能为键前缀。有关使用 JSONPaths 文件的更多信息，请参阅 the section called “JSONPaths 文件”。

在某些情况下，由 `jsonpaths_file` 指定的文件的前缀与由 `copy_from_s3_objectpath` 为数据文件指定的路径的前缀相同。如果是这样，COPY 会将 JSONPaths 文件作为数据文件读取并返回错误。例如，假设您的数据文件使用对象路径 `s3://mybucket/my_data.json`，并且您的 JSONPaths 文件是 `s3://mybucket/my_data.jsonpaths`。在这种情况下，COPY 会尝试加载 `my_data.jsonpaths` 作为数据文件。

“noshred”

使用此选项，Amazon Redshift 不会在加载 JSON 文档时将 JSON 结构的属性分解为多个列。

JSON 数据文件

JSON 数据文件包含一组对象或数组。COPY 会将每个 JSON 对象或数组加载到目标表中的一行中。与某个行对应的每个对象或数组都必须是独立的根级结构；即，它不能是另一个 JSON 结构的成员。

JSON 对象 以大括号 ({ }) 开头和结尾，并包含名称-值对的无序集合。每个成对的名称和值由冒号分隔，而每个名称/值对由逗号分隔。预设情况下，名称-值对中的对象键（即名称）必须与表中的对应列的名称匹配。Amazon Redshift 表中的列名称始终小写，因此，匹配的 JSON 字段名称键也必须为小写。如果您的列名称与 JSON 键不匹配，请使用 the section called “JSONPaths 文件” 将列显式映射到键。

JSON 对象中的顺序不重要。与列名称不匹配的任何名称都将被忽略。下面显示了一个简单 JSON 对象的结构。

```
{
  "column1": "value1",
  "column2": value2,
  "notacolumn" : "ignore this value"
}
```

JSON 数组 以中括号 ([]) 开头和结尾，并包含由逗号分隔的值的有序集合。如果您的数据文件使用了数组，则必须指定 JSONPaths 文件以将值与列匹配。下面显示了一个简单 JSON 数组的结构。

```
["value1", value2]
```

JSON 必须格式正确。例如，对象或数组不能用逗号或除空格以外的任何其他字符分隔。字符串必须括在双引号字符中。引号字符必须是简单引号 (0x22)，而不能是倾斜引号或“智能”引号。

单个 JSON 对象或数组（包括大括号或中括号）的最大大小为 4 MB。这与最大行大小不同。如果超过了单个 JSON 对象或数组的最大大小，则即使生成的行大小未达到 4 MB 的行大小限制，COPY 命令也会失败。

在计算行大小时，Amazon Redshift 在内部对竖线字符 (|) 计为两个字符。如果您的输入数据中包含大量竖线字符，则即使对象大小小于 4 MB，行大小也可能超过 4 MB。

COPY 会将 \n 作为换行符加载并且会将 \t 作为制表符加载。要加载反斜杠，请使用反斜杠 (\\ ) 对其进行转义。

COPY 将在指定的 JSON 源中搜索格式正确且有效的 JSON 对象或数组。如果 COPY 在找到可用的 JSON 结构之前遇到任何非空格字符，或在有效的 JSON 对象或数组之间遇到此类字符，COPY 将为每个实例返回错误。这些错误将计入 MAXERROR 错误计数。当错误计数等于或超过 MAXERROR 时，COPY 将失败。

对于每个错误，Amazon Redshift 都会在 STL_LOAD_ERRORS 系统表中记录一行。LINE_NUMBER 列将记录导致错误的 JSON 对象的最后一行。

如果指定了 IGNOREHEADER，COPY 将忽略 JSON 数据中指定数量的行。JSON 数据中的换行符始终计入到 IGNOREHEADER 计算中。

默认情况下，COPY 将空字符串作为空字段加载。如果指定了 EMPTYASNULL，COPY 会将 CHAR 和 VARCHAR 字段的空字符串作为 NULL 加载。其他数据类型（如 INT）的空字符串始终作为 NULL 加载。

不支持将以下选项与 JSON 一起使用：

- CSV
- DELIMITER
- ESCAPE
- FILLRECORD
- FIXEDWIDTH
- IGNOREBLANKLINES
- NULL AS
- READRATIO
- REMOVEQUOTES

有关更多信息，请参阅 从 JSON 格式数据执行的 COPY 操作。有关 JSON 数据结构的更多信息，请转到 www.json.org。

JSONPaths 文件

如果您正在从 JSON 格式的源数据或 Avro 源数据加载，则在预设情况下，COPY 会将源数据中的第一层数据元素映射到目标表中的列。它的操作方式是通过将名称-值对中的每个名称（即对象键）与目标表中的列的名称匹配。

如果您的列名称与对象键不匹配，或要映射到数据层次结构中的更深层次，则可以使用 JSONPaths 文件将 JSON 或 Avro 数据元素显式映射到列。JSONPaths 文件通过匹配目标表或列列表中的列顺序来将 JSON 数据元素映射到列。

JSONPaths 文件只能包含一个 JSON 对象（非数组）。JSON 对象是一个名称-值对。对象键（即名称-值对的名称）必须为 `"jsonpaths"`。名称-值对中的值 是一组 JSONPath 表达式。每个 JSONPath 表达式都引用 JSON 数据层次结构或 Avro schema 中的一个元素，这与 XPath 表达式引用 XML 文档中的元素相似。有关更多信息，请参阅 JSONPath 表达式。

要使用 JSONPaths 文件，请将 JSON 或 AVRO 关键字添加到 COPY 命令。使用以下格式指定 JSONPath 文件的 S3 桶名称和对象路径。

```
COPY tablename
FROM 'data_source'
CREDENTIALS 'credentials-args'
FROM AS { AVRO | JSON } 's3://jsonpaths_file';
```

s3://jsonpaths_file 参数必须是显式引用单个文件（如 's3://mybucket/jsonpaths.txt'）的 Amazon S3 对象键。它不能是键前缀。

在某些情况下，如果您从 Amazon S3 加载，由 jsonpaths_file 指定的文件的前缀与由 copy_from_s3_objectpath 为数据文件指定的路径的前缀相同。如果是这样，COPY 会将 JSONPaths 文件作为数据文件读取并返回错误。例如，假设您的数据文件使用对象路径 s3://mybucket/my_data.json，并且您的 JSONPaths 文件是 s3://mybucket/my_data.jsonpaths。在这种情况下，COPY 会尝试加载 my_data.jsonpaths 作为数据文件。

如果键名称是除 "jsonpaths" 以外的任何字符串，则 COPY 命令不会返回错误，但会忽略 jsonpaths_file 并改为使用 'auto' 参数。

如果出现以下任一情况，COPY 命令将失败：

- JSON 格式不正确。
- 存在多个 JSON 对象。
- 对象外部存在除空格以外的任何字符。
- 数组元素是一个空字符串或者不是一个字符串。

MAXERROR 不适用于 JSONPaths 文件。

即使指定了 ENCRYPTED 选项，也不得加密 JSONPaths 文件。

有关更多信息，请参阅 从 JSON 格式数据执行的 COPY 操作。

JSONPath 表达式

JSONPaths 文件使用 JSONPath 表达式将数据字段映射到目标列。每个 JSONPath 表达式对应于 Amazon Redshift 目标表中的一个列。JSONPath 数组元素的顺序必须与目标表或列列表（如果使用了列列表）中列的顺序一致。

如上所示，字段名称和值均需要使用双引号字符。引号字符必须是简单引号 (0x22)，而不能是倾斜引号或“智能”引号。

如果 JSONPath 表达式引用的对象元素在 JSON 数据中找不到，则 COPY 将尝试加载 NULL 值。如果引用的对象的格式不正确，则 COPY 将返回加载错误。

如果 JSONPath 表达式引用的数组元素在 JSON 或 Avro 数据中找不到，则 COPY 将失败并返回以下错误：`Invalid JSONPath format: Not an array or index out of range.`请从 JSONPaths 中删除在源数据中不存在的所有数组元素，并确认源数据中数组的格式正确。

JSONPath 表达式可使用括号表示法或点表示法，但不能将两者结合使用。以下示例显示了使用括号表示法的 JSONPath 表达式。

```
{
    "jsonpaths": [
        "$['venuename']",
        "$['venuecity']",
        "$['venuestate']",
        "$['venueseats']"
    ]
}
```

以下示例显示了使用点表示法的 JSONPath 表达式。

```
{
    "jsonpaths": [
        "$.venuename",
        "$.venuecity",
        "$.venuestate",
        "$.venueseats"
    ]
}
```

在 Amazon Redshift COPY 语法的上下文中，JSONPath 表达式必须指定 JSON 或 Avro 分层数据结构中单个名称元素的显式路径。Amazon Redshift 不支持可能解析为不确定路径或多个名称元素的任何 JSONPath 元素（如通配符或筛选表达式）。

有关更多信息，请参阅 从 JSON 格式数据执行的 COPY 操作。

将 JSONPaths 与 Avro 数据一起使用

以下示例显示了具有多个层次 Avro schema。

```
{
    "name": "person",
    "type": "record",
    "fields": [
        {"name": "id", "type": "int"},
        {"name": "guid", "type": "string"},
        {"name": "isActive", "type": "boolean"},
        {"name": "age", "type": "int"},
        {"name": "name", "type": "string"},
        {"name": "address", "type": "string"},
        {"name": "latitude", "type": "double"},
        {"name": "longitude", "type": "double"},
        {
            "name": "tags",
            "type": {
                        "type" : "array",
                        "name" : "inner_tags",
                        "items" : "string"
                    }
        },
        {
            "name": "friends",
            "type": {
                        "type" : "array",
                        "name" : "inner_friends",
                        "items" : {
                                    "name" : "friends_record",
                                    "type" : "record",
                                    "fields" : [
                                                {"name" : "id", "type" : "int"},
                                                {"name" : "name", "type" : "string"}
                                               ]
                                  }
                    }
        },
        {"name": "randomArrayItem", "type": "string"}
    ]
}
```

以下示例显示了使用 AvroPath 表达式引用前面的 schema 的 JSONPaths 文件。

```
{
    "jsonpaths": [
```

```
        "$.id",
        "$.guid",
        "$.address",
        "$.friends[0].id"
    ]
}
```

JSONPaths 示例包含以下元素：

jsonpaths

包含 AvroPath 表达式的 JSON 对象的名称。

[…]

方括号将包含路径元素的 JSON 数组括起。

$

美元符号表示 Avro schema 中的根元素，即 "fields" 数组。

"$.id",

AvroPath 表达式的目标。在此实例中，目标是 "fields" 数组中名为 "id" 的元素。表达式用逗号分隔。

"$.friends[0].id"

方括号表示数组索引。JSONPath 表达式使用从零开始的索引，因此该表达式引用 "friends" 数组中名为 "id" 的第一个元素。

Avro schema 语法需要使用内部字段 来定义记录和数组数据类型的结构。AvroPath 表达式将会忽略内部字段。例如，字段 "friends" 定义了一个名为 "inner_friends" 的数组，该数组又定义了一个名为 "friends_record" 的记录。要引用字段 "id" 的 AvroPath 表达式可忽略额外字段以直接引用目标字段。以下 AvroPath 表达式引用了两个属于 "friends" 数组的字段。

```
"$.friends[0].id"
"$.friends[0].name"
```

列式数据格式参数

除标准数据格式以外，COPY 支持 Amazon S3 中有关 COPY 的以下列式数据格式。支持列式中的 COPY，其中带有特定限制。有关更多信息，请参阅 从列式数据格式中执行 COPY 操作。

ORC

从使用优化的行列式 (ORC) 文件格式的文件中加载数据。

PARQUET

从使用 Parquet 文件格式的文件中加载数据。

文件压缩参数

您可以通过指定以下参数来从压缩的数据文件加载。

文件压缩参数

BZIP2

一个值，用于指定输入文件采用压缩 bzip2 格式（.bz2 文件）。COPY 操作将读取每个压缩文件并在加载时解压数据。

GZIP

一个值，用于指定输入文件采用压缩 gzip 格式（.gz 文件）。COPY 操作将读取每个压缩文件并在加载时解压数据。

LZOP

一个值，用于指定输入文件采用压缩 lzop 格式（.lzo 文件）。COPY 操作将读取每个压缩文件并在加载时解压数据。

> **Note**
>
> COPY 不支持使用 lzop --filter 选项压缩的文件。

ZSTD

一个值，用于指定输入文件采用压缩 Zstandard 格式（.zst 文件）。COPY 操作将读取每个压缩文件并在加载时解压数据。有关更多信息，请参阅 ZSTD。

> **Note**
>
> 只有从 Amazon S3 进行 COPY 操作时，才支持 ZSTD。

数据转换参数

在加载表时，COPY 会尝试将源数据中的字符串隐式转换为目标列的数据类型。如果您需要指定不同于默认行为的转换，或者默认转换会产生错误，则可以通过指定以下参数来管理数据转换。有关这些参数语法的更多信息，请参阅 COPY 语法。

- ACCEPTANYDATE
- ACCEPTINVCHARS
- BLANKSASNULL
- DATEFORMAT
- EMPTYASNULL
- ENCODING
- ESCAPE
- EXPLICIT_IDS
- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL AS
- REMOVEQUOTES
- ROUNDEC
- TIMEFORMAT
- TRIMBLANKS
- TRUNCATECOLUMNS

数据转换参数

ACCEPTANYDATE

允许加载包括无效格式（如 00/00/00 00:00:00）在内的任何日期格式，而不生成错误。此参数仅适用于 TIMESTAMP 和 DATE 列。始终将 ACCEPTANYDATE 与 DATEFORMAT 参数结合使用。如果数据的日期格式与 DATEFORMAT 规范不匹配，则 Amazon Redshift 会将 NULL 值插入该字段中。

ACCEPTINVCHARS [AS] ['replacement_char']

允许将数据加载到 VARCHAR 列中，即使数据包含无效的 UTF-8 字符。如果指定 ACCEPTINVCHARS，则 COPY 会将每个无效的 UTF-8 字符替换为长度相等且包含由 replacement_char 指定的字符的字符串。例如，如果替换字符为“^”，则将使用“^^^”替换无效的三字节字符。

替换字符可以是除 NULL 之外的任何 ASCII 字符。默认值为一个问号 (?)。有关无效的 UTF-8 字符的信息，请参阅多字节字符加载错误。

COPY 将返回包含无效 UTF-8 字符的行的数量，并将为每个受影响的行在 STL_REPLACEMENTS 系统表中添加一个条目，每个节点切片最多有 100 行。还将替换其他无效的 UTF-8 字符，但不会记录这些替换事件。

如果未指定 ACCEPTINVCHARS，则 COPY 在遇到无效 UTF-8 字符时将返回错误。

ACCEPTINVCHARS 仅对 VARCHAR 列有效。

BLANKSASNULL

将仅包含空格字符的空白字段作为 NULL 加载。此选项仅适用于 CHAR 和 VARCHAR 列。其他数据类型（如 INT）的空白字段始终作为 NULL 加载。例如，包含三个连续的空格字符（并且无其他字符）的字符串将作为 NULL 加载。如果不使用此选项，则默认行为是按原样加载空白字符。

DATEFORMAT [AS] {'dateformat_string' | 'auto' }

如果未指定 DATEFORMAT，则默认格式为 'YYYY-MM-DD'。例如，一种有效的替代格式为 'MM-DD-YYYY'。

如果 COPY 命令未识别日期或时间值的格式，或者日期或时间值使用不同的格式，请将 'auto' 参数与 DATEFORMAT 或 TIMEFORMAT 参数结合使用。在使用 DATEFORMAT 和 TIMEFORMAT 字符串时，'auto' 参数将识别一些不受支持的格式。'auto' 的关键字区分大小写。有关更多信息，请参阅 在 DATEFORMAT 和 TIMEFORMAT 中使用自动识别。

日期格式可包含时间信息（小时、分钟、秒），但此信息将被忽略。AS 关键字是可选的。有关更多信息，请参阅 DATEFORMAT 和 TIMEFORMAT 字符串。

EMPTYASNULL

指示 Amazon Redshift 应将空 CHAR 和 VARCHAR 字段作为 NULL 加载。其他数据类型（如 INT）的空字段始终作为 NULL 加载。当数据包含两个连续的分隔符且分隔符之间没有字符时，将出现空字段。EMPTYASNULL 和 NULL AS ''（空字符串）将产生相同的行为。

ENCODING [AS] file_encoding

指定加载数据的编码类型。COPY 命令在加载过程中将数据从指定的编码转换为 UTF-8。

file_encoding 的有效值如下所示：

- UTF8
- UTF16
- UTF16LE
- UTF16BE

默认为 UTF8。

源文件名必须使用 UTF-8 编码。

下列文件必须使用 UTF-8 编码，即使为加载数据指定了不同的编码：

- 清单文件
- JSONPaths 文件

随下列参数提供的参数字符串必须使用 UTF-8：

- FIXEDWIDTH 'fixedwidth_spec'
- ACCEPTINVCHARS 'replacement_char'
- DATEFORMAT 'dateformat_string'
- TIMEFORMAT 'timeformat_string'
- NULL AS 'null_string'

固定宽度的数据文件必须使用 UTF-8 编码。字段宽度基于字符数，而不是字节数。

所有加载数据必须使用指定编码。如果 COPY 遇到不同的编码，将跳过文件并返回错误。

如果您指定 UTF16，则您的数据必须具有字节顺序标记 (BOM)。如果您知道您的 UTF-16 数据是否为 little-endian (LE) 或 big-endian (BE)，则不管是否存在 BOM，均可使用 UTF16LE 或 UTF16BE。

ESCAPE

指定此参数后，输入数据中的反斜杠字符 (\) 将被视为转义字符。紧跟在反斜杠字符后面的字符将作为当前列值的一部分加载到表中，即使它是通常用作特殊用途的字符。例如，您可使用此参数转义分隔符字符、引号、嵌入的换行符或转义字符本身，前提是这些字符中的任何字符是列值的合法部分。

如果您指定 ESCAPE 参数与 REMOVEQUOTES 参数的组合，则可转义并保留可能会被删除的引号（' 或 "）。默认 null 字符串 \N 按原样工作，但也可在输入数据中转义为 \\N。只要您未使用 NULL AS 参数指定替换 null 字符串，\N 和 \\N 就会产生相同的结果。

> **Note**
>
> 控制字符 0x00 (NUL) 无法转义，应从输入数据中删除或进行转换。此字符将被视为记录结束 (EOR) 标记，并导致记录的剩余部分被截断。

您无法对 FIXEDWIDTH 加载使用 ESCAPE 参数，并且无法指定转义字符本身；转义字符始终为反斜杠字符。此外，您必须确保输入数据在合适的位置包含转义字符。

下面是在指定 ESCAPE 参数的情况下的输入数据和产生的加载数据的一些示例。第 4 行的结果假设还指定了 REMOVEQUOTES 参数。输入数据包含两个用竖线分隔的字段：

```
1|The quick brown fox\[newline]
jumped over the lazy dog.
2| A\\B\\C
3| A \| B \| C
4| 'A Midsummer Night\'s Dream'
```

加载到第 2 列的数据看上去与下面类似：

```
The quick brown fox
jumped over the lazy dog.
A\B\C
A|B|C
A Midsummer Night's Dream
```

> **Note**
>
> 对加载的输入数据应用转义字符是用户的责任。此要求的一个例外情况是在您重新加载之前使用 ESCAPE 参数卸载的数据时。在此情况下，数据将已经包含必需的转义字符。

ESCAPE 参数不会解释 octal、hex、Unicode 或其他转义序列表示法。例如，如果您的源数据包含 octal 换行符值 (\012) 并且您尝试使用 ESCAPE 参数加载此数据，则 Amazon Redshift 会将值 012 加载到表中并且不会将此值解释为要转义的换行符。

为了转义源自 Microsoft Windows 平台的数据中的换行符，您可能需要使用两个转义字符：一个用于回车，一个用于换行。您也可以在加载文件（例如，通过使用 dos2unix 实用工具）之前删除回车符。

EXPLICIT_IDS

如果要将表中自动生成的值替换为源数据文件中的显式值，请对具有 IDENTITY 列的表使用 EXPLICIT_IDS。如果命令包含一个列列表，则该列表必须包含 IDENTITY 列才能使用此参数。EXPLICIT_IDS 值的数据格式必须与 CREATE TABLE 定义指定的 IDENTITY 格式匹配。

在对表运行带 EXPLICIT_IDS 选项的 COPY 命令时，Amazon Redshift 不会检查表中 IDENTITY 列的唯一性。

如果某个列使用 GENERATED BY DEFAULT AS IDENTITY 进行定义，则可以复制该列。使用您提供的值生成或更新值。EXPLICIT_IDS 选项不是必需项。COPY 不会更新身份高级别水印。

有关使用 EXPLICIT_IDS 的 COPY 命令的示例，请参阅加载具有显式的 IDENTITY 列值的 VENUE。

FILLRECORD

当一些记录的末尾缺少相邻列时，允许加载数据文件。将缺少的列加载为 NULL。对于文本和 CSV 格式，如果缺少的是 VARCHAR 列，则会加载零长度字符串而非 NULL。要从文本和 CSV 将 NULL 加载到 VARCHAR 列，请指定 EMPTYASNULL 关键字。仅当列定义允许 NULL 时，NULL 替换才会工作。

例如，如果表定义包含 4 个可以为 null 的 CHAR 列，并且记录包含值 `apple, orange, banana, mango`，则 COPY 命令可能加载并填充仅包含 `apple, orange` 值的记录。缺少的 CHAR 值将作为 NULL 值加载。

IGNOREBLANKLINES

忽略数据文件中仅包含换行符的空行并且不尝试加载它们。

IGNOREHEADER [AS] number_rows

将指定的 number_rows 视为文件标题并且不加载它们。使用 IGNOREHEADER 跳过并行加载的所有文件中的文件标题。

NULL AS 'null_string'

加载将 null_string 匹配为 NULL 的字段，其中 null_string 可以是任何字符串。如果您的数据包含 null 终止符（也称为 NUL (UTF-8 0000) 或二进制零 (0x000)），则 COPY 会将其视为任何其他字

符。例如，包含 '1' || NUL || '2' 的记录被复制为长度为 3 个字节的字符串。如果字段仅包含 NUL，您可使用 NULL AS 通过指定 '\0' 或 '\000' 来将 null 终止符替换为 NULL，例如，NULL AS '\0' 或 NULL AS '\000'。如果指定包含以 NUL 和 NULL AS 结尾的字符串的字段，则将在末尾处插入 NUL。请勿将“\n”（换行符）用于 null_string 值。Amazon Redshift 将保留“\n”以用作行分隔符。默认 null_string 为 '\N'。

> **Note**
>
> 如果您尝试将 null 加载到定义为 NOT NULL 的列中，则 COPY 命令将失败。

REMOVEQUOTES

删除传入数据中的字符串周围的引号。将保留引号中的所有字符（包括分隔符）。如果字符串具有开始单引号或双引号但没有对应的结束引号，则 COPY 命令将无法加载相应行并返回错误。下表显示了包含引号的字符串和最终加载值的一些简单示例。

输入字符串	使用 REMOVEQUOTES 选项加载的值
"The delimiter is a pipe (|) character"	The delimiter is a pipe (|) character
'Black'	Black
"White"	White
Blue'	Blue'
'Blue	Value not loaded: error condition
"Blue	Value not loaded: error condition
' 'Black' ' '	' 'Black' '
' '	<空格>

ROUNDEC

当输入值的小数位数超出列的小数位数时，会将数值向上取整。默认情况下，COPY 将在必要时截断值以匹配列的小数位数。例如，如果将值 20.259 加载到 DECIMAL(8,2) 列中，则 COPY 默认

情况下会将此值截断为 20.25。如果指定 ROUNDEC，则 COPY 会将值取整为 20.26。INSERT 命令始终在必要时将值取整以匹配列的小数位数，因此包含 ROUNDEC 参数的 COPY 命令的行为方式与 INSERT 命令相同。

TIMEFORMAT [AS] {'timeformat_string' | 'auto' | 'epochsecs' | 'epochmillisecs' }

指定时间格式。如果未指定 TIMEFORMAT，则默认格式为 YYYY-MM-DD HH:MI:SS（对于 TIMESTAMP 列）或 YYYY-MM-DD HH:MI:SSOF（对于 TIMESTAMPTZ 列），其中 OF 是与协调世界时 (UTC) 的时差。您不能在 timeformat_string 中包括时区标识符。要加载格式与默认格式不同的 TIMESTAMPTZ 数据，请指定“自动”；有关更多信息，请参阅 在 DATEFORMAT 和 TIMEFORMAT 中使用自动识别。有关 timeformat_string 的更多信息，请参阅 DATEFORMAT 和 TIMEFORMAT 字符串。

在使用 DATEFORMAT 和 TIMEFORMAT 字符串时，'auto' 参数将识别一些不受支持的格式。如果 COPY 命令未识别日期或时间值的格式，或者日期和时间值使用不同的格式，请将 'auto' 参数与 DATEFORMAT 或 TIMEFORMAT 参数结合使用。有关更多信息，请参阅 在 DATEFORMAT 和 TIMEFORMAT 中使用自动识别。

如果源数据以纪元时间（自 1970 年 1 月 1 日 00:00:00 UTC 以来的秒数或微秒数）表示，请指定 'epochsecs' 或 'epochmillisecs'。

'auto'、'epochsecs' 和 'epochmillisecs' 关键字区分大小写。

AS 关键字是可选的。

TRIMBLANKS

删除 VARCHAR 字符串的尾部空格字符。此参数仅适用于具有 VARCHAR 数据类型的列。

TRUNCATECOLUMNS

将列中的数据截断为合适的字符数以符合列规范。仅适用于具有 VARCHAR 或 CHAR 数据类型的列以及大小为 4 MB 或以下的行。

数据加载操作

通过指定以下参数来管理加载操作的默认行为，以进行故障排除或缩短加载时间。

- COMPROWS
- COMPUPDATE
- IGNOREALLERRORS

- MAXERROR
- NOLOAD
- STATUPDATE

参数

COMPROWS numrows

指定要用作压缩分析的样本大小的行数。将对来自每个数据切片的行运行分析。例如，如果指定 COMPROWS 1000000 (1000000) 且系统总共包含 4 个切片，则将为每个切片读取和分析的行数不超过 250000。

如果未指定 COMPROWS，则每个切片的样本大小默认为 100000。小于每个切片 100000 行这一默认值的 COMPROWS 值将自动升级到此默认值。但是，如果加载的数据量不足以生成有意义的样本，则不会执行自动压缩。

如果 COMPROWS 数量大于输入文件中的行数，则 COPY 命令仍将继续并对所有可用行运行压缩分析。此参数接受的范围介于 1000 到 2147483647 (2,147,483,647) 之间。

COMPUPDATE [PRESET | { ON | TRUE } | { OFF | FALSE }],

控制是否在 COPY 期间自动应用压缩编码。

如果 COMPUPDATE 为 PRESET，则在目标表为空时，COPY 命令会为每个列选择压缩编码，即使列已经有除 RAW 之外的编码也不例外。可以替换当前指定的列编码。每个列的编码基于列数据类型。不会对数据进行采样。Amazon Redshift 自动分配压缩编码，如下所示：

- 为定义为排序键的列分配 RAW 压缩。
- 为定义为 BOOLEAN、REAL 或 DOUBLE PRECISION 数据类型的列分配 RAW 压缩。
- 定义为 SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIMESTAMP 或 TIMESTAMPTZ 的列分配了 AZ64 压缩。
- 定义为 CHAR 或 VARCHAR 的列分配了 LZO 压缩。

如果省略了 COMPUPDATE，只有在目标表为空并且您没有为任何列指定编码（而非 RAW）时，COPY 命令才会为每个列选择压缩编码。每个列的编码是由 Amazon Redshift 确定的。不会对数据进行采样。

如果 COMPUPDATE 为 ON（或 TRUE）或者指定 COMPUPDATE 而没有提供选项，在表为空时，COPY 命令将应用自动压缩，即使表列已具有除 RAW 以外的编码。可以替换当前指定的列编码。每个列的编码基于样本数据分析。有关更多信息，请参阅 使用自动压缩加载表。

在 COMPUPDATE 为 OFF（或 FALSE）时，将禁用自动压缩。不会更改列编码。

有关分析压缩的系统表的信息，请参阅 STL_ANALYZE_COMPRESSION。

IGNOREALLERRORS

您可以指定此选项来忽略加载操作期间出现的所有错误。

如果您指定了 MAXERROR 选项，则无法指定 IGNOREALLERRORS 选项。不能为列式格式（包括 ORC 和 Parquet）指定 IGNOREALLERRORS 选项。

MAXERROR [AS] error_count

如果加载操作返回 error_count 数量的错误或更多错误，加载将失败。如果加载操作返回较少的错误，则将继续并返回指示无法加载的行数的 INFO 消息。使用此参数可允许加载操作在某些行因为格式设置错误或数据中的其他不一致性而未能加载到表中时继续。

如果您希望加载操作在出现第一个错误时就失败，请将此值设置为 0 或 1。AS 关键字是可选的。MAXERROR 默认值为 0，限制值为 100000。

由于 Amazon Redshift 的并行处理特性，报告的实际错误数量可能高出指定的 MAXERROR。如果 Amazon Redshift 集群中的任何节点检测到已超出 MAXERROR，则每个节点将报告它遇到的所有错误。

NOLOAD

检查数据文件的有效性，而不用实际加载数据。通过使用 NOLOAD 参数，可以在运行实际数据加载之前，确保数据文件加载而不产生任何错误。将 COPY 与 NOLOAD 参数结合运行将比加载数据要快很多，因为前者仅分析文件。

STATUPDATE [{ ON | TRUE } | { OFF | FALSE }]

在成功的 COPY 命令结束时，控制对优化器统计数据的自动计算和刷新。默认情况下，如果未使用 STATUPDATE 参数，则将在表最初为空时自动更新统计数据。

将数据插入非空表中将明显更改表的大小，我们建议通过运行 ANALYZE 命令或使用 STATUPDATE ON 参数来更新统计数据。

使用 STATUPDATE ON（或 TRUE），不管表最初是否为空，都将自动更新统计数据。如果使用 STATUPDATE，则当前用户必须是表所有者或超级用户。如果未指定 STATUPDATE，则仅需要 INSERT 权限。

通过使用 STATUPDATE OFF（或 FALSE），将从不更新统计数据。

有关更多信息，请参阅分析表。

按字母顺序排列的参数列表

以下列表提供指向每个 COPY 命令参数（按字母顺序排列）的描述的链接。

- ACCEPTANYDATE
- ACCEPTINVCHARS
- ACCESS_KEY_ID and SECRET_ACCESS_KEY
- AVRO
- BLANKSASNULL
- BZIP2
- COMPROWS
- COMPUPDATE
- CREDENTIALS
- CSV
- DATEFORMAT
- DELIMITER
- EMPTYASNULL
- ENCODING
- ENCRYPTED
- ESCAPE
- EXPLICIT_IDS
- FILLRECORD
- FIXEDWIDTH
- FORMAT
- FROM
- GZIP
- IAM_ROLE
- IGNOREALLERRORS
- IGNOREBLANKLINES
- IGNOREHEADER

- JSON
- LZOP
- MANIFEST
- MASTER_SYMMETRIC_KEY
- MAXERROR
- NOLOAD
- NULL AS
- READRATIO
- REGION
- REMOVEQUOTES
- ROUNDEC
- SESSION_TOKEN
- SHAPEFILE
- SSH
- STATUPDATE
- TIMEFORMAT
- SESSION_TOKEN
- TRIMBLANKS
- TRUNCATECOLUMNS
- ZSTD

使用说明

主题

- 访问其他 AWS 资源的权限
- 将 COPY 与 Amazon S3 接入点别名一起使用
- 从 Amazon S3 中加载多字节数据
- 加载 GEOMETRY 或 GEOGRAPHY 数据类型的列
- 加载 HLLSKETCH 数据类型
- 加载 VARBYTE 数据类型的列
- 在读取多个文件时出现错误

- 从 JSON 格式数据执行的 COPY 操作
- 从列式数据格式中执行 COPY 操作
- DATEFORMAT 和 TIMEFORMAT 字符串
- 在 DATEFORMAT 和 TIMEFORMAT 中使用自动识别

访问其他 AWS 资源的权限

要在您的集群和其他 AWS 资源（如 Amazon S3 、Amazon DynamoDB、Amazon EMR 或 Amazon EC2）之间移动数据，您的集群必须具有访问相应资源和执行所需操作的权限。例如，要从 Amazon S3 加载数据，COPY 必须具有对桶的 LIST 访问权限以及对桶对象的 GET 访问权限。有关最低权限的更多信息，请参阅 COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限。

要获取访问资源的授权，您的集群必须经过身份验证。您可以选择以下身份验证方法之一：

- 基于角色的访问控制 – 对于基于角色的访问控制，您指定您的集群用于身份验证和授权的 AWS Identity and Access Management (IAM) 角色。为了保护您的 AWS 凭证和敏感数据，我们强烈建议使用基于角色的身份验证。
- 基于密钥的访问控制 – 对于基于密钥的访问控制，您以纯文本形式为用户提供 AWS 访问凭证（访问密钥 ID 和秘密访问密钥）。

基于角色的访问控制

利用基于角色的访问控制，您的集群将代表您临时代入 IAM 角色。然后，基于对角色的授权，您的集群可访问所需的 AWS 资源。

创建 IAM 角色类似于向用户授予权限，因为它是一个 AWS 身份，具有确定该身份在 AWS 中可执行和不可执行的操作的权限策略。但是，任何实体都可以根据需要代入某个角色，角色并不是唯一地与某个用户关联。此外，角色没有任何关联的凭证（密码或访问密钥）。相反，如果将角色与集群关联，则会动态创建访问密钥并将其提供给集群。

我们建议使用基于角色的访问控制，因为除了保护您的 AWS 凭证之外，它还将提供对 AWS 资源和敏感用户数据的更安全、精细的访问控制。

基于角色的身份验证具有以下优点：

- 您可以使用 AWS 标准 IAM 工具定义 IAM 角色并将该角色与多个集群关联。当您修改某个角色的访问策略时，更改将自动应用于使用该角色的所有集群。
- 您可定义为特定集群和数据库用户授予对特定 AWS 资源和操作的访问权限的精细 IAM 策略。

- 您的集群将在运行时获取临时会话凭证并按需刷新凭证直到操作完成。如果您使用了基于密钥的临时凭证，并且临时凭证在操作完成前到期，操作将失败。
- 您的访问密钥 ID 和秘密访问密钥 ID 不会在 SQL 代码中存储或传输。

要使用基于角色的访问控制，您必须先使用 Amazon Redshift 服务角色类型创建 IAM 角色，然后将此角色附加到您的集群。此角色至少必须具有 COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限中列出的权限。有关创建 IAM 角色并将其附加到集群的步骤，请参阅《Amazon Redshift 管理指南》中的授权 Amazon Redshift 代表您访问其他 AWS 服务。

通过使用 Amazon Redshift 管理控制台、CLI 或 API，您可将角色添加到集群或查看与集群关联的角色。有关更多信息，请参阅《Amazon Redshift 管理指南》中的将 IAM 角色与集群关联。

当您创建 IAM 角色时，IAM 将返回该角色的 Amazon 资源名称 (ARN)。要指定 IAM 角色，请利用 IAM_ROLE 参数或 CREDENTIALS 参数提供角色 ARN。

例如，假设以下角色已附加到集群。

```
"IamRoleArn": "arn:aws:iam::0123456789012:role/MyRedshiftRole"
```

以下 COPY 命令示例使用 IAM_ROLE 参数，其 ARN 在上一示例中用于身份验证和访问 Amazon S3。

```
copy customer from 's3://mybucket/mydata'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

以下 COPY 命令示例使用 CREDENTIALS 参数指定 IAM 角色。

```
copy customer from 's3://mybucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

此外，超级用户还可以向数据库用户和组授予 ASSUMEROLE 权限，以便为 COPY 操作提供对角色的访问权限。有关信息，请参阅 GRANT。

基于密钥的访问控制

利用基于密钥的访问控制，您将为获权访问包含数据的 AWS 资源的 IAM 用户提供访问密钥 ID 和秘密访问密钥。您可以配合使用 ACCESS_KEY_ID and SECRET_ACCESS_KEY 参数或使用 CREDENTIALS 参数。

> **Note**
>
> 我们强烈建议使用 IAM 角色进行身份验证而不是提供纯文本访问密钥 ID 和秘密访问密钥。如果您选择基于密钥的访问控制，则不要使用 AWS 账户（根）凭证。应始终创建 IAM 用户并提供该用户的访问密钥 ID 和秘密访问密钥。有关创建 IAM 用户的步骤，请参阅在您的 AWS 账户中创建 IAM 用户。

要使用 ACCESS_KEY_ID 和 SECRET_ACCESS_KEY 进行身份验证，请使用授权用户的访问密钥 ID 和完整的秘密访问密钥替换 *<access-key-id>* 和 *<secret-access-key>*，如下所示。

```
ACCESS_KEY_ID '<access-key-id>'
SECRET_ACCESS_KEY '<secret-access-key>';
```

要使用 CREDENTIALS 参数进行身份验证，请使用授权用户的访问密钥 ID 和完整的秘密访问密钥替换 *<access-key-id>* 和 *<secret-access-key>*，如下所示。

```
CREDENTIALS
'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>';
```

IAM 用户必须至少具有 COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限 中列出的权限。

临时安全凭证

如果您使用基于密钥的访问控制，则可通过使用临时安全凭证进一步限制用户对您的数据具有的访问权限。基于角色的身份验证将自动使用临时凭证。

> **Note**
>
> 我们强烈建议您使用 role-based access control，而不要创建临时凭证并提供纯文本形式的访问密钥 ID 和秘密访问密钥。基于角色的访问控制将自动使用临时凭证。

临时安全证书可增强安全性，因为它们时效短，过期后无法重复使用。使用令牌生成的访问密钥 ID 和秘密访问密钥无法脱离令牌使用，具有这些临时安全凭证的用户仅可以在凭证未过期前访问您的资源。

要为用户授予对您的资源的临时访问权限，请调用 AWS Security Token Service (AWS STS) API 操作。AWS STS API 操作将返回临时安全凭证，其中包括一个安全令牌、一个访问密钥 ID 和一个秘密访问密钥。您为需要临时访问您的资源的用户颁发临时安全凭证。这些用户可以是现有的 IAM 用户，

也可以是非 AWS 用户。有关创建临时安全凭证的更多信息，请参阅《IAM 用户指南》中的使用临时安全凭证。

您可以将 ACCESS_KEY_ID and SECRET_ACCESS_KEY 参数与 SESSION_TOKEN 参数或 CREDENTIALS 参数配合使用。您还必须提供随令牌一起提供的访问密钥 ID 和秘密访问密钥。

要使用 ACCESS_KEY_ID、SECRET_ACCESS_KEY 和 SESSION_TOKEN 进行身份验证，请根据如下所示替换 *<temporary-access-key-id>*、*<temporary-secret-access-key>* 和 *<temporary-token>*。

```
ACCESS_KEY_ID '<temporary-access-key-id>'
SECRET_ACCESS_KEY '<temporary-secret-access-key>'
SESSION_TOKEN '<temporary-token>';
```

要使用 CREDENTIALS 进行身份验证，请在凭证字符串中包括 session_token=*<temporary-token>*，如下所示。

```
CREDENTIALS
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;session_token=<temporary-token>';
```

以下示例为具有临时安全凭证的 COPY 命令。

```
copy table-name
from 's3://objectpath'
access_key_id '<temporary-access-key-id>'
secret_access_key '<temporary-secret-access-key>'
session_token '<temporary-token>';
```

以下示例使用临时凭证和文件加密加载 LISTING 表。

```
copy listing
from 's3://mybucket/data/listings_pipe.txt'
access_key_id '<temporary-access-key-id>'
secret_access_key '<temporary-secret-access-key>'
session_token '<temporary-token>'
master_symmetric_key '<root-key>'
encrypted;
```

以下示例将 CREDENTIALS 参数与临时凭证和文件加密配合使用，加载 LISTING 表。

```
copy listing
from 's3://mybucket/data/listings_pipe.txt'
credentials
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;session_token=<temporary-token>;master_symmetric_key=<root-key>'
encrypted;
```

> **Important**
>
> 临时安全凭证必须在整个 COPY 或 UNLOAD 操作持续时间有效。如果临时安全凭证在操作过程中过期，相应命令将失败，事务将被回滚。例如，如果临时安全凭证在 15 分钟后过期而 COPY 操作需要一个小时，则 COPY 操作将失败，无法完成。如果您使用基于角色的访问，则会自动刷新临时安全凭证直到操作完成。

COPY、UNLOAD 和 CREATE LIBRARY 的 IAM 权限

CREDENTIALS 参数引用的 IAM 角色或用户必须至少具有以下权限：

- 对于从 Amazon S3 执行的 COPY 的操作，这是指对 Amazon S3 桶执行 LIST 以及对正在加载的 Amazon S3 对象以及清单文件（如果已使用）执行 GET 的权限。
- 对于从 Amazon S3、Amazon EMR 执行 COPY 的操作、以及从使用 JSON 格式数据的远程主机 (SSH) 执行 COPY 的操作，这是指对 Amazon S3 上的 JSONPaths 文件（如果已使用）执行 LIST 和 GET 的权限。
- 对于从 DynamoDB 执行 COPY 的操作，这是指对所加载的 DynamoDB 表执行 SCAN 和 DESCRIBE 的权限。
- 对于从 Amazon EMR 集群执行的 COPY 操作，这是指对 Amazon EMR 集群上的 `ListInstances` 操作的权限。
- 对于向 Amazon S3 执行 UNLOAD 的操作，这是指正在将数据文件卸载到的 Amazon S3 桶的 GET、LIST 和 PUT 权限。
- 对于从 Amazon S3 执行 CREATE LIBRARY 的操作，这是指对 Amazon S3 桶执行 LIST 以及对正在导入的 Amazon S3 对象执行 GET 的权限。

> **Note**
>
> 如果您在运行 COPY、UNLOAD 或 CREATE LIBRARY 命令时收到错误消息 `S3ServiceException: Access Denied`，则您的集群对于 Amazon S3 没有适当的访问权限。

您可以通过向附加到集群的 IAM 角色、向用户或向用户所属的组附加 IAM 策略，来管理 IAM 权限。例如，`AmazonS3ReadOnlyAccess` 托管策略可授予对 Amazon S3 资源的 LIST 和 GET 权限。有关 IAM 策略的更多信息，请参阅《IAM 用户指南》中的管理 IAM 策略。

将 COPY 与 Amazon S3 接入点别名一起使用

COPY 支持 Amazon S3 接入点别名。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的为您的接入点使用桶式别名。

从 Amazon S3 中加载多字节数据

如果您的数据包含非 ASCII 多字节字符（例如中文或西里尔语字符），则必须将该数据加载到 VARCHAR 列。VARCHAR 数据类型支持四字节的 UTF-8 字符，而 CHAR 数据类型仅接受单字节的 ASCII 字符。您不能将五字节或更长的字符加载到 Amazon Redshift 表中。有关更多信息，请参阅 多字节字符。

加载 GEOMETRY 或 GEOGRAPHY 数据类型的列

您可以从字符分隔文本文件（如 CSV 文件）中的数据执行对 `GEOMETRY` 或 `GEOGRAPHY` 列的 COPY 操作。数据必须采用已知二进制（WKB 或 EWKB）格式或已知文本（WKT 或 EWKT）格式的十六进制格式，并且符合 COPY 命令的单个输入行的最大大小范围要求。有关更多信息，请参阅 COPY。

有关如何从 shapefile 加载的信息，请参阅将 shapefile 加载到 Amazon Redshift。

有关 `GEOMETRY` 或 `GEOGRAPHY` 数据类型的更多信息，请参阅在 Amazon Redshift 中查询空间数据。

加载 HLLSKETCH 数据类型

您只能以 Amazon Redshift 支持的稀疏或密集格式复制 HLL 草图。要在 HyperLogLog 草图上使用 COPY 命令，请对密集 HyperLogLog 草图使用 Base64 格式，对稀疏 HyperLogLog 草图使用 JSON 格式。有关更多信息，请参阅 HyperLogLog 函数。

以下示例使用 CREATE TABLE 和 COPY 将 CSV 文件中的数据导入到表中。首先，该示例使用 CREATE TABLE 创建表 `t1`。

```
CREATE TABLE t1 (sketch hllsketch, a bigint);
```

然后，它使用 COPY 将 CSV 文件中的数据导入到表 t1 中。

```
COPY t1 FROM s3://DOC-EXAMPLE-BUCKET/unload/' IAM_ROLE
 'arn:aws:iam::0123456789012:role/MyRedshiftRole' NULL AS 'null' CSV;
```

加载 VARBYTE 数据类型的列

您可以从 CSV、Parquet 和 ORC 格式的文件加载数据。对于 CSV，从以十六进制表示 VARBYTE 数据的文件中加载数据。你无法使用 FIXEDWIDTH 选项加载 VARBYTE 数据。不支持 COPY 的 ADDQUOTES 或 REMOVEQUOTES 选项。不可将 VARBYTE 列用作分区列。

在读取多个文件时出现错误

COPY 命令是原子和事务性的。换言之，甚至在 COPY 命令读取多个文件中的数据时，整个过程也将视为单个事务。如果 COPY 在读取某个文件时遇到错误，则将自动重试直至此过程超时（请参阅statement_timeout），或者如果在较长时间（15 到 30 分钟）内无法从 Amazon S3 下载数据，则将确保一次仅下载一个文件。如果 COPY 命令失败，则将取消整个事务并回滚所有更改。有关处理加载错误的更多信息，请参阅解决数据加载问题。

在成功启动 COPY 命令后，此命令不会在会话终止（例如客户端断开）时失败。但是，如果 COPY 命令位于因会话终止而未完成的 BEGIN … END 事务数据块中，则整个事务（包括 COPY）都将回滚。有关事务的更多信息，请参阅 BEGIN。

从 JSON 格式数据执行的 COPY 操作

JSON 数据结构由一组对象 或数组 组成。JSON 对象 以大括号开头和结尾，并包含名称-值对的无序集合。每个名称和值由冒号分隔，而每个名称/值对由逗号分隔。名称是用双引号括起的字符串。引号字符必须是简单引号 (0x22)，而不能是倾斜引号或“智能”引号。

JSON 数组 以中括号开头和结尾，并包含由逗号分隔的值的有序集合。值可以是用双引号括起的字符串、数字、布尔值 true 或 false、null、JSON 对象或数组。

JSON 对象和数组可以嵌套，从而实现分层的数据结构。以下示例显示了包含两个有效对象的 JSON 数据结构。

```
{
    "id": 1006410,
    "title": "Amazon Redshift Database Developer Guide"
```

```
}
{
    "id": 100540,
    "name": "Amazon Simple Storage Service User Guide"
}
```

下面显示了与两个 JSON 数组相同的数据。

```
[
    1006410,
    "Amazon Redshift Database Developer Guide"
]
[
    100540,
    "Amazon Simple Storage Service User Guide"
]
```

JSON 的 COPY 选项

将 COPY 与 JSON 格式数据结合使用时，可以指定以下选项：

- `'auto'` – COPY 自动从 JSON 文件加载字段。
- `'auto ignorecase'` – COPY 自动从 JSON 文件加载字段，同时忽略字段名称的大小写。
- `s3://jsonpaths_file` – COPY 使用 JSONPaths 文件解析 JSON 源数据。JSONPaths 文件是一个包含单个 JSON 对象的文本文件，其中的对象名称 `"jsonpaths"` 与 JSONPath 表达式数组配对。如果该名称是 `"jsonpaths"` 之外的任何字符串，则 COPY 将使用 `'auto'` 参数而不是使用 JSONPaths 文件。

有关说明如何使用 `'auto'`、`'auto ignorecase'` 或 JSONPaths 文件以及使用 JSON 对象或数组加载数据的示例，请参阅从 JSON 中复制的示例。

JSONPath 选项

在 Amazon Redshift COPY 语法中，JSONPath 表达式使用括号表示法或点表示法指定 JSON 层次数据结构中单个名称元素的显式路径。Amazon Redshift 不支持可能解析为不确定路径或多个名称元素的任何 JSONPath 元素（如通配符或筛选表达式）。因此，Amazon Redshift 无法解析复杂、多级的数据结构。

下面是包含使用括号表示法的 JSONPath 表达式的 JSONPaths 文件的示例。美元符号 ($) 表示根级别结构。

```
{
    "jsonpaths": [
       "$['id']",
       "$['store']['book']['title']",
 "$['location'][0]"
    ]
}
```

在上面的示例中，$['location'][0] 引用数组中的第一个元素。JSON 使用从 0 开始的数组索引。数组索引必须是正整数（大于或等于零）。

以下示例显示了使用点表示法的前一个 JSONPaths 文件。

```
{
    "jsonpaths": [
       "$.id",
       "$.store.book.title",
 "$.location[0]"
    ]
}
```

您不能在 jsonpaths 数组中将括号表示法和点表示法混合。括号表示法和点表示法中均可使用括号来引用数组元素。

使用点表示法时，JSONPath 表达式不能包含下列字符：

- 单直引号 (')
- 句点或点 (.)
- 中括号 ([]) （除非用于引用数组元素）

如果 JSONPath 表达式引用的名称-值对中的值是对象或数组，则整个对象或数组将作为字符串加载，包括大括号或中括号。例如，假定 JSON 数据包含以下对象。

```
{
    "id": 0,
    "guid": "84512477-fa49-456b-b407-581d0d851c3c",
    "isActive": true,
    "tags": [
        "nisi",
```

```
        "culpa",
        "ad",
        "amet",
        "voluptate",
        "reprehenderit",
        "veniam"
    ],
    "friends": [
        {
            "id": 0,
            "name": "Martha Rivera"
        },
        {
            "id": 1,
            "name": "Renaldo"
        }
    ]
}
```

JSONPath 表达式 `$['tags']` 之后将返回以下值。

```
"["nisi","culpa","ad","amet","voluptate","reprehenderit","veniam"]"
```

JSONPath 表达式 `$['friends'][1]` 之后将返回以下值。

```
"{"id": 1,"name": "Renaldo"}"
```

`jsonpaths` 数组中的每个 JSONPath 表达式对应于 Amazon Redshift 目标表中的一个列。`jsonpaths` 数组元素的顺序必须与目标表或列列表（如果使用了列列表）中列的顺序一致。

有关说明如何使用 `'auto'` 参数或 JSONPaths 文件以及使用 JSON 对象或数组加载数据的示例，请参阅从 JSON 中复制的示例。

有关如何复制多个 JSON 文件的信息，请参阅使用清单指定数据文件。

在 JSON 中转义字符

COPY 会将 `\n` 作为换行符加载并且会将 `\t` 作为制表符加载。要加载反斜杠，请使用反斜杠（`\\`）对其进行转义。

例如，假设您在桶 `escape.json` 中名为 `s3://mybucket/json/` 的文件中具有以下 JSON。

```
{
  "backslash": "This is a backslash: \\",
  "newline": "This sentence\n is on two lines.",
  "tab": "This sentence \t contains a tab."
}
```

运行下列命令以创建 ESCAPES 表并加载 JSON。

```
create table escapes (backslash varchar(25), newline varchar(35), tab varchar(35));

copy escapes from 's3://mybucket/json/escape.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as json 'auto';
```

查询 ESCAPES 表以查看结果。

```
select * from escapes;

       backslash        |      newline      |               tab
------------------------+-------------------+----------------------------------
 This is a backslash: \ | This sentence     | This sentence    contains a tab.
                        :  is on two lines.
(1 row)
```

数值精度丢失

当您将数字从 JSON 格式的数据文件加载到定义为数字数据类型的列时，您可能会丢失精度。某些浮点值在计算机系统中无法准确表示。因此，您从 JSON 文件复制的数据可能无法按预期进行舍入。为避免精度丢失，我们建议使用以下替代方法之一：

- 通过用双引号字符将值括起来将数字表示为字符串。
- 使用 ROUNDEC 对数字进行舍入而不是截断。
- 不要使用 JSON 或 Avro 文件，而应使用 CSV、字符分隔或固定宽度的文本文件。

从列式数据格式中执行 COPY 操作

COPY 可采用以下列式格式从 Amazon S3 中加载数据：

- ORC

- Parquet

有关从列式数据格式中使用 COPY 的示例，请参阅COPY 示例。

COPY 支持列式数据，但要注意以下几点：

- Amazon S3 桶必须与 Amazon Redshift 数据库位于同一 AWS 区域。
- 要通过 VPC 端点访问您的 Amazon S3 数据，请使用 IAM 策略和 IAM 角色设置访问权限，如《Amazon Redshift 管理指南》中的将 Amazon Redshift Spectrum 与增强 VPC 路由结合使用中所述。
- COPY 不自动应用压缩编码。
- 仅支持以下 COPY 参数：
  - ACCEPTINVCHARS（从 ORC 或 Parquet 文件中复制时）。
  - FILLRECORD
  - FROM
  - IAM_ROLE
  - CREDENTIALS
  - STATUPDATE
  - MANIFEST
  - EXPLICIT_IDS
- 如果 COPY 在加载时遇到错误，则命令失败。列式数据类型不支持 ACCEPTANYDATE 和 MAXERROR。
- 错误消息发送至 SQL 客户端。一些错误记录在 STL_LOAD_ERRORS 和 STL_ERROR 中。
- COPY 会按列在列式数据文件中出现的相同顺序将值插入到目标表的列中。目标表中的列数和数据文件中的列数必须匹配。
- 如果您为 COPY 操作指定的文件包含下列扩展名之一，我们将解压缩数据，而无需添加任何参数：
  - `.gz`
  - `.snappy`
  - `.bz2`
- 从 Parquet 和 ORC 文件格式的 COPY 操作使用 Redshift Spectrum 和桶访问。要对这些格式执行 COPY 操作，请确保没有任何阻止使用 Amazon S3 预签名 URL 的 IAM 策略。Amazon Redshift 生成的预签名 URL 有效期为 1 小时，这样 Amazon Redshift 就有足够的时间从 Amazon S3 存储桶中加载所有文件。COPY 操作从列式数据格式中扫描的每个文件都会生成一个唯一的预签名 URL。对

于包含 s3:signatureAge 操作的存储桶策略，请确保将该值至少设置为 3,600,000 毫秒。有关更多信息，请参阅将 Amazon Redshift Spectrum 与增强型 VPC 路由结合使用。

DATEFORMAT 和 TIMEFORMAT 字符串

COPY 命令使用 DATEFORMAT 和 TIMEFORMAT 选项来解析源数据中的日期和时间值。DATEFORMAT 和 TIMEFORMAT 是格式化字符串，必须与源数据的日期和时间值的格式相匹配。例如，加载具有日期值 Jan-01-1999 的源数据的 COPY 命令必须包括以下 DATEFORMAT 字符串：

```
COPY ...
            DATEFORMAT AS 'MON-DD-YYYY'
```

有关管理 COPY 数据转换的更多信息，请参阅数据转换参数。

DATEFORMAT 和 TIMEFORMAT 字符串可包含日期时间分隔符（例如'-'、'/'或':'）以及下表中的日期部分和时间部分格式。

> **Note**
>
> 如果您无法将日期或时间值的格式与以下日期部分和时间部分相匹配，或如果您的日期和时间值使用的格式彼此不同，则请将 'auto' 参数与 DATEFORMAT 或 TIMEFORMAT 参数结合使用。在使用 DATEFORMAT 或 TIMEFORMAT 字符串时，'auto' 参数会识别几种不受支持的格式。有关更多信息，请参阅 在 DATEFORMAT 和 TIMEFORMAT 中使用自动识别。

日期部分或时间部分	意义
YY	不带世纪的年份
YYYY	带世纪的年份
MM	用数字表示的月份
MON	用名称表示的月份（缩写名称或完整名称）
DD	用数字表示的日
HH 或 HH24	小时（24 小时制）

日期部分或时间部分	意义
	Note 在用于 SQL 函数的 DATETIME 格式字符串中，HH 与 HH12 相同。但是，在用于 COPY 的 DATEFORMAT 和 TIMEFORMAT 字符串中，HH 与 HH24 相同。
HH12	小时（12 小时制）
MI	分钟
SS	秒
AM 或 PM	经线标识符（适用于 12 小时制）

默认格式日期是 YYYY-MM-DD。不带时区 (TIMESTAMP) 的默认时间戳格式是 YYYY-MM-DD HH:MI:SS。带时区 (TIMESTAMPTZ) 的默认时间戳格式是 YYYY-MM-DD HH:MI:SSOF，其中 OF 是与 UTC 的偏移两个（例如，-8:00）。您不能在 timeformat_string 中包含时区标识符（TZ、tz 或 OF）。秒 (SS) 字段还支持小数秒到微秒级别的细节。要加载格式与默认格式不同的 TIMESTAMPTZ 数据，请指定“自动”。

以下是您在源数据中会遇到的一些示例日期或时间，以及相应的 DATEFORMAT 或 TIMEFORMAT 字符串。

源数据日期或时间的示例	DATEFORMAT 或 TIMEFORMAT 语法
03/31/2003	DATEFORMAT AS 'MM/DD/YYYY'
March 31, 2003	DATEFORMAT AS 'MON DD, YYYY'
03.31.2003 18:45:05	TIMEFORMAT AS 'MM.DD.YYYY HH:MI:SS'

源数据日期或时间的示例	DATEFORMAT 或 TIMEFORMAT 语法
03.31.2003 18:45:05.123456	

示例

有关使用 TIMEFORMAT 的示例，请参阅加载时间戳或日期戳。

在 DATEFORMAT 和 TIMEFORMAT 中使用自动识别

如果您指定 'auto' 作为 DATEFORMAT 或 TIMEFORMAT 参数的参数，Amazon Redshift 将自动识别并转换源数据中的日期格式或时间格式。下面是一个示例。

```
copy favoritemovies from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
dateformat 'auto';
```

在与 DATEFORMAT 和 TIMEFORMAT 的 'auto' 参数一起使用时，COPY 将识别并转换在 DATEFORMAT 和 TIMEFORMAT 字符串 中的表中列出的日期和时间格式。此外，'auto' 参数将识别下列在使用 DATEFORMAT 和 TIMEFORMAT 字符串时不受支持的格式。

格式	有效输入字符串的示例
ISO 8601	2019-02-11T05:09:12.195Z
Julian	J2451187
BC	Jan-08-95 BC
YYYYMMDD HHMISS	19960108 040809
YYMMDD HHMISS	960108 040809
YYYY.DDD	1996.008
YYYY-MM-DD HH:MI:SS.SSS	1996-01-08 04:05:06.789

格式	有效输入字符串的示例
DD Mon HH:MI:SS YYYY TZ	17 Dec 07:37:16 1997 PST
MM/DD/YYYY HH:MI:SS.SS TZ	12/17/1997 07:37:16.00 PST
YYYY-MM-DD HH:MI:SS+/-TZ	1997-12-17 07:37:16-08
DD.MM.YYYY HH:MI:SS TZ	12.17.1997 07:37:16.00 PST

自动识别不支持 epochsec 和 epochmillisec。

要测试是否将自动转换日期或时间戳值，请使用 CAST 函数尝试将字符串转换为日期或时间戳值。例如，下列命令测试时间戳值 'J2345678 04:05:06.789'：

```
create table formattest (test char(21));
insert into formattest values('J2345678 04:05:06.789');
select test, cast(test as timestamp) as timestamp, cast(test as date) as date from
 formattest;

        test          |      timestamp      | date
----------------------+---------------------+------------
J2345678 04:05:06.789   1710-02-23 04:05:06 1710-02-23
```

如果 DATE 列的源数据包含时间信息，则将截断时间部分。如果 TIMESTAMP 列的源数据省略时间信息，则将使用 00:00:00 作为时间部分。

COPY 示例

> **Note**
>
> 为便于阅读，这些示例包含换行符。请不要在您的 credentials-args 字符串中包含换行符或空格。

主题

从 DynamoDB 表中加载 FAVORITEMOVIES

AWS 开发工具包包括一个创建名为 Movies 的 DynamoDB 表的简单示例。（有关此示例，请参阅 DynamoDB 入门。）以下示例加载包含 DynamoDB 表中数据的 Amazon Redshift MOVIES 表。Amazon Redshift 表必须已存在于数据库中。

```
copy favoritemovies from 'dynamodb://Movies'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

从 Amazon S3 桶中加载 LISTING

以下示例从 Amazon S3 桶加载 LISTING。COPY 命令将加载 /data/listing/ 文件夹中的所有文件。

```
copy listing
from 's3://mybucket/data/listing/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

从 Amazon EMR 集群中加载 LISTING

以下示例从 Amazon EMR 集群的 lzop 压缩文件加载使用制表符分隔数据的 SALES 表。COPY 加载 myoutput/ 文件夹中每个以 part- 开头的文件。

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/part-*'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '\t' lzop;
```

以下示例将加载包含 Amazon EMR 集群中的 JSON 格式的数据的 SALES 表。COPY 加载 myoutput/json/ 文件夹中的每个文件。

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/json/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
JSON 's3://mybucket/jsonpaths.txt';
```

使用清单指定数据文件

您可以使用清单确保 COPY 命令将从 Amazon S3 加载所有必需的文件，而且仅加载必需的文件。当您需要从不同的桶加载多个文件或加载未共享相同前缀的文件时，您也可使用清单。

例如，假设您需要加载下列三个文件：custdata1.txt、custdata2.txt 和 custdata3.txt。您可使用以下命令通过指定前缀来加载 mybucket 中以 custdata 开头的所有文件：

```
copy category
from 's3://mybucket/custdata'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

如果由于错误仅存在两个文件，则 COPY 仅加载这两个文件并成功完成，从而导致不完整的数据加载。如果桶还包含恰巧使用相同前缀的不需要的文件（例如名为 custdata.backup 的文件），则 COPY 还加载此文件，从而导致加载不需要的数据。

为了确保加载所有必需的文件并防止加载不需要的文件，您可使用清单文件。清单是 JSON 格式的文本文件，其中列出了要通过 COPY 命令处理的文件。例如，以下清单将加载上例中的三个文件。

```
{
   "entries":[
      {
         "url":"s3://mybucket/custdata.1",
         "mandatory":true
      },
      {
         "url":"s3://mybucket/custdata.2",
         "mandatory":true
      },
      {
         "url":"s3://mybucket/custdata.3",
         "mandatory":true
      }
   ]
}
```

可选的 mandatory 标志指示 COPY 是否应在文件不存在时终止。默认为 false。如果未找到任何文件，则无论 mandatory 设置如何，COPY 都会终止。在此示例中，如果未找到任何文件，COPY 将返回错误。将忽略可能会在仅指定键前缀（如 custdata.backup）的情况下选取的不需要的文件，因为它们不在清单上。

在从采用 ORC 或 Parquet 格式的数据文件中加载时，需要 meta 字段，如以下示例所示。

```
{
   "entries":[
      {
         "url":"s3://mybucket-alpha/orc/2013-10-04-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      },
      {
         "url":"s3://mybucket-beta/orc/2013-10-05-custdata",
```

```
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      }
   ]
}
```

以下示例使用名为 cust.manifest 的清单。

```
copy customer
from 's3://mybucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as orc
manifest;
```

您可以使用清单来加载不同桶或文件中未共享相同前缀的文件。以下示例显示了用于加载名称以日期戳开头的文件中的数据的 JSON。

```
{
  "entries": [
    {"url":"s3://mybucket/2013-10-04-custdata.txt","mandatory":true},
    {"url":"s3://mybucket/2013-10-05-custdata.txt","mandatory":true},
    {"url":"s3://mybucket/2013-10-06-custdata.txt","mandatory":true},
    {"url":"s3://mybucket/2013-10-07-custdata.txt","mandatory":true}
  ]
}
```

此清单可列出位于不同桶中的文件，前提是桶与集群位于同一 AWS 区域。

```
{
  "entries": [
    {"url":"s3://mybucket-alpha/custdata1.txt","mandatory":false},
    {"url":"s3://mybucket-beta/custdata1.txt","mandatory":false},
    {"url":"s3://mybucket-beta/custdata2.txt","mandatory":false}
  ]
}
```

从以竖线（默认分隔符）分隔的文件中加载 LISTING

以下示例是一个非常简单的示例，其中未指定任何选项并且输入文件包含默认分隔符，即竖线字符（“|”）。

```
copy listing
from 's3://mybucket/data/listings_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

使用 Parquet 格式的列式数据加载 LISTING

以下示例从 Amazon S3 上的名为 parquet 的文件夹加载数据。

```
copy listing
from 's3://mybucket/data/listings/parquet/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as parquet;
```

使用 ORC 格式的列式数据加载 LISTING

以下示例从 Amazon S3 上名为 `orc` 的文件夹加载数据。

```
copy listing
from 's3://mybucket/data/listings/orc/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as orc;
```

使用选项加载 EVENT

以下示例将竖线分隔的数据加载到 EVENT 表中并应用下列规则：

- 如果使用了引号对来括起任何字符串，则会删除它们。
- 空字符串和包含空白的字符串将作为 NULL 值加载。
- 如果返回了 5 个以上的错误，则加载失败。
- 时间戳值必须遵循指定的格式；例如，有效的时间戳为 `2008-09-26 05:43:12`。

```
copy event
from 's3://mybucket/data/allevents_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
removequotes
emptyasnull
blanksasnull
maxerror 5
delimiter '|'
```

```
timeformat 'YYYY-MM-DD HH:MI:SS';
```

从固定宽度的数据文件中加载 VENUE

```
copy venue
from 's3://mybucket/data/venue_fw.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth 'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6';
```

上例假设数据文件与所示的样本数据是使用相同的方式设置格式的。在下面的示例中，空格充当占位符，以便所有列的宽度与规范中的规定相同：

```
1  Toyota Park              Bridgeview  IL0
2  Columbus Crew Stadium    Columbus    OH0
3  RFK Stadium              Washington  DC0
4  CommunityAmerica BallparkKansas City KS0
5  Gillette Stadium         Foxborough  MA68756
```

从 CSV 文件中加载 CATEGORY

假设您要加载具有下表中所示值的 CATEGORY。

catid	catgroup	catname	catdesc
12	Shows	Musicals	Musical theatre
13	Shows	Plays	All "non-musical" theatre
14	Shows	Opera	All opera, light, and "rock" opera
15	Concerts	Classical	All symphony, concerto, and choir concerts

以下示例显示了字段值用逗号隔开的文本文件的内容。

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,All "non-musical" theatre
14,Shows,Opera,All opera, light, and "rock" opera
15,Concerts,Classical,All symphony, concerto, and choir concerts
```

如果您在加载文件时使用 DELIMITER 参数指定逗号分隔的输入，则 COPY 命令失败，因为一些输入字段包含逗号。您可通过使用 CSV 参数并将包含逗号的字段括在引号字符中来避免以上问题。如果用引号括起来的字符串中出现引号字符，则需要通过双引号字符来进行转义。默认引号字符为双引号，因此您需要使用一个额外的双引号对每个双引号进行转义。您的新输入文件与下面类似。

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,"All ""non-musical"" theatre"
14,Shows,Opera,"All opera, light, and ""rock"" opera"
15,Concerts,Classical,"All symphony, concerto, and choir concerts"
```

假定文件名为 `category_csv.txt`，则可通过使用以下 COPY 命令加载文件：

```
copy category
from 's3://mybucket/data/category_csv.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
csv;
```

或者，若要避免对输入中的双引号进行转义，可通过使用 QUOTE AS 参数来指定其他引号字符。例如，以下版本的 `category_csv.txt` 使用“%”作为引号字符。

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,%All "non-musical" theatre%
14,Shows,Opera,%All opera, light, and "rock" opera%
15,Concerts,Classical,%All symphony, concerto, and choir concerts%
```

以下 COPY 命令使用 QUOTE AS 来加载 `category_csv.txt`：

```
copy category
from 's3://mybucket/data/category_csv.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
csv quote as '%';
```

加载具有显式的 IDENTITY 列值的 VENUE

以下示例假设在创建 VENUE 表时，至少将一个列（如 `venueid` 列）指定为 IDENTITY 列。此命令将覆盖 IDENTITY 列的自动生成值的默认 IDENTITY 行为，并将改为从 venue.txt 文件加载显式值。使用 EXLICIT_IDS 选项时，Amazon Redshift 不会检查表中是否加载了重复的 IDENTITY 值。

```
copy venue
```

```
from 's3://mybucket/data/venue.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
explicit_ids;
```

从以竖线分隔的 GZIP 文件中加载 TIME

以下示例从用竖线分隔的 GZIP 文件加载 TIME 表：

```
copy time
from 's3://mybucket/data/timerows.gz'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
gzip
delimiter '|';
```

加载时间戳或日期戳

以下示例加载具有带格式的时间戳的数据。

> **Note**
>
> HH:MI:SS 的 TIMEFORMAT 还可支持超出 SS 的高达微秒细节级别的小数秒。此示例中使用的文件 time.txt 包含一行，即 2009-01-12 14:15:57.119568。

```
copy timestamp1
from 's3://mybucket/data/time.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
timeformat 'YYYY-MM-DD HH:MI:SS';
```

此复制的结果如下所示：

```
select * from timestamp1;
c1
----------------------------
2009-01-12 14:15:57.119568
(1 row)
```

从具有默认值的文件中加载数据

以下示例使用 TICKIT 数据库中的 VENUE 表的变体。考虑使用以下语句定义的 VENUE_NEW 表：

```
create table venue_new(
venueid smallint not null,
venuename varchar(100) not null,
venuecity varchar(30),
venuestate char(2),
venueseats integer not null default '1000');
```

考虑未包含任何 VENUESEATS 列值的 venue_noseats.txt 数据文件，如以下示例中所示：

```
1|Toyota Park|Bridgeview|IL|
2|Columbus Crew Stadium|Columbus|OH|
3|RFK Stadium|Washington|DC|
4|CommunityAmerica Ballpark|Kansas City|KS|
5|Gillette Stadium|Foxborough|MA|
6|New York Giants Stadium|East Rutherford|NJ|
7|BMO Field|Toronto|ON|
8|The Home Depot Center|Carson|CA|
9|Dick's Sporting Goods Park|Commerce City|CO|
10|Pizza Hut Park|Frisco|TX|
```

以下 COPY 语句将成功地从此文件中加载表并对已省略的列应用 DEFAULT 值（“1000”）：

```
copy venue_new(venueid, venuename, venuecity, venuestate)
from 's3://mybucket/data/venue_noseats.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

现在查看加载的表：

```
select * from venue_new order by venueid;
venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
1 | Toyota Park                | Bridgeview      | IL         |       1000
2 | Columbus Crew Stadium      | Columbus        | OH         |       1000
3 | RFK Stadium                | Washington      | DC         |       1000
4 | CommunityAmerica Ballpark  | Kansas City     | KS         |       1000
5 | Gillette Stadium           | Foxborough      | MA         |       1000
6 | New York Giants Stadium    | East Rutherford | NJ         |       1000
7 | BMO Field                  | Toronto         | ON         |       1000
8 | The Home Depot Center      | Carson          | CA         |       1000
9 | Dick's Sporting Goods Park | Commerce City   | CO         |       1000
```

```
10 | Pizza Hut Park             | Frisco          | TX         |       1000
(10 rows)
```

在以下示例中，除了假设此文件中未包含任何 VENUESEATS 数据之外，还假设未包含任何 VENUENAME 数据：

```
1||Bridgeview|IL|
2||Columbus|OH|
3||Washington|DC|
4||Kansas City|KS|
5||Foxborough|MA|
6||East Rutherford|NJ|
7||Toronto|ON|
8||Carson|CA|
9||Commerce City|CO|
10||Frisco|TX|
```

通过使用相同的表定义，以下 COPY 语句失败，因为未为 VENUENAME 指定任何 DEFAULT 值，并且 VENUENAME 是一个非 NULL 列：

```
copy venue(venueid, venuecity, venuestate)
from 's3://mybucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

现在考虑使用 IDENTITY 列的 VENUE 表的变体：

```
create table venue_identity(
venueid int identity(1,1),
venuename varchar(100) not null,
venuecity varchar(30),
venuestate char(2),
venueseats integer not null default '1000');
```

与上例一样，假设 VENUESEATS 列没有源文件中的对应值。以下 COPY 语句成功地加载表（包括预先定义的 IDENTITY 数据值）而不是自动生成这些值：

```
copy venue(venueid, venuename, venuecity, venuestate)
from 's3://mybucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
delimiter '|' explicit_ids;
```

此语句将失败，因为它未包含 IDENTITY 列（列列表中缺少 VENUEID），而是包含 EXPLICIT_IDS 参数：

```
copy venue(venuename, venuecity, venuestate)
from 's3://mybucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' explicit_ids;
```

此语句将失败，因为它不包含 EXPLICIT_IDS 参数：

```
copy venue(venueid, venuename, venuecity, venuestate)
from 's3://mybucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

使用 ESCAPE 选项复制数据

以下示例演示如何加载与分隔符字符（在此示例中为竖线字符）匹配的字符。在输入文件中，确保使用反斜杠字符 (\) 转义您要加载的所有竖线字符 (|)。然后使用 ESCAPE 参数加载此文件。

```
$ more redshiftinfo.txt
1|public\|event\|dwuser
2|public\|sales\|dwuser

create table redshiftinfo(infoid int,tableinfo varchar(50));

copy redshiftinfo from 's3://mybucket/data/redshiftinfo.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' escape;

select * from redshiftinfo order by 1;
infoid |       tableinfo
-------+--------------------
1      | public|event|dwuser
2      | public|sales|dwuser
(2 rows)
```

如果没有 ESCAPE 参数，此 COPY 命令将失败，并返回 `Extra column(s) found` 错误。

> **Important**
>
> 如果使用包含 ESCAPE 参数的 COPY 加载数据，则还必须在 UNLOAD 命令中指定 ESCAPE 参数与以生成反向输出文件。同样，如果您使用 ESCAPE 参数执行 UNLOAD 命令，则在您对相同数据执行 COPY 操作时需要使用 ESCAPE 参数。

从 JSON 中复制的示例

在以下示例中，您加载具有以下数据的 CATEGORY 表。

CATID	CATGROUP	CATNAME	CATDESC
1	Sports	MLB	Major League Baseball
2	Sports	NHL	National Hockey League
3	Sports	NFL	National Football League
4	Sports	NBA	National Basketball Association
5	Concerts	Classical	All symphony, concerto, and choir concerts

主题

- 使用“auto”选项从 JSON 数据中加载
- 使用“auto ignorecase”选项从 JSON 数据中加载
- 使用 JSONPaths 文件从 JSON 数据中加载
- 使用 JSONPaths 文件从 JSON 数组中加载

使用“auto”选项从 JSON 数据中加载

要使用 `'auto'` 选项从 JSON 数据加载，JSON 数据必须包含一组对象。键名称必须与列名称匹配，但顺序并不重要。下面显示了名为 `category_object_auto.json` 的文件的内容。

```
{
    "catdesc": "Major League Baseball",
    "catid": 1,
```

```
    "catgroup": "Sports",
    "catname": "MLB"
}
{
    "catgroup": "Sports",
    "catid": 2,
    "catname": "NHL",
    "catdesc": "National Hockey League"
}{
    "catid": 3,
    "catname": "NFL",
    "catgroup": "Sports",
    "catdesc": "National Football League"
}
{
    "bogus": "Bogus Sports LLC",
    "catid": 4,
    "catgroup": "Sports",
    "catname": "NBA",
    "catdesc": "National Basketball Association"
}
{
    "catid": 5,
    "catgroup": "Shows",
    "catname": "Musicals",
    "catdesc": "All symphony, concerto, and choir concerts"
}
```

若要从上例中的 JSON 数据文件加载，请执行以下 COPY 命令。

```
copy category
from 's3://mybucket/category_object_auto.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 'auto';
```

使用“auto ignorecase”选项从 JSON 数据中加载

要使用 'auto ignorecase' 选项从 JSON 数据加载，JSON 数据必须包含一组对象。键名称的大小写不必与列名称匹配，顺序并不重要。下面显示了名为 category_object_auto-ignorecase.json 的文件的内容。

```
{
    "CatDesc": "Major League Baseball",
```

```
    "CatID": 1,
    "CatGroup": "Sports",
    "CatName": "MLB"
}
{
    "CatGroup": "Sports",
    "CatID": 2,
    "CatName": "NHL",
    "CatDesc": "National Hockey League"
}{
    "CatID": 3,
    "CatName": "NFL",
    "CatGroup": "Sports",
    "CatDesc": "National Football League"
}
{
    "bogus": "Bogus Sports LLC",
    "CatID": 4,
    "CatGroup": "Sports",
    "CatName": "NBA",
    "CatDesc": "National Basketball Association"
}
{
    "CatID": 5,
    "CatGroup": "Shows",
    "CatName": "Musicals",
    "CatDesc": "All symphony, concerto, and choir concerts"
}
```

若要从上例中的 JSON 数据文件加载，请执行以下 COPY 命令。

```
copy category
from 's3://mybucket/category_object_auto ignorecase.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 'auto ignorecase';
```

使用 JSONPaths 文件从 JSON 数据中加载

如果 JSON 数据对象未直接对应于列名称，则可使用 JSONPaths 文件将 JSON 元素映射到列。顺序在 JSON 源数据中也不重要，但 JSONPaths 文件表达式的顺序必须与列顺序匹配。假设您具有以下名为 `category_object_paths.json` 的数据文件。

```
{
```

```
    "one": 1,
    "two": "Sports",
    "three": "MLB",
    "four": "Major League Baseball"
}
{
    "three": "NHL",
    "four": "National Hockey League",
    "one": 2,
    "two": "Sports"
}
{
    "two": "Sports",
    "three": "NFL",
    "one": 3,
    "four": "National Football League"
}
{
    "one": 4,
    "two": "Sports",
    "three": "NBA",
    "four": "National Basketball Association"
}
{
    "one": 6,
    "two": "Shows",
    "three": "Musicals",
    "four": "All symphony, concerto, and choir concerts"
}
```

以下名为 `category_jsonpath.json` 的 JSONPaths 文件会将源数据映射到表列。

```
{
    "jsonpaths": [
        "$['one']",
        "$['two']",
        "$['three']",
        "$['four']"
    ]
}
```

若要从上例中的 JSON 数据文件加载，请执行以下 COPY 命令。

```
copy category
from 's3://mybucket/category_object_paths.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 's3://mybucket/category_jsonpath.json';
```

使用 JSONPaths 文件从 JSON 数组中加载

若要从包含一组数组的 JSON 数据加载，必须使用 JSONPaths 文件将数组元素映射到列。假设您具有以下名为 `category_array_data.json` 的数据文件。

```
[1,"Sports","MLB","Major League Baseball"]
[2,"Sports","NHL","National Hockey League"]
[3,"Sports","NFL","National Football League"]
[4,"Sports","NBA","National Basketball Association"]
[5,"Concerts","Classical","All symphony, concerto, and choir concerts"]
```

以下名为 `category_array_jsonpath.json` 的 JSONPaths 文件会将源数据映射到表列。

```
{
    "jsonpaths": [
        "$[0]",
        "$[1]",
        "$[2]",
        "$[3]"
    ]
}
```

若要从上例中的 JSON 数据文件加载，请执行以下 COPY 命令。

```
copy category
from 's3://mybucket/category_array_data.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 's3://mybucket/category_array_jsonpath.json';
```

从 Avro 中复制的示例

在以下示例中，您加载具有以下数据的 CATEGORY 表。

CATID	CATGROUP	CATNAME	CATDESC
1	Sports	MLB	Major League Baseball

CATID	CATGROUP	CATNAME	CATDESC
2	Sports	NHL	National Hockey League
3	Sports	NFL	National Football League
4	Sports	NBA	National Basketball Association
5	Concerts	Classical	All symphony, concerto, and choir concerts

主题

- 使用“auto”选项从 Avro 数据中加载
- 使用“auto ignorecase”选项从 Avro 数据中加载
- 使用 JSONPaths 文件从 Avro 数据中加载

使用“auto”选项从 Avro 数据中加载

若要使用 'auto' 参数从 Avro 数据加载，Avro schema 中的字段名称必须与列名称匹配。在使用 'auto' 参数时，顺序并不重要。下面显示了名为 category_auto.avro 的文件的 schema。

```
{
    "name": "category",
    "type": "record",
    "fields": [
        {"name": "catid", "type": "int"},
        {"name": "catdesc", "type": "string"},
        {"name": "catname", "type": "string"},
        {"name": "catgroup", "type": "string"},
}
```

Avro 文件中的数据为二进制格式，不是人类可读的格式。下面显示了 category_auto.avro 文件中的数据的 JSON 表示形式。

```
{
   "catid": 1,
   "catdesc": "Major League Baseball",
   "catname": "MLB",
```

```
   "catgroup": "Sports"
}
{
   "catid": 2,
   "catdesc": "National Hockey League",
   "catname": "NHL",
   "catgroup": "Sports"
}
{
   "catid": 3,
   "catdesc": "National Basketball Association",
   "catname": "NBA",
   "catgroup": "Sports"
}
{
   "catid": 4,
   "catdesc": "All symphony, concerto, and choir concerts",
   "catname": "Classical",
   "catgroup": "Concerts"
}
```

要从上例中的 Avro 数据文件加载，请执行以下 COPY 命令。

```
copy category
from 's3://mybucket/category_auto.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as avro 'auto';
```

使用“auto ignorecase”选项从 Avro 数据中加载

若要使用 `'auto ignorecase'` 参数从 Avro 数据加载，Avro schema 中的字段名称的大小写不必与列名称的大小写匹配。在使用 `'auto ignorecase'` 参数时，顺序并不重要。下面显示了名为 `category_auto-ignorecase.avro` 的文件的 schema。

```
{
    "name": "category",
    "type": "record",
    "fields": [
        {"name": "CatID", "type": "int"},
        {"name": "CatDesc", "type": "string"},
        {"name": "CatName", "type": "string"},
        {"name": "CatGroup", "type": "string"},
```

```
}
```

Avro 文件中的数据为二进制格式，不是人类可读的格式。下面显示了 category_auto-ignorecase.avro 文件中的数据的 JSON 表示形式。

```
{
   "CatID": 1,
   "CatDesc": "Major League Baseball",
   "CatName": "MLB",
   "CatGroup": "Sports"
}
{
   "CatID": 2,
   "CatDesc": "National Hockey League",
   "CatName": "NHL",
   "CatGroup": "Sports"
}
{
   "CatID": 3,
   "CatDesc": "National Basketball Association",
   "CatName": "NBA",
   "CatGroup": "Sports"
}
{
   "CatID": 4,
   "CatDesc": "All symphony, concerto, and choir concerts",
   "CatName": "Classical",
   "CatGroup": "Concerts"
}
```

要从上例中的 Avro 数据文件加载，请执行以下 COPY 命令。

```
copy category
from 's3://mybucket/category_auto-ignorecase.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as avro 'auto ignorecase';
```

使用 JSONPaths 文件从 Avro 数据中加载

如果 Avro schema 中的字段名称未直接对应于列名称，则可使用 JSONPaths 文件将 schema 元素映射到列。JSONPaths 文件表达式的顺序必须与列顺序一致。

假设您具有名为 category_paths.avro 的数据文件，其中包含的数据与上例相同，但具有以下架构。

```
{
    "name": "category",
    "type": "record",
    "fields": [
        {"name": "id", "type": "int"},
        {"name": "desc", "type": "string"},
        {"name": "name", "type": "string"},
        {"name": "group", "type": "string"},
        {"name": "region", "type": "string"}
     ]
}
```

以下名为 category_path.avropath 的 JSONPaths 文件会将源数据映射到表列。

```
{
    "jsonpaths": [
        "$['id']",
        "$['group']",
        "$['name']",
        "$['desc']"
    ]
}
```

要从上例中的 Avro 数据文件加载，请执行以下 COPY 命令。

```
copy category
from 's3://mybucket/category_object_paths.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format avro 's3://mybucket/category_path.avropath ';
```

使用 ESCAPE 选项为 COPY 准备文件

以下示例描述了在使用包含 ESCAPE 参数的 COPY 命令将数据导入到 Amazon Redshift 表中之前，如何准备数据以“转义”换行符。如果未准备数据以限定换行符，则 Amazon Redshift 将会在您运行 COPY 命令时返回加载错误，因为换行符一般用作记录分隔符。

例如，考虑要复制到 Amazon Redshift 表中的一个文件或外部表中的一个列。如果该文件或列包含 XML 格式的内容或类似数据，则需要确保使用反斜杠字符 (\) 转义此内容中的所有换行符 (\n)。

包含嵌入换行符的文件或表提供了相对轻松的匹配模式。每个嵌入的换行符很有可能始终跟随一个 > 字符（在这二者之间可能还包含一些空格字符（' ' 或制表符）），如下面的名为 nlTest1.txt 的文本文件的示例中所示。

```
$ cat nlTest1.txt
<xml start>
<newline characters provide>
<line breaks at the end of each>
<line in content>
</xml>|1000
<xml>
</xml>|2000
```

在以下示例中，您可运行文本处理实用工具预先处理源文件，并在需要的位置插入转义字符。（| 字符旨在用作分隔符，以便在列数据复制到 Amazon Redshift 表中后分隔这些数据。）

```
$ sed -e ':a;N;$!ba;s/>[[:space:]]*\n/>\\\n/g' nlTest1.txt > nlTest2.txt
```

同样，可使用 Perl 执行类似操作：

```
cat nlTest1.txt | perl -p -e 's/>\s*\n/>\\\n/g' > nlTest2.txt
```

为了便于将 nlTest2.txt 文件中的数据加载到 Amazon Redshift 中，我们在 Amazon Redshift 中创建了一个包含两列的表。第一列 c1 是字符列，用于放置 nlTest2.txt 文件中 XML 格式的内容。第二列 c2 将放置从同一文件加载的整数值。

在运行 sed 命令后，可使用 ESCAPE 参数将 nlTest2.txt 文件中的数据正确地加载到 Amazon Redshift 表中。

> **Note**
>
> 如果您在 COPY 命令中包含 ESCAPE 参数，则它会将一些包含反斜杠字符的特殊字符（包括换行符）进行转义。

```
copy t2 from 's3://mybucket/data/nlTest2.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
escape
delimiter as '|';
```

```
select * from t2 order by 2;

c1           |  c2
-------------+------
<xml start>
<newline characters provide>
<line breaks at the end of each>
<line in content>
</xml>
| 1000
<xml>
</xml>       | 2000
(2 rows)
```

您可以类似方式准备从外部数据库导出的数据文件。例如，对于 Oracle 数据库，可对要复制到 Amazon Redshift 中的表中的每个受影响的列使用 REPLACE 函数。

```
SELECT c1, REPLACE(c2, \n',\\n' ) as c2 from my_table_with_xml
```

此外，许多用于定期处理大量数据的数据库导出和提取、转换、加载 (ETL) 工具提供了指定转义字符和分隔符字符的选项。

将 shapefile 加载到 Amazon Redshift

以下示例演示如何使用 COPY 加载 Esri shapefile。有关加载 shapefile 的更多信息，请参阅将 shapefile 加载到 Amazon Redshift。

加载一个 shapefile

以下步骤介绍如何使用 COPY 命令从 Amazon S3 中摄取 OpenStreetMap 数据。此示例假定 Geofabrik 下载站点的 Norway shapefile 归档已经上载到 AWS 区域中的私有 Amazon S3 桶。`.shp`、`.shx` 和 `.dbf` 文件必须共享相同的 Amazon S3 前缀和文件名。

无需简化即可摄取数据

以下命令可创建表并摄取可适合最大几何大小的数据，而无需进行任何简化。在您的首选 GIS 软件中打开 `gis_osm_natural_free_1.shp`，然后检查此图层中的列。预设情况下，IDENTITY 或 GEOMETRY 列位于首位。当 GEOMETRY 列位于首位时，您可以创建表，如下所示。

```
CREATE TABLE norway_natural (
```

```
   wkb_geometry GEOMETRY,
   osm_id BIGINT,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

或者，当 IDENTITY 列位于首位时，您可以创建表，如下所示。

```
CREATE TABLE norway_natural_with_id (
   fid INT IDENTITY(1,1),
   wkb_geometry GEOMETRY,
   osm_id BIGINT,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

现在，您可以使用 COPY 摄取数据。

```
COPY norway_natural FROM 's3://bucket_name/shapefiles/norway/
gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO: Load into table 'norway_natural' completed, 83891 record(s) loaded successfully
```

或者，您可以按如下所示摄取数据。

```
COPY norway_natural_with_id FROM 's3://bucket_name/shapefiles/norway/
gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO: Load into table 'norway_natural_with_id' completed, 83891 record(s) loaded
 successfully.
```

通过简化摄取数据

以下命令创建一个表，并尝试在不进行任何简化的情况下摄取无法适合最大几何大小的数据。检查 `gis_osm_water_a_free_1.shp` shapefile 并创建相应的表，如下所示。

```
CREATE TABLE norway_water (
   wkb_geometry GEOMETRY,
   osm_id BIGINT,
```

```
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

当 COPY 命令运行时，会导致错误。

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
ERROR:  Load into table 'norway_water' failed.  Check 'stl_load_errors' system table
 for details.
```

查询 STL_LOAD_ERRORS 显示几何体过大。

```
SELECT line_number, btrim(colname), btrim(err_reason) FROM stl_load_errors WHERE query
 = pg_last_copy_id();
 line_number |    btrim     |                                  btrim
-------------+--------------
+-------------------------------------------------------------------------
     1184705 | wkb_geometry | Geometry size: 1513736 is larger than maximum supported
 size: 1048447
```

为了克服这个问题，将 SIMPLIFY AUTO 参数添加到 COPY 命令中以简化几何体。

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
SIMPLIFY AUTO
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';

INFO:  Load into table 'norway_water' completed, 1989196 record(s) loaded successfully.
```

要查看简化的行和几何体，请查询 SVL_SPATIAL_SIMPLIFY。

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
 query | line_number | maximum_tolerance | initial_size | simplified | final_size |
 final_tolerance
-------+-------------+-------------------+--------------+------------+------------
+----------------------
    20 |     1184704 |                -1 |      1513736 | t          |    1008808 |
 1.276386653895e-05
```

```
    20 |     1664115 |               -1 |      1233456 | t          |    1023584 |
 6.11707814796635e-06
```

使用容差低于自动计算容差的 SIMPLIFY AUTO max_tolerance 可能会导致摄入误差。在这种情况下，请使用 MAXERROR 忽略错误。

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
SIMPLIFY AUTO 1.1E-05
MAXERROR 2
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';

INFO:  Load into table 'norway_water' completed, 1989195 record(s) loaded successfully.
INFO:  Load into table 'norway_water' completed, 1 record(s) could not be loaded.
 Check 'stl_load_errors' system table for details.
```

再次查询 SVL_SPATIAL_SIMPLIFY 来识别 COPY 未成功加载的记录。

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
 query | line_number | maximum_tolerance | initial_size | simplified | final_size |
 final_tolerance
-------+-------------+-------------------+--------------+------------+------------
+-----------------
    29 |     1184704 |           1.1e-05 |      1513736 | f          |          0 |
            0
    29 |     1664115 |           1.1e-05 |      1233456 | t          |     794432 |
     1.1e-05
```

在这个示例中，第一条记录没有成功适应，因此 simplified 列显示为 false。第二条记录已在给定容差范围内加载。但是，最终大小比使用自动计算的容差大，而不指定最大公差。

正在从压缩的 shapefile 文件加载

Amazon Redshift COPY 支持从压缩的 shapefile 中摄取数据。所有 shapefile 组件必须具有相同的 Amazon S3 前缀和相同的压缩后缀。例如，假设您要加载上面示例中的数据。在本例中，文件 gis_osm_water_a_free_1.shp.gz、gis_osm_water_a_free_1.dbf.gz 和 gis_osm_water_a_free_1.shx.gz 必须共享相同的 Amazon S3 目录。COPY 命令需要 GZIP 选项，FROM 子句必须指定正确的压缩文件，如下所示。

```
COPY norway_natural FROM 's3://bucket_name/shapefiles/norway/compressed/
gis_osm_natural_free_1.shp.gz'
```

```
FORMAT SHAPEFILE
GZIP
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO:  Load into table 'norway_natural' completed, 83891 record(s) loaded successfully.
```

正在将数据加载到具有不同列顺序的表

如果您有一个没有将 GEOMETRY 作为第一列的表，则可以使用列映射将列映射到目标表。例如，创建一个将 osm_id 指定为第一列的表。

```
CREATE TABLE norway_natural_order (
   osm_id BIGINT,
   wkb_geometry GEOMETRY,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

然后使用列映射摄取 shapefile。

```
COPY norway_natural_order(wkb_geometry, osm_id, code, fclass, name)
FROM 's3://bucket_name/shapefiles/norway/gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO:  Load into table 'norway_natural_order' completed, 83891 record(s) loaded
 successfully.
```

将数据加载到具有 geography 列的表中

如果某个表具有 GEOGRAPHY 列，请首先提取到 GEOMETRY 列，然后将对象强制转换为 GEOGRAPHY 对象。例如，在将 shapefile 复制到 GEOMETRY 列后，对表进行更改，添加 GEOGRAPHY 数据类型。

```
ALTER TABLE norway_natural ADD COLUMN wkb_geography GEOGRAPHY;
```

然后将 geometry 转换为 geography。

```
UPDATE norway_natural SET wkb_geography = wkb_geometry::geography;
```

（可选）您可以删除 GEOMETRY 列。

```
ALTER TABLE norway_natural DROP COLUMN wkb_geometry;
```

带有 NOLOAD 选项的 COPY 命令

要在实际加载数据之前验证数据文件，请使用带有 NOLOAD 选项的 COPY 命令。Amazon Redshift 会解析输入文件并显示发生的任何错误。以下示例使用 NOLOAD 选项，实际上并未将任何行加载到表中。

```
COPY public.zipcode1
FROM 's3://mybucket/mydata/zipcode.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
NOLOAD
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/myRedshiftRole';

Warnings:
Load into table 'zipcode1' completed, 0 record(s) loaded successfully.
```

CREATE DATABASE

创建新数据库。

要创建数据库，您必须是超级用户或拥有 CREATEDB 权限。要创建与零 ETL 集成关联的数据库，您必须是超级用户或同时拥有 CREATEDB 和 CREATEUSER 权限。

您不能在以下事务块中运行 CREATE DATABASE：(BEGIN ... END)。有关事务的更多信息，请参阅可序列化的隔离。

语法

```
CREATE DATABASE database_name
[ { [ WITH ]
    [ OWNER [=] db_owner ]
    [ CONNECTION LIMIT { limit | UNLIMITED } ]
    [ COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE } ]
    [ ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT } ]
  }
  | { [ WITH PERMISSIONS ] FROM DATASHARE datashare_name ] OF [ ACCOUNT account_id ]
 NAMESPACE namespace_guid }
  | { FROM { { ARN '<arn>' } { WITH DATA CATALOG SCHEMA '<schema>' | WITH NO DATA
 CATALOG SCHEMA } }
             | { INTEGRATION '<integration_id>'} }
```

```
  | { IAM_ROLE  {default | 'SESSION' | 'arn:aws:iam::<account-id>:role/<role-name>' } }
```

参数

database_name

新数据库的名称。有关有效名称的更多信息，请参阅名称和标识符。

WITH

可选关键字。

OWNER

指定数据库所有者。

=

可选字符。

db_owner

数据库所有者的用户名。

CONNECTION LIMIT { limit | UNLIMITED }

允许用户同时打开的数据库连接的最大数量。此限制不适用于超级用户。使用 UNLIMITED 关键字设置允许的并行连接的最大数量。可能对每个用户的连接数量也会施加限制。有关更多信息，请参阅 CREATE USER。默认为 UNLIMITED。要查看当前连接，请查询 STV_SESSIONS 系统视图。

> **Note**
>
> 如果用户及数据库连接限制均适用，当用户尝试连接时，必须有一个同时低于这两个限制的未使用的连接槽可用。

COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }

指定字符串搜索或比较是 CASE_SENSITIVE 还是 CASE_INSENSITIVE 的子句。默认值为 CASE_SENSITIVE。

ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }

指定针对数据库运行查询时使用的隔离级别的子句。

- SERIALIZABLE 隔离 – 为并发事务提供完全可序列性。有关更多信息，请参阅 可序列化的隔离。

- SNAPSHOT 隔离 – 提供隔离级别，来防止出现更新和删除冲突。这是在预调配集群或无服务器命名空间中创建的数据库的默认设置。

您可以按如下方式查看数据库运行的并发模型：

- 查询 STV_DB_ISOLATION_LEVEL 目录视图。有关更多信息，请参阅 STV_DB_ISOLATION_LEVEL。

```
SELECT * FROM stv_db_isolation_level;
```

- 查询 PG_DATABASE_INFO 视图。

```
SELECT datname, datconfig FROM pg_database_info;
```

每个数据库的隔离级别出现在 `concurrency_model` 键旁边。值为 1 时，表示 SNAPSHOT。值为 2 时，表示 SERIALIZABLE。

在 Amazon Redshift 数据库中，SERIALIZABLE 和 SNAPSHOT 隔离都是可序列化隔离级别的类型。也就是说，根据 SQL 标准，防止脏读、不可重复读和幻读。这两种隔离级别都可保证某事务对该事务开始时就存在的数据快照进行操作，并且没有任何其他事务可以更改该快照。但是，SNAPSHOT 隔离不能提供完全可序列性，因为它不能防止在不同的表行上进行写入偏斜插入和更新。

以下场景说明了使用 SNAPSHOT 隔离级别的写入偏斜更新。名为 `Numbers` 的表包含名为 `digits` 的列，其中包含 `0` 和 `1` 值。每个用户的 UPDATE 语句不会与其他用户重叠。但是，`0` 和 `1` 值可交换。它们运行的 SQL 遵循以下时间表，结果如下：

时间	用户 1 操作	用户 2 操作
1	BEGIN;	
2		BEGIN;
3	SELECT * FROM Numbers	

时间	用户 1 操作	用户 2 操作
	digits - ----- 0 1	
4		SELECT * FROM Numbers; digits ------ 0 1
5	UPDATE Numbers SET digits=0 WHERE digits=1;	
6	SELECT * FROM Numbers digits - ----- 0 0	
7	COMMIT	
8		Update Numbers SET digits=1 WHERE digits=0;

时间	用户 1 操作	用户 2 操作
9		SELECT * FROM Numbers; digits ------ 1 1
10		COMMIT;
11	SELECT * FROM Numbers; digits ----- 1 0	
12		SELECT * FROM Numbers; digits ------ 1 0

如果使用可序列化的隔离运行同一场景，则 Amazon Redshift 因可序列化违规而终止用户 2 并返回错误 1023。有关更多信息，请参阅 如何修复可序列化的隔离错误。在这种情况下，只有用户 1 可以成功提交。并非所有工作负载都需要可序列化隔离作为一项要求，在这种情况下，快照隔离足以作为数据库的目标隔离级别。

FROM ARN '<ARN>'

用于创建数据库的 AWS Glue 数据库 ARN。

{ DATA CATALOG SCHEMA '<schema>' | WITH NO DATA CATALOG SCHEMA }

> **Note**
>
> 仅当您的 CREATE DATABASE 命令也使用 FROM ARN 参数时，此参数才适用。

指定是否使用架构创建数据库以帮助访问 AWS Glue Data Catalog 中的对象。

FROM INTEGRATION '<integration_id>'

指定是否使用零 ETL 集成标识符创建数据库。您可以从 SVV_INTEGRATION 系统视图中检索 `integration_id`。有关示例，请参阅创建数据库以接收零 ETL 集成的结果。有关使用零 ETL 集成创建数据库的更多信息，请参阅《Amazon Redshift 管理指南》中的在 Amazon Redshift 中创建目标数据库。

IAM_ROLE { default | 'SESSION' | 'arn:aws:iam::*<AWS ##-id>*:role/*<role-name>*' }

> **Note**
>
> 仅当您的 CREATE DATABASE 命令也使用 FROM ARN 参数时，此参数才适用。

如果您在运行 CREATE DATABASE 命令时指定与集群关联的 IAM 角色，则 Amazon Redshift 将在您对数据库运行查询时使用该角色的凭证。

指定 `default` 关键字表示要使用设置为默认并与集群关联的 IAM 角色。

如果您使用联合身份连接到 Amazon Redshift 集群并访问使用此命令创建的外部架构中的表，则使用 `'SESSION'`。有关使用联合身份的示例，请参阅使用联合身份管理 Amazon Redshift 对本地资源和 Amazon Redshift Spectrum 外部表的访问权限，其中说明了如何配置联合身份。

使用 IAM 角色的 Amazon 资源名称（ARN），您的集群使用该角色进行身份验证和授权。IAM 角色至少必须有权在要访问的 Amazon S3 桶上执行 LIST 操作和有权在该桶包含的 Amazon S3 对象上执行 GET 操作。要了解有关在使用 AWS Glue Data Catalog 为数据共享创建数据库时使用 IAM_ROLE 的更多信息，请参阅以使用者身份使用 Lake Formation 托管的数据共享。

下面显示了单个 ARN 的 IAM_ROLE 参数字符串的语法。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

您可以将角色串联起来，以便集群可以承担另一个 IAM 角色 (可能属于其他账户)。您最多可串联 10 个角色。有关更多信息，请参阅 在 Amazon Redshift Spectrum 中链接 IAM 角色。

对于此 IAM 角色，请附加类似于以下内容的 IAM 权限策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

有关创建 IAM 角色以用于联合查询的步骤，请参阅创建密钥和 IAM 角色以使用联合查询。

> **Note**
>
> 请不要在链接的角色列表中包含空格。

下面显示了串联三个角色的语法。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-
account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

将 CREATE DATABASE 与数据共享结合使用的语法

以下语法描述了用于从数据共享中创建数据库以在同一 AWS 账户内共享数据的 CREATE DATABASE 命令。

```
CREATE DATABASE database_name
[ [ WITH PERMISSIONS ] FROM DATASHARE datashare_name ] OF [ ACCOUNT account_id ]
 NAMESPACE namespace_guid
```

以下语法描述了用于从数据共享中创建数据库以在 AWS 账户间共享数据的 CREATE DATABASE 命令。

```
CREATE DATABASE database_name
[ [ WITH PERMISSIONS ] FROM DATASHARE datashare_name ] OF ACCOUNT account_id
 NAMESPACE namespace_guid
```

将 CREATE DATABASE 与数据共享结合使用的参数

FROM DATASHARE

指示数据库所在位置的关键词。

datashare_name

创建使用者数据库所在的数据共享的名称。

WITH PERMISSIONS

指定通过数据共享创建的数据库需要对象级权限才能访问各个数据库对象。如果没有此子句，则被授予对数据库的 USAGE 权限的用户或角色将自动有权访问数据库中的所有数据库对象。

NAMESPACE namespace_guid

指定数据共享所属的创建者命名空间的值。

ACCOUNT account_id

指定数据共享所属的创建者账户的值。

用于数据共享的 CREATE DATABASE 的使用说明

作为数据库超级用户，当您使用 CREATE DATABASE 从 AWS 账户内的数据共享创建数据库时，指定 NAMESPACE 选项。ACCOUNT 选项为可选项。当您使用 CREATE DATABASE 从 AWS 账户间的数据共享创建数据库时，同时指定创建者的 ACCOUNT 和 NAMESPACE。

对于使用者集群上的数据共享，一个数据共享仅可创建一个使用者数据库。不能创建多个引用同一数据共享的使用者数据库。

CREATE DATABASE（从 AWS Glue Data Catalog）

要使用 AWS Glue 数据库 ARN 创建数据库，请在 CREATE DATABASE 命令中指定 ARN。

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH NO DATA CATALOG SCHEMA;
```

或者，您也可以为 IAM_ROLE 参数提供一个值。有关参数和接受的值的更多信息，请参阅参数。

以下示例演示了如何使用 IAM 角色从 ARN 创建数据库。

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH NO DATA CATALOG SCHEMA
 IAM_ROLE <iam-role-arn>
```

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH NO DATA CATALOG SCHEMA
 IAM_ROLE default;
```

您也可以使用 DATA CATALOG SCHEMA 创建数据库。

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH DATA CATALOG SCHEMA
 <sample_schema> IAM_ROLE default;
```

创建数据库以接收零 ETL 集成的结果

要使用零 ETL 集成标识创建数据库，请在 CREATE DATABASE 命令中指定 `integration_id`。

```
CREATE DATABASE destination_db_name FROM INTEGRATION 'integration_id';
```

例如，首先从 SVV_INTEGRATION 中获取集成 ID；

```
SELECT integration_id FROM SVV_INTEGRATION;
```

然后使用检索到的其中一个集成 ID 创建接收零 ETL 集成的数据库。

```
CREATE DATABASE sampledb FROM INTEGRATION 'a1b2c3d4-5678-90ab-cdef-EXAMPLE11111';
```

CREATE DATABASE 限制

Amazon Redshift 针对数据库强制实施以下限制：

- 每个集群最多 60 个用户定义的数据库。
- 数据库名称最多为 127 个字节。
- 数据库名称不能使用保留字。

数据库排序规则

排序规则是一组规则，用于定义数据库引擎如何对 SQL 中的字符类型数据进行比较和排序。不区分大小写的排序规则是最常用的排序规则。Amazon Redshift 使用不区分大小写的排序规则以帮助从其他数据仓库系统迁移。凭借对不区分大小写的排序规则的本机支持，Amazon Redshift 继续使用重要的调整或优化方法，如分配键、排序键或范围限制扫描。

COLLATE 子句指定数据库中所有 CHAR 和 VARCHAR 列的默认排序规则。如果指定了 CASE_INSENSITIVE，则所有 CHAR 或 VARCHAR 列都使用不区分大小写的排序规则。有关排序规则的信息，请参阅排序规则序列。

在不区分大小写的列中插入或摄取的数据将保持其原始大小写。但是所有基于比较的字符串操作，包括排序和分组都不区分大小写。模式匹配操作（如类似于的 LIKE 谓词）和正则表达式函数也不区分大小写。

以下 SQL 操作支持适用的排序规则语义：

- 比较运算符：=、<>、<、<=、>、>=。
- LIKE 运算符
- ORDER BY 子句
- GROUP BY 子句
- 使用字符串比较的聚合函数，例如 MIN、MAX 和 LISTAGG
- 窗口函数，如 PARTITION BY 子句和 ORDER BY 子句
- 标量函数 greatest() 和 least()、STRPOS()、REGEXP_COUNT()、REGEXP_REPLACE()、REGEXP_INSTR()、REGEXP_SUBS

- Distinct 子句
- UNION、INTERSECT 和 EXCEPT
- IN LIST

对于外部查询（包括 Amazon Redshift Spectrum 和 Aurora PostgreSQL 联合查询），VARCHAR 或 CHAR 列的排序规则与当前数据库级别的排序规则相同。

以下示例将查询 Amazon Redshift Spectrum 表：

```
SELECT ci_varchar FROM spectrum.test_collation
WHERE ci_varchar = 'AMAZON';

ci_varchar
----------
amazon
Amazon
AMAZON
AmaZon
(4 rows)
```

有关如何使用数据库排序规则创建表的信息，请参阅CREATE TABLE。

有关 COLLATE 函数的信息，请参阅COLLATE 函数。

数据库排序规则限制

以下是在 Amazon Redshift 中使用数据库排序规则时的限制：

- 所有系统表或视图（包括 PG 目录表和 Amazon Redshift 系统表）均区分大小写。
- 当使用者数据库和生产者数据库具有不同的数据库级排序规则时，Amazon Redshift 不支持跨数据库和跨集群查询。
- Amazon Redshift 不支持仅领导节点查询中不区分大小写的排序规则。

  以下示例显示了一个不受支持的不区分大小写的查询以及 Amazon Redshift 发送的错误：

  ```
  SELECT collate(usename, 'case_insensitive') FROM pg_user;
  ERROR:  Case insensitive collation is not supported in leader node only query.
  ```

- Amazon Redshift 不支持区分大小写和不区分大小写的列之间的交互，例如比较、函数、联接或集合运算。

以下示例显示了区分大小写和不区分大小写的列交互时出现的错误：

```
CREATE TABLE test
  (ci_col varchar(10) COLLATE case_insensitive,
   cs_col varchar(10) COLLATE case_sensitive,
   cint int,
   cbigint bigint);
```

```
SELECT ci_col = cs_col FROM test;
ERROR:  Query with different collations is not supported yet.
```

```
SELECT concat(ci_col, cs_col) FROM test;
ERROR:  Query with different collations is not supported yet.
```

```
SELECT ci_col FROM test UNION SELECT cs_col FROM test;
ERROR:  Query with different collations is not supported yet.
```

```
SELECT * FROM test a, test b WHERE a.ci_col = b.cs_col;
ERROR:  Query with different collations is not supported yet.
```

```
Select Coalesce(ci_col, cs_col) from test;
ERROR:  Query with different collations is not supported yet.
```

```
Select case when cint > 0 then ci_col else cs_col end from test;
ERROR:  Query with different collations is not supported yet.
```

- Amazon Redshift 不支持 SUPER 数据类型的排序规则。不支持在不区分大小写的数据库中创建 SUPER 列以及 SUPER 列和不区分大小写的列之间的交互。

  以下示例在不区分大小写的数据库中创建一个以 SUPER 作为数据类型的表：

  ```
  CREATE TABLE super_table (a super);
  ERROR: SUPER column is not supported in case insensitive database.
  ```

  以下示例使用与 SUPER 数据进行比较的不区分大小写的字符串查询数据：

  ```
  CREATE TABLE test_super_collation
  ```

```
(s super, c varchar(10) COLLATE case_insensitive, i int);
```

```
SELECT s = c FROM test_super_collation;
ERROR:  Coercing from case insensitive string to SUPER is not supported.
```

要使这些查询起作用，请使用 COLLATE 函数将一个列的排序规则转换为与另一列匹配。有关更多信息，请参阅 COLLATE 函数。

示例

创建数据库

以下示例创建名为 TICKIT 的数据库并将所有权授予用户 DWUSER。

```
create database tickit
with owner dwuser;
```

要查看有关数据库的详细信息，请查询 PG_DATABASE_INFO 目录表。

```
select datname, datdba, datconnlimit
from pg_database_info
where datdba > 1;

 datname     | datdba | datconnlimit
-------------+--------+-------------
 admin       |    100 | UNLIMITED
 reports     |    100 | 100
 tickit      |    100 | 100
```

以下示例使用 SNAPSHOT 隔离级别创建名为 **sampledb** 的数据库。

```
CREATE DATABASE sampledb ISOLATION LEVEL SNAPSHOT;
```

以下示例从数据共享 salesshare 中创建了数据库 sales_db。

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

数据库排序规则示例

创建不区分大小写的数据库

以下示例将创建 sampledb 数据库、创建 T1 表，并将数据插入 T1 表中。

```
create database sampledb collate case_insensitive;
```

连接到您刚刚使用 SQL 客户端创建的新数据库。使用 Amazon Redshift 查询编辑器 v2 时，在编辑器中选择 sampledb。使用 RSQL 时，请使用如下命令。

```
\connect sampledb;
```

```
CREATE TABLE T1 (
  col1 Varchar(20) distkey sortkey
);
```

```
INSERT INTO T1 VALUES ('bob'), ('john'), ('Mary'), ('JOHN'), ('Bob');
```

然后，查询查找含有 John 的结果。

```
SELECT * FROM T1 WHERE col1 = 'John';

 col1
 ------
 john
 JOHN
(2 row)
```

按不区分大小写的顺序排序

以下示例显示了表 T1 中不区分大小写的排序。Bob 与 bob 或 John 与 john 的排序具有不确定性，因为它们在不区分大小写的列中是相等的。

```
SELECT * FROM T1 ORDER BY 1;

 col1
 ------
 bob
 Bob
```

```
 JOHN
 john
 Mary
(5 rows)
```

同样，以下示例显示了 GROUP BY 子句中不区分大小写的排序。Bob 和 bob 是相等的，并且属于同一个组。结果中显示哪一个是不确定的。

```
SELECT col1, count(*) FROM T1 GROUP BY 1;

 col1 | count
 -----+------
 Mary |  1
 bob  |  2
 JOHN |  2
(3 rows)
```

使用窗口函数对不区分大小写的列进行查询

以下示例在不区分大小写的列上查询窗口函数。

```
SELECT col1, rank() over (ORDER BY col1) FROM T1;

 col1 | rank
 -----+------
 bob  |   1
 Bob  |   1
 john |   3
 JOHN |   3
 Mary |   5
(5 rows)
```

使用 DISTINCT 关键词进行查询

以下示例查询带有 DISTINCT 关键词的 T1 表。

```
SELECT DISTINCT col1 FROM T1;

 col1
 ------
 bob
 Mary
```

```
 john
(3 rows)
```

使用 UNION 子句进行查询

以下示例显示来自表 T1 和 T2 的 UNION 的结果。

```
CREATE TABLE T2 AS SELECT * FROM T1;
```

```
SELECT col1 FROM T1 UNION SELECT col1 FROM T2;

 col1
 ------
 john
 bob
 Mary
(3 rows)
```

CREATE DATASHARE

在当前数据库中创建一个新数据共享。此数据共享的拥有者为 CREATE DATASHARE 命令的发布者。

Amazon Redshift 将每个数据共享与一个 Amazon Redshift 数据库相关联。您只能将关联数据库中的对象添加到数据共享中。您可以在同一个 Amazon Redshift 数据库上创建多个数据共享。

有关数据共享的信息，请参阅管理数据共享任务。

要查看有关数据共享的信息，请使用SHOW DATASHARES。

所需的权限

以下是 CREATE DATASHARE 所需的权限：

- Superuser
- 具有 CREATE DATASHARE 权限的用户
- 数据库拥有者

语法

```
CREATE DATASHARE datashare_name
```

```
[[SET] PUBLICACCESSIBLE [=] TRUE | FALSE ];
```

参数

datashare_name

数据共享的名称。数据共享名称在集群命名空间中必须是唯一的。

[[SET] PUBLICACCESSIBLE]

指定是否可以将数据共享共享给可公开访问的集群的子句。

SET PUBLICACCESSIBLE 的默认值为 FALSE。

使用说明

预设情况下，数据共享的拥有者仅拥有共享，而不拥有共享中的对象。

只有超级用户和数据库拥有者才能使用 CREATE DATASHARE 并将 ALTER 权限委派给其他用户或组。

示例

以下示例创建了数据共享 salesshare。

```
CREATE DATASHARE salesshare;
```

以下示例创建了 AWS Data Exchange 管理的数据共享 demoshare。

```
CREATE DATASHARE demoshare SET PUBLICACCESSIBLE TRUE, MANAGEDBY ADX;
```

CREATE EXTERNAL FUNCTION

根据 Amazon Redshift 的 AWS Lambda 创建标量用户定义函数 (UDF)。有关 Lambda 用户定义函数的更多信息，请参阅创建标量 Lambda UDF。

所需的权限

以下是 CREATE EXTERNAL FUNCTION 所需的权限：

- Superuser

- 具有 CREATE [OR REPLACE] EXTERNAL FUNCTION 权限的用户

语法

```
CREATE [ OR REPLACE ] EXTERNAL FUNCTION external_fn_name ( [data_type] [, ...] )
RETURNS data_type
{ VOLATILE | STABLE }
LAMBDA 'lambda_fn_name'
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>'
RETRY_TIMEOUT milliseconds
MAX_BATCH_ROWS count
MAX_BATCH_SIZE size [ KB | MB ];
```

参数

OR REPLACE

一个子句，指定如果某个函数与已存在的此函数具有相同的名称和输入参数数据类型或签名，则替换现有的函数。您只能将某个函数替换为定义一组相同数据类型的新函数。您必须是超级用户才能替换函数。

如果您定义的函数与现有函数具有相同的名称，但签名不同，则创建新的函数。换而言之，函数名称将会重载。有关更多信息，请参阅 重载函数名称。

external_fn_name

外部函数的名称。如果您指定 schema 名称（例如 myschema.myfunction），则使用指定的 schema 创建函数。否则，将在当前 schema 中创建该函数。有关有效名称的更多信息，请参阅名称和标识符。

我们建议您将所有 UDF 名称添加前缀 f_。Amazon Redshift 保留 f_ 前缀，用于 UDF 名称。通过使用 f_ 前缀，您可以帮助确保您的 UDF 名称现在或将来不会与 Amazon Redshift 的任何内置 SQL 函数名称发生冲突。有关更多信息，请参阅 对 UDF 命名。

data_type

输入参数的数据类型。有关更多信息，请参阅 数据类型。

RETURNS data_type

函数返回的值的数据类型。RETURNS 数据类型可以是任何标准的 Amazon Redshift 数据类型。有关更多信息，请参阅 Python UDF 数据类型。

VOLATILE | STABLE

通知查询优化程序有关函数的不稳定性。

要获得最大程度的优化，将函数标记为其有效的最严格稳定性类别。按照严格性顺序，从最不严格的开始，稳定性类别如下所示：

- VOLATILE
- STABLE

VOLATILE

对于相同的参数，函数会对连续的调用返回不同的结果，甚至对于单个语句中的行也是如此。查询优化器无法对不稳定函数的行为做出假设。使用不稳定函数的查询必须对每个输入重新计算该函数。

STABLE

对于相同的参数，可保证函数对在单个语句内处理的所有后续调用返回相同的结果。在不同的语句中调用时，函数可能会返回不同的结果。通过这一类别，优化器可以减少在单个语句中调用函数的次数。

请注意，如果所选的严格性对函数无效，则存在优化器可能会基于这种严格性跳过某些调用的风险。这可能会导致结果集不正确。

Lambda UDF 目前不支持 IMMUTABLE 子句。

LAMBDA 'lambda_fn_name'

Amazon Redshift 调用的函数的名称。

有关创建 AWS Lambda 函数的步骤，请参阅《AWS Lambda 开发人员指南》中的使用控制台创建 Lambda 函数。

有关 Lambda 函数所需权限的信息，请参阅《AWS Lambda 开发人员指南》中的 AWS Lambda 权限。

IAM_ROLE { default | 'arn:aws:iam::*<AWS ##-id>*:role/*<role-name>*'

使用默认关键字让 Amazon Redshift 使用 IAM 角色，该角色设置为默认值并在 CREATE EXTERNAL FUNCTION 命令运行时与集群关联。

使用 IAM 角色的 Amazon 资源名称（ARN），您的集群使用该角色进行身份验证和授权。CREATE EXTERNAL FUNCTION 命令被授权通过此 IAM 角色调用 Lambda 函数。如果您的

集群具有有权调用所附加 Lambda 函数的现有 IAM 角色，您可以替换您的角色的 ARN。有关更多信息，请参阅 配置 Lambda UDF 的授权参数。

以下显示 IAM_ROLE 参数的语法。

```
IAM_ROLE 'arn:aws:iam::aws-account-id:role/role-name'
```

RETRY_TIMEOUT 毫秒

Amazon Redshift 用于重试退避延迟的总时间（以毫秒为单位）。

Amazon Redshift 不会立即重试任何失败的查询，而是执行退避并等待一定时间间隔再重试。然后，Amazon Redshift 将重试请求，以重新运行失败的查询，直到所有延迟的总和等于或超过您指定的 RETRY_TIMEOUT 值。默认值为 20000 毫秒。

当调用 Lambda 函数时，Amazon Redshift 会重试接收 `TooManyRequestsException`、`EC2ThrottledException` 和 `ServiceException` 等错误的查询。

您可以将 RETRY_TIMEOUT 参数设置为 0 毫秒，以防止对 Lambda UDF 进行任何重试。

MAX_BATCH_ROWS count

Amazon Redshift 在单个批处理请求中为单个 lambda 调用发送的最大行数。

此参数的最小值为 1。最大值为 INT_MAX 或 2,147,483,647。

此参数为可选的。默认值为 INT_MAX 或 2,147,483,647。

MAX_BATCH_SIZE size [KB | MB]

Amazon Redshift 在单个批处理请求中为单个 lambda 调用发送的数据负载的最大大小。

此参数的最小值为 1 KB。最大值为 5 MB。

此参数的默认值为 5 MB。

KB 和 MB 是可选的。如果您未设置计量单位，则 Amazon Redshift 默认使用 KB。

使用说明

创建 Lambda UDF 时请考虑以下几点：

- Lambda 函数调用输入参数的顺序不是固定的或有保证的。这可能因正在运行的查询实例而异，具体取决于集群配置。
- 不能保证函数对每个输入参数应用一次，且只应用一次。Amazon Redshift 和 AWS Lambda 之间的交互可能会导致使用相同输入的重复调用。

示例

以下是使用标量 Lambda 用户定义函数 (UDF) 的示例。

使用 Node.js Lambda 函数的标量 Lambda UDF 示例

以下示例创建一个名为 exfunc_sum 的外部函数，它将两个整数作为输入参数。此函数以整数输出形式返回总和。要调用的 Lambda 函数的名称为 lambda_sum。用于此 Lambda 函数的语言是 Node.js 12.x。确保指定 IAM 角色。此示例使用 'arn:aws:iam::123456789012:user/johndoe' 作为 IAM 角色。

```
CREATE EXTERNAL FUNCTION exfunc_sum(INT,INT)
RETURNS INT
VOLATILE
LAMBDA 'lambda_sum'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test';
```

Lambda 函数接受请求负载并对每一行进行迭代。将单行中的所有值相加以计算该行的总和，并将其保存在响应数组中。结果数组中的行数与请求负载中接收的行数相似。

JSON 响应负载必须在“results”字段中包含结果数据，才能被外部函数识别。发送到 Lambda 函数的请求中的参数字段包含数据负载。在批处理请求的情况下，数据负载中可能有多行。以下 Lambda 函数对请求数据负载中的所有行进行迭代。它还分别对一行中的所有值进行迭代。

```
exports.handler = async (event) => {
    // The 'arguments' field in the request sent to the Lambda function contains the
 data payload.
    var t1 = event['arguments'];

    // 'len(t1)' represents the number of rows in the request payload.
    // The number of results in the response payload should be the same as the number
 of rows received.
    const resp = new Array(t1.length);
```

```
    // Iterating over all the rows in the request payload.
    for (const [i, x] of t1.entries())
    {
        var sum = 0;
        // Iterating over all the values in a single row.
        for (const y of x) {
            sum = sum + y;
        }
        resp[i] = sum;
    }
    // The 'results' field should contain the results of the lambda call.
    const response = {
        results: resp
    };
    return JSON.stringify(response);
};
```

以下示例使用文本值调用外部函数。

```
select exfunc_sum(1,2);
exfunc_sum
------------
 3
(1 row)
```

以下示例创建一个名为 t_sum 的表（其中包含整数数据类型的两个列 c1 和 c2）并插入两行数据。然后通过传递此表的列名称来调用外部函数。这两个表行作为单个 Lambda 调用在请求负载中的批处理请求中发送。

```
CREATE TABLE t_sum(c1 int, c2 int);
INSERT INTO t_sum VALUES (4,5), (6,7);
SELECT exfunc_sum(c1,c2) FROM t_sum;
 exfunc_sum
---------------
 9
 13
(2 rows)
```

使用 RETRY_TIMEOUT 属性的标量 Lambda UDF 示例

在以下部分中，您可以找到如何在 Lambda UDF 中使用 RETRY_TIMEOUT 属性的示例。

AWS Lambda 函数具有并发限制，您可以为每个函数设置这些限制。有关并发限制的更多信息，请参阅《AWS Lambda 开发人员指南》中的为 Lambda 函数管理并发和 AWS 计算博客上的博客文章 Managing AWS Lambda Function Concurrency。

当 Lambda UDF 服务的请求数超过并发限制时，新请求将接收 TooManyRequestsException 错误。Lambda UDF 会重试此错误，直到发送到 Lambda 函数的请求之间的所有延迟总和等于或超过您设置的 RETRY_TIMEOUT 值。默认的 RETRY_TIMEOUT 值为 20000 毫秒。

下面的示例使用一个名为 exfunc_sleep_3 的 Lambda 函数。此函数接受请求负载，对每一行进行迭代，并将输入转换为大写。然后它会睡眠 3 秒钟并返回结果。此 Lambda 函数使用的语言是 Python 3.8。

结果数组中的行数与请求负载中接收的行数相似。JSON 响应负载必须在 results 字段中包含结果数据，才能被外部函数识别。发送到 Lambda 函数的请求中的 arguments 字段包含数据负载。在批处理请求的情况下，数据负载中可能有多行。

在保留并发中，此函数的并发限制被专门设置为 1，以演示 RETRY_TIMEOUT 属性的使用情况。当属性设置为 1 时，Lambda 函数一次只能处理一个请求。

```
import json
import time
def lambda_handler(event, context):
    t1 = event['arguments']
    # 'len(t1)' represents the number of rows in the request payload.
    # The number of results in the response payload should be the same as the number of
 rows received.
    resp = [None]*len(t1)

    # Iterating over all rows in the request payload.
    for i, x in enumerate(t1):
        # Iterating over all the values in a single row.
        for j, y in enumerate(x):
            resp[i] = y.upper()

    time.sleep(3)
    ret = dict()
    ret['results'] = resp
    ret_json = json.dumps(ret)
    return ret_json
```

以下两个附加示例对 RETRY_TIMEOUT 属性进行了说明。它们各自调用了一个 Lambda UDF。在调用 Lambda UDF 时，每个示例都运行相同的 SQL 查询，以便同时从两个并发数据库会话

调用 Lambda UDF。当第一个调用 Lambda UDF 的查询由 UDF 提供时，第二个查询会收到 TooManyRequestsException 错误。出现此结果是因为您将 UDF 中的保留并发专门设置为 1。有关如何为 Lambda 函数设置保留并发的信息，请参阅配置预留并发。

下面的第一个示例将 Lambda UDF 的 RETRY_TIMEOUT 属性设置为 0 毫秒。如果 Lambda 请求收到来自 Lambda 函数的任何异常，则 Amazon Redshift 不会进行任何重试。出现此结果的原因是 RETRY_TIMEOUT 属性被设置为 0。

```
CREATE OR REPLACE EXTERNAL FUNCTION exfunc_upper(varchar)
RETURNS varchar
VOLATILE
LAMBDA 'exfunc_sleep_3'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test'
RETRY_TIMEOUT 0;
```

将 RETRY_TIMEOUT 设置为 0 时，您可以从单独的数据库会话运行以下两个查询以查看不同的结果。

使用 Lambda UDF 的第一个 SQL 查询成功运行。

```
select exfunc_upper('Varchar');
 exfunc_upper
 --------------
 VARCHAR
(1 row)
```

同时从单独的数据库会话运行的第二个查询将收到 TooManyRequestsException 错误。

```
select exfunc_upper('Varchar');
ERROR:  Rate Exceeded.; Exception: TooManyRequestsException; ShouldRetry: 1
DETAIL:
-----------------------------------------------
error:  Rate Exceeded.; Exception: TooManyRequestsException; ShouldRetry: 1
code:      32103
context:query:     0
location:  exfunc_client.cpp:102
process:   padbmaster [pid=26384]
-----------------------------------------------
```

下面的第二个示例将 Lambda UDF 的 RETRY_TIMEOUT 属性设置为 3000 毫秒。即使同时运行第二个查询，Lambda UDF 也会重试，直到总延迟为 3000 毫秒。因此，两个查询都成功运行。

```
CREATE OR REPLACE EXTERNAL FUNCTION exfunc_upper(varchar)
RETURNS varchar
VOLATILE
LAMBDA 'exfunc_sleep_3'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test'
RETRY_TIMEOUT 3000;
```

将 RETRY_TIMEOUT 被设置为 3000 时，您可以从单独的数据库会话运行以下两个查询以查看相同的结果。

运行 Lambda UDF 的第一个 SQL 查询成功运行。

```
select exfunc_upper('Varchar');
 exfunc_upper
 --------------
 VARCHAR
(1 row)
```

第二个查询同时运行，且 Lambda UDF 重试，直到总延迟为 3000 毫秒。

```
select exfunc_upper('Varchar');
 exfunc_upper
--------------
 VARCHAR
(1 row)
```

使用 Python Lambda 函数的标量 Lambda UDF 示例

以下示例创建名为 exfunc_multiplication 的外部函数，然后将数字相乘并返回整数。此示例合并了 Lambda 响应中的 success 和 error_msg 字段。当乘法结果中存在整数溢出时，success 字段设置为 false，且 error_msg 消息设置为 Integer multiplication overflow。exfunc_multiplication 函数将三个整数作为输入参数，并将总和作为整数输出返回。

被调用的 Lambda 函数的名称为 lambda_multiplication。此 Lambda 函数使用的语言是 Python 3.8。确保指定 IAM 角色。

```
CREATE EXTERNAL FUNCTION exfunc_multiplication(int, int, int)
RETURNS INT
```

```
VOLATILE
LAMBDA 'lambda_multiplication'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test';
```

Lambda 函数接受请求负载并对每一行进行迭代。将单行中的所有值相乘以计算该行的结果，并将其保存在响应列表中。此示例使用默认设置为 true 的 Boolean success 值。如果行的乘法结果具有整数溢出，则 success 值设置为 false。然后，迭代循环中断。

创建响应负载时，如果 success 值为 false，则以下 Lambda 函数将 `error_msg` 字段添加到负载中。它还将错误消息设置为 `Integer multiplication overflow`。如果 success 值为 true，则结果数据将添加到结果字段中。结果数组（如果有）中的行数与请求负载中接收的行数相似。

发送到 Lambda 函数的请求中的参数字段包含数据负载。在批处理请求的情况下，数据负载中可能有多行。以下 Lambda 函数对请求数据负载中的所有行进行迭代，并分别对一行中的所有值进行迭代。

```
import json
def lambda_handler(event, context):
    t1 = event['arguments']
    # 'len(t1)' represents the number of rows in the request payload.
    # The number of results in the response payload should be the same as the number of
 rows received.
    resp = [None]*len(t1)

    # By default success is set to 'True'.
    success = True
    # Iterating over all rows in the request payload.
    for i, x in enumerate(t1):
        mul = 1
        # Iterating over all the values in a single row.
        for j, y in enumerate(x):
            mul = mul*y

        # Check integer overflow.
        if (mul >= 9223372036854775807 or mul <= -9223372036854775808):
            success = False
            break
        else:
            resp[i] = mul
    ret = dict()
    ret['success'] = success
    if not success:
        ret['error_msg'] = "Integer multiplication overflow"
```

```
    else:
        ret['results'] = resp
    ret_json = json.dumps(ret)

    return ret_json
```

以下示例使用文本值调用外部函数。

```
SELECT exfunc_multiplication(8, 9, 2);
  exfunc_multiplication
--------------------------
          144
(1 row)
```

以下示例创建一个名为 t_multi 的表，其中包含整数数据类型的三个列 c1、c2 和 c3。通过传递此表的列名称来调用外部函数。以导致整数溢出的方式插入数据，从而显示错误的传播方式。

```
CREATE TABLE t_multi (c1 int, c2 int, c3 int);
INSERT INTO t_multi VALUES (2147483647, 2147483647, 4);
SELECT exfunc_multiplication(c1, c2, c3) FROM t_multi;
DETAIL:
  -----------------------------------------------
  error:  Integer multiplication overflow
  code:      32004context:
  context:
  query:     38
  location:  exfunc_data.cpp:276
  process:   query2_16_38 [pid=30494]
  -----------------------------------------------
```

CREATE EXTERNAL SCHEMA

在当前数据库中创建一个新外部 schema。您可以使用此外部 schema 连接到 Amazon RDS for PostgreSQL 或 Amazon Aurora PostgreSQL 兼容版本数据库。您还可以创建引用外部数据目录（如 AWS Glue、Athena）中的数据库或 Apache Hive 元存储（如 Amazon EMR）中的数据库的外部 schema。

此 schema 的所有者为 CREATE EXTERNAL SCHEMA 命令的发布者。要移交外部 schema 的所有权，请使用 ALTER SCHEMA 更改所有者。要为其他用户或用户组授予架构的访问权限，请使用 GRANT 命令。

您无法针对外部表的权限使用 GRANT 或 REVOKE 命令。相反，您可以授予或撤销对外部 schema 的权限。

> **Note**
>
> 如果您当前在 Amazon Athena 数据目录中有 Redshift Spectrum 外部表，则可以将您的 Athena 数据目录迁移到 AWS Glue Data Catalog。要将 AWS Glue Data Catalog 用于 Redshift Spectrum，您可能需要更改您的 AWS Identity and Access Management (IAM) 策略。有关更多信息，请参阅《Athena 用户指南》中的升级到 AWS Glue Data Catalog。

要查看外部 schema 的详细信息，请查询 SVV_EXTERNAL_SCHEMAS 系统视图。

语法

以下语法描述了用于使用外部数据目录引用数据的 CREATE EXTERNAL SCHEMA 命令。有关更多信息，请参阅 使用 Amazon Redshift Spectrum 查询外部数据。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM { [ DATA CATALOG ] | HIVE METASTORE | POSTGRES | MYSQL | KINESIS | MSK |
 REDSHIFT }
[ DATABASE 'database_name' ]
[ SCHEMA 'schema_name' ]
[ REGION 'aws-region' ]
[ URI 'hive_metastore_uri' [ PORT port_number ] ]
IAM_ROLE { default | 'SESSION' | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' }
[ SECRET_ARN 'ssm-secret-arn' ]
[ AUTHENTICATION { none | iam } ]
[ CLUSTER_ARN 'arn:aws:kafka:<region>:<AWS ##-id>:cluster/msk/<cluster uuid>' ]
[ CATALOG_ROLE { 'SESSION' | 'catalog-role-arn-string' } ]
[ CREATE EXTERNAL DATABASE IF NOT EXISTS ]
[ CATALOG_ID 'Amazon Web Services account ID containing Glue or Lake Formation
 database' ]
```

以下语法描述了用于使用至 RDS POSTGRES 或 Aurora PostgreSQL 的联合查询引用数据的 CREATE EXTERNAL SCHEMA 命令。您还可以创建引用串流源的外部 Schema，例如 Kinesis Data Streams。有关更多信息，请参阅 在 Amazon Redshift 中使用联合查询来查询数据。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM POSTGRES
```

```
DATABASE 'federated_database_name' [SCHEMA 'schema_name']
URI 'hostname' [ PORT port_number ]
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' }
SECRET_ARN 'ssm-secret-arn'
```

以下语法描述了用于使用至 RDS MySQL 或 Aurora MySQL 的联合查询引用数据的 CREATE EXTERNAL SCHEMA 命令。有关更多信息，请参阅 在 Amazon Redshift 中使用联合查询来查询数据。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM MYSQL
DATABASE 'federated_database_name'
URI 'hostname' [ PORT port_number ]
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' }
SECRET_ARN 'ssm-secret-arn'
```

以下语法描述了用于引用 Kinesis 流中数据的 CREATE EXTERNAL SCHEMA 命令。有关更多信息，请参阅 串流摄取。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] schema_name
FROM KINESIS
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' }
```

以下语法描述了 CREATE EXTERNAL SCHEMA 命令，该命令用于引用 Amazon Managed Streaming for Apache Kafka 集群及其要从中摄取的主题。CLUSTER_ARN 指定了您正在从中读取数据的 Amazon MSK 集群。有关更多信息，请参阅 串流摄取。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] schema_name
FROM MSK
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>' }
AUTHENTICATION { none | iam }
CLUSTER_ARN 'msk-cluster-arn';
```

以下语法描述了用于使用跨 数据库查询引用数据的 CREATE EXTERNAL SCHEMA 命令。

```
CREATE EXTERNAL SCHEMA local_schema_name
FROM  REDSHIFT
DATABASE 'redshift_database_name' SCHEMA 'redshift_schema_name'
```

参数

IF NOT EXISTS

一个子句，指示如果指定 schema 已存在，则此命令不应进行任何更改，并应返回一条指示 schema 存在的消息，而不是以错误终止。此子句在编写脚本时很有用，可使脚本在 CREATE EXTERNAL SCHEMA 尝试创建已存在的 schema 时不会失败。

local_schema_name

新外部 schema 的名称。有关有效名称的更多信息，请参阅名称和标识符。

FROM [DATA CATALOG] | HIVE METASTORE | POSTGRES | MYSQL | KINESIS | MSK | REDSHIFT

指示外部数据库所在位置的关键词。

DATA CATALOG 指示外部数据库是在 Athena 数据目录或 AWS Glue Data Catalog 中定义的。

如果外部数据库是在位于其他 AWS 区域的外部 Data Catalog 中定义的，则 REGION 参数为必填项。DATA CATALOG 是默认值。

HIVE METASTORE 指示外部数据库是在 Apache Hive 元存储中定义的。如果指定了 HIVE METASTORE，则 URI 为必填项。

POSTGRES 指示外部数据库是在 RDS PostgreSQL 或 Aurora PostgreSQL 中定义的。

MYSQL 表示外部数据库是在 RDS MySQL 或 Aurora MySQL 中定义的。

KINESIS 指示数据来源是一个来自 Kinesis Data Streams 的流。

MSK 表示数据来源是来自 Amazon MSK 的主题。

FROM REDSHIFT

指示数据库位于 Amazon Redshift 中的关键词。

DATABASE“redshift_database_name”SCHEMA“redshift_schema_name”

Amazon Redshift 数据库的名称。

redshift_schema_name 指示 Amazon Redshift 中的 schema。默认的 redshift_schema_name 为 `public`。

数据库“federated_database_name”

在支持的 PostgreSQL or MySQL 数据库引擎中指示外部数据库名称的关键词。

[SCHEMA 'schema_name']

schema_name 指示支持的 PostgreSQL 数据库引擎中的 schema。默认的 schema_name 是 `public`。

在设置对受支持的 MySQL 数据库引擎的联合查询时，无法指定 SCHEMA。

REGION 'aws-region'

如果外部数据库是在 Athena 数据目录或 AWS Glue Data Catalog 中定义的，则为数据库所在的 AWS 区域。如果数据库是在外部 Data Catalog 中定义的，则此参数为必填项。

URI 'hive_metastore_uri' [PORT port_number]

支持的 PostgreSQL 或 MySQL 数据库引擎的主机名 URI 和 port_number。hostname 是副本集的头节点。端点必须可以从 Amazon Redshift 集群进行访问（路由）。默认的 PostgreSQL port_number 为 5432。默认的 MySQL port_number 为 3306。

如果数据库位于 Hive 元存储中，请指定 URI 并选择性地指定元存储的端口号。默认端口号为 9083。

URI 不包含协议规范（“http://”）。有效 URI 示例：`uri '172.10.10.10'`。

> **Note**
>
> 支持的 PostgreSQL 或 MySQL 数据库引擎必须位于与 Amazon Redshift 集群相同的 VPC 中。创建一个安全组，链接 Amazon Redshift 和 RDS PostgreSQL 或 Aurora PostgreSQL。

IAM_ROLE { default | 'SESSION' | 'arn:aws:iam::*<AWS ##-id>*:role/*<role-name>*' }

使用默认关键字让 Amazon Redshift 使用设置为默认值并在 CREATE EXTERNAL SCHEMA 命令运行时与集群关联的 IAM 角色。

如果您使用联合身份连接到 Amazon Redshift 集群并访问使用此命令创建的外部架构中的表，则使用 `'SESSION'`。有关更多信息，请参阅使用联合身份管理 Amazon Redshift 对本地资源和 Amazon Redshift Spectrum 外部表的访问权限，其中说明了如何配置联合身份。请注意，此配置（使用 `'SESSION'` 替代 ARN）仅在使用 `DATA CATALOG` 创建架构时才能使用。

使用 IAM 角色的 Amazon 资源名称（ARN），您的集群使用该角色进行身份验证和授权。IAM 角色至少必须有权在要访问的 Amazon S3 桶上执行 LIST 操作和有权在该桶包含的 Amazon S3 对象上执行 GET 操作。

下面显示了单个 ARN 的 IAM_ROLE 参数字符串的语法。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

您可以将角色串联起来，以便集群可以承担另一个 IAM 角色 (可能属于其他账户)。您最多可串联 10 个角色。有关串联角色的示例，请参阅在 Amazon Redshift Spectrum 中链接 IAM 角色。

对于此 IAM 角色，请附加类似于以下内容的 IAM 权限策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

有关创建 IAM 角色以用于联合查询的步骤，请参阅创建密钥和 IAM 角色以使用联合查询。

> **Note**
>
> 请不要在链接的角色列表中包含空格。

下面显示了串联三个角色的语法。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

SECRET_ARN 'ssm-secret-arn'

使用 AWS Secrets Manager 创建的支持的 PostgreSQL 或 MySQL 数据库引擎密钥的 Amazon 资源名称（ARN）。有关如何为密钥创建和检索 ARN 的信息，请参阅《AWS Secrets Manager 用户指南》中的创建基本密钥和检索密值密钥。

CATALOG_ROLE { 'SESSION' | catalog-role-arn-string}

利用 `'SESSION'`，通过使用用于对数据目录进行身份验证和授权的联合身份来连接到 Amazon Redshift 集群。有关完成联合身份的步骤的更多信息，请参阅使用联合身份管理 Amazon Redshift 对本地资源和 Amazon Redshift Spectrum 外部表的访问权限。请注意，仅在 DATA CATALOG 中创建架构时才能使用 `'SESSION'` 角色。

使用 IAM 角色的 Amazon 资源名称（ARN），您的集群使用该角色对数据目录进行身份验证和授权。

如果未指定 CATALOG_ROLE，Amazon Redshift 会使用指定的 IAM_ROLE。目录角色必须有权限访问 AWS Glue 或 Athena 中的 Data Catalog。有关更多信息，请参阅 适用于 Amazon Redshift Spectrum 的 IAM 策略。

下面显示了单个 ARN 的 CATALOG_ROLE 参数字符串的语法。

```
CATALOG_ROLE 'arn:aws:iam::<aws-account-id>:role/<catalog-role>'
```

您可以将角色串联起来，以便集群可以承担另一个 IAM 角色 (可能属于其他账户)。您最多可串联 10 个角色。有关更多信息，请参阅 在 Amazon Redshift Spectrum 中链接 IAM 角色。

> **Note**
>
> 串联角色的列表不能包含空格。

下面显示了串联三个角色的语法。

```
CATALOG_ROLE 'arn:aws:iam::<aws-account-id>:role/<catalog-role-1-
name>,arn:aws:iam::<aws-account-id>:role/<catalog-role-2-name>,arn:aws:iam::<aws-
account-id>:role/<catalog-role-3-name>'
```

CREATE EXTERNAL DATABASE IF NOT EXISTS

一个子句，用于在指定的外部数据库不存在时使用由 DATABASE 参数指定的名称创建外部数据库。如果指定的外部数据库存在，该命令不会进行任何更改。在这种情况下，该命令将返回指示外部数据库存在的消息，而不是以错误终止。

> **Note**
>
> CREATE EXTERNAL DATABASE IF NOT EXISTS 不能与 HIVE METASTORE 一起使用。
> 要将 CREATE EXTERNAL DATABASE IF NOT EXISTS 与为 AWS Lake Formation 启用的 Data Catalog 搭配使用，需要获得对于 Data Catalog 的 CREATE_DATABASE 权限。

CATALOG_ID 'Amazon Web Services 账户 ID，包含 Glue 或 Lake Formation 数据库'

用于存储数据目录数据库的账户 ID。

只有在您计划使用联合身份（用于对数据目录进行身份验证和授权）连接到 Amazon Redshift 集群或 Amazon Redshift Serverless 时，才能通过设置以下任一项来指定 CATALOG_ID：

- CATALOG_ROLE 到 'SESSION'
- 将 IAM_ROLE 设置为 'SESSION'，并将 'CATALOG_ROLE' 设置为默认值

有关完成联合身份的步骤的更多信息，请参阅使用联合身份管理 Amazon Redshift 对本地资源和 Amazon Redshift Spectrum 外部表的访问权限。

AUTHENTICATION

为串流摄取定义的身份验证类型。具有身份验证类型的串流摄取使用 Amazon Managed Streaming for Apache Kafka。AUTHENTICATION 类型为：

- 无 – 指定没有身份验证步骤。
- iam – 指定 IAM 身份验证。选择此选项时，请确保 IAM 角色具有 IAM 身份验证的权限。有关定义外部模式的更多信息，请参阅开始使用 Amazon Managed Streaming for Apache Kafka 串流摄取。

CLUSTER_ARN

对于串流摄取，指您从中进行流式传输的 Amazon Managed Streaming for Apache Kafka 集群的集群标识符。有关更多信息，请参阅串流摄取。

使用说明

有关使用 Athena 数据目录时的限制，请参阅《AWS 一般参考》中的 Athena 限制。

有关使用 AWS Glue Data Catalog 时的限制，请参阅《AWS 一般参考》中的 AWS Glue 限制。

这些限制不适用于 Hive 元存储。

每个数据库最多有 9900 个架构。有关更多信息，请参阅《Amazon Redshift 管理指南》中的配额和限制。

要取消注册架构，请使用 DROP SCHEMA 命令。

要查看外部架构的详细信息，请查询以下系统视图：

- SVV_EXTERNAL_SCHEMAS
- SVV_EXTERNAL_TABLES
- SVV_EXTERNAL_COLUMNS

示例

以下示例使用位于美国西部（俄勒冈州）区域的名为 sampledb 的数据目录中的数据库，创建一个外部模式。将此示例与 Athena 或 AWS Glue 数据目录结合使用。

```
create external schema spectrum_schema
from data catalog
database 'sampledb'
region 'us-west-2'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole';
```

以下示例创建一个外部 schema 并新建一个名为 spectrum_db 的新外部数据库。

```
create external schema spectrum_schema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
```

```
create external database if not exists;
```

以下示例创建一个使用名为 hive_db 的 Hive 元存储数据库的外部 schema。

```
create external schema hive_schema
from hive metastore
database 'hive_db'
uri '172.10.10.10' port 99
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole';
```

以下示例将角色串联起来，以使用角色 myS3Role 来访问 Amazon S3 ，并且使用 myAthenaRole 进行数据目录访问。有关更多信息，请参阅 在 Amazon Redshift Spectrum 中链接 IAM 角色。

```
create external schema spectrum_schema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myRedshiftRole,arn:aws:iam::123456789012:role/
myS3Role'
catalog_role 'arn:aws:iam::123456789012:role/myAthenaRole'
create external database if not exists;
```

以下示例创建引用 Aurora PostgreSQL 数据库的外部 schema。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] myRedshiftSchema
FROM POSTGRES
DATABASE 'my_aurora_db' SCHEMA 'my_aurora_schema'
URI 'endpoint to aurora hostname' PORT 5432
IAM_ROLE 'arn:aws:iam::123456789012:role/MyAuroraRole'
SECRET_ARN 'arn:aws:secretsmanager:us-east-2:123456789012:secret:development/
MyTestDatabase-AbCdEf'
```

以下示例创建一个外部架构来引用导入到使用者集群上的 sales_db。

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA 'public';
```

以下示例创建引用 Aurora MySQL 数据库的外部 schema。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] myRedshiftSchema
FROM MYSQL
DATABASE 'my_aurora_db'
URI 'endpoint to aurora hostname'
```

```
IAM_ROLE 'arn:aws:iam::123456789012:role/MyAuroraRole'
SECRET_ARN 'arn:aws:secretsmanager:us-east-2:123456789012:secret:development/
MyTestDatabase-AbCdEf'
```

CREATE EXTERNAL TABLE

在指定 schema 中创建一个新外部表。所有外部表必须在外部 schema 中创建。外部 schema 和外部表不支持搜索路径。有关更多信息，请参阅 CREATE EXTERNAL SCHEMA。

除了使用 CREATE EXTERNAL TABLE 命令创建的外部表之外，Amazon Redshift 还可引用在 AWS Glue 或 AWS Lake Formation 目录或 Apache Hive 元存储中定义的外部表。使用 CREATE EXTERNAL SCHEMA 命令可注册在外部目录中定义的外部数据库并使外部表可在 Amazon Redshift 中使用。如果外部表已存在于 AWS Glue 或 AWS Lake Formation 目录或 Hive 元存储中，您无需使用 CREATE EXTERNAL TABLE 创建该表。要查看外部表，请查询 SVV_EXTERNAL_TABLES 系统视图。

通过运行 CREATE EXTERNAL TABLE AS 命令，可以根据查询中的列定义创建外部表，并将该查询的结果写入 Amazon S3 中。结果采用 Apache Parquet 或分隔的文本格式。如果外部表具有一个或多个分区键，Amazon Redshift 会根据这些分区键对新文件进行分区，并自动将新分区注册到外部目录中。有关 CREATE EXTERNAL TABLE AS 的更多信息，请参阅使用说明。

您可使用用于其他 Amazon Redshift 表的同一 SELECT 语法查询外部表。您还可以使用 INSERT 语法将新文件写入 Amazon S3 上外部表的位置。有关更多信息，请参阅 INSERT（外部表）。

要使用外部表创建视图，请在 CREATE VIEW 语句中包含 WITH NO SCHEMA BINDING 子句。

您无法在事务 (BEGIN … END) 内运行 CREATE EXTERNAL TABLE。有关事务的更多信息，请参阅可序列化的隔离。

所需的权限

要创建外部表，您必须是外部 schema 的所有者或是超级用户。要移交外部 schema 的所有权，请使用 ALTER SCHEMA 更改所有者。对外部表的访问由对外部 schema 的访问权限控制。您无法对外部表的权限执行 GRANT 或 REVOKE 操作。但是，您可授予或撤销对外部 schema 的 USAGE 权限。

使用说明 包含有关外部表特定权限的更多信息。

语法

```
CREATE EXTERNAL TABLE
external_schema.table_name
```

```
(column_name data_type [, …] )
[ PARTITIONED BY (col_name data_type [, … ] )]
[ { ROW FORMAT DELIMITED row_format |
  ROW FORMAT SERDE 'serde_name'
  [ WITH SERDEPROPERTIES ( 'property_name' = 'property_value' [, ...] ) ] } ]
STORED AS file_format
LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }
[ TABLE PROPERTIES ( 'property_name'='property_value' [, ...] ) ]
```

以下是 CREATE EXTERNAL TABLE AS 的语法。

```
CREATE EXTERNAL TABLE
external_schema.table_name
[ PARTITIONED BY (col_name [, … ] ) ]
[ ROW FORMAT DELIMITED row_format ]
STORED AS file_format
LOCATION { 's3://bucket/folder/' }
[ TABLE PROPERTIES ( 'property_name'='property_value' [, ...] ) ]
 AS
 { select_statement }
```

参数

external_schema.table_name

要创建的表的名称（由外部 schema 名称进行限定）。外部表必须在外部 schema 中创建。有关更多信息，请参阅 CREATE EXTERNAL SCHEMA。

表名称的最大长度为 127 个字节；更长的名称将被截断为 127 个字节。您可以使用 UTF-8 多字节字符，每个字符最多为四个字节。Amazon Redshift 对每个集群强制实施 9900 个表的限制，包括用户定义的临时表以及查询处理或系统维护期间由 Amazon Redshift 创建的临时表。您也可以使用数据库名称限定表名称。在下面的示例中，spectrum_db 是数据库名称，spectrum_schema 是外部 schema 名称，而 test 是表名称。

```
create external table spectrum_db.spectrum_schema.test (c1 int)
stored as parquet
location 's3://mybucket/myfolder/';
```

如果指定数据库或 schema 不存在，则不会创建表，并且语句将返回错误。您无法在系统数据库 template0、template1、padb_harvest 或 sys:internal 中创建表或视图。

表名称对于指定 schema 必须是唯一名称。

有关有效名称的更多信息，请参阅名称和标识符。

(column_name data_type)

要创建的每个列的名称和数据类型。

列名称的最大长度为 127 个字节；更长的名称将被截断为 127 个字节。您可以使用 UTF-8 多字节字符，每个字符最多为四个字节。不能指定列名称 "$path" 或 "$size"。有关有效名称的更多信息，请参阅名称和标识符。

预设情况下，Amazon Redshift 使用伪列 $path 和 $size 创建外部表。您可以通过将 spectrum_enable_pseudo_columns 配置参数设置为 false 来禁用为会话创建 pseudocolumns 的功能。有关更多信息，请参阅 Pseudocolumns 。

如果已启用 pseudocolumns，则可在单个表中定义的最大列数为 1598。如果未启用 pseudocolumns，则可在单个表中定义的最大列数为 1600。

如果您创建的是“宽表”，请确保在加载和查询处理期间，不要让列列表超出中间结果的行宽度边界。有关更多信息，请参阅 使用说明。

对于 CREATE EXTERNAL TABLE AS 命令，不需要列的列表，因为列是从查询派生的。

data_type

支持以下数据类型：

- SMALLINT (INT2)
- INTEGER (INT, INT4)
- BIGINT (INT8)
- DECIMAL (NUMERIC)
- REAL (FLOAT4)
- DOUBLE PRECISION (FLOAT8)
- BOOLEAN (BOOL)
- CHAR (CHARACTER)
- VARCHAR (CHARACTER VARYING)
- VARBYTE (CHARACTER VARYING) – 可与 Parquet 和 ORC 数据文件一起使用，并且只能用于非分区表。
- DATE – 只能与文本、Parquet 或 ORC 数据文件一起使用，或者用作分区列。

- TIMESTAMP

对于 DATE，您可以使用以下所示的格式。对于使用数字表示的月份值，支持以下格式：

- mm-dd-yyyy - 例如：05-01-2017这是默认模式。
- yyyy-mm-dd，其中年份由 2 位以上的数字表示。例如：2017-05-01。

对于使用三个字母缩写表示的月份值，则支持以下格式：

- mmm-dd-yyyy - 例如：may-01-2017这是默认模式。
- dd-mmm-yyyy，其中年份由 2 位以上的数字表示。例如：01-may-2017。
- yyyy-mmm-dd，其中年份由 2 位以上的数字表示。例如：2017-may-01。

对于始终小于 100 的年份值，请按以下方式计算年份：

- 如果年份值小于 70，则在计算年份时加上 2000。例如，以 mm-dd-yyyy 格式表示的日期 05-01-17 将被转换为 05-01-2017。
- 如果年份值小于 100 但大于 69，则在计算年份时加上 1900。例如，以 mm-dd-yyyy 格式表示的日期 05-01-89 将被转换为 05-01-1989。
- 对于以两位数表示的年份值，请添加前导零，以 4 位数表示年份。

文本文件中的时间戳值的格式必须为 yyyy-mm-dd HH:mm:ss.SSSSSS，如以下时间戳值所示：2017-05-01 11:30:59.000000。

VARCHAR 列的长度的定义单位是字节而不是字符。例如，VARCHAR(12) 列可包含 12 个单字节字符或 6 个双字节字符。查询外部表时，将截断结果以适合定义的列大小，而不返回错误。有关更多信息，请参阅 存储和范围。

为获得最佳性能，我们建议您指定适合您数据的最小列大小。要查找列中值的最大大小（以字节为单位），请使用 OCTET_LENGTH 函数。以下示例返回电子邮件列中值的大小上限。

```
select max(octet_length(email)) from users;

max
---
 62
```

PARTITIONED BY (col_name data_type [, …])

用于定义包含一个或多个分区列的已分区表的子句。单独的数据目录用于每个指定的组合，这在某些情况下可提高查询性能。已分区列在表数据本身中不存在。如果您将与表列相同的某个值用于 col_name，则会产生错误。

创建分区表后，使用 ALTER TABLE … ADD PARTITION 语句更改表，以将新分区注册到外部目录。在添加分区时，您应定义包含分区数据的子文件夹在 Amazon S3 上的位置。

例如，如果表 spectrum.lineitem_part 是使用 PARTITIONED BY (l_shipdate date) 定义的，请运行以下 ALTER TABLE 命令来添加分区。

```
ALTER TABLE spectrum.lineitem_part ADD PARTITION (l_shipdate='1992-01-29')
LOCATION 's3://spectrum-public/lineitem_partition/l_shipdate=1992-01-29';
```

如果您使用 CREATE EXTERNAL TABLE AS，则不需要运行 ALTER TABLE...ADD PARTITION。Amazon Redshift 会自动在外部目录中注册新分区。Amazon Redshift 还会根据表中定义的一个或多个分区键自动将相应的数据写入 Amazon S3 中的分区。

要查看分区，请查询 SVV_EXTERNAL_PARTITIONS 系统视图。

> **Note**
>
> 对于 CREATE EXTERNAL TABLE AS 命令，您不需要指定分区列的数据类型，因为此列是从查询派生的。

ROW FORMAT DELIMITED rowformat

用于指定基础数据的格式的子句。rowformat 的可能值如下所示：

- LINES TERMINATED BY 'delimiter'
- FIELDS TERMINATED BY 'delimiter'

为 'delimiter' 指定一个 ASCII 字符。您可以指定八进制格式的非打印 ASCII 字符，具体格式为 '*ddd*'，其中 *d* 是一个八进制数 (0–7)，最大为“\177”。以下示例使用八进制形式指定 BEL（响铃）字符。

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\007'
```

如果省略了 ROW FORMAT，则默认格式为 DELIMITED FIELDS TERMINATED BY '\A'（标题开头）和 LINES TERMINATED BY '\n'（换行符）。

ROW FORMAT SERDE 'serde_name', [WITH SERDEPROPERTIES ('property_name' = 'property_value' [, ...])]

用于为基础数据指定 SERDE 格式的子句。

'serde_name'

SerDe 的名称。您可以指定以下格式：

- org.apache.hadoop.hive.serde2.RegexSerDe
- com.amazonaws.glue.serde.GrokSerDe
- org.apache.hadoop.hive.serde2.OpenCSVSerde

  此参数支持 OpenCSVSerde 的以下 SerDe 属性：

  ```
  'wholeFile' = 'true'
  ```

  将 `wholeFile` 属性设置为 `true`，以正确解析 OpenCSV 请求的引号字符串中的新行字符 (\n)。
- org.openx.data.jsonserde.JsonSerDe
  - JSON SERDE 还支持 Ion 文件。
  - JSON 必须格式正确。
  - Ion 和 JSON 中的时间戳必须使用 ISO8601 格式。
  - 该参数支持 JsonSerDe 的以下 SerDe 属性：

    ```
    'strip.outer.array'='true'
    ```

    处理 Ion/JSON 文件，其中包含一个包含在方括号 ([...]) 中的非常大的数组，就像它在数组中包含多个 JSON 记录一样。
- com.amazon.ionhiveserde.IonHiveSerDe

  除数据类型外，Amazon ION 格式还提供文本和二进制格式。对于引用 ION 格式数据的外部表，将外部表中的每个列映射到 ION 格式数据中的对应元素。有关更多信息，请参阅 Amazon Ion。此外，您还需要指定输入和输出格式。

WITH SERDEPROPERTIES ('property_name' = 'property_value' [, ...])]

（可选）指定以逗号分隔的属性名称和值。

如果省略了 ROW FORMAT，则默认格式为 DELIMITED FIELDS TERMINATED BY '\A'（标题开头）和 LINES TERMINATED BY '\n'（换行符）。

STORED AS file_format

数据文件的文件格式。

有效格式如下所示：

- PARQUET
- RCFILE (仅针对使用 ColumnarSerDe 而不是 LazyBinaryColumnarSerDe 的数据)
- SEQUENCEFILE
- TEXTFILE（针对文本文件，包括 JSON 文件）。
- ORC
- AVRO
- INPUTFORMAT 'input_format_classname' OUTPUTFORMAT 'output_format_classname'

CREATE EXTERNAL TABLE AS 命令只支持两种文件格式：TEXTFILE 和 PARQUET。

对于 INPUTFORMAT 和 OUTPUTFORMAT，指定类名称，如下例所示。

```
'org.apache.hadoop.mapred.TextInputFormat'
```

LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file'}

包含数据文件的 Amazon S3 桶或文件夹的路径或包含 Amazon S3 对象路径列表的清单文件。桶必须与 Amazon Redshift 集群位于同一 AWS 区域。有关受支持的 AWS 区域的列表，请参阅Amazon Redshift Spectrum 注意事项。

如果路径指定桶或文件夹，例如 `'s3://mybucket/custdata/'`，Redshift Spectrum 会扫描指定的桶或文件夹和任意子文件夹中的文件。Redshift Spectrum 将忽略隐藏文件和以句点或下划线开头的文件。

如果路径指定清单文件，则 `'s3://bucket/manifest_file'` 参数必须显式引用单个文件，例如 `'s3://mybucket/manifest.txt'`。它不能引用键前缀。

清单是 JSON 格式的文本文件，其中列出了要从 Amazon S3 加载的每个文件的 URL 以及文件的大小（以字节为单位）。URL 包含文件的桶名称和完整对象路径。在清单中指定的文件可以位于不同的桶中，但所有桶都必须位于 Amazon Redshift 集群所在的同一 AWS 区域。如果某个文件被列出两次，那么该文件也会被加载两次。以下示例显示了加载三个文件的清单的 JSON。

```
{
  "entries": [
    {"url":"s3://mybucket-alpha/custdata.1", "meta": { "content_length":
 5956875 } },
```

```
    {"url":"s3://mybucket-alpha/custdata.2", "meta": { "content_length":
 5997091 } },
    {"url":"s3://mybucket-beta/custdata.1", "meta": { "content_length": 5978675 } }
  ]
}
```

您可以强制包含特定文件。为此，请在清单中的文件级别包含一个 mandatory 选项。当您查询缺少强制性文件的外部表时，SELECT 语句将失败。确保外部表定义中包含的所有文件都存在。如果它们并非全部存在，则会显示一个错误，显示未找到的第一个强制性文件。以下示例显示 mandatory 选项设置为 true 的清单的 JSON。

```
{
  "entries": [
    {"url":"s3://mybucket-alpha/custdata.1", "mandatory":true, "meta":
 { "content_length": 5956875 } },
    {"url":"s3://mybucket-alpha/custdata.2", "mandatory":false, "meta":
 { "content_length": 5997091 } },
    {"url":"s3://mybucket-beta/custdata.1", "meta": { "content_length": 5978675 } }
  ]
}
```

要引用使用 UNLOAD 创建的文件，您可以使用通过 UNLOAD 和 MANIFEST 参数创建的清单。该清单文件与 从 Amazon S3 执行 COPY 操作 的清单文件兼容，但使用不同的密钥。不使用的密钥会被忽略。

TABLE PROPERTIES ('property_name'='property_value' [, ...])

用于设置表属性的表定义的子句。

> **Note**
>
> 表属性区分大小写。

'compression_type'='value'

一个属性，它在文件名不包含扩展名时设置要使用的压缩类型。如果您设置此属性且存在文件扩展名，则将忽略该扩展名，并使用由此属性设置的值。压缩类型的有效值如下：

- bzip2
- gzip

- 无
- snappy

'data_cleansing_enabled'='true / false’

该属性设置该表的数据处理是否已启用。当“data_cleansing_enabled”设置为 true 时，表的数据处理将启用。当“data_cleansing_enabled”设置为 false 时，表的数据处理将关闭。以下是由此属性控制的表级别数据处理属性的列表：

- column_count_mismatch_handling
- invalid_char_handling
- numeric_overflow_handling
- replacement_char
- surplus_char_handling

有关示例，请参阅数据处理示例。

'invalid_char_handling'='value'

指定当查询结果包含无效的 UTF-8 字符值时要执行的操作。您可以指定以下操作：

DISABLED

不执行无效字符处理。

FAIL

取消返回包含无效 UTF-8 值的数据的查询。

SET_TO_NULL

将无效 UTF-8 值替换为 null。

DROP_ROW

将行中的每个值替换为 null。

REPLACE

使用 `replacement_char`，将无效字符替换为您指定的替换字符。

'replacement_char'='character’

当您将 `invalid_char_handling` 设置为 `REPLACE` 时，请指定要使用的替换字符。

'numeric_overflow_handling'='value’

指定 ORC 数据包含大于列定义（例如，SMALLINT 或 int16）的整数（例如，BIGINT 或 int64）时要执行的操作。您可以指定以下操作：

DISABLED

关闭无效字符处理。

FAIL

当数据包含无效字符时取消查询。

SET_TO_NULL

将无效字符设置为 null。

DROP_ROW

将行中的每个值设置为 null。

'surplus_bytes_handling'='value'

指定如何处理加载的数据，其长度超过为包含 VARBYTE 数据的列所定义的数据类型的长度。默认情况下，对于超出列宽度的数据，Redshift Spectrum 会将该值设置为 null。

当查询返回超过数据类型长度的数据时，您可以指定以下要执行的操作：

SET_TO_NULL

将超过列宽度的数据替换为 null。

DISABLED

不执行超额字节处理。

FAIL

取消返回超过列宽度的数据的查询。

DROP_ROW

剔除包含超过列宽度的数据的所有行。

TRUNCATE

移除超出列定义的最大字符数的字符。

'surplus_char_handling'='value'

指定如何处理加载的数据，其长度超过包含 VARCHAR、CHAR 或字符串数据列所定义的数据类型长度。默认情况下，对于超出列宽度的数据，Redshift Spectrum 会将该值设置为 null。

当查询返回超过列宽的数据时，您可以指定以下要执行的操作：

SET_TO_NULL

将超过列宽度的数据替换为 null。

DISABLED

不执行超额字符处理。

FAIL

取消返回超过列宽度的数据的查询。

DROP_ROW

将行中的每个值替换为 null。

TRUNCATE

移除超出列定义的最大字符数的字符。

'column_count_mismatch_handling'='value’

确定文件包含的行值是否小于或大于外部表定义中指定的列数。此属性仅适用于未压缩的文本文件格式。您可以指定以下操作：

DISABLED

列计数不匹配处理处于关闭状态。

FAIL

如果检测到列计数不匹配，则查询失败。

SET_TO_NULL

使用 NULL 填充缺失值并忽略每行中的其他值。

DROP_ROW

剔除包含扫描中列计数不匹配错误的所有行。

'numRows'='row_count'

用于为表定义设置 numRows 值的属性。若要显式更新外部表的统计数据，请设置 numRows 属性来指示表的大小。Amazon Redshift 不分析外部表来生成表统计数据，查询优化程序会使用这些统计数据来生成查询计划。如果没有为外部表设置表统计数据，则 Amazon Redshift 假设外部表是较大的表，本地表是较小的表，以此来生成查询执行计划。

'skip.header.line.count'='line_count'

用于设置在每个源文件开头要跳过的行数的属性。

'serialization.null.format'=' '

一个属性，指定当存在与某个字段中提供的文本完全匹配的项时，Spectrum 应返回 NULL 值。

'orc.schema.resolution'='mapping_type'

一个属性，用于设置使用 ORC 数据格式的表的列映射类型。其他数据格式将忽略此属性。

列映射类型的有效值如下：

- 名称
- position

如果省略 orc.schema.resolution 属性，默认情况下会按名称映射列。如果将 orc.schema.resolution 设置为“name”或“position”之外的任何其他值，则按位置映射列。有关列映射的更多信息，请参阅将外部表列映射到 ORC 列。

> **Note**
>
> COPY 命令仅按位置映射到 ORC 数据文件。orc.schema.resolution 表属性对 COPY 命令行为无效。

'write.parallel'='on / off’

一个属性，用于设置是否 CREATE EXTERNAL TABLE AS 应并行写入数据。默认情况下，CREATE EXTERNAL TABLE AS 根据集群中的切片数量将数据并行写入到多个文件。默认选项为打开。当“write.parallel”设置为关闭时，CREATE EXTERNAL TABLE AS 以串行方式将一个或多个数据文件写入到 Amazon S3。该表属性还适用于指向同一外部表的所有后续 INSERT 语句。

'write.maxfilesize.mb'='size'

设置由 CREATE EXTERNAL TABLE AS 写入到 Amazon S3 中的每个文件的最大大小（以 MB 为单位）的属性。大小必须是介于 5 到 6200 之间的有效整数。默认最大文件大小为 6,200 MB。该表属性还适用于指向同一外部表的所有后续 INSERT 语句。

'write.kms.key.id'='value'

您可以指定一个 AWS Key Management Service 密钥，为 Amazon S3 对象启用服务器端加密 (SSE)，其中 value 为以下值之一：

- `auto`，使用存储在 Amazon S3 桶中的默认 AWS KMS 密钥。
- kms-key，用于指定加密数据。

select_statement

通过定义任何查询将一行或多行插入外部表的语句。查询生成的所有行都将根据表定义以文本或 Parquet 格式写入到 Amazon S3。

示例

示例 中提供了一系列示例。

使用说明

本主题包含 CREATE EXTERNAL TABLE 的用法说明。您无法使用用于标准 Amazon Redshift 表（如 PG_TABLE_DEF、STV_TBL_PERM、PG_CLASS 或 information_schema）的同一资源查看 Amazon Redshift Spectrum 表的详细信息。如果您的商业智能或分析工具无法识别 Redshift Spectrum 外部表，请将您的应用程序为配置查询 SVV_EXTERNAL_TABLES 和 SVV_EXTERNAL_COLUMNS。

CREATE EXTERNAL TABLE AS

在某些情况下，您可能会对 AWS Glue Data Catalog、AWS Lake Formation 外部目录或 Apache Hive 元存储运行 CREATE EXTERNAL TABLE AS 命令。在这种情况下，您使用 AWS Identity and Access Management (IAM) 角色创建外部架构。此 IAM 角色必须同时具有在 Amazon S3 上读取和写入的权限。

如果您使用 Lake Formation 目录，则 IAM 角色必须具有在目录中创建表的权限。在这种情况下，它还必须对目标 Amazon S3 路径具有数据湖位置权限。此 IAM 角色成为新 AWS Lake Formation 表的所有者。

为确保文件名是唯一的，Amazon Redshift 预设情况下对上载到 Amazon S3 的每个文件的名称使用以下格式。

*<date>*_*<time>*_*<microseconds>*_*<query_id>*_*<slice-number>*_part_*<part-number>*.*<format>*.

示例是 20200303_004509_810669_1007_0001_part_00.parquet。

运行 CREATE EXTERNAL TABLE AS 命令时，请考虑以下事项：

- Amazon S3 位置必须为空。
- Amazon Redshift 仅在使用 STORED AS 子句时才支持 PARQUET 和 TEXTFILE 格式。
- 您不需要定义列定义列表。新外部表的列名和列数据类型直接从 SELECT 查询获得。
- 您无需在 PARTITIONED BY 子句中定义分区列的数据类型。如果指定分区键，则此列的名称必须存在于 SELECT 查询结果中。当有多个分区列时，它们在 SELECT 查询中的顺序并不重要。Amazon Redshift 使用在 PARTITIONED BY 子句中定义的顺序来创建外部表。
- Amazon Redshift 根据分区键值自动将输出文件分区到分区文件夹中。预设情况下，Amazon Redshift 从输出文件中删除分区列。
- 不支持 LINES TERMINATED BY 'delimiter' 子句。
- 不支持 ROW FORMAT SERDE 'serde_name' 子句。
- 不支持使用清单文件。因此，您无法在 Amazon S3 上的清单文件中定义 LOCATION 子句。
- Amazon Redshift 自动在命令末尾更新“numRows”表属性。
- 'compression_type' 表属性对于 PARQUET 文件格式仅接受 'none' 或 'snappy'。
- Amazon Redshift 不允许在外部 SELECT 查询中使用 LIMIT 子句。相反，您可以使用嵌套的 LIMIT 子句。
- 您可以使用 STL_UNLOAD_LOG 跟踪由每个 CREATE EXTERNAL TABLE AS 操作写入到 Amazon S3 的文件。

创建和查询外部表的权限

要创建外部表，请确保您是外部架构的所有者或超级用户。要移交外部 schema 的所有权，请使用 ALTER SCHEMA。以下示例将 spectrum_schema schema 的所有者更改为 newowner。

```
alter schema spectrum_schema owner to newowner;
```

要运行 Redshift Spectrum 查询，您需要以下权限：

- schema 的使用权限
- 在当前数据库中创建临时表的权限

以下示例将 schema spectrum_schema 的使用权限授予 spectrumusers 用户组。

```
grant usage on schema spectrum_schema to group spectrumusers;
```

以下示例将数据库 spectrumdb 的临时权限授予 spectrumusers 用户组。

```
grant temp on database spectrumdb to group spectrumusers;
```

Pseudocolumns

预设情况下，Amazon Redshift 使用伪列 $path 和 $size 创建外部表。选择这些列可针对查询返回的每行查看 Amazon S3 上数据文件的路径以及数据文件的大小。$path 和 $size 列名称必须用双引号分隔。SELECT * 子句不返回 pseudocolumns。如下例所示，必须在查询中显式包含 $path 和 $size 列名称。

```
select "$path", "$size"
from spectrum.sales_part
where saledate = '2008-12-01';
```

您可以通过将 spectrum_enable_pseudo_columns 配置参数设置为 false 来禁用为会话创建 pseudocolumns 的功能。

> **Important**
>
> 选择 $size 或 $path 将产生费用，因为 Redshift Spectrum 会扫描 Amazon S3 中的数据文件来确定结果集的大小。有关更多信息，请参阅 Amazon Redshift 定价。

设置数据处理选项

您可以设置表参数以指定外部表中查询的数据的输入处理，其中包括：

- 包含 VARCHAR、CHAR 和字符串数据的列中的多余字符。有关更多信息，请参阅外部表属性 surplus_char_handling。

- 包含 VARCHAR、CHAR 和字符串数据的列中的无效字符。有关更多信息，请参阅外部表属性 `invalid_char_handling`。
- 为外部表属性 `invalid_char_handling` 指定 REPLACE 时要使用的替换字符。
- 包含整数和十进制数据的列的转换溢出处理。有关更多信息，请参阅外部表属性 `numeric_overflow_handling`。
- Surplus_bytes_handling 为包含 varbyte 数据的列中的超额字节指定输入处理。有关更多信息，请参阅外部表属性 `surplus_bytes_handling`。

示例

以下示例在名为 `spectrum` 的 Amazon Redshift 外部 schema 中创建一个名为 SALES 的表。数据位于制表符分隔的文本文件中。TABLE PROPERTIES 子句将 numRows 属性设置为 170000 行。

根据您用于运行 CREATE EXTERNAL TABLE 的身份，可能需要配置 IAM 权限。作为最佳实践，我们建议将权限策略附加到 IAM 角色，然后根据需要将其分配给用户和组。有关更多信息，请参阅 Amazon Redshift 中的 Identity and Access Management。

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
saledate date,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/'
table properties ('numRows'='170000');
```

以下示例创建一个使用 JsonSerDe 以 JSON 格式引用数据的表。

```
create external table spectrum.cloudtrail_json (
event_version int,
event_id bigint,
```

```
event_time timestamp,
event_type varchar(10),
awsregion varchar(20),
event_name varchar(max),
event_source varchar(max),
requesttime timestamp,
useragent varchar(max),
recipientaccountid bigint)
row format serde 'org.openx.data.jsonserde.JsonSerDe'
with serdeproperties (
'dots.in.keys' = 'true',
'mapping.requesttime' = 'requesttimestamp'
) location 's3://mybucket/json/cloudtrail';
```

以下 CREATE EXTERNAL TABLE AS 示例创建一个未分区的外部表。然后，它将 SELECT 查询的结果以 Apache Parquet 格式写入到目标 Amazon S3 位置。

```
CREATE EXTERNAL TABLE spectrum.lineitem
STORED AS parquet
LOCATION 'S3://mybucket/cetas/lineitem/'
AS SELECT * FROM local_lineitem;
```

以下示例创建分区的外部表，并在 SELECT 查询中包含分区列。

```
CREATE EXTERNAL TABLE spectrum.partitioned_lineitem
PARTITIONED BY (l_shipdate, l_shipmode)
STORED AS parquet
LOCATION 'S3://mybucket/cetas/partitioned_lineitem/'
AS SELECT l_orderkey, l_shipmode, l_shipdate, l_partkey FROM local_table;
```

如需外部数据目录中的现有数据库的列表，请查询 SVV_EXTERNAL_DATABASES 系统视图。

```
select eskind,databasename,esoptions from svv_external_databases order by databasename;
```

```
eskind | databasename | esoptions
-------+--------------
+-------------------------------------------------------------------------------
     1 | default      | {"REGION":"us-
west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
     1 | sampledb     | {"REGION":"us-
west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

```
     1 | spectrumdb   | {"REGION":"us-
west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

要查看外部表的详细信息，请查询 SVV_EXTERNAL_TABLES 和 SVV_EXTERNAL_COLUMNS 系统视图。

以下示例将查询 SVV_EXTERNAL_TABLES 视图。

```
select schemaname, tablename, location from svv_external_tables;
```

```
schemaname | tablename            | location
-----------+----------------------
+-----------------------------------------------------
spectrum   | sales                | s3://redshift-downloads/tickit/spectrum/sales
spectrum   | sales_part           | s3://redshift-downloads/tickit/spectrum/
sales_partition
```

以下示例将查询 SVV_EXTERNAL_COLUMNS 视图。

```
select * from svv_external_columns where schemaname like 'spectrum%' and tablename
 ='sales';
```

```
schemaname | tablename | columnname | external_type | columnnum | part_key
-----------+-----------+------------+---------------+-----------+---------
spectrum   | sales     | salesid    | int           |         1 |        0
spectrum   | sales     | listid     | int           |         2 |        0
spectrum   | sales     | sellerid   | int           |         3 |        0
spectrum   | sales     | buyerid    | int           |         4 |        0
spectrum   | sales     | eventid    | int           |         5 |        0
spectrum   | sales     | saledate   | date          |         6 |        0
spectrum   | sales     | qtysold    | smallint      |         7 |        0
spectrum   | sales     | pricepaid  | decimal(8,2)  |         8 |        0
spectrum   | sales     | commission | decimal(8,2)  |         9 |        0
spectrum   | sales     | saletime   | timestamp     |        10 |        0
```

要查看表分区，请使用以下查询。

```
select schemaname, tablename, values, location
from svv_external_partitions
where tablename = 'sales_part';
```

```
schemaname | tablename  | values         | location
-----------+------------+----------------
+-------------------------------------------------------------------------
spectrum   | sales_part | ["2008-01-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-01
spectrum   | sales_part | ["2008-02-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-02
spectrum   | sales_part | ["2008-03-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-03
spectrum   | sales_part | ["2008-04-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-04
spectrum   | sales_part | ["2008-05-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-05
spectrum   | sales_part | ["2008-06-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-06
spectrum   | sales_part | ["2008-07-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-07
spectrum   | sales_part | ["2008-08-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-08
spectrum   | sales_part | ["2008-09-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-09
spectrum   | sales_part | ["2008-10-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-10
spectrum   | sales_part | ["2008-11-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-11
spectrum   | sales_part | ["2008-12-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-12
```

以下示例将为外部表返回相关数据文件的总大小。

```
select distinct "$path", "$size"
   from spectrum.sales_part;

 $path                                                                      | $size
----------------------------------------------------------------------------+-------
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/ |  1616
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/ |  1444
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/ |  1444
```

分区示例

要创建按日期分区的外部表，请运行以下命令。

```
create external table spectrum.sales_part(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (saledate date)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales_partition/'
table properties ('numRows'='170000');
```

要添加分区，请运行以下 ALTER TABLE 命令。

```
alter table spectrum.sales_part
add if not exists partition (saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-02-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-03-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-04-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-04/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-05-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-05/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-06-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-06/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-07-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-07/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-08-01')
```

```
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-08/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-09-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-09/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-10-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-10/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-11-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-11/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-12-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-12/';
```

要从分区表中选择数据，请运行以下查询。

```
select top 10 spectrum.sales_part.eventid, sum(spectrum.sales_part.pricepaid)
from spectrum.sales_part, event
where spectrum.sales_part.eventid = event.eventid
  and spectrum.sales_part.pricepaid > 30
  and saledate = '2008-12-01'
group by spectrum.sales_part.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
    914 | 36173.00
   5478 | 27303.00
   5061 | 26383.00
   4406 | 26252.00
   5324 | 24015.00
   1829 | 23911.00
   3601 | 23616.00
   3665 | 23214.00
   6069 | 22869.00
   5638 | 22551.00
```

要查看外部表分区，请查询 SVV_EXTERNAL_PARTITIONS 系统视图。

```
select schemaname, tablename, values, location from svv_external_partitions
where tablename = 'sales_part';
```

```
schemaname | tablename  | values         | location
-----------+------------+----------------
+-----------------------------------------------
spectrum   | sales_part | ["2008-01-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-01
spectrum   | sales_part | ["2008-02-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-02
spectrum   | sales_part | ["2008-03-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-03
spectrum   | sales_part | ["2008-04-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-04
spectrum   | sales_part | ["2008-05-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-05
spectrum   | sales_part | ["2008-06-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-06
spectrum   | sales_part | ["2008-07-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-07
spectrum   | sales_part | ["2008-08-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-08
spectrum   | sales_part | ["2008-09-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-09
spectrum   | sales_part | ["2008-10-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-10
spectrum   | sales_part | ["2008-11-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-11
spectrum   | sales_part | ["2008-12-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-12
```

行格式示例

下面显示为以 AVRO 格式存储的数据文件指定 ROW FORMAT SERDE 参数的示例。

```
create external table spectrum.sales(salesid int, listid int, sellerid int,
 buyerid int, eventid int, dateid int, qtysold int, pricepaid decimal(8,2), comment
 VARCHAR(255))
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
WITH SERDEPROPERTIES ('avro.schema.literal'='{\"namespace\": \"dory.sample\",\"name\":
 \"dory_avro\",\"type\": \"record\", \"fields\": [{\"name\":\"salesid\", \"type\":\"int
\"},
{\"name\":\"listid\", \"type\":\"int\"},
{\"name\":\"sellerid\", \"type\":\"int\"},
{\"name\":\"buyerid\", \"type\":\"int\"},
{\"name\":\"eventid\",\"type\":\"int\"},
```

```
{\"name\":\"dateid\",\"type\":\"int\"},
{\"name\":\"qtysold\",\"type\":\"int\"},
{\"name\":\"pricepaid\", \"type\": {\"type\": \"bytes\", \"logicalType\": \"decimal\",
 \"precision\": 8, \"scale\": 2}}, {\"name\":\"comment\",\"type\":\"string\"}]}')
STORED AS AVRO
location 's3://mybucket/avro/sales' ;
```

下面显示了使用 RegEx 指定 ROW FORMAT SERDE 参数的示例。

```
create external table spectrum.types(
cbigint bigint,
cbigint_null bigint,
cint int,
cint_null int)
row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe'
with serdeproperties ('input.regex'='([^\\x01]+)\\x01([^\\x01]+)\\x01([^\\x01]+)\
\x01([^\\x01]+)')
stored as textfile
location 's3://mybucket/regex/types';
```

下面显示了使用 Grok 指定 ROW FORMAT SERDE 参数的示例。

```
create external table spectrum.grok_log(
timestamp varchar(255),
pid varchar(255),
loglevel varchar(255),
progname varchar(255),
message varchar(255))
row format serde 'com.amazonaws.glue.serde.GrokSerDe'
with serdeproperties ('input.format'='[DFEWI], \\[%{TIMESTAMP_ISO8601:timestamp} #
%{POSINT:pid:int}\\] *(?<loglevel>:DEBUG|FATAL|ERROR|WARN|INFO) -- +%{DATA:progname}:
 %{GREEDYDATA:message}')
stored as textfile
location 's3://mybucket/grok/logs';
```

下面显示了一个有关在 S3 桶中定义 Amazon S3 服务器访问日志的示例。您可以使用 Redshift Spectrum 查询 Amazon S3 访问日志。

```
CREATE EXTERNAL TABLE spectrum.mybucket_s3_logs(
bucketowner varchar(255),
bucket varchar(255),
requestdatetime varchar(2000),
```

```
remoteip varchar(255),
requester varchar(255),
requested varchar(255),
operation varchar(255),
key varchar(255),
requesturi_operation varchar(255),
requesturi_key varchar(255),
requesturi_httpprotoversion varchar(255),
httpstatus varchar(255),
errorcode varchar(255),
bytessent bigint,
objectsize bigint,
totaltime varchar(255),
turnaroundtime varchar(255),
referrer varchar(255),
useragent varchar(255),
versionid varchar(255)
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
'input.regex' = '([^ ]*) ([^ ]*) \\[(.*?)\\] ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*)
 \"([^ ]*)\\s*([^ ]*)\\s*([^ ]*)\" (- |[^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*)
 ([^ ]*) (\"[^\"]*\") ([^ ]*).*$')
LOCATION 's3://mybucket/s3logs’;
```

以下示例为 ION 格式的数据指定了 ROW FORMAT SERDE 参数。

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'com.amazon.ionhiveserde.IonHiveSerDe'
STORED AS
INPUTFORMAT 'com.amazon.ionhiveserde.formats.IonInputFormat'
OUTPUTFORMAT 'com.amazon.ionhiveserde.formats.IonOutputFormat'
LOCATION 's3://s3-bucket/prefix'
```

数据处理示例

以下示例访问该文件：spi_global_rankings.csv。您可以将 spi_global_rankings.csv 文件上载到 Amazon S3 桶以尝试这些示例。

以下示例创建外部架构 schema_spectrum_uddh 和数据库 spectrum_db_uddh。对于 aws-account-id，请输入您的 AWS 账户 ID，而对于 role-name，请输入您的 Redshift Spectrum 角色名称。

```
create external schema schema_spectrum_uddh
from data catalog
database 'spectrum_db_uddh'
iam_role 'arn:aws:iam::aws-account-id:role/role-name'
create external database if not exists;
```

以下示例在外部架构 schema_spectrum_uddh 中创建外部表 soccer_league。

```
CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league
(
  league_rank smallint,
  prev_rank   smallint,
  club_name   varchar(15),
  league_name varchar(20),
  league_off  decimal(6,2),
  league_def  decimal(6,2),
  league_spi  decimal(6,2),
  league_nspi integer
)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n\l'
stored as textfile
LOCATION 's3://spectrum-uddh/league/'
table properties ('skip.header.line.count'='1');
```

请检查 soccer_league 表中的行数。

```
select count(*) from schema_spectrum_uddh.soccer_league;
```

此时将显示行数。

```
count
645
```

以下查询显示前 10 个俱乐部。由于俱乐部 Barcelona 字符串中包含无效字符，因此对该名称显示 NULL。

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
```

```
where league_rank between 1 and 10;
```

```
league_rank club_name league_name   league_nspi
1  Manchester City Barclays Premier Lea  34595
2  Bayern Munich German Bundesliga  34151
3  Liverpool Barclays Premier Lea  33223
4  Chelsea  Barclays Premier Lea  32808
5  Ajax  Dutch Eredivisie  32790
6  Atletico  Madrid Spanish Primera Divi 31517
7  Real Madrid Spanish Primera Divi  31469
8  NULL         Spanish Primera Divi            31321
9  RB Leipzig German Bundesliga  31014
10  Paris Saint-Ger French Ligue 1   30929
```

以下示例更改了 soccer_league 表，以指定用于插入一个问号 (?) 来替换意外字符的 invalid_char_handling、replacement_char 和 data_cleansing_enabled 外部表属性。

```
alter  table schema_spectrum_uddh.soccer_league
set table properties
 ('invalid_char_handling'='REPLACE','replacement_char'='?','data_cleansing_enabled'='true');
```

以下示例将查询排名从 1 到 10 的团队的表 soccer_league。

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

由于表属性已更改，结果显示了前 10 位俱乐部，在第八行中对于俱乐部 Barcelona 采用问号 (?) 替换字符。

```
league_rank club_name league_name  league_nspi
1  Manchester City Barclays Premier Lea 34595
2  Bayern Munich German Bundesliga 34151
3  Liverpool Barclays Premier Lea 33223
4  Chelsea  Barclays Premier Lea 32808
5  Ajax  Dutch Eredivisie 32790
6  Atletico Madrid Spanish Primera Divi 31517
7  Real Madrid Spanish Primera Divi 31469
8  Barcel?na Spanish Primera Divi 31321
9  RB Leipzig German Bundesliga 31014
```

```
10  Paris Saint-Ger French Ligue 1  30929
```

以下示例更改了 soccer_league 表，以指定用于剔除包含意外字符的行的 invalid_char_handling 外部表属性。

```
alter table schema_spectrum_uddh.soccer_league
set table properties
 ('invalid_char_handling'='DROP_ROW','data_cleansing_enabled'='true');
```

以下示例将查询排名从 1 到 10 的团队的表 soccer_league。

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

结果显示排名靠前的俱乐部，不包括对应于俱乐部 Barcelona 的第八行。

```
league_rank   club_name         league_name            league_nspi
1             Manchester City   Barclays Premier Lea   34595
2             Bayern Munich     German Bundesliga      34151
3             Liverpool         Barclays Premier Lea   33223
4             Chelsea           Barclays Premier Lea   32808
5             Ajax              Dutch Eredivisie       32790
6             Atletico Madrid   Spanish Primera Divi   31517
7             Real Madrid       Spanish Primera Divi   31469
9             RB Leipzig        German Bundesliga      31014
10            Paris Saint-Ger   French Ligue 1         30929
```

CREATE EXTERNAL VIEW（预览版）

以下是预览版 Data Catalog for Amazon Redshift 中的预发行文档视图。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

您可以在预览版中创建 Amazon Redshift 集群，以便测试 Amazon Redshift 的新功能。您无法在生产环境中使用这些功能，也无法将预览版集群移动到生产集群或另一个跟踪上的集群。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

在预览版中创建集群

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择预置集群控制面板，然后选择集群。列出您的账户在当前 AWS 区域 区域中的集群。列表中的各个列中显示了每个集群的一部分属性。
3. 集群列表页面上会显示一个横幅，其中介绍了预览版。选择创建预览版集群按钮以打开创建集群页面。
4. 输入集群的属性。选择包含要测试的功能的预览版跟踪。我们建议输入的集群名称指明要对该集群进行预览版跟踪。为您的集群选择选项，包括标记为 -preview 的选项，用于要测试的功能。有关创建集群的一般信息，请参阅《Amazon Redshift 管理指南》中的创建集群。
5. 选择创建集群以在预览模式下创建集群。

   > **Note**
   >
   > `preview_2023` 跟踪是最新可用的预览版跟踪。此版本仅支持创建具有 RA3 节点类型的集群。不支持节点类型 DC2 以及任何更早的节点类型。

6. 当您的预览集群可用时，使用 SQL 客户端加载和查询数据。

Data Catalog 视图预览功能仅在以下区域中可用。

- 美国东部（俄亥俄州）(us-east-2)
- 美国东部（弗吉尼亚州北部）(us-east-1)
- 美国西部（北加利福尼亚）(us-west-1)
- 亚太地区（东京）(ap-northeast-1)
- 欧洲地区（爱尔兰）(eu-west-1)
- 欧洲地区（斯德哥尔摩）(eu-north-1)

您也可以创建预览工作组来测试 Data Catalog 视图。您无法在生产中使用这些功能，也无法将您的工作组移至其他工作组。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。有关如何创建预览工作组的说明，请参阅创建预览工作组。

在 Data Catalog 中创建视图。Data Catalog 视图是一种单一视图架构，可与其他 SQL 引擎（如 Amazon Athena 和 Amazon EMR）配合使用。您可以通过选择的引擎查询视图。有关 Data Catalog 视图的更多信息，请参阅创建 Data Catalog 视图（预览版）。

语法

```
CREATE EXTERNAL VIEW schema_name.view_name [ IF NOT EXISTS ]
{catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name |
 external_schema_name.view_name}
AS query_definition;
```

参数

schema_name.view_name

附加到 AWS Glue 数据库的架构，后面是视图的名称。

PROTECTED

指定只有在 query_definition 中的查询能成功完成时，CREATE EXTERNAL VIEW 命令才能完成。

IF NOT EXISTS

如果视图尚不存在，则创建该视图。

catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name | external_schema_name.view_name

创建视图时要使用的架构符号。可以指定使用您创建的 Glue 数据库 AWS Glue Data Catalog 或您创建的外部架构。有关更多信息，请参阅 CREATE DATABASE 和 CREATE EXTERNAL SCHEMA。

query_definition

Amazon Redshift 为更改视图而运行的 SQL 查询的定义。

示例

以下示例创建一个名为 sample_schema.glue_data_catalog_view 的 Data Catalog 视图。

```
CREATE EXTERNAL PROTECTED VIEW sample_schema.glue_data_catalog_view IF NOT EXISTS
AS SELECT * FROM sample_database.remote_table "remote-table-name";
```

CREATE FUNCTION

使用 SQL SELECT 子句或 Python 程序创建新的标量用户定义的函数 (UDF)。

有关更多信息以及示例，请参阅 创建用户定义的函数。

所需的权限

您必须通过以下方式之一获得权限，才能运行 CREATE OR REPLACE FUNCTION：

- CREATE FUNCTION：
  - 超级用户可以使用可信和不受信任的语言来创建函数。
  - 具有 CREATE [OR REPLACE] FUNCTION 权限的用户可以使用可信语言创建函数。
- REPLACE FUNCTION：
  - Superuser
  - 具有 CREATE [OR REPLACE] FUNCTION 权限的用户
  - 函数拥有者

语法

```
CREATE [ OR REPLACE ] FUNCTION f_function_name
( { [py_arg_name  py_arg_data_type |
sql_arg_data_type } [ , ... ] ] )
RETURNS data_type
{ VOLATILE | STABLE | IMMUTABLE }
AS $$
  { python_program | SELECT_clause }
$$ LANGUAGE { plpythonu | sql }
```

参数

OR REPLACE

指定如果某个函数与已存在的此函数具有相同的名称和输入参数数据类型或签名，则替换现有的函数。您只能将某个函数替换为定义一组相同数据类型的新函数。您必须是超级用户才能替换函数。

如果您定义的函数与现有函数具有相同的名称，但签名不同，则创建新的函数。换而言之，函数名称将会重载。有关更多信息，请参阅 重载函数名称。

f_function_name

函数的名称。如果您指定 schema 名称（例如 `myschema.myfunction`），则使用指定的 schema 创建函数。否则，将在当前 schema 中创建该函数。有关有效名称的更多信息，请参阅名称和标识符。

我们建议您将所有 UDF 名称添加前缀 f_。Amazon Redshift 保留 f_ 前缀供 UDF 名称使用。因此，使用 f_ 前缀，您可以确保您的 UDF 名称不会与现有或未来的 Amazon Redshift 内置 SQL 函数名称冲突。有关更多信息，请参阅 对 UDF 命名。

如果输入参数的数据类型不同，您可以定义多个具有相同函数名称的函数。换而言之，函数名称将会重载。有关更多信息，请参阅 重载函数名称。

py_arg_name py_arg_data_type | sql_arg_data 类型

对于 Python UDF，为参数名称和数据类型的列表。对于 SQL UDF，为数据类型的列表，不含参数名称。在 Python UDF 中，使用参数名称引用参数。在 SQL UDF 中，使用 $1、$2 等基于参数在参数列表中的顺序引用参数。

对于 SQL UDF，输入和返回数据类型可以是任何标准 Amazon Redshift 数据类型。对于 Python UDF，输入和返回数据类型可以为 SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE 或 TIMESTAMP。此外，Python 用户定义的函数 (UDF) 支持数据类型 ANYELEMENT。此数据类型会根据在运行时提供的相应参数的数据类型，自动转换为标准数据类型。如果多个参数使用 ANYELEMENT，则它们会根据列表中的第一个 ANYELEMENT 参数，在运行时全部解析为相同的数据类型。有关更多信息，请参阅Python UDF 数据类型和数据类型。

您可以指定最多 32 个参数。

RETURNS data_type

函数返回的值的数据类型。RETURNS 数据类型可以是任何标准的 Amazon Redshift 数据类型。此外，Python UDF 可以使用 ANYELEMENT 数据类型，它会根据在运行时提供的参数，自动转换为标准数据类型。如果您为返回数据类型指定 ANYELEMENT，则至少有一个参数必须使用 ANYELEMENT。在调用函数时，实际返回数据类型会匹配为 ANYELEMENT 参数提供的数据类型。有关更多信息，请参阅 Python UDF 数据类型。

VOLATILE | STABLE | IMMUTABLE

通知查询优化程序有关函数的不稳定性。

如果您将函数标记为其有效的最严格稳定性类别，您将获得最大程度的优化。但是，如果类别过于严格，则存在优化程序错误地忽略某些调用的风险，从而导致不正确的结果集。按照严格性顺序，从最不严格的开始，稳定性类别如下所示：

- VOLATILE
- STABLE
- IMMUTABLE

VOLATILE

对于相同的参数，函数会对连续的调用返回不同的结果，甚至对于单个语句中的行也是如此。优化查询程序无法对不稳定函数的行为做出任何假设，因此使用不稳定函数的查询必须对每个输入行重新计算该函数。

STABLE

对于相同的参数，可保证函数对在单个语句内处理的所有行返回相同的结果。在不同的语句中调用时，函数可能会返回不同的结果。此类别使优化程序能够将单个语句内对该函数的多个调用优化为对该语句的单个调用。

IMMUTABLE

对于相同的参数，函数始终返回相同的结果。当查询使用常量参数调用 IMMUTABLE 函数时，优化程序会预先计算函数。

AS $$ statement $$

包含要执行的语句的构造。需要文字关键字 AS $$ 和 $$。

Amazon Redshift 要求您使用称为“美元引号”的格式，在您的函数中包含语句。包含的任何内容将按原样传递。您不必对任何特殊字符进行转义，因为字符串的内容是按照其字面涵义编写的。

通过美元引号 格式，您可以使用一对美元符号 ($$) 来指示要运行的语句的开头和结尾，如以下示例所示。

```
$$ my statement $$
```

（可选）在每对美元符号之间，可以指定字符串来帮助识别语句。您使用的字符串必须在括起字符对的开始和结束都是相同的。该字符串区分大小写，它遵循与不带括起字符的标识符相同的约束，但有一点除外，它不能包含美元符号。以下示例使用字符串 test。

```
$test$ my statement $test$
```

有关“美元引号”格式的更多信息，请参阅 PostgreSQL 文档的词法结构中的“使用美元符号括起的常量字符串”。

python_program

返回值的有效 Python 可执行程序。您在函数中传递的语句必须符合 Python 网站上的 Python 代码样式指南中规定的缩进要求。有关更多信息，请参阅 适用于 UDF 的 Python 语言支持。

SQL_clause

SQL SELECT 子句。

SELECT 子句不能包含以下任何类型的子句：

- FROM
- INTO
- WHERE
- GROUP BY
- ORDER BY
- LIMIT

LANGUAGE { plpythonu | sql }

对于 Python，指定 `plpythonu`。对于 SQL，指定 `sql`。您必须具有使用 SQL 或 plpythonu 语言的权限。有关更多信息，请参阅 UDF 安全性和权限。

使用说明

嵌套函数

您可以从一个 SQL UDF 中调用另一个 SQL 用户定义函数 (UDF)。当您运行 CREATE FUNCTION 命令时，嵌套函数必须存在。Amazon Redshift 不会跟踪 UDF 的依赖项，因此，如果您删除嵌套函数，Amazon Redshift 不会返回错误。但是，如果嵌套函数不存在，则 UDF 将失败。例如，以下函数调用 SELECT 子句中的 `f_sql_greater` 函数。

```
create function f_sql_commission (float, float )
  returns float
stable
as $$
  select f_sql_greater ($1, $2)
$$ language sql;
```

UDF 安全性和权限

要创建 UDF，您必须具有使用 SQL 或 plpythonu (Python) 语言的权限。默认情况下，向 PUBLIC 授予 USAGE ON LANGUAGE SQL。但是，您必须明确授予 USAGE ON LANGUAGE PLPYTHONU 权限才能指定用户或组。

要撤销 SQL 的使用权限，请先从 PUBLIC 撤销使用权限。然后，仅向允许创建 SQL UDF 的特定用户或组授予 SQL 使用权限。以下示例将从 PUBLIC 撤销对 SQL 的使用权限，然后向用户组 udf_devs 授予使用权限。

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

要运行 UDF，您必须拥有每个函数的执行权限。默认情况下，向 PUBLIC 授予新 UDF 的执行权限。要限制使用，请从 PUBLIC 撤消函数的执行权限。然后向特定的个人或组授予权限。

以下示例从 PUBLIC 撤消了对函数 f_py_greater 的执行权限，然后向用户组 udf_devs 授予使用权限。

```
revoke execute on function f_py_greater(a float, b float) from PUBLIC;
grant execute on function f_py_greater(a float, b float) to group udf_devs;
```

默认情况下，超级用户拥有全部权限。

有关更多信息，请参阅GRANT和REVOKE。

示例

标量 Python UDF 示例

以下示例创建用于比较两个整数值并返回较大值的 Python UDF。

```
create function f_py_greater (a float, b float)
  returns float
stable
as $$
  if a > b:
    return a
  return b
$$ language plpythonu;
```

以下示例查询 SALES 表并调用新的 f_py_greater 函数，以返回 COMMISSION 和 PRICEPAID 的 20% 这两个值中较大的值。

```
select f_py_greater (commission, pricepaid*0.20) from sales;
```

标量 SQL UDF 示例

以下示例创建一个用于比较两个数并返回较大值的函数。

```
create function f_sql_greater (float, float)
  returns float
stable
as $$
  select case when $1 > $2 then $1
    else $2
  end
$$ language sql;
```

以下查询将调用新的 f_sql_greater 函数以查询 SALES 表，并返回 COMMISSION 或 PRICEPAID 的 20% (两个值中的较大者)。

```
select f_sql_greater (commission, pricepaid*0.20) from sales;
```

CREATE GROUP

定义新的用户组。只有超级用户可以创建组。

语法

```
CREATE GROUP group_name
[ [ WITH ] [ USER username ] [, ...] ]
```

参数

group_name

新用户组的名称。以两个下划线开头的组名保留供 Amazon Redshift 内部使用。有关有效名称的更多信息，请参阅名称和标识符。

WITH

可选语法，用于指示 CREATE GROUP 的额外参数。

USER

向组中添加一个或多个用户。

username

要添加到组中的用户的名称。

示例

以下示例创建名为 ADMIN_GROUP 的用户组，其中包含两个用户 ADMIN1 和 ADMIN2。

```
create group admin_group with user admin1, admin2;
```

CREATE IDENTITY PROVIDER

定义新的身份提供者。只有超级用户可以创建身份提供者。

语法

```
CREATE IDENTITY PROVIDER identity_provider_name TYPE type_name
NAMESPACE namespace_name
[PARAMETERS parameter_string]
[APPLICATION_ARN arn]
[IAM_ROLE iam_role]
```

参数

identity_provider_name

新身份提供者的名称。有关有效名称的更多信息，请参阅名称和标识符。

type_name

要与之交互的身份提供者。Azure 目前是唯一受支持的身份提供者。

namespace_name

命名空间。这是身份提供者目录的唯一速记标识符。

parameter_string

一个包含格式正确的 JSON 对象的字符串，其中包含身份提供者所需的参数和值。

arn

IAM Identity Center 托管应用程序的 Amazon 资源名称（ARN）。仅当身份提供者类型为 AWSIDC 时，此参数才适用。

iam_role

提供连接到 IAM Identity Center 的权限的 IAM 角色。仅当身份提供者类型为 AWSIDC 时，此参数才适用。

示例

下面的示例创建一个名为 oauth_standard、TYPE 为 azure 的身份提供者，以与 Microsoft Azure Active Directory (AD) 建立通信。

```
CREATE IDENTITY PROVIDER oauth_standard TYPE azure
NAMESPACE 'aad'
PARAMETERS '{"issuer":"https://sts.windows.net/2sdfdsf-d475-420d-b5ac-667adad7c702/",
"client_id":"87f4aa26-78b7-410e-bf29-57b39929ef9a",
"client_secret":"BUAH~ewrqewrqwerUUY^%tHe1oNZShoiU7",
"audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift"]
}'
```

您可以将 IAM Identity Center 托管应用程序与现有预调配集群或 Amazon Redshift Serverless 工作组进行连接。这使您能够通过 IAM Identity Center 管理对 Redshift 数据库的访问权限。为此，请运行如以下示例所示的 SQL 命令。您必须是数据库管理员。

```
CREATE IDENTITY PROVIDER "redshift-idc-app" TYPE AWSIDC
NAMESPACE 'awsidc'
APPLICATION_ARN 'arn:aws:sso::123456789012:application/ssoins-12345f67fe123d4/apl-
a0b0a12dc123b1a4'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyRedshiftRole';
```

在本例中，应用程序 ARN 标识要连接到的托管应用程序。您可以通过运行 `SELECT * FROM SVV_IDENTITY_PROVIDERS;` 找到该应用程序。

有关使用 CREATE IDENTITY PROVIDER 的更多信息，包括其他示例，请参阅 Amazon Redshift 的原生身份提供者 (IdP) 联合身份验证。有关设置从 Redshift 到 IAM Identity Center 的连接的更多信息，请参阅将 Redshift 与 IAM Identity Center 连接，为用户提供单点登录体验。

CREATE LIBRARY

安装一个 Python 库，在使用 CREATE FUNCTION 命令创建用户定义的函数 (UDF) 时，用户可使用该库进行整合。用户安装的库的总大小不能超过 100MB。

CREATE LIBRARY 无法在事务块 (BEGIN … END) 内运行。有关事务的更多信息，请参阅 可序列化的隔离。

Amazon Redshift 支持 Python 2.7 版本。有关更多信息，请参阅 www.python.org。

有关更多信息，请参阅 导入自定义 Python 库模块。

所需的权限

以下是 CREATE LIBRARY 所需的权限：

- Superuser
- 具有 CREATE LIBRARY 权限或具有指定语言权限的用户

语法

```
CREATE [ OR REPLACE ] LIBRARY library_name LANGUAGE plpythonu
FROM
{ 'https://file_url'
| 's3://bucketname/file_name'
authorization
  [ REGION [AS] 'aws_region']
  IAM_ROLE { default | ‘arn:aws:iam::<AWS ##-id>:role/<role-name>’ }
}
```

参数

OR REPLACE

指定如果存在与已存在的库同名的库，则替换现有库。REPLACE 将立即提交。如果依赖于所替换库的 UDF 正在并行运行，UDF 可能失败或返回意外结果，即使 UDF 正在事务中运行也是如此。您必须是所有者或超级用户才能替换库。

library_name

要安装的库的名称。无法创建包含与 Python Standard Library 模块或 Amazon Redshift 预安装的 Python 模块同名的模块的库。如果现有用户安装的库与要安装的库使用相同的 Python 包，则必须先删除现有库，然后再安装新库。有关更多信息，请参阅 适用于 UDF 的 Python 语言支持。

LANGUAGE plpythonu

要使用的语言。Python (plpythonu) 是唯一支持的语言。Amazon Redshift 支持 Python 2.7 版本。有关更多信息，请参阅 www.python.org。

FROM

库文件的位置。您可以指定 Amazon S3 桶和对象名称，也可以指定用于从公共网站下载文件的 URL。必须以 .zip 文件的形式打包库。有关更多信息，请参阅 Python 文档中的构建和安装 Python 模块。

https://file_url

用于从公共网站下载文件的 URL。URL 最多可包含三个重定向。以下是文件 URL 的示例。

```
'https://www.example.com/pylib.zip'
```

s3://bucket_name/file_name

包含库文件的单个 Amazon S3 对象的路径。以下是 Amazon S3 对象路径的示例。

```
's3://mybucket/my-pylib.zip'
```

如果您指定 Amazon S3 桶，则还必须为有权下载该文件的 AWS 用户提供凭证。

> **Important**
>
> 如果 Amazon S3 桶不在您的 Amazon Redshift 集群所在的 AWS 区域内，则必须使用 REGION 选项指定数据所在的 AWS 区域。aws_region 的值必须匹配 COPY 命令的 REGION 参数描述中的表中所列的 AWS 区域。

授权

一个子句，指示您的集群在访问包含库文件的 Amazon S3 桶时使用的身份验证和授权方法。您的集群必须有权利用 LIST 和 GET 操作访问 Amazon S3。

authorization 的语法与 COPY 命令 authorization 的语法相同。有关更多信息，请参阅 授权参数。

```
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id>:role/<role-name>'
```

使用默认关键字让 Amazon Redshift 使用设置为默认值并在 CREATE LIBRARY 命令运行时与集群关联的 IAM 角色。

使用 IAM 角色的 Amazon 资源名称（ARN），您的集群使用该角色进行身份验证和授权。如果您指定 IAM_ROLE，则无法使用 ACCESS_KEY_ID 和 SECRET_ACCESS_KEY、SESSION_TOKEN 或 CREDENTIALS。

（可选）如果 Amazon S3 桶使用服务器端加密，也可以在 credentials-args 字符串中提供加密密钥。如果您使用临时安全凭证，请在 credentials-args 字符串中提供临时令牌。

有关更多信息，请参阅 临时安全凭证。

REGION [AS] aws_region

Amazon S3 桶所在的 AWS 区域。当 Amazon S3 桶与 Amazon Redshift 集群不在同一个 AWS 区域时，需要 REGION。aws_region 的值必须匹配 COPY 命令的 REGION 参数描述中的表中所列的 AWS 区域。

预设情况下，CREATE LIBRARY 假定 Amazon S3 桶位于 Amazon Redshift 集群所在的 AWS 区域。

示例

以下两个示例将安装 urlparse Python 模块，该模块会打包到名为 `urlparse3-1.0.3.zip` 的文件中。

以下命令从已上载到位于美国东部区域的 Amazon S3 桶的包安装名为 `f_urlparse` 的 UDF 库。

```
create library f_urlparse
language plpythonu
from 's3://mybucket/urlparse3-1.0.3.zip'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
region as 'us-east-1';
```

以下示例从网站上的库文件安装名为 `f_urlparse` 的库。

```
create library f_urlparse
language plpythonu
from 'https://example.com/packages/urlparse3-1.0.3.zip';
```

CREATE MASKING POLICY

创建新的动态数据掩蔽策略以模糊处理给定格式的数据。有关动态数据掩蔽的更多信息，请参阅 动态数据掩蔽。

超级用户和具有 sys:secadmin 角色的用户或角色可以创建屏蔽策略。

语法

```
CREATE MASKING POLICY
   policy_name [IF NOT EXISTS]
   WITH (input_columns)
   USING (masking_expression);
```

参数

policy_name

屏蔽策略的名称。屏蔽策略不能与数据库中已存在的另一个屏蔽策略的名称相同。

input_columns

格式为 (col1 type, col2 type ...) 的列名元组。

列名用作屏蔽表达式的输入。列名不必与要掩蔽的列的名称相匹配，但输入和输出数据类型必须匹配。

masking_expression

用于转换目标列的 SQL 表达式。可以使用诸如字符串操作函数之类的数据操作函数来编写该表达式，也可以与使用 SQL、Python 或 AWS Lambda 编写的用户定义函数结合使用。您可以让具有多个输出的屏蔽策略包括一个列表达式元组。如果您使用常量作为掩蔽表达式，则必须将其显式转换为与输入类型匹配的类型。

您必须对在屏蔽表达式中使用的任何用户定义函数具有 USAGE 权限。

CREATE MATERIALIZED VIEW

基于一个或多个 Amazon Redshift 表创建实体化视图。您还可以将实体化视图建立在使用 Spectrum 或联合查询创建的外部表的基础上。有关 Spectrum 的信息，请参阅使用 Amazon Redshift Spectrum 查询外部数据。有关联合查询的信息，请参阅 在 Amazon Redshift 中使用联合查询来查询数据。

语法

```
CREATE MATERIALIZED VIEW mv_name
[ BACKUP { YES | NO } ]
[ table_attributes ]
[ AUTO REFRESH { YES | NO } ]
AS query
```

参数

BACKUP

一个子句，用于指定实体化视图是否包含在存储于 Amazon S3 中的自动和手动集群快照中。

BACKUP 的默认值为 YES。

您可以指定 BACKUP NO 来节省创建快照和从快照还原时的处理时间，并减少 Amazon S3 中所需的存储量。

> **Note**
>
> BACKUP NO 设置不会影响自动将数据复制到集群内的其他节点，因此当发生节点故障时，指定了 BACKUP NO 的表将被还原。

table_attributes

用于指定实体化视图中数据的分布方式的子句，包括以下内容：

- 实体化视图的分配方式，格式为 DISTSTYLE { EVEN | ALL | KEY }。如果忽略此子句，则分配方式为 EVEN。有关更多信息，请参阅 分配方式。
- 实体化视图的分配键，格式为 DISTKEY (*distkey_identifier*)。有关更多信息，请参阅指定分配方式。
- 实体化视图的排序键，格式为 SORTKEY (*column_name* [, ...])。有关更多信息，请参阅 使用排序键。

AS query

一个定义实体化视图及其内容的有效 SELECT 语句。来自查询的结果集定义了实体化视图的列和行。有关创建实体化视图时的限制的信息，请参阅限制。

此外，查询中使用的具体 SQL 语言结构将决定实体化视图可进行增量刷新还是完全刷新。有关刷新方法的信息，请参阅 REFRESH MATERIALIZED VIEW。有关增量刷新的限制的信息，请参阅增量刷新限制。

如果查询包含的 SQL 命令不支持递增刷新，则 Amazon Redshift 会显示一条消息，指示实体化视图将使用完全刷新。该消息可能显示，也可能不显示，具体取决于 SQL 客户端应用程序。选中 `state` 的 STV_MV_INFO 列可查看实体化视图使用的刷新类型。

AUTO REFRESH

一个子句，用于定义是否应使用其基表中的最新更改自动刷新实体化视图。默认值为 NO。有关更多信息，请参阅 刷新实体化视图。

使用说明

要创建实体化视图，您必须具有以下权限：

- 针对架构的 CREATE 权限。
- 对基表具有表级或列级 SELECT 权限以创建实体化视图。如果您对特定列具有列级权限，则可以仅在这些列上创建实体化视图。

对数据共享中的实体化视图进行增量刷新

在共享基表时，Amazon Redshift 支持对消费者数据共享中的实体化视图进行自动和增量刷新。增量刷新是一项操作，其中 Amazon Redshift 可识别上次刷新后发生的一个或多个基表中的更改，并仅更新实体化视图中的相应记录。与完全刷新相比，这项操作的运行速度更快，并且可以提高工作负载性能。您不必为了利用增量刷新而更改实体化视图的定义。

在实体化视图中使用增量刷新时，有几个限制需要注意：

- 无论是本地数据库还是远程数据库，实体化视图只能引用一个数据库。
- 增量刷新仅适用于新的实体化视图。因此，您必须删除现有的实体化视图并重新创建它们，才能进行增量刷新。

有关在数据共享中创建实体化视图的更多信息，请参阅在 Amazon Redshift 数据共享中使用视图，其中包含多个查询示例。

对实体化视图或基表的 DDL 更新

在 Amazon Redshift 中使用实体化视图时，请遵循以下有关对实体化视图或基表进行的数据定义语言 (DDL) 更新的使用说明。

- 您可以向基表添加列，而不会影响引用该基表的任何实体化视图。
- 某些操作可能会使实体化视图处于根本无法刷新的状态。例如，重命名或删除列、更改列类型、更改架构名称等此类操作。可以查询此类实体化视图，但不能对其进行刷新。在此情况下，必须删除并重新创建实体化视图。
- 通常，无法更改实体化视图的定义（其 SQL 语句）。
- 无法重命名实体化视图。

限制

您无法定义一个引用或包括以下任何内容的实体化视图：

- 标准视图或系统表和视图。
- 临时表。
- 用户定义的函数。
- ORDER BY、LIMIT 或 OFFSET 子句。
- 对基表的后期绑定引用。换句话说，在实体化视图的定义 SQL 查询中引用的任何基表或相关列必须存在且必须有效。
- 仅领导节点函数：CURRENT_SCHEMA、CURRENT_SCHEMAS、HAS_DATABASE_PRIVILEGE、HAS_SCHEMA_PRIVIL
- 当实体化视图定义包含可变函数或外部 schema 时，不能使用 AUTO REFRESH YES 选项。在一个实体化视图上定义另一个实体化视图时，也不能使用它。
- 您不必在实体化视图上手动运行 ANALYZE。该分析目前只通过 AUTO ANALYZE 发生。有关更多信息，请参阅 分析表。

示例

以下示例从三个联接和聚合的基表创建实体化视图。每个行均代表一个类别以及已售出的票数。查询 tickets_mv 实体化视图时，直接在 tickets_mv 实体化视图中访问预计算的数据。

```
CREATE MATERIALIZED VIEW tickets_mv AS
    select   catgroup,
```

```
    sum(qtysold) as sold
    from     category c, event e, sales s
    where    c.catid = e.catid
    and      e.eventid = s.eventid
    group by catgroup;
```

以下示例创建一个类似于上一个示例的实体化视图，并使用聚合函数 MAX()。

```
CREATE MATERIALIZED VIEW tickets_mv_max AS
    select   catgroup,
    max(qtysold) as sold
    from     category c, event e, sales s
    where    c.catid = e.catid
    and      e.eventid = s.eventid
    group by catgroup;

SELECT name, state FROM STV_MV_INFO;
```

以下示例使用 UNION ALL 子句联接 Amazon Redshift `public_sales` 表和 Redshift Spectrum `spectrum.sales` 表来创建实体化视图 `mv_sales_vw`。有关适用于 Amazon Redshift Spectrum 的 CREATE EXTERNAL TABLE 命令的信息，请参阅CREATE EXTERNAL TABLE。Redshift Spectrum 外部表引用 Amazon S3 上的数据。

```
CREATE MATERIALIZED VIEW mv_sales_vw as
select salesid, qtysold, pricepaid, commission, saletime from public.sales
union all
select salesid, qtysold, pricepaid, commission, saletime from spectrum.sales
```

以下示例根据联合查询外部表创建实体化视图 `mv_fq`。有关联合查询的信息，请参阅 CREATE EXTERNAL SCHEMA。

```
CREATE MATERIALIZED VIEW mv_fq as select firstname, lastname from apg.mv_fq_example;

select firstname, lastname from mv_fq;
 firstname | lastname
-----------+----------
 John      | Day
 Jane      | Doe
(2 rows)
```

以下示例显示了实体化视图的定义。

```
SELECT pg_catalog.pg_get_viewdef('mv_sales_vw'::regclass::oid, true);

pg_get_viewdef
-------------------------------------------------
create materialized view mv_sales_vw as select a from t;
```

以下示例显示了如何在实体化视图定义中设置 AUTO REFRESH 以及如何指定 DISTSTYLE。首先，创建一个简单的基表。

```
CREATE TABLE baseball_table (ball int, bat int);
```

然后，创建实体化视图。

```
CREATE MATERIALIZED VIEW mv_baseball DISTSTYLE ALL AUTO REFRESH YES AS SELECT ball AS
 baseball FROM baseball_table;
```

现在，您就可以查询 mv_baseball 实体化视图了。要检查是否对实体化视图开启了 AUTO REFRESH，请参阅 STV_MV_INFO。

以下示例创建了一个实体化视图，该视图引用了另一个数据库中的源表。它假设包含源表的数据库 database_A 与您在 database_B 中创建的实体化视图位于同一个集群或工作组中。（您可以用自己的数据库替换示例中的数据库。）首先，在 database_A 中创建一个名为 cities 的表，其中包含 cityname 列。将该列的数据类型设为 VARCHAR。创建源表后，在 database_B 中运行以下命令，以创建其源为 cities 表的实体化视图。确保在 FROM 子句中指定源表的数据库和架构：

```
CREATE MATERIALIZED VIEW cities_mv AS
SELECT  cityname
FROM    database_A.public.cities;
```

查询您创建的实体化视图。该查询检索原始源为 database_A 中的 cities 表的记录：

```
select * from cities_mv;
```

当您运行 SELECT 语句时，cities_mv 会返回记录。只有在运行 REFRESH 语句时，才会刷新源表中的记录。另请注意，您不能直接在实体化视图中更新记录。有关刷新实体化视图中的数据的信息，请参阅REFRESH MATERIALIZED VIEW。

有关实体化视图概述以及用于刷新和删除实体化视图的 SQL 命令的详细信息，请参阅以下主题：

- 在 Amazon Redshift 中创建实体化视图

- REFRESH MATERIALIZED VIEW
- DROP MATERIALIZED VIEW

CREATE MODEL

主题

先决条件

在使用 CREATE MODEL 语句之前，请完成 用于使用 Amazon Redshift ML 的集群设置 中的先决条件。下面简要概述了这些先决条件。

- 使用 AWS 管理控制台或 AWS 命令行界面 (AWS CLI) 创建 Amazon Redshift 集群。
- 在创建集群时附加 AWS Identity and Access Management (IAM) policy。
- 要允许 Amazon Redshift 和 SageMaker 代入角色以便与其他服务交互，请向 IAM 角色添加相应的信任策略。

有关 IAM 角色、信任策略和其他先决条件的详细信息，请参阅用于使用 Amazon Redshift ML 的集群设置。

您可以在下面找到 CREATE MODEL 语句的不同使用案例。

- 简单 CREATE MODEL
- 根据用户指导创建模型
- 带有 AUTO OFF 的 CREATE XGBoost 模型
- 自带模型 (BYOM) – 本地推理
- 带有 K-MANES 的 CREATE MODEL

- Full CREATE MODEL

所需的权限

以下是 CREATE MODEL 所需的权限：

- Superuser
- 具有 CREATE MODEL 权限的用户
- 具有 GRANT CREATE MODEL 权限的角色

成本控制

Amazon Redshift ML 使用现有集群资源创建预测模型，因此您无需支付额外费用。但是，如果您需要调整集群的大小或训练模型，则可能需要支付额外费用。Amazon Redshift ML 使用 Amazon SageMaker 来训练模型，这确实会产生额外的相关费用。可以通过多种方法控制额外成本，例如，限制训练可花费的最长时间，或限制用于训练模型的训练样本数量。有关更多信息，请参阅使用 Amazon Redshift ML 的成本。

Full CREATE MODEL

下面总结了完整的 CREATE MODEL 语法的基本选项。

Full CREATE MODEL 语法

以下是 CREATE MODEL 语句的完整语法。

> **Important**
>
> 使用 CREATE MODEL 语句创建模型时，请按照以下语法中关键词的顺序进行操作。

```
CREATE MODEL model_name
    FROM { table_name | ( select_statement )  | 'job_name' }
    [ TARGET column_name ]
    FUNCTION function_name ( data_type [, ...] )
    [ RETURNS super ]
    IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
    [ AUTO ON / OFF ]
      -- default is AUTO ON
    [ MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER | KMEANS | FORECAST } ]
```

```
      -- not required for non AUTO OFF case, default is the list of all supported types
      -- required for AUTO OFF
   [ PROBLEM_TYPE ( REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION ) ]
      -- not supported when AUTO OFF
   [ OBJECTIVE ( 'MSE' | 'Accuracy' | 'F1' | 'F1_Macro' | 'AUC' |
                 'reg:squarederror' | 'reg:squaredlogerror'| 'reg:logistic'|
                 'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' |
'binary:hinge',
                 'multi:softmax' | 'RMSE' | 'WAPE' | 'MAPE' | 'MASE' |
'AverageWeightedQuantileLoss' ) ]
     -- for AUTO ON: first 5 are valid
     -- for AUTO OFF: 6-13 are valid
     -- for FORECAST: 14-18 are valid
   [ PREPROCESSORS 'string' ]
     -- required for AUTO OFF, when it has to be 'none'
     -- optional for AUTO ON
   [ HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT ( Key 'value' (,...) ) } ]
     -- support XGBoost hyperparameters, except OBJECTIVE
     -- required and only allowed for AUTO OFF
     -- default NUM_ROUND is 100
     -- NUM_CLASS is required if objective is multi:softmax (only possible for AUTO
OFF)
    [ SETTINGS (
      S3_BUCKET 'bucket',  |
        -- required
     TAGS 'string', |
        -- optional
     KMS_KEY_ID 'kms_string', |
        -- optional
     S3_GARBAGE_COLLECT on / off, |
        -- optional, defualt is on.
     MAX_CELLS integer, |
        -- optional, default is 1,000,000
     MAX_RUNTIME integer (, ...) |
        -- optional, default is 5400 (1.5 hours)
     HORIZON integer, |
        -- required if creating a forecast model
     FREQUENCY integer, |
        -- required if creating a forecast model
     PERCENTILES string
        -- optional if creating a forecast model
   ) ]
```

参数

model_name

模型的名称。schema 中的模型名称必须是唯一的。

FROM { table_name | (select_query) | 'job_name'}

指定训练数据的 table_name 或查询。它们可以是系统中的现有表，也可以是用括号（即 ()）括起来的兼容 Amazon RedShift 的 SELECT 查询。查询结果中必须至少有两列。

TARGET column_name

成为预测目标的列的名称。FROM 子句中必须存在该列。

FUNCTION function_name (data_type [, ...])

要创建的函数的名称以及输入参数的数据类型。您可以提供数据库中架构的架构名称而不是函数名称。

RETURNS SUPER（预览版）

要从模型返回的数据类型。返回的 SUPER 数据类型仅适用于远程 BYOM 模型。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

使用默认关键字让 Amazon Redshift 使用设置为默认值并在 CREAT MODEL 命令运行时与集群关联的 IAM 角色。或者，您可以指定 IAM 角色的 ARN 来使用该角色。

[AUTO ON / OFF]

打开或关闭预处理器、算法和超参数选择的 CREATE MODEL 自动发现。在创建预测模型时指定 On 表示使用 AutoPredictor，其中 Amazon Forecast 会将算法的最佳组合应用于数据集中的每个时间序列。

MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER | KMEANS | FORECAST }

（可选）指定模型类型。您可以指定是否要训练特定模型类型的模型，如 XGBoost、多层感知机 (MLP)、KMEANS 或线性学习器，这些都是 Amazon SageMaker Autopilot 支持的算法。如果未指定参数，则在训练期间搜索所有受支持的模型类型，以找到最佳模型。您还可以在 Redshift ML 中创建预测模型，以创建准确的时间序列预测。

PROBLEM_TYPE (REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION)

（可选）指定问题类型。如果您知道问题类型，您可以将 Amazon Redshift 限制为仅搜索该特定模型类型的最佳模型。如果未指定此参数，则会在训练期间根据您的数据发现问题类型。

OBJECTIVE ('MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC' | 'reg:squarederror' | 'reg:squaredlogerror' | 'reg:logistic' | 'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' | 'binary:hinge' | 'multi:softmax' | 'RMSE' | 'WAPE' | 'MAPE' | 'MASE' | 'AverageWeightedQuantileLoss')

（可选）指定用于测量机器学习系统预测质量的目标指标的名称。此指标在训练过程中进行了优化，以便从数据中为模型参数值提供最佳估计值。如果未明确指定指标，则默认行为是自动使用 MSE：用于回归，F1：用于二进制分类，精度：用于多类分类。有关目标的更多信息，请参阅《Amazon SageMaker API 参考》中的 AutoMLJobObjective以及 XGBOOST 文档中的了解任务参数。值 RMSE、WAPE、MAPE、MASE 和 AverageWeightedQuantileLoss 仅适用于预测模型。有关更多信息，请参阅 CreateAutoPredictor API 操作。

PREPROCESSORS 'string'

（可选）将预处理器的某些组合指定为某些列的集合。格式是 columnSet 的列表，以及要应用于每组列的适当转换。Amazon Redshift 将特定转换器列表中的所有转换器应用于相应 ColumnSet 中的所有列。例如，要将带有 Imputer 的 OneHotEncoder 应用于列 t1 和 t2，请使用下面的示例命令。

```
CREATE MODEL customer_churn
FROM customer_data
TARGET 'Churn'
FUNCTION predict_churn
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
PROBLEM_TYPE BINARY_CLASSIFICATION
OBJECTIVE 'F1'
PREPROCESSORS '[
...
  {"ColumnSet": [
      "t1",
      "t2"
    ],
    "Transformers": [
      "OneHotEncoder",
      "Imputer"
    ]
  },
  {"ColumnSet": [
      "t3"
    ],
    "Transformers": [
      "OneHotEncoder"
    ]
  },
  {"ColumnSet": [
```

```
      "temp"
    ],
    "Transformers": [
      "Imputer",
      "NumericPassthrough"
    ]
  }
]'
SETTINGS (
  S3_BUCKET 'bucket'
)
```

HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT (key ‘value’ (,..)) }

指定默认的 XGBoost 参数是被使用还是被用户指定的值覆盖。值必须用单引号引起来。以下是 XGBoost 的参数示例及其默认值。

参数名称	参数值	默认值	备注
num_class	整数	对于多类分类是必需的。	不适用
num_round	整数	100	不适用
tree_method	字符串	自动	不适用
max_depth	整数	6	[0, 10]
min_child_weight	Float	1	最小值：0，最大值：120
subsample	Float	1	最小值：0.5，最大值：1
gamma	Float	0	最小值：0，最大值：5

参数名称	参数值	默认值	备注
alpha	Float	0	最小值：0，最大值：1000
eta	Float	0.3	最小值：0.1，最大值：0.5
colsample_byleve	Float	1	最小值：0.1，最大值：1
colsample_bynode	Float	1	最小值：0.1，最大值：1
colsample_bytree	Float	1	最小值：0.5，最大值：1
lambda	Float	1	最小值：0，最大值：1000
max_delta_step	整数	0	[0, 10]

SETTINGS (S3_BUCKET 'bucket', | TAGS 'string', | KMS_KEY_ID 'kms_string' , | S3_GARBAGE_COLLECT on / off, | MAX_CELLS integer , | MAX_RUNTIME (,...) , | HORIZON integer, | FREQUENCY forecast_frequency, | PERCENTILES array of strings)

S3_BUCKET 子句指定用于存储中间结果的 Amazon S3 位置。

（可选）TAGS 参数是以逗号分隔的键/值对列表，可用于标记在 Amazon SageMaker 和 Amazon Forecast 中创建的资源。标签有助于组织资源和分配成本。键/值对中的值是可选的，因此您可以使用格式 `key=value` 或只是通过创建键来创建标签。有关 Amazon Redshift 中的标签的更多信息，请参阅标记概述。

（可选）KMS_KEY_ID 指定 Amazon Redshift 是否将服务器端加密与 AWS KMS 键结合使用来保护静态数据。传输中的数据由安全套接字层 (SSL) 保护。

（可选）S3_GARBAGE_COLLECT { ON | OFF } 指定 Amazon Redshift 是否对用于训练模型的生成数据集和模型执行垃圾回收。如果设置为 OFF，则用于训练模型的生成数据集和模型将保留在 Amazon S3 中，并可用于其他目的。如果设置为 ON，则 Amazon Redshift 会在训练完成后删除 Amazon S3 中的构件。默认为 ON。

（可选）MAX_CELLS 指定训练数据中的单元格的数量。此值是记录数（在训练查询或表中）乘以列数的乘积。默认值是 1000000。

（可选）MAX_RUNTIME 指定最长训练时间。根据数据集的大小，训练任务通常可以更早完成。这将指定训练所需的最长时间。默认值为 5400（90 分钟）。

HORIZON 指定了预测模型可以返回的最大预测数。模型一旦经过训练，就无法更改此整数。如果训练预测模型，则此参数是必需的。

FREQUENCY 指定了您希望预测以时间为单位的粒度。可用的选项为 `Y | M | W | D | H | 30min | 15min | 10min | 5min | 1min`。如果训练预测模型，则此参数是必需的。

（可选）PERCENTILES 是一个以逗号分隔的字符串，指定用于训练预测器的预测类型。预测类型可以是从 0.01 到 0.99 的分位数，增量为 0.01 或更高。您也可以使用均值指定均值预测。您最多可以指定五种预测类型。

使用说明

使用 CREATE MODEL 时，请注意以下事项：

- CREATE MODEL 语句在异步模式下运行，并在将训练数据导出到 Amazon S3 时返回。Amazon SageMaker 中的其余训练步骤将在后台进行。当训练正在进行时，相应的推理函数可见，但无法运行。您可以查询 STV_ML_MODEL_INFO 以查看训练状态。
- 预设情况下，在 Auto 模型中，训练最多可以在后台运行 90 分钟，并且可以延长。要取消训练，只需运行 DROP MODEL 命令。
- 您用于创建模型的 Amazon Redshift 集群和用于暂存训练数据和模型构件的 Amazon S3 桶必须位于同一 AWS 区域。
- 在模型训练期间，Amazon Redshift 和 SageMaker 将中间构件存储在您提供的 Amazon S3 桶中。默认情况下，Amazon Redshift 会在 CREATE MODEL 操作结束时执行垃圾回收。Amazon Redshift 从 Amazon S3 中删除这些对象。要在 Amazon S3 上保留这些构件，请设置 S3_GARBAGE COLLECT OFF 选项。
- 您必须在 FROM 子句中提供的训练数据中至少使用 500 行。
- 使用 CREATE MODEL 语句时，最多只能在 FROM { table_name | (select_query) } 子句中指定 256 个功能（输入）列。
- 对于 AUTO ON，您可以用作训练集的列类型包括 SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE、BOOLEAN、CHAR、VARCHAR、DAT

和 TIMESTAMPTZ。对于 AUTO OFF，您可以用作训练集的列类型包括 SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE 和 BOOLEAN。

- 不可使用 DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP、TIMESTAMPTZ、GEOMETRY、GEOGRAPHY、HI 或 VARBYTE 作为目标列类型。
- 要提高模型精度，请执行以下操作之一：
  - 在 FROM 子句中指定训练数据时，在 CREATE MODEL 命令中添加尽可能多的相关列。
  - 对于 MAX_RUNTIME 和 MAX_CELLS，请使用较大的值。此参数的值越大，就会增加训练模型的成本。
- 一旦计算训练数据并导出到 Amazon S3 桶后，CREATE MODEL 语句执行就会立即返回。在此之后，您可以使用 SHOW MODEL 命令检查训练的状态。当在后台训练的模型失败时，可以使用 SHOW MODEL 检查错误。您无法重试失败的模型。使用 DROP MODEL 移除失败的模型并重新创建新模型。有关 SHOW MODEL 的更多信息，请参阅SHOW MODEL。
- 本地 BYOM 支持的模型类型与 Amazon Redshift ML 为非 BYOM 案例支持的模型类型相同。Amazon Redshift 支持普通的 XGBoost（使用 XGBoost 版本 1.0 或更高版本），没有预处理器的 KMEANS 模型，以及经过 Amazon SageMaker Autopilot 训练的 XGBOOST/MLP/线性学习器模型。它支持后者与 Autopilot 指定的预处理器，该预处理器也由 Amazon SageMaker Neo 提供支持。
- 如果您的 Amazon Redshift 集群增强了为 Virtual Private Cloud (VPC) 启用的路由，请确保为您的集群所在的 VPC 创建 Amazon S3 VPC 端点和 SageMaker VPC 端点。这样做使得流量可以在 CREATE MODEL 期间通过您的 VPC 在服务之间运行。有关更多信息，请参阅 SageMaker Clarify 作业 Amazon VPC 子网和安全组。

使用案例

以下使用案例演示了如何使用 CREATE MODEL 来满足您的需求。

简单 CREATE MODEL

下面总结了 CREATE MODEL 语法的基本选项。

简单的 CREATE MODEL 语法

```
CREATE MODEL model_name
    FROM { table_name | ( select_query ) }
    TARGET column_name
```

```
  FUNCTION prediction_function_name
  IAM_ROLE { default }
  SETTINGS (
    S3_BUCKET 'bucket',
    [ MAX_CELLS integer ]
  )
```

简单 CREATE MODEL 参数

model_name

模型的名称。schema 中的模型名称必须是唯一的。

FROM { table_name | (select_query) }

指定训练数据的 table_name 或查询。它们可以是系统中的现有表，也可以是用括号（即 () ）括起来的兼容 Amazon RedShift 的 SELECT 查询。查询结果中必须至少有两列。

TARGET column_name

成为预测目标的列的名称。FROM 子句中必须存在该列。

FUNCTION prediction_function_name

一个值，它指定由 CREATE MODEL 生成并用于使用此模型进行预测的 Amazon Redshift 机器学习函数的名称。该函数在与模型对象相同的 schema 中创建，并且可以重载。

Amazon Redshift 机器学习支持模型，例如用于回归和分类的 Xtreme Gradient Boosted 树 (XGBoost) 模型。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

使用默认关键字让 Amazon Redshift 使用设置为默认值并在 CREAT MODEL 命令运行时与集群关联的 IAM 角色。或者，您可以指定 IAM 角色的 ARN 来使用该角色。

S3_BUCKET 'bucket'

您之前创建的 Amazon S3 桶的名称，该桶用于在 Amazon Redshift 和 SageMaker 之间共享训练数据和构件。在卸载训练数据之前，Amazon Redshift 会在此桶中创建一个子文件夹。训练完成后，Amazon Redshift 会删除创建的子文件夹及其内容。

MAX_CELLS 整数

要从 FROM 子句中导出的最大单元格数。默认值为 1000000。

单元格数量是训练数据中的行数（由 FROM 子句表或查询生成）乘以列数的乘积。如果训练数据中的单元格数大于 max_cells 参数指定的单元格数，则 CREATE MODEL 会缩小 FROM 子句训练数据的取样，以减小 MAX_CELLS 下面的训练集的大小。允许更大的训练数据集可以产生更高的精度，但也意味着模型需要更长的时间来训练并且成本更高。

有关使用 Amazon Redshift 的成本的信息，请参阅使用 Amazon Redshift ML 的成本。

有关与各种单元格数量相关的成本免费试用详细信息，请参阅 Amazon Redshift 定价。

根据用户指导创建模型

除了 简单 CREATE MODEL 中所述的选项之外，您还可以在下面找到 CREATE MODEL 选项的描述。

预设情况下，CREATE MODEL 会搜索特定数据集的预处理和模型的最佳组合。您可能需要对模型进行额外的控制或引入其他领域知识（例如问题类型或目标）。在客户流失情况下，如果结果“客户不活跃”很少，则 F1 目标通常优先于精度目标。由于高精度模型可能会始终预测“客户处于活动状态”，因此可以实现高精度，但商业价值却很少。有关 F1 目标的信息，请参阅 Amazon SageMaker API 参考中的 AutoMLJobObjective。

然后，CREATE MODEL 将遵循您对目标等指定方面的建议。同时，CREATE MODEL 会自动发现最佳预处理器和最佳超参数。

使用用户指导语法创建模型

CREATE MODEL 在您可以指定的方面以及 Amazon Redshift 自动发现的方面提供了更大的灵活度。

```
CREATE MODEL model_name
    FROM { table_name | ( select_statement ) }
    TARGET column_name
    FUNCTION function_name
    IAM_ROLE { default }
    [ MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER} ]
    [ PROBLEM_TYPE ( REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION ) ]
    [ OBJECTIVE ( 'MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC') ]
    SETTINGS (
      S3_BUCKET 'bucket', |
      S3_GARBAGE_COLLECT { ON | OFF }, |
      KMS_KEY_ID 'kms_key_id', |
      MAX_CELLS integer, |
      MAX_RUNTIME integer (, ...)
```

```
)
```

使用用户指导参数创建模型

MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER }

（可选）指定模型类型。您可以指定是否要训练特定模型类型的模型，如 XGBoost、多层感知机 (MLP) 或线性学习器，这些是 Amazon SageMaker Autopilot 支持的所有算法。如果未指定参数，则在训练期间搜索所有受支持的模型类型，以找到最佳模型。

PROBLEM_TYPE (REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION)

（可选）指定问题类型。如果您知道问题类型，您可以将 Amazon Redshift 限制为仅搜索该特定模型类型的最佳模型。如果未指定此参数，则会在训练期间根据您的数据发现问题类型。

OBJECTIVE ('MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC')

（可选）指定用于测量机器学习系统预测质量的目标指标的名称。此指标在训练过程中进行了优化，以便从数据中为模型参数值提供最佳估计值。如果未明确指定指标，则默认行为是自动使用 MSE：用于回归，F1：用于二进制分类，精度：用于多类分类。有关目标的更多信息，请参阅 Amazon SageMaker API 参考中的 AutoMLJobObjective。

MAX_CELLS 整数

（可选）指定训练数据中的单元格的数量。此值是记录数（在训练查询或表中）乘以列数的乘积。默认值为 1000000。

MAX_RUNTIME 整数

（可选）指定最长训练时间。根据数据集的大小，训练任务通常可以更早完成。这将指定训练所需的最长时间。默认值为 5400（90 分钟）。

S3_GARBAGE_COLLECT { ON | OFF }

（可选）指定 Amazon Redshift 是否对用于训练模型的生成数据集和模型执行垃圾回收。如果设置为 OFF，则用于训练模型的生成数据集和模型将保留在 Amazon S3 中，并可用于其他目的。如果设置为 ON，则 Amazon Redshift 会在训练完成后删除 Amazon S3 中的构件。默认为 ON。

KMS_KEY_ID 'kms_key_id'

（可选）指定 Amazon Redshift 是否将服务器端加密与 AWS KMS 键结合使用来保护静态数据。传输中的数据由安全套接字层 (SSL) 保护。

PREPROCESSORS 'string'

（可选）将预处理器的某些组合指定为某些列的集合。格式是 columnSet 的列表，以及要应用于每组列的适当转换。Amazon Redshift 将特定转换器列表中的所有转换器应用于相应 ColumnSet 中的所有列。例如，要将带有 Imputer 的 OneHotEncoder 应用于列 t1 和 t2，请使用下面的示例命令。

```
CREATE MODEL customer_churn
FROM customer_data
TARGET 'Churn'
FUNCTION predict_churn
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
PROBLEM_TYPE BINARY_CLASSIFICATION
OBJECTIVE 'F1'
PREPROCESSORS '[
...
{"ColumnSet": [
    "t1",
    "t2"
  ],
  "Transformers": [
    "OneHotEncoder",
    "Imputer"
  ]
},
{"ColumnSet": [
    "t3"
  ],
  "Transformers": [
    "OneHotEncoder"
  ]
},
{"ColumnSet": [
    "temp"
  ],
  "Transformers": [
    "Imputer",
    "NumericPassthrough"
  ]
}
]'
SETTINGS (
S3_BUCKET 'bucket'
)
```

Amazon Redshift 支持以下转换器：

- OneHotEncoder – 通常用于将离散值编码为具有一个非零值的二进制向量。该转换器适用于许多机器学习模型。
- OrdinalEncoder – 将离散值编码为单个整数。该转换器适用于特定机器学习模型，如 MLP 和线性学习器。
- NumericPassthrough – 将输入按原样传递到模型中。
- Imputer – 填充缺少的值，而不是数字 (NaN) 值。
- ImputerWithIndicator – 填充缺少值和 NaN 值。此转换器器还会创建一个指示器，指示是否有任何值缺失以及被填充。
- Normalizer – 标准化值，这可以提高许多机器学习算法的性能。
- DateTimeVectorizer – 创建向量嵌入，表示可在机器学习模型中使用的日期时间数据类型列。
- PCA – 将数据投影到低维空间中，以减少功能数量，同时保留尽可能多的信息。
- StandardScaler – 通过去除平均值并缩放至单位方差来标准化功能。
- MinMax – 通过将每个功能缩放至给定范围来转换功能。

Amazon Redshift ML 存储经过训练的转换器，并将其作为预测查询的一部分自动应用。在从模型生成预测时，您不需要指定它们。

带有 AUTO OFF 的 CREATE XGBoost 模型

AUTO OFF CREATE MODEL 的目标通常与默认的 CREATE MODEL 不同。

作为高级用户，在训练这些模型时便已经知道所需的模型类型和要使用的超参数，因此，可以使用带有 AUTO OFF 的 CREATE MODEL 关闭预处理器和超参数的 CREATE MODEL 自动发现。为此，您可以显式指定模型类型。XGBoost 目前是 AUTO 被设置为 OFF 时支持的唯一模型类型。您可以指定超参数。Amazon Redshift 对您指定的任何超参数使用默认值。

带有 AUTO OFF 语法的 CREATE XGBoost 模型

```
CREATE MODEL model_name
    FROM { table_name | (select_statement ) }
    TARGET column_name
    FUNCTION function_name
    IAM_ROLE { default }
    AUTO OFF
    MODEL_TYPE XGBOOST
    OBJECTIVE { 'reg:squarederror' | 'reg:squaredlogerror' | 'reg:logistic' |
```

```
                'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' |
 'binary:hinge' |
                'multi:softmax' | 'rank:pairwise' | 'rank:ndcg' }
   HYPERPARAMETERS DEFAULT EXCEPT (
       NUM_ROUND '10',
       ETA '0.2',
       NUM_CLASS '10',
       (, ...)
   )
   PREPROCESSORS 'none'
   SETTINGS (
     S3_BUCKET 'bucket', |
     S3_GARBAGE_COLLECT { ON | OFF }, |
     KMS_KEY_ID 'kms_key_id', |
     MAX_CELLS integer, |
     MAX_RUNTIME integer (, ...)
   )
```

使用 AUTO OFF 参数创建 XGBoost 模型

AUTO OFF

关闭预处理器、算法和超参数选择的 CREATE MODEL 自动发现。

MODEL_TYPE XGBOOST

指定使用 XGBOOST 来训练模型。

OBJECTIVE str

指定算法识别的目标。Amazon Redshift 支持 reg:squarederror、reg:squaredlogerror、reg:logistic、reg:pseudohubererror、reg:tweedie、binary:logisti
有关这些目标的更多信息，请参阅 XGBoost 文档中的学习任务参数。

HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT (key ‘value’ (,..)) }

指定默认的 XGBoost 参数是被使用还是被用户指定的值覆盖。值必须用单引号引起来。以下是 XGBoost 的参数示例及其默认值。

参数名称	参数值	默认值	备注
num_class	整数	对于多类	不适用

参数名称	参数值	默认值	备注
		分类是必需的。	
num_round	整数	100	不适用
tree_method	字符串	自动	不适用
max_depth	整数	6	[0, 10]
min_child_weight	Float	1	最小值：0，最大值：120
subsample	Float	1	最小值：0.5，最大值：1
gamma	Float	0	最小值：0，最大值：5
alpha	Float	0	最小值：0，最大值：1000
eta	Float	0.3	最小值：0.1，最大值：0.5
colsample_byleve	Float	1	最小值：0.1，最大值：1
colsample_bynode	Float	1	最小值：0.1，最大值：1
colsample_bytree	Float	1	最小值：0.5，最大值：1
lambda	Float	1	最小值：0，最大值：1000
max_delta_step	整数	0	[0, 10]

以下示例为 XGBoost 准备数据。

```
DROP TABLE IF EXISTS abalone_xgb;

    CREATE TABLE abalone_xgb (
    length_val float,
    diameter float,
    height float,
    whole_weight float,
    shucked_weight float,
    viscera_weight float,
    shell_weight float,
    rings int,
    record_number int);

    COPY abalone_xgb
    FROM 's3://redshift-downloads/redshift-ml/abalone_xg/'
    REGION 'us-east-1'
    IAM_ROLE default
    IGNOREHEADER 1 CSV;
```

以下示例创建具有指定高级选项的 XGBoost 模型，如 MODEL_TYPE、OBJECTIVE 和 PREPROCESSORS。

```
DROP MODEL abalone_xgboost_multi_predict_age;

    CREATE MODEL abalone_xgboost_multi_predict_age
    FROM ( SELECT length_val,
                  diameter,
                  height,
                  whole_weight,
                  shucked_weight,
                  viscera_weight,
                  shell_weight,
                  rings
   FROM abalone_xgb WHERE record_number < 2500 )
    TARGET rings FUNCTION ml_fn_abalone_xgboost_multi_predict_age
    IAM_ROLE default
    AUTO OFF
    MODEL_TYPE XGBOOST
    OBJECTIVE 'multi:softmax'
    PREPROCESSORS 'none'
    HYPERPARAMETERS DEFAULT EXCEPT (NUM_ROUND '100', NUM_CLASS '30')
```

```
    SETTINGS (S3_BUCKET 'your-bucket');
```

以下示例使用推断查询来预测记录编号大于 2500 的鱼的年龄。它使用从上述命令创建的函数 ml_fn_abalone_xgboost_multi_predict_age。

```
select ml_fn_abalone_xgboost_multi_predict_age(length_val,
                                               diameter,
                                               height,
                                               whole_weight,
                                               shucked_weight,
                                               viscera_weight,
                                               shell_weight)+1.5 as age
from abalone_xgb where record_number > 2500;
```

自带模型 (BYOM) – 本地推理

Amazon Redshift ML 支持使用自带模型 (BYOM) 进行本地推理。

下面总结了 BYOM 的 CREATE MODEL 语法的选项。您可以将在 Amazon Redshift 之外训练的模型与 Amazon SageMaker 结合使用，以用于 Amazon Redshift 本地的数据库内推理。

用于本地推理的 CREATE MODEL 语法

下面介绍了用于本地推理的 CREATE MODEL 语法。

```
CREATE MODEL model_name
    FROM ('job_name' | 's3_path' )
    FUNCTION function_name ( data_type [, ...] )
    RETURNS data_type
    IAM_ROLE { default }
    [ SETTINGS (
      S3_BUCKET 'bucket', | --required
      KMS_KEY_ID 'kms_string') --optional
    ];
```

Amazon Redshift 目前仅支持针对 BYOM 的预先训练的 XGBoost、MLP 和线性学习器模型。您可以使用此路径导入 SageMaker Autopilot 和直接在 Amazon SageMaker 中训练的模型，以便进行本地推理。

用于本地推理的 CREATE MODEL 参数

model_name

模型的名称。schema 中的模型名称必须是唯一的。

FROM ('job_name' | 's3_path')

job_name 将 Amazon SageMaker 任务名称用作输入。任务名称可以是 Amazon SageMaker 训练任务名称，也可以是 Amazon SageMaker Autopilot 任务名称。任务必须在拥有 Amazon Redshift 集群的相同AWS账户中创建。要查找作业名称，请启动 Amazon SageMaker。在训练下拉菜单中，选择训练作业。

's3_path' 指定创建模型时要使用的 .tar.gz 模型构件文件的 S3 位置。

FUNCTION function_name (data_type [, ...])

要创建的函数的名称以及输入参数的数据类型。您可以提供 schema 名称。

RETURNS data_type

函数返回的值的数据类型。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}

使用默认关键字让 Amazon Redshift 使用 IAM 角色，该角色设置为默认值并在 CREATE MODEL 命令运行时与集群关联。

使用 IAM 角色的 Amazon 资源名称（ARN），您的集群使用该角色进行身份验证和授权。

SETTINGS (S3_BUCKET 'bucket', | KMS_KEY_ID 'kms_string')

S3_BUCKET 子句指定用于存储中间结果的 Amazon S3 位置。

（可选）KMS_KEY_ID 子句指定 Amazon Redshift 是否将服务器端加密与 AWS KMS 键结合使用来保护静态数据。传输中的数据由安全套接字层 (SSL) 保护。

有关更多信息，请参阅 根据用户指导创建模型。

用于本地推理的 CREATE MODEL 语法示例

以下示例创建之前在 Amazon Redshift 之外的 Amazon SageMaker 中训练过的模型。由于 Amazon Redshift ML 支持模型类型进行本地推理，因此以下 CREATE MODEL 将创建一个可在 Amazon Redshift 中本地使用的函数。您可以提供 SageMaker 训练任务名称。

```
CREATE MODEL customer_churn
    FROM 'training-job-customer-churn-v4'
    FUNCTION customer_churn_predict (varchar, int, float, float)
    RETURNS int
    IAM_ROLE default
    SETTINGS (S3_BUCKET 'your-bucket');
```

创建模型后，您可以将函数 customer_churn_predict 与指定参数类型结合使用以进行预测。

自带模型 (BYOM) – 远程推理

Amazon Redshift ML 还支持使用自带模型 (BYOM) 进行远程推理。

下面总结了 BYOM 的 CREATE MODEL 语法的选项。

⚠ 以下是针对预览版 Amazon Redshift ML 中 BYOM 模型的输入的 SUPER 数据类型的预发行文档。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

指定使用 SUPER 数据类型作为输入数据和返回数据类型，表示您要创建一个托管在 Amazon SageMaker JumpStart 中的大型语言模型 (LLM)。创建 LLM 目前只能作为预览功能使用。在以下 AWS 区域中提供此预览。

- 美国东部（俄亥俄州）(us-east-2)
- 美国东部（弗吉尼亚州北部）(us-east-1)
- 亚太地区（东京）(ap-northeast-1)
- 欧洲地区（爱尔兰）(eu-west-1)
- 欧洲地区（斯德哥尔摩）(eu-north-1)

您可以在预览版中创建 Amazon Redshift 集群，以便测试 Amazon Redshift 的新功能。您无法在生产环境中使用这些功能，也无法将预览版集群移动到生产集群或另一个跟踪上的集群。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

在预览版中创建集群

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。

2. 在导航菜单上，选择预置集群控制面板，然后选择集群。列出您的账户在当前 AWS 区域 区域中的集群。列表中的各个列中显示了每个集群的一部分属性。
3. 集群列表页面上会显示一个横幅，其中介绍了预览版。选择创建预览版集群按钮以打开创建集群页面。
4. 输入集群的属性。选择包含要测试的功能的预览版跟踪。我们建议输入的集群名称指明要对该集群进行预览版跟踪。为您的集群选择选项，包括标记为 -preview 的选项，用于要测试的功能。有关创建集群的一般信息，请参阅《Amazon Redshift 管理指南》中的创建集群。
5. 选择创建集群以在预览模式下创建集群。

> **Note**
>
> preview_2023 跟踪是最新可用的预览版跟踪。此版本仅支持创建具有 RA3 节点类型的集群。不支持节点类型 DC2 以及任何更早的节点类型。

6. 当您的预览集群可用时，使用 SQL 客户端加载和查询数据。

您也可以创建预览工作组来创建 LLM。您无法在生产中使用这些功能，也无法将您的工作组移至其他工作组。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。有关如何创建预览工作组的说明，请参阅创建预览工作组。

用于远程推理的 CREATE MODEL 语法

下面介绍了用于远程推理的 CREATE MODEL 语法。

```
CREATE MODEL model_name
    FUNCTION function_name ( data_type [, ...] )
    RETURNS data_type
    SAGEMAKER 'endpoint_name'[:'model_name']
    IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
```

用于远程推理的 CREATE MODEL 参数

model_name

模型的名称。schema 中的模型名称必须是唯一的。

FUNCTION fn_name ([data_type] [, ...])

函数的名称和输入参数的数据类型。有关所有支持的数据类型，请参阅数据类型。Geography、geometry 和 hllsketch 不受支持。指定使用 SUPER 数据类型作为输入数

据和返回数据类型，表示您要创建一个托管在 Amazon SageMaker JumpStart 中的大型语言模型 (LLM)。

或者，您可以指定仅将 SUPER 数据类型用作输入数据，而不将其同时用作返回的数据类型。使用 SUPER 数据类型作为输入只可用作预览特征。

您也可以提供架构名称而不是函数名称。

RETURNS data_type

函数返回的值的数据类型。有关所有支持的数据类型，请参阅数据类型。Geography、geometry 和 hllsketch 不受支持。指定使用 SUPER 数据类型作为输入数据和返回数据类型，表示您要创建一个托管在 Amazon SageMaker JumpStart 中的大型语言模型 (LLM)。

或者，您可以指定仅将 SUPER 数据类型用作返回的数据类型，而不将其同时用作输入数据。

SAGEMAKER 'endpoint_name'[:'model_name']

Amazon SageMaker 端点的名称。如果端点名称指向多模型端点，请添加要使用的模型名称。端点必须与 Amazon Redshift 集群托管于同一 AWS 区域。要查找端点，请启动 Amazon SageMaker。在推理下拉菜单中，选择端点。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}

使用默认关键字让 Amazon Redshift 使用 IAM 角色，该角色设置为默认值并在 CREATE MODEL 命令运行时与集群关联。或者，您可以指定 IAM 角色的 ARN 来使用该角色。

当模型部署到 SageMaker 端点时，SageMaker 会在 Amazon Redshift 中创建模型的信息。然后它通过外部函数执行推理。您可以使用 SHOW MODEL 命令查看 Amazon Redshift 集群上的模型信息。

用于远程推理的 CREATE MODEL 使用说明

在使用 CREATE MODEL 进行远程推理之前，请考虑以下事项：

- 如果您使用 SUPER 数据类型作为输入数据，BYOM 模型只能支持一个参数，并且返回的输出也必须是 SUPER 数据类型。
- 模型必须通过 SageMaker 中的文本/CSV 内容类型接受逗号分隔值 (CSV) 格式的输入。仅当您未使用 SUPER 数据类型作为输入时才适用。
- 端点必须由拥有 Amazon Redshift 集群的相同AWS账户中托管。
- 模型的输出必须是创建函数时指定的类型的单个值，格式为逗号分隔值 (CSV)，方法是通过 SageMaker 中的文本/CSV 的内容类型。Varchar 数据类型不应包含在引号中，并且每个输出都必须位于新行中。仅在您指定模型不应返回 SUPER 数据类型时才适用。

- 模型接受 null 值作为空字符串。
- 确保 Amazon SageMaker 端点有足够的资源来容纳来自 Amazon Redshift 的推理调用，或者可以自动扩展 Amazon SageMaker 端点。
- 当返回的类型为 SUPER 时，模型输出必须为 JSON 和 `application/jsonlines` 内容类型。
- 当输入和输出类型均为 SUPER 时，模型必须通过内容类型 `application/json` 接受并返回 JSON。

用于远程推理的 CREATE MODEL 语法示例

以下示例创建一个使用 SageMaker 端点进行预测的模型。确保端点正在运行以进行预测，并在 CREATE MODEL 命令中指定其名称。

```
CREATE MODEL remote_customer_churn
    FUNCTION remote_fn_customer_churn_predict (varchar, int, float, float)
    RETURNS int
    SAGEMAKER 'customer-churn-endpoint'
    IAM_ROLE default;
```

以下示例使用 SUPER 数据类型作为输入数据创建大型语言模型 (LLM)，并输出 SUPER 数据类型。LLM 由 SageMaker Jumpstart 托管。

```
CREATE MODEL sample_super_data_model
FUNCTION sample_super_data_model_predict(super)
RETURNS super
SAGEMAKER 'sample_super_data_model_endpoint'
IAM_ROLE default;
```

带有 K-MANES 的 CREATE MODEL

Amazon Redshift 支持 K-Means 算法，该算法可对未标记的数据进行分组。此算法可解决需要在数据中发现分组的集群问题。根据未分类数据的相似与不同之处进行分组和分区。

带有 K-MANS 语法的 CREATE MODEL

```
CREATE MODEL model_name
    FROM { table_name | ( select_statement ) }
    FUNCTION function_name
    IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
    AUTO OFF
```

```
    MODEL_TYPE KMEANS
    PREPROCESSORS 'string'
    HYPERPARAMETERS DEFAULT EXCEPT ( K 'val' [, ...] )
    SETTINGS (
      S3_BUCKET 'bucket',
      KMS_KEY_ID 'kms_string', |
        -- optional
      S3_GARBAGE_COLLECT on / off, |
        -- optional
      MAX_CELLS integer, |
        -- optional
      MAX_RUNTIME integer
        -- optional);
```

带有 K-MANES 参数的 CREATE MODEL

AUTO OFF

关闭预处理器、算法和超参数选择的 CREATE MODEL 自动发现。

MODEL_TYPE KMEANS

指定使用 KMEANS 来训练模型。

PREPROCESSORS 'string'

将预处理器的某些组合指定为某些列的集合。格式是 columnSet 的列表，以及要应用于每组列的适当转换。Amazon Redshift 支持 3 个 K-Means 预处理器，即 StandardScaler、MinMax 和 NumericPassthrough。如果您不想对 K-Means 应用任何预处理，请明确选择 NumericPassthrough 作为转换器。有关支持的转换器的更多信息，请参阅使用用户指导参数创建模型。

K-Means 算法使用欧氏距离来计算相似度。对数据进行预处理可确保模型的功能保持在同等级别并生成可靠的结果。

HYPERPARAMETERS DEFAULT EXCEPT (K 'val' [, ...])

指定是否使用 K-Means 参数。使用 K-Means 算法，必须指定 K 参数。有关更多信息，请参阅《Amazon SageMaker 开发人员指南》中的 K-Means 超参数。

以下示例为 K-Means 准备数据。

```
CREATE MODEL customers_clusters
FROM customers
```

```
FUNCTION customers_cluster
IAM_ROLE default
AUTO OFF
MODEL_TYPE KMEANS
PREPROCESSORS '[
{
  "ColumnSet": [ "*" ],
  "Transformers": [ "NumericPassthrough" ]
}
]'
HYPERPARAMETERS DEFAULT EXCEPT ( K '5' )
SETTINGS (S3_BUCKET 'bucket');

select customer_id, customers_cluster(...) from customers;
customer_id | customers_cluster
--------------------
12345            1
12346            2
12347            4
12348            0
```

CREATE MODEL with Forecast

Redshift ML 中的预测模型使用 Amazon Forecast 来创建准确的时间序列预测。这样做可以让您使用一段时间内的历史数据来就未来的事件进行预测。Amazon Forecast 的常见使用案例包括：使用零售产品数据来决定如何为库存定价，使用制造数量数据来预测一件商品的订购量，以及使用 Web 流量数据来预测 Web 服务器可能收到多少流量。

Amazon Forecast 中的配额限制在 Amazon Redshift 预测模型中执行。例如，最大预测数为 100，但该数量是可调整的。删除预测模型不会自动删除 Amazon Forecast 中的关联资源。如果您删除 Redshift 集群，则所有关联模型也会一并删除。

请注意，预测模型目前仅在以下区域中可用：

- 美国东部（俄亥俄州）(us-east-2)
- 美国东部（弗吉尼亚北部）(us-east-1)
- 美国西部（俄勒冈州）(us-west-2)
- 亚太地区（孟买）(ap-south-1)
- 亚太地区（首尔）(ap-northeast-2)
- 亚太地区（新加坡）(ap-southeast-1)

- 亚太地区（悉尼）(ap-southeast-2)
- 亚太地区（东京）(ap-northeast-1)
- 欧洲地区（法兰克福）(eu-central-1)
- 欧洲地区（爱尔兰）(eu-west-1)

CREATE MODEL with Forecast 语法

```
CREATE [ OR REPLACE ] MODEL forecast_model_name
FROM { table_name | ( select_query ) }
TARGET column_name
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}
AUTO ON
MODEL_TYPE FORECAST
SETTINGS (
  S3_BUCKET 'bucket',
  HORIZON integer,
  FREQUENCY forecast_frequency
  [PERCENTILES '0.1', '0.5', '0.9']
```

CREATE MODEL with Forecast 参数

forecast_model_name

模型的名称。模型名称必须唯一。

FROM { table_name | (select_query) }

指定训练数据的 table_name 或查询。这既可以是系统中的现有表，也可以是用括号括起来的兼容 Amazon RedShift 的 SELECT 查询。表或查询结果必须至少包含三列：(1) 一个指定时间序列名称的 varchar 列。每个数据集可以有多个时间序列；(2) 一个日期时间列；以及 (3) 要预测的目标列。此目标列必须为整数或浮点类型。如果您提供的数据集包含三列以上，Amazon Redshift 会假定所有其他列都是相关时间序列的一部分。请注意，相关时间序列必须为整数或浮点类型。有关相关时间序列的更多信息，请参阅使用相关时间序列数据集。

TARGET column_name

成为预测目标的列的名称。FROM 子句中必须存在该列。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

使用默认关键字让 Amazon Redshift 使用设置为默认值并在 CREAT MODEL 命令运行时与集群关联的 IAM 角色。或者，您可以指定 IAM 角色的 ARN 来使用该角色。

AUTO ON

打开算法和超参数选择的 CREATE MODEL 自动发现。在创建预测模型时指定 On 表示使用 Forecast AutoPredictor，其中 Amazon Forecast 会将算法的最佳组合应用于数据集中的每个时间序列。

MODEL_TYPE FORECAST

指定使用 FORECAST 来训练模型。

S3_BUCKET 'bucket'

您之前创建的 Amazon Simple Storage Service 桶的名称，该桶用于在 Amazon Redshift 和 Amazon Forecast 之间共享训练数据和构件。在卸载训练数据之前，Amazon Redshift 会在此桶中创建一个子文件夹。训练完成后，Amazon Redshift 会删除创建的子文件夹及其内容。

HORIZON 整数

预测模型可以返回的最大预测数。模型一旦经过训练，您就无法更改此整数。

FREQUENCY forecast_frequency

指定您希望的预测时间粒度。可用的选项为 `Y | M | W | D | H | 30min | 15min | 10min | 5min | 1min`。如果要训练预测模型，则为必填项。

PERCENTILES 字符串

一个以逗号分隔的字符串，指定用于训练预测器的预测类型。预测类型可以是从 0.01 到 0.99 的分位数，增量为 0.01 或更高。您也可以使用均值指定均值预测。您最多可以指定五种预测类型。

以下示例演示了如何创建简单的预测模型。

```
CREATE MODEL forecast_example
FROM forecast_electricity_
TARGET target
IAM_ROLE 'arn:aws:iam::<account-id>:role/<role-name>'
AUTO ON
MODEL_TYPE FORECAST
SETTINGS (S3_BUCKET 'redshift-ml-bucket',
          HORIZON 24,
          FREQUENCY 'H',
          PERCENTILES '0.25,0.50,0.75,mean',
          S3_GARBAGE_COLLECT OFF);
```

创建预测模型后，您可以使用预测数据创建新表。

```
CREATE TABLE forecast_model_results as SELECT Forecast(forecast_example)
```

然后，您可以查询新表以获得预测。

```
SELECT * FROM forecast_model_results
```

CREATE PROCEDURE

创建新的存储过程或者替换当前数据库的现有过程。

有关更多信息以及示例，请参阅 在 Amazon Redshift 中创建存储过程。

所需的权限

您必须通过以下方式之一获得权限，才能运行 CREATE OR REPLACE PROCEDURE：

- CREATE PROCEDURE：
  - Superuser
  - 对创建存储过程的架构具有 CREATE 和 USAGE 权限的用户
- REPLACE PROCEDURE：
  - Superuser
  - 程序拥有者

语法

```
CREATE [ OR REPLACE ] PROCEDURE sp_procedure_name
  ( [ [ argname ] [ argmode ] argtype [, ...] ] )
[ NONATOMIC ]
AS $$
  procedure_body
$$ LANGUAGE plpgsql
[ { SECURITY INVOKER | SECURITY DEFINER } ]
[ SET configuration_parameter { TO value | = value } ]
```

参数

OR REPLACE

一个子句，指定如果某个过程与已存在的此过程具有相同的名称和输入参数数据类型或签名，则替换现有的过程。您只能将某个过程替换为定义一组相同数据类型的新过程。

如果您定义的过程与现有过程具有相同的名称，但签名不同，则您将创建新的过程。换句话说，过程名称已重载。有关更多信息，请参阅 重载过程名称。

sp_procedure_name

过程的名称。如果您指定 schema 名称（例如 **`myschema.myprocedure`**），则在指定的 schema 中创建该过程。否则，将在当前 schema 中创建过程。有关有效名称的更多信息，请参阅名称和标识符。

我们建议您将所有的存储过程名称添加前缀 `sp_`。Amazon Redshift 保留 `sp_` 前缀，用于存储过程名称。通过使用前缀 `sp_`，可以确保存储过程名称不会与任何现有或将来的 Amazon Redshift 内置存储过程或函数名称冲突。有关更多信息，请参阅 命名存储过程。

如果输入参数的数据类型或签名不同，您可以定义多个具有相同名称的过程。换句话说，在这种情况下过程名称会重载。有关更多信息，请参阅重载过程名称

[argname] [argmode] argtype

参数名称、参数模式和数据类型的列表。仅需要数据类型。名称和模式是可选的，可以切换它们的位置。

参数模式可以是 IN、OUT 或 INOUT。默认值为 IN。

您可以使用 OUT 和 INOUT 参数从过程调用中返回一个或多个值。当存在 OUT 或 INOUT 参数时，过程调用返回一个包含 n 列的结果行，其中 n 是 OUT 或 INOUT 参数的总数。

INOUT 参数同时是输入和输出参数。输入参数 包括 IN 和 INOUT 参数，而输出参数 包括 OUT 和 INOUT 参数。

OUT 参数未指定为 CALL 语句的一部分。在存储过程 CALL 语句中指定 INOUT 参数。当从嵌套调用传递和返回值时，以及返回 `refcursor` 时，INOUT 参数很有用。有关 `refcursor` 类型的更多信息，请参阅 游标。

参数数据类型可以是任何标准的 Amazon Redshift 数据类型。另外，参数数据类型可以是 `refcursor`。

您最多可以指定 32 个输入参数和 32 个输出参数。

AS $$ procedure_body $$

包含要执行的过程的构造。需要文字关键字 AS $$ 和 $$。

Amazon Redshift 要求您使用称为“美元引号”的格式，在您的过程中包含语句。包含的任何内容将按原样传递。您不必对任何特殊字符进行转义，因为字符串的内容是按照其字面涵义编写的。

通过美元引号 格式，您可以使用一对美元符号 ($$) 来指示要运行的语句的开头和结尾，如以下示例所示。

```
$$ my statement $$
```

（可选）在每对美元符号之间，可以指定字符串来帮助识别语句。您使用的字符串必须在括起字符对的开始和结束都是相同的。该字符串区分大小写，它遵循与不带括起字符的标识符相同的约束，但有一点除外，它不能包含美元符号。以下示例使用字符串 test。

```
$test$ my statement $test$
```

此语法对嵌套的美元引号也很有用。有关“美元引号”格式的更多信息，请参阅 PostgreSQL 文档的词法结构中的“使用美元符号括起的常量字符串”。

procedure_body

一组有效的 PL/pgSQL 语句。PL/pgSQL 语句使用程序性结构（包括循环和条件表达式）增强 SQL 命令，以控制逻辑流。可以在过程主体中使用大部分 SQL 命令，包括数据修改语言 (DML)（例如 COPY、UNLOAD 和 INSERT）以及数据定义语言 (DDL)（例如 CREATE TABLE）。有关更多信息，请参阅 PL/pgSQL 语言参考。

LANGUAGE plpgsql

语言值。指定 `plpgsql`。您必须具有使用 `plpgsql` 语言的权限。有关更多信息，请参阅 GRANT。

NONATOMIC

在非原子事务模式下创建存储过程。NONATOMIC 模式会自动提交过程内部的语句。此外，当 NONATOMIC 过程内部出现错误时，如果错误由异常块处理，则不会重新引发错误。有关更多信息，请参阅管理事务 和RAISE。

当您将存储过程定义为 `NONATOMIC` 时，请考虑以下各项：

- 当您进行嵌套的存储过程调用时，所有过程都必须在相同的事务模式下创建。
- 在 NONATOMIC 模式下创建过程时，不支持 SECURITY DEFINER 选项和 SET configuration_parameter 选项。
- 任何打开的游标（显式或隐式）在处理隐式提交时会自动关闭。因此，您必须在开始游标循环之前打开显式事务，以确保不会隐式提交循环迭代中的任何 SQL。

SECURITY INVOKER | SECURITY DEFINER

指定 NONATOMIC 时不支持 SECURITY DEFINER 选项。

该过程的安全模式确定过程在运行时的访问权限。该过程必须具有访问基础数据库对象的权限。

对于 SECURITY INVOKER 模式，该过程使用调用该过程的用户的权限。用户必须对基础数据库对象具有显式权限。默认值为 SECURITY INVOKER。

对于 SECURITY DEFINER 模式，此过程使用过程拥有者的权限。过程拥有者定义为在运行时拥有此过程的用户，而不一定是最初定义此过程的用户。调用该过程的用户需要具有该过程的执行权限，但不需要对基础对象具有任何权限。

SET configuration_parameter { TO value | = value }

指定 NONATOMIC 时不支持这些选项。

输入过程时，SET 子句会将指定的 configuration_parameter 设置为指定的值。然后，当该过程退出时，该子句会将 configuration_parameter 还原为其早期值。

使用说明

如果存储过程是使用 SECURITY DEFINER 选项创建的，则在存储过程中调用 CURRENT_USER 函数时，Amazon Redshift 会返回存储过程拥有者的用户名。

示例

> **Note**
>
> 如果在运行这些示例时，您遇到了类似于下文的错误：
>
> ```
> ERROR: 42601: [Amazon](500310) unterminated dollar-quoted string at or near "$$
> ```
>
> 请参阅 Amazon Redshift 中的存储过程概览。

以下示例创建带有两个输入参数的过程。

```
CREATE OR REPLACE PROCEDURE test_sp1(f1 int, f2 varchar(20))
AS $$
DECLARE
  min_val int;
BEGIN
  DROP TABLE IF EXISTS tmp_tbl;
  CREATE TEMP TABLE tmp_tbl(id int);
  INSERT INTO tmp_tbl values (f1),(10001),(10002);
  SELECT INTO min_val MIN(id) FROM tmp_tbl;
  RAISE INFO 'min_val = %, f2 = %', min_val, f2;
END;
$$ LANGUAGE plpgsql;
```

> **Note**
>
> 在编写存储过程时，我们建议使用最佳实践来保护敏感值：
> 不要在存储过程逻辑中对任何敏感信息进行硬编码。例如，不要在存储过程主体的 CREATE USER 语句中分配用户密码。这会带来安全风险，因为硬编码值可以作为架构元数据记录在目录表中。而是应通过参数将诸如密码之类的敏感值作为参量传递给存储过程。
> 有关存储过程的更多信息，请参阅 CREATE PROCEDURE 和在 Amazon Redshift 中创建存储过程。有关目录表的更多信息，请参阅系统目录表。

以下示例使用一个 IN 参数、一个 OUT 参数和一个 INOUT 参数创建一个过程。

```
CREATE OR REPLACE PROCEDURE test_sp2(f1 IN int, f2 INOUT varchar(256), out_var OUT
 varchar(256))
AS $$
DECLARE
  loop_var int;
BEGIN
  IF f1 is null OR f2 is null THEN
    RAISE EXCEPTION 'input cannot be null';
  END IF;
  DROP TABLE if exists my_etl;
  CREATE TEMP TABLE my_etl(a int, b varchar);
    FOR loop_var IN 1..f1 LOOP
        insert into my_etl values (loop_var, f2);
        f2 := f2 || '+' || f2;
```

```
    END LOOP;
  SELECT INTO out_var count(*) from my_etl;
END;
$$ LANGUAGE plpgsql;
```

CREATE RLS POLICY

创建新的行级安全策略以提供对数据库对象的精细访问。

超级用户和具有 sys:secadmin 角色的用户或角色可以创建策略。

语法

```
CREATE RLS POLICY policy_name
[ WITH (column_name data_type [, ...]) [ [AS] relation_alias ] ]
USING ( using_predicate_exp )
```

参数

policy_name

策略的名称。

WITH (column_name data_type [, ...])

指定 column_name 和 data_type 引用策略附加到的表的列。

仅当 RLS 策略没有引用策略附加到的表的任何列时，您才可以省略 WITH 子句。

AS relation_alias

为 RLS 策略将附加到的表指定可选别名。

USING (using_predicate_exp)

指定应用于查询的 WHERE 子句的筛选器。Amazon Redshift 会在查询级别的用户谓词之前应用策略谓词。例如，**`current_user = 'joe' and price > 10`** 限制 Joe 只能查看价格高于 10 美元的记录。

使用说明

在使用 CREATE RLS POLICY 语句时，请遵守以下规则：

- Amazon Redshift 支持可以作为查询的 WHERE 子句的组成部分的筛选器。

- 所有附加到表的策略都必须使用相同的表别名创建。
- 您不需要对查找表的 SELECT 权限。当您创建策略时，Amazon Redshift 会授予对相应策略的查找表的 SELECT 权限。查找表是在策略定义中使用的表对象。
- Amazon Redshift 行级安全性不支持策略定义中的以下对象类型：目录表、跨数据库关系、外部表、常规视图、后期绑定视图、已开启 RLS 策略的表以及临时表。

示例

以下 SQL 语句为 CREATE RLS POLICY 示例创建了表、用户和角色。

```
-- Create users and roles reference in the policy statements.
CREATE ROLE analyst;

CREATE ROLE consumer;

CREATE USER bob WITH PASSWORD 'Name_is_bob_1';

CREATE USER alice WITH PASSWORD 'Name_is_alice_1';

CREATE USER joe WITH PASSWORD 'Name_is_joe_1';

GRANT ROLE sys:secadmin TO bob;

GRANT ROLE analyst TO alice;

GRANT ROLE consumer TO joe;

GRANT ALL ON TABLE tickit_category_redshift TO PUBLIC;
```

以下示例创建了一个名为 policy_concerts 的策略。

```
CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');
```

CREATE ROLE

创建一个新的自定义角色，该角色是权限的集合。有关 Amazon Redshift 系统定义的角色列表，请参阅the section called “Amazon Redshift 系统定义的角色”。查询 SVV_ROLES 以查看集群或工作组中当前创建的角色。

可以创建的角色数量有配额。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 中的配额和限制。

所需的权限

以下是 CREATE ROLE 所需的权限：

- Superuser
- 具有 CREATE ROLE 权限的用户

语法

```
CREATE ROLE role_name
[ EXTERNALID external_id ]
```

参数

role_name

角色的名称。角色名称必须唯一且不能与任何用户名相同。角色名称不能为保留字。

超级用户或具有 CREATE ROLE 权限的普通用户可以创建角色。如果用户不是超级用户，但已被授予该角色的 USAGE 权限及 WITH GRANT OPTION 选项以及 ALTER 权限，则可以将此角色授予任何人。

EXTERNALID external_id

角色的标识符，与身份提供者关联。有关更多信息，请参阅 Amazon Redshift 的原生身份提供者 (IdP) 联合身份验证。

示例

下面的示例将创建角色 `sample_role1`。

```
CREATE ROLE sample_role1;
```

以下示例创建角色 `sample_role1`，其中包含与身份提供者关联的外部 ID。

```
CREATE ROLE sample_role1 EXTERNALID "ABC123";
```

CREATE SCHEMA

定义当前数据库的新 schema。

所需的权限

以下是 CREATE SCHEMA 所需的权限：

- Superuser
- 具有 CREATE SCHEMA 权限的用户

语法

```
CREATE SCHEMA [ IF NOT EXISTS ] schema_name [ AUTHORIZATION username ]
           [ QUOTA {quota [MB | GB | TB] | UNLIMITED} ] [ schema_element [ ... ]

CREATE SCHEMA AUTHORIZATION username[ QUOTA {quota [MB | GB | TB] | UNLIMITED} ]
 [ schema_element [ ... ] ]
```

参数

IF NOT EXISTS

这个子句指示，如果指定的 schema 已存在，则命令不应进行任何更改，并返回一条指示 schema 存在的消息，而不是以错误终止。

此子句在编写脚本时很有用，可使脚本在 CREATE SCHEMA 尝试创建已存在的 schema 时不会失败。

schema_name

新 schema 的名称。schema 名称不能为 PUBLIC。有关有效名称的更多信息，请参阅名称和标识符。

> **Note**
>
> search_path 配置参数中的 schema 列表确定了在不使用 schema 名称的情况下引用同名对象的优先顺序。

AUTHORIZATION

一个向指定的用户提供所有权的子句。

username

schema 所有者的名称。

schema_element

要在 schema 中创建的一个或多个对象的定义。

QUOTA

指定的 schema 可以使用的最大磁盘空间量。此空间是整体磁盘使用情况。它包括所有永久表、指定 schema 下的实体化视图以及在每个计算节点上具有 ALL 分布的所有表的重复副本。schema 配额不考虑作为临时命名空间或 schema 的一部分创建的临时表。

要查看已配置的 schema 配额，请参阅SVV_SCHEMA_QUOTA_STATE。

要查看已超出 schema 配额的记录，请参阅STL_SCHEMA_QUOTA_VIOLATIONS。

Amazon Redshift 将选定值转换为 MB。如果您未指定值，则 GB 是默认的测量单位。

您必须是数据库超级用户才能设置和更改 schema 配额。不具有超级用户身份但具有 CREATE SCHEMA 权限的用户可以创建具有定义的配额的 schema。如果创建一个 schema 而不定义配额，则该 schema 具有无限的配额。如果将配额设置为低于 schema 所使用的当前值，则在您释放磁盘空间之前，Amazon Redshift 将不允许进一步摄入。DELETE 语句从表中删除数据，并且仅当 VACUUM 运行时才释放磁盘空间。

Amazon Redshift 会在提交事务之前检查每个事务是否存在违反配额的情况。Amazon Redshift 会根据设置的配额检查每个修改后的 schema 的大小（schema 中所有表使用的磁盘空间）。由于配额冲突检查是在事务结束时进行的，因此，大小限制可能会在事务被提交之前暂时超过事务配额。当事务超出配额时，Amazon Redshift 会停止事务，禁止后续提取，并恢复所有更改，直到您释放磁盘空间。由于后台 VACUUM 和内部清理，在取消事务后检查架构时，架构可能不完整。

作为例外，Amazon Redshift 会忽略配额违规并在某些情况下提交事务。Amazon Redshift 会针对仅由以下一个或多个语句组成的事务执行此操作，而在同一事务中没有 INSERT 或 COPY 摄入语句：

- DELETE
- TRUNCATE

- VACUUM
- DROP TABLE
- ALTER TABLE APPEND 仅在将数据从完整 schema 移至非完整 schema 时使用

UNLIMITED

Amazon Redshift 不对 schema 的总大小的增长施加任何限制。

限制

Amazon Redshift 针对 schema 强制实施以下限制。

- 每个数据库最多有 9900 个 schemas。

示例

以下示例创建一个名为 US_SALES 的 schema 并向用户 DWUSER 授予所有权。

```
create schema us_sales authorization dwuser;
```

以下示例创建一个名为 US_SALES 的 schema，向用户 DWUSER 授予所有权，并将配额设置为 50 GB。

```
create schema us_sales authorization dwuser QUOTA 50 GB;
```

要查看新 schema，请查询 PG_NAMESPACE 目录表，如下所示。

```
select nspname as schema, usename as owner
from pg_namespace, pg_user
where pg_namespace.nspowner = pg_user.usesysid
and pg_user.usename ='dwuser';

   schema |  owner
----------+----------
 us_sales | dwuser
(1 row)
```

以下示例创建 US_SALES schema，如果 schema 已存在，将不执行任何操作并返回一条消息。

```
create schema if not exists us_sales;
```

CREATE TABLE

在当前数据库中创建一个新表。您可以定义一个列的列表，用于存储不同类型的数据。此表的所有者为 CREATE TABLE 命令的发布者。

所需的权限

以下是 CREATE TABLE 所需的权限：

- Superuser
- 具有 CREATE TABLE 权限的用户

语法

```
CREATE [ [LOCAL ] { TEMPORARY | TEMP } ] TABLE
[ IF NOT EXISTS ] table_name
( { column_name data_type [column_attributes] [ column_constraints ]
  | table_constraints
  | LIKE parent_table [ { INCLUDING | EXCLUDING } DEFAULTS ] }
  [, ... ]  )
[ BACKUP { YES | NO } ]
[table_attributes]

where column_attributes are:
  [ DEFAULT default_expr ]
  [ IDENTITY ( seed, step ) ]
  [ GENERATED BY DEFAULT AS IDENTITY ( seed, step ) ]
  [ ENCODE encoding ]
  [ DISTKEY ]
  [ SORTKEY ]
  [ COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE  ]

and column_constraints are:
  [ { NOT NULL | NULL } ]
  [ { UNIQUE  |  PRIMARY KEY } ]
  [ REFERENCES reftable [ ( refcolumn ) ] ]

and table_constraints  are:
  [ UNIQUE ( column_name [, ... ] ) ]
```

```
  [ PRIMARY KEY ( column_name [, ... ] )  ]
  [ FOREIGN KEY (column_name [, ... ] ) REFERENCES reftable [ ( refcolumn ) ]


and table_attributes are:
  [ DISTSTYLE { AUTO | EVEN | KEY | ALL } ]
  [ DISTKEY ( column_name ) ]
  [ [COMPOUND | INTERLEAVED ] SORTKEY ( column_name [,...]) |  [ SORTKEY AUTO ] ]
  [ ENCODE AUTO ]
```

参数

LOCAL

可选。虽然语句中接受此关键词，但它在 Amazon Redshift 中没有任何作用。

TEMPORARY | TEMP

一个关键字，可创建仅在当前会话中可见的临时表。在从中创建表的会话结束时，将自动删除此表。临时表可具有与永久表相同的名称。在特定于会话的单独 schema 中创建临时表。(您无法为此 schema 指定名称。) 此临时架构将成为搜索路径中的第一个架构，因此临时表优先于永久表，除非您使用架构名称来限定表名以访问永久表。有关 schema 和优先顺序的更多信息，请参阅 search_path。

> **Note**
>
> 默认情况下，数据库用户有权通过其在 PUBLIC 组中自动获得的成员资格来创建临时表。要拒绝向用户授予此权限，可从 PUBLIC 组撤消 TEMP 权限，然后仅将 TEMP 权限显式授予特定用户或用户组。

IF NOT EXISTS

这个子句指示，如果指定的表已存在，命令应不进行任何更改并返回一条指示表存在的消息，而不是出错停止。请注意，现有表可能与已创建的表完全不同；仅比较表名。

此子句在编写脚本时很有用，可使脚本在 CREATE TABLE 尝试创建已存在的表时不会失败。

table_name

要创建的表的名称。

> **Important**
>
> 如果指定以“#”开始的表名，表会创建为临时表。以下是示例：
>
> ```
> create table #newtable (id int);
> ```
>
> 您还可使用“#”引用该表。例如：
>
> ```
> select * from #newtable;
> ```

表名称的最大长度为 127 个字节；更长的名称将被截断为 127 个字节。您可以使用 UTF-8 多字节字符，每个字符最多为四个字节。Amazon Redshift 按节点类型对每个集群强制实施表数量的配额，包括用户定义的临时表以及查询处理或系统维护期间由 Amazon Redshift 创建的临时表。也可使用数据库名称和 schema 名称来限定表名称。在下面的示例中，`tickit` 是数据库名称，`public` 是 schema 名称，而 `test` 是表名称。

```
create table tickit.public.test (c1 int);
```

如果数据库或 schema 不存在，则不会创建表，并且语句将返回错误。您无法在系统数据库 `template0`、`template1`、`padb_harvest` 或 `sys:internal` 中创建表或视图。

如果提供 schema 名称，则在该 schema 中创建新表（假定创建者有权访问 schema）。表名称必须是该 schema 中的唯一名称。如果未指定 schema，则可使用当前数据库 schema 创建表。如果您创建的是临时表，则无法指定 schema 名称，因为特定 schema 中存在临时表。

同一个数据库中可同时存在多个同名的临时表，前提是这些临时表是在单独的会话中创建的，因为这些表将分配给不同的 schema。有关有效名称的更多信息，请参阅名称和标识符。

column_name

要在新表中创建的列的名称。列名称的最大长度为 127 个字节；更长的名称将被截断为 127 个字节。您可以使用 UTF-8 多字节字符，每个字符最多为四个字节。可在单个表中定义的列的最大数目为 1,600。有关有效名称的更多信息，请参阅名称和标识符。

> **Note**
>
> 如果您创建的是“宽表”，请注意，在加载和查询处理期间，不要让列列表超出中间结果的行宽度边界。有关更多信息，请参阅 使用说明。

data_type

要创建的列的数据类型。对于 CHAR 和 VARCHAR 列，您可以使用 MAX 关键字而不是声明最大长度。MAX 将最大长度设置为 4096 字节（对于 CHAR）或 65535 字节（对于 VARCHAR）。GEOMETRY 对象的最大大小为 1048447 字节。

有关 Amazon Redshift 支持的数据类型的信息，请参阅数据类型。

DEFAULT default_expr

一个为列分配默认数据值的子句。default_expr 的数据类型必须匹配列的数据类型。DEFAULT 值必须是无变量的表达式。不允许子查询、对当前表中其他列的交叉引用和用户定义的函数。

default_expr 表达式可用于未为列指定值的任何 INSERT 操作。如果未指定默认值，则列的默认值为 null。

如果具有定义的列列表的 COPY 操作忽略具有 DEFAULT 值的列，则 COPY 命令将插入 default_expr 的值。

IDENTITY(seed, step)

一个指定列为 IDENTITY 列的子句。IDENTITY 列包含唯一的自动生成的值。IDENTITY 列的数据类型必须为 INT 或 BIGINT。

使用 `INSERT` 或 `INSERT INTO [tablename] VALUES()` 语句添加行时，这些值以指定为 seed 的值开始，并按照指定为 step 的数字递增。

使用 `INSERT INTO [tablename] SELECT * FROM` 或 `COPY` 语句加载表时，数据将并行加载并分发到节点切片。为确保身份值唯一，Amazon Redshift 在创建身份值时跳过多个值。身份值是唯一的，但顺序可能与源文件中的顺序不匹配。

GENERATED BY DEFAULT AS IDENTITY (seed, step)

指定该列为默认 IDENTITY 列并使您可以自动为该列分配唯一值的子句。IDENTITY 列的数据类型必须为 INT 或 BIGINT。添加不含值的行时，这些值以指定为 seed 的值开始，并按照指定为 step 的数字递增。有关如何生成值的信息，请参阅IDENTITY。

此外，在 INSERT、UPDATE 或 COPY 期间，您可以提供没有 EXPLICIT_IDS 的值。Amazon Redshift 会使用该值插入到身份列，而不是使用系统生成的值。该值可以是重复值、小于 seed 的值或介于 step 值之间的值。Amazon Redshift 不检查列中值的唯一性。提供一个值不会影响下一个系统生成的值。

> **Note**
>
> 如果您需要在列中保持唯一性，请勿添加重复值。而是添加小于 seed 或介于 step 值之间的唯一值。

记住与默认身份列有关的以下信息：

- 默认身份列为 NOT NULL。不能插入 NULL。
- 要将生成的值插入默认身份列，请使用关键字 DEFAULT。

```
INSERT INTO tablename (identity-column-name) VALUES (DEFAULT);
```

- 覆盖默认身份列的值不会影响下一个生成的值。
- 您无法使用 ALTER TABLE ADD COLUMN 语句添加默认身份列。
- 您可以使用 ALTER TABLE APPEND 语句附加默认身份列。

ENCODE encoding

列的压缩编码。ENCODE AUTO 是表的默认设置。Amazon Redshift 会自动管理表中所有列的压缩编码。如果为表中的任何列指定压缩编码，则表不再设置为 ENCODE AUTO。Amazon Redshift 不再自动管理表中所有列的压缩编码。您可以为表指定 ENCODE AUTO 选项，以使 Amazon Redshift 能够自动管理表中所有列的压缩编码。

Amazon Redshift 会自动为您未指定压缩编码的列分配初始压缩编码，如下所示：

- 默认情况下，会为临时表中的所有列分配 RAW 压缩。
- 为定义为排序键的列分配 RAW 压缩。
- 定义为 BOOLEAN、REAL、DOUBLE PRECISION、GEOMETRY 或 GEOGRAPHY 数据类型的列分配了 RAW 压缩。
- 定义为 SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP 或 TIMESTAMPTZ 的列分配了 AZ64 压缩。
- 定义为 CHAR、VARCHAR 或 VARBYTE 的列分配了 LZO 压缩。

> **Note**
>
> 如果您不希望压缩某个列，请显式指定 RAW 编码。

支持以下compression encodings (p. 55)：

- AZ64
- BYTEDICT
- DELTA
- DELTA32K
- LZO
- MOSTLY8
- MOSTLY16
- MOSTLY32
- RAW（无压缩）
- RUNLENGTH
- TEXT255
- TEXT32K
- ZSTD

DISTKEY

一个指定列为表的分配键的关键字。一个表中只能有一个列可成为分配键。可在列名后使用 DISTKEY 关键字，也可使用 DISTKEY (column_name) 语法将该关键字用作表定义的一部分。每种方法的效果相同。有关更多信息，请参阅本主题后面的 DISTSTYLE 参数。

分配键列的数据类型可以为：BOOLEAN、REAL、DOUBLE PRECISION、SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP 或 TIMESTAMPTZ、CHAR 或 VARCHAR。

SORTKEY

一个指定列为表的排序键的关键字。在数据加载到表中后，将按指定为排序键的一个或多个列对数据进行排序。可在列名后使用 SORTKEY 关键字来指定单列排序键，也可使用 SORTKEY (column_name [, ...]) 语法将一个或多个列指定为表的排序键列。仅使用此语法创建复合排序键。

您最多可以为每个表定义 400 个 SORTKEY 列。

排序键列的数据类型可以为：BOOLEAN、REAL、DOUBLE PRECISION、SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP 或 TIMESTAMPTZ、CHAR 或 VARCHAR。

COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE

指定列中的字符串搜索或比较是 CASE_SENSITIVE 还是 CASE_INSENSITIVE 的子句。默认值与数据库的当前区分大小写的配置相同。

要查找数据库排序规则信息，请使用以下命令：

```
SELECT db_collation();

db_collation
----------------
 case_sensitive
(1 row)
```

NOT NULL | NULL

NOT NULL 指定列中不允许包含 null 值。NULL（默认值）指定列接受 null 值。默认情况下，将 IDENTITY 列声明为 NOT NULL。

UNIQUE

一个指定列只能包含唯一值的关键字。唯一表约束的行为与列约束的行为相同，但前者能够跨多个列。要定义唯一表约束，请使用 UNIQUE (column_name [, ...]) 语法。

> **Important**
>
> 唯一约束是信息性的，而不由系统强制实施。

PRIMARY KEY

一个指定列为表的主键的关键字。通过使用列定义，只能将一个列定义为主键。要使用多列主键定义表约束，请使用 PRIMARY KEY (column_name [, ...]) 语法。

通过将一个列标识为主键，可提供有关 schema 设计的元数据。主键意味着其他表可将此列集用作行的唯一标识符。可以为一个表指定一个主键，作为列约束或表约束。主键约束指定的列集应不同于为同一表定义的任何唯一约束指定的其他列集。

PRIMARY KEY 列也定义为 NOT NULL。

> **Important**
>
> 主键约束仅为信息性的。这些约束不由系统强制实施，而是由计划程序使用。

References reftable [(refcolumn)]

一个指定外键约束的子句，该外键约束暗示列包含的值必须与被引用表的某个行的被引用列中的值匹配。被引用列应为引用的表中的唯一或主键约束的列。

> **Important**
>
> 外键约束仅为信息性的。这些约束不由系统强制实施，而是由计划程序使用。

LIKE parent_table [{ INCLUDING | EXCLUDING } DEFAULTS]

一个指定现有表的子句，新表自动从该表中复制列名、数据类型和 NOT NULL 约束。新表和父表是分离的，对父表所做的所有更改都不适用于新表。仅在指定 INCLUDING DEFAULTS 的情况下复制已复制列定义的默认表达式。默认行为是排除默认表达式，以便新表的所有列包含 null 默认值。

使用 LIKE 选项创建的表不继承主键约束和外键约束。分配样式、排序键、BACKUP 和 NULL 属性由 LIKE 表继承，但您无法在 CREATE TABLE ... LIKE 语句中明确设置这些 属性。

BACKUP { YES | NO }

一个子句，指定表是否应包含在自动和手动集群快照中。对于不会包含关键数据的表（如暂存表），请指定 BACKUP NO 以节省在创建快照并从快照还原时的处理时间，从而减小在 Amazon Simple Storage Service 上占用的存储空间。BACKUP NO 设置不会影响数据到集群内的其他节点的自动复制，因此当发生节点故障时，指定了 BACKUP NO 的表将被还原。默认值为 BACKUP YES。

DISTSTYLE { AUTO | EVEN | KEY | ALL }

定义整个表的数据分配样式的关键词。Amazon Redshift 会根据为表指定的分配样式将表行分配给计算节点。默认值为 AUTO。

为表选择的分配样式将影响数据库的整体性能。有关更多信息，请参阅 使用数据分配样式。可能的分配样式如下：

- AUTO：Amazon Redshift 可基于表数据指定最佳分配方式。例如，如果指定 AUTO 分配方式，Amazon Redshift 最初向小型表指定的是 ALL 分配方式。当表变大时，Amazon Redshift 可

能会将分配方式更改为 KEY，选择主键（或复合主键的列）作为 DISTKEY。如果表变大且没有任何一列适合用作 DISTKEY，Amazon Redshift 会将分配方式更改为 EVEN。分配方式的更改在后台进行，对用户查询的影响极小。

要查看应用于表的分配方式，请查询 PG_CLASS 系统目录表。有关更多信息，请参阅 查看分配方式。

- EVEN：表中的数据在轮询分配中跨集群中的节点均匀分布。行 ID 用来确定分配，并且为每个节点分配的行数大致相同。
- KEY：按 DISTKEY 列中的值分配数据。在您将联接表的联接列设置为分配键时，来自这两个表的联接行将在计算节点上并置。在并置数据时，优化程序可更高效地执行联接。如果您指定 DISTSTYLE KEY，则必须为表指定 DISTKEY 列或者将此列指定为列定义的一部分。有关更多信息，请参阅本主题前面的 DISTKEY 参数。
- ALL：向每个节点分配整个表的副本。此分配样式可确保任何联接所需的所有行在每个节点上都可用，但这将使存储要求成倍提高，并且会增加表的加载和维护次数。将 ALL 分配样式用于 KEY 分配不适用的部分维度表时会缩短执行时间，但必须针对维护成本来权衡性能改进。

DISTKEY (column_name)

一个约束，指定要用作表的分配键的列。可在列名后使用 DISTKEY 关键字，也可使用 DISTKEY (column_name) 语法将该关键字用作表定义的一部分。每种方法的效果相同。有关更多信息，请参阅本主题前面的 DISTSTYLE 参数。

[COMPOUND | INTERLEAVED] SORTKEY (column_name [,...]) | [SORTKEY AUTO]

为表指定一个或多个排序键。在数据加载到表中后，将按指定为排序键的列对数据进行排序。可在列名后使用 SORTKEY 关键字来指定单列排序键，也可使用 SORTKEY （column_name [, ...] ）语法将一个或多个列指定为表的排序键列。

您可以选择指定 COMPOUND 或 INTERLEAVED 排序样式。如果使用列指定 SORTKEY，则默认值为 COMPOUND。有关更多信息，请参阅 使用排序键。

如果您不指定任何排序键选项，则默认设置为 AUTO。

最多可以为每个表定义 400 个 COMPOUND SORTKEY 列或 8 个 INTERLEAVED SORTKEY 列。

AUTO

指定 Amazon Redshift 会基于表数据分配最佳排序键。例如，如果指定了 AUTO 排序键，Amazon Redshift 最初不会为表分配排序键。如果 Amazon Redshift 确定排序键将提高查询的性能，那么 Amazon Redshift 可能会更改您的表的排序键。表的实际排序通过自动表排序完成。有关更多信息，请参阅 自动表排序。

Amazon Redshift 不会修改具有现有排序键或分配键的表。一个例外情况是，如果表具有从未在 JOIN 中使用过的分配键，则可能会在 Amazon Redshift 确定有更好的键时更改键。

要查看表的排序键，请查询 SVV_TABLE_INFO 系统目录视图。有关更多信息，请参阅 SVV_TABLE_INFO。要查看 Amazon Redshift Advisor 对表的建议，请查询 SVV_ALTER_TABLE_RECOMMENDATIONS 系统目录视图。有关更多信息，请参阅 SVV_ALTER_TABLE_RECOMMENDATIONS。要查看 Amazon Redshift 所采取的操作，请查询 SVL_AUTO_WORKER_ACTION 系统目录视图。有关更多信息，请参阅 SVL_AUTO_WORKER_ACTION。

COMPOUND

指定使用由所有列出的列构成的复合键按这些列的列出顺序对数据进行排序。当查询根据排序列的顺序扫描行时，复合排序键最有用。当查询依赖辅助排序列时，使用复合键进行排序所带来的性能好处会减少。您最多可以为每个表定义 400 个 COMPOUND SORTKEY 列。

INTERLEAVED

指定使用交错排序键对数据进行排序。可以为一个交错排序键最多指定 8 个列。

交错排序为排序键中的每个列或列子集提供了相同的权重，以便查询不会依赖列在排序键中的顺序。当查询使用一个或多个辅助排序列时，交错排序会大大提高查询性能。交错排序产生的数据加载和 vacuum 操作的开销成本较低。

> **Important**
>
> 不要在具有单调递增属性的列（例如，身份列、日期或时间戳）上使用交错排序键。

ENCODE AUTO

使 Amazon Redshift 能够自动调整表中所有列的编码类型，以优化查询性能。ENCODE AUTO 会保留您在创建表时指定的初始编码类型。然后，如果 Amazon Redshift 确定新的编码类型可以提高查询性能，则 Amazon Redshift 可以更改表列的编码类型。如果不在表中的任何列上指定编码类型，则 ENCODE AUTO 是默认设置。

UNIQUE (column_name [,...])

一个约束，指定包含一个或多个表列的组只能包含唯一值。唯一表约束的行为与列约束的行为相同，但前者能够跨多个列。在唯一约束的上下文中，null 值不被视为相同。每个唯一表约束指定的列集必须不同于由为表定义的任何其他唯一键约束或主键约束指定的列集。

> **Important**
>
> 唯一约束是信息性的，而不由系统强制实施。

PRIMARY KEY (column_name [,...])

一个约束，指定表的一个列或大量列只能包含唯一（不重复）的非 null 值。通过将一个列集标识为主键，也会提供有关 schema 设计的元数据。主键意味着其他表可将此列集用作行的唯一标识符。可以为一个表指定一个主键，作为单个列约束或表约束。主键约束指定的列集应不同于为同一表定义的任何唯一约束指定的其他列集。

> **Important**
>
> 主键约束仅为信息性的。这些约束不由系统强制实施，而是由计划程序使用。

FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn)]

一个指定外键约束的约束，该约束要求新表的一个或多个列只能包含与被引用表的某个行的一个或多个被引用列中的值匹配的值。如果忽略 refcolumn，则使用 reftable 的主键。被引用列必须为被引用表中的唯一或主键约束的列。

> **Important**
>
> 外键约束仅为信息性的。这些约束不由系统强制实施，而是由计划程序使用。

使用说明

唯一键、主键和外键约束仅供参考；在您填充表时，Amazon Redshift 并不强制实施它们。例如，如果您将数据插入到具有依赖关系的表中，即使插入操作违反了约束，插入也会成功。但是，主键和外键用作规划提示，如果您应用程序中的 ETL 处理或其他一些处理强制其完整性，则应声明它们。有关如何删除具有依赖关系的表的信息，请参阅 DROP TABLE。

限制和配额

创建表时，请考虑以下限制。

- 集群中按节点类型的最大表数有限制。有关更多信息，请参阅《Amazon Redshift 管理指南》中的限制。
- 表名的最大字符数为 127。
- 可在单个表中定义的列的最大数目为 1,600。
- 可在单个表中定义的 SORTKEY 列的最大数目为 400。

列级设置和表级设置摘要

可在列级或表级设置若干属性和设置。在某些情况下，在列级或表级设置属性或约束的效果相同。在其他情况下，它们会产生不同的结果。

下面的列表汇总了列级和表级设置：

DISTKEY

无论是在列级设置还是在表级设置，效果是相同的。

如果在列级或表级设置 DISTKEY，则 DISTSTYLE 必须设置为 KEY 或者根本不设置 DISTSTYLE。只能在表级设置 DISTSTYLE。

SORTKEY

如果在列级进行设置，则 SORTKEY 必须为单个列。如果在表级设置 SORTKEY，则一个或多个列可构成复合或交错复合排序键。

COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE

Amazon Redshift 不支持更改列的区分大小写配置。将新列附加到表中时，Amazon Redshift 会使用默认值区分大小写。在附加新列时，Amazon Redshift 不支持 COLLATE 关键字。

有关如何使用数据库排序规则创建数据库的信息，请参阅CREATE DATABASE。

有关 COLLATE 函数的信息，请参阅COLLATE 函数。

UNIQUE

在列级别，可将一个或多个键设置为 UNIQUE；UNIQUE 约束分别应用于每个列。如果在表级设置 UNIQUE，则一个或多个列可构成复合 UNIQUE 约束。

PRIMARY KEY

如果在列级进行设置，则 PRIMARY KEY 必须为单个列。如果在表级设置 PRIMARY KEY，则一个或多个列可构成复合主键。

FOREIGN KEY

无论是在列级设置还是在表级设置 FOREIGN KEY，效果是相同的。在列级别，语法为 REFERENCES reftable [(refcolumn)]。

分配传入数据

如果传入数据的哈希分配方案与目标表的哈希分配方案匹配，则在加载数据时，没有实际必要的数据物理分配。例如，如果为新表设置分配键并从相同键列上分配的另一个表中插入数据，则使用相同的节点和切片就地加载数据。不过，如果源表和目标表都设置为 EVEN 分配，则数据将重新分配到目标表。

宽表

您也许能够创建很宽的表，但无法对表执行查询处理，例如 INSERT 或 SELECT 语句。具有宽度固定的列（例如 CHAR）的表的最大宽度为 64KB - 1（即 65535 字节）。如果表包含 VARCHAR 列，则表可以具有更大的声明宽度，而不会返回错误，因为 VARCHARS 列不会将其完全声明的宽度计入计算出的查询处理限制。针对 VARCHAR 列的有效查询处理限制将因大量因素而异。

如果表对于插入或选择操作来说太宽，您将收到以下错误。

```
ERROR:  8001
DETAIL:  The combined length of columns processed in the SQL statement
exceeded the query-processing limit of 65535 characters (pid:7627)
```

示例

有关说明 CREATE TABLE 命令用法的示例，请参阅示例 主题。

示例

以下示例演示 Amazon Redshift CREATE TABLE 语句中的各种列和表属性。有关 CREATE TABLE 的更多信息，包括参数定义，请参阅 CREATE TABLE。

许多示例使用来自 TICKIT 示例数据集的表和数据。有关更多信息，请参阅示例数据库。

在 CREATE TABLE 命令中，您可以使用数据库名称和架构名称作为表名称前缀。例如，dev_database.public.sales。数据库名称必须是您已连接到的数据库。在其他数据库中创建数据库对象的任何尝试都会失败，并出现无效的操作错误。

使用分配键、复合排序键和压缩创建表

以下示例利用为多个列定义的压缩在 TICKIT 数据库中创建 SALES 表。LISTID 声明为分配键，LISTID 和 SELLERID 声明为多列复合排序键。还为表定义了主键约束和外键约束。创建示例中的表之前，如果不存在约束，则可能需要向外键引用的每个列添加 UNIQUE 约束。

```
create table sales(
salesid integer not null,
listid integer not null,
sellerid integer not null,
buyerid integer not null,
eventid integer not null encode mostly16,
dateid smallint not null,
qtysold smallint not null encode mostly8,
pricepaid decimal(8,2) encode delta32k,
commission decimal(8,2) encode delta32k,
saletime timestamp,
primary key(salesid),
foreign key(listid) references listing(listid),
foreign key(sellerid) references users(userid),
foreign key(buyerid) references users(userid),
foreign key(dateid) references date(dateid))
distkey(listid)
compound sortkey(listid,sellerid);
```

结果如下：

```
schemaname | tablename | column     | type                          | encoding | distkey
 | sortkey | notnull
-----------+-----------+------------+-------------------------------+----------+---------
+---------+--------
public     | sales     | salesid    | integer                       | lzo      | false
 |       0 | true
public     | sales     | listid     | integer                       | none     | true
 |       1 | true
public     | sales     | sellerid   | integer                       | none     | false
 |       2 | true
public     | sales     | buyerid    | integer                       | lzo      | false
 |       0 | true
public     | sales     | eventid    | integer                       | mostly16 | false
 |       0 | true
public     | sales     | dateid     | smallint                      | lzo      | false
 |       0 | true
```

```
public     | sales     | qtysold    | smallint                    | mostly8  | false
 |       0 | true
public     | sales     | pricepaid  | numeric(8,2)                | delta32k | false
 |       0 | false
public     | sales     | commission | numeric(8,2)                | delta32k | false
 |       0 | false
public     | sales     | saletime   | timestamp without time zone | lzo      | false
 |       0 | false
```

以下示例使用不区分大小写的列 col1 创建表 t1。

```
create table T1 (
  col1 Varchar(20) collate case_insensitive
 );

insert into T1 values ('bob'), ('john'), ('Tom'), ('JOHN'), ('Bob');
```

查询表：

```
select * from T1 where col1 = 'John';

col1
------
 john
 JOHN
(2 rows)
```

使用交错排序键创建表

以下示例使用交错排序键创建 CUSTOMER 表。

```
create table customer_interleaved (
  c_custkey      integer        not null,
  c_name         varchar(25)    not null,
  c_address      varchar(25)    not null,
  c_city         varchar(10)    not null,
  c_nation       varchar(15)    not null,
  c_region       varchar(12)    not null,
  c_phone        varchar(15)    not null,
  c_mktsegment      varchar(10)    not null)
diststyle all
interleaved sortkey (c_custkey, c_city, c_mktsegment);
```

使用 IF NOT EXISTS 创建表

以下示例创建 CITIES 表，如果该表已存在，则不执行任何操作并返回一条消息：

```
create table if not exists cities(
cityid integer not null,
city varchar(100) not null,
state char(2) not null);
```

使用 ALL 分配创建表

以下示例使用 ALL 分配创建 VENUE 表。

```
create table venue(
venueid smallint not null,
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer,
primary key(venueid))
diststyle all;
```

使用 EVEN 分配创建表

以下示例创建一个包含三个列的名为 MYEVENT 的表。

```
create table myevent(
eventid int,
eventname varchar(200),
eventcity varchar(30))
diststyle even;
```

均匀分配表，并且不对表进行排序。表没有声明的 DISTKEY 或 SORTKEY 列。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'myevent';

  column   |          type          | encoding | distkey | sortkey
-----------+------------------------+----------+---------+---------
 eventid   | integer                | lzo      | f       |       0
 eventname | character varying(200) | lzo      | f       |       0
```

```
 eventcity | character varying(30)  | lzo      | f       |       0
(3 rows)
```

创建与另一个表类似的临时表

以下示例创建一个名为“TEMPEVENT”的临时表，该表从 EVENT 表继承其列。

```
create temp table tempevent(like event);
```

此表还继承其父表的 DISTKEY 和 SORTKEY 属性：

```
select "column", type, encoding, distkey, sortkey
 from pg_table_def where tablename = 'tempevent';

  column   |            type             | encoding | distkey | sortkey
-----------+-----------------------------+----------+---------+---------
 eventid   | integer                     | none     | t       |       1
 venueid   | smallint                    | none     | f       |       0
 catid     | smallint                    | none     | f       |       0
 dateid    | smallint                    | none     | f       |       0
 eventname | character varying(200)      | lzo      | f       |       0
 starttime | timestamp without time zone | bytedict | f       |       0
(6 rows)
```

创建具有 IDENTITY 列的表

以下示例创建一个名为 VENUE_IDENT 的表，该表具有名为 VENUEID 的 IDENTITY 列。该列从 0 开始，并为每个记录增加 1。VENUEID 还被声明为表的主键。

```
create table venue_ident(venueid bigint identity(0, 1),
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer,
primary key(venueid));
```

创建具有默认 IDENTITY 列的表

下面的示例创建了一个名为 t1 的表。此表拥有名为 hist_id 的 IDENTITY 列和名为 base_id 的默认 IDENTITY 列。

```
CREATE TABLE t1(
  hist_id BIGINT IDENTITY NOT NULL, /* Cannot be overridden */
  base_id BIGINT GENERATED BY DEFAULT AS IDENTITY NOT NULL, /* Can be overridden */
  business_key varchar(10) ,
  some_field varchar(10)
);
```

在表中插入一行，表明 hist_id 和 base_id 值均已生成。

```
INSERT INTO T1 (business_key, some_field) values ('A','MM');
```

```
SELECT * FROM t1;

 hist_id | base_id | business_key | some_field
---------+---------+--------------+------------
       1 |       1 | A            | MM
```

插入第二行，表明 base_id 的默认值已生成。

```
INSERT INTO T1 (base_id, business_key, some_field) values (DEFAULT, 'B','MNOP');
```

```
SELECT * FROM t1;

 hist_id | base_id | business_key | some_field
---------+---------+--------------+------------
       1 |       1 | A            | MM
       2 |       2 | B            | MNOP
```

插入第三行，表明 base_id 的值不需要是唯一的。

```
INSERT INTO T1 (base_id, business_key, some_field) values (2,'B','MNNN');
```

```
SELECT * FROM t1;

 hist_id | base_id | business_key | some_field
---------+---------+--------------+------------
       1 |       1 | A            | MM
       2 |       2 | B            | MNOP
```

```
       3 |       2 | B            | MNNN
```

创建具有 DEFAULT 列值的表

以下示例创建一个 CATEGORYDEF 表，该表声明每个列的默认值：

```
create table categorydef(
catid smallint not null default 0,
catgroup varchar(10) default 'Special',
catname varchar(10) default 'Other',
catdesc varchar(50) default 'Special events',
primary key(catid));

insert into categorydef values(default,default,default,default);
```

```
select * from categorydef;

 catid | catgroup | catname |    catdesc
-------+----------+---------+----------------
     0 | Special  | Other   | Special events
(1 row)
```

DISTSTYLE、DISTKEY 和 SORTKEY 选项

以下示例显示 DISTKEY、SORTKEY 和 DISTSTYLE 选项的工作方式。在此示例中，COL1 是分配键；因此，必须将分配样式设置为 KEY 或不设置分配样式。默认情况下，该表没有排序键，所以不会进行排序：

```
create table t1(col1 int distkey, col2 int) diststyle key;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't1';

column |  type   | encoding | distkey | sortkey
-------+---------+----------+---------+---------
col1   | integer | az64     | t       | 0
col2   | integer | az64     | f       | 0
```

在以下示例中，将同一个列定义为分配键和排序键。同样，必须将分配样式设置为 KEY 或者不设置分配样式。

```
create table t2(col1 int distkey sortkey, col2 int);
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't2';

column |  type   | encoding | distkey | sortkey
-------+---------+----------+---------+---------
col1   | integer | none     | t       | 1
col2   | integer | az64     | f       | 0
```

在以下示例中，未将任何列设置为分配键，COL2 设置为排序键，分配键设置为 ALL：

```
create table t3(col1 int, col2 int sortkey) diststyle all;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't3';

Column |  Type   | Encoding | DistKey | SortKey
-------+---------+----------+---------+--------
col1   | integer | az64     | f       | 0
col2   | integer | none     | f       | 1
```

在以下示例中，分配样式设置为 EVEN，并且未显式定义排序键；因此，表将均匀分配而不进行排序。

```
create table t4(col1 int, col2 int) diststyle even;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't4';

             column |  type   |encoding | distkey | sortkey
--------+---------+---------+---------+--------
col1    | integer | az64    | f       | 0
col2    | integer | az64    | f       | 0
```

使用 ENCODE AUTO 选项创建表

下面的示例使用自动压缩编码创建表 t1。不为任何列指定编码类型时，ENCODE AUTO 是表的默认设置。

```
create table t1(c0 int, c1 varchar);
```

下面的示例通过指定 ENCODE AUTO 使用自动压缩编码创建表 t2。

```
create table t2(c0 int, c1 varchar) encode auto;
```

下面的示例通过指定 ENCODE AUTO 使用自动压缩编码创建表 t3。列 c0 是用 DELTA 的初始编码类型定义的。如果另一个编码可以提供更好的查询性能，则 Amazon Redshift 可以更改编码。

```
create table t3(c0 int encode delta, c1 varchar) encode auto;
```

下面的示例通过指定 ENCODE AUTO 使用自动压缩编码创建表 t4。列 c0 使用 DELTA 的初始编码进行定义，而列 c1 则是用 LZO 的初始编码定义的。如果另一个编码可以提供更好的查询性能，则 Amazon Redshift 可以更改这些编码。

```
create table t4(c0 int encode delta, c1 varchar encode lzo) encode auto;
```

CREATE TABLE AS

主题

创建基于查询的新表。此表的所有者为发出命令的用户。

新表与命令的查询定义的数据一起加载。表列具有与查询的输出列关联的名称和数据类型。CREATE TABLE AS (CTAS) 命令创建一个新表并评估查询以加载新表。

语法

```
CREATE [ [ LOCAL ] { TEMPORARY | TEMP } ]
TABLE table_name
[ ( column_name [, ... ] ) ]
[ BACKUP { YES | NO } ]
```

```
[ table_attributes ]
AS query

where table_attributes are:
[ DISTSTYLE { AUTO | EVEN | ALL | KEY } ]
[ DISTKEY( distkey_identifier ) ]
[ [ COMPOUND | INTERLEAVED ] SORTKEY( column_name [, ...] ) ]
```

参数

LOCAL

虽然语句中接受此可选关键词，但它在 Amazon Redshift 中没有任何作用。

TEMPORARY | TEMP

创建一个临时表。在创建临时表的会话结束时将自动删除该临时表。

table_name

要创建的表的名称。

> **Important**
>
> 如果指定以“#”开始的表名，表会创建为临时表。例如：
>
> ```
> create table #newtable (id) as select * from oldtable;
> ```

表名称的最大长度为 127 个字节；更长的名称将被截断为 127 个字节。Amazon Redshift 会按节点类型强制执行每个集群的表数量配额。可使用数据库和 schema 名称限定表名，如下表所示。

```
create table tickit.public.test (c1) as select * from oldtable;
```

在此示例中，`tickit` 为数据库名称，`public` 为 schema 名称。如果数据库或 schema 不存在，此语句将返回错误。

如果提供 schema 名称，则在该 schema 中创建新表（假定创建者有权访问 schema）。表名称必须是该 schema 中的唯一名称。如果未指定 schema，则可使用当前数据库 schema 创建表。如果您创建的是临时表，则无法指定 schema 名称，因为特定 schema 中存在临时表。

如果多个同名临时表是在单独的会话中创建的，则这些同名临时表能够同时存在于同一个数据库中。这些表将分配给不同的 schemas。

column_name

新表中的列的名称。如果没有提供列名，则采用查询的输出列名中的列名。默认列名用于表达式。有关有效名称的更多信息，请参阅名称和标识符。

BACKUP { YES | NO }

一个子句，指定表是否应包含在自动和手动集群快照中。对于不会包含关键数据的表（如暂存表），请指定 BACKUP NO 以节省在创建快照并从快照还原时的处理时间，从而减小在 Amazon Simple Storage Service 上占用的存储空间。BACKUP NO 设置不会影响数据到集群内的其他节点的自动复制，因此当发生节点故障时，指定了 BACKUP NO 的表将被还原。默认值为 BACKUP YES。

DISTSTYLE { AUTO | EVEN | KEY | ALL }

定义整个表的数据分配样式。Amazon Redshift 会根据为表指定的分配样式将表行分配给计算节点。默认值为 DISTSTYLE AUTO。

为表选择的分配样式将影响数据库的整体性能。有关更多信息，请参阅 使用数据分配样式。

- AUTO：Amazon Redshift 可基于表数据指定最佳分配方式。要查看应用于表的分配方式，请查询 PG_CLASS 系统目录表。有关更多信息，请参阅 查看分配方式。
- EVEN：表中的数据在轮询分配中跨集群中的节点均匀分布。行 ID 用来确定分配，并且为每个节点分配的行数大致相同。这是默认分配方法。
- KEY：按 DISTKEY 列中的值分配数据。在您将联接表的联接列设置为分配键时，来自这两个表的联接行将在计算节点上并置。在并置数据时，优化程序可更高效地执行联接。如果您指定 DISTSTYLE KEY，则必须命名 DISTKEY 列。
- ALL：向每个节点分配整个表的副本。此分配样式可确保任何联接所需的所有行在每个节点上都可用，但这将使存储要求成倍提高，并且会增加表的加载和维护次数。将 ALL 分配样式用于 KEY 分配不适用的部分维度表时会缩短执行时间，但必须针对维护成本来权衡性能改进。

DISTKEY (column)

指定分配键的列名或位置号。使用在表的可选列列表或所选查询列表中指定的名称。或者，使用位置号，其中所选的第一列为 1，所选的第二列为 2，以此类推。一个表中只能有一个列可成为分配键：

- 如果将一个列声明为 DISTKEY 列，则必须将 DISTSTYLE 设置为 KEY 或者根本不设置 DISTSTYLE。

- 如果未声明 DISTKEY 列，则可将 DISTSTYLE 设置为 EVEN。
- 如果您不指定 DISTKEY 或 DISTSTYLE，CTAS 会根据 SELECT 子句的查询计划确定新表的分配样式。有关更多信息，请参阅 继承列和表属性。

您可以将同一个列定义为分配键和排序键；此方法旨在当有问题的列为查询中的联接列时加速联接。

[COMPOUND | INTERLEAVED] SORTKEY (column_name [, ...])

为表指定一个或多个排序键。在数据加载到表中后，将按指定为排序键的列对数据进行排序。

您可以选择指定 COMPOUND 或 INTERLEAVED 排序样式。默认为 COMPOUND。有关更多信息，请参阅 使用排序键。

最多可以为每个表定义 400 个 COMPOUND SORTKEY 列或 8 个 INTERLEAVED SORTKEY 列。

如果您不指定 SORTKEY，CTAS 会根据 SELECT 子句的查询计划的新表排序键。有关更多信息，请参阅 继承列和表属性。

COMPOUND

指定使用由所有列出的列构成的复合键按这些列的列出顺序对数据进行排序。当查询根据排序列的顺序扫描行时，复合排序键最有用。当查询依赖辅助排序列时，使用复合键进行排序所带来的性能好处会减少。您最多可以为每个表定义 400 个 COMPOUND SORTKEY 列。

INTERLEAVED

指定使用交错排序键对数据进行排序。可以为一个交错排序键最多指定 8 个列。

交错排序为排序键中的每个列或列子集提供了相同的权重，以便查询不会依赖列在排序键中的顺序。当查询使用一个或多个辅助排序列时，交错排序会大大提高查询性能。交错排序产生的数据加载和 vacuum 操作的开销成本较低。

AS query

Amazon Redshift 支持的任何查询（SELECT 语句）。

CTAS 使用说明

限制

Amazon Redshift 会按节点类型强制执行每个集群的表数量配额。

表名的最大字符数为 127。

可在单个表中定义的列的最大数目为 1,600。

继承列和表属性

CREATE TABLE AS (CTAS) 表不从其父表继承约束、身份列、默认列值或主键。

您不能为 CTAS 表指定列压缩编码。Amazon Redshift 自动分配压缩编码，如下所示：

- 为定义为排序键的列分配 RAW 压缩。
- 定义为 BOOLEAN、REAL、DOUBLE PRECISION、GEOMETRY 或 GEOGRAPHY 数据类型的列分配了 RAW 压缩。
- 定义为 SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP 或 TIMESTAMPTZ 的列分配了 AZ64 压缩。
- 定义为 CHAR、VARCHAR 或 VARBYTE 的列分配了 LZO 压缩。

有关更多信息，请参阅压缩编码 和数据类型。

要显式分配列编码，请使用 CREATE TABLE。

CTAS 根据 SELECT 子句的查询计划确定新表的分配样式和排序键。

对于复杂查询，如包含连接、聚合、ORDER BY 子句或 LIMIT 子句，CTAS 会尽最大努力基于查询计划选择最佳分配样式和分类键。

> **Note**
>
> 对于使用大型数据集或复杂查询实现最佳性能，我们建议使用典型数据集进行测试。

通常，您可以通过检查查询计划查看查询优化器选择哪些列（如果有）进行数据排序和分配，预测 CTAS 将选择哪个分配键和分类键。如果查询计划的顶端节点为一个表（XN 顺序扫描）的简单顺序扫描，则 CTAS 通常使用源表的分配样式和分类键。如果查询计划的顶部节点是除顺序扫描外的任何情况（如 XN 限制、XN 排序、XN HashAggregate 等），CTAS 会尽最大努力根据查询计划选择最佳分配样式和分类键。

例如，假设您使用 SELECT 子句创建了以下五个表：

- 简单的 SELECT 语句
- Limit 子句
- 使用 LISTID 的 ORDER BY 子句
- 使用 QTYSOLD 的 ORDER BY 子句
- 使用 GROUP BY 子句的 SUM 聚合函数。

以下示例显示每个 CTAS 语句的查询计划。

```
explain create table sales1_simple as select listid, dateid, qtysold from sales;
                           QUERY PLAN
-----------------------------------------------------------------
 XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(1 row)


explain create table sales2_limit as select listid, dateid, qtysold from sales limit
 100;
                              QUERY PLAN
----------------------------------------------------------------------
 XN Limit  (cost=0.00..1.00 rows=100 width=8)
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(2 rows)


explain create table sales3_orderbylistid as select listid, dateid, qtysold from sales
 order by listid;
                               QUERY PLAN
-----------------------------------------------------------------------
 XN Sort  (cost=1000000016724.67..1000000017155.81 rows=172456 width=8)
   Sort Key: listid
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(3 rows)


explain create table sales4_orderbyqty as select listid, dateid, qtysold from sales
 order by qtysold;
                               QUERY PLAN
-----------------------------------------------------------------------
 XN Sort  (cost=1000000016724.67..1000000017155.81 rows=172456 width=8)
   Sort Key: qtysold
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
```

```
(3 rows)


explain create table sales5_groupby as select listid, dateid, sum(qtysold) from sales
 group by listid, dateid;
                            QUERY PLAN
------------------------------------------------------------------------
 XN HashAggregate  (cost=3017.98..3226.75 rows=83509 width=8)
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(2 rows)
```

要查看每个表的分配键和分类键，请查询 PG_TABLE_DEF 系统目录表，如下所示。

```
select * from pg_table_def where tablename like 'sales%';

      tablename       |   column   | distkey | sortkey
----------------------+------------+---------+---------
 sales                | salesid    | f       |       0
 sales                | listid     | t       |       0
 sales                | sellerid   | f       |       0
 sales                | buyerid    | f       |       0
 sales                | eventid    | f       |       0
 sales                | dateid     | f       |       1
 sales                | qtysold    | f       |       0
 sales                | pricepaid  | f       |       0
 sales                | commission | f       |       0
 sales                | saletime   | f       |       0
 sales1_simple        | listid     | t       |       0
 sales1_simple        | dateid     | f       |       1
 sales1_simple        | qtysold    | f       |       0
 sales2_limit         | listid     | f       |       0
 sales2_limit         | dateid     | f       |       0
 sales2_limit         | qtysold    | f       |       0
 sales3_orderbylistid | listid     | t       |       1
 sales3_orderbylistid | dateid     | f       |       0
 sales3_orderbylistid | qtysold    | f       |       0
 sales4_orderbyqty    | listid     | t       |       0
 sales4_orderbyqty    | dateid     | f       |       0
 sales4_orderbyqty    | qtysold    | f       |       1
 sales5_groupby       | listid     | f       |       0
 sales5_groupby       | dateid     | f       |       0
 sales5_groupby       | sum        | f       |       0
```

下表汇总了结果。为简便起见，我们在说明计划中忽略了成本、行和宽度详细信息。

表	CTAS SELECT 语句	说明计划顶部节点	分配键	排序键
S1_SIMPLE	`select listid, dateid, qtysold from sales`	`XN Seq Scan on sales ...`	LISTID	DATEID
S2_LIMIT	`select listid, dateid, qtysold from sales limit 100`	`XN Limit ...`	无 (EVEN)	无
S3_ORDER_BY_LISTID	`select listid, dateid, qtysold from sales order by listid`	`XN Sort ...` `Sort Key: listid`	LISTID	LISTID
S4_ORDER_BY_QTY	`select listid, dateid, qtysold from sales order by qtysold`	`XN Sort ...` `Sort Key: qtysold`	LISTID	QTYSOLD
S5_GROUP_BY	`select listid, dateid, sum(qtysold) from sales group by listid, dateid`	`XN HashAggregate ...`	无 (EVEN)	无

您可以在 CTAS 语句中明确指定分配样式和分类键。例如，下面的语句使用 EVEN 分配创建了一个表并指定 SALESID 作为分类键。

```
create table sales_diseteven
diststyle even
sortkey (salesid)
as
select eventid, venueid, dateid, eventname
from event;
```

压缩编码

ENCODE AUTO 用作表的默认设置。Amazon Redshift 会自动管理表中所有列的压缩编码。

分配传入数据

如果传入数据的哈希分配方案与目标表的哈希分配方案匹配，则在加载数据时，没有实际必要的数据物理分配。例如，如果为新表设置分配键并从相同键列上分配的另一个表中插入数据，则使用相同的节点和切片就地加载数据。不过，如果源表和目标表都设置为 EVEN 分配，则数据将重新分配到目标表。

自动 ANALYZE 操作

Amazon Redshift 自动分析您使用 CTAS 命令创建的表。在最初创建这些表时，您不需要对这些表运行 ANALYZE 命令。如果修改了这些表，则应通过用来分析其他表的方式来分析这些表。

CTAS 示例

以下示例为 EVENT 表创建名为 EVENT_BACKUP 的表：

```
create table event_backup as select * from event;
```

生成的表继承 EVENT 表中的分配键和排序键。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'event_backup';

column    | type                        | encoding | distkey | sortkey
----------+-----------------------------+----------+---------+--------
catid     | smallint                    | none     | false   |       0
dateid    | smallint                    | none     | false   |       1
eventid   | integer                     | none     | true    |       0
eventname | character varying(200)      | none     | false   |       0
starttime | timestamp without time zone | none     | false   |       0
venueid   | smallint                    | none     | false   |       0
```

以下命令通过选择 EVENT 表中的四个列来创建一个名为 EVENTDISTSORT 的新表。新表按 EVENTID 进行分配并按 EVENTID 和 DATEID 进行排序：

```
create table eventdistsort
distkey (1)
```

```
sortkey (1,3)
as
select eventid, venueid, dateid, eventname
from event;
```

结果如下：

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'eventdistsort';

column   |          type          | encoding | distkey | sortkey
---------+------------------------+----------+---------+-------
eventid   | integer                | none     | t       | 1
venueid   | smallint               | none     | f       | 0
dateid    | smallint               | none     | f       | 2
eventname | character varying(200)| none     | f       | 0
```

可通过对分配键和排序键使用列名来创建完全相同的表。例如：

```
create table eventdistsort1
distkey (eventid)
sortkey (eventid, dateid)
as
select eventid, venueid, dateid, eventname
from event;
```

以下语句对表应用 EVEN 分配，但不定义明确的排序键。

```
create table eventdisteven
diststyle even
as
select eventid, venueid, dateid, eventname
from event;
```

该表不继承 EVENT 表 (EVENTID) 中的排序键，因为已为新表指定 EVEN 分配。新表没有排序键和分配键。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'eventdisteven';

column    |          type          | encoding | distkey | sortkey
```

```
----------+------------------------+----------+---------+---------
eventid   | integer                | none     | f       | 0
venueid   | smallint               | none     | f       | 0
dateid    | smallint               | none     | f       | 0
eventname | character varying(200) | none     | f       | 0
```

以下语句应用 EVEN 分配并定义排序键：

```
create table eventdistevensort diststyle even sortkey (venueid)
as select eventid, venueid, dateid, eventname from event;
```

生成的表具有排序键，但不具有分配键。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'eventdistevensort';

column    |          type          | encoding | distkey | sortkey
----------+------------------------+----------+---------+-------
eventid   | integer                | none     | f       | 0
venueid   | smallint               | none     | f       | 1
dateid    | smallint               | none     | f       | 0
eventname | character varying(200) | none     | f       | 0
```

以下语句基于来自传入数据（基于 EVENTID 列进行排序）的其他键列重新分配 EVENT 表，但不定义 SORTKEY 列；因此，不会对表进行排序。

```
create table venuedistevent distkey(venueid)
as select * from event;
```

结果如下：

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'venuedistevent';

 column   |            type             | encoding | distkey | sortkey
----------+-----------------------------+----------+---------+-------
eventid   | integer                     | none     | f       | 0
venueid   | smallint                    | none     | t       | 0
catid     | smallint                    | none     | f       | 0
dateid    | smallint                    | none     | f       | 0
eventname | character varying(200)      | none     | f       | 0
```

```
starttime | timestamp without time zone | none     | f       | 0
```

CREATE USER

创建新的数据库用户。数据库用户可以根据他们的权限和角色，在数据库中检索数据、运行命令和执行其他操作。您必须是数据库超级用户才能执行此命令。

所需的权限

以下是 CREATE USER 所需的权限：

- Superuser
- 具有 CREATE USER 权限的用户

语法

```
CREATE USER name [ WITH ]
PASSWORD { 'password' | 'md5hash' | 'sha256hash' | DISABLE }
[ option [ ... ] ]

where option can be:

CREATEDB | NOCREATEDB
| CREATEUSER | NOCREATEUSER
| SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }
| IN GROUP groupname [, ... ]
| VALID UNTIL 'abstime'
| CONNECTION LIMIT { limit | UNLIMITED }
| SESSION TIMEOUT limit
| EXTERNALID external_id
```

参数

name

要创建的用户的名称。用户名称不能为 PUBLIC。有关有效名称的更多信息，请参阅名称和标识符。

WITH

可选关键字。Amazon Redshift 将会忽略 WITH

PASSWORD { 'password' | 'md5hash' | 'sha256hash' | DISABLE }

设置用户的密码。

默认情况下，用户可以更改自己的密码，除非密码被禁用。要禁用用户的密码，请指定 DISABLE。禁用某个用户的密码后，将从系统中删除该密码，而此用户只能使用临时 AWS Identity and Access Management (IAM) 用户凭证进行登录。有关更多信息，请参阅使用 IAM 身份验证生成数据库用户凭证。只有超级用户才能启用或禁用密码。您不能禁用超级用户的密码。要启用密码，请运行 ALTER USER 并指定密码。

您可采用明文、MD5 哈希字符串或 SHA256 哈希字符串的形式指定密码。

> **Note**
>
> 使用 AWS Management Console、AWS CLI 或 Amazon Redshift API 启动新集群时，必须为初始数据库用户提供一个明文密码。您稍后可以使用 ALTER USER 更改密码。

对于明文，密码必须遵循以下约束：

- 它的长度必须介于 8 到 64 个字符之间。
- 它必须包含至少一个大写字母、一个小写字母和一个数字。
- 它可以使用带有 ASCII 代码 33–126 的任何 ASCII 字符，但 '（单引号）、"（双引号）、\、/ 或 @ 除外。

作为以明文形式传递 CREATE USER 密码参数的更安全的替代方法，您可以指定包含密码和用户名的 MD5 哈希。

> **Note**
>
> 当您指定 MD5 哈希字符串时，CREATE USER 命令将检查是否存在有效的 MD5 哈希字符串，但它不会验证字符串的密码部分。在这种情况下，可以创建无法用于登录数据库的密码（如空字符串）。

要指定 MD5 密码，请执行以下步骤：

1. 联接密码和用户名。

   例如，对于密码 `ez` 和用户 `user1`，联接后的字符串为 `ezuser1`。

2. 将联接后的字符串转换为 32 字符 MD5 哈希字符串。您可以使用任何 MD5 实用工具创建哈希字符串。以下示例使用 Amazon Redshift MD5 函数 和联接运算符 (||) 返回 32 字符的 MD5 哈希字符串。

```
select md5('ez' || 'user1');

md5
--------------------------------
153c434b4b77c89e6b94f12c5393af5b
```

3. 在 MD5 哈希字符串前面联接“md5”并提供联接后的字符串作为 md5hash 参数。

```
create user user1 password 'md5153c434b4b77c89e6b94f12c5393af5b';
```

4. 使用登录凭证登录数据库。

   对于此示例，请使用密码 user1 以 ez 的身份登录。

还有一种安全的替代方法，即指定密码字符串的 SHA-256 哈希值；您也可以提供自己的有效 SHA-256 摘要和用于创建摘要的 256 位加密盐。

- 摘要 – 哈希函数的输出。
- 加密盐 – 随机生成的与密码相结合使用的数据，帮助减少哈希函数输出中的模式。

```
'sha256|Mypassword'
```

```
'sha256|digest|256-bit-salt'
```

在以下示例中，Amazon Redshift 生成并管理加密盐。

```
CREATE USER admin PASSWORD 'sha256|Mypassword1';
```

在以下示例中，提供了有效的 SHA-256 摘要和用于创建摘要的 256 位加密盐。

要指定密码并使用您自己的加密盐对其进行哈希处理，请按照以下步骤操作：

1. 创建 256 位加密盐。您可以通过使用任何十六进制字符串生成器生成长度为 64 个字符的字符串来获取加密盐。在本例中，加密盐为 c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6。

2. 使用 FROM_HEX 函数将您的加密盐转换为二进制。这是因为 SHA2 函数需要加密盐的二进制表示。查看以下语句。

```
SELECT
 FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6');
```

3. 使用 CONCAT 函数将加密盐附加到密码。在本示例中，密码为 Mypassword1。查看以下语句。

```
SELECT
 CONCAT('Mypassword1',FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f951303
```

4. 使用 SHA2 函数根据您的密码和加密盐组合创建摘要。查看以下语句。

```
SELECT
 SHA2(CONCAT('Mypassword1',FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9
 0);
```

5. 使用前面步骤中的摘要和加密盐来创建用户。查看以下语句。

```
CREATE USER admin PASSWORD 'sha256|
821708135fcc42eb3afda85286dee0ed15c2c461d000291609f77eb113073ec2|
c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6';
```

6. 使用登录凭证登录数据库。

   对于此示例，请使用密码 admin 以 Mypassword1 的身份登录。

如果在未指定哈希函数的情况下设置纯文本形式的密码，则会将用户名用作加密盐生成 MD5 摘要。

CREATEDB | NOCREATEDB

CREATEDB 选项允许新用户创建数据库。默认值为 NOCREATEDB。

CREATEUSER | NOCREATEUSER

使用 CREATEUSER 选项可以创建具备所有数据库权限的超级用户，包括 CREATE USER。默认值为 NOCREATEUSER。有关更多信息，请参阅 superuser。

SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }

一个子句，它指定用户必须对 Amazon Redshift 系统表和视图具有的访问级别。

拥有 SYSLOG ACCESS RESTRICTED 权限的普通用户在用户可见的系统表和视图中只能查看该用户生成的行。默认值为 RESTRICTED。

拥有 SYSLOG ACCESS UNRESTRICTED 权限的普通用户可以查看用户可见的系统表和视图中的所有行，包括其他用户生成的行。UNRESTRICTED 不向普通用户授予对超级用户可见的表的访问权限。只有超级用户可以查看超级用户可见的表。

> **Note**
>
> 如果向用户授予对系统表的无限制访问权限，用户便可以看到由其他用户生成的数据。例如，STL_QUERY 和 STL_QUERYTEXT 包含 INSERT、UPDATE 和 DELETE 语句的完整文本（其中可能包含敏感的用户生成数据）。

SVV_TRANSACTIONS 中的所有行都对所有用户可见。

有关更多信息，请参阅 系统表和视图中的数据可见性。

IN GROUP groupname

指定用户所属的现有组的名称。可列出多个组名。

VALID UNTIL abstime

VALID UNTIL 选项设置一个绝对时间，用户的密码在该时间后将不再有效。默认情况下，密码没有时间限制。

CONNECTION LIMIT { limit | UNLIMITED }

允许用户同时打开的数据库连接的最大数量。此限制不适用于超级用户。使用 UNLIMITED 关键字设置允许的并行连接的最大数量。对每个数据库的连接数量可能也会施加限制。有关更多信息，请参阅 CREATE DATABASE。默认为 UNLIMITED。要查看当前连接，请查询 STV_SESSIONS 系统视图。

> **Note**
>
> 如果用户及数据库连接限制均适用，当用户尝试连接时，必须有一个同时低于这两个限制的未使用的连接槽可用。

SESSION TIMEOUT limit

会话保持非活动或空闲状态的最长时间（秒）。范围在 60 秒（1 分钟）到 1,728,000 秒（20 天）之间。如果没有为用户设置会话超时，则应用集群设置。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 中的配额和限制。

设置会话超时时，它仅应用于新会话。

要查看有关活动用户会话的信息，包括开始时间、用户名和会话超时，请查询 STV_SESSIONS 系统视图。要查看有关用户会话历史记录的信息，请查询 STL_SESSIONS 视图。要检索有关数据库用户的信息（包括会话超时值），请查询 SVL_USER_INFO 视图。

EXTERNALID external_id

用户的标识符，与身份提供者关联。用户必须已禁用密码。有关更多信息，请参阅 Amazon Redshift 的原生身份提供者 (IdP) 联合身份验证。

使用说明

默认情况下，所有用户都对 PUBLIC schema 具有 CREATE 和 USAGE 权限。要禁止用户在数据库的 PUBLIC schema 中创建对象，请使用 REVOKE 命令删除该权限。

使用 IAM 身份验证创建数据库用户凭证时，您可能希望创建能够仅使用临时凭证登录的超级用户。您不能禁用超级用户的密码，但可以使用随机生成的 MD5 哈希字符串创建一个未知密码。

```
create user iam_superuser password 'md5A1234567890123456780123456789012' createuser;
```

无论 `enable_case_sensitive_identifier` 配置选项的设置如何，系统都会保留双引号括起来的 username 的大小写。有关更多信息，请参阅 enable_case_sensitive_identifier。

示例

以下命令创建一个名为 dbuser 的用户，密码为“abcD1234”，具有数据库创建权限，连接限制为 30。

```
create user dbuser with password 'abcD1234' createdb connection limit 30;
```

查询 PG_USER_INFO 目录表，查看有关数据库用户的详细信息。

```
select * from pg_user_info;
```

```
 usename   | usesysid | usecreatedb | usesuper | usecatupd | passwd   | valuntil |
 useconfig | useconnlimit
-----------+----------+-------------+----------+-----------+----------+----------
+-----------+-------------
 rdsdb     |        1 | true        | true     | true      | ******** | infinity |
      |
 adminuser |      100 | true        | true     | false     | ******** |          |
      | UNLIMITED
 dbuser    |      102 | true        | false    | false     | ******** |          |
      | 30
```

在以下示例中，账户密码在 2017 年 6 月 10 日前有效。

```
create user dbuser with password 'abcD1234' valid until '2017-06-10';
```

以下示例创建一个具有包含特殊字符的区分大小写的密码的用户。

```
create user newman with password '@AbC4321!';
```

要在 MD5 密码中使用一个反斜线（“\”），请在源字符串中用另一个反斜杠对该反斜杠进行转义。以下示例创建一个名为 slashpass 的用户并使用一个反斜杠（“\”）作为密码。

```
select md5('\\'||'slashpass');

md5
----------------------------------
0c983d1a624280812631c5389e60d48c
```

使用 md5 密码创建用户。

```
create user slashpass password 'md50c983d1a624280812631c5389e60d48c';
```

下面的示例创建了一个名为 dbuser 的用户，其空闲会话超时设置为 120 秒。

```
CREATE USER dbuser password 'abcD1234' SESSION TIMEOUT 120;
```

下面的示例创建了一个名为 bob 的用户。命名空间为 myco_aad。这只是一个示例。要成功运行该命令，您必须具有已注册的身份提供商。有关更多信息，请参阅 Amazon Redshift 的原生身份提供者 (IdP) 联合身份验证。

```
CREATE USER myco_aad:bob EXTERNALID "ABC123" PASSWORD DISABLE;
```

CREATE VIEW

在数据库中创建一个视图。视图实际上不是具体化的；每当查询中引用视图时，系统都会运行定义视图的查询。要使用外部表创建视图，请包括 WITH NO SCHEMA BINDING 子句。

要创建标准视图，您需要对基础表或基础视图的访问权限。要查询标准视图，您需要选择针对视图本身的权限，但不需要选择针对基础表的权限。如果您创建的视图引用了其他架构中的表或视图，或者创建的视图引用了实体化视图，则需要使用权限。要查询后期绑定视图，您需要选择针对后期绑定视图本身的权限。您还应该确定后期绑定视图的所有者具有对引用的对象（表、视图或用户定义的函数）的选择特权。有关后期绑定视图的更多信息，请参阅使用说明。

所需的权限

以下是 CREATE VIEW 所需的权限：

- CREATE VIEW：
  - Superuser
  - 具有 CREATE [OR REPLACE] VIEW 权限的用户
- REPLACE VIEW：
  - Superuser
  - 具有 CREATE [OR REPLACE] VIEW 权限的用户
  - 视图拥有者

语法

```
CREATE [ OR REPLACE ] VIEW name [ ( column_name [, ...] ) ] AS query
[ WITH NO SCHEMA BINDING ]
```

参数

OR REPLACE

如果存在同名视图，则将替换视图。您只能将视图替换为生成相同的列集的新查询（使用相同的列名和数据类型）。CREATE OR REPLACE VIEW 将锁定视图以阻止读取和写入，直至操作完成。

替换视图时，将保留它的其他属性（如所有权和授予的权限）。

名称

视图的名称。如果提供 schema 名称（例如，`myschema.myview`），则使用指定 schema 创建视图。否则，将在当前 schema 中创建视图。视图名称必须与同一 schema 中任何其他视图或表的名称不同。

如果指定以“#”开头的视图名称，则视图创建为仅在当前会话中可见的临时视图。

有关有效名称的更多信息，请参阅名称和标识符。您无法在系统数据库 template0、template1、padb_harvest 或 sys:internal 中创建表或视图。

column_name

要用于视图中的列的名称的可选列表。如果未提供列名，则从查询派生列名。可在单个视图中定义的列的最大数目为 1,600。

query

计算结果为表的查询（采用 SELECT 语句的形式）。此表定义视图中的列和行。

WITH NO SCHEMA BINDING

指定视图未绑定到基础数据库对象（例如表和用户定义的函数）的子句。因此，视图与其引用的对象之间不存在依赖关系。即使引用的对象不存在，您也可以创建视图。由于不存在依赖关系，删除或更改引用的对象不会影响视图。在查询视图之前，Amazon Redshift 不会检查依赖关系。要查看有关后期绑定视图的详细信息，请运行 PG_GET_LATE_BINDING_VIEW_COLS 函数。

在包括 WITH NO SCHEMA BINDING 子句时，必须使用 schema 名称来限定 SELECT 语句中引用的表和视图。创建视图时，即使引用的表不存在，schema 也必须存在。例如，以下语句将返回错误。

```
create view myevent as select eventname from event
with no schema binding;
```

以下语句将成功运行。

```
create view myevent as select eventname from public.event
with no schema binding;
```

> **Note**
>
> 无法从视图进行更新、插入或删除操作。

使用说明

后期绑定视图

后期绑定视图不检查基础数据库对象（如表和视图），直至该视图被查询为止。因此，您可以修改或删除基础对象，而不必删除和重新创建视图。如果您删除了基础对象，对后期绑定视图的查询将失败。如果对后期绑定视图的查询引用了基础对象中不存在的列，查询将失败。

如果您删除然后重新创建了后期绑定视图的基础表或视图，将使用默认访问权限创建新对象。您可能需要为将查询视图的用户授予对基础对象的权限。

要创建后期绑定视图，请加入 WITH NO SCHEMA BINDING 子句。以下示例创建一个没有 schema 绑定的视图。

```
create view event_vw as select * from public.event
with no schema binding;
```

```
select * from event_vw limit 1;

eventid | venueid | catid | dateid | eventname     | starttime
--------+---------+-------+--------+---------------+--------------------
      2 |     306 |     8 |   2114 | Boris Godunov | 2008-10-15 20:00:00
```

以下示例显示您可以修改基础表，而不必重新创建视图。

```
alter table event rename column eventname to title;
```

```
select * from event_vw limit 1;

eventid | venueid | catid | dateid | title         | starttime
--------+---------+-------+--------+---------------+--------------------
      2 |     306 |     8 |   2114 | Boris Godunov | 2008-10-15 20:00:00
```

您只能在后期绑定视图中引用 Amazon Redshift Spectrum 外部表。后期绑定视图的一种应用是查询 Amazon Redshift和 Redshift Spectrum 表。例如，您可以使用 UNLOAD 命令将较旧的数据归档至 Amazon S3 。然后，创建一个 Redshift Spectrum 外部表，该表引用 Amazon S3 中的数据并创建一个可查询这两个表的视图。以下示例使用 UNION ALL 子句来联接 Amazon Redshift SALES 表和 Redshift Spectrum SPECTRUM.SALES 表。

```
create view sales_vw as
```

```
select * from public.sales
union all
select * from spectrum.sales
with no schema binding;
```

有关创建 Redshift Spectrum 外部表（包括 SPECTRUM.SALES 表）的更多信息，请参阅Amazon Redshift Spectrum 入门。

从后期绑定视图创建标准视图时，标准视图的定义包含创建标准视图时后期绑定视图的定义。不会跟踪后期绑定视图的依赖关系，因此标准视图中不会跟踪对后期绑定视图的更改。

要更新标准视图以引用后期绑定视图的最新定义，请使用创建标准视图时使用的初始视图定义运行 CREATE OR REPLACE VIEW。

请参阅以下示例，了解如何从后期绑定视图中创建标准视图。

```
create view sales_vw_lbv as
select * from public.sales
with no schema binding;

show view sales_vw_lbv;
                            Show View DDL statement
-------------------------------------------------------------------------------
 create view sales_vw_lbv as select * from public.sales with no schema binding;
(1 row)

create view sales_vw as
select * from sales_vw_lbv;

show view sales_vw;
                                                 Show View DDL statement
-------------------------------------------------------------------------------------------------------
 SELECT sales_vw_lbv.price, sales_vw_lbv."region" FROM (SELECT sales.price,
 sales."region" FROM sales) sales_vw_lbv;
(1 row)
```

请注意，标准视图的 DDL 语句中所示的后期绑定视图是在创建标准视图时定义的，并且不会随着您之后对后期绑定视图所做的任何更改而更新。

示例

示例命令使用一组名为 TICKIT 数据库的对象和数据示例。有关更多信息，请参阅示例数据库。

以下命令从名为 EVENT 的表创建一个名为 myevent 的视图。

```
create view myevent as select eventname from event
where eventname = 'LeAnn Rimes';
```

以下命令从名为 USERS 的表创建一个名为 myuser 的视图。

```
create view myuser as select lastname from users;
```

以下命令从名为 USERS 的表创建或替换一个名为 myuser 的视图。

```
create or replace view myuser as select lastname from users;
```

以下示例创建一个没有 schema 绑定的视图。

```
create view myevent as select eventname from public.event
with no schema binding;
```

DEALLOCATE

取消分配预编译语句。

语法

```
DEALLOCATE [PREPARE] plan_name
```

参数

PREPARE

此关键字是可选的，并且将被忽略。

plan_name

要取消分配的预编译语句的名称。

使用说明

DEALLOCATE 用于取消分配先前准备好的 SQL 语句。如果您未明确取消分配预编译语句，则将在当前会话结束时取消分配该语句。有关预编译语句的更多信息，请参阅PREPARE。

另请参阅

EXECUTE, PREPARE

DECLARE

定义新游标。使用游标从大型查询的结果集中一次性检索几个行。

在提取游标的第一行时，会在领导节点上、内存中或磁盘上具体化整个结果集（如果需要）。由于将游标用于大型结果集可能会降低性能，因此建议使用备用方法（如果可能）。有关更多信息，请参阅 使用游标时的性能注意事项。

您必须在事务块中声明游标。对于每个会话，一次只能打开一个游标。

有关更多信息，请参阅FETCH、CLOSE。

语法

```
DECLARE cursor_name CURSOR FOR query
```

参数

cursor_name

新游标的名称。

query

填充游标的 SELECT 语句。

DECLARE CURSOR 使用说明

如果您的客户端应用程序使用 ODBC 连接，并且您的查询创建的结果集太大，无法存储到内存中，则可使用光标将结果集流式传输到客户端应用程序。在使用游标时，会在领导节点上具体化整个结果集，随后您的客户端可按递增方式提取结果。

> **Note**
>
> 若要在 ODBC 中为 Microsoft Windows 启用游标，请在您用于 Amazon Redshift 的 ODBC DSN 中启用使用声明/取回选项。建议使用 ODBC DSN 选项对话框中的 Cache Size 字段将多

节点集群中的 ODBC 缓存大小设置为 4,000 或更大值，以最大程度地减少往返操作。在单节点集群上，将缓存大小设置为 1000。

由于使用游标可能会对性能产生负面影响，我们建议您尽可能地使用替代方法。有关更多信息，请参阅使用游标时的性能注意事项。

支持 Amazon Redshift 游标，但存在以下限制：

- 对于每个会话，一次只能打开一个游标。
- 游标必须在事务内部使用 (BEGIN … END)。
- 所有游标的累计最大结果集大小受到集群节点类型的限制。如果您需要更大的结果集，可以调整为 XL 和 8XL 节点配置。

  有关更多信息，请参阅 游标约束。

游标约束

在提取游标的第一行时，会在领导节点上具体化整个结果集。如果结果集无法存储到内存中，则会根据需要写入磁盘。为了保护领导节点的完整性，Amazon Redshift 会根据集群节点类型对所有游标结果集的大小强制实施约束。

下表显示每个集群节点类型的最大结果集总大小。最大结果集大小以 MB 为单位。

节点类型	每个集群的最大结果集大小 (MB)
RA3 16XL 多个节点	14400000
DC2 Large 单节点	8000
DC2 Large 多节点	192000
DC2 8XL 多节点	3200000
RA3 4XL 多个节点	3200000
RA3 XLPLUS 多节点	1000000
RA3 XLPLUS 单节点	64000

节点类型	每个集群的最大结果集大小 (MB)
Amazon Redshift Serverless	150000

要查看某个集群的活动游标配置，请以超级用户身份查询 STV_CURSOR_CONFIGURATION 系统表。要查看活动光标的状态，请查询 STV_ACTIVE_CURSORS 系统表。用户只能看到自己游标所对应的行，而超级用户可以查看所有游标。

使用游标时的性能注意事项

由于在开始将结果返回给客户端之前，游标会在领导节点上具体化整个结果集，因此对超大型结果集使用游标时，会对性能产生负面影响。我们强烈建议不要对超大型结果集使用游标。在一些情况下，例如您的应用程序使用 ODBC 连接时，游标可能是唯一可行的解决方案。我们建议尽可能地利用下面这些备选方案：

- 使用 UNLOAD 导出大型表。在使用 UNLOAD 时，计算节点将并行工作，从而将数据直接发送到 Amazon Simple Storage Service 上的数据文件。有关更多信息，请参阅 卸载数据。
- 在您的客户端应用程序中设置 JDBC 提取大小参数。如果您使用 JDBC 连接，并且遇到客户端内存不足错误，则可以通过设置 JDBC 提取大小参数，让您的客户端按较小的批次检索结果集。有关更多信息，请参阅 设置 JDBC 提取大小参数。

DECLARE CURSOR 示例

以下示例声明一个名为 LOLLAPALOOZA 的游标，以便为 Lollapalooza 事件选择销售信息，然后使用该游标从结果集中提取行：

```
-- Begin a transaction

begin;

-- Declare a cursor

declare lollapalooza cursor for
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Lollapalooza';

-- Fetch the first 5 rows in the cursor lollapalooza:
```

```
fetch forward 5 from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-05-01 19:00:00 |   92.00000000 |       3
 Lollapalooza | 2008-11-15 15:00:00 |  222.00000000 |       2
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       3
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       4
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       1
(5 rows)

-- Fetch the next row:

fetch next from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-10-06 14:00:00 |  114.00000000 |       2

-- Close the cursor and end the transaction:

close lollapalooza;
commit;
```

以下示例使用表中的所有结果遍历 refcursor：

```
CREATE TABLE tbl_1 (a int, b int);
INSERT INTO tbl_1 values (1, 2),(3, 4);

CREATE OR REPLACE PROCEDURE sp_cursor_loop() AS $$
DECLARE
    target record;
    curs1 cursor for select * from tbl_1;
BEGIN
    OPEN curs1;
    LOOP
        fetch curs1 into target;
        exit when not found;
        RAISE INFO 'a %', target.a;
    END LOOP;
    CLOSE curs1;
END;
```

```
$$ LANGUAGE plpgsql;

CALL sp_cursor_loop();

SELECT message
   from svl_stored_proc_messages
   where querytxt like 'CALL sp_cursor_loop()%';

  message
----------
      a 1
      a 3
```

删除

从表中删除行。

> **Note**
>
> 单个 SQL 语句的最大大小为 16MB。

语法

```
[ WITH [RECURSIVE] common_table_expression [, common_table_expression , ...] ]
DELETE [ FROM ] { table_name | materialized_view_name }
    [ { USING } table_name, ... ]
    [ WHERE condition ]
```

参数

WITH 子句

可选子句，指定一个或多个 common-table-expressions。请参阅 WITH 子句。

FROM

FROM 关键字是可选的，不过在指定 USING 子句时除外。语句 `delete from event;` 和 `delete event;` 执行相同的操作，可从 EVENT 表中删除所有行。

> **Note**
>
> 要从表中删除所有行，请对表执行 TRUNCATE。TRUNCATE 的效率要比 DELETE 高很多，不需要 VACUUM 和 ANALYZE。不过请注意，TRUNCATE 在其运行的事务中提交事务。

table_name

一个临时或永久表。只有表的所有者或对表具有 DELETE 权限的用户才能从表中删除行。

考虑对大型表使用 TRUNCATE 命令来快速执行非限定的删除操作；请参阅 TRUNCATE。

> **Note**
>
> 在从表中删除大量行之后：
>
> - 对表执行 Vacuum 操作，以回收存储空间并对行重新排序。
> - 分析表以更新查询计划程序的统计数据。

materialized_view_name

实体化视图。DELETE 语句适用于用于串流摄取的实体化视图。只有实体化视图的所有者或对实体化视图具有 DELETE 权限的用户才能从中删除行。

如果行级别安全性 (RLS) 策略未向用户授予 IGNORE RLS 权限，则您无法在用于串流摄取的实体化视图上运行 DELETE。但有一个例外：如果向执行 DELETE 操作的用户授予了 IGNORE RLS，则此操作会成功运行。有关更多信息，请参阅 RLS 策略拥有权和管理。

USING table_name, ...

在 WHERE 子句条件中引用附加表时，使用 USING 关键字可以引入表列表。例如，以下语句从 EVENT 表中删除满足 EVENT 和 SALES 表上的联接条件的所有行。必须在 FROM 列表中明确指定 SALES 表：

```
delete from event using sales where event.eventid=sales.eventid;
```

如果您在 USING 子句中重复目标表名称，DELETE 操作将运行自联接。您可以在 WHERE 子句中使用子查询，以取代 USING 语法来编写相同的查询。

WHERE condition

一个限制只删除那些符合条件的行的可选子句。例如，条件可以是对列的限制，也可以是联接条件或基于查询结果的条件。查询可以引用 DELETE 命令的目标以外的其他表。例如：

```
delete from t1
where col1 in(select col2 from t2);
```

如果未指定条件，将删除表中的所有行。

示例

从 CATEGORY 表中删除所有行：

```
delete from category;
```

从 CATEGORY 表中删除 CATID 值在 0 与 9 之间的行：

```
delete from category
where catid between 0 and 9;
```

从 LISTING 表中删除其 SELLERID 值在 SALES 表中不存在的行：

```
delete from listing
where listing.sellerid not in(select sales.sellerid from sales);
```

以下两个查询均根据与 EVENT 表的联接以及 CATID 列的额外限制，从 CATEGORY 表中删除一行：

```
delete from category
using event
where event.catid=category.catid and category.catid=9;
```

```
delete from category
where catid in
(select category.catid from category, event
where category.catid=event.catid and category.catid=9);
```

以下查询从 mv_cities 实体化视图中删除所有行。此示例中的实体化视图名称是一个示例：

```
delete from mv_cities;
```

DESC DATASHARE

显示使用 ALTER DATASHARE 添加到数据共享的数据库对象的列表。Amazon Redshift 显示表、视图和函数的名称、数据库、schema 和类型。

使用系统视图可以找到有关数据共享对象的其他信息。有关更多信息，请参阅 SVV_DATASHARE_OBJECTS 和 SVV_DATASHARES。

语法

```
DESC DATASHARE datashare_name [ OF [ ACCOUNT account_id ] NAMESPACE namespace_guid ]
```

参数

datashare_name

数据共享的名称。

NAMESPACE namespace_guid

指定数据共享使用的命名空间的值。当您以使用者集群管理员身份运行 DESC DATAHSARE 时，请指定 NAMESPACE 参数以查看入站数据共享。

ACCOUNT account_id

它指定数据共享所属的账户的值。

使用说明

作为使用者账户管理员，当您运行 DESC DATASHARE 以查看 AWS 账户内的入站数据共享时，请指定 NAMESPACE 选项。当您运行 DESC DATASHARE 以查看跨 AWS 账户的入站数据共享时，请指定 ACCOUNT 和 NAMESPACE 选项。

示例

以下示例显示生产者集群上出站数据共享的信息。

```
DESC DATASHARE salesshare;
```

```
producer_account |          producer_namespace          | share_type  | share_name   |
 object_type |        object_name           |  include_new
-----------------+--------------------------------------+-------------+--------------
+-------------+------------------------------+--------------
 123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d  | OUTBOUND    |  salesshare  |
 TABLE       | public.tickit_sales_redshift |
 123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d  | OUTBOUND    |  salesshare  |
 SCHEMA      | public                       |   t
```

以下示例显示使用者集群上入站数据共享的信息。

```
DESC DATASHARE salesshare of ACCOUNT '123456789012' NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';

 producer_account |          producer_namespace          | share_type | share_name |
 object_type |         object_name          |  include_new
-----------------+--------------------------------------+------------+------------
+-------------+------------------------------+--------------
 123456789012     | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare |
 table       | public.tickit_sales_redshift |
 123456789012     | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare |
 schema      | public                       |
(2 rows)
```

DESC IDENTITY PROVIDER

显示有关身份提供者的信息。只有超级用户可以描述身份提供者。

语法

```
DESC IDENTITY PROVIDER identity_provider_name
```

参数

identity_provider_name

身份提供者的名称。

示例

以下示例显示有关您的身份提供者的信息。

```
DESC IDENTITY PROVIDER azure_idp;
```

示例输出。

```
  uid   |   name    | type  |             instanceid              | namespc |

                                          params

                          | enabled
--------+-----------+-------+-------------------------------------+---------
+-----------------------------------------------------------------------------------------------------------------
+---------
 126692 | azure_idp | azure | e40d4bb2-7670-44ae-bfb8-5db013221d73 | aad     |
 {"issuer":"https://login.microsoftonline.com/e40d4bb2-7670-44ae-bfb8-5db013221d73/
v2.0", "client_id":"871c010f-5e61-4fb1-83ac-98610a7e9110", "client_secret":'',
 "audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift", "https://
analysis.windows.net/powerbi/connector/AWSRDS"]} | t
(1 row)
```

DETACH MASKING POLICY

将已附加的动态数据掩蔽策略与列分离。有关动态数据掩蔽的更多信息，请参阅 动态数据掩蔽。

超级用户和具有 sys:secadmin 角色的用户或角色可以分离策略。

语法

```
DETACH MASKING POLICY policy_name
   ON { table_name }
   ( output_column_names )
   FROM { user_name | ROLE role_name | PUBLIC };
```

参数

policy_name

要分离的屏蔽策略的名称。

table_name

要从中分离屏蔽策略的表的名称。

output_column_names

附加了屏蔽策略的列的名称。

user_name

附加了屏蔽策略的用户的名称。

在单个 DETACH MASKING POLICY 语句中，您只能设置 user_name、role_name 和 PUBLIC 中的一项。

role_name

附加了屏蔽策略的角色的名称。

在单个 DETACH MASKING POLICY 语句中，您只能设置 user_name、role_name 和 PUBLIC 中的一项。

PUBLIC

显示策略已附加到表中的所有用户。

在单个 DETACH MASKING POLICY 语句中，您只能设置 user_name、role_name 和 PUBLIC 中的一项。

DETACH RLS POLICY

将表上的行级别安全性策略与一个或多个用户或角色分离。

超级用户和具有 sys:secadmin 角色的用户或角色可以分离策略。

语法

```
DETACH RLS POLICY policy_name ON [TABLE] table_name [, ...]
FROM { user_name | ROLE role_name | PUBLIC } [, ...]
```

参数

policy_name

策略的名称。

ON [TABLE]table_name [, ...]

行级别安全性策略与之分离的表或视图。

FROM { user_name | ROLE role_name | PUBLIC} [, ...]

指定策略是否与一个或多个指定的用户或角色分离。

使用说明

在使用 DETACH RLS POLICY 语句时，请遵守以下规则：

- 您可以将策略与关系、用户、角色或公共分离。

示例

以下示例将表上的策略与角色分离。

```
DETACH RLS POLICY policy_concerts ON tickit_category_redshift FROM ROLE analyst, ROLE
 dbadmin;
```

DROP DATABASE

删除数据库。

您不能在以下事务块中运行 DROP DATABASE：(BEGIN ... END)。有关事务的更多信息，请参阅 可序列化的隔离。

语法

```
DROP DATABASE database_name
```

参数

database_name

要删除的数据库的名称。您不能删除 dev、padb_harvest、template0、template1 或 sys:internal 数据库，并且不能删除当前数据库。

要删除外部数据库，请删除外部架构。有关更多信息，请参阅 DROP SCHEMA。

DROP DATABASE 使用说明

使用 DROP DATABASE 语句时，请注意以下事项：

- 通常，我们不建议您使用 DROP DATABASE 语句删除包含 AWS Data Exchange 数据共享的数据库。如果您这样做的话，有权访问数据共享的 AWS 账户 将失去访问权限。执行这种类型的更改可能会违反 AWS Data Exchange 中的数据产品条款。

  以下示例显示了删除包含 AWS Data Exchange 数据共享的数据库会出现的错误。

  ```
  DROP DATABASE test_db;
  ERROR:   Drop of database test_db that contains ADX-managed datashare(s)
   requires session variable datashare_break_glass_session_var to be set to value
   'ce8d280c10ad41'
  ```

  要允许删除数据库，请设置以下变量，然后再次运行 DROP DATABASE 语句。

  ```
  SET datashare_break_glass_session_var to 'ce8d280c10ad41';
  ```

  ```
  DROP DATABASE test_db;
  ```

  在这种情况下，Amazon Redshift 会生成一个随机的一次性值来设置会话变量，以允许对包含 AWS Data Exchange 数据共享的数据库执行 DROP DATABASE。

示例

以下示例删除名为 TICKIT_TEST 的数据库：

```
drop database tickit_test;
```

DROP DATASHARE

删除数据共享。此命令无法撤消。

只有超级用户或数据共享拥有者才可以删除数据共享。

所需的权限

以下是 DROP DATASHARE 所需的权限：

- Superuser
- 具有 DROP DATASHARE 权限的用户
- 数据共享拥有者

语法

```
DROP DATASHARE datashare_name;
```

参数

datashare_name

要删除的数据共享的名称。

DROP DATASHARE 使用说明

使用 DROP DATASHARE 语句时，请注意以下事项：

- 通常，我们不建议您使用 DROP DATASHARE 语句删除 AWS Data Exchange 数据共享。如果您这样做的话，有权访问数据共享的 AWS 账户 将失去访问权限。执行这种类型的更改可能会违反 AWS Data Exchange 中的数据产品条款。

  以下示例显示了删除 AWS Data Exchange 数据共享时会出现的错误。

  ```
  DROP DATASHARE salesshare;
  ERROR:  Drop of ADX-managed datashare salesshare requires session variable
   datashare_break_glass_session_var to be set to value '620c871f890c49'
  ```

  要允许删除 AWS Data Exchange 数据共享，请设置以下变量，然后再次运行 DROP DATASHARE 语句。

  ```
  SET datashare_break_glass_session_var to '620c871f890c49';
  ```

  ```
  DROP DATASHARE salesshare;
  ```

  在这种情况下，Amazon Redshift 会生成一个随机的一次性值来设置会话变量，以允许对 AWS Data Exchange 数据共享执行 DROP DATASHARE。

示例

以下示例将删除名为 salesshare 的数据共享。

```
DROP DATASHARE salesshare;
```

DROP EXTERNAL VIEW（预览版）

以下是预览版 Data Catalog for Amazon Redshift 中的预发行文档视图。文档和特征都可能会更改。我们建议您只在测试集群中使用此功能，而不要在生产环境中使用。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

您可以在预览版中创建 Amazon Redshift 集群，以便测试 Amazon Redshift 的新功能。您无法在生产环境中使用这些功能，也无法将预览版集群移动到生产集群或另一个跟踪上的集群。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

在预览版中创建集群

1. 登录 AWS Management Console，然后通过以下网址打开 Amazon Redshift 控制台：https://console.aws.amazon.com/redshiftv2/。
2. 在导航菜单上，选择预置集群控制面板，然后选择集群。列出您的账户在当前 AWS 区域 区域中的集群。列表中的各个列中显示了每个集群的一部分属性。
3. 集群列表页面上会显示一个横幅，其中介绍了预览版。选择创建预览版集群按钮以打开创建集群页面。
4. 输入集群的属性。选择包含要测试的功能的预览版跟踪。我们建议输入的集群名称指明要对该集群进行预览版跟踪。为您的集群选择选项，包括标记为 -preview 的选项，用于要测试的功能。有关创建集群的一般信息，请参阅《Amazon Redshift 管理指南》中的创建集群。
5. 选择创建集群以在预览模式下创建集群。

   > **Note**
   >
   > preview_2023 跟踪是最新可用的预览版跟踪。此版本仅支持创建具有 RA3 节点类型的集群。不支持节点类型 DC2 以及任何更早的节点类型。

6. 当您的预览集群可用时，使用 SQL 客户端加载和查询数据。

Data Catalog 视图预览功能仅在以下区域中可用。

- 美国东部（俄亥俄州）(us-east-2)
- 美国东部（弗吉尼亚州北部）(us-east-1)
- 美国西部（北加利福尼亚）(us-west-1)
- 亚太地区（东京）(ap-northeast-1)
- 欧洲地区（爱尔兰）(eu-west-1)
- 欧洲地区（斯德哥尔摩）(eu-north-1)

您也可以创建预览工作组来测试 Data Catalog 视图。您无法在生产中使用这些功能，也无法将您的工作组移至其他工作组。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。有关如何创建预览工作组的说明，请参阅https://docs.aws.amazon.com/redshift/latest/mgmt/serverless-workgroup-preview.html。

从数据库中删除外部视图。删除外部视图会将其从与该视图关联的所有 SQL 引擎（例如 Amazon Athena 和 Amazon EMR Spark）中删除。此命令无法逆转。有关 Data Catalog 视图的更多信息，请参阅创建 Data Catalog 视图（预览版）。

语法

```
DROP EXTERNAL VIEW schema_name.view_name [ IF EXISTS ]
{catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name |
 external_schema_name.view_name}
```

参数

schema_name.view_name

附加到 AWS Glue 数据库的架构，后面是视图的名称。

IF EXISTS

仅当视图存在时才会将其删除。

catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name | external_schema_name.view_name

删除视图时要使用的架构符号。可以指定使用您创建的 Glue 数据库 AWS Glue Data Catalog 或您创建的外部架构。有关更多信息，请参阅 CREATE DATABASE 和 CREATE EXTERNAL SCHEMA。

query_definition

Amazon Redshift 为更改视图而运行的 SQL 查询的定义。

示例

以下示例删除了一个名为 sample_schema.glue_data_catalog_view 的 Data Catalog 视图。

```
DROP EXTERNAL VIEW sample_schema.glue_data_catalog_view IF EXISTS
```

DROP FUNCTION

从数据库中删除用户定义的函数 (UDF)。由于可能存在名称相同但签名不同的多个函数，因此必须指定函数签名或参数数据类型列表。不能删除 Amazon Redshift 内置函数。

此命令无法撤消。

所需的权限

以下是 DROP FUNCTION 所需的权限：

- Superuser
- 具有 DROP FUNCTION 权限的用户
- 函数拥有者

语法

```
DROP FUNCTION name
( [arg_name] arg_type   [, ...] )
[ CASCADE | RESTRICT ]
```

参数

name

要删除的函数的名称。

arg_name

输入参数的名称。由于在确定函数身份时只需要参数数据类型，因此 DROP FUNCTION 会忽略参数名称。

arg_type

输入参数的数据类型。您可以通过逗号分隔的列表形式最多提供 32 个数据类型。

CASCADE

一个关键字，用于自动删除依赖于该函数的对象，例如视图。

要创建不依赖于函数的视图，请在定义视图时包括 WITH NO SCHEMA BINDING 子句。有关更多信息，请参阅 CREATE VIEW。

RESTRICT

一个关键字，用于指定如果有任何对象依赖于该函数，则不删除函数并返回一条消息。此操作是默认操作。

示例

以下示例删除名为 `f_sqrt` 的函数：

```
drop function f_sqrt(int);
```

要删除具有依赖项的函数，请使用 CASCADE 选项，如以下示例所示：

```
drop function f_sqrt(int)cascade;
```

DROP GROUP

删除用户组。此命令无法撤消。此命令不删除组中的单个用户。

要删除单个用户，请参阅 DROP USER。

语法

```
DROP GROUP name
```

参数

名称

要删除的用户组的名称。

示例

以下示例删除 guests 用户组：

```
DROP GROUP guests;
```

如果组具有对象的任何权限，则不能删除组。如果您尝试删除这样的组，将收到以下错误。

```
ERROR: group "guests" can't be dropped because the group has a privilege on some object
```

如果组具有对象的权限，则必须撤消权限，然后再删除组。要查找 guests 组有权访问的对象，请使用以下示例。有关示例中使用的元数据视图的更多信息，请参阅 SVV_RELATION_PRIVILEGES。

```
SELECT DISTINCT namespace_name, relation_name, identity_name, identity_type
FROM svv_relation_privileges
WHERE identity_type='group' AND identity_name='guests';

+----------------+---------------+---------------+---------------+
| namespace_name | relation_name | identity_name | identity_type |
+----------------+---------------+---------------+---------------+
| public         | table1        | guests        | group         |
+----------------+---------------+---------------+---------------+
| public         | table2        | guests        | group         |
+----------------+---------------+---------------+---------------+
```

以下示例从 guests 用户组撤消对 public schema 中的所有表的所有权限，然后删除该组。

```
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM GROUP guests;
DROP GROUP guests;
```

DROP IDENTITY PROVIDER

删除身份提供者。此命令无法撤消。只有超级用户可以删除身份提供者。

语法

```
DROP IDENTITY PROVIDER identity_provider_name [ CASCADE ]
```

参数

identity_provider_name

要删除的身份提供者的名称。

CASCADE

删除身份提供者时，会删除附加到身份提供者的用户和角色。

示例

以下示例删除 oauth_provider 身份提供者。

```
DROP IDENTITY PROVIDER oauth_provider;
```

如果删除身份提供者，某些用户可能无法登录或无法使用配置为使用身份提供者的客户端工具。

DROP LIBRARY

从数据库中删除自定义 Python 库。只有库所有者或超级用户可以删除库。

DROP LIBRARY 无法在事务块 (BEGIN … END) 中运行。有关事务的更多信息，请参阅 可序列化的隔离。

此命令无法撤消。DROP LIBRARY 命令将立即提交。如果依赖于所删除库的 UDF 正在并行运行，UDF 可能失败，即使 UDF 正在事务中运行也是如此。

有关更多信息，请参阅 CREATE LIBRARY。

所需的权限

以下是 DROP LIBRARY 所需的权限：

- Superuser
- 具有 DROP LIBRARY 权限的用户

- 库拥有者

语法

```
DROP LIBRARY library_name
```

参数

library_name

库的名称。

DROP MASKING POLICY

从所有数据库中删除动态数据掩蔽策略。您无法删除仍然附加到一个或多个表的屏蔽策略。有关动态数据掩蔽的更多信息，请参阅 动态数据掩蔽。

超级用户和具有 sys:secadmin 角色的用户或角色可以删除屏蔽策略。

语法

```
DROP MASKING POLICY policy_name;
```

参数

policy_name

要删除的屏蔽策略的名称。

DROP MODEL

从数据库中删除模型。只有模型拥有者或超级用户才可以删除模型。

DROP MODEL 还会删除从此模型派生的所有相关预测函数、与模型相关的所有 Amazon Redshift 构建以及与模型相关的所有 Amazon S3 数据。当模型仍在 Amazon SageMaker 中进行训练时，DROP MODEL 将取消这些操作。

此命令无法撤消。DROP MODEL 命令将立即提交。

所需的权限

以下是 DROP MODEL 所需的权限：

- Superuser
- 具有 DROP MODEL 权限的用户
- 模型拥有者
- Schema 拥有者

语法

```
DROP MODEL [ IF EXISTS ] model_name
```

参数

IF EXISTS

一个子句，指示如果指定 schema 已存在，则此命令不应进行任何更改，并应返回一条指示 schema 存在的消息。

model_name

模型的名称。schema 中的模型名称必须是唯一的。

示例

以下示例删除模型 demo_ml.customer_churn。

```
DROP MODEL demo_ml.customer_churn
```

DROP MATERIALIZED VIEW

删除实体化视图。

有关实体化视图的更多信息，请参阅在 Amazon Redshift 中创建实体化视图。

语法

```
DROP MATERIALIZED VIEW [ IF EXISTS ] mv_name [ CASCADE | RESTRICT ]
```

参数

IF EXISTS

一个子句，旨在检查指定的实体化视图是否存在。如果实体化视图不存在，则 DROP MATERIALIZED VIEW 命令会返回一条错误消息。此子句在编写脚本时非常有用，可以防止在删除不存在的实体化视图时出现脚本失败的情况。

mv_name

要删除的实体化视图的名称。

CASCADE

一个子句，用于指示自动删除实体化视图所依赖的对象，例如其他视图。

RESTRICT

一个子句，用于指示如果有任何对象依赖该实体化视图，则不删除该视图。这是默认模式。

使用说明

仅实体化视图的拥有者才能对该视图使用 DROP MATERIALIZED VIEW。超级用户或被特别授予 DROP 权限的用户可以是例外。

当您为实体化视图编写 drop 语句并且存在具有匹配名称的视图时，会导致错误，指示您使用 DROP VIEW。即使在您使用 DROP MATERIALIZED VIEW IF EXISTS 的情况下也会发生这一错误。

示例

以下示例删除 tickets_mv 实体化视图。

```
DROP MATERIALIZED VIEW tickets_mv;
```

DROP PROCEDURE

删除过程。要删除过程，需要提供过程名称和输入参数数据类型（签名）。（可选）您可以包含完整的参数数据类型，包括 OUT 参数。要查找过程的签名，请使用 SHOW PROCEDURE 命令。有关过程签名的更多信息，请参阅PG_PROC_INFO。

所需的权限

以下是 DROP PROCEDURE 所需的权限：

- Superuser
- 具有 DROP PROCEDURE 权限的用户
- 过程拥有者

语法

```
DROP PROCEDURE sp_name ( [ [ argname ] [ argmode ] argtype [, ...] ] )
```

参数

sp_name

要删除的过程的名称。

argname

输入参数的名称。由于在确定过程身份时只需要参数数据类型，因此 DROP PROCEDURE 会忽略参数名称。

argmode

参数的模式，可以是 IN、OUT 或 INOUT。OUT 参数是可选的，因为它们不用于标识存储过程。

argtype

输入参数的数据类型。有关支持的数据类型的列表，请参阅数据类型。

示例

以下示例删除名为 `quarterly_revenue` 的存储过程。

```
DROP PROCEDURE quarterly_revenue(volume INOUT bigint, at_price IN numeric,result OUT
 int);
```

DROP RLS POLICY

删除所有数据库中所有表的行级别安全性策略。

超级用户和具有 sys:secadmin 角色的用户或角色可以删除策略。

语法

```
DROP RLS POLICY [ IF EXISTS ] policy_name [ CASCADE | RESTRICT ]
```

参数

IF EXISTS

指示指定策略是否已存在的子句。

policy_name

策略的名称。

CASCADE

一个子句，用于指示在删除策略之前自动将策略与所有附加的表分离。

RESTRICT

一个子句，用于指示在将策略附加到某些表时不删除策略。这是默认模式。

示例

以下示例删除了行级别安全性策略。

```
DROP RLS POLICY policy_concerts;
```

DROP ROLE

从数据库中删除角色。只有创建角色的角色拥有者、使用 WITH ADMIN 选项的用户或超级用户才能删除角色。

您不能删除已授予用户的角色或依赖此角色的其他角色。

所需的权限

以下是 DROP ROLE 所需的权限：

- Superuser
- 角色拥有者，是创建角色的用户，或者是已被授予具有 WITH ADMIN OPTION 权限的角色的用户。

语法

```
DROP ROLE role_name [ FORCE | RESTRICT ]
```

参数

role_name

角色的名称。

[FORCE | RESTRICT]

默认设置为 RESTRICT。当您尝试删除继承了其他角色的角色时，Amazon Redshift 会返回错误。使用 FORCE 删除所有角色分配（如果存在）。

示例

下面的示例将删除角色 sample_role。

```
DROP ROLE sample_role FORCE;
```

下面的示例尝试使用默认 RESTRICT 选项删除授予用户的角色 sample_role1。

```
CREATE ROLE sample_role1;
GRANT sample_role1 TO user1;
DROP ROLE sample_role1;
ERROR:  cannot drop this role since it has been granted on a user
```

要成功删除已授予用户的 sample_role1，请使用 FORCE 选项。

```
DROP ROLE sample_role1 FORCE;
```

下面的示例尝试使用默认 RESTRICT 选项删除另一个角色依赖的角色 sample_role2。

```
CREATE ROLE sample_role1;
CREATE ROLE sample_role2;
GRANT sample_role1 TO sample_role2;
DROP ROLE sample_role2;
ERROR:  cannot drop this role since it depends on another role
```

要成功删除被另一个角色依赖的 sample_role2，请使用 FORCE 选项。

```
DROP ROLE sample_role2 FORCE;
```

DROP SCHEMA

删除 schema。对于外部架构，您还可以删除与该架构关联的外部数据库。此命令无法撤消。

所需的权限

以下是 DROP SCHEMA 所需的权限：

- Superuser
- Schema 拥有者
- 具有 DROP SCHEMA 权限的用户

语法

```
DROP SCHEMA [ IF EXISTS ] name [, ...]
[ DROP EXTERNAL DATABASE ]
[ CASCADE | RESTRICT ]
```

参数

IF EXISTS

一个子句，指示如果指定的 schema 不存在，则命令不应进行任何更改，并返回一条指示 schema 不存在的消息，而不是以错误终止。

此子句在编写脚本时很有用，可使在 DROP SCHEMA 针对不存在的 schema 运行时脚本不会失败。

名称

要删除的架构的名称。您可以指定以逗号分隔的多个架构名称。

DROP EXTERNAL DATABASE

这个子句指示，如果删除外部架构，则删除与它关联的外部数据库（如果存在）。如果不存在外部数据库，则该命令将返回一条消息，说明不存在外部数据库。如果删除多个外部架构，则删除与指定架构关联的所有数据库。

如果外部数据库包含从属对象（如表），则还应包括 CASCADE 选项以删除从属对象。

删除外部数据库时，还会删除与该数据库关联的任何其他外部架构的数据库。使用该数据库的其他外部架构中定义的表也将被删除。

DROP EXTERNAL DATABASE 不支持存储在 HIVE 元存储中的外部数据库。

CASCADE

一个关键字，指示自动删除架构中所有对象。如果指定了 DROP EXTERNAL DATABASE，则还会删除外部数据库中的所有对象。

RESTRICT

一个关键字，指示如果架构中包含任何对象，则不删除该架构或外部数据库。此操作是默认操作。

示例

以下示例删除名为 S_SALES 的架构。本示例使用 RESTRICT 作为安全机制，以便在 schema 包含对象的情况下不会将其删除。在这种情况下，您需要先删除架构对象，然后再删除架构。

```
drop schema s_sales restrict;
```

以下示例删除名为 S_SALES 的架构以及依赖该架构的所有对象。

```
drop schema s_sales cascade;
```

以下示例删除 S_SALES schema（如果存在）；如果不存在该 schema，则不执行任何操作并返回一条消息。

```
drop schema if exists s_sales;
```

以下示例删除名为 S_SPECTRUM 的外部架构以及与之关联的外部数据库。此示例使用 RESTRICT，以便在架构和数据库包含任何对象时不会将其删除。在这种情况下，您需要先删除从属对象，然后再删除架构和数据库。

```
drop schema s_spectrum drop external database restrict;
```

以下示例删除多个架构、与之关联的外部数据库以及任何从属对象。

```
drop schema s_sales, s_profit, s_revenue drop external database cascade;
```

DROP TABLE

从数据库中删除表。

如果您正在尝试清空表中的行，而不是删除表，请使用 DELETE 或 TRUNCATE 命令。

DROP TABLE 删除目标表上存在的约束。可以使用一条 DROP TABLE 命令删除多个表。

针对外部表的 DROP TABLE 不能在事务 (BEGIN … END) 内运行。有关事务的更多信息，请参阅 可序列化的隔离。

要查找向组授予 DROP 特权的示例，请参阅 GRANT 示例。

所需的权限

以下是 DROP TABLE 所需的权限：

- Superuser
- 具有 DROP TABLE 权限的用户
- 对模式拥有 USAGE 权限的表拥有者

语法

```
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

参数

IF EXISTS

一个子句，指示如果指定的表不存在，则命令不应进行任何更改，并返回一条指示表不存在的消息，而不是以错误终止。

此子句在编写脚本时很有用，可使在 DROP TABLE 针对不存在的表运行时脚本不会失败。

名称

要删除的表的名称。

CASCADE

一个子句，用于指示自动删除依赖该表的对象，例如视图。

要使创建的视图不依赖于其他数据库对象（例如视图和表），请在定义视图时包括 WITH NO SCHEMA BINDING 子句。有关更多信息，请参阅 CREATE VIEW。

RESTRICT

一个子句，指示如果任何对象依赖该表，则不删除该表。此操作是默认操作。

示例

删除没有依赖项的表

以下示例创建一个名为 FEEDBACK 且没有依赖项的表，然后删除该表：

```
create table feedback(a int);

drop table feedback;
```

如果表包含由视图或其他表引用的列，Amazon Redshift 将显示一条消息，如下所示。

```
Invalid operation: cannot drop table feedback because other objects depend on it
```

同时删除两个表

以下命令集创建一个 FEEDBACK 表和一个 BUYERS 表，然后在一条命令中同时删除这两个表：

```
create table feedback(a int);

create table buyers(a int);

drop table feedback, buyers;
```

删除具有依赖项的表

以下步骤说明如何使用 CASCADE 开关删除名为 FEEDBACK 的表。

首先，使用 CREATE TABLE 命令创建一个名为 FEEDBACK 的简单表：

```
create table feedback(a int);
```

下一步，使用 CREATE VIEW 命令创建一个名为 FEEDBACK_VIEW 的视图，并使该视图依赖于 FEEDBACK 表：

```
create view feedback_view as select * from feedback;
```

以下示例删除 FEEDBACK 表，同时会删除 FEEDBACK_VIEW 视图，因为 FEEDBACK_VIEW 视图依赖于 FEEDBACK 表：

```
drop table feedback cascade;
```

查看表的依赖项

要返回表的依赖关系，请使用以下示例。用您自己的架构和表替换 *my_schema* 和 *my_table*。

```
SELECT dependent_ns.nspname as dependent_schema
, dependent_view.relname as dependent_view
, source_ns.nspname as source_schema
, source_table.relname as source_table
, pg_attribute.attname as column_name
FROM pg_depend
JOIN pg_rewrite ON pg_depend.objid = pg_rewrite.oid
JOIN pg_class as dependent_view ON pg_rewrite.ev_class = dependent_view.oid
JOIN pg_class as source_table ON pg_depend.refobjid = source_table.oid
JOIN pg_attribute ON pg_depend.refobjid = pg_attribute.attrelid
    AND pg_depend.refobjsubid = pg_attribute.attnum
JOIN pg_namespace dependent_ns ON dependent_ns.oid = dependent_view.relnamespace
JOIN pg_namespace source_ns ON source_ns.oid = source_table.relnamespace
WHERE
source_ns.nspname = 'my_schema'
AND source_table.relname = 'my_table'
AND pg_attribute.attnum > 0
ORDER BY 1,2
LIMIT 10;
```

要删除 *my_table* 及其依赖关系，请使用以下示例。此示例还返回已删除的表的所有依赖关系。

```
DROP TABLE my_table CASCADE;

SELECT dependent_ns.nspname as dependent_schema
, dependent_view.relname as dependent_view
, source_ns.nspname as source_schema
, source_table.relname as source_table
, pg_attribute.attname as column_name
FROM pg_depend
```

```
JOIN pg_rewrite ON pg_depend.objid = pg_rewrite.oid
JOIN pg_class as dependent_view ON pg_rewrite.ev_class = dependent_view.oid
JOIN pg_class as source_table ON pg_depend.refobjid = source_table.oid
JOIN pg_attribute ON pg_depend.refobjid = pg_attribute.attrelid
    AND pg_depend.refobjsubid = pg_attribute.attnum
JOIN pg_namespace dependent_ns ON dependent_ns.oid = dependent_view.relnamespace
JOIN pg_namespace source_ns ON source_ns.oid = source_table.relnamespace
WHERE
source_ns.nspname = 'my_schema'
AND source_table.relname = 'my_table'
AND pg_attribute.attnum > 0
ORDER BY 1,2
LIMIT 10;

+------------------+----------------+---------------+--------------+-------------+
| dependent_schema | dependent_view | source_schema | source_table | column_name |
+------------------+----------------+---------------+--------------+-------------+
```

使用 IF EXISTS 删除表

以下示例删除 FEEDBACK 表（如果存在）；如果不存在该表，则不执行任何操作并返回一条消息：

```
drop table if exists feedback;
```

DROP USER

从数据库中删除用户。可以使用一条 DROP USER 命令删除多个用户。您必须是数据库超级用户或具有 DROP USER 权限才能运行此命令。

语法

```
DROP USER [ IF EXISTS ] name [, ... ]
```

参数

IF EXISTS

一个子句，指示如果指定的用户不存在，则命令不应进行任何更改，并返回一条指示用户不存在的消息，而不是终止并显示错误。

此子句在编写脚本时很有用，可使在 DROP USER 针对不存在的用户运行时脚本不会失败。

name

要删除的用户的名称。您可以使用逗号来分隔各个用户名，从而指定多个用户。

使用说明

您无法删除名为 rdsdb 的用户或数据库的管理员用户（通常名为 awsuser 或 admin）。

如果用户拥有任何数据库对象（例如架构、数据库、表或视图），或者用户具有对数据库、表、列或组的任何权限，则您不能删除该用户。如果您试图删除此类用户，则会收到以下错误之一。

```
ERROR: user "username" can't be dropped because the user owns some object [SQL
 State=55006]

ERROR: user "username" can't be dropped because the user has a privilege on some object
 [SQL State=55006]
```

有关如何查找数据库用户拥有的对象的详细说明，请参阅《知识中心》中的如何解决 Amazon Redshift 中的“无法删除用户”错误？。

> **Note**
>
> Amazon Redshift 在删除用户前只检查当前数据库。如果用户拥有数据库对象或者对另一数据库中的对象具有任何权限，DROP USER 不会返回错误。如果您删除的用户拥有另一个数据库中的对象，这些对象的所有者将更改为“未知”。

如果用户拥有对象，请先删除对象或者将其所有权更改为其他用户，然后再删除原始用户。如果用户具有对象的权限，请先撤消权限，然后再删除用户。以下示例说明先删除对象、更改所有权、撤消权限然后再删除用户的过程。

```
drop database dwdatabase;
alter schema dw owner to dwadmin;
revoke all on table dwtable from dwuser;
drop user dwuser;
```

示例

以下示例删除名为 paulo 的用户：

```
drop user paulo;
```

以下示例删除两个用户，即 paulo 和 martha：

```
drop user paulo, martha;
```

在以下示例中，如果用户 paulo 存在，则删除该用户；如果不存在，则不执行任何操作并返回一条消息：

```
drop user if exists paulo;
```

DROP VIEW

从数据库中删除视图。可以使用一条 DROP VIEW 命令删除多个视图。此命令无法撤消。

所需的权限

以下是 DROP VIEW 所需的权限：

- Superuser
- 具有 DROP VIEW 权限的用户
- 视图拥有者

语法

```
DROP VIEW [ IF EXISTS ] name [, ... ] [ CASCADE | RESTRICT ]
```

参数

IF EXISTS

一个子句，指示如果指定的视图不存在，则命令不应进行任何更改，并返回一条指示视图不存在的消息，而不是以错误终止。

此子句在编写脚本时很有用，可使在 DROP VIEW 针对不存在的视图运行时脚本不会失败。

名称

要删除的视图的名称。

CASCADE

一个子句，用于指示自动删除依赖该视图的对象，例如其他视图。

要使创建的视图不依赖于其他数据库对象（例如视图和表），请在定义视图时包括 WITH NO SCHEMA BINDING 子句。有关更多信息，请参阅 CREATE VIEW。

请注意，如果您包含 CASCADE 参数并且删除的数据库对象数不少于十个，则数据库客户端可能不会在摘要结果中列出所有已删除的对象。这通常是因为 SQL 客户端工具对返回的结果数有默认限制。

RESTRICT

一个子句，指示如果任何对象依赖该视图，则不删除该视图。此操作是默认操作。

示例

以下示例删除名为 event 的视图：

```
drop view event;
```

要删除具有依赖项的视图，请使用 CASCADE 选项。例如，假如我们从名为 EVENT 的表开始。接下来，我们使用 CREATE VIEW 命令创建 EVENT 表的 eventview 视图，如以下示例所示：

```
create view eventview as
select dateid, eventname, catid
from event where catid = 1;
```

现在，我们创建另一个名为 myeventview 的视图，该视图基于第一个视图 eventview：

```
create view myeventview as
select eventname, catid
from eventview where eventname <> ' ';
```

此时创建了两个视图：eventview 和 myeventview。

myeventview 视图是以 eventview 为父视图的子视图。

要删除 eventview 视图，要使用的命令如下：

```
drop view eventview;
```

请注意，如果您在这种情况下运行此命令，会收到以下错误：

```
drop view eventview;
ERROR: can't drop view eventview because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too.
```

要对此进行补救，请执行以下命令（按照错误消息中的建议）：

```
drop view eventview cascade;
```

现在，已成功删除 eventview 和 myeventview。

以下示例删除 eventview 视图（如果存在）；如果不存在该视图，则不执行任何操作并返回一条消息：

```
drop view if exists eventview;
```

END

提交当前事务。执行与 COMMIT 命令完全相同的功能。

有关更详细的文档，请参阅 COMMIT。

语法

```
END [ WORK | TRANSACTION ]
```

参数

WORK

可选关键字。

TRANSACTION

可选关键字；WORK 和 TRANSACTION 同义。

示例

下面的示例都结束事务块并提交事务：

```
end;
```

```
end work;
```

```
end transaction;
```

在执行以下任一命令后，Amazon Redshift 将结束事务数据块并执行提交。

EXECUTE

执行先前预编译的语句。

语法

```
EXECUTE plan_name [ (parameter [, ...]) ]
```

参数

plan_name

要运行的预编译语句的名称。

parameter

预编译语句的某个参数的实际值。它必须是一个生成某个类型的值的表达式，而且该类型必须与在创建该预编译语句的 PREPARE 命令中为此参数位置指定的数据类型兼容。

使用说明

EXECUTE 用于执行先前预编译的语句。由于预编译语句只在会话持续时间内存在，因此预编译语句必须已由先前在当前会话中执行的 PREPARE 语句创建。

如果先前的 PREPARE 语句指定了一些参数，则必须将兼容的参数集传递到 EXECUTE 语句，否则 Amazon Redshift 返回错误。与函数不同的是，预编译语句不会根据指定参数的类型或数量来重载；预编译语句的名称在一个数据库会话内必须是唯一的。

为预编译语句发出 EXECUTE 命令时，Amazon Redshift 可能会选择修改查询执行计划（以便根据指定的参数值来提高性能），然后再执行预编译语句。此外，对于预编译语句的每次新执行，Amazon Redshift 可能会根据使用 EXECUTE 语句指定的其他参数值，再次修改查询执行计划。要查看 Amazon Redshift 为任何给定 EXECUTE 语句选择的查询执行计划，请使用 EXPLAIN 命令。

有关创建和使用预编译语句的示例和更多信息，请参阅 PREPARE。

另请参阅

DEALLOCATE, PREPARE

EXPLAIN

显示查询语句的执行计划，而不运行查询。有关查询分析工作流程的信息，请参阅查询分析工作流程。

语法

```
EXPLAIN [ VERBOSE ] query
```

参数

VERBOSE

显示完整的查询计划，而不只是摘要。

query

要解释的查询语句。查询可以是 SELECT、INSERT、CREATE TABLE AS、UPDATE 或 DELETE 语句。

使用说明

在创建临时表时需要花费一定的时间，因此 EXPLAIN 性能有时会受到影响。例如，使用公共子表达式优化的查询需要创建和分析临时表，以返回 EXPLAIN 输出。查询计划依赖于临时表的 schema 和统计数据。因此，此类查询的 EXPLAIN 命令的运行时间可能会超过预期。

您只能对以下命令使用 EXPLAIN：

- SELECT
- SELECT INTO
- CREATE TABLE AS
- INSERT
- UPDATE
- DELETE

如果您对其他 SQL 命令（例如数据定义语言 (DDL) 或数据库操作）使用 EXPLAIN 命令，则该命令将失败。

Amazon Redshift 使用 EXPLAIN 输出相对单位成本来选择查询计划。Amazon Redshift 会比较各种资源估算值的大小来确定计划。

查询计划和执行步骤

特定 Amazon Redshift 查询语句的执行计划会将查询的执行和计算细分为一系列单独的步骤和表操作，它们最后为该查询生成一个最终结果集。有关查询计划的信息，请参阅查询处理。

下表提供了步骤的汇总，Amazon Redshift 可以针对用户提交供执行的任何查询，使用这些步骤来制定执行计划。

EXPLAIN 运算符	查询执行步骤	描述
SCAN：		
顺序扫描	scan	Amazon Redshift 关系扫描或表扫描运算符或步骤。从头开始按顺序扫描整个表直至结尾；另外，如果在 WHERE 子句中指定它，则还会评估每一行的查询约束（筛选条件）。它还用于运行 INSERT、UPDATE 和 DELETE 语句。
JOINS：Amazon Redshift 根据要联接的表的物理设计、联接所需的数据位置以及查询本身的特定属性，来使用不同的联接运算符。子查询扫描 – 使用子查询扫描和附加来运行 UNION 查询。		
嵌套循环	nloop	优化程度最差的联接；主要用于交叉联接（笛卡尔积；没有联接条件）和一些不等式联接。
哈希联接	hjoin	还用于内部联接以及左和右外部联接，速度通常比嵌套循环联接要快。哈希联接读取外部表，对联接列进行哈希处理，然后在内部哈希表查找匹配项。步骤会溢到磁盘。（hjoin 的内部输入是可基于磁盘的哈希步骤。）
合并联接	mjoin	还用于内部联接和外部联接（用于在联接列上进行分配和排序的联接表）。在不考虑其他成本的情况下，通常这是最快的 Amazon Redshift 联接算法。

EXPLAIN 运算符	查询执行步骤	描述
AGGREGATION：用于涉及聚合函数以及 GROUP BY 操作的查询的运算符和步骤。		
聚合	aggr	标量聚合函数的运算符/步骤。
HashAggregate	aggr	分组聚合函数的运算符/步骤。可利用哈希表溢到磁盘的优势来从磁盘运行。
GroupAggregate	aggr	有时，force_hash_grouping 设置的 Amazon Redshift 配置设置为 off，此时为分组聚合查询选择该运算符。
SORT：在查询必须对结果集进行排序或合并时使用的运算符和步骤。		
排序	sort	排序操作执行 ORDER BY 子句指定的排序，此外还执行其他一些操作，例如 UNION 和联接。可从磁盘运行。
合并	merge	根据从并行执行的操作得到的临时排序结果，来生成某个查询的最终排序结果。
EXCEPT、INTERSECT 和 UNION 操作：		
SetOp Except [Distinct]	hjoin	用于 EXCEPT 查询。可利用输入哈希可基于磁盘的优势来从磁盘运行。
Hash Intersect [Distinct]	hjoin	用于 INTERSECT 查询。可利用输入哈希可基于磁盘的优势来从磁盘运行。
Append [All |Distinct]	save	附加操作与子查询扫描结合使用，来实施 UNION 和 UNION ALL 查询。可利用“save”操作的优势来从磁盘运行。
杂项/其他：		
哈希	hash	用于内部联接以及左和右外部联接（为哈希联接提供输入）。Hash 运算符为联接运算的内部表创建哈希表。（内部表是用来检查匹配项的表；在联接两个表时，通常是其中较小的那个表。）

EXPLAIN 运算符	查询执行步骤	描述
限制	limit	评估 LIMIT 子句。
具体化	save	具体化嵌套循环联接和某些合并联接的输入中的行。可从磁盘运行。
--	parse	用于在加载期间解析文字输入数据。
--	project	用于重新整理列和计算表达式，即项目数据。
结果	--	运行不涉及任何表访问的标量函数。
--	return	将行返回到领导节点或客户端。
子计划	--	用于特定的子查询。
唯一	unique	消除 SELECT DISTINCT 和 UNION 查询中的重复项。
窗口	window	计算汇总并对窗口函数进行排名。可从磁盘运行。
网络操作：		
网络（广播）	bcast	广播也是 Join Explain 运算符和步骤的一个属性。
网络（分布）	dist	将行分布到计算节点，以便由数据仓库集群执行并行处理。
网络（发送到领导节点）	return	将结果发送回领导节点，以待进一步处理。
DML 操作（用于修改数据的运算符）：		
插入（使用结果）	insert	插入数据。
删除（扫描 + 筛选）	delete	删除数据。可从磁盘运行。
更新（扫描 + 筛选）	delete、insert	以删除和插入的方式实施。

将 EXPLAIN 用于 RLS

如果查询包含受行级别安全性 (RLS) 策略约束的表，则 EXPLAIN 将显示一个特殊的 RLS SecureScan 节点。Amazon Redshift 还会将相同的节点类型记录到 STL_EXPLAIN 系统表中。EXPLAIN 不会显示应用于 dim_tbl 的 RLS 谓词。RLS SecureScan 节点类型用作执行计划包含当前用户不可见的其他操作的指示器。

以下示例说明了 RLS SecureScan 节点。

```
EXPLAIN
SELECT D.cint
FROM fact_tbl F INNER JOIN dim_tbl D ON F.k_dim = D.k
WHERE F.k_dim / 10 > 0;
                               QUERY PLAN
------------------------------------------------------------------------
 XN Hash Join DS_DIST_ALL_NONE  (cost=0.08..0.25 rows=1 width=4)
   Hash Cond: ("outer".k_dim = "inner"."k")
   ->  *XN* *RLS SecureScan f  (cost=0.00..0.14 rows=2 width=4)*
         Filter: ((k_dim / 10) > 0)
   ->  XN Hash  (cost=0.07..0.07 rows=2 width=8)
         ->  XN Seq Scan on dim_tbl d  (cost=0.00..0.07 rows=2 width=8)
               Filter: (("k" / 10) > 0)
```

为了能够对受 RLS 约束的查询计划进行全面调查，Amazon Redshift 提供了 EXPLAIN RLS 系统权限。被授予此权限的用户可以检查还包含 RLS 谓词的完整查询计划。

以下示例说明了 RLS SecureScan 节点下方的额外顺序扫描，该顺序扫描还包括 RLS 策略谓词 (k_dim **>** 1)。

```
EXPLAIN SELECT D.cint
FROM fact_tbl F INNER JOIN dim_tbl D ON F.k_dim = D.k
WHERE F.k_dim / 10 > 0;
                                   QUERY PLAN
--------------------------------------------------------------------------------
 XN Hash Join DS_DIST_ALL_NONE  (cost=0.08..0.25 rows=1 width=4)
   Hash Cond: ("outer".k_dim = "inner"."k")
   *->  XN RLS SecureScan f  (cost=0.00..0.14 rows=2 width=4)
         Filter: ((k_dim / 10) > 0)*
         ->  *XN* *Seq Scan on fact_tbl rls_table  (cost=0.00..0.06 rows=5 width=8)
               Filter: (k_dim > 1)*
   ->  XN Hash  (cost=0.07..0.07 rows=2 width=8)
         ->  XN Seq Scan on dim_tbl d  (cost=0.00..0.07 rows=2 width=8)
```

```
                Filter: (("k" / 10) > 0)
```

在向用户授予 EXPLAIN RLS 权限的同时，Amazon Redshift 会在 STL_EXPLAIN 系统表中记录包括 RLS 谓词在内的完整查询计划。在未授予此权限时运行的查询将在没有 RLS 内部构件的情况下被记录。授予或删除 EXPLAIN RLS 权限不会改变 Amazon Redshift 为之前的查询记录到 STL_EXPLAIN 的内容。

受 AWS Lake Formation-RLS 保护的 Redshift 关系

以下示例说明了 LF SecureScan 节点，您可以使用它来查看 Lake Formation-RLS 关系。

```
EXPLAIN
SELECT *
FROM lf_db.public.t_share
WHERE a > 1;
QUERY PLAN
------------------------------------------------------------------
XN LF SecureScan t_share  (cost=0.00..0.02 rows=2 width=11)
(2 rows)
```

示例

> **Note**
>
> 对于这些示例，输出样本可能有所不同，具体取决于 Amazon Redshift 配置。

以下示例为从 EVENT 和 VENUE 表中选择 EVENTID、EVENTNAME、VENUEID 和 VENUENAME 的查询返回查询计划：

```
explain
select eventid, eventname, event.venueid, venuename
from event, venue
where event.venueid = venue.venueid;
```

```
                                QUERY PLAN
------------------------------------------------------------------------------
XN Hash Join DS_DIST_OUTER  (cost=2.52..58653620.93 rows=8712 width=43)
Hash Cond: ("outer".venueid = "inner".venueid)
```

```
->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=23)
->  XN Hash  (cost=2.02..2.02 rows=202 width=22)
->  XN Seq Scan on venue  (cost=0.00..2.02 rows=202 width=22)
(5 rows)
```

以下示例使用详细输出为同一查询返回查询计划：

```
explain verbose
select eventid, eventname, event.venueid, venuename
from event, venue
where event.venueid = venue.venueid;
```

```
                                QUERY PLAN
------------------------------------------------------------------------------
{HASHJOIN
:startup_cost 2.52
:total_cost 58653620.93
:plan_rows 8712
:plan_width 43
:best_pathkeys <>
:dist_info DS_DIST_OUTER
:dist_info.dist_keys (
TARGETENTRY
{
VAR
:varno 2
:varattno 1
...

XN Hash Join DS_DIST_OUTER  (cost=2.52..58653620.93 rows=8712 width=43)
Hash Cond: ("outer".venueid = "inner".venueid)
->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=23)
->  XN Hash  (cost=2.02..2.02 rows=202 width=22)
->  XN Seq Scan on venue  (cost=0.00..2.02 rows=202 width=22)
(519 rows)
```

以下示例返回 CREATE TABLE AS (CTAS) 语句的查询计划：

```
explain create table venue_nonulls as
select * from venue
where venueseats is not null;
```

```
QUERY PLAN
-------------------------------------------------------------
XN Seq Scan on venue  (cost=0.00..2.02 rows=187 width=45)
Filter: (venueseats IS NOT NULL)
(2 rows)
```

FETCH

使用游标检索行。有关声明游标的信息，请参阅DECLARE。

FETCH 根据游标中的当前位置检索行。在创建游标时，系统会将它放在第一行的前面。在执行 FETCH 后，系统会将游标放在检索的最后一行上。如果 FETCH 在运行时超出了可用行的结尾，例如跟在 FETCH ALL 后面，则游标会保留在最后一行的后面。

FORWARD 0 会提取当前行而不移动游标；即提取最近提取过的行。如果游标放在第一行的前面或最后一行的后面，则不返回任何行。

在提取游标的第一行时，会在领导节点上、内存中或磁盘上具体化整个结果集（如果需要）。由于将游标用于大型结果集可能会降低性能，因此建议使用备用方法（如果可能）。有关更多信息，请参阅 使用游标时的性能注意事项。

有关更多信息，请参阅DECLARE、CLOSE。

语法

```
FETCH [ NEXT | ALL | {FORWARD [ count | ALL ] } ] FROM cursor
```

参数

NEXT

提取下一行。这是默认模式。

ALL

提取所有剩余的行。（与 FORWARD ALL 相同。）对于单节点集群，不支持 ALL。

FORWARD [count | ALL]

提取后面 count 行，或者提取所有剩余的行。FORWARD 0 提取当前行。对于单节点集群，count 的最大值为 1000。对于单节点集群，不支持 FORWARD ALL。

cursor

新游标的名称。

FETCH 示例

以下示例声明一个名为 LOLLAPALOOZA 的游标，以便为 Lollapalooza 事件选择销售信息，然后使用该游标从结果集中提取行：

```
-- Begin a transaction

begin;

-- Declare a cursor

declare lollapalooza cursor for
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Lollapalooza';

-- Fetch the first 5 rows in the cursor lollapalooza:

fetch forward 5 from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-05-01 19:00:00 |   92.00000000 |       3
 Lollapalooza | 2008-11-15 15:00:00 |  222.00000000 |       2
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       3
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       4
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       1
(5 rows)

-- Fetch the next row:

fetch next from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-10-06 14:00:00 |  114.00000000 |       2

-- Close the cursor and end the transaction:
```

```
close lollapalooza;
commit;
```

GRANT

为用户或角色定义访问权限。

权限包括各种访问选项，例如读取表和视图中的数据、写入数据、创建表和删除表的功能。使用此命令可授予对表、数据库、架构、函数、过程、语言或列的特定权限。要撤销对数据库对象的权限，请使用 REVOKE 命令。

权限还包括以下数据共享生产者访问选项：

- 向使用者命名空间和账户授予数据共享访问权限。
- 通过在数据共享中添加或删除对象来授予更改数据共享的权限。
- 通过在数据共享中添加或删除使用者命名空间来授予共享数据共享的权限。

数据共享使用者访问选项如下：

- 向用户授予对通过数据共享创建的数据库或指向此类数据库的外部架构的完全访问权限。
- 向用户授予对通过数据共享创建的数据库的对象级权限，就像对本地数据库对象一样。要授予此级别的权限，在从数据共享创建数据库时必须使用 WITH PERMISSIONS 子句。有关更多信息，请参阅 CREATE DATABASE。

有关数据共享权限的更多信息，请参阅共享数据共享。

您还可以授予角色来管理数据库权限，以及控制用户可以对您的数据执行的操作。通过定义角色并向用户分配角色，您可以限制这些用户能够执行的操作，例如限制用户只能使用 CREATE TABLE 和 INSERT 命令。有关 CREATE ROLE 命令的更多信息，请参阅 the section called “CREATE ROLE”。Amazon Redshift 有一些系统定义的角色，您也可以使用这些角色向用户授予特定权限。有关更多信息，请参阅 the section called “Amazon Redshift 系统定义的角色”。

您只能将对于外部架构的 GRANT 或 REVOKE USAGE 权限授予使用 ON SCHEMA 语法的数据库用户和用户组。将 ON EXTERNAL SCHEMA 与 AWS Lake Formation 搭配使用时，您只能向 AWS Identity and Access Management (IAM) 角色授予 GRANT 和 REVOKE 权限。有关权限的列表，请参阅“语法”。

对于存储过程，唯一可以授予的权限是 EXECUTE。

您不能在以下事务数据块内的（外部资源）上运行 GRANT：(BEGIN ... END)。有关事务的更多信息，请参阅 可序列化的隔离。

要查看已向用户授予了数据库的哪些权限，请使用 HAS_DATABASE_PRIVILEGE。要查看已向用户授予了针对架构的哪些权限，请使用 HAS_SCHEMA_PRIVILEGE。要查看已向用户授予了表的哪些权限，请使用 HAS_TABLE_PRIVILEGE。

语法

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES | ALTER | TRUNCATE }
 [,...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...] | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { { CREATE | TEMPORARY | TEMP | ALTER } [,...] | ALL [ PRIVILEGES ] }
    ON DATABASE db_name [, ...]
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { { CREATE | USAGE | ALTER } [,...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { FUNCTION function_name ( [ [ argname ] argtype [, ...] ] ) [, ...] | ALL
 FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { PROCEDURE procedure_name ( [ [ argname ] argtype [, ...] ] ) [, ...] | ALL
 PROCEDURES IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT USAGE
    ON LANGUAGE language_name [, ...]
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

授予表的列级别权限

以下是 Amazon Redshift 表和视图上的列级别权限的语法。

```
GRANT { { SELECT | UPDATE } ( column_name [, ...] ) [, ...] | ALL [ PRIVILEGES ]
 ( column_name [,...] ) }
     ON { [ TABLE ] table_name [, ...] }

     TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

授予 ASSUMEROLE 权限

以下是向具有指定角色的用户和组授予 ASSUMEROLE 权限的语法。要开始使用 ASSUMEROLE 权限，请参阅有关授予 ASSUMEROLE 权限的使用说明。

```
GRANT ASSUMEROLE
       ON { 'iam_role' [, ...] | default | ALL }
       TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
       FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]
```

授予将 Redshift Spectrum 与 Lake Formation 集成的权限

以下是 Redshift Spectrum 与 Lake Formation 集成的语法。

```
GRANT { SELECT | ALL [ PRIVILEGES ] } ( column_list )
    ON EXTERNAL TABLE schema_name.table_name
    TO { IAM_ROLE iam_role } [, ...] [ WITH GRANT OPTION ]

GRANT { { SELECT | ALTER | DROP | DELETE | INSERT }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL TABLE schema_name.table_name [, ...]
    TO { { IAM_ROLE iam_role } [, ...] | PUBLIC } [ WITH GRANT OPTION ]

GRANT { { CREATE | ALTER | DROP }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL SCHEMA schema_name [, ...]
    TO { IAM_ROLE iam_role } [, ...] [ WITH GRANT OPTION ]
```

授予数据共享权限

生产者端数据共享权限

以下是使用 GRANT 向用户或角色授予 ALTER 或 SHARE 权限的语法。用户可以使用 ALTER 权限更改数据共享，也可以向具有 SHARE 权限的使用者授予使用权限。您只能向用户和角色授予对数据共享的 ALTER 和 SHARE 权限。

```
GRANT { ALTER | SHARE } ON DATASHARE datashare_name
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

以下是使用 GRANT 授予 Amazon Redshift 上的数据共享使用权限的语法。您可以使用 USAGE 权限向使用者授予对数据共享的访问权限。您不能将此权限授予用户或用户组。此权限也不支持 GRANT 语句的 WITH GRANT OPTION。只有具有先前针对数据共享授予了他们 SHARE 权限的用户或用户组才能运行此类型的 GRANT 语句。

```
GRANT USAGE
    ON DATASHARE datashare_name
    TO NAMESPACE 'namespaceGUID' | ACCOUNT 'accountnumber' [ VIA DATA CATALOG ]
```

以下是如何向 Lake Formation 账户授予数据共享使用权的一个示例。

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '123456789012' VIA DATA CATALOG;
```

使用者端数据共享权限

以下是根据数据共享创建的特定数据库或 schema 的 GRANT 数据共享使用权限的语法。

使用者访问通过数据共享创建的数据库所需的其他权限会有所不同，具体取决于用于从数据共享创建数据库的 CREATE DATABASE 命令是否使用 WITH PERMISSIONS 子句。有关 CREATE DATABASE 命令和 WITH PERMISSIONS 子句的更多信息，请参阅CREATE DATABASE。

未使用 WITH PERMISSIONS 子句创建的数据库

在不使用 WITH PERMISSIONS 子句的情况下，对通过数据共享创建的数据库授予 USAGE 权限时，无需对共享数据库中的对象单独授予权限。在不使用 WITH PERMISSIONS 子句的情况下，获准使用从数据共享创建的数据库的实体可以自动访问数据库中的所有对象。

使用 WITH PERMISSIONS 子句创建的数据库

如果共享数据库是使用 WITH PERMISSIONS 子句从数据共享中创建的，则在授予数据库 USAGE 权限时，使用者端实体仍必须获得共享数据库中数据库对象的相关权限才能访问这些对象，就像对本地数据库对象授予权限一样。要向通过数据共享创建的数据库中的对象授予权限，请使用由三部分组成的语

法 database_name.schema_name.object_name。要向外部架构中指向共享数据库中共享架构的对象授予权限，请使用由两部分组成的语法 schema_name.object_name。

```
GRANT USAGE ON { DATABASE shared_database_name [, ...] | SCHEMA shared_schema}
    TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

授予限定范围权限

限定范围权限允许您向用户或角色授予对数据库或架构中某一类型的所有对象的访问权限。具有限定范围权限的用户和角色对数据库或架构中的所有当前和未来对象具有指定权限。

以下是向用户和角色授予限定范围权限的语法。有关限定范围权限的更多信息，请参阅限定范围权限。

```
GRANT { CREATE | USAGE | ALTER } [,...] | ALL [ PRIVILEGES ] }
FOR SCHEMAS IN
DATABASE db_name
TO { username [ WITH GRANT OPTION ] | ROLE role_name } [, ...]

GRANT
{ { SELECT | INSERT | UPDATE | DELETE | DROP | ALTER | TRUNCATE | REFERENCES }
 [, ...] } | ALL [PRIVILEGES] } }
FOR TABLES IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO { username [ WITH GRANT OPTION ] | ROLE role_name} [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
FOR FUNCTIONS IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO { username [ WITH GRANT OPTION ] | ROLE role_name | } [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
FOR PROCEDURES IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO { username [ WITH GRANT OPTION ] | ROLE role_name | } [, ...]

GRANT USAGE
FOR LANGUAGES IN
{DATABASE db_name}
TO { username [ WITH GRANT OPTION ] | ROLE role_name } [, ...]
```

请注意，范围限定的权限不区分函数的权限和过程的权限。例如，以下语句授予 bob 对架构 Sales_schema 中函数和过程的 EXECUTE 权限。

```
GRANT EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema TO bob;
```

授予机器学习权限

以下是有关 Amazon Redshift 上机器学习模型权限的语法。

```
GRANT CREATE MODEL
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON MODEL model_name [, ...]

    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

授予角色权限

以下是授予角色对 Amazon Redshift 权限的语法。

```
GRANT { ROLE role_name } [, ...] TO { { user_name [ WITH ADMIN OPTION ] } |
 ROLE role_name }[, ...]
```

以下是向角色授予对 Amazon Redshift 的系统权限的语法。

```
GRANT
  {
    { CREATE USER | DROP USER | ALTER USER |
    CREATE SCHEMA | DROP SCHEMA |
    ALTER DEFAULT PRIVILEGES |
    ACCESS CATALOG |
    CREATE TABLE | DROP TABLE | ALTER TABLE |
    CREATE OR REPLACE FUNCTION | CREATE OR REPLACE EXTERNAL FUNCTION |
    DROP FUNCTION |
    CREATE OR REPLACE PROCEDURE | DROP PROCEDURE |
    CREATE OR REPLACE VIEW | DROP VIEW |
    CREATE MODEL | DROP MODEL |
    CREATE DATASHARE | ALTER DATASHARE | DROP DATASHARE |
    CREATE LIBRARY | DROP LIBRARY |
    CREATE ROLE | DROP ROLE |
    TRUNCATE TABLE
    VACUUM | ANALYZE | CANCEL }[, ...]
```

```
  }
  | { ALL [ PRIVILEGES ] }
TO { ROLE role_name } [, ...]
```

授予对行级别安全性策略筛选器的解释权限

以下语法用于授予解释 EXPLAIN 计划中查询的行级别安全性策略筛选器的权限。您可以使用 REVOKE 语句撤消该权限。

```
GRANT EXPLAIN RLS TO ROLE rolename
```

以下语法用于授予为查询绕开行级别安全性策略的权限。

```
GRANT IGNORE RLS TO ROLE rolename
```

向策略对象授予 RLS 查找表的权限

以下语法用于授予对指定行级别安全性策略的权限。

```
GRANT SELECT ON [ TABLE ] table_name [, ...]
TO RLS POLICY policy_name [, ...]
```

参数

SELECT

授予使用 SELECT 语句从表或视图中选择数据的权限。对于 UPDATE 或 DELETE 操作，也需要 SELECT 权限来引用现有的列值。

INSERT

授予使用 INSERT 语句或 COPY 语句将数据加载到表中的权限。

UPDATE

授予使用 UPDATE 语句更新表列的权限。UPDATE 操作也需要 SELECT 权限，因为这些操作必须引用表列才能确定要更新哪些行或者计算列的新值。

DELETE

授予从表中删除数据行的权限。DELETE 操作也需要 SELECT 权限，因为这些操作必须引用表列才能确定要删除哪些行。

DROP

授予删除表的权限。此权限仅在 Amazon Redshift 以及为 Lake Formation 启用的 AWS Glue Data Catalog 中适用。

REFERENCES

授予创建外键约束的权限。您必须授予对被引用表和引用表的此权限；否则，用户将无法创建约束。

ALTER

根据数据库对象，向用户或用户组授予以下权限：

- 对于表，ALTER 授予更改表或视图的权限。有关更多信息，请参阅 ALTER TABLE。
- 对于数据库，ALTER 授予更改数据库的权限。有关更多信息，请参阅 ALTER DATABASE。
- 对于模式，ALTER 授予更改模式的权限。有关更多信息，请参阅 ALTER SCHEMA。
- 对于外部表，ALTER 授予对为 Lake Formation 启用的 AWS Glue Data Catalog 中的表进行更改的权限。此权限仅在使用 Lake Formation 时适用。

TRUNCATE

授予截断表的权限。如果没有此权限，则只有表的拥有者或超级用户才可以截断表。有关 TRUNCATE 命令的更多信息，请参阅 the section called “TRUNCATE”。

ALL [PRIVILEGES]

一次性向指定的用户或用户组授予所有可用权限。PRIVILEGES 关键字是可选的。

GRANT ALL ON SCHEMA 不会授予对外部架构的 CREATE 权限。

您可以针对为 Lake Formation 启用的 AWS Glue Data Catalog 中的表授予 ALL 权限。在这种情况下，各个权限（如 SELECT、ALTER 等）将记录在 Data Catalog 中。

ASSUMEROLE

向具有指定角色的用户、角色或组授予运行 COPY、UNLOAD、EXTERNAL FUNCTION 和 CREATE MODEL 命令的权限。用户、角色或组在运行指定的命令时代入该角色。要开始使用 ASSUMEROLE 权限，请参阅有关授予 ASSUMEROLE 权限的使用说明。

ON [TABLE] table_name

授予对表或视图的指定权限。TABLE 关键字是可选的。您可以在一个语句中列出多个表和视图。

ON ALL TABLES IN SCHEMA schema_name

授予对被引用架构中的所有表和视图的指定权限。

(column_name [,...]) ON TABLE table_name

向用户、组或 PUBLIC 授予对 Amazon Redshift 表或视图的指定列的指定权限。

(column_list) ON EXTERNAL TABLE schema_name.table_name

向 IAM 角色授予对引用架构中 Lake Formation 表的指定列的指定权限。

ON EXTERNAL TABLE schema_name.table_name

向 IAM 角色授予对引用架构中指定 Lake Formation 表的指定权限。

ON EXTERNAL SCHEMA schema_name

向 IAM 角色授予对引用架构的指定权限。

ON iam_role

向 IAM 角色授予指定权限。

TO username

指示接收权限的用户。

TO IAM_ROLE iam_role

指示接收权限的 IAM 角色。

WITH GRANT OPTION

指示接收权限的用户随之可以将相同的权限授予其他用户。您无法将 WITH GRANT OPTION 授予组或 PUBLIC。

ROLE role_name

将权限授予角色。

GROUP group_name

将权限授予用户组。可以是逗号分隔的列表，用于指定多个用户组。

PUBLIC

向所有用户授予指定的权限，包括以后创建的用户。PUBLIC 表示一个始终包含所有用户的组。单个用户的权限包含向 PUBLIC 授予的权限、向用户所属的所有组授予的权限以及向用户单独授予的任何权限。

将 PUBLIC 授予 Lake Formation EXTERNAL TABLE 会将权限授予 Lake Formation 所有人组。

CREATE

根据数据库对象，向用户或用户组授予以下权限：

- 对于数据库，CREATE 允许用户在数据库中创建 schemas。
- 对于 schema，CREATE 允许用户在 schema 中创建对象。要重命名对象，用户必须具有 CREATE 权限并拥有要重命名的对象。
- Amazon Redshift Spectrum 外部 schema 不支持 CREATE ON SCHEMA。要授予在外部 schema 中使用外部表的权限，请向需要访问权限的用户授予 USAGE ON SCHEMA。仅允许外部 schema 的所有者或超级用户在外部 schema 中创建外部表。要移交外部 schema 的所有权，请使用 ALTER SCHEMA 更改所有者。

TEMPORARY | TEMP

授予在指定的数据库中创建临时表的权限。要运行 Amazon Redshift Spectrum 查询，数据库用户必须有权在数据库中创建临时表。

> **Note**
>
> 默认情况下，向用户授予权限以通过其在 PUBLIC 组中自动获得的成员资格来创建临时表。要删除任何用户创建临时表的权限，请撤销 PUBLIC 组的 TEMP 权限。然后，明确授予特定用户或用户组创建临时表的权限。

ON DATABASE db_name

授予对数据库的指定权限。

USAGE

授予对特定架构的 USAGE 权限，这将使用户能够访问该架构中的对象。必须单独为本地 Amazon Redshift 架构授予对这些对象执行特定操作的权限（例如，对表的 SELECT 或 UPDATE 权限）。默认情况下，所有用户都对 PUBLIC 架构具有 CREATE 和 USAGE 权限。

使用 ON SCHEMA 语法向外部 Schema 授予 USAGE 权限时，无需单独授予对外部 Schema 中对象的操作。相应的目录权限控制对外部 Schema 对象的细粒度权限。

ON SCHEMA schema_name

授予对数据库的指定权限。

Amazon Redshift Spectrum 外部架构不支持 GRANT ALL ON SCHEMA 中的 GRANT CREATE ON SCHEMA 和 CREATE 权限。要授予在外部 schema 中使用外部表的权限，请向需要访问权限

的用户授予 USAGE ON SCHEMA。仅允许外部 schema 的所有者或超级用户在外部 schema 中创建外部表。要移交外部 schema 的所有权，请使用 ALTER SCHEMA 更改所有者。

EXECUTE ON ALL FUNCTIONS IN SCHEMA schema_name

授予对被引用架构中的所有函数的指定权限。

对于在 pg_catalog 命名空间中定义的 pg_proc 内置条目，Amazon Redshift 不支持 GRANT 或 REVOKE 语句。

EXECUTE ON PROCEDURE procedure_name

授予对特定存储过程的 EXECUTE 权限。由于存储过程名称可重载，因此您必须包含过程的参数列表。有关更多信息，请参阅 命名存储过程。

EXECUTE ON ALL PROCEDURES IN SCHEMA schema_name

授予对被引用架构中所有存储过程的指定权限。

USAGE ON LANGUAGE language_name

授予对某种语言的 USAGE 权限。

需要 USAGE ON LANGUAGE 权限才能运行 CREATE FUNCTION 命令来创建用户定义函数 (UDF)。有关更多信息，请参阅 UDF 安全性和权限。

需要 USAGE ON LANGUAGE 权限才能通过运行 CREATE PROCEDURE 命令来创建存储过程。有关更多信息，请参阅 存储过程的安全性和权限 。

对于 Python UDF，使用 `plpythonu`。对于 SQL UDF，使用 `sql`。对于存储过程，请使用 `plpgsql`。

FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]

指定被授予权限的 SQL 命令。您可以指定 ALL 以授予对 COPY、UNLOAD、EXTERNAL FUNCTION 和 CREATE MODEL 语句的权限。此子句仅适用于授予 ASSUMEROLE 权限。

ALTER

向用户授予 ALTER 权限，以便将对象添加到数据共享中或从中删除，或设置属性 PUBLICACCESSIBLE。有关更多信息，请参阅 ALTER DATASHARE。

SHARE

向用户和用户组授予将数据使用者添加到数据共享的权限。要使特定使用者（账户或命名空间）能够从其集群访问数据共享，需要此权限。使用者可以相同或不同的 AWS 账户，集群命名空间与全局唯一标识符 (GUID) 指定的集群命名空间相同或不同。

ON DATASHARE datashare_name

授予对被引用数据共享的指定权限。有关使用者访问控制粒度的信息，请参阅 在 Amazon Redshift 中共享不同级别的数据。

USAGE

将 USAGE 授予同一账户中的使用者账户或命名空间时，该账户中的特定使用者账户或命名空间可以以只读方式访问数据共享和数据共享的对象。

TO NAMESPACE 'clusternamespace GUID'

指示同一账户中的一个命名空间，使用者可以在其中接收对数据共享的指定权限。命名空间使用 128 位的字符数字 GUID。

TO ACCOUNT 'accountnumber' [VIA DATA CATALOG]

指示使用者可以在其中接收对数据共享的指定权限的另一个账号。指定‘VIA DATA CATALOG’表示向 Lake Formation 账户授予使用数据共享的权限。省略此参数意味着您向拥有该集群的账户授予使用权限。

ON DATABASE shared_database_name> [, ...]

授予对在指定数据共享中创建的指定数据库的指定使用权限。

ON SCHEMA shared_schema

授予对在指定数据共享中创建的指定架构的指定权限。

FOR { SCHEMAS | TABLES | FUNCTIONS | PROCEDURES | LANGUAGES } IN

指定要向其授予权限的数据库对象。IN 后面的参数定义了所授予权限的范围。

CREATE MODEL

向特定用户或用户组授予 CREATE MODEL 权限。

ON MODEL model_name

授予对特定模型的 EXECUTE 权限。

ACCESS CATALOG

授予查看该角色可访问对象的相关元数据的权限。

{ role } [, ...]

要向其他角色、用户或 PUBLIC 授予的角色。

PUBLIC 表示一个始终包含所有用户的组。单个用户的权限包含向 PUBLIC 授予的权限、向用户所属的所有组授予的权限以及向用户单独授予的任何权限。

TO { { user_name [WITH ADMIN OPTION] } | role }[, ...]

将指定角色授予具有 WITH ADMIN OPTION 权限的指定用户、其他角色或 PUBLIC。

WITH ADMIN OPTION 条件句会向所有被授予者提供所有授予的角色的管理选项。

EXPLAIN RLS TO ROLE rolename

向角色授予权限，使其可以解释 EXPLAIN 计划中某个查询的行级别安全性策略筛选器。

IGNORE RLS TO ROLE rolename

向角色授予绕开某个查询的行级别安全性策略的权限。

使用说明

要了解有关 GRANT 使用说明的更多信息，请参阅the section called “使用说明”。

示例

有关如何使用 GRANT 的示例，请参阅the section called “示例”。

使用说明

要授予关于对象的权限，您必须满足以下条件之一：

- 是对象所有者。
- 是超级用户。
- 拥有该对象和权限的授予权限。

例如，以下命令使用户 HR 能够对 employees 表执行 SELECT 命令并对其他用户授予和撤销相同的权限。

```
grant select on table employees to HR with grant option;
```

HR 无法授予 SELECT 之外的任何操作的权限或 employees 表之外的任何其他表的权限。

再例如，以下命令使用户 HR 能够对 employees 表执行 ALTER 命令并对其他用户授予和撤销相同的权限。

```
grant ALTER on table employees to HR with grant option;
```

HR 无法授予除 ALTER 之外的任何操作的权限或 employees 表之外的任何其他表的权限。

获得视图的权限并不意味着对基础表具有权限。同样，获得 schema 的权限并不意味着对该 schema 中的表具有权限。而是显式授予对基础表的访问权限。

要授予针对 AWS Lake Formation 表的权限，与表的外部架构关联的 IAM 角色必须有权授予对外部表的权限。以下示例创建具有关联 IAM 角色 myGrantor 的外部架构。IAM 角色 myGrantor 有权向其他角色授予权限。GRANT 命令使用与外部架构关联的 IAM 角色 myGrantor 的权限来向 IAM 角色 myGrantee 授予权限。

```
create external schema mySchema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myGrantor'
create external database if not exists;
```

```
grant select
on external table mySchema.mytable
to iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

如果您向 IAM 角色授予 ALL 权限，则会在相关的启用了 Lake Formation 的 Data Catalog 中授予各个权限。例如，以下 GRANT ALL 会导致在 Lake Formation 控制台中显示授予的各个权限（SELECT、ALTER、DROP、DELETE 和 INSERT）。

```
grant all
on external table mySchema.mytable
to iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

超级用户可以访问所有对象，不管设置对象权限的 GRANT 和 REVOKE 命令如何。

列级访问控制的使用说明

以下使用说明适用于 Amazon Redshift 表和视图上的列级权限。这些说明描述了表；除非我们明确指出例外，否则相同的说明适用于视图。

- 对于 Amazon Redshift 表，您只能在列级别授予 SELECT 和 UPDATE 权限。对于 Amazon Redshift 视图，您只能在列级别授予 SELECT 权限。

- ALL 关键字是在表上列级 GRANT 的上下文中使用时组合的 SELECT 和 UPDATE 权限的同义词。
- 如果您没有表中所有列的 SELECT 权限，则执行 SELECT * 操作将仅返回您有权访问的那些列。使用视图时，SELECT * 操作会尝试访问视图中的所有列。如果您并没有访问所有列的权限，则这些查询会失败，并出现权限被拒绝错误。
- 在以下情况下，SELECT * 不会仅扩展到可访问的列：
  - 您无法使用 SELECT * 创建仅包含可访问列的常规视图。
  - 您无法使用 SELECT * 创建仅包含可访问列的实体化视图。
- 如果对表或视图具有 SELECT 或 UPDATE 权限并添加一列，则您对表或视图及其所有列仍具有相同的权限。
- 只有表的拥有者或超级用户才能授予列级权限。
- 列级权限不支持 WITH GRANT OPTION 子句。
- 您不能同时在表级别和列级别保持相同的权限。例如，用户 `data_scientist` 不能既对表 `employee` 具有 SELECT 权限，又对列 `employee.department` 具有 SELECT 权限。在对表和表中的列授予相同权限时，请考虑以下结果：
  - 如果用户对表具有表级权限，则在列级别授予相同权限不起作用。
  - 如果用户对表具有表级权限，则撤消表中一个或多个列的相同权限将返回错误。而应撤消表级别的权限。
  - 如果用户具有列级权限，则在表级别授予相同权限将返回错误。
  - 如果用户具有列级权限，则在表级别撤消相同的权限将撤消对表上所有列的列和表权限。
- 您不能为后期绑定视图授予列级权限。
- 要创建实体化视图，您必须对基表具有表级 SELECT 权限。即使您对特定列具有列级权限，也无法仅在这些列上创建实体化视图。但是，您可以授予对于实体化视图（类似于常规视图）的列的 SELECT 权限。
- 要查找列级别权限的授予，请使用 PG_ATTRIBUTE_INFO 视图。

有关授予 ASSUMEROLE 权限的使用说明

以下使用说明适用于在 Amazon Redshift 中授予 ASSUMEROLE 权限。

您可以使用 ASSUMEROLE 权限，通过 IAM 角色控制数据库用户、角色或组对 COPY、UNLOAD、EXTERNAL FUNCTION 或 CREATE MODEL 等命令的访问权限。向用户、角色或组授予对 IAM 角色的 ASSUMEROLE 权限后，该用户、角色或组可以在运行命令时代入该角色。ASSUMEROLE 权限让您可以根据需要授予对相应命令的访问权限。

只有数据库超级用户才可以授予或撤销用户、角色和组的 ASSUMEROLE 权限。超级用户始终保留 ASSUMEROLE 权限。

要为用户、角色和组启用 ASSUMEROLE 权限，超级用户需要执行以下两项操作：

- 在集群上运行以下语句一次：

```
revoke assumerole on all from public for all;
```

- 向用户、角色和组授予相应命令的 ASSUMEROLE 权限。

在授予 ASSUMEROLE 权限时，您可以在 ON 子句中指定角色链接。您可以使用逗号来分隔角色链中的角色，例如 Role1,Role2,Role3。如果在授予 ASSUMEROLE 权限时指定了角色链接，则在执行由 ASSUMEROLE 权限授予的操作时，必须指定角色链。在执行由 ASSUMEROLE 权限授予的操作时，您无法在角色链中指定各个角色。例如，如果某个用户、角色或组被授予角色链 Role1,Role2,Role3，则不能仅指定 Role1 来执行操作。

如果用户尝试执行 COPY、UNLOAD、EXTERNAL FUNCTION 或 CREATE MODEL 操作，但尚未被授予 ASSUMEROLE 权限，则会显示类似于以下内容的消息。

```
ERROR:  User awsuser does not have ASSUMEROLE permission on IAM role
 "arn:aws:iam::123456789012:role/RoleA" for COPY
```

要列出已通过 ASSUMEROLE 权限授予对 IAM 角色和命令的访问权限的用户，请参阅HAS_ASSUMEROLE_PRIVILEGE。要列出已授予您指定的用户的 IAM 角色和命令权限，请参阅PG_GET_IAM_ROLE_BY_USER。要列出已被授权访问您指定的 IAM 角色的用户、角色和组，请参阅 PG_GET_GRANTEE_BY_IAM_ROLE。

有关授予机器学习权限的使用说明

您不能直接授予或撤销与机器学习函数相关的权限。机器学习函数属于机器学习模型，其权限通过模型来控制。相反，您可以授予与机器学习模型相关的权限。以下示例演示如何向所有用户授予权限，以便运行与 customer_churn 模型关联的机器学习函数。

```
GRANT EXECUTE ON MODEL customer_churn TO PUBLIC;
```

您还可以向用户授予对机器学习模型 customer_churn 的所有权限。

```
GRANT ALL on MODEL customer_churn TO ml_user;
```

如果架构中有机器学习函数，则授予与机器学习函数相关的 EXECUTE 权限将失败，即使该机器学习函数已通过 GRANT EXECUTE ON MODEL 获得 EXECUTE 权限。我们建议在使用 CREATE MODEL 命令时，通过单独的架构将机器学习函数单独保存在单独架构本身中。以下示例演示了如何执行此操作。

```
CREATE MODEL ml_schema.customer_churn
FROM customer_data
TARGET churn
FUNCTION ml_schema.customer_churn_prediction
IAM_ROLE default
SETTINGS (
 S3_BUCKET 'your-s3-bucket'
);
```

示例

以下示例向用户 fred 授予对 SALES 表的 SELECT 权限。

```
grant select on table sales to fred;
```

以下示例向用户 fred 授予对 QA_TICKIT schema 中所有表的 SELECT 权限。

```
grant select on all tables in schema qa_tickit to fred;
```

以下示例向用户组 QA_USERS 授予对 schema QA_TICKIT 的全部 schema 权限。Schema 权限包括 CREATE 和 USAGE。USAGE 向用户授予访问 schema 中对象的权限，但不授予对这些对象的 INSERT 或 SELECT 之类的权限。单独授予对每个对象的权限。

```
create group qa_users;
grant all on schema qa_tickit to group qa_users;
```

以下示例向组 QA_USERS 中的所有用户授予对 QA_TICKIT schema 中的 SALES 表的所有权限。

```
grant all on table qa_tickit.sales to group qa_users;
```

以下示例向组 QA_USERS 和 RO_USERS 中的所有用户授予对 QA_TICKIT 架构中的 SALES 表的所有权限。

```
grant all on table qa_tickit.sales to group qa_users, group ro_users;
```

以下示例向组 QA_USERS 中的所有用户授予对 QA_TICKIT schema 中的 SALES 表的 DROP 权限。

```
grant drop on table qa_tickit.sales to group qa_users;>
```

以下命令序列说明，具有对 schema 的访问权限并不表示授予对 schema 中的表的权限。

```
create user schema_user in group qa_users password 'Abcd1234';
create schema qa_tickit;
create table qa_tickit.test (col1 int);
grant all on schema qa_tickit to schema_user;

set session authorization schema_user;
select current_user;


current_user
--------------
schema_user
(1 row)


select count(*) from qa_tickit.test;


ERROR: permission denied for relation test [SQL State=42501]


set session authorization dw_user;
grant select on table qa_tickit.test to schema_user;
set session authorization schema_user;
select count(*) from qa_tickit.test;


count
-------
0
(1 row)
```

以下命令序列说明，具有对视图的访问权限并不意味着具有对基础表的访问权限。虽然名为 VIEW_USER 的用户已被授予 VIEW_DATE 的所有权限，但该用户无法从 DATE 表中选择数据。

```
create user view_user password 'Abcd1234';
create view view_date as select * from date;
grant all on view_date to view_user;
```

```
set session authorization view_user;
select current_user;


current_user
--------------
view_user
(1 row)


select count(*) from view_date;


count
-------
365
(1 row)


select count(*) from date;


ERROR:  permission denied for relation date
```

以下示例向用户 cust_name 授予对 cust_phone 表的 cust_profile 和 user1 列的 SELECT 权限。

```
grant select(cust_name, cust_phone) on cust_profile to user1;
```

以下示例向 cust_name 组授予对 cust_phone 和 cust_contact_preference 列的 SELECT 权限，并授予对 cust_profile 表的 sales_group 列的 UPDATE 权限。

```
grant select(cust_name, cust_phone), update(cust_contact_preference) on cust_profile to
 group sales_group;
```

下面的示例演示如何使用 ALL 关键字向 cust_profile 组授予对 sales_admin 表的三列的 SELECT 和 UPDATE 权限。

```
grant ALL(cust_name, cust_phone,cust_contact_preference) on cust_profile to group
 sales_admin;
```

以下示例向用户 cust_name 授予对 cust_profile_vw 视图的 user2 列的 SELECT 权限。

```
grant select(cust_name) on cust_profile_vw to user2;
```

授予数据共享访问权限的示例

以下示例显示了 GRANT 数据共享对特定数据库或基于数据共享创建的 schema 的使用权限。

在以下示例中，生产者端管理员向指定命名空间授予对 salesshare 数据共享的 USAGE 权限。

```
GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

在以下示例中，使用者端管理员向 Bob 授予对 sales_db 的 USAGE 权限。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

在以下示例中，使用者端管理员向 Analyst_role 角色授予对 sales_schema 架构的 GRANT USAGE 权限。sales_schema 是一个指向 sales_db 的外部架构。

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

此时，Bob 和 Analyst_role 可以访问 sales_schema 和 sales_db 中的所有数据库对象。

以下示例显示了对共享数据库中的对象授予额外的对象级权限。只有在用于创建共享数据库的 CREATE DATABASE 命令使用了 WITH PERMISSIONS 子句时，才需要这些额外的权限。如果 CREATE DATABASE 命令未使用 WITH PERMISSIONS，则在共享数据库上授予 USAGE 权限将授予对该数据库中所有对象的完全访问权限。

```
GRANT SELECT ON sales_db.sales_schema.tickit_sales_redshift to Bob;
```

授予限定范围权限的示例

以下示例将 Sales_db 数据库中所有当前和将来架构的使用权限授予给 Sales 角色。

```
GRANT USAGE FOR SCHEMAS IN DATABASE Sales_db TO ROLE Sales;
```

以下示例向用户 alice 授予对 Sales_db 数据库中所有当前和未来表的 SELECT 权限，同时还向 alice 授予权限以向其他用户授予对 Sales_db 中表的限定范围权限。

```
GRANT SELECT FOR TABLES IN DATABASE Sales_db TO alice WITH GRANT OPTION;
```

以下示例向用户 bob 授予对 Sales_schema 架构中函数的 EXECUTE 权限。

```
GRANT EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema TO bob;
```

以下示例向 Sales 角色授予对 ShareDb 数据库 ShareSchema 架构中所有表的所有权限。指定架构时，您可以使用由两部分组成的格式 database.schema 指定架构的数据库。

```
GRANT ALL FOR TABLES IN SCHEMA ShareDb.ShareSchema TO ROLE Sales;
```

下面的示例与前一个示例相同。您可以使用 DATABASE 关键字而不是使用由两部分组成的格式来指定数据库。

```
GRANT ALL FOR TABLES IN SCHEMA ShareSchema DATABASE ShareDb TO ROLE Sales;
```

授予 ASSUMEROLE 权限的示例

下面是授予 ASSUMEROLE 权限的示例。

以下示例显示了 REVOKE 语句，超级用户可以在集群上运行一次该语句，以便为用户和组启用 ASSUMEROLE 权限。然后，超级用户向用户和组授予相应命令的 ASSUMEROLE 权限。有关为用户和组启用 ASSUMEROLE 权限的信息，请参阅有关授予 ASSUMEROLE 权限的使用说明。

```
revoke assumerole on all from public for all;
```

以下示例向用户 reg_user1 授予 IAM 角色 Redshift-S3-Read 的 ASSUMEROLE 权限来执行 COPY 操作。

```
grant assumerole on 'arn:aws:iam::123456789012:role/Redshift-S3-Read'
to reg_user1 for copy;
```

以下示例向用户 reg_user1 授予 IAM 角色链 RoleA、RoleB 的 ASSUMEROLE 权限来执行 UNLOAD 操作。

```
grant assumerole
on 'arn:aws:iam::123456789012:role/RoleA,arn:aws:iam::210987654321:role/RoleB'
```

```
to reg_user1
for unload;
```

以下是使用 IAM 角色链 RoleA、RoleB 执行 UNLOAD 命令的示例。

```
unload ('select * from venue limit 10')
to 's3://companyb/redshift/venue_pipe_'
iam_role 'arn:aws:iam::123456789012:role/RoleA,arn:aws:iam::210987654321:role/RoleB';
```

以下示例向用户 reg_user1 授予 IAM 角色 Redshift-Exfunc 的 ASSUMEROLE 权限来创建外部函数。

```
grant assumerole on 'arn:aws:iam::123456789012:role/Redshift-Exfunc'
to reg_user1 for external function;
```

以下示例向用户 reg_user1 授予 IAM 角色 Redshift-model 的 ASSUMEROLE 权限来创建机器学习模型。

```
grant assumerole on 'arn:aws:iam::123456789012:role/Redshift-ML'
to reg_user1 for create model;
```

授予 ROLE 权限的示例

下面的示例将向 user1 授予 sample_role1。

```
CREATE ROLE sample_role1;
GRANT ROLE sample_role1 TO user1;
```

以下示例使用 WITH ADMIN OPTION 将 sample_role1 授予 user1，为用户 1 设置当前会话，而 user1 将 sample_role1 授予 user2。

```
GRANT ROLE sample_role1 TO user1 WITH ADMIN OPTION;
SET SESSION AUTHORIZATION user1;
GRANT ROLE sample_role1 TO user2;
```

下面的示例将向 sample_role2 授予 sample_role1。

```
GRANT ROLE sample_role1 TO ROLE sample_role2;
```

下面的示例将向 sample_role3 和 sample_role4 授予 sample_role2。然后尝试将 sample_role3 授予 sample_role1。

```
GRANT ROLE sample_role2 TO ROLE sample_role3;
GRANT ROLE sample_role3 TO ROLE sample_role2;
ERROR: cannot grant this role, a circular dependency was detected between these roles
```

下面的示例将向 sample_role1 授予 CREATE USER 系统权限。

```
GRANT CREATE USER TO ROLE sample_role1;
```

下面的示例将向 user1 授予系统定义角色 sys:dba。

```
GRANT ROLE sys:dba TO user1;
```

下面的示例尝试将循环依赖关系中的 sample_role3 授予 sample_role2。

```
CREATE ROLE sample_role3;
GRANT ROLE sample_role2 TO ROLE sample_role3;
GRANT ROLE sample_role3 TO ROLE sample_role2; -- fail
ERROR:  cannot grant this role, a circular dependency was detected between these roles
```

INSERT

主题

将新行插入到表中。您可以使用 VALUES 语法插入单行，使用 VALUES 语法插入多行，或者插入查询结果所定义的一行或多行 (INSERT INTO...SELECT)。

> **Note**
>
> 我们强烈建议您使用 COPY 命令来加载大量数据。使用单个 INSERT 语句填充表可能过于缓慢。此外，如果您的数据在其他 Amazon Redshift 数据库表中已经存在，请使用 INSERT

INTO SELECT 或 CREATE TABLE AS 来提高性能。有关使用 COPY 命令加载表的更多信息，请参阅加载数据。

> **Note**
>
> 单个 SQL 语句的最大大小为 16MB。

语法

```
INSERT INTO table_name [ ( column [, ...] ) ]
{DEFAULT VALUES |
VALUES ( { expression | DEFAULT } [, ...] )
[, ( { expression | DEFAULT } [, ...] )
[, ...] ] |
query }
```

参数

table_name

一个临时或永久表。只有表的所有者或对表具有 INSERT 权限的用户才能插入行。如果您使用 query 子句插入行，必须对查询中指定的表拥有 SELECT 权限。

> **Note**
>
> 使用 INSERT（外部表）可将 SELECT 查询的结果插入到外部目录中的现有表中。有关更多信息，请参阅 INSERT（外部表）。

column

您可以将值插入到表的一个或多个列中。您可以按任意顺序列出目标列名。如果不指定列的列表，则要插入的值必须按照在 CREATE TABLE 语句中声明的顺序对应于表列。如果要插入的值数小于表中的列数，则将加载前 n 列。

对于在 INSERT 语句中未列出（隐式或显式）的任何列，声明的默认值或 null 值将被加载到这些列。

DEFAULT VALUES

如果在创建表时，已向表中的列分配默认值，则可使用这些关键字插入完全包含默认值的行。如果有任何列不具有默认值，则会向这些列插入 null。如果任何列被声明为 NOT NULL，则 INSERT 语句会返回错误。

VALUES

使用此关键字可插入一行或多行，每一行包括一个或多个值。每一行的 VALUES 列表必须与列的列表对应。要插入多个行，请在每个表达式列表之间使用逗号分隔符。不要重复使用 VALUES 关键字。多行 INSERT 语句的所有 VALUES 列表必须包含相同数量的值。

expression

单个值，或计算结果为单个值的表达式。每个值必须与该值所插入到的列的数据类型相兼容。对于其数据类型与列的已声明数据类型不匹配的值，会尽可能地自动转换为兼容的数据类型。例如：

- 一个数字值 1.1 将以 1 值插入到 INT 列中。
- 一个数字值 100.8976 将以 100.90 值插入到 DEC(5,2) 列中。

您可以通过在表达式中包含类型强制转换语法，显式地将值转换成某个兼容的数据类型。例如，表 T1 中的列 COL1 是 CHAR(3) 列：

```
insert into t1(col1) values('Incomplete'::char(3));
```

此语句将值 Inc 插入到列中。

对于单行 INSERT VALUES 语句，可以使用标量子查询作为表达式。子查询的结果将插入到适当的列中。

> **Note**
>
> 对于多行 INSERT VALUES 语句，不支持将子查询用作表达式。

DEFAULT

使用此关键字可以按照在创建表时定义的方式，为列插入默认值。如果列不存在默认值，则插入 null 值。对于具有 NOT NULL 约束的列，如果没有在 CREATE TABLE 语句中明确为该列分配默认值，则不能将默认值插入到该列中。

query

通过定义任何查询，将一行或多行插入到表中。查询生成的所有行都将插入到表中。查询必须返回与表列相兼容的列的列表，不过列名称不一定要匹配。

使用说明

> **Note**
>
> 我们强烈建议您使用 COPY 命令来加载大量数据。使用单个 INSERT 语句填充表可能过于缓慢。此外，如果您的数据在其他 Amazon Redshift 数据库表中已经存在，请使用 INSERT INTO SELECT 或 CREATE TABLE AS 来提高性能。有关使用 COPY 命令加载表的更多信息，请参阅加载数据。

所插入值的数据格式必须与 CREATE TABLE 定义指定的数据格式匹配。

在表中插入大量新行后：

- 对表执行 Vacuum 操作，以回收存储空间并对行重新排序。
- 分析表以更新查询计划程序的统计数据。

如果在将值插入到 DECIMAL 列时，这些值超过了指定的小数位数，则会根据需要对加载的值向上取整。例如，如果将值 `20.259` 插入到 DECIMAL(8,2) 列中，则存储的值是 `20.26`。

您可以插入到 GENERATED BY DEFAULT AS IDENTITY 列。您可以使用您提供的值来更新定义为 GENERATED BY DEFAULT AS IDENTITY 的列。有关更多信息，请参阅 GENERATED BY DEFAULT AS IDENTITY。

INSERT 示例

TICKIT 数据库中的 CATEGORY 表包含以下行：

```
 catid | catgroup |  catname  |                 catdesc
-------+----------+-----------+--------------------------------------------
     1 | Sports   | MLB       | Major League Baseball
     2 | Sports   | NHL       | National Hockey League
     3 | Sports   | NFL       | National Football League
     4 | Sports   | NBA       | National Basketball Association
     5 | Sports   | MLS       | Major League Soccer
```

```
     6 | Shows    | Musicals  | Musical theatre
     7 | Shows    | Plays     | All non-musical theatre
     8 | Shows    | Opera     | All opera and light opera
     9 | Concerts | Pop       | All rock and pop music concerts
    10 | Concerts | Jazz      | All jazz singers and bands
    11 | Concerts | Classical | All symphony, concerto, and choir concerts
(11 rows)
```

使用与 CATEGORY 表类似的 schema 来创建 CATEGORY_STAGE 表，但为列定义默认值：

```
create table category_stage
(catid smallint default 0,
catgroup varchar(10) default 'General',
catname varchar(10) default 'General',
catdesc varchar(50) default 'General');
```

下面的 INSERT 语句从 CATEGORY 表中选择所有行并将它们插入 CATEGORY_STAGE 表。

```
insert into category_stage
(select * from category);
```

查询两旁的括号是可选的。

此命令在 CATEGORY_STAGE 表中插入新行，并按顺序为每列指定值：

```
insert into category_stage values
(12, 'Concerts', 'Comedy', 'All stand-up comedy performances');
```

您还可以插入结合使用特定值和默认值的新行：

```
insert into category_stage values
(13, 'Concerts', 'Other', default);
```

运行以下查询以返回插入的行：

```
select * from category_stage
where catid in(12,13) order by 1;

 catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
    12 | Concerts | Comedy  | All stand-up comedy performances
    13 | Concerts | Other   | General
```

```
(2 rows)
```

下面的示例说明了一些多行 INSERT VALUES 语句。第一个示例为两行插入特定的 CATID 值，并为这两行中的其他列插入默认值。

```
insert into category_stage values
(14, default, default, default),
(15, default, default, default);

select * from category_stage where catid in(14,15) order by 1;
 catid | catgroup | catname | catdesc
-------+----------+---------+---------
    14 | General  | General | General
    15 | General  | General | General
(2 rows)
```

下一个示例插入包含特定值和默认值的各种组合的三行：

```
insert into category_stage values
(default, default, default, default),
(20, default, 'Country', default),
(21, 'Concerts', 'Rock', default);

select * from category_stage where catid in(0,20,21) order by 1;
 catid | catgroup | catname | catdesc
-------+----------+---------+---------
     0 | General  | General | General
    20 | General  | Country | General
    21 | Concerts | Rock    | General
(3 rows)
```

本示例中的第一组 VALUES 生成的结果与为单行 INSERT 语句指定 DEFAULT VALUES 所生成的结果相同。

以下示例说明当表具有 IDENTITY 列时的 INSERT 行为。首先，创建 CATEGORY 表的新版本，然后将行从 CATEGORY 插入到新表：

```
create table category_ident
(catid int identity not null,
catgroup varchar(10) default 'General',
catname varchar(10) default 'General',
catdesc varchar(50) default 'General');
```

```
insert into category_ident(catgroup,catname,catdesc)
select catgroup,catname,catdesc from category;
```

请注意，您不能将特定的整数值插入到 CATID IDENTITY 列。IDENTITY 列值会自动生成。

以下示例说明了不能在多行 INSERT VALUES 语句中将子查询用作表达式：

```
insert into category(catid) values
((select max(catid)+1 from category)),
((select max(catid)+2 from category));

ERROR: can't use subqueries in multi-row VALUES
```

以下示例显示了在临时表中插入的内容，临时表中使用 WITH SELECT 子句填充了 venue 表中的数据。有关 venue 表的更多信息，请参阅 示例数据库。

首先，创建临时表 #venuetemp。

```
CREATE TABLE #venuetemp AS SELECT * FROM venue;
```

列出 #venuetemp 表中的行。

```
SELECT * FROM #venuetemp ORDER BY venueid;

venueid | venuename                | venuecity  | venuestate| venueseats
--------+--------------------------+------------+-----------+------------
1       Toyota Park                Bridgeview   IL          0
2       Columbus Crew Stadium      Columbus     OH          0
3       RFK Stadium                Washington   DC          0
4       CommunityAmerica Ballpark  Kansas City  KS          0
5       Gillette Stadium           Foxborough   MA          68756
...
```

使用 WITH SELECT 子句在 #venuetemp 表中插入 10 个重复的行。

```
INSERT INTO #venuetemp (WITH venuecopy AS (SELECT * FROM venue) SELECT * FROM venuecopy
 ORDER BY 1 LIMIT 10);
```

列出 #venuetemp 表中的行。

```
SELECT * FROM #venuetemp ORDER BY venueid;

venueid | venuename                | venuecity   | venuestate| venueseats
--------+--------------------------+-------------+-----------+------------
1       Toyota Park                Bridgeview    IL          0
1       Toyota Park                Bridgeview    IL          0
2       Columbus Crew Stadium      Columbus      OH          0
2       Columbus Crew Stadium      Columbus      OH          0
3       RFK Stadium                Washington    DC          0
3       RFK Stadium                Washington    DC          0
4       CommunityAmerica Ballpark  Kansas City   KS          0
4       CommunityAmerica Ballpark  Kansas City   KS          0
5       Gillette Stadium           Foxborough    MA          68756
5       Gillette Stadium           Foxborough    MA          68756
...
```

INSERT（外部表）

将 SELECT 查询的结果插入（如 AWS Glue、AWS Lake Formation 或 Apache Hive 元存储的）外部目录上的现有外部表中。使用与用于 CREATE EXTERNAL SCHEMA 命令相同的 AWS Identity and Access Management (IAM) 角色与外部目录和 Amazon S3 进行交互。

对于未分区的表，INSERT（外部表）命令根据指定的表属性和文件格式将数据写入在表中定义的 Amazon S3 位置。

对于已分区的表，INSERT（外部表）根据表中指定的分区键将数据写入 Amazon S3 位置。在 INSERT 操作完成后，它还会自动在外部目录中注册新分区。

您不能在事务数据块 (BEGIN ... END) 中运行 INSERT（外部表）。有关事务的更多信息，请参阅 可序列化的隔离。

语法

```
INSERT INTO external_schema.table_name
{ select_statement }
```

参数

external_schema.table_name

现有外部架构和要插入到的目标外部表的名称。

select_statement

通过定义任何查询将一行或多行插入外部表的语句。查询生成的所有行都将根据表定义以文本或 Parquet 格式写入到 Amazon S3 。查询必须返回与外部表中的列数据类型兼容的列列表。但是，列名称不必匹配。

使用说明

SELECT 查询中的列数必须与数据列和分区列的总和相同。每个数据列的位置和数据类型必须与外部表的位置和数据类型相匹配。分区列的位置必须位于 SELECT 查询的末尾，与在 CREATE EXTERNAL TABLE 命令中定义它们的顺序相同。列名称不必匹配。

在某些情况下，您可能需要对 AWS Glue 数据目录或 Hive 元存储运行 INSERT（外部表）命令。在 AWS Glue 的情况下，用于创建外部架构的 IAM 角色必须对 Amazon S3 和 AWS Glue 具有读取和写入权限。如果您使用 AWS Lake Formation 目录，则此 IAM 角色将成为新 Lake Formation 表的拥有者。此 IAM 角色必须至少具有以下权限：

- 对外部表的 SELECT、INSERT、UPDATE 权限
- 外部表的 Amazon S3 路径上的数据位置权限

为确保文件名是唯一的，Amazon Redshift 预设情况下对上载到 Amazon S3 的每个文件的名称使用以下格式。

*<date>*_*<time>*_*<microseconds>*_*<query_id>*_*<slice-number>*_part_*<part-number>*.*<format>*.

示例是 20200303_004509_810669_1007_0001_part_00.parquet。

运行 INSERT（外部表）命令时，请考虑以下事项：

- 不支持非 PARQUET 或 TEXTFILE 格式的外部表。
- 此命令支持现有的表属性，例如 'write.parallel'、'write.maxfilesize.mb'、'compression_type' 和 'serialization.null.format'。要更新这些值，请运行 ALTER TABLE SET TABLE PROPERTIES 命令。
- 'numRows' 表属性会在 INSERT 操作结束时自动更新。如果表属性不是由 CREATE EXTERNAL TABLE AS 操作创建的，则必须已经定义或添加到表中。
- 外部 SELECT 查询不支持 LIMIT 子句。请改为使用嵌套 LIMIT 子句。
- 您可以使用 STL_UNLOAD_LOG 表跟踪每个 INSERT（外部表）操作写入 Amazon S3 的文件。

- Amazon Redshift 仅支持 Amazon S3 标准加密用于 INSERT（外部表）。

INSERT（外部表）示例

以下示例将 SELECT 语句的结果插入到外部表中。

```
INSERT INTO spectrum.lineitem
SELECT * FROM local_lineitem;
```

以下示例使用静态分区将 SELECT 语句的结果插入到已分区的外部表中。分区列在 SELECT 语句中是硬编码的。分区列必须位于查询的末尾。

```
INSERT INTO spectrum.customer
SELECT name, age, gender, 'May', 28 FROM local_customer;
```

以下示例使用动态分区将 SELECT 语句的结果插入到已分区的外部表中。分区列不是硬编码的。数据自动添加到现有分区文件夹中，或者，如果添加了新分区，则会将数据自动添加到新文件夹中。

```
INSERT INTO spectrum.customer
SELECT name, age, gender, month, day FROM local_customer;
```

LOCK

限制对数据库表的访问。此命令只在事务块内部运行时才有意义。

LOCK 命令在“ACCESS EXCLUSIVE”模式下获取表级别的锁定；如果需要，会等待所有冲突的锁定释放。通过这种方式明确地锁定表时，会导致从其他事务或会话尝试对表执行的读取和写入操作等待。当一个用户明确地锁定表时，会阻止另一个用户从该表中选择数据或在该表中加载数据。当包含 LOCK 命令的事务完成后，会释放锁定。

引用表的命令会隐式地获取限制性较低的表锁定，例如写入操作。例如，如果某个用户尝试从表中读取数据，而另一个用户正在更新该表，那么读取的数据将是已提交的数据的快照。（在某些情况下，如果查询违反了可序列化隔离规则，则将停止。）请参阅 管理并发写入操作。

一些 DDL 操作（例如 DROP TABLE 和 TRUNCATE）会创建独占锁定。这些操作会阻止读取数据。

如果发生锁定冲突，Amazon Redshift 会显示错误消息，对启动冲突事务的用户发出提示。收到锁定冲突的事务将会停止。每次发生锁定冲突时，Amazon Redshift 会将一个条目写入到 STL_TR_CONFLICT 表。

语法

```
LOCK [ TABLE ] table_name [, ...]
```

参数

TABLE

可选关键字。

table_name

要锁定的表的名称。通过使用逗号分隔的表名列表可以锁定多个表。您不能锁定视图。

示例

```
begin;

lock event, sales;

...
```

MERGE

按条件将源表中的行合并到目标表中。通常这只能通过单独使用多个插入、更新或删除语句来实现。有关 MERGE 允许您合并的操作的更多信息，请参阅 UPDATE、DELETE 和 INSERT。

语法

```
MERGE INTO target_table
USING source_table [ [ AS ] alias ]
ON match_condition
[ WHEN MATCHED THEN { UPDATE SET col_name = { expr } [,...] | DELETE }
WHEN NOT MATCHED THEN INSERT [ ( col_name [,...] ) ] VALUES ( { expr } [, ...] ) |
REMOVE DUPLICATES ]
```

参数

target_table

MERGE 语句合并到的临时表或永久表。

source_table

提供要合并到 target_table 的行的临时表或永久表。source_table 也可以是 Spectrum 表。source_table 不能是视图或子查询。

alias

source_table 的临时备用名称。

此参数为可选的。在别名前加上 AS 也是可选的。

match_condition

在源表列和目标表列之间指定同等的谓词，用于确定 source_table 中的行是否可以与 target_table 中的行匹配。如果满足条件，MERGE 会对该行运行 matched_clause。否则 MERGE 会为该行运行 not_matched_clause。

WHEN MATCHED

指定当源行和目标行之间的匹配条件计算结果为 True 时要运行的操作。您可以指定 UPDATE 操作或 DELETE 操作。

UPDATE

更新 target_table 中的匹配行。仅更新您在 col_name 中指定的值。

删除

删除 target_table 中的匹配行。

WHEN NOT MATCHED

指定当匹配条件计算结果为 False 或 Unknown 时要运行的操作。只能为此子句指定 INSERT 插入操作。

INSERT

在 target_table 中插入一行。可以按任意顺序列出目标 col_name。如果您不提供任何 col_name 值，则默认顺序是表中所有列的声明顺序。

col_name

要修改的一个或多个列名。指定目标列时不包括表名。

expr

定义新 col_name 值的表达式。

REMOVE DUPLICATES

指定 MERGE 命令在简化模式下运行。简化模式具有以下要求：

- target_table 和 source_table 必须具有相同的列数和兼容的列类型。
- 在 MERGE 命令中省略 WHEN 子句以及 UPDATE 和 INSERT 子句。
- 在 MERGE 命令中使用 REMOVE DUPLICATES 子句。

在简化模式下，MERGE 执行以下操作：

- target_table 中与 source_table 中存在匹配项的行将更新为与 source_table 中的值相匹配。
- source_table 中与 target_table 中不存在匹配项的行将插入到 target_table 中。
- 当 target_table 中的多个行与 source_table 中的同一行匹配时，将删除重复的行。Amazon Redshift 保留一行并对其进行更新。与 source_table 中的行不匹配的重复行将保持不变。

与使用 WHEN MATCHED 和 WHEN NOT MATCHED 相比，使用 REMOVE DUPLICATES 可获得更好的性能。如果 target_table 和 source_table 兼容，并且您不需要在 target_table 中保留重复行，我们建议您使用 REMOVE DUPLICATES。

使用说明

- 要运行 MERGE 语句，您必须是 source_table 和 target_table 的所有者，或者具有这些表的 SELECT 权限。此外，您必须拥有 target_table 的 UPDATE、DELETE 和 INSERT 权限，具体取决于 MERGE 语句中包括的操作。
- target_table 不能是系统表、目录表或外部表。
- source_table 和 target_table 不能是同一个表。
- 不能在 MERGE 语句中使用 WITH 子句。
- source_table 中的行不能匹配 target_table 中的多行。

考虑以下示例：

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (1, 'Bob'), (2, 'John');
INSERT INTO source VALUES (1, 'Tony'), (1, 'Alice'), (3, 'Bill');

MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN UPDATE SET id = source.id, name = source.name
```

```
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);
ERROR: Found multiple matches to update the same tuple.

MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN DELETE
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);
ERROR: Found multiple matches to update the same tuple.
```

这两个 MERGE 语句中的操作均失败，因为 source 表中有多个行具有 ID 值 1。

- match_condition 和 expr 不能部分引用 SUPER 类型列。例如，如果您的 SUPER 类型对象是数组或结构，则不能为 match_condition 或 expr 使用该列的个别元素，但您可以使用整列。

  考虑以下示例：

```
CREATE TABLE IF NOT EXISTS target (key INT, value SUPER);
CREATE TABLE IF NOT EXISTS source (key INT, value SUPER);

INSERT INTO target VALUES (1, JSON_PARSE('{"key": 88}'));
INSERT INTO source VALUES (1, ARRAY(1, 'John')), (2, ARRAY(2, 'Bill'));

MERGE INTO target USING source ON target.key = source.key
WHEN matched THEN UPDATE SET value = source.value[0]
WHEN NOT matched THEN INSERT VALUES (source.key, source.value[0]);
ERROR: Partial reference of SUPER column is not supported in MERGE statement.
```

  有关 SUPER 类型的更多信息，请参阅 SUPER 类型。

- 如果 source_table 很大，则将 target_table 和 source_table 中的联接列定义为分配键可以提高性能。
- 要使用 REMOVE DUPLICATES 子句，您需要对 target_table 拥有 SELECT、INSERT 和 DELETE 权限。

示例

以下示例创建了两个表，然后对它们运行 MERGE 操作，更新目标表中的匹配行并插入不匹配的行。然后，它将另一个值插入源表并运行另一个 MERGE 操作，这次是删除匹配行并从源表插入新行。

首先创建并填充源表和目标表。

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));
```

```
INSERT INTO target VALUES (101, 'Bob'), (102, 'John'), (103, 'Susan');
INSERT INTO source VALUES (102, 'Tony'), (103, 'Alice'), (104, 'Bill');

SELECT * FROM target;
 id  |    name
-----+------------
 101 | Bob
 102 | John
 103 | Susan
(3 rows)

SELECT * FROM source;
 id  |    name
-----+------------
 102 | Tony
 103 | Alice
 104 | Bill
(3 rows)
```

接下来，将源表合并到目标表，用匹配行更新目标表，并插入源表中没有匹配的行。

```
MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN UPDATE SET id = source.id, name = source.name
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);

SELECT * FROM target;
 id  |    name
-----+------------
 101 | Bob
 102 | Tony
 103 | Alice
 104 | Bill
(4 rows)
```

请注意，ID 值为 102 和 103 的行已更新，以匹配目标表中的名称值。此外，在目标表中插入一个 ID 值为 104 且名称值为 Bill 的新行。

接下来，在源表中插入新行。

```
INSERT INTO source VALUES (105, 'David');

SELECT * FROM source;
```

```
 id  |    name
-----+------------
 102 | Tony
 103 | Alice
 104 | Bill
 105 | David
(4 rows)
```

最后，运行合并操作，删除目标表中的匹配行，然后插入不匹配的行。

```
MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN DELETE
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);

SELECT * FROM target;
 id  |    name
-----+------------
 101 | Bob
 105 | David
(2 rows)
```

从目标表中删除 ID 值为 102、103 和 104 的行，并在目标表中插入 ID 值为 105 且名称值为 David 的新行。

以下示例显示了使用 REMOVE DUPLICATES 子句的 MERGE 命令。

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (30, 'Tony'), (11, 'Alice'), (23, 'Bill');
INSERT INTO source VALUES (23, 'David'), (22, 'Clarence');

MERGE INTO target USING source ON target.id = source.id REMOVE DUPLICATES;

SELECT * FROM target;
id | name
---+------------
30 | Tony
11 | Alice
23 | David
22 | Clarence
(4 rows)
```

以下示例显示了使用 REMOVE DUPLICATES 子句的 MERGE 命令，如果重复行在 source_table 中有匹配的行，则从 target_table 中移除重复行。

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (30, 'Tony'), (30, 'Daisy'), (11, 'Alice'), (23, 'Bill'),
 (23, 'Nikki');
INSERT INTO source VALUES (23, 'David'), (22, 'Clarence');

MERGE INTO target USING source ON target.id = source.id REMOVE DUPLICATES;

SELECT * FROM target;
id | name
---+------------
30 | Tony
30 | Daisy
11 | Alice
23 | David
22 | Clarence
(5 rows)
```

MERGE 运行后，target_table 中只有一行的 ID 值为 23。由于 source_table 中没有 ID 值为 30 的行，因此 ID 值为 30 的两个重复行仍保留在 target_table 中。

另请参阅

INSERT, UPDATE, 删除

PREPARE

准备语句以便执行。

PREPARE 会创建一个预编译语句。在运行 PREPARE 语句时，会对指定的语句（SELECT、INSERT、UPDATE 或 DELETE）进行解析、重写和计划。为预编译语句发出 EXECUTE 命令时，Amazon Redshift 可能会选择修改查询执行计划（以便根据指定的参数值来提高性能），然后再执行预编译语句。

语法

```
PREPARE plan_name [ (datatype [, ...] ) ] AS statement
```

参数

plan_name

为此特定预编译语句指定的任意名称。它在单个会话中必须是唯一的，随后用于执行或取消分配先前预编译的语句。

datatype

预编译语句的参数的数据类型。要在预编译语句自身中引用参数，请使用 $1、$2 等等。

statement

任意 SELECT、INSERT、UPDATE 或 DELETE 语句。

使用说明

预编译语句可以接受参数和值，它们在执行语句时将会被取代。要在预编译语句中包含参数，请在 PREPARE 语句中提供数据类型的列表；并在要预编译的语句自身中，通过使用 $1, $2, ... 表示法来按位置引用参数。执行该语句时，会在 EXECUTE 语句中为这些参数指定实际值。有关更多信息，请参阅 EXECUTE。

预编译语句只在当前会话的持续时间内有效。当会话结束时，预编译语句将被丢弃，因此必须重新创建该语句才能再次使用。这也意味着一个预编译语句不能同时由多个数据库客户端使用；不过，每个客户端可以创建自己的预编译语句加以使用。可以使用 DEALLOCATE 命令手动删除预编译语句。

当一个会话用于执行大量类似的语句时，预编译语句能够发挥最大的性能优势。正如上文所述，对于预编译语句的每次新执行，Amazon Redshift 可能会根据指定的参数值，再次修改查询执行计划来提高性能。要检查 Amazon Redshift 为任何特定 EXECUTE 语句选择的查询执行计划，请使用 EXPLAIN 命令。

有关查询计划的更多信息以及 Amazon Redshift 为查询优化收集的统计数据，请参阅 ANALYZE 命令。

示例

创建一个临时表，准备 INSERT 语句并执行该语句：

```
DROP TABLE IF EXISTS prep1;
```

```
CREATE TABLE prep1 (c1 int, c2 char(20));
PREPARE prep_insert_plan (int, char)
AS insert into prep1 values ($1, $2);
EXECUTE prep_insert_plan (1, 'one');
EXECUTE prep_insert_plan (2, 'two');
EXECUTE prep_insert_plan (3, 'three');
DEALLOCATE prep_insert_plan;
```

准备 SELECT 语句并执行该语句：

```
PREPARE prep_select_plan (int)
AS select * from prep1 where c1 = $1;
EXECUTE prep_select_plan (2);
EXECUTE prep_select_plan (3);
DEALLOCATE prep_select_plan;
```

另请参阅

DEALLOCATE, EXECUTE

REFRESH MATERIALIZED VIEW

刷新实体化视图。

创建实体化视图时，其内容将反映当时基础数据库表的状态。实体化视图中的数据将保持不变，即使应用程序更改基础表中的数据也是如此。要更新实体化视图中的数据，您可以随时使用 REFRESH MATERIALIZED VIEW 语句。使用此语句时，Amazon Redshift 会标识已在一个或多个基表中进行的更改，然后将这些更改应用于实体化视图。

有关实体化视图的更多信息，请参阅在 Amazon Redshift 中创建实体化视图。

语法

```
REFRESH MATERIALIZED VIEW mv_name
```

参数

mv_name

要刷新的实体化视图的名称。

使用说明

只有实体化视图的拥有者才能对该实体化视图执行 REFRESH MATERIALIZED VIEW 操作。此外，所有者必须对基础基表具有 SELECT 权限才能成功运行 REFRESH MATERIALIZED VIEW。

REFRESH MATERIALIZED VIEW 命令作为其自己的事务运行。遵循 Amazon Redshift 事务语义来确定基表中的哪些数据对于 REFRESH 命令是可见的，或者何时使命令 REFRESH 所做的更改对 Amazon Redshift 中运行的其他事务可见。

- 对于增量实体化视图，REFRESH MATERIALIZED VIEW 操作仅使用那些已提交的基表行。因此，如果刷新操作在同一事务中的数据操作语言 (DML) 语句之后运行，则对该 DML 语句的更改将对于刷新不可见。
- 对于实体化视图的完全刷新，根据通常的 Amazon Redshift 事务语义，REFRESH MATERIALIZED VIEW 会看到对刷新事务可见的所有基表行。
- 根据输入参数类型，Amazon Redshift 仍支持以下采用特定输入参数类型的函数所对应的实体化视图的增量刷新：DATE（时间戳）、DATE_PART（日期、时间、间隔、time-tz）、DATE_TRUNC（时间戳、间隔）。
- 基表位于数据共享中的实例化视图支持增量刷新。

Amazon Redshift 中的一些操作会与实体化视图进行交互。这些操作中的某些操作可能会强制 REFRESH MATERIALIZED VIEW 完全重新计算实体化视图，即使定义该视图的查询仅使用可以用于增量刷新的 SQL 功能。例如：

- 如果未刷新实体化视图，则可能会阻止后台 vacuum 操作。在内部定义的阈值期后，将允许运行 vacuum 操作。在发生此 vacuum 操作时，所有依赖的实体化视图都将标记为在下次刷新时重新计算（即使它们是增量的）。有关 VACUUM 的信息，请参阅 VACUUM。有关事件和状态更改的更多信息，请参阅 STL_MV_STATE。
- 一些由用户对基表发起的操作会强制在下次运行 REFRESH 操作时完全重新计算实体化视图。此类操作的示例包括手动调用的 VACUUM、经典调整大小、ALTER DISTKEY 操作、ALTER SORTKEY 操作和截断操作。有关事件和状态更改的更多信息，请参阅 STL_MV_STATE。

对数据共享中的实体化视图进行增量刷新

在共享基表时，Amazon Redshift 支持对消费者数据共享中的实体化视图进行自动和增量刷新。增量刷新是一项操作，其中 Amazon Redshift 可识别上次刷新后发生的一个或多个基表中的更改，并仅更新实体化视图中的相应记录。有关此行为的更多信息，请参阅 CREATE MATERIALIZED VIEW。

增量刷新限制

Amazon Redshift 目前不支持使用以下任意 SQL 元素通过查询定义的实体化视图的递增刷新：

- OUTER JOIN（RIGHT、LEFT 或 FULL）。
- 集操作：UNION、INTERSECT、EXCEPT、MINUS。
- 当它出现在子查询中，并且聚合函数或 GROUP BY 子句存在于查询中时，UNION ALL。
- 聚合函数：
  MEDIAN、PERCENTILE_CONT、LISTAGG、STDDEV_SAMP、STDDEV_POP、APPROXIMATE COUNT、APPROXIMATE PERCENTILE 以及按位聚合函数。

  > **Note**
  >
  > 支持 COUNT、SUM、MIN、MAX 和 AVG 聚合函数。

- DISTINCT 聚合函数，如 DISTINCT COUNT、DISTINCT SUM 等等。
- 窗口函数。
- 使用临时表进行查询优化的查询，例如优化常用的子表达式。
- 子查询
- 在定义实体化视图的查询中引用以下格式的外部表。
  - Delta Lake
  - Hudi

  对于使用引用其他格式的外部表定义的实体化视图，预览版跟踪支持增量刷新。有关设置预览版集群的更多信息，请参阅《Amazon Redshift 管理指南》中的创建预览版集群。有关设置预览工作组的信息，请参阅《Amazon Redshift 管理指南》中的创建预览工作组。
- 可变函数，如日期-时间函数，RANDOM 和非 STABLE 用户定义函数。
- 有关零 ETL 集成的增量刷新限制，请参阅将零 ETL 集成与 Amazon Redshift 结合使用时的注意事项。

有关实体化视图限制的更多信息，包括 VACUUM 等后台操作对实体化视图刷新操作的影响，请参阅使用说明。

示例

以下示例刷新 `tickets_mv` 实体化视图。

```
REFRESH MATERIALIZED VIEW tickets_mv;
```

RESET

将配置参数的值还原为其默认值。

可以重置单个指定的参数或一次性重置所有参数。要将参数设置为特定的值，请使用 SET 命令。要显示参数的当前值，请使用 SHOW 命令。

语法

```
RESET { parameter_name | ALL }
```

以下语句将会话上下文变量的值设置为 NULL。

```
RESET { variable_name | ALL }
```

参数

parameter_name

要重置的参数的名称。有关参数的更多文档，请参阅修改服务器配置。

ALL

重置所有运行时参数，包括所有会话上下文变量。

variable

要重置的变量的名称。如果 RESET 的值是会话上下文变量，则 Amazon Redshift 会将它设置为 NULL。

示例

以下示例将 query_group 参数重置为其默认值：

```
reset query_group;
```

以下示例将所有运行时参数重置为其默认值。

```
reset all;
```

以下示例重置上下文变量。

```
RESET app_context.user_id;
```

REVOKE

从用户或角色删除访问权限，例如，用于创建、删除或更新表的权限。

您只能使用 ON SCHEMA 语法将针对外部架构的 GRANT 或 REVOKE USAGE 权限授予数据库用户和角色。将 ON EXTERNAL SCHEMA 与 AWS Lake Formation 搭配使用时，您只能向 AWS Identity and Access Management (IAM) 角色授予 GRANT 和 REVOKE 权限。有关权限的列表，请参阅“语法”。

对于存储过程，默认情况下，向 PUBLIC 授予 USAGE ON LANGUAGE plpgsql 权限。EXECUTE ON PROCEDURE 权限默认情况下只授予拥有者和超级用户。

在 REVOKE 命令中指定要删除的权限。要授予权限，请使用 GRANT 命令。

语法

```
REVOKE [ GRANT OPTION FOR ]
{ { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES | ALTER | TRUNCATE } [,...] |
 ALL [ PRIVILEGES ] }
ON { [ TABLE ] table_name [, ...] | ALL TABLES IN SCHEMA schema_name [, ...] }
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]


REVOKE [ GRANT OPTION FOR ]
{ { CREATE | TEMPORARY | TEMP | ALTER } [,...] | ALL [ PRIVILEGES ] }
ON DATABASE db_name [, ...]
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]


REVOKE [ GRANT OPTION FOR ]
{ { CREATE | USAGE | ALTER } [,...] | ALL [ PRIVILEGES ] }
ON SCHEMA schema_name [, ...]
```

```
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]


REVOKE [ GRANT OPTION FOR ]
EXECUTE
    ON FUNCTION function_name ( [ [ argname ] argtype [, ...] ] ) [, ...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]


REVOKE [ GRANT OPTION FOR ]
{ { EXECUTE } [,...] | ALL [ PRIVILEGES ] }
    ON PROCEDURE procedure_name ( [ [ argname ] argtype [, ...] ] ) [, ...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]


REVOKE [ GRANT OPTION FOR ]
USAGE
    ON LANGUAGE language_name [, ...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]
```

撤销表的列级别权限

以下是 Amazon Redshift 表和视图上的列级别权限的语法。

```
REVOKE { { SELECT | UPDATE } ( column_name [, ...] ) [, ...] | ALL [ PRIVILEGES ]
 ( column_name [,...] ) }
     ON { [ TABLE ] table_name [, ...] }
     FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
     [ RESTRICT ]
```

撤销 ASSUMEROLE 权限

以下是从具有指定角色的用户和组上撤销 ASSUMEROLE 权限的语法。

```
REVOKE ASSUMEROLE
    ON { 'iam_role' [, ...]  | default | ALL }
    FROM { user_name | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
    FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL }
```

撤销 Lake Formation 的 Redshift Spectrum 的权限

以下是 Redshift Spectrum 与 Lake Formation 集成的语法。

```
REVOKE [ GRANT OPTION FOR ]
{ SELECT | ALL [ PRIVILEGES ] } ( column_list )
    ON EXTERNAL TABLE schema_name.table_name
    FROM { IAM_ROLE iam_role } [, ...]

REVOKE [ GRANT OPTION FOR ]
{ { SELECT | ALTER | DROP | DELETE | INSERT }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL TABLE schema_name.table_name [, ...]
    FROM { { IAM_ROLE iam_role } [, ...] | PUBLIC }

REVOKE [ GRANT OPTION FOR ]
{ { CREATE | ALTER | DROP }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL SCHEMA schema_name [, ...]
    FROM { IAM_ROLE iam_role } [, ...]
```

撤销数据共享权限

生产者端数据共享权限

以下是使用 REVOKE 删除用户或角色的 ALTER 或 SHARE 权限的语法。权限已被撤销的用户无法再更改数据共享，也无法向使用者授予使用权限。

```
REVOKE { ALTER | SHARE } ON DATASHARE datashare_name
 FROM { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

以下是使用 REVOKE 删除使用者对数据共享的访问权限的语法。

```
REVOKE USAGE
 ON DATASHARE datashare_name
 FROM NAMESPACE 'namespaceGUID' [, ...] | ACCOUNT 'accountnumber' [ VIA DATA CATALOG ]
 [, ...]
```

以下是从 Lake Formation 账户中撤销数据共享使用权限的示例。

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '123456789012' VIA DATA CATALOG;
```

使用者端数据共享权限

以下是根据数据共享创建的特定数据库或 schema 的 REVOKE 数据共享使用权限的语法。撤销使用 WITH PERMISSIONS 子句创建的数据库的使用权限并不能撤销您授予用户或角色的任何其他权限，包括为基础对象授予的对象级权限。如果您向该用户或角色重新授予使用权限，他们将保留在您撤销使用权限之前拥有的所有其他权限。

```
REVOKE USAGE ON { DATABASE shared_database_name [, ...] | SCHEMA shared_schema}
 FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

撤销限定范围权限

限定范围权限允许您向用户或角色授予对数据库或架构中某一类型的所有对象的访问权限。具有限定范围权限的用户和角色对数据库或架构中的所有当前和未来对象具有指定权限。

以下是撤销用户和角色的限定范围权限的语法。有关范围限定权限的更多信息，请参阅限定范围权限。

```
REVOKE [ GRANT OPTION ]
{ CREATE | USAGE | ALTER } [,...] | ALL [ PRIVILEGES ] }
FOR SCHEMAS IN
DATABASE db_name
FROM { username | ROLE role_name } [, ...]

REVOKE [ GRANT OPTION ]
{ { SELECT | INSERT | UPDATE | DELETE | DROP | ALTER | TRUNCATE | REFERENCES }
 [, ...] } | ALL [PRIVILEGES] } }
FOR TABLES IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
FROM { username] | ROLE role_name} [, ...]

REVOKE [ GRANT OPTION ] { EXECUTE | ALL [ PRIVILEGES ] }
FOR FUNCTIONS IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
FROM { username | ROLE role_name | } [, ...]

REVOKE [ GRANT OPTION ] { EXECUTE | ALL [ PRIVILEGES ] }
FOR PROCEDURES IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
FROM { username | ROLE role_name | } [, ...]

REVOKE [ GRANT OPTION ] USAGE
FOR LANGUAGES IN
```

```
{DATABASE db_name}
FROM { username | ROLE role_name } [, ...]
```

请注意，范围限定的权限不区分函数的权限和过程的权限。例如，以下语句撤销 bob 对架构 Sales_schema 中函数和过程的 EXECUTE 权限。

```
REVOKE EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema FROM bob;
```

撤销机器学习权限

以下是有关 Amazon Redshift 上机器学习模型权限的语法。

```
REVOKE [ GRANT OPTION FOR ]
    CREATE MODEL FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
    [ RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON MODEL model_name [, ...]

    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
    [ RESTRICT ]
```

撤销角色权限

以下是撤销在 Amazon Redshift 上的角色权限的语法。

```
REVOKE [ ADMIN OPTION FOR ] { ROLE role_name } [, ...] FROM { user_name } [, ...]
```

```
REVOKE { ROLE role_name } [, ...] FROM { ROLE role_name } [, ...]
```

以下是撤销角色在 Amazon Redshift 上的系统权限的语法。

```
REVOKE
  {
    { CREATE USER | DROP USER | ALTER USER |
    CREATE SCHEMA | DROP SCHEMA |
    ALTER DEFAULT PRIVILEGES |
    ACCESS CATALOG |
```

```
    CREATE TABLE | DROP TABLE | ALTER TABLE |
    CREATE OR REPLACE FUNCTION | CREATE OR REPLACE EXTERNAL FUNCTION |
    DROP FUNCTION |
    CREATE OR REPLACE PROCEDURE | DROP PROCEDURE |
    CREATE OR REPLACE VIEW | DROP VIEW |
    CREATE MODEL | DROP MODEL |
    CREATE DATASHARE | ALTER DATASHARE | DROP DATASHARE |
    CREATE LIBRARY | DROP LIBRARY |
    CREATE ROLE | DROP ROLE
    TRUNCATE TABLE
    VACUUM | ANALYZE | CANCEL }[, ...]
  }
  | { ALL [ PRIVILEGES ] }
FROM { ROLE role_name } [, ...]
```

撤销对行级别安全性策略筛选器的解释权限

以下语法用于撤消解释 EXPLAIN 计划中查询的行级别安全性策略筛选器的权限。您可以使用 REVOKE 语句撤消该权限。

```
REVOKE EXPLAIN RLS FROM ROLE rolename
```

以下语法用于授予为查询绕开行级别安全性策略的权限。

```
REVOKE IGNORE RLS FROM ROLE rolename
```

以下语法用于从指定的行级别安全性策略撤销权限。

```
REVOKE SELECT ON [ TABLE ] table_name [, ...]
            FROM RLS POLICY policy_name [, ...]
```

参数

GRANT OPTION FOR

仅撤消将指定权限授予其他用户的选项，而不撤消权限本身。您无法从组或 PUBLIC 撤消 GRANT OPTION。

SELECT

撤消用于通过 SELECT 语句从表或视图中选择数据的权限。

INSERT

撤消用于通过 INSERT 语句或 COPY 语句将数据加载到表中的权限。

UPDATE

撤消用于通过 UPDATE 语句更新表列的权限。

删除

撤消用于从表中删除数据行的权限。

REFERENCES

撤消用于创建外键约束的权限。您应该在被引用表和引用表上撤消此权限。

TRUNCATE

撤销截断表的权限。如果没有此权限，则只有表的拥有者或超级用户才可以截断表。有关 TRUNCATE 命令的更多信息，请参阅 the section called “TRUNCATE”。

ALL [PRIVILEGES]

一次性从指定的用户或组撤消所有可用权限。PRIVILEGES 关键字是可选的。

ALTER

根据数据库对象，从用户或用户组撤销以下权限：

- 对于表，ALTER 撤销更改表或视图的权限。有关更多信息，请参阅 ALTER TABLE。
- 对于数据库，ALTER 撤销更改更数据库的权限。有关更多信息，请参阅 ALTER DATABASE。
- 对于架构，ALTER 撤销更改模式的权限。有关更多信息，请参阅 ALTER SCHEMA。
- 对于外部表，ALTER 撤销对为 Lake Formation 启用的 AWS Glue Data Catalog 中的表进行更改的权限。此权限仅在使用 Lake Formation 时适用。

DROP

撤消对表的删除权限。此权限仅在 Amazon Redshift 以及为 Lake Formation 启用的 AWS Glue Data Catalog 中适用。

ASSUMEROLE

从具有指定角色的用户、角色或组撤消运行 COPY、UNLOAD、EXTERNAL FUNCTION 或 CREATE MODEL 命令的权限。

ON [TABLE] table_name

撤消对表或视图的指定权限。TABLE 关键字是可选的。

ON ALL TABLES IN SCHEMA schema_name

撤消对引用的 Schema 中的所有表的指定权限。

(column_name [,...]) ON TABLE table_name

撤消用户、组或 PUBLIC 对 Amazon Redshift 表或视图的指定列的指定权限。

(column_list) ON EXTERNAL TABLE schema_name.table_name

从 IAM 角色撤消对于引用的 Schema 中 Lake Formation 表的指定列的指定权限。

ON EXTERNAL TABLE schema_name.table_name

从 IAM 角色撤消对于引用的 Schema 中的指定 Lake Formation 表的指定权限。

ON EXTERNAL SCHEMA schema_name

从 IAM 角色撤消对于引用的 Schema 的指定权限。

FROM IAM_ROLE iam_role

表示丢失权限的 IAM 角色。

ROLE role_name

从指定的角色撤销权限。

GROUP group_name

从指定的用户组撤消权限。

PUBLIC

从所有用户撤消指定权限。PUBLIC 表示一个始终包含所有用户的组。单个用户的权限包含向 PUBLIC 授予的权限、向用户所属的所有组授予的权限以及向用户单独授予的任何权限。

从 Lake Formation 外部表中撤销 PUBLIC 会将该权限从 Lake Formation 所有人组撤销。

CREATE

根据数据库对象，从用户或组撤销以下权限：

- 对于数据库，对 REVOKE 使用 CREATE 子句将阻止用户在数据库中创建 schema。

- 对于 schemas，对 REVOKE 使用 CREATE 子句将阻止用户在 schema 中创建对象。要重命名对象，用户必须具有 CREATE 权限并拥有要重命名的对象。

> **Note**
>
> 默认情况下，所有用户都对 PUBLIC Schema 具有 CREATE 和 USAGE 权限。

TEMPORARY | TEMP

撤销在指定的数据库中创建临时表的权限。

> **Note**
>
> 默认情况下，向用户授予权限以通过其在 PUBLIC 组中自动获得的成员资格来创建临时表。要删除任意用户创建临时表的权限，请撤销 PUBLIC 组的 TEMP 权限，然后明确向特定用户或用户组授予用于创建临时表的权限。

ON DATABASE db_name

撤销对指定数据库的权限。

USAGE

撤销对特定架构中的对象的 USAGE 权限，这将使用户无法访问这些对象。必须单独撤销这些对象的特定操作权限（例如，对函数的 EXECUTE 权限）。

> **Note**
>
> 默认情况下，所有用户都对 PUBLIC Schema 具有 CREATE 和 USAGE 权限。

ON SCHEMA schema_name

撤销对指定架构的权限。您可以使用架构权限来控制表的创建；数据库的 CREATE 权限仅控制架构的创建。

RESTRICT

仅撤销用户直接授予的那些权限。此行为是默认行为。

EXECUTE ON PROCEDURE procedure_name

撤销对特定存储过程的 EXECUTE 权限。由于存储过程名称可重载，因此您必须包含过程的参数列表。有关更多信息，请参阅 命名存储过程。

EXECUTE ON ALL PROCEDURES IN SCHEMA procedure_name

撤销对引用的架构中的所有过程的指定权限。

USAGE ON LANGUAGE language_name

撤销对某种语言的 USAGE 权限。对于 Python 用户定义的函数 (UDF)，请使用 plpythonu。对于 SQL UDF，使用 sql。对于存储过程，请使用 plpgsql。

要创建 UDF，您必须具有使用 SQL 或 plpythonu (Python) 语言的权限。默认情况下，向 PUBLIC 授予 USAGE ON LANGUAGE SQL。但是，您必须明确授予 USAGE ON LANGUAGE PLPYTHONU 权限才能指定用户或组。

要撤销 SQL 的使用权限，请先从 PUBLIC 撤销使用权限。然后，仅向允许创建 SQL UDF 的特定用户或组授予 SQL 使用权限。以下示例将从 PUBLIC 撤销对 SQL 的使用权限，然后向用户组 udf_devs 授予使用权限。

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

有关更多信息，请参阅 UDF 安全性和权限。

要撤销存储过程的使用权限，请先从 PUBLIC 撤销使用权限。然后，仅向允许创建存储过程的特定用户或组授予 plpgsql 使用权限。有关更多信息，请参阅 存储过程的安全性和权限 。

FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]

指定撤销其权限的 SQL 命令。您可以指定 ALL 以撤销对 COPY、UNLOAD、EXTERNAL FUNCTION 和 CREATE MODEL 语句的权限。此子句仅适用于撤销 ASSUMEROLE 权限。

ALTER

撤销用户或用户组的 ALTER 权限，允许那些不拥有数据共享的用户或用户组更改数据共享。将对象添加到数据共享中或从中删除，或设置属性 PUBLICACCESSIBLE 时需要此权限。有关更多信息，请参阅 ALTER DATASHARE。

SHARE

撤销用户和用户组将使用者添加到数据共享的权限。需要撤销此权限才能阻止特定使用者从其集群访问数据共享。

ON DATASHARE datashare_name

授予对被引用数据共享的指定权限。

FROM 用户名

指示失去权限的用户。

FROM GROUP group_name

指示失去权限的用户组。

WITH GRANT OPTION

指示失去权限的用户随之可以对其他用户撤销相同的权限。您无法为组或 PUBLIC 撤消 WITH GRANT OPTION。

USAGE

撤销同一账户中的使用者账户或命名空间的 USAGE 时，该账户中的特定使用者账户或命名空间不能以只读方式访问数据共享和数据共享的对象。

撤销 USAGE 权限将撤销使用者对数据共享的访问权限。

FROM NAMESPACE 'clusternamespace GUID'

指示同一账户中使用者失去对数据共享的权限的命名空间。命名空间使用 128 位字母数字全局唯一标识符 (GUID)。

FROM ACCOUNT 'accountnumber' [VIA DATA CATALOG]

指示另一个账户中使用者失去对数据共享的权限的账号。指定‘VIA DATA CATALOG’表示您要从 Lake Formation 账户撤销使用数据共享的权限。省略账号意味着您要撤销拥有集群的账户的权限。

ON DATABASE shared_database_name> [, ...]

撤销对在指定数据共享中创建的指定数据库的指定使用权限。

ON SCHEMA shared_schema

撤销对在指定数据共享中创建的指定架构的指定权限。

FOR { SCHEMAS | TABLES | FUNCTIONS | PROCEDURES | LANGUAGES } IN

指定要撤销权限的数据库对象。IN 后面的参数定义了所撤销权限的范围。

CREATE MODEL

撤销 CREATE MODEL 在指定数据库中创建机器学习模型的权限。

ON MODEL model_name

撤销特定模型的 EXECUTE 权限。

ACCESS CATALOG

撤销查看该角色可访问对象的相关元数据的权限。

[ADMIN OPTION FOR] { role } [, ...]

您从具有 WITH ADMIN OPTION 的指定用户撤销的角色。

FROM { role } [, ...]

从其撤消了指定角色的角色。

使用说明

要了解有关 REVOKE 使用说明的更多信息，请参阅the section called “使用说明”。

示例

有关如何使用 REVOKE 的示例，请参阅the section called “示例”。

使用说明

要撤消对象的权限，您必须满足下列条件之一：

- 是对象所有者。
- 是超级用户。
- 拥有该对象和权限的授予权限。

例如，以下命令使用户 HR 能够对 employees 表执行 SELECT 命令并对其他用户授予和撤销相同的权限。

```
grant select on table employees to HR with grant option;
```

HR 无法撤销 SELECT 之外的任何操作的权限或 employees 表之外的任何其他表的权限。

超级用户可以访问所有对象，不管设置对象权限的 GRANT 和 REVOKE 命令如何。

PUBLIC 表示一个始终包含所有用户的组。默认情况下，PUBLIC 的所有成员都对 PUBLIC schema 具有 CREATE 和 USAGE 权限。要限制任何用户对 PUBLIC schema 的权限，您必须首先从 PUBLIC 撤销对 PUBLIC schema 的所有权限，然后向特定用户或组授予权限。以下示例控制 PUBLIC schema 中的表创建权限。

```
revoke create on schema public from public;
```

要从 Lake Formation 表撤销权限，与表的外部架构关联的 IAM 角色必须有权撤销对外部表的权限。以下示例创建具有关联 IAM 角色 myGrantor 的外部架构。IAM 角色 myGrantor 有权撤销其他角色的权限。REVOKE 命令使用与外部架构关联的 IAM 角色 myGrantor 的权限来撤销对 IAM 角色 myGrantee 的权限。

```
create external schema mySchema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myGrantor'
create external database if not exists;
```

```
revoke select
on external table mySchema.mytable
from iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

> **Note**
>
> 如果 IAM 角色在为 Lake Formation 启用的 AWS Glue Data Catalog 中也有 ALL 权限，则不会撤销 ALL 权限。只撤销 SELECT 权限。您可以在 Lake Formation 控制台中查看 Lake Formation 权限。

有关撤销 ASSUMEROLE 权限的使用说明

以下使用说明适用于在 Amazon Redshift 中撤销 ASSUMEROLE 权限。

只有数据库超级用户才可以撤消用户和组的 ASSUMEROLE 权限。超级用户始终会保留 ASSUMEROLE 权限。

要为用户和组启用 ASSUMEROLE 的使用权限，超级用户要在集群上运行一次以下语句。要为用户和组启用 ASSUMEROLE 的使用权限，超级用户要在集群上将以下语句运行一次。

```
revoke assumerole on all from public for all;
```

有关撤销机器学习权限的使用说明

您不能直接授予或撤销与机器学习函数相关的权限。机器学习函数属于机器学习模型，其权限通过模型来控制。相反，您可以撤销与机器学习模型相关的权限。以下示例演示如何从与模型 customer_churn 关联的所有用户撤销运行权限。

```
REVOKE EXECUTE ON MODEL customer_churn FROM PUBLIC;
```

您还可以撤销某个用户对机器学习模型 customer_churn 的所有权限。

```
REVOKE ALL on MODEL customer_churn FROM ml_user;
```

如果架构中有机器学习函数，则授予或撤销与机器学习函数相关的 EXECUTE 权限将失败，即使该机器学习函数已通过 GRANT EXECUTE ON MODEL 获得 EXECUTE 权限。我们建议在使用 CREATE MODEL 命令时，通过单独的架构将机器学习函数单独保存在单独架构本身中。以下示例演示了如何执行此操作。

```
CREATE MODEL ml_schema.customer_churn
FROM customer_data
TARGET churn
FUNCTION ml_schema.customer_churn_prediction
IAM_ROLE default
SETTINGS (
 S3_BUCKET 'your-s3-bucket'
);
```

示例

以下示例从 GUESTS 用户组撤消对 SALES 表的 INSERT 权限。此命令使 GUESTS 的成员无法通过使用 INSERT 命令将数据加载到 SALES 表中。

```
revoke insert on table sales from group guests;
```

以下示例从用户 fred 撤消对 QA_TICKIT 架构中所有表的 SELECT 权限。

```
revoke select on all tables in schema qa_tickit from fred;
```

以下示例撤消用户 bobr 从视图中选择的权限。

```
revoke select on table eventview from bobr;
```

以下示例从所有用户撤消在 TICKIT 数据库中创建临时表的权限。

```
revoke temporary on database tickit from public;
```

以下示例从用户 cust_name 撤消对 cust_phone 表的 cust_profile 和 user1 列的 SELECT 权限。

```
revoke select(cust_name, cust_phone) on cust_profile from user1;
```

以下示例从 cust_name 组中撤消对 cust_phone 和 cust_contact_preference 列的 SELECT 权限，并撤消对 cust_profile 表的 sales_group 列的 UPDATE 权限。

```
revoke select(cust_name, cust_phone), update(cust_contact_preference) on cust_profile
 from group sales_group;
```

下面的示例演示如何使用 ALL 关键字从 cust_profile 组撤消对 sales_admin 表的三列的 SELECT 和 UPDATE 权限。

```
revoke ALL(cust_name, cust_phone,cust_contact_preference) on cust_profile from group
 sales_admin;
```

以下示例从 cust_name 用户撤消对 cust_profile_vw 视图的 user2 列的 SELECT 权限。

```
revoke select(cust_name) on cust_profile_vw from user2;
```

撤销通过数据共享创建的数据库的 USAGE 权限的示例

以下示例从 13b8833d-17c6-4f16-8fe4-1a018f5ed00d 命名空间撤销对 salesshare 数据共享的访问权限。

```
REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

以下示例从 Bob 撤销对 sales_db 的 USAGE 权限。

```
REVOKE USAGE ON DATABASE sales_db FROM Bob;
```

以下示例从 Analyst_role 撤销对 sales_schema 的 USAGE 权限。

```
REVOKE USAGE ON SCHEMA sales_schema FROM ROLE Analyst_role;
```

撤销限定范围权限的示例

以下示例从 Sales 角色撤销 Sales_db 数据库中所有当前和将来架构的使用权限。

```
REVOKE USAGE FOR SCHEMAS IN DATABASE Sales_db FROM ROLE Sales;
```

以下示例从用户 alice 撤销授予对 Sales_db 数据库中所有当前和将来表的 SELECT 权限的能力。alice 保留对 Sales_db 中所有表的访问权限。

```
REVOKE GRANT OPTION SELECT FOR TABLES IN DATABASE Sales_db FROM alice;
```

以下示例从用户 bob 撤销对 Sales_schema 架构中函数的 EXECUTE 权限。

```
REVOKE EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema FROM bob;
```

以下示例从 Sales 角色撤销对 ShareDb 数据库 ShareSchema 架构中所有表的所有权限。指定架构时，您还可以使用由两部分组成的格式 database.schema 指定架构的数据库。

```
REVOKE ALL FOR TABLES IN SCHEMA ShareDb.ShareSchema FROM ROLE Sales;
```

下面的示例与前一个示例相同。您可以使用 DATABASE 关键字而不是使用由两部分组成的格式来指定架构的数据库。

```
REVOKE ALL FOR TABLES IN SCHEMA ShareSchema DATABASE ShareDb FROM ROLE Sales;
```

撤销 ASSUMEROLE 权限的示例

以下是撤销 ASSUMEROLE 权限的示例。

超级用户必须通过在集群上运行一次以下语句来为用户和组启用 ASSUMEROLE 的使用权限。

```
revoke assumerole on all from public for all;
```

以下语句撤消用户 reg_user1 在所有角色中对所有操作的 ASSUMEROLE 权限。

```
revoke assumerole on all from reg_user1 for all;
```

撤销 ROLE 权限的示例

下面的示例将从 sample_role2 撤销 sample_role1。

```
CREATE ROLE sample_role2;
GRANT ROLE sample_role1 TO ROLE sample_role2;
REVOKE ROLE sample_role1 FROM ROLE sample_role2;
```

下面的示例将撤销 user1 的系统权限。

```
GRANT ROLE sys:DBA TO user1;
REVOKE ROLE sys:DBA FROM user1;
```

下面的示例将从 user1 撤销 sample_role1 和 sample_role2。

```
CREATE ROLE sample_role1;
CREATE ROLE sample_role2;
GRANT ROLE sample_role1, ROLE sample_role2 TO user1;
REVOKE ROLE sample_role1, ROLE sample_role2 FROM user1;
```

下面的示例将从 user1 撤销具有 ADMIN OPTION 的 sample_role2。

```
GRANT ROLE sample_role2 TO user1 WITH ADMIN OPTION;
REVOKE ADMIN OPTION FOR ROLE sample_role2 FROM user1;
REVOKE ROLE sample_role2 FROM user1;
```

下面的示例将从 sample_role5 撤销 sample_role1 和 sample_role2。

```
CREATE ROLE sample_role5;
GRANT ROLE sample_role1, ROLE sample_role2 TO ROLE sample_role5;
REVOKE ROLE sample_role1, ROLE sample_role2 FROM ROLE sample_role5;
```

下面的示例将撤销 sample_role1 的 CREATE SCHEMA 和 DROP SCHEMA 系统权限。

```
GRANT CREATE SCHEMA, DROP SCHEMA TO ROLE sample_role1;
```

```
REVOKE CREATE SCHEMA, DROP SCHEMA FROM ROLE sample_role1;
```

ROLLBACK

停止当前事务，并放弃该事务执行的所有更新。

此命令执行与 ABORT 命令相同的功能。

语法

```
ROLLBACK [ WORK | TRANSACTION ]
```

参数

WORK

可选关键字。存储过程中不支持此关键字。

TRANSACTION

可选关键字。WORK 和 TRANSACTION 是同义词。两者在存储过程中均不受支持。

有关在存储过程中使用 ROLLBACK 的信息，请参阅管理事务。

示例

以下示例创建一个表，然后启动将数据插入到表的事务。然后，ROLLBACK 命令将回滚数据插入操作，以便将表保持为空表。

以下命令创建一个名为 MOVIE_GROSS 的示例表：

```
create table movie_gross( name varchar(30), gross bigint );
```

下一组命令启动将两个数据行插入到表中的事务：

```
begin;

insert into movie_gross values ( 'Raiders of the Lost Ark', 23400000);

insert into movie_gross values ( 'Star Wars', 10000000 );
```

接下来，以下命令从表中选择数据，表明已插入成功：

```
select * from movie_gross;
```

命令输出表明已成功插入两行：

```
name           |  gross
-------------------------+----------
Raiders of the Lost Ark | 23400000
Star Wars               | 10000000
(2 rows)
```

现在，此命令将数据更改回滚到事务开始的位置：

```
rollback;
```

现在，从表中选择数据时，显示这是一个空表：

```
select * from movie_gross;

name | gross
------+-------
(0 rows)
```

SELECT

返回表、视图和用户定义的函数中的行。

> **Note**
>
> 单个 SQL 语句的最大大小为 16MB。

语法

```
[ WITH with_subquery [, ...] ]
SELECT
[ TOP number | [ ALL | DISTINCT ]
* | expression [ AS output_name ] [, ...] ]
[ FROM table_reference [, ...] ]
```

```
[ WHERE condition ]
[ [ START WITH expression ] CONNECT BY expression ]
[ GROUP BY expression [, ...] ]
[ HAVING condition ]
[ QUALIFY condition ]
[ { UNION | ALL | INTERSECT | EXCEPT | MINUS } query ]
[ ORDER BY expression [ ASC | DESC ] ]
[ LIMIT { number | ALL } ]
[ OFFSET start ]
```

主题

WITH 子句

WITH 子句是一个可选子句，该子句在查询中位于 SELECT 列表之前。WITH 子句定义一个或多个 common_table_expressions。每个通用表表达式 (CTE) 均定义一个临时表，它与视图定义类似。您可以在 FROM 子句中引用这些临时表。它们仅在它们所属的查询运行时使用。WITH 子句中的每个子 CTE 均指定一个表名、一个可选的列名称列表以及一个计算结果为表的查询表达式（SELECT 语句）。当您在定义临时表的同一查询表达式的 FROM 子句中引用临时表名时，CTE 是递归的。

WITH 子句子查询是定义可在单个查询的执行过程中使用的表的有效方式。在所有情况下，在 SELECT 语句的主体中使用子查询可获得相同的结果，不过 WITH 子句子查询可能在编写和阅读方面更加简单。如果可能，会将已引用多次的 WITH 子句子查询优化为常用子表达式；即，可以计算 WITH 子查询一次并重用其结果。（请注意，常用子表达式不只是限于 WITH 子句中定义的子表达式。）

语法

```
[ WITH [RECURSIVE] common_table_expression [, common_table_expression , ...] ]
```

其中 common_table_expression 可以是非递归的，也可以是递归的。以下是非递归形式：

```
CTE_table_name [ ( column_name [, ...] ) ] AS ( query )
```

以下是 common_table_expression 的递归形式：

```
CTE_table_name (column_name [, ...] ) AS ( recursive_query )
```

参数

RECURSIVE

将查询标识为递归 CTE 的关键词。如果 WITH 子句中定义的任何 common_table_expression 是递归的，则需要此关键词。即使 WITH 子句包含多个递归 CTE，您也只能紧跟 WITH 关键词之后指定一次 RECURSIVE 关键词。通常，递归 CTE 是一个包含两个部分的 UNION ALL 子查询。

common_table_expression

定义一个您可以在 FROM 子句 中引用并且仅在执行其所属的查询期间使用的临时表。

CTE_table_name

临时表的唯一名称，该临时表用于定义 WITH 子句子查询的结果。不能在单个 WITH 子句中使用重复名称。必须为每个子查询提供一个可在 FROM 子句中引用的表名。

column_name

WITH 子句子查询的输出列名称的列表（用逗号分隔）。指定的列名数目必须等于或小于子查询定义的列数。对于非递归的 CTE，column_name 子句是可选的。对于递归的 CTE，column_name 列表是必需的。

query

Amazon Redshift 支持的任何 SELECT 查询。请参阅 SELECT。

recursive_query

由两个 SELECT 子查询组成的 UNION ALL 查询：

- 第一个 SELECT 子查询没有对同一个 CTE_table_name 进行递归引用。它返回一个结果集，该结果集是递归的初始种子。此部分称为初始成员或种子成员。

- 第二个 SELECT 子查询在其 FROM 子句中引用同一个 CTE_table_name。它被称为递归成员。recursive_query 包含一个 WHERE 条件来结束 recursive_query。

使用说明

可在以下 SQL 语句中使用 WITH 子句：

- SELECT
- SELECT INTO
- CREATE TABLE AS
- CREATE VIEW
- DECLARE
- EXPLAIN
- INSERT INTO...SELECT
- PREPARE
- UPDATE (在 WHERE 子句子查询中。您无法在子查询中定义递归 CTE。递归 CTE 必须位于 UPDATE 子句之前。)
- DELETE

如果包含 WITH 子句的查询的 FROM 子句未引用 WITH 子句所定义的任何表，则将忽略 WITH 子句，并且查询将正常执行。

WITH 子句子查询所定义的表只能在 WITH 子句开始的 SELECT 查询范围内引用。例如，可以在 SELECT 列表的子查询的 FROM 子句、WHERE 子句或 HAVING 子句中引用这样的表。不能在子查询中使用 WITH 子句，也不能在主查询或其他子查询的 FROM 子句中引用其表。此查询模式会为 WITH 子句表生成 `relation table_name doesn't exist` 形式的错误消息。

不能在 WITH 子句子查询中指定另一个 WITH 子句。

不能对 WITH 子句子查询定义的表进行前向引用。例如，以下查询返回一个错误，因为在表 W1 的定义中对表 W2 进行了前向引用：

```
with w1 as (select * from w2), w2 as (select * from w1)
select * from sales;
ERROR:  relation "w2" does not exist
```

WITH 子句子查询不能包含 SELECT INTO 语句；不过，您可以在 SELECT INTO 语句中使用 WITH 子句。

递归公用表表达式

递归公用表表达式 (CTE) 是一个引用自身的 CTE。递归 CTE 在查询分层数据（如显示员工和经理之间的报告关系的组织结构图）时非常有用。请参阅 示例：递归 CTE。

另一个常见用途是多级物料清单，当产品由多个组件组成并且每个组件本身也包含其它组件或子装配件时。

通过在递归查询的第二个 SELECT 子查询中包含 WHERE 子句来确保限制递归的深度。有关示例，请参阅 示例：递归 CTE。否则，可能会出现类似于以下内容的错误：

- `Recursive CTE out of working buffers.`
- `Exceeded recursive CTE max rows limit, please add correct CTE termination predicates or change the max_recursion_rows parameter.`

> **Note**
>
> `max_recursion_rows` 参数用于设置递归 CTE 可以返回的最大行数，以防止无限递归循环。我们建议更改该值时不要超过默认值。这样可以防止查询中的无限递归问题占用集群的过多空间。

您可以指定递归 CTE 结果的排序顺序和限制。您可以在递归 CTE 的最终结果中包含分组依据和不同选项。

不能在子查询中指定 WITH RECURSIVE 子句。recursive_query 成员不能包含排序依据或限制子句。

示例

以下示例说明了包含 WITH 语句的查询的最简单示例。在名为 VENUECOPY 的 WITH 查询中，选择 VENUE 表中的所有行。主查询又选择 VENUECOPY 中的所有行。VENUECOPY 表仅在此查询的持续时间内存在。

```
with venuecopy as (select * from venue)
select * from venuecopy order by 1 limit 10;
```

```
 venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
1 | Toyota Park                | Bridgeview      | IL         |          0
2 | Columbus Crew Stadium      | Columbus        | OH         |          0
3 | RFK Stadium                | Washington      | DC         |          0
4 | CommunityAmerica Ballpark  | Kansas City     | KS         |          0
5 | Gillette Stadium           | Foxborough      | MA         |      68756
6 | New York Giants Stadium    | East Rutherford | NJ         |      80242
7 | BMO Field                  | Toronto         | ON         |          0
8 | The Home Depot Center      | Carson          | CA         |          0
9 | Dick's Sporting Goods Park | Commerce City   | CO         |          0
v     10 | Pizza Hut Park             | Frisco          | TX         |          0
(10 rows)
```

以下示例显示一个 WITH 子句，该子句生成两个分别名为 VENUE_SALES 和 TOP_VENUES 的表。第二个 WITH 查询表从第一个表中进行选择。而主查询块的 WHERE 子句又包含一个约束 TOP_VENUES 表的子查询。

```
with venue_sales as
(select venuename, venuecity, sum(pricepaid) as venuename_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
group by venuename, venuecity),

top_venues as
(select venuename
from venue_sales
where venuename_sales > 800000)

select venuename, venuecity, venuestate,
sum(qtysold) as venue_qty,
sum(pricepaid) as venue_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
and venuename in(select venuename from top_venues)
group by venuename, venuecity, venuestate
order by venuename;
```

```
        venuename       |   venuecity   | venuestate | venue_qty | venue_sales
------------------------+---------------+------------+-----------+-------------
August Wilson Theatre   | New York City | NY         |      3187 |  1032156.00
Biltmore Theatre        | New York City | NY         |      2629 |   828981.00
```

```
Charles Playhouse       | Boston        | MA         |      2502 |   857031.00
Ethel Barrymore Theatre | New York City | NY         |      2828 |   891172.00
Eugene O'Neill Theatre  | New York City | NY         |      2488 |   828950.00
Greek Theatre           | Los Angeles   | CA         |      2445 |   838918.00
Helen Hayes Theatre     | New York City | NY         |      2948 |   978765.00
Hilton Theatre          | New York City | NY         |      2999 |   885686.00
Imperial Theatre        | New York City | NY         |      2702 |   877993.00
Lunt-Fontanne Theatre   | New York City | NY         |      3326 |  1115182.00
Majestic Theatre        | New York City | NY         |      2549 |   894275.00
Nederlander Theatre     | New York City | NY         |      2934 |   936312.00
Pasadena Playhouse      | Pasadena      | CA         |      2739 |   820435.00
Winter Garden Theatre   | New York City | NY         |      2838 |   939257.00
(14 rows)
```

以下两个示例演示基于 WITH 子句子查询的表引用范围的规则。第一个查询运行，但第二个查询失败，并出现意料中的错误。在第一个查询中，主查询的 SELECT 列表中包含 WITH 子句子查询。SELECT 列表中的子查询的 FROM 子句中将引用 WITH 子句定义的表 (HOLIDAYS)：

```
select caldate, sum(pricepaid) as daysales,
(with holidays as (select * from date where holiday ='t')
select sum(pricepaid)
from sales join holidays on sales.dateid=holidays.dateid
where caldate='2008-12-25') as dec25sales
from sales join date on sales.dateid=date.dateid
where caldate in('2008-12-25','2008-12-31')
group by caldate
order by caldate;

caldate   | daysales | dec25sales
----------+----------+------------
2008-12-25 | 70402.00 |   70402.00
2008-12-31 | 12678.00 |   70402.00
(2 rows)
```

第二个查询失败，因为它尝试在主查询和 SELECT 列表子查询中引用 HOLIDAYS 表。主查询引用超出范围。

```
select caldate, sum(pricepaid) as daysales,
(with holidays as (select * from date where holiday ='t')
select sum(pricepaid)
from sales join holidays on sales.dateid=holidays.dateid
where caldate='2008-12-25') as dec25sales
```

```
from sales join holidays on sales.dateid=holidays.dateid
where caldate in('2008-12-25','2008-12-31')
group by caldate
order by caldate;

ERROR:  relation "holidays" does not exist
```

示例：递归 CTE

以下是递归 CTE 的示例，该示例返回直接或间接向 John 报告的员工。递归查询包含一个 WHERE 子句，用于将递归深度限制为小于 4 个级别。

```
--create and populate the sample table
  create table employee (
  id int,
  name varchar (20),
  manager_id int
  );

  insert into employee(id, name, manager_id)  values
(100, 'Carlos', null),
(101, 'John', 100),
(102, 'Jorge', 101),
(103, 'Kwaku', 101),
(110, 'Liu', 101),
(106, 'Mateo', 102),
(110, 'Nikki', 103),
(104, 'Paulo', 103),
(105, 'Richard', 103),
(120, 'Saanvi', 104),
(200, 'Shirley', 104),
(201, 'Sofía', 102),
(205, 'Zhang', 104);

--run the recursive query
  with recursive john_org(id, name, manager_id, level) as
( select id, name, manager_id, 1 as level
  from employee
  where name = 'John'
  union all
  select e.id, e.name, e.manager_id, level + 1 as next_level
  from employee e, john_org j
  where e.manager_id = j.id and level < 4
```

```
 )
select distinct id, name, manager_id from john_org order by manager_id;
```

以下是查询的结果。

```
  id        name      manager_id
------+-----------+--------------
 101    John           100
 102    Jorge          101
 103    Kwaku          101
 110    Liu            101
 201    Sofía          102
 106    Mateo          102
 110    Nikki          103
 104    Paulo          103
 105    Richard        103
 120    Saanvi         104
 200    Shirley        104
 205    Zhang          104
```

以下是 John 所在部门的组织结构图。

SELECT 列表

主题

- 语法
- 参数
- 使用说明
- 示例

SELECT 列表指定希望查询返回的列、函数和表达式。列表表示查询的输出。

有关 SQL 函数的更多信息，请参阅 SQL 函数参考。有关表达式的更多信息，请参阅条件表达式。

语法

```
SELECT
[ TOP number ]
```

```
[ ALL | DISTINCT ] * | expression [ AS column_alias ] [, ...]
```

参数

TOP number

TOP 将正整数用作其参数，用于定义返回到客户端的行数。使用 TOP 子句的行为与使用 LIMIT 子句的行为相同。返回的行数是固定的，但行集不固定。要返回一致的行集，请将 TOP 或 LIMIT 与 ORDER BY 子句结合使用。

ALL

一个冗余关键字，定义未指定 DISTINCT 的情况下的默认行为。SELECT ALL * 与 SELECT * 的含义相同（选择所有列的所有行并保留重复条目）。

DISTINCT

一个选项，用于根据一个或多个列中的匹配值消除结果集中的重复行。

> **Note**
>
> 如果您的应用程序允许无效的外键或主键，则会导致查询返回错误的结果。例如，如果主键列不包含所有唯一值，则 SELECT DISTINCT 查询可能会返回重复的行。有关更多信息，请参阅定义表约束。

* （星号）

返回表的完整内容（所有列和所有行）。

expression

由查询引用的表中存在的一个或多个列构成的表达式。表达式可包含 SQL 函数。例如：

```
avg(datediff(day, listtime, saletime))
```

AS column_alias

在最终结果集中使用的列的临时名称。AS 关键字是可选的。例如：

```
avg(datediff(day, listtime, saletime)) as avgwait
```

如果您没有为不是简单列名的表达式指定别名，则结果集将对该列应用默认名称。

> **Note**
>
> 在目标列表中定义别名后，它将立即被识别。您可以在其他表达式中使用在同一目标列表中晚于该别名定义的某个别名。以下示例对此进行了说明。
>
> ```
> select clicks / impressions as probability, round(100 * probability, 1) as
>  percentage from raw_data;
> ```
>
> 横向别名引用的好处是，当在同一目标列表中构建更复杂的表达式时，不需要重复已指定别名的表达式。当 Amazon Redshift 分析这种类型的引用时，它只会内联之前定义的别名。如果在 FROM 子句中定义了一个与之前指定了别名的表达式同名的列，FROM 子句中定义的列将优先。例如，在上述查询中，如果表 raw_data 中有一个名为“probability”的列，那么目标列表中的第二个表达式中的“probability”引用该列而不是别名“probability”。

使用说明

TOP 是一个 SQL 扩展；它提供 LIMIT 行为的替代。不能在同一个查询中使用 TOP 和 LIMIT。

示例

以下示例从 SALES 表中返回 10 行。尽管查询使用 TOP 子句，但它仍然返回一组不可预测的行，因为没有指定 ORDERBY 子句，

```
select top 10 *
from sales;
```

以下查询具有相同的功能，但使用的是 LIMIT 子句而非 TOP 子句：

```
select *
from sales
limit 10;
```

以下示例使用 TOP 子句返回 SALES 表的前 10 行，按 QTYSOLD 列降序排序。

```
select top 10 qtysold, sellerid
from sales
order by qtysold desc, sellerid;
```

```
qtysold | sellerid
--------+----------
8 |      518
8 |      520
8 |      574
8 |      718
8 |      868
8 |     2663
8 |     3396
8 |     3726
8 |     5250
8 |     6216
(10 rows)
```

以下示例返回 SALES 表中的前两个 QTYSOLD 和 SELLERID 值（按 QTYSOLD 列排序）：

```
select top 2 qtysold, sellerid
from sales
order by qtysold desc, sellerid;

qtysold | sellerid
--------+----------
8 |      518
8 |      520
(2 rows)
```

以下示例显示 CATEGORY 表中不同类别组的列表：

```
select distinct catgroup from category
order by 1;

catgroup
----------
Concerts
Shows
Sports
(3 rows)

--the same query, run without distinct
select catgroup from category
order by 1;
```

```
catgroup
----------
Concerts
Concerts
Concerts
Shows
Shows
Shows
Sports
Sports
Sports
Sports
Sports
(11 rows)
```

以下示例返回 2008 年 12 月的一组不同的周编号。如果没有 DISTINCT 子句，该语句将返回 31 行，或对于月份的每一天返回 1 行。

```
select distinct week, month, year
from date
where month='DEC' and year=2008
order by 1, 2, 3;

week | month | year
-----+-------+------
49 | DEC   | 2008
50 | DEC   | 2008
51 | DEC   | 2008
52 | DEC   | 2008
53 | DEC   | 2008
(5 rows)
```

FROM 子句

查询中的 FROM 子句列出从中选择数据的表引用（表、视图和子查询）。如果列出多个表引用，则必须在 FROM 子句或 WHERE 子句中使用适当的语法来联接表。如果未指定联接条件，则系统将查询作为交叉联接（笛卡尔乘积）进行处理。

主题

- 语法
- 参数

- 使用说明
- PIVOT 和 UNPIVOT 示例
- JOIN 示例

语法

```
FROM table_reference [, ...]
```

其中，table_reference 是下列项之一：

```
with_subquery_table_name [ table_alias ]
table_name [ * ] [ table_alias ]
( subquery ) [ table_alias ]
table_reference [ NATURAL ] join_type table_reference
   [ ON join_condition | USING ( join_column [, ...] ) ]
table_reference PIVOT (
   aggregate(expr) [ [ AS ] aggregate_alias ]
   FOR column_name IN ( expression [ AS ] in_alias [, ...] )
) [ table_alias ]
table_reference UNPIVOT [ INCLUDE NULLS | EXCLUDE NULLS ] (
   value_column_name
   FOR name_column_name IN ( column_reference [ [ AS ]
   in_alias ] [, ...] )
) [ table_alias ]
UNPIVOT expression AS value_alias [ AT attribute_alias ]
```

可选的 table_alias 可用于为表和复杂表引用指定临时名称，如果需要，也可以为其列指定临时名称，如下所示：

```
[ AS ] alias [ ( column_alias [, ...] ) ]
```

参数

with_subquery_table_name

WITH 子句中的子查询定义的表。

table_name

表或视图的名称。

alias

表或视图的临时备用名称。必须为派生自子查询的表提供别名。在其他表引用中，别名是可选的。AS 关键字始终是可选的。表别名提供了用于标识查询的其他部分（例如 WHERE 子句）中的表的快捷方法。例如：

```
select * from sales s, listing l
where s.listid=l.listid
```

column_alias

表或视图中的列的临时备用名称。

subquery

一个计算结果为表的查询表达式。表仅在查询的持续时间内存在，并且通常会向表提供一个名称或别名。但别名不是必需的。您也可以为派生自子查询的表定义列名称。如果您希望将子查询的结果联接到其他表并且希望在查询中的其他位置选择或约束这些列，则指定列的别名是非常重要的。

子查询可以包含 ORDER BY 子句，但在未指定 LIMIT 或 OFFSET 子句的情况下，该子句可能没有任何作用。

NATURAL

定义一个联接，该联接自动将两个表中同名列的所有配对用作联接列。不需要显式联接条件。例如，如果 CATEGORY 和 EVENT 表都具有名为 CATID 的列，则这两个表的自然联接为基于其 CATID 列的联接。

> **Note**
>
> 如果指定 NATURAL 联接，但表中没有要联接的同名列配对，则查询默认为交叉联接。

join_type

指定下列类型的联接之一：

- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN

- CROSS JOIN

交叉联接是未限定的联接；它们返回两个表的笛卡尔乘积。

内部联接和外部联接是限定的联接。它们的限定方式包括：隐式（在自然联接中）；在 FROM 语句中使用 ON 或 USING 语法；或者使用 WHERE 子句条件。

内部联接仅基于联接条件或联接列的列表返回匹配的行。外部联接返回与内部联接相同的所有行，还返回“左侧”表和/或“右侧”表中的非匹配行。左侧表是第一个列出的表，右侧表是第二个列出的表。非匹配行包含 NULL 值以填补输出列中的空白。

ON join_condition

联接规范的类型，其中将联接列声明为紧跟 ON 关键字的条件。例如：

```
sales join listing
on sales.listid=listing.listid and sales.eventid=listing.eventid
```

USING (join_column [, ...])

联接规范的类型，其中用圆括号将列出的联接列括起来。如果指定多个联接列，则用逗号将它们分隔开。USING 关键字必须在列表之前。例如：

```
sales join listing
using (listid,eventid)
```

PIVOT

将输出由行转换为列，以便以易于阅读的格式表示表格数据。在多个列中水平表示输出。PIVOT 类似于带有聚合的 GROUP BY 查询，它使用聚合表达式来指定输出格式。但是，与 GROUP BY 不同的是，它以列而非行的形式返回输出。

有关演示如何使用 PIVOT 和 UNPIVOT 查询的示例，请参阅 PIVOT 和 UNPIVOT 示例。

UNPIVOT

使用 UNPIVOT 将列旋转为行 – 此运算符将输入表或查询结果中的结果列转换为行，以使输出更易于阅读。UNPIVOT 将其输入列的数据合并为两个结果列：名称列和值列。名称列包含输入中的列名称，例如行条目。值列包含输入列中的值，例如聚合结果。例如，不同类别中的项目计数。

使用 UNPIVOT (SUPER) 反转置对象 - 您可以执行对象反转置，其中 expression 是引用另一个 FROM 子句项的 SUPER 表达式。有关更多信息，请参阅 对象逆透视。它还提供了一些示例，展示如何查询半结构化数据，例如 JSON 格式的数据。

使用说明

联接列必须具有可比较的数据类型。

NATURAL 或 USING 联接仅将每对联接列中的一个联接列保留在中间结果集中。

使用 ON 语法的联接会将两个联接列都保留在其中间结果集中。

另请参阅 WITH 子句。

PIVOT 和 UNPIVOT 示例

PIVOT 和 UNPIVOT 是 FROM 子句中的参数，它们分别将查询输出从行轮换到列，以及从列轮换到行。它们以便于阅读的格式呈现表格查询结果。以下示例使用测试数据和查询来说明如何使用它们。

有关这些参数及其他参数的更多信息，请参阅 FROM 子句。

PIVOT 示例

设置示例表和数据并使用它们运行后续示例查询。

```
CREATE TABLE part (
    partname varchar,
    manufacturer varchar,
    quality int,
    price decimal(12, 2)
);

INSERT INTO part VALUES ('prop', 'local parts co', 2, 10.00);
INSERT INTO part VALUES ('prop', 'big parts co', NULL, 9.00);
INSERT INTO part VALUES ('prop', 'small parts co', 1, 12.00);

INSERT INTO part VALUES ('rudder', 'local parts co', 1, 2.50);
INSERT INTO part VALUES ('rudder', 'big parts co', 2, 3.75);
INSERT INTO part VALUES ('rudder', 'small parts co', NULL, 1.90);

INSERT INTO part VALUES ('wing', 'local parts co', NULL, 7.50);
INSERT INTO part VALUES ('wing', 'big parts co', 1, 15.20);
INSERT INTO part VALUES ('wing', 'small parts co', NULL, 11.80);
```

`partname` 上的 PIVOT，在 `price` 上有一个 `AVG` 聚合。

```
SELECT *
```

```
FROM (SELECT partname, price FROM part) PIVOT (
    AVG(price) FOR partname IN ('prop', 'rudder', 'wing')
);
```

查询将生成以下输出。

```
  prop   |  rudder  |  wing
---------+----------+---------
 10.33   | 2.71     | 11.50
```

在前面的示例中，结果转换为列。以下示例显示了一个按行而不是按列返回平均价格的 GROUP BY 查询。

```
SELECT partname, avg(price)
FROM (SELECT partname, price FROM part)
WHERE partname IN ('prop', 'rudder', 'wing')
GROUP BY partname;
```

查询将生成以下输出。

```
 partname |  avg
----------+-------
 prop     | 10.33
 rudder   |  2.71
 wing     | 11.50
```

一个 PIVOT 示例，将 manufacturer 作为隐式列。

```
SELECT *
FROM (SELECT quality, manufacturer FROM part) PIVOT (
    count(*) FOR quality IN (1, 2, NULL)
);
```

查询将生成以下输出。

```
 manufacturer      | 1  | 2  | null
-------------------+----+----+------
 local parts co    | 1  | 1  |  1
 big parts co      | 1  | 1  |  1
 small parts co    | 1  | 0  |  2
```

PIVOT 定义中未引用的输入表列被隐式添加到结果表中。就是上一个示例中 manufacturer 列这种情况。示例还显示，对于 IN 运算符，NULL 为有效值。

上述示例中的 PIVOT 返回与以下查询类似的信息，其中包含 GROUP BY。区别在于 PIVOT 为列 2 和制造商 small parts co 返回值 0。GROUP BY 查询不包含相应的行。在大多数情况下，如果一行没有针对给定列的输入数据，则 PIVOT 会插入 NULL。但是，计数聚合不会返回 NULL，0 是默认值。

```
SELECT manufacturer, quality, count(*)
FROM (SELECT quality, manufacturer FROM part)
WHERE quality IN (1, 2) OR quality IS NULL
GROUP BY manufacturer, quality
ORDER BY manufacturer;
```

查询将生成以下输出。

```
 manufacturer        | quality | count
---------------------+---------+-------
 big parts co        |         |     1
 big parts co        |       2 |     1
 big parts co        |       1 |     1
 local parts co      |       2 |     1
 local parts co      |       1 |     1
 local parts co      |         |     1
 small parts co      |       1 |     1
 small parts co      |         |     2
```

PIVOT 运算符接受聚合表达式和 IN 运算符的每个值上的可选别名。使用别名自定义列名。如果没有聚合别名，则仅使用 IN 列表别名。否则，将聚合别名附加到列名，并使用下划线将其与列名分开。

```
SELECT *
FROM (SELECT quality, manufacturer FROM part) PIVOT (
    count(*) AS count FOR quality IN (1 AS high, 2 AS low, NULL AS na)
);
```

查询将生成以下输出。

```
 manufacturer      | high_count  | low_count | na_count
-------------------+-------------+-----------+----------
 local parts co    |           1 |         1 |        1
 big parts co      |           1 |         1 |        1
 small parts co    |           1 |         0 |        2
```

设置以下示例表和数据，并使用它们运行后续示例查询。该数据表示一系列酒店的预订日期。

```
CREATE TABLE bookings (
    booking_id int,
    hotel_code char(8),
    booking_date date,
    price decimal(12, 2)
);

INSERT INTO bookings VALUES (1, 'FOREST_L', '02/01/2023', 75.12);
INSERT INTO bookings VALUES (2, 'FOREST_L', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (3, 'FOREST_L', '02/04/2023', 85.54);

INSERT INTO bookings VALUES (4, 'FOREST_L', '02/08/2023', 75.00);
INSERT INTO bookings VALUES (5, 'FOREST_L', '02/11/2023', 75.00);
INSERT INTO bookings VALUES (6, 'FOREST_L', '02/14/2023', 90.00);

INSERT INTO bookings VALUES (7, 'FOREST_L', '02/21/2023', 60.00);
INSERT INTO bookings VALUES (8, 'FOREST_L', '02/22/2023', 85.00);
INSERT INTO bookings VALUES (9, 'FOREST_L', '02/27/2023', 90.00);

INSERT INTO bookings VALUES (10, 'DESERT_S', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (11, 'DESERT_S', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (12, 'DESERT_S', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (13, 'DESERT_S', '02/05/2023', 75.00);
INSERT INTO bookings VALUES (14, 'DESERT_S', '02/06/2023', 34.00);
INSERT INTO bookings VALUES (15, 'DESERT_S', '02/09/2023', 85.00);

INSERT INTO bookings VALUES (16, 'DESERT_S', '02/12/2023', 23.00);
INSERT INTO bookings VALUES (17, 'DESERT_S', '02/13/2023', 76.00);
INSERT INTO bookings VALUES (18, 'DESERT_S', '02/14/2023', 85.00);

INSERT INTO bookings VALUES (19, 'OCEAN_WV', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (20, 'OCEAN_WV', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (21, 'OCEAN_WV', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (22, 'OCEAN_WV', '02/06/2023', 75.00);
INSERT INTO bookings VALUES (23, 'OCEAN_WV', '02/09/2023', 34.00);
INSERT INTO bookings VALUES (24, 'OCEAN_WV', '02/12/2023', 85.00);

INSERT INTO bookings VALUES (25, 'OCEAN_WV', '02/13/2023', 23.00);
INSERT INTO bookings VALUES (26, 'OCEAN_WV', '02/14/2023', 76.00);
INSERT INTO bookings VALUES (27, 'OCEAN_WV', '02/16/2023', 85.00);
```

```
INSERT INTO bookings VALUES (28, 'CITY_BLD', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (29, 'CITY_BLD', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (30, 'CITY_BLD', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (31, 'CITY_BLD', '02/12/2023', 75.00);
INSERT INTO bookings VALUES (32, 'CITY_BLD', '02/13/2023', 34.00);
INSERT INTO bookings VALUES (33, 'CITY_BLD', '02/17/2023', 85.00);

INSERT INTO bookings VALUES (34, 'CITY_BLD', '02/22/2023', 23.00);
INSERT INTO bookings VALUES (35, 'CITY_BLD', '02/23/2023', 76.00);
INSERT INTO bookings VALUES (36, 'CITY_BLD', '02/24/2023', 85.00);
```

在此示例查询中，对预订记录进行统计以得出每周的总数。每周的结束日期成为列名。

```
SELECT * FROM
    (SELECT
       booking_id,
       (date_trunc('week', booking_date::date) + '5 days'::interval)::date as enddate,
       hotel_code AS "hotel code"
FROM bookings
) PIVOT (
    count(booking_id) FOR enddate IN ('2023-02-04','2023-02-11','2023-02-18')
);
```

查询将生成以下输出。

```
 hotel code | 2023-02-04  | 2023-02-11 | 2023-02-18
------------+-------------+------------+----------
 FOREST_L   |           3 |          2 |        1
 DESERT_S   |           4 |          3 |        2
 OCEAN_WV   |           3 |          3 |        3
 CITY_BLD   |           3 |          1 |        2
```

Amazon Redshift 不支持 CROSSTAB 对多列进行透视。但您可以将行数据更改为列，与使用 PIVOT 进行聚合类似，使用如下所示的查询。这使用与上一个示例相同的预订示例数据。

```
SELECT
  booking_date,
  MAX(CASE WHEN hotel_code = 'FOREST_L' THEN 'forest is booked' ELSE '' END) AS
 FOREST_L,
```

```
  MAX(CASE WHEN hotel_code = 'DESERT_S' THEN 'desert is booked' ELSE '' END) AS
 DESERT_S,
  MAX(CASE WHEN hotel_code = 'OCEAN_WV' THEN 'ocean is booked' ELSE '' END)  AS
 OCEAN_WV
FROM bookings
GROUP BY booking_date
ORDER BY booking_date asc;
```

示例查询会导致在表示已预订了哪些酒店的短语旁边列出预订日期。

```
 booking_date  | forest_l         | desert_s         | ocean_wv
---------------+------------------+------------------+--------------------
 2023-02-01    | forest is booked | desert is booked |  ocean is booked
 2023-02-02    | forest is booked | desert is booked |  ocean is booked
 2023-02-04    | forest is booked | desert is booked |  ocean is booked
 2023-02-05    |                  | desert is booked |
 2023-02-06    |                  | desert is booked |
```

以下是 PIVOT 的使用说明：

- PIVOT 可以应用于表、子查询和公用表表达式（CTE）。PIVOT 不可应用于任何 JOIN 表达式、递归 CTE、PIVOT 或 UNPIVOT 表达式。此外，也不支持 SUPER 取消嵌套的表达式和 Redshift Spectrum 嵌套表。
- PIVOT 支持 COUNT、SUM、MIN、MAX 和 AVG 聚合函数。
- PIVOT 聚合表达式必须是对受支持的聚合函数的调用。不支持聚合顶部的复杂表达式。聚合参数仅可包含对 PIVOT 输入表的引用。此外，也不支持对父查询的关联引用。聚合参数可能包含子查询。它们可以在内部关联或在 PIVOT 输入表上关联。
- PIVOT IN 列表值不得为列引用或子查询。每个值必须与 FOR 列引用类型兼容。
- 如果 IN 列表值没有别名，PIVOT 会生成默认的列名。对于常量 IN 值（例如“abc”或 5），默认列名是常量本身。对于任何复杂表达式，列名都是标准的 Amazon Redshift 默认名称，例如 ? column?。

UNPIVOT 示例

设置示例数据并使用它来运行后续示例。

```
CREATE TABLE count_by_color (quality varchar, red int, green int, blue int);

INSERT INTO count_by_color VALUES ('high', 15, 20, 7);
```

```
INSERT INTO count_by_color VALUES ('normal', 35, NULL, 40);
INSERT INTO count_by_color VALUES ('low', 10, 23, NULL);
```

红、绿和蓝输入列上的 UNPIVOT。

```
SELECT *
FROM (SELECT red, green, blue FROM count_by_color) UNPIVOT (
    cnt FOR color IN (red, green, blue)
);
```

查询将生成以下输出。

```
 color | cnt
-------+-----
 red   |  15
 red   |  35
 red   |  10
 green |  20
 green |  23
 blue  |   7
 blue  |  40
```

默认情况下，将跳过输入列中的 NULL 值，且不会产生结果行。

以下示例显示了带有 INCLUDE NULLS 的 UNPIVOT。

```
SELECT *
FROM (
    SELECT red, green, blue
    FROM count_by_color
) UNPIVOT INCLUDE NULLS (
    cnt FOR color IN (red, green, blue)
);
```

以下是结果输出。

```
 color | cnt
-------+-----
 red   |  15
 red   |  35
 red   |  10
 green |  20
```

```
 green |
 green |  23
 blue  |   7
 blue  |  40
 blue  |
```

如果已设置 INCLUDING NULLS 参数，NULL 输入值将生成结果行。

带有 quality 的 The following query shows UNPIVOT 作为隐式列。

```
SELECT *
FROM count_by_color UNPIVOT (
    cnt FOR color IN (red, green, blue)
);
```

查询将生成以下输出。

```
 quality | color | cnt
---------+-------+-----
 high    | red   |  15
 normal  | red   |  35
 low     | red   |  10
 high    | green |  20
 low     | green |  23
 high    | blue  |   7
 normal  | blue  |  40
```

UNPIVOT 定义中未引用的输入表列被隐式添加到结果表中。在该示例中，quality 列就是这种情况。

以下示例显示了带有 UNPIVOT 列表中值别名的 IN。

```
SELECT *
FROM count_by_color UNPIVOT (
    cnt FOR color IN (red AS r, green AS g, blue AS b)
);
```

上一查询将产生以下输出。

```
 quality | color | cnt
---------+-------+-----
 high    | r     |  15
 normal  | r     |  35
```

```
low     | r     |  10
high    | g     |  20
low     | g     |  23
high    | b     |   7
normal  | b     |  40
```

UNPIVOT 运算符接受每个 IN 列表值上的可选别名。每个别名会提供每个 value 列中的数据自定义。

以下是 UNPIVOT 的使用说明。

- UNPIVOT 可以应用于表、子查询和公用表表达式（CTE）。UNPIVOT 不可应用于任何 JOIN 表达式、递归 CTE、PIVOT 或 UNPIVOT 表达式。此外，也不支持 SUPER 取消嵌套的表达式和 Redshift Spectrum 嵌套表。
- UNPIVOT IN 列表必须仅包含输入表列引用。IN 列表列必须具有它们都与之兼容的常见类型。UNPIVOT 值列具有这一常见类型。UNPIVOT 名称列属于类型 VARCHAR。
- 如果 IN 列表值没有别名，UNPIVOT 则使用列名作为默认值。

JOIN 示例

SQL JOIN 子句用于根据公共字段合并两个或多个表中的数据。根据指定的联接方法，结果可能会发生变化，也可能不发生变化。有关 JOIN 子句的语法的更多信息，请参阅参数。

以下示例使用 TICKIT 示例数据中的数据。有关数据库架构的更多信息，请参阅示例数据库。要了解如何加载示例数据，请参阅《Amazon Redshift 入门指南》中的加载数据。

下面的查询是 LISTING 表和 SALES 表之间的内部联接（不带 JOIN 关键字），其中 LISTING 表中的 LISTID 介于 1 和 5 之间。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果显示 LISTID 1、4 和 5 符合条件。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing, sales
where listing.listid = sales.listid
and listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     4 |  76.00 |  11.40
```

```
     5 | 525.00 |  78.75
```

以下查询是一个左外部联接。当在其他表中找不到匹配项时，左外部联接和右外部联接保留某个已联接表中的值。左表和右表是语法中列出的第一个表和第二个表。NULL 值用于填补结果集中的“空白”。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果表明 LISTID 2 和 3 不会生成任何销售额。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing left outer join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     2 | NULL   | NULL
     3 | NULL   | NULL
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

以下查询是一个右外部联接。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果显示 ListID 1、4 和 5 符合条件。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing right outer join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

以下查询是一个完全联接。当在其他表中找不到匹配项时，完全联接保留已联接表中的值。左表和右表是语法中列出的第一个表和第二个表。NULL 值用于填补结果集中的“空白”。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果表明 LISTID 2 和 3 不会生成任何销售额。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
```

```
from listing full join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     2 | NULL   | NULL
     3 | NULL   | NULL
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

以下查询是一个完全联接。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果中只有不会导致任何销售额的行（ListID 2 和 3）。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing full join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
and (listing.listid IS NULL or sales.listid IS NULL)
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     2 | NULL   | NULL
     3 | NULL   | NULL
```

以下示例是与 ON 子句的内部联接。在这种情况下，不返回 NULL 行。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from sales join listing
on sales.listid=listing.listid and sales.eventid=listing.eventid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

以下查询是 LISTING 表和 SALES 表的交叉联接或笛卡尔联接，其中包含限制结果的谓词。此查询匹配 SALES 表和 LISTING 表中的 LISTID 列值，对应于这两个表中的 LISTID 1、2、3、4 和 5。结果显示 20 个行符合条件。

```
select sales.listid as sales_listid, listing.listid as listing_listid
from sales cross join listing
where sales.listid between 1 and 5
and listing.listid between 1 and 5
order by 1,2;

sales_listid | listing_listid
-------------+---------------
1            | 1
1            | 2
1            | 3
1            | 4
1            | 5
4            | 1
4            | 2
4            | 3
4            | 4
4            | 5
5            | 1
5            | 1
5            | 2
5            | 2
5            | 3
5            | 3
5            | 4
5            | 4
5            | 5
5            | 5
```

以下示例是两个表之间的自然联接。在这种情况下，列 listid、sellerid、eventid 和 dateid 在两个表中具有相同的名称和数据类型，因此用作联接列。结果限制为 5 行。

```
select listid, sellerid, eventid, dateid, numtickets
from listing natural join sales
order by 1
limit 5;

listid | sellerid  | eventid | dateid | numtickets
-------+-----------+---------+--------+-----------
```

```
113    | 29704     | 4699    | 2075   | 22
115    | 39115     | 3513    | 2062   | 14
116    | 43314     | 8675    | 1910   | 28
118    | 6079      | 1611    | 1862   | 9
163    | 24880     | 8253    | 1888   | 14
```

以下示例是使用 USING 子句在两个表之间进行的联接。在这种情况下，列 listid 和 eventid 用作联接列。结果限制为 5 行。

```
select listid, listing.sellerid, eventid, listing.dateid, numtickets
from listing join sales
using (listid, eventid)
order by 1
limit 5;

listid | sellerid | eventid | dateid | numtickets
-------+----------+---------+--------+-----------
1      | 36861    | 7872    | 1850   | 10
4      | 8117     | 4337    | 1970   | 8
5      | 1616     | 8647    | 1963   | 4
5      | 1616     | 8647    | 1963   | 4
6      | 47402    | 8240    | 2053   | 18
```

以下查询是 FROM 子句中的两个子查询的内部联接。此查询查找不同类别的活动（音乐会和演出）的已售门票数和未售门票数：这些 FROM 子句子查询是表 子查询；它们可返回多个列和行。

```
select catgroup1, sold, unsold
from
(select catgroup, sum(qtysold) as sold
from category c, event e, sales s
where c.catid = e.catid and e.eventid = s.eventid
group by catgroup) as a(catgroup1, sold)
join
(select catgroup, sum(numtickets)-sum(qtysold) as unsold
from category c, event e, sales s, listing l
where c.catid = e.catid and e.eventid = s.eventid
and s.listid = l.listid
group by catgroup) as b(catgroup2, unsold)

on a.catgroup1 = b.catgroup2
order by 1;
```

```
catgroup1 |  sold  | unsold
----------+--------+--------
Concerts  | 195444 |1067199
Shows     | 149905 | 817736
```

WHERE 子句

WHERE 子句包含用于联接表或将谓词应用于表中的列的条件。可在 WHERE 子句或 FROM 子句中使用适当的语法对表进行内部联接。外部联接条件必须在 FROM 子句中指定。

语法

```
[ WHERE condition ]
```

condition

任何具有布尔型结果的搜索条件，例如，联接条件或表列上的谓词。以下示例是有效的联接条件：

```
sales.listid=listing.listid
sales.listid<>listing.listid
```

以下示例是表中的列上的有效条件：

```
catgroup like 'S%'
venueseats between 20000 and 50000
eventname in('Jersey Boys','Spamalot')
year=2008
length(catdesc)>25
date_part(month, caldate)=6
```

条件可以是简单条件或复杂条件；对于复杂条件，可以使用圆括号来分隔逻辑单元。在下面的示例中，用圆括号将联接条件括起来。

```
where (category.catid=event.catid) and category.catid in(6,7,8)
```

使用说明

您可在 WHERE 子句中使用别名来引用选择列表表达式。

不能限制 WHERE 子句中的聚合函数的结果；要实现此目的，请使用 HAVING 子句。

WHERE 子句中受限制的列必须派生自 FROM 子句中的表引用。

示例

以下查询使用不同的 WHERE 子句限制的组合，包括 SALES 表和 EVENT 表的联接条件、EVENTNAME 列上的谓词以及 STARTTIME 列上的两个谓词。

```
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Hannah Montana'
and date_part(quarter, starttime) in(1,2)
and date_part(year, starttime) = 2008
order by 3 desc, 4, 2, 1 limit 10;

eventname    |      starttime      |   costperticket   | qtysold
---------------+---------------------+-------------------+---------
Hannah Montana | 2008-06-07 14:00:00 |     1706.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |     1658.00000000 |       2
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       3
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       2
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       4
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       1
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       4
(10 rows)
```

WHERE 子句中的 Oracle 样式的外部联接

为了与 Oracle 兼容，Amazon Redshift 支持 WHERE 子句联接条件中的 Oracle 外部联接运算符 (+)。此运算符仅用于定义外部联接条件；不要尝试在其他上下文中使用它。在大多数情况下，会无提示忽略此运算符的其他使用方式。

外部联接返回与内部联接相同的所有行，还返回一个或两个表中的非匹配行。在 FROM 子句中，可以指定左、右和完全外部联接。在 WHERE 子句中，只能指定左和右外部联接。

要对表 TABLE1 和 TABLE2 执行外部联接并返回 TABLE1 中的非匹配行（左外部联接），请在 FROM 子句中指定 TABLE1 LEFT OUTER JOIN TABLE2，或者将 (+) 运算符应用于 WHERE 子句中的 TABLE2 中的所有联接列。对于 TABLE1 中的在 TABLE2 中没有匹配行的所有行，在查询结果中，会为包含 TABLE2 中列的任何选择列表表达式显示 Null 值。

要为 TABLE2 中的在 TABLE1 中没有匹配行的所有行生成相同的行为，请在 FROM 子句中指定 TABLE1 RIGHT OUTER JOIN TABLE2，或者将 (+) 运算符应用于 WHERE 子句中的 TABLE1 中的所有联接列。

基本语法

```
[ WHERE {
[ table1.column1 = table2.column1(+) ]
[ table1.column1(+) = table2.column1 ]
}
```

第一个条件等效于：

```
from table1 left outer join table2
on table1.column1=table2.column1
```

第二个条件等效于：

```
from table1 right outer join table2
on table1.column1=table2.column1
```

> **Note**
>
> 此处显示的语法涵盖了基于一对联接列的 equijoin 的单个示例。不过，其他类型的比较条件和多个联接列对也有效。

例如，以下 WHERE 子句定义基于两个列对的外部联接。(+) 运算符必须附加到两个条件中的同一个表：

```
where table1.col1 > table2.col1(+)
and table1.col2 = table2.col2(+)
```

使用说明

如果可能，请在 WHERE 子句中使用标准 FROM 子句 OUTER JOIN 语法而非 (+) 运算符。包含 (+) 运算符的查询需遵循以下规则：

- 只能在 WHERE 语句中以及对表或视图中的列的引用中使用 (+) 运算符。

- 不能对表达式应用 (+) 运算符。不过，表达式可以包含使用 (+) 运算符的列。例如，以下联接条件返回语法错误：

```
event.eventid*10(+)=category.catid
```

不过，以下联接条件有效：

```
event.eventid(+)*10=category.catid
```

- 不能在同时包含 FROM 子句联接语法的查询块中使用 (+) 运算符。
- 如果两个表基于多个联接条件进行联接，则必须在所有这些条件中都使用或都不使用 (+) 运算符。使用混合语法样式的联接将作为内部联接执行，不会产生警告。
- 如果将外部查询中的表与内部查询所生成的表联接，则 (+) 运算符不会产生外部联接。
- 要使用 (+) 运算符来将表外部联接到自身，则必须在 FROM 子句中定义表别名并在联接条件中引用这些别名：

```
select count(*)
from event a, event b
where a.eventid(+)=b.catid;

count
-------
8798
(1 row)
```

- 不能将包含 (+) 运算符的联接条件与 OR 条件或 IN 条件结合使用。例如：

```
select count(*) from sales, listing
where sales.listid(+)=listing.listid or sales.salesid=0;
ERROR:  Outer join operator (+) not allowed in operand of OR or IN.
```

- 在对两个以上的表执行外部联接的 WHERE 子句中，只能将 (+) 运算符应用于指定的表一次。在下面的示例中，不能在两个连续联接中使用 (+) 运算符来引用 SALES 表。

```
select count(*) from sales, listing, event
where sales.listid(+)=listing.listid and sales.dateid(+)=date.dateid;
ERROR:  A table may be outer joined to at most one other table.
```

- 如果 WHERE 子句外部联接条件将 TABLE2 中的一个列与常量比较，则对该列应用 (+) 运算符。如果您包括运算符，则会消除 TABLE1 中的外部联接的行（受限制列包含 null 值）。请参阅下面的“示例”部分。

示例

以下联接查询指定 SALES 和 LISTING 表的左外部联接（基于其 LISTID 列）：

```
select count(*)
from sales, listing
where sales.listid = listing.listid(+);

count
--------
172456
(1 row)
```

以下等效查询生成相同的结果，不过使用的是 FROM 子句联接语法：

```
select count(*)
from sales left outer join listing on sales.listid = listing.listid;

count
--------
172456
(1 row)
```

SALES 表不包含 LISTING 表中所有列表的记录，因为并非所有列表都生成销售值。以下查询对 SALES 和 LISTING 执行外部联接，并返回 LISTING 中的行（甚至在 SALES 表未报告给定列表 ID 的销售值也是如此）。PRICE 和 COMM 列（派生自 SALES 表）在结果集中为这些非匹配行包含 null 值。

```
select listing.listid, sum(pricepaid) as price,
sum(commission) as comm
from listing, sales
where sales.listid(+) = listing.listid and listing.listid between 1 and 5
group by 1 order by 1;

listid | price  |  comm
--------+--------+--------
1 | 728.00 | 109.20
```

```
2 |        |
3 |        |
4 |  76.00 |  11.40
5 | 525.00 |  78.75
(5 rows)
```

请注意，在使用 WHERE 子句联接运算符时，FROM 子句中表的顺序并不重要。

WHERE 子句中更复杂的外部联接条件的示例是，对两个表列之间的比较以及带常量的比较执行与 运算的条件：

```
where category.catid=event.catid(+) and eventid(+)=796;
```

请注意，在两个位置使用 (+) 运算符：首先是在表之间的相等比较中，其次是在 EVENTID 列的比较条件中。此语法的结果是，在计算 EVENTID 的限制时保留外部联接的行。如果您从 EVENTID 限制中删除 (+) 运算符，则查询会将此限制视为筛选条件而不是视为外部联接条件的一部分。反过来，将从结果集中消除包含 EVENTID 的 null 值的外部联接的行。

下面是演示此行为的完整查询：

```
select catname, catgroup, eventid
from category, event
where category.catid=event.catid(+) and eventid(+)=796;

catname | catgroup | eventid
-----------+----------+---------
Classical | Concerts |
Jazz | Concerts |
MLB | Sports   |
MLS | Sports   |
Musicals | Shows    | 796
NBA | Sports   |
NFL | Sports   |
NHL | Sports   |
Opera | Shows    |
Plays | Shows    |
Pop | Concerts |
(11 rows)
```

使用 FROM 子句语法的等效查询如下：

```
select catname, catgroup, eventid
```

```
from category left join event
on category.catid=event.catid and eventid=796;
```

如果从该查询的 WHERE 子句版本中删除第二个 (+) 运算符，则它仅返回 1 行（其中包含 eventid=796 的行）。

```
select catname, catgroup, eventid
from category, event
where category.catid=event.catid(+) and eventid=796;

catname | catgroup | eventid
-----------+----------+---------
Musicals | Shows    | 796
(1 row)
```

GROUP BY 子句

GROUP BY 子句标识查询的分组列。必须在查询使用标准函数（例如，SUM、AVG 和 COUNT）计算聚合时声明分组列。有关更多信息，请参阅 聚合函数。

语法

```
GROUP BY group_by_clause [, ...]

group_by_clause := {
    expr |
    GROUPING SETS ( () | group_by_clause [, ...] ) |
    ROLLUP ( expr [, ...] ) |
    CUBE ( expr [, ...] )
    }
```

参数

expr

列或表达式的列表必须匹配查询的选择列表中的非聚合表达式的列表。例如，考虑以下简单查询。

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by listid, eventid
```

```
order by 3, 4, 2, 1
limit 5;

listid | eventid | revenue | numtix
-------+---------+---------+--------
89397  |      47 |   20.00 |      1
106590 |      76 |   20.00 |      1
124683 |     393 |   20.00 |      1
103037 |     403 |   20.00 |      1
147685 |     429 |   20.00 |      1
(5 rows)
```

在此查询中，选择列表包含两个聚合表达式。第一个聚合表达式使用 SUM 函数，第二个聚合表达式使用 COUNT 函数。必须将其余两个列（LISTID 和 EVENTID）声明为分组列。

GROUP BY 子句中的表达式也可以使用序号来引用选择列表。例如，上一个示例的缩略形式如下。

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by 1,2
order by 3, 4, 2, 1
limit 5;

listid | eventid | revenue | numtix
-------+---------+---------+--------
89397  |      47 |   20.00 |      1
106590 |      76 |   20.00 |      1
124683 |     393 |   20.00 |      1
103037 |     403 |   20.00 |      1
147685 |     429 |   20.00 |      1
(5 rows)
```

GROUPING SETS/ROLLUP/CUBE

您可以使用聚合扩展 GROUPING SETS、ROLLUP 和 CUBE 在单个语句中执行多个 GROUP BY 操作的工作。有关聚合扩展和相关函数的更多信息，请参阅聚合扩展。

聚合扩展

Amazon Redshift 支持聚合扩展，以便在单个语句中完成多个 GROUP BY 操作。

聚合扩展的示例使用 orders 表，该表包含电子公司的销售数据。您可以使用以下项创建 orders。

```
CREATE TABLE ORDERS (
    ID INT,
    PRODUCT CHAR(20),
    CATEGORY CHAR(20),
    PRE_OWNED CHAR(1),
    COST DECIMAL
);

INSERT INTO ORDERS VALUES
    (0, 'laptop',       'computers',    'T', 1000),
    (1, 'smartphone',   'cellphones',   'T', 800),
    (2, 'smartphone',   'cellphones',   'T', 810),
    (3, 'laptop',       'computers',    'F', 1050),
    (4, 'mouse',        'computers',    'F', 50);
```

GROUPING SETS

在单个语句中计算一个或多个分组集。分组集是单个 GROUP BY 子句的集合，这是一组 0 列或更多列，您可以通过这些列对查询的结果集进行分组。GROUP BY GROUPING SETS 等效于对一个按不同列分组的结果集运行 UNION ALL 查询。例如，GROUP BY GROUPING SETS((a), (b)) 等效于 GROUP BY a UNION ALL GROUP BY b。

以下示例返回按产品类别和所售产品类型分组的订单表产品的成本。

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY GROUPING SETS(category, product);

       category       |       product        | total
----------------------+----------------------+-------
 computers            |                      |  2100
 cellphones           |                      |  1610
                      | laptop               |  2050
                      | smartphone           |  1610
                      | mouse                |    50

(5 rows)
```

ROLLUP

假设在一个层次结构中，前面的列被视为后续列的父列。ROLLUP 按提供的列对数据进行分组，除了分组行之外，还返回额外的小计行，表示所有分组列级别的总计。例如，您可以使用 GROUP BY ROLLUP((a), (b)) 返回先按 a 分组的结果集，然后在假设 b 是 a 的一个子部分的情况下按 b 分组。ROLLUP 还会返回包含整个结果集而不包含分组列的行。

GROUP BY ROLLUP((a), (b)) 等效于 GROUP BY GROUPING SETS((a,b), (a), ())。

以下示例返回先按类别分组，然后按产品分组，且产品是类别细分项的订单表产品的成本。

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY ROLLUP(category, product) ORDER BY 1,2;

       category       |       product        | total
----------------------+----------------------+-------
 cellphones           | smartphone           |  1610
 cellphones           |                      |  1610
 computers            | laptop               |  2050
 computers            | mouse                |    50
 computers            |                      |  2100
                      |                      |  3710
(6 rows)
```

CUBE

按提供的列对数据进行分组，除了分组行之外，还返回额外的小计行，表示所有分组列级别的总计。CUBE 返回与 ROLLUP 相同的行，同时为 ROLLUP 未涵盖的每个分组列组合添加额外的小计行。例如，您可以使用 GROUP BY CUBE ((a), (b)) 返回先按 a 分组的结果集，然后在假设 b 是 a 的一个子部分的情况下按 b 分组，再然后是单独按 b 分组的结果集。CUBE 还会返回包含整个结果集而不包含分组列的行。

GROUP BY CUBE((a), (b)) 等效于 GROUP BY GROUPING SETS((a, b), (a), (b), ())。

以下示例返回先按类别分组，然后按产品分组，且产品是类别细分项的订单表产品的成本。与前面的 ROLLUP 示例不同，该语句返回每个分组列组合的结果。

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY CUBE(category, product) ORDER BY 1,2;
```

```
       category       |       product        | total
----------------------+----------------------+-------
 cellphones           | smartphone           |  1610
 cellphones           |                      |  1610
 computers            | laptop               |  2050
 computers            | mouse                |    50
 computers            |                      |  2100
                      | laptop               |  2050
                      | mouse                |    50
                      | smartphone           |  1610
                      |                      |  3710
(9 rows)
```

GROUPING/GROUPING_ID 函数

ROLLUP 和 CUBE 为结果集添加 NULL 值以表示小计行。例如，GROUP BY ROLLUP((a), (b)) 返回 b 分组列中值为 NULL 的一行或多行，以表明它们是 a 分组列中字段的小计。这些 NULL 值仅用于满足返回元组的格式。

当您使用 ROLLUP 和 CUBE 对本身存储 NULL 值的关系运行 GROUP BY 操作时，这样生成的结果集会包含看起来具有相同分组列的行。返回前面的示例，如果 b 分组列包含存储的 NULL 值，则 GROUP BY ROLLUP((a), (b)) 返回 b 分组列中值为 NULL 且不是小计的行。

要区分由 ROLLUP 和 CUBE 创建的 NULL 值和在表本身存储的 NULL 值，可以使用 GROUPING 函数或其别名 GROUPING_ID。GROUPING 采用单个分组集作为其参数，并且对于结果集中的每一行，返回与该位置的分组列对应的 0 或 1 位值，然后将该值转换为整数。如果该位置的值是由聚合扩展创建的 NULL 值，则 GROUPING 返回 1。对于所有其他值（包括存储的 NULL 值），该示例返回 0。

例如，GROUPING(category, product) 为给定行返回以下值，具体取决于该行的分组列值。就本示例而言，表中的所有 NULL 值都是由聚合扩展创建的 NULL 值。

类别列	产品列	GROUPING 函数位值	十进制值
非 Null	非 Null	00	0
非 Null	NULL	01	1
NULL	非 Null	10	2
NULL	NULL	11	3

GROUPING 函数按照以下格式显示在查询的 SELECT 列表部分。

```
SELECT ... [GROUPING( expr )...] ...
  GROUP BY ... {CUBE | ROLLUP| GROUPING SETS} ( expr ) ...
```

以下示例与前面的 CUBE 示例相同，但为其分组集添加了 GROUPING 函数。

```
SELECT category, product,
       GROUPING(category) as grouping0,
       GROUPING(product) as grouping1,
       GROUPING(category, product) as grouping2,
       sum(cost) as total
FROM orders
GROUP BY CUBE(category, product) ORDER BY 3,1,2;

       category       |       product        | grouping0 | grouping1 | grouping2 |
 total
----------------------+----------------------+-----------+-----------+-----------
+-------
 cellphones           | smartphone           |         0 |         0 |         0 |
 1610
 cellphones           |                      |         0 |         1 |         1 |
 1610
 computers            | laptop               |         0 |         0 |         0 |
 2050
 computers            | mouse                |         0 |         0 |         0 |
 50
 computers            |                      |         0 |         1 |         1 |
 2100
                      | laptop               |         1 |         0 |         2 |
 2050
                      | mouse                |         1 |         0 |         2 |
 50
                      | smartphone           |         1 |         0 |         2 |
 1610
                      |                      |         1 |         1 |         3 |
 3710
(9 rows)
```

部分 ROLLUP 和 CUBE

您可以只使用小计的一部分来运行 ROLLUP 和 CUBE 操作。

部分 ROLLUP 和 CUBE 操作的语法如下所示。

```
GROUP BY expr1, { ROLLUP | CUBE }(expr2, [, ...])
```

在这里，GROUP BY 子句仅在 expr2 及更高级别创建小计行。

以下示例显示了对订单表执行的部分 ROLLUP 和 CUBE 操作，首先按产品是否为二手产品进行分组，然后对 category 和 product 列运行 ROLLUP 和 CUBE。

```
SELECT pre_owned, category, product,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY pre_owned, ROLLUP(category, product) ORDER BY 4,1,2,3;

 pre_owned |       category       |       product        | group_id | total
-----------+----------------------+----------------------+----------+-------
 F         | computers            | laptop               |        0 |  1050
 F         | computers            | mouse                |        0 |    50
 T         | cellphones           | smartphone           |        0 |  1610
 T         | computers            | laptop               |        0 |  1000
 F         | computers            |                      |        2 |  1100
 T         | cellphones           |                      |        2 |  1610
 T         | computers            |                      |        2 |  1000
 F         |                      |                      |        6 |  1100
 T         |                      |                      |        6 |  2610
(9 rows)

SELECT pre_owned, category, product,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY pre_owned, CUBE(category, product) ORDER BY 4,1,2,3;

 pre_owned |       category       |       product        | group_id | total
-----------+----------------------+----------------------+----------+-------
 F         | computers            | laptop               |        0 |  1050
 F         | computers            | mouse                |        0 |    50
 T         | cellphones           | smartphone           |        0 |  1610
 T         | computers            | laptop               |        0 |  1000
 F         | computers            |                      |        2 |  1100
 T         | cellphones           |                      |        2 |  1610
 T         | computers            |                      |        2 |  1000
```

```
 F          |                       | laptop                |        4 |  1050
 F          |                       | mouse                 |        4 |    50
 T          |                       | laptop                |        4 |  1000
 T          |                       | smartphone            |        4 |  1610
 F          |                       |                       |        6 |  1100
 T          |                       |                       |        6 |  2610
(13 rows)
```

由于 ROLLUP 和 CUBE 操作中不包括 pre-owned 列，因此不存在包括所有其他行的总计行。

连接分组

您可以连接多个 GROUPING SETS/ROLLUP/CUBE 子句来计算不同级别的小计。连接分组返回所提供分组集的笛卡尔乘积。

连接 GROUPING SETS/ROLLUP/CUBE 子句的语法如下所示。

```
GROUP BY {ROLLUP|CUBE|GROUPING SETS}(expr1[, ...]),
         {ROLLUP|CUBE|GROUPING SETS}(expr1[, ...])[, ...]
```

考虑以下示例，看看小型连接分组如何生成大的最终结果集。

```
SELECT pre_owned, category, product,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY CUBE(category, product), GROUPING SETS(pre_owned, ())
ORDER BY 4,1,2,3;

 pre_owned |       category       |       product         | group_id | total
-----------+----------------------+-----------------------+----------+-------
 F         | computers            | laptop                |        0 |  1050
 F         | computers            | mouse                 |        0 |    50
 T         | cellphones           | smartphone            |        0 |  1610
 T         | computers            | laptop                |        0 |  1000
           | cellphones           | smartphone            |        1 |  1610
           | computers            | laptop                |        1 |  2050
           | computers            | mouse                 |        1 |    50
 F         | computers            |                       |        2 |  1100
 T         | cellphones           |                       |        2 |  1610
 T         | computers            |                       |        2 |  1000
           | cellphones           |                       |        3 |  1610
```

```
           | computers            |                      |        3 |  2100
 F         |                      | laptop               |        4 |  1050
 F         |                      | mouse                |        4 |    50
 T         |                      | laptop               |        4 |  1000
 T         |                      | smartphone           |        4 |  1610
           |                      | laptop               |        5 |  2050
           |                      | mouse                |        5 |    50
           |                      | smartphone           |        5 |  1610
 F         |                      |                      |        6 |  1100
 T         |                      |                      |        6 |  2610
           |                      |                      |        7 |  3710
(22 rows)
```

嵌套分组

您可以使用 GROUPING SETS/ROLLUP/CUBE 操作作为 GROUPING SETS expr 来形成嵌套分组。展平嵌套 GROUPING SETS 内的子分组。

嵌套分组的语法如下所示。

```
GROUP BY GROUPING SETS({ROLLUP|CUBE|GROUPING SETS}(expr[, ...])[, ...])
```

考虑以下示例。

```
SELECT category, product, pre_owned,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY GROUPING SETS(ROLLUP(category), CUBE(product, pre_owned))
ORDER BY 4,1,2,3;

       category       |       product        | pre_owned | group_id | total
----------------------+----------------------+-----------+----------+-------
 cellphones           |                      |           |        3 |  1610
 computers            |                      |           |        3 |  2100
                      | laptop               | F         |        4 |  1050
                      | laptop               | T         |        4 |  1000
                      | mouse                | F         |        4 |    50
                      | smartphone           | T         |        4 |  1610
                      | laptop               |           |        5 |  2050
                      | mouse                |           |        5 |    50
                      | smartphone           |           |        5 |  1610
```

```
                    |                    | F          |        6 |  1100
                    |                    | T          |        6 |  2610
                    |                    |            |        7 |  3710
                    |                    |            |        7 |  3710
(13 rows)
```

请注意，由于 ROLLUP(category) 和 CUBE(product, pre_owned) 都包含分组集 ()，因此表示总计的行是重复的。

使用说明

- GROUP BY 子句最多支持 64 个分组集。对于 ROLLUP 和 CUBE，或者 GROUPING SETS、ROLLUP 和 CUBE 的某种组合，此限制适用于隐含的分组集数。例如，GROUP BY CUBE((a), (b)) 计为 4 个分组集，而不是 2 个。
- 使用聚合扩展时，不能使用常量作为分组列。
- 无法创建包含重复列的分组集。

HAVING 子句

HAVING 子句将条件应用于查询返回的中间分组结果集。

语法

```
[ HAVING condition ]
```

例如，您可以限制 SUM 函数的结果：

```
having sum(pricepaid) >10000
```

在应用所有 WHERE 子句条件并完成 GROUP BY 操作后，应用 HAVING 条件。

条件本身采用与任何 WHERE 子句条件相同的形式。

使用说明

- HAVING 子句条件中引用的任何列必须为分组列或引用了聚合函数结果的列。
- 在 HAVING 子句中，无法指定：
  - 引用选择列表项的序号。仅 GROUP BY 和 ORDER BY 子句接受序号。

示例

以下查询按名称计算所有活动的门票总销售额，然后消除总销售额小于 $800000 的活动。HAVING 条件应用于选择列表中聚合函数的结果：sum(pricepaid)。

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
group by 1
having sum(pricepaid) > 800000
order by 2 desc, 1;

eventname        |    sum
-----------------+-----------
Mamma Mia!       | 1135454.00
Spring Awakening |  972855.00
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
```

以下查询计算类似的结果集。不过，在本示例中，HAVING 条件将应用于未在选择列表中指定的聚合：sum(qtysold)。将从最终结果中消除未售出 2000 张以上的门票的活动。

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
group by 1
having sum(qtysold) >2000
order by 2 desc, 1;

eventname        |    sum
-----------------+-----------
Mamma Mia!       | 1135454.00
Spring Awakening |  972855.00
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
Chicago          |  790993.00
Spamalot         |  714307.00
```

以下查询按名称计算所有活动的门票总销售额，然后消除总销售额小于 $800000 的活动。HAVING 条件应用于选择列表中聚合函数的结果（对 sum(pricepaid) 使用别名 pp）。

```
select eventname, sum(pricepaid) as pp
from sales join event on sales.eventid = event.eventid
group by 1
having pp > 800000
order by 2 desc, 1;

eventname        |    pp
-----------------+-----------
Mamma Mia!       | 1135454.00
Spring Awakening |  972855.00
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
```

QUALIFY 子句

QUALIFY 子句根据用户指定的搜索条件，筛选先前计算的窗口函数的结果。您可以使用此子句将筛选条件应用于窗口函数的结果，而无需使用子查询。

此子句与 HAVING 子句类似，后者应用条件以进一步从 WHERE 子句筛选行。QUALIFY 和 HAVING 的区别在于，QUALIFY 子句的筛选结果可以基于对数据运行窗口函数的结果。在一个查询中可以同时使用 QUALIFY 和 HAVING 子句。

语法

```
QUALIFY condition
```

> **Note**
>
> 如果您在 FROM 子句之后直接使用 QUALIFY 子句，则 FROM 关系名称必须在 QUALIFY 子句之前指定了别名。

示例

本节中的示例使用下面的示例数据。

```
create table store_sales (ss_sold_date date, ss_sold_time time,
               ss_item text, ss_sales_price float);
insert into store_sales values ('2022-01-01', '09:00:00', 'Product 1', 100.0),
```

```
                                ('2022-01-01', '11:00:00', 'Product 2', 500.0),
                                ('2022-01-01', '15:00:00', 'Product 3', 20.0),
                                ('2022-01-01', '17:00:00', 'Product 4', 1000.0),
                                ('2022-01-01', '18:00:00', 'Product 5', 30.0),
                                ('2022-01-02', '10:00:00', 'Product 6', 5000.0),
                                ('2022-01-02', '16:00:00', 'Product 7', 5.0);
```

以下示例演示了如何查找每天 12:00 之后售出的两件最昂贵的商品。

```
SELECT *
FROM store_sales ss
WHERE ss_sold_time > time '12:00:00'
QUALIFY row_number()
OVER (PARTITION BY ss_sold_date ORDER BY ss_sales_price DESC) <= 2


 ss_sold_date | ss_sold_time |  ss_item  | ss_sales_price
--------------+--------------+-----------+----------------
 2022-01-01   | 17:00:00     | Product 4 |           1000
 2022-01-01   | 18:00:00     | Product 5 |             30
 2022-01-02   | 16:00:00     | Product 7 |              5
```

然后，您可以找到每天售出的最后一件商品。

```
SELECT *
FROM store_sales ss
QUALIFY last_value(ss_item)
OVER (PARTITION BY ss_sold_date ORDER BY ss_sold_time ASC
      ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) = ss_item;

ss_sold_date | ss_sold_time |  ss_item  | ss_sales_price
--------------+--------------+-----------+----------------
 2022-01-01   | 18:00:00     | Product 5 |             30
 2022-01-02   | 16:00:00     | Product 7 |              5
```

以下示例返回与前一个查询相同的记录，即每天售出的最后一件商品，但它没有使用 QUALIFY 子句。

```
SELECT * FROM (
  SELECT *,
  last_value(ss_item)
  OVER (PARTITION BY ss_sold_date ORDER BY ss_sold_time ASC
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) ss_last_item
```

```
  FROM store_sales ss
)
WHERE ss_last_item = ss_item;

 ss_sold_date | ss_sold_time |  ss_item  | ss_sales_price | ss_last_item
--------------+--------------+-----------+----------------+--------------
 2022-01-02   | 16:00:00     | Product 7 |              5 | Product 7
 2022-01-01   | 18:00:00     | Product 5 |             30 | Product 5
```

UNION、INTERSECT 和 EXCEPT

主题

- 语法
- 参数
- 集合运算符的计算顺序
- 使用说明
- 示例 UNION 查询
- 示例 UNION ALL 查询
- 示例 INTERSECT 查询
- 示例 EXCEPT 查询

UNION、INTERSECT 和 EXCEPT 集合运算符 用于比较和合并两个单独的查询表达式的结果。例如，如果您希望知道网站的哪些用户既是买家又是卖家且其用户名存储在单独的列或表中，则可查找这两类用户的交集。如果您希望知道哪些网站用户是买家而不是卖家，则可使用 EXCEPT 运算符查找这两个用户列表的差集。如果您希望构建一个所有用户的列表（无论角色如何），则可使用 UNION 运算符。

语法

```
query
{ UNION [ ALL ] | INTERSECT | EXCEPT | MINUS }
query
```

参数

query

一个查询表达式，该表达式（采用其选择列表形式）对应于紧跟 UNION、INTERSECT 或 EXCEPT 运算符的第二个查询表达式。这两个表达式必须包含数量相同并且数据类型兼容的输出列；否则，无法比较和合并两个结果集。集合运算不允许不同类别的数据类型之间的隐式转换；有关更多信息，请参阅 类型兼容性和转换。

您可以构建包含无限数量的查询表达式并任意组合使用 UNION、INTERSECT 和 EXCEPT 运算符来将这些表达式链接起来的查询。例如，假定表 T1、T2 和 T3 包含兼容的列集，则以下查询结构是有效的：

```
select * from t1
union
select * from t2
except
select * from t3
order by c1;
```

联合

从两个查询表达式返回行的集合运算，无论行派生自一个查询表达式还是两个查询表达式。

INTERSECT

返回派生自两个查询表达式的行的集合运算。将丢弃未同时由两个表达式返回的行。

EXCEPT | MINUS

返回派生自两个查询表达式之一的行的集合运算。要符合结果的要求，行必须存在于第一个结果表而不存在于第二个结果表中。MINUS 和 EXCEPT 完全同义。

ALL

ALL 关键字保留由 UNION 生成的任何重复行。未使用 ALL 关键字时的默认行为是丢弃这些重复项。不支持 INTERSECT ALL、EXCEPT ALL 和 MINUS ALL。

集合运算符的计算顺序

UNION 和 EXCEPT 集合运算符是左关联的。如果未指定圆括号来影响优先顺序，则将以从左到右的顺序来计算这些集合运算符的组合。例如，在以下查询中，首先计算 T1 和 T2 的 UNION，然后对 UNION 结果执行 EXCEPT 操作：

```
select * from t1
union
select * from t2
except
select * from t3
order by c1;
```

在同一个查询中使用运算符组合时，INTERSECT 运算符优先于 UNION 和 EXCEPT 运算符。例如，以下查询将计算 T2 和 T3 的交集，然后计算得到的结果与 T1 的并集：

```
select * from t1
union
select * from t2
intersect
select * from t3
order by c1;
```

通过添加圆括号，可以强制实施不同的计算顺序。在以下示例中，将 T1 和 T2 的并集结果与 T3 执行交集运算，并且查询可能会生成不同的结果。

```
(select * from t1
union
select * from t2)
intersect
(select * from t3)
order by c1;
```

使用说明

- 集合运算查询结果中返回的列名是来自第一个查询表达式中的表的列名（或别名）。由于这些列名可能会造成误解（因为列中的值派生自位于集合运算符任一侧的表），您可能需要为结果集提供有意义的别名。
- 集合运算符之前的查询表达式不应包含 ORDER BY 子句。仅在包含集合运算符的查询结尾处使用 ORDER BY 子句时，该子句才会生成有意义的排序结果。在这种情况下，ORDER BY 子句应用于所有集合运算的最终结果。最外层的查询也可以包含标准 LIMIT 和 OFFSET 子句。
- 当集合运算符查询返回小数结果时，将提升对应的结果列以返回相同的精度和小数位数。例如，在以下查询中，T1.REVENUE 为 DECIMAL(10,2) 列而 T2.REVENUE 为 DECIMAL(8,4) 列，小数结果将提升为 DECIMAL(12,4)：

```
select t1.revenue union select t2.revenue;
```

小数位数为 4，因为这是两个列的最大小数位数。精度为 12，因为 T1.REVENUE 要求小数点左侧有 8 位数 (12 - 4 = 8)。此类提升可确保 UNION 两侧的所有值都适合结果。对于 64 位值，最大结果精度为 19，最大结果小数位数为 18。对于 128 位值，最大结果精度为 38，最大结果小数位数为 37。

如果生成的数据类型超出 Amazon Redshift 精度和小数位数限制，则查询将返回错误。

- 对于集合运算，如果对于每个相应的列对，两个数据值相等 或都为 NULL，则两个行将被视为相同。例如，如果表 T1 和 T2 都包含一列和一行，并且两个表中的行都为 NULL，则对这两个表执行的 INTERSECT 运算将返回该行。

示例 UNION 查询

在以下 UNION 查询中，SALES 表中的行将与 LISTING 表中的行合并。从每个表中选择三个兼容的列；在这种情况下，对应的列具有相同的名称和数据类型。

最终结果集按 LISTING 表中的第一列进行排序且最多包含 5 个具有最高 LISTID 值的行。

```
select listid, sellerid, eventid from listing
union select listid, sellerid, eventid from sales
order by listid, sellerid, eventid desc limit 5;

listid | sellerid | eventid
--------+----------+---------
1 |    36861 |    7872
2 |    16002 |    4806
3 |    21461 |    4256
4 |     8117 |    4337
5 |     1616 |    8647
(5 rows)
```

以下示例说明如何将文本值添加到 UNION 查询的输出，以便您查看哪个查询表达式生成了结果集中的每一行。查询将第一个查询表达式中的行标识为“B”（针对买家），并将第二个查询表达式中的行标识为“S”（针对卖家）。

查询标识门票事务费用等于或大于 $10000 的买家和卖家。UNION 运算符的任一侧的两个查询表达式之间的唯一差异就是 SALES 表的联接列。

```
select listid, lastname, firstname, username,
pricepaid as price, 'S' as buyorsell
from sales, users
where sales.sellerid=users.userid
and pricepaid >=10000
union
select listid, lastname, firstname, username, pricepaid,
'B' as buyorsell
from sales, users
where sales.buyerid=users.userid
and pricepaid >=10000
order by 1, 2, 3, 4, 5;

listid | lastname | firstname | username |   price   | buyorsell
--------+----------+-----------+----------+-----------+-----------
209658 | Lamb     | Colette   | VOR15LYI |  10000.00 | B
209658 | West     | Kato      | ELU81XAA |  10000.00 | S
212395 | Greer    | Harlan    | GXO71KOC |  12624.00 | S
212395 | Perry    | Cora      | YWR73YNZ |  12624.00 | B
215156 | Banks    | Patrick   | ZNQ69CLT |  10000.00 | S
215156 | Hayden   | Malachi   | BBG56AKU |  10000.00 | B
(6 rows)
```

以下示例使用 UNION ALL 运算符，因为需要在结果中保留重复行（如果发现重复行）。对于一系列特定的活动 ID，查询为与每个活动关联的每个销售值返回 0 行或多个行，并为该活动的每个列表返回 0 行或 1 个行。活动 ID 对于 LISTING 和 EVENT 表中的每个行是唯一的，但对于 SALES 表中的活动和列表 ID 的相同组合，可能有多个销售值。

结果集中的第三个列标识行的来源。如果行来自 SALES 表，则在 SALESROW 列中将其标记为“Yes”。（SALESROW 是 SALES.LISTID 的别名。）如果行来自 LISTING 表，则在 SALESROW 列中将其标记为“No”。

在本示例中，结果集包含针对列表 500，活动 7787 的三个销售行。换而言之，将针对此列表和活动组合执行三个不同的事务。其他两个列表（501 和 502）不生成任何销售值，因此只有查询为这些列表 ID 生成的行来自 LISTING 表 (SALESROW = 'No')。

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
```

```
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(6 rows)
```

如果运行不带 ALL 关键字的相同查询，则结果只保留其中一个销售交易。

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(4 rows)
```

示例 UNION ALL 查询

以下示例使用 UNION ALL 运算符，因为需要在结果中保留重复行（如果发现重复行）。对于一系列特定的活动 ID，查询为与每个活动关联的每个销售值返回 0 行或多个行，并为该活动的每个列表返回 0 行或 1 个行。活动 ID 对于 LISTING 和 EVENT 表中的每个行是唯一的，但对于 SALES 表中的活动和列表 ID 的相同组合，可能有多个销售值。

结果集中的第三个列标识行的来源。如果行来自 SALES 表，则在 SALESROW 列中将其标记为“Yes”。（SALESROW 是 SALES.LISTID 的别名。）如果行来自 LISTING 表，则在 SALESROW 列中将其标记为“No”。

在本示例中，结果集包含针对列表 500，活动 7787 的三个销售行。换而言之，将针对此列表和活动组合执行三个不同的事务。其他两个列表（501 和 502）不生成任何销售值，因此只有查询为这些列表 ID 生成的行来自 LISTING 表 (SALESROW = 'No')。

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(6 rows)
```

如果运行不带 ALL 关键字的相同查询，则结果只保留其中一个销售交易。

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(4 rows)
```

示例 INTERSECT 查询

将以下示例与第一个 UNION 示例进行比较。这两个示例之间的唯一差异是所使用的集合运算符，但结果完全不同。仅其中一行相同：

```
235494 |    23875 |    8771
```

这是在包含 5 行的有限结果中，同时在两个表中找到的唯一行。

```
select listid, sellerid, eventid from listing
intersect
select listid, sellerid, eventid from sales
order by listid desc, sellerid, eventid
limit 5;

listid | sellerid | eventid
--------+----------+---------
235494 |    23875 |    8771
235482 |     1067 |    2667
235479 |     1589 |    7303
235476 |    15550 |     793
235475 |    22306 |    7848
(5 rows)
```

下面的查询查找 3 月份同时在纽约和洛杉矶举办的活动（已销售这些活动的门票）。这两个查询表达式之间的差异是 VENUECITY 列上的约束。

```
select distinct eventname from event, sales, venue
where event.eventid=sales.eventid and event.venueid=venue.venueid
and date_part(month,starttime)=3 and venuecity='Los Angeles'
intersect
select distinct eventname from event, sales, venue
where event.eventid=sales.eventid and event.venueid=venue.venueid
and date_part(month,starttime)=3 and venuecity='New York City'
order by eventname asc;

eventname
----------------------------
A Streetcar Named Desire
Dirty Dancing
Electra
Running with Annalise
```

```
Hairspray
Mary Poppins
November
Oliver!
Return To Forever
Rhinoceros
South Pacific
The 39 Steps
The Bacchae
The Caucasian Chalk Circle
The Country Girl
Wicked
Woyzeck
(16 rows)
```

示例 EXCEPT 查询

TICKIT 数据库中的 CATEGORY 表包含以下 11 行：

```
 catid | catgroup |  catname  |                  catdesc
-------+----------+-----------+--------------------------------------------
   1   | Sports   | MLB       | Major League Baseball
   2   | Sports   | NHL       | National Hockey League
   3   | Sports   | NFL       | National Football League
   4   | Sports   | NBA       | National Basketball Association
   5   | Sports   | MLS       | Major League Soccer
   6   | Shows    | Musicals  | Musical theatre
   7   | Shows    | Plays     | All non-musical theatre
   8   | Shows    | Opera     | All opera and light opera
   9   | Concerts | Pop       | All rock and pop music concerts
  10   | Concerts | Jazz      | All jazz singers and bands
  11   | Concerts | Classical | All symphony, concerto, and choir concerts
(11 rows)
```

假定 CATEGORY_STAGE 表（临时表）包含一个额外行：

```
 catid | catgroup |  catname  |                  catdesc
-------+----------+-----------+--------------------------------------------
1 | Sports   | MLB       | Major League Baseball
2 | Sports   | NHL       | National Hockey League
3 | Sports   | NFL       | National Football League
4 | Sports   | NBA       | National Basketball Association
5 | Sports   | MLS       | Major League Soccer
```

```
6 | Shows    | Musicals  | Musical theatre
7 | Shows    | Plays     | All non-musical theatre
8 | Shows    | Opera     | All opera and light opera
9 | Concerts | Pop       | All rock and pop music concerts
10 | Concerts | Jazz      | All jazz singers and bands
11 | Concerts | Classical | All symphony, concerto, and choir concerts
12 | Concerts | Comedy    | All stand up comedy performances
(12 rows)
```

返回两个表之间的差异。换而言之，返回 CATEGORY_STAGE 表中存在但 CATEGORY 表中不存在的行：

```
select * from category_stage
except
select * from category;

catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
12 | Concerts | Comedy  | All stand up comedy performances
(1 row)
```

以下等效查询使用同义词 MINUS。

```
select * from category_stage
minus
select * from category;

catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
12 | Concerts | Comedy  | All stand up comedy performances
(1 row)
```

如果反转 SELECT 表达式的顺序，则查询不返回任何行。

ORDER BY 子句

主题

- 语法
- 参数
- 使用说明
- 使用 ORDER BY 的示例

ORDER BY 子句对查询的结果集进行排序。

语法

```
[ ORDER BY expression [ ASC | DESC ] ]
[ NULLS FIRST | NULLS LAST ]
[ LIMIT { count | ALL } ]
[ OFFSET start ]
```

参数

expression

一个表达式，通常情况下通过指定选择列表中的一个或多个列，来定义查询结果集的排序顺序。根据二进制 UTF-8 排序方式返回结果。您也可以指定：

- 未在选择列表中的列
- 由查询引用的表中存在的一个或多个列构成的表达式
- 表示选择列表条目的位置（如果不存在选择列表，则为表中列的位置）的序号
- 定义选择列表条目的别名

当 ORDER BY 子句包含多个表达式时，将根据第一个表达式对结果集进行排序，然后将第二个表达式应用于具有第一个表达式中的匹配值的行，以此类推。

ASC | DESC

一个定义表达式的排序顺序的选项，如下所示：

- ASC：升序（例如，按数值的从低到高的顺序和字符串的从 A 到 Z 的顺序）。如果未指定选项，则默认情况下将按升序对数据进行排序。
- DESC：降序（按数值的从高到低的顺序和字符串的从 Z 到 A 的顺序）。

NULLS FIRST | NULLS LAST

一个选项，指定是应将 NULL 值排在最前（位于非 null 值之前）还是排在最后（位于非 null 值之后）。默认情况下，按 ASC 顺序最后对 NULL 值进行排序和排名，按 DESC 顺序首先对 NULL 值进行排序和排名。

LIMIT number | ALL

一个选项，用于控制查询返回的排序行的数目。LIMIT 数字必须为正整数；最大值为 2147483647。

LIMIT 0 不返回任何行。可以使用此语法进行测试：检查查询运行（不显示任何行）或返回表中列的列表。如果使用 LIMIT 0 返回列的列表，则 ORDER BY 子句是多余的。默认值为 LIMIT ALL。

OFFSET start

一个选项，指定在开始返回行之前跳过 start 前的行数。OFFSET 数字必须为正整数；最大值为 2147483647。在与 LIMIT 选项结合使用时，将先跳过 OFFSET 行，然后再开始计算返回的 LIMIT 行数。如果不使用 LIMIT 选项，则结果集中的行数会减少跳过的行数。仍必须扫描 OFFSET 子句跳过的行，因此使用较大的 OFFSET 值可能会非常低效。

使用说明

请注意，使用 ORDER BY 子句时预期会发生以下行为：

- NULL 值被视为“高于”所有其他值。对于默认的升序排序顺序，NULL 值将排在最后。要更改此行为，请使用 NULLS FIRST 选项。
- 当查询不包含 ORDER BY 子句时，系统将返回具有不可预测的行顺序的结果集。同一查询执行两次可能会返回具有不同顺序的结果集。
- 可在不使用 ORDER BY 子句的情况下使用 LIMIT 和 OFFSET 选项；不过，要返回一致的行集，请将这两个选项与 ORDER BY 子句结合使用。
- 在任何并行系统（例如 Amazon Redshift）中，当 ORDER BY 不生成唯一排序时，行的顺序是不确定的。也就是说，如果 ORDER BY 表达式生成重复值，则这些行的返回顺序可能会因系统或 Amazon Redshift 运行而异。
- Amazon Redshift 不支持 ORDER BY 子句中的字符串文本。

使用 ORDER BY 的示例

返回 CATEGORY 表中的所有 11 行，这些行按第二列 CATGROUP 进行排序。对于具有相同 CATGROUP 值的结果，按字符串长度对 CATDESC 列值进行排序。然后，按列 CATID 和 CATNAME 排序。

```
select * from category order by 2, length(catdesc), 1, 3;

catid | catgroup |  catname  |                 catdesc
------+----------+-----------+--------------------------------------
10    | Concerts | Jazz      | All jazz singers and bands
9     | Concerts | Pop       | All rock and pop music concerts
11    | Concerts | Classical | All symphony, concerto, and choir conce
```

```
6     | Shows    | Musicals  | Musical theatre
7     | Shows    | Plays     | All non-musical theatre
8     | Shows    | Opera     | All opera and light opera
5     | Sports   | MLS       | Major League Soccer
1     | Sports   | MLB       | Major League Baseball
2     | Sports   | NHL       | National Hockey League
3     | Sports   | NFL       | National Football League
4     | Sports   | NBA       | National Basketball Association
(11 rows)
```

返回 SALES 表中的选定列（按最高的 QTYSOLD 值排序）。将结果限制为前 10 行：

```
select salesid, qtysold, pricepaid, commission, saletime from sales
order by qtysold, pricepaid, commission, salesid, saletime desc
limit 10;

salesid | qtysold | pricepaid | commission |      saletime
--------+---------+-----------+------------+---------------------
15401   |       8 |    272.00 |      40.80 | 2008-03-18 06:54:56
61683   |       8 |    296.00 |      44.40 | 2008-11-26 04:00:23
90528   |       8 |    328.00 |      49.20 | 2008-06-11 02:38:09
74549   |       8 |    336.00 |      50.40 | 2008-01-19 12:01:21
130232  |       8 |    352.00 |      52.80 | 2008-05-02 05:52:31
55243   |       8 |    384.00 |      57.60 | 2008-07-12 02:19:53
16004   |       8 |    440.00 |      66.00 | 2008-11-04 07:22:31
489     |       8 |    496.00 |      74.40 | 2008-08-03 05:48:55
4197    |       8 |    512.00 |      76.80 | 2008-03-23 11:35:33
16929   |       8 |    568.00 |      85.20 | 2008-12-19 02:59:33
(10 rows)
```

通过使用 LIMIT 0 语法返回列的列表，但不返回行：

```
select * from venue limit 0;
venueid | venuename | venuecity | venuestate | venueseats
---------+-----------+-----------+------------+------------
(0 rows)
```

CONNECT BY 子句

CONNECT BY 子句指定层次结构中行之间的关系。您可以使用 CONNECT BY 按层次结构顺序选择行，方法是将表联接到表本身并处理分层数据。例如，您可以使用它以递归方式循环浏览组织结构图和列出数据。

分层查询按以下顺序处理：

1. 如果 FROM 子句有联接，则首先对其进行处理。
2. 对 CONNECT BY 子句求值。
3. 对 WHERE 子句求值。

语法

```
[START WITH start_with_conditions]
CONNECT BY connect_by_conditions
```

> **Note**
>
> 虽然 START 和 CONNECT 不是保留字，但如果您在查询中使用 START 和 CONNECT 作为表别名，请使用分隔标识符（双引号）或 AS，以避免在运行时失败。

```
SELECT COUNT(*)
FROM Employee "start"
CONNECT BY PRIOR id = manager_id
START WITH name = 'John'
```

```
SELECT COUNT(*)
FROM Employee AS start
CONNECT BY PRIOR id = manager_id
START WITH name = 'John'
```

参数

start_with_conditions

指定层次结构根行的条件

connect_by_conditions

指定层次结构的父行和子行之间关系的条件。必须使用用于引用父行的 一元运算符限定至少一个条件。

```
PRIOR column = expression
-- or
```

```
expression > PRIOR column
```

运算符

您可以在 CONNECT BY 查询中使用以下运算符。

LEVEL

返回层次结构中当前行级别的伪列。为根行返回 1，为根行的子行返回 2，依此类推。

PRIOR

一元运算符，用于计算层次结构中当前行的父行的表达式。

示例

以下示例是 CONNECT BY 查询，该示例返回直接或间接向 John 报告的员工数量，不超过 4 个级别。

```
SELECT id, name, manager_id
FROM employee
WHERE LEVEL < 4
START WITH name = 'John'
CONNECT BY PRIOR id = manager_id;
```

以下是查询的结果。

```
id      name      manager_id
------+----------+--------------
  101     John        100
  102     Jorge       101
  103     Kwaku       101
  110     Liu         101
  201     Sofía       102
  106     Mateo       102
  110     Nikki       103
  104     Paulo       103
  105     Richard     103
  120     Saanvi      104
  200     Shirley     104
  205     Zhang       104
```

此示例的表定义：

```
CREATE TABLE employee (
   id INT,
   name VARCHAR(20),
   manager_id INT
   );
```

以下是插入到表中的行。

```
INSERT INTO employee(id, name, manager_id)  VALUES
(100, 'Carlos', null),
(101, 'John', 100),
(102, 'Jorge', 101),
(103, 'Kwaku', 101),
(110, 'Liu', 101),
(106, 'Mateo', 102),
(110, 'Nikki', 103),
(104, 'Paulo', 103),
(105, 'Richard', 103),
(120, 'Saanvi', 104),
(200, 'Shirley', 104),
(201, 'Sofía', 102),
(205, 'Zhang', 104);
```

以下是 John 所在部门的组织结构图。

子查询示例

以下示例说明子查询适合 SELECT 查询的不同方式。有关使用子查询的另一个示例，请参阅JOIN 示例。

SELECT 列表子查询

以下示例在 SELECT 列表中包含一个子查询。此子查询是标量：它只返回一列和一个值，该值将在从外部查询返回的每个行的结果中重复。此查询将子查询计算出的 Q1SALES 值与外部查询定义的 2008 年其他两个季度（第 2 季度和第 3 季度）的销售值进行比较。

```
select qtr, sum(pricepaid) as qtrsales,
(select sum(pricepaid)
from sales join date on sales.dateid=date.dateid
where qtr='1' and year=2008) as q1sales
from sales join date on sales.dateid=date.dateid
```

```
where qtr in('2','3') and year=2008
group by qtr
order by qtr;

qtr  |  qtrsales   |   q1sales
-------+-------------+-------------
2     | 30560050.00 | 24742065.00
3     | 31170237.00 | 24742065.00
(2 rows)
```

WHERE 子句子查询

以下示例在 WHERE 子句中包含一个表子查询。此子查询生成多个行。在本示例中，行只包含一列，但表子查询可以包含多个列和行，就像任何其他表一样。

此查询查找门票销量排名前 10 位的卖家。前 10 位卖家的列表受子查询的限制，这将删除居住在设有售票点的城市的用户。可以使用不同的方式编写此查询；例如，可将子查询重新编写为主查询中的联接。

```
select firstname, lastname, city, max(qtysold) as maxsold
from users join sales on users.userid=sales.sellerid
where users.city not in(select venuecity from venue)
group by firstname, lastname, city
order by maxsold desc, city desc
limit 10;

firstname | lastname  |      city      | maxsold
-----------+-----------+----------------+---------
Noah       | Guerrero | Worcester      |       8
Isadora    | Moss     | Winooski       |       8
Kieran     | Harrison | Westminster    |       8
Heidi      | Davis    | Warwick        |       8
Sara       | Anthony  | Waco           |       8
Bree       | Buck     | Valdez         |       8
Evangeline | Sampson  | Trenton        |       8
Kendall    | Keith    | Stillwater     |       8
Bertha     | Bishop   | Stevens Point  |       8
Patricia   | Anderson | South Portland |       8
(10 rows)
```

WITH 子句子查询

请参阅 WITH 子句。

关联的子查询

以下示例将关联子查询 包含在 WHERE 子句中；此类型的子查询包含其列与由外部查询生成的列之间的一个或多个关联。在本示例中，关联为 `where s.listid=l.listid`。对于外部查询生成的每一行，将执行子查询以限定或取消限定行。

```
select salesid, listid, sum(pricepaid) from sales s
where qtysold=
(select max(numtickets) from listing l
where s.listid=l.listid)
group by 1,2
order by 1,2
limit 5;

salesid | listid |   sum
--------+--------+----------
 27     |     28 | 111.00
 81     |    103 | 181.00
 142    |    149 | 240.00
 146    |    152 | 231.00
 194    |    210 | 144.00
(5 rows)
```

不支持的关联子查询模式

查询计划程序使用名为“子查询去相关性”的查询重写方法来优化多个关联子查询模式以便在 MPP 环境中执行。有多种类型的关联子查询采用 Amazon Redshift 无法去相关性且不支持的模式。包含以下关联引用的查询会返回错误：

- 跳过查询块的关联引用，也称为“跨级关联引用”。例如，在以下查询中，包含关联引用的块与跳过的块由 NOT EXISTS 谓词连接：

  ```
  select event.eventname from event
  where not exists
  (select * from listing
  where not exists
  (select * from sales where event.eventid=sales.eventid));
  ```

  在本示例中，跳过的块是针对 LISTING 表执行的子查询。关联引用将 EVENT 表和 SALES 表关联起来。
- 来自作为外部查询中 ON 子句的一部分的子查询的关联引用：

```
select * from category
left join event
on category.catid=event.catid and eventid =
(select max(eventid) from sales where sales.eventid=event.eventid);
```

ON 子句包含从子查询中的 SALES 到外部查询中的 EVENT 的关联引用。

- 针对 Amazon Redshift 系统表的 Null 敏感型关联引用。例如：

```
select attrelid
from stv_locks sl, pg_attribute
where sl.table_id=pg_attribute.attrelid and 1 not in
(select 1 from pg_opclass where sl.lock_owner = opcowner);
```

- 来自包含窗口函数的子查询内部的关联引用。

```
select listid, qtysold
from sales s
where qtysold not in
(select sum(numtickets) over() from listing l where s.listid=l.listid);
```

- GROUP BY 列中对关联查询结果的引用。例如：

```
select listing.listid,
(select count (sales.listid) from sales where sales.listid=listing.listid) as list
from listing
group by list, listing.listid;
```

- 来自带聚合函数和 GROUP BY 子句（通过 IN 谓词连接到外部查询）的关联引用。（此限制不适用于 MIN 和 MAX 聚合函数。）例如：

```
select * from listing where listid in
(select sum(qtysold)
from sales
where numtickets>4
group by salesid);
```

SELECT INTO

选择任何查询所定义的行，并将这些行插入到新表中。您可以指定是创建临时表还是永久表。

语法

```
[ WITH with_subquery [, ...] ]
SELECT
[ TOP number ] [ ALL | DISTINCT ]
* | expression [ AS output_name ] [, ...]
INTO [ TEMPORARY | TEMP ] [ TABLE ] new_table
[ FROM table_reference [, ...] ]
[ WHERE condition ]
[ GROUP BY expression [, ...] ]
[ HAVING condition [, ...] ]
[ { UNION | INTERSECT | { EXCEPT | MINUS } } [ ALL ] query ]
[ ORDER BY expression
[ ASC | DESC ]
[ LIMIT { number | ALL } ]
[ OFFSET start ]
```

有关此命令的参数的详细信息，请参阅SELECT。

示例

选择 EVENT 表中的所有行并创建 NEWEVENT 表：

```
select * into newevent from event;
```

选择聚合查询的结果并将这些结果插入到名为 PROFITS 的临时表中：

```
select username, lastname, sum(pricepaid-commission) as profit
into temp table profits
from sales, users
where sales.sellerid=users.userid
group by 1, 2
order by 3 desc;
```

SET

设置服务器配置参数的值。使用 SET 命令仅覆盖当前会话或事务持续时间的设置。

使用 RESET 命令将参数还原为其默认值。

您可以通过多种方式更改服务器配置参数。有关更多信息，请参阅 修改服务器配置。

语法

```
SET { [ SESSION | LOCAL ]
{ SEED | parameter_name } { TO | = }
{ value | 'value' | DEFAULT } |
SEED TO value }
```

以下语句将设置会话上下文变量的值。

```
SET { [ SESSION | LOCAL ]
variable_name { TO | = }
{ value | 'value'  }
```

参数

SESSION

指定设置对当前会话有效。默认值。

variable_name

指定为会话设置的上下文变量的名称。

命名约定是一个用点分隔的、包含两个部分的名称，例如 identifier.identifier。只允许使用一个点分隔符。使用遵循 Amazon Redshift 的标准标识符规则的标识符。有关更多信息，请参阅名称和标识符。不允许使用分隔标识符。

LOCAL

指定设置对当前事务有效。

SEED TO value

设置由 RANDOM 函数用于生成随机数的内部种子。

SET SEED 采用介于 0 和 1 之间的数值，并将此数乘以 (2^{31}-1) 以用于 RANDOM 函数 函数。如果在多次调用 RANDOM 之前使用 SET SEED，则 RANDOM 会按可预测的顺序生成数字。

parameter_name

要设置的参数的名称。有关参数的信息，请参阅修改服务器配置。

值

新的参数值。使用单引号将值设置为特定字符串。如果使用 SET SEED，则此参数包含 SEED 值。

DEFAULT

将参数设置为默认值。

示例

更改当前会话的参数

以下示例设置日期样式：

```
set datestyle to 'SQL,DMY';
```

设置工作负载管理的查询组

如果查询组在队列定义中作为集群的 WLM 配置的一部分列出，则可将 QUERY_GROUP 参数设置为列出的查询组名称。后续查询将分配给关联的查询队列。QUERY_GROUP 设置在会话的持续时间内或遇到 RESET QUERY_GROUP 命令之前保持有效。

此示例将两个查询作为查询组“priority”的一部分运行，然后重置查询组。

```
set query_group to 'priority';
select tbl, count(*)from stv_blocklist;
select query, elapsed, substring from svl_qlog order by query desc limit 5;
reset query_group;
```

有关更多信息，请参阅 实施工作负载管理。

更改会话的默认身份命名空间

数据库用户可以设置 default_identity_namespace。此示例说明如何使用 SET SESSION 来覆盖当前会话持续时间的设置，然后显示新的身份提供者值。当您将身份提供者与 Redshift 和 IAM Identity Center 结合使用时，最常使用此方法。有关将身份提供者与 Redshift 结合使用的更多信息，请参阅将 Redshift 与 IAM Identity Center 连接，为用户提供单点登录体验。

```
SET SESSION default_identity_namespace = 'MYCO';

SHOW default_identity_namespace;
```

运行命令后，您可以运行 GRANT 语句或 CREATE 语句，如下所示：

```
GRANT SELECT ON TABLE mytable TO alice;
```

```
GRANT UPDATE ON TABLE mytable TO salesrole;

CREATE USER bob password 'md50c983d1a624280812631c5389e60d48c';
```

在这种情况下，设置默认身份命名空间的效果等同于为每个身份添加命名空间前缀。在本例中，`alice` 替换为 `MYCO:alice`。有关与 IAM Identity Center 的 Redshift 配置相关的设置的更多信息，请参阅 ALTER SYSTEM 和 ALTER IDENTITY PROVIDER。

设置查询组的标签

QUERY_GROUP 参数为在同一个会话中 SET 命令之后执行的一个或多个查询定义标签。反过来，在执行查询并可将其用于约束从 STL_QUERY 和 STV_INFLIGHT 系统表以及 SVL_QLOG 视图返回的结果时，将记录此标签。

```
show query_group;
query_group
-------------
unset
(1 row)

set query_group to '6 p.m.';


show query_group;
query_group
-------------
6 p.m.
(1 row)

select * from sales where salesid=500;
salesid | listid | sellerid | buyerid | eventid | dateid | ...
---------+--------+----------+---------+---------+--------+-----
500 |    504 |     3858 |    2123 |    5871 |   2052 | ...
(1 row)

reset query_group;

select query, trim(label) querygroup, pid, trim(querytxt) sql
from stl_query
where label ='6 p.m.';
query | querygroup |  pid  |                  sql
-------+------------+-------+----------------------------------------
```

```
57 | 6 p.m.     | 30711 | select * from sales where salesid=500;
(1 row)
```

查询组标签是一个非常有用的机制，可用于隔离作为脚本一部分运行的单个查询或查询组。您不需要通过查询 ID 来标识和跟踪查询；而可以通过查询的标签来跟踪查询。

设置用于生成随机数的种子值

以下示例将 SEED 选项与 SET 结合使用，使 RANDOM 函数按可预测的顺序生成数字。

首先，返回三个 RANDOM 整数，而不先设置 SEED 值：

```
select cast (random() * 100 as int);
int4
------
6
(1 row)

select cast (random() * 100 as int);
int4
------
68
(1 row)

select cast (random() * 100 as int);
int4
------
56
(1 row)
```

现在，将 SEED 值设置为 .25，并返回 3 个以上的 RANDOM 数字：

```
set seed to .25;

select cast (random() * 100 as int);
int4
------
21
(1 row)

select cast (random() * 100 as int);
int4
------
```

```
79
(1 row)

select cast (random() * 100 as int);
int4
------
12
(1 row)
```

最后，将 SEED 值重置为 .25，并验证 RANDOM 是否返回与前三个调用相同的结果：

```
set seed to .25;

select cast (random() * 100 as int);
int4
------
21
(1 row)

select cast (random() * 100 as int);
int4
------
79
(1 row)

select cast (random() * 100 as int);
int4
------
12
(1 row)
```

以下示例设置自定义上下文变量。

```
SET app_context.user_id TO 123;
SET app_context.user_id TO 'sample_variable_value';
```

SET SESSION AUTHORIZATION

设置当前会话的用户名。

您可以使用 SET SESSION AUTHORIZATION 命令以非特权用户身份临时运行会话或事务，来测试数据库访问。您必须是数据库超级用户才能执行此命令。

语法

```
SET [ LOCAL ] SESSION AUTHORIZATION { user_name | DEFAULT }
```

参数

LOCAL

指定设置对当前事务有效。忽略此参数将指定设置对当前会话有效。

user_name

要设置的用户的名称。可以使用标识符或字符串文本的形式来编写用户名。

DEFAULT

将会话用户名设置为默认值。

示例

以下示例将当前会话的用户名设置为 dwuser:

```
SET SESSION AUTHORIZATION 'dwuser';
```

以下示例将当前事务的用户名设置为 dwuser:

```
SET LOCAL SESSION AUTHORIZATION 'dwuser';
```

此示例将当前会话的用户名设置为默认用户名 :

```
SET SESSION AUTHORIZATION DEFAULT;
```

SET SESSION CHARACTERISTICS

此命令已被弃用。

SHOW

显示服务器配置参数的当前值。如果 SET 命令生效，此值可以是特定于当前会话的值。有关配置参数的列表，请参阅配置参考。

语法

```
SHOW { parameter_name | ALL }
```

以下语句显示会话上下文变量的当前值。如果变量不存在，Amazon Redshift 会引发错误。

```
SHOW variable_name
```

参数

parameter_name

显示指定参数的当前值。

ALL

显示所有参数的当前值。

variable_name

显示指定变量的当前值。

示例

以下示例显示 query_group 参数的值：

```
show query_group;

query_group

unset
(1 row)
```

以下示例显示所有参数及其值的列表：

```
show all;
name        |   setting
--------------------+--------------
datestyle          | ISO, MDY
extra_float_digits | 0
query_group        | unset
```

```
search_path        | $user,public
statement_timeout  | 0
```

以下示例显示指定变量的当前值。

```
SHOW app_context.user_id;
```

SHOW COLUMNS

显示表中列的列表以及一些列属性。

每个输出行由以逗号分隔的数据库名称、架构名称、表名、列名、序号位置、列默认值、可为空、数据类型、字符最大长度、数字精度和注释的列表组成。有关这些属性的更多信息，请参阅 SVV_ALL_COLUMNS。

如果 SHOW COLUMNS 命令生成的列数超过 10000 列，则会返回错误。

语法

```
SHOW COLUMNS FROM TABLE database_name.schema_name.table_name [LIKE 'filter_pattern']
 [LIMIT row_limit ]
```

参数

database_name

包含要列出的表的数据库的名称。

要显示 AWS Glue Data Catalog 中的表，请指定（awsdatacatalog）作为数据库名称，并确保系统配置 data_catalog_auto_mount 设置为 true。有关更多信息，请参阅 ALTER SYSTEM。

schema_name

包含要列出的表的架构的名称。

要显示 AWS Glue Data Catalog 表，请提供 AWS Glue 数据库名称作为架构名称。

table_name

包含要列出的列的表的名称。

filter_pattern

一个有效的 UTF-8 字符表达式，具有与表名称匹配的模式。LIKE 选项执行区分大小写的匹配，支持以下模式匹配元字符：

元字符	描述
%	匹配任意序列的零个或多个字符。
_	匹配任何单个字符。

如果 filter_pattern 不包含元字符，则模式仅表示字符串本身；在此情况下，LIKE 的行为与等于运算符相同。

row_limit

要返回的最大行数。row_limit 可以是 0–10000。

示例

以下示例显示了名为 dev 的 Amazon Redshift 数据库中的列，这些列位于架构 public 和表 tb 中。

```
SHOW COLUMNS FROM TABLE dev.public.tb;

 database_name | schema_name | table_name | column_name | ordinal_position
 | column_default | is_nullable | data_type | character_maximum_length |
 numeric_precision | remarks
---------------+-------------+------------+-------------+------------------
+-----------------+-------------+-----------+--------------------------
+-------------------+---------
 dev           | public      | tb         | col         |                1 |
     | YES         | integer   |                           |                32 |
```

以下示例显示了名为 awsdatacatalog 的 AWS Glue Data Catalog 数据库中的表，这些表位于架构 batman 和表 nation 中。输出仅限于 2 行。

```
SHOW COLUMNS FROM TABLE awsdatacatalog.batman.nation LIMIT 2;

 database_name  | schema_name | table_name | column_name | ordinal_position
 | column_default | is_nullable | data_type | character_maximum_length |
 numeric_precision | remarks
```

```
---------------+-------------+-----------+-------------+-----------------
+---------------+-------------+----------+-------------------------
+-------------------+---------
 awsdatacatalog | batman      | nation     | n_nationkey |               1 |
      |             | integer   |                         |                 |
 awsdatacatalog | batman      | nation     | n_name      |               2 |
      |             | character |                         |                 |
```

SHOW EXTERNAL TABLE

显示外部表的定义，包括表属性和列属性。您可以使用 SHOW EXTERNAL TABLE 语句的输出来重新创建表。

有关外部表创建的更多信息，请参阅CREATE EXTERNAL TABLE。

语法

```
SHOW EXTERNAL TABLE [external_database].external_schema.table_name [ PARTITION ]
```

参数

external_database

关联的外部数据库的名称。此参数为可选的。

external_schema

关联的外部 schema 的名称。

table_name

要显示的表的名称。

PARTITION

显示 ALTER TABLE 语句以将分区添加到表定义。

示例

以下示例基于定义如下的外部表：

```
CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
```

```
     csmallint smallint,
     cint int,
     cbigint bigint,
     cfloat float4,
     cdouble float8,
     cchar char(10),
     cvarchar varchar(255),
     cdecimal_small decimal(18,9),
     cdecimal_big decimal(30,15),
     ctimestamp TIMESTAMP,
     cboolean boolean,
     cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime TIMESTAMP)
STORED AS PARQUET
LOCATION 's3://mybucket-test-copy/alldatatypes_parquet_partitioned';
```

以下是 SHOW EXTERNAL TABLE 命令和表 my_schema.alldatatypes_parquet_test_partitioned 的输出示例。

```
SHOW EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned;
```

```
"CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
    csmallint smallint,
    cint int,
    cbigint bigint,
    cfloat float4,
    cdouble float8,
    cchar char(10),
    cvarchar varchar(255),
    cdecimal_small decimal(18,9),
    cdecimal_big decimal(30,15),
    ctimestamp timestamp,
    cboolean boolean,
    cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime timestamp)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://mybucket-test-copy/alldatatypes_parquet_partitioned';"
```

以下是同一个表的 SHOW EXTERNAL TABLE 命令和输出的示例，但参数中也指定了数据库。

```
SHOW EXTERNAL TABLE my_database.my_schema.alldatatypes_parquet_test_partitioned;
```

```
"CREATE EXTERNAL TABLE my_database.my_schema.alldatatypes_parquet_test_partitioned (
    csmallint smallint,
    cint int,
    cbigint bigint,
    cfloat float4,
    cdouble float8,
    cchar char(10),
    cvarchar varchar(255),
    cdecimal_small decimal(18,9),
    cdecimal_big decimal(30,15),
    ctimestamp timestamp,
    cboolean boolean,
    cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime timestamp)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://mybucket-test-copy/alldatatypes_parquet_partitioned';"
```

以下是使用 PARTITION 参数时的 SHOW EXTERNAL TABLE 命令和输出的示例。输出包含 ALTER TABLE 语句，可用于将分区添加到表定义。

```
SHOW EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned PARTITION;
```

```
"CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
    csmallint smallint,
    cint int,
    cbigint bigint,
    cfloat float4,
    cdouble float8,
    cchar char(10),
    cvarchar varchar(255),
    cdecimal_small decimal(18,9),
    cdecimal_big decimal(30,15),
    ctimestamp timestamp,
    cboolean boolean,
```

```
    cstring varchar(16383)
)
PARTITIONED BY (cdate date)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://mybucket-test-copy/alldatatypes_parquet_partitioned';
ALTER TABLE my_schema.alldatatypes_parquet_test_partitioned ADD IF NOT
 EXISTS PARTITION (cdate='2021-01-01') LOCATION 's3://mybucket-test-copy/
alldatatypes_parquet_partitioned2/cdate=2021-01-01';
ALTER TABLE my_schema.alldatatypes_parquet_test_partitioned ADD IF NOT
 EXISTS PARTITION (cdate='2021-01-02') LOCATION 's3://mybucket-test-copy/
alldatatypes_parquet_partitioned2/cdate=2021-01-02';"
```

SHOW DATABASES

显示来自指定账户 ID 的数据库。

语法

```
SHOW DATABASES FROM
DATA CATALOG [ ACCOUNT  '<id1>', '<id2>', ... ]
[ LIKE '<expression>' ]
[ IAM_ROLE default | 'SESSION' | 'arn:aws:iam::<account-id>:role/<role-name>' ]
```

参数

ACCOUNT '<id1>', '<id2>', ...

要从中列出数据库的 AWS Glue Data Catalog 账户。省略此参数将指示 Amazon Redshift 应显示拥有该集群的账户中的数据库。

LIKE '<expression>'

从数据库列表中筛选出那些与您指定的表达式匹配的数据库。此参数支持使用通配符 %（百分号）和 _（下划线）的模式。

IAM_ROLE default | 'SESSION' | 'arn:aws:iam::<account-id>:role/<role-name>'

如果您在运行 SHOW DATABASES 命令时指定与集群关联的 IAM 角色，则 Amazon Redshift 将在您对数据库运行查询时使用该角色的凭证。

指定 `default` 关键字表示要使用设置为默认并与集群关联的 IAM 角色。

如果您使用联合身份连接到 Amazon Redshift 集群并访问使用 the section called “CREATE DATABASE” 命令创建的外部数据库中的表，则使用 'SESSION'。有关使用联合身份的示例，请参阅使用联合身份管理 Amazon Redshift 对本地资源和 Amazon Redshift Spectrum 外部表的访问权限，其中说明了如何配置联合身份。

使用 IAM 角色的 Amazon 资源名称（ARN），您的集群使用该角色进行身份验证和授权。IAM 角色至少必须有权在要访问的 Amazon S3 桶上执行 LIST 操作和有权在该桶包含的 Amazon S3 对象上执行 GET 操作。要了解有关从 AWS Glue Data Catalog 中为数据共享创建并使用 IAM_ROLE 的数据库的更多信息，请参阅以使用者身份使用 Lake Formation 托管式数据共享。

下面显示了单个 ARN 的 IAM_ROLE 参数字符串的语法。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

您可以将角色串联起来，以便集群可以承担另一个 IAM 角色 (可能属于其他账户)。您最多可串联 10 个角色。有关更多信息，请参阅 在 Amazon Redshift Spectrum 中链接 IAM 角色。

对于此 IAM 角色，请附加类似于以下内容的 IAM 权限策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

有关创建 IAM 角色以用于联合查询的步骤，请参阅[创建密钥和 IAM 角色以使用联合查询](#)。

> **Note**
>
> 请不要在链接的角色列表中包含空格。

下面显示了串联三个角色的语法。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-
account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

示例

以下示例显示了来自账户 ID 123456789012 的所有 Data Catalog 数据库。

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012'

  catalog_id  | database_name |                          database_arn
 |     type     |                                        target_database
                            | location | parameters
--------------+---------------+-----------------------------------------------------------
+--------------
+-------------------------------------------------------------------------------------------
+----------+------------
 123456789012 |   database1   | arn:aws:glue:us-east-1:123456789012:database/database1
 | Data Catalog |
                            |          |
 123456789012 |   database2   | arn:aws:glue:us-east-1:123456789012:database/database2
 | Data Catalog | arn:aws:redshift:us-
east-1:123456789012:datashare:035c45ea-61ce-86f0-8b75-19ac6102c3b7/database2 |
 |
```

以下示例演示了如何在使用 IAM 角色的凭证时显示账户 ID 123456789012 中的所有数据目录数据库。

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012' IAM_ROLE default;
```

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012' IAM_ROLE <iam-role-arn>;
```

SHOW MODEL

显示有关机器学习模型的有用信息，包括其状态、用于创建模型的参数以及具有输入参数类型的预测函数。可以使用 SHOW MODEL 中的信息重新创建模型。如果基表已更改，则使用相同的 SQL 语句运行 CREATE MODEL 会导致生成不同的模型。SHOW MODEL 返回的信息对于模型拥有者和具有 EXECUTE 权限的用户而言是不同的。当模型从 Amazon Redshift 中进行训练或模型为 BYOM 模型时，SHOW MODEL 会显示不同的输出。

语法

```
SHOW MODEL ( ALL | model_name )
```

参数

ALL

返回用户可以使用的所有模型及其 schema。

model_name

模型的名称。schema 中的模型名称必须是唯一的。

使用说明

SHOW MODEL 命令将返回以下内容：

- 模型名称。
- 创建模型所在的 schema。
- 模型的拥有者。
- 模型创建时间。
- 模型的状态，如 READY、TRAINING 或 FAILED。

- 模型失败的原因消息。
- 如果模型已完成训练，则会出现验证错误。
- 为非 BYOM 方法派生模型所需的估计成本。只有模型的拥有者可查看此信息。
- 用户指定的参数及其值的列表，特别是以下内容：
  - 指定的 TARGET 列。
  - 模型类型，AUTO 或 XGBoost。
  - 问题类型，例如 REGRESSION、BINARY_CLASSIFICATION、MULTICLASS_CLASSIFICATION。此参数特定于 AUTO。
  - Amazon SageMaker 训练工作或创建模型的 Amazon SageMaker Autopilot 任务的名称。您可以使用此任务名称在 Amazon SageMaker 上查找有关该模型的更多信息。
  - 目标，如 MSE、F1、精度。此参数特定于 AUTO。
  - 所创建的函数的名称。
  - 推理的类型，本地或远程。
  - 预测函数输入参数。
  - 非自带模型 (BYOM) 的预测函数输入参数类型。
  - 预测函数的返回类型。此参数特定于 BYOM。
  - 具有远程推理功能的 BYOM 模型的 Amazon SageMaker 端点的名称。
  - IAM 角色。只有模型的拥有者可以看到此内容。
  - 所用的 S3 桶。只有模型的拥有者可以看到此内容。
  - AWS KMS 键（如果提供了一个）。只有模型的拥有者可以看到此内容。
  - 模型可以运行的最长时间。
- 如果模型类型不是 AUTO，则 Amazon Redshift 还会显示提供的超参数列表及其值。

您还可以在其他目录表（如 pg_proc）中查看 SHOW MODEL 提供的一些信息。Amazon Redshift 返回有关在 pg_proc 目录表中注册的预测函数的信息。此信息包括预测函数的输入参数名称及其类型。Amazon Redshift 会在 SHOW MODEL 命令中返回相同的信息。

```
SELECT * FROM pg_proc WHERE proname ILIKE '%<function_name>%';
```

示例

以下示例显示了显示模型输出。

```
SHOW MODEL ALL;

Schema Name |  Model Name
------------+---------------
 public     | customer_churn
```

customer_churn 的拥有者可查看以下输出。仅具有 EXECUTE 权限的用户无法看到 IAM 角色、Amazon S3 桶和模式的估计成本。

```
SHOW MODEL customer_churn;

       Key                  |           Value
----------------------------+-----------------------------------
 Model Name                 | customer_churn
 Schema Name                | public
 Owner                      | 'owner'
 Creation Time              | Sat, 15.01.2000 14:45:20
 Model State                | READY
 validation:F1              | 0.855
 Estimated Cost             | 5.7
                            |
 TRAINING DATA:             |
 Table                      | customer_data
 Target Column              | CHURN
                            |
 PARAMETERS:                |
 Model Type                 | auto
 Problem Type               | binary_classification
 Objective                  | f1
 Function Name              | predict_churn
 Function Parameters        | age zip average_daily_spend average_daily_cases
 Function Parameter Types   | int int float float
 IAM Role                   | 'iam_role'
 KMS Key                    | 'kms_key'
 Max Runtime                | 36000
```

SHOW DATASHARES

显示集群中来自同一账户或各账户间的入站和出站共享。如果您未指定数据共享名称，则 Amazon Redshift 会显示集群中所有数据库中的所有数据共享。具有 ALTER 和 SHARE 权限的用户可以看到他们对其拥有权限的共享。

语法

```
SHOW DATASHARES [ LIKE 'namepattern' ]
```

参数

LIKE

一个可选子句，用于将指定的名称模式与数据共享的描述进行比较。使用此子句时，Amazon Redshift 仅显示名称与指定名称模式匹配的数据共享。

namepattern

请求的数据共享的名称或要使用通配符匹配的名称的一部分。

示例

以下示例显示集群中的入站和出站共享。

```
SHOW DATASHARES;
SHOW DATASHARES LIKE 'sales%';

share_name   | share_owner | source_database | consumer_database | share_type |
 createdate          | is_publicaccessible | share_acl | producer_account |
 producer_namespace
-------------+-------------+-----------------+-------------------+------------
+---------------------+---------------------+-----------+------------------
+--------------------------------------
'salesshare' | 100         | dev             |                   | outbound
 | 2020-12-09 01:22:54.| False               |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

SHOW PROCEDURE

显示给定存储过程的定义，包括其签名。您可以使用 SHOW PROCEDURE 的输出来重新创建存储过程。

语法

```
SHOW PROCEDURE sp_name [( [ [ argname ] [ argmode ] argtype [, ...] ] )]
```

参数

sp_name

要显示的过程的名称。

[argname] [argmode] argtype

用于标识存储过程的输入参数类型。（可选）您可以包含完整的参数数据类型，包括 OUT 参数。如果存储过程名称是唯一的（即未重载），则此部分是可选的。

示例

以下示例显示 test_spl2 过程的定义。

```
show procedure test_sp2(int, varchar);
                                        Stored Procedure Definition
-------------------------------------------------------------------------------------------------
CREATE OR REPLACE PROCEDURE public.test_sp2(f1 integer, INOUT f2 character varying, OUT
 character varying)
LANGUAGE plpgsql
AS $_$
DECLARE
out_var alias for $3;
loop_var int;
BEGIN
IF f1 is null OR f2 is null THEN
RAISE EXCEPTION 'input cannot be null';
END IF;
CREATE TEMP TABLE etl(a int, b varchar);
FOR loop_var IN 1..f1 LOOP
insert into etl values (loop_var, f2);
f2 := f2 || '+' || f2;
END LOOP;
SELECT INTO out_var count(*) from etl;
END;
$_$

(1 row)
```

SHOW SCHEMAS

显示数据库中的架构列表以及一些架构属性。

每个输出行都包含数据库名称、架构名称、架构所有者、架构类型、架构 ACL、源数据库和架构选项。有关这些属性的更多信息，请参阅 SVV_ALL_SCHEMAS。

如果 SHOW SCHEMAS 命令可以生成超过 10000 个架构，则会返回错误。

语法

```
SHOW SCHEMAS FROM DATABASE database_name [LIKE 'filter_pattern'] [LIMIT row_limit ]
```

参数

database_name

包含要列出的表的数据库的名称。

要显示 AWS Glue Data Catalog 中的表，请指定（awsdatacatalog）作为数据库名称，并确保系统配置 data_catalog_auto_mount 设置为 true。有关更多信息，请参阅 ALTER SYSTEM。

filter_pattern

一个有效的 UTF-8 字符表达式，具有与架构名称匹配的模式。LIKE 选项执行区分大小写的匹配，支持以下模式匹配元字符：

元字符	描述
%	匹配任意序列的零个或多个字符。
_	匹配任何单个字符。

如果 filter_pattern 不包含元字符，则模式仅表示字符串本身；在此情况下，LIKE 的行为与等于运算符相同。

row_limit

要返回的最大行数。row_limit 可以是 0–10000。

示例

以下示例显示了来自名为 dev 的 Amazon Redshift 数据库的架构。

```
SHOW SCHEMAS FROM DATABASE dev;

 database_name |     schema_name      | schema_owner | schema_type |         schema_acl
          | source_database | schema_option
---------------+----------------------+--------------+-------------
+----------------------------+-----------------+---------------
 dev           | pg_automv            |            1 | local       |
          |                 |
 dev           | pg_catalog           |            1 | local       | jpuser=UC/
jpuser~=U/jpuser  |                 |
 dev           | public               |            1 | local       | jpuser=UC/
jpuser~=UC/jpuser |                 |
 dev           | information_schema   |            1 | local       | jpuser=UC/
jpuser~=U/jpuser  |                 |
 dev           | schemad79cd6d93bf043 |            1 | local       |
          |                 |
```

以下示例显示了名为 awsdatacatalog 的 AWS Glue Data Catalog 数据库中的架构。最大输出行数为 5。

```
SHOW SCHEMAS FROM DATABASE awsdatacatalog LIMIT 5;

 database_name  |     schema_name      | schema_owner | schema_type | schema_acl |
 source_database | schema_option
----------------+----------------------+--------------+-------------+------------
+-----------------+---------------
 awsdatacatalog | 000_too_many_glue_db |              | EXTERNAL    |            |
           |
 awsdatacatalog | 123_default          |              | EXTERNAL    |            |
           |
 awsdatacatalog | adhoc                |              | EXTERNAL    |            |
           |
 awsdatacatalog | all_shapes_10mb      |              | EXTERNAL    |            |
           |
 awsdatacatalog | all_shapes_1g        |              | EXTERNAL    |            |
           |
```

SHOW TABLE

显示表的定义，包括表属性、表约束、列属性和列约束。您可以使用 SHOW TABLE 语句的输出来重新创建表。

有关表创建的更多信息，请参阅CREATE TABLE。

语法

```
SHOW TABLE [schema_name.]table_name
```

参数

schema_name

（可选）相关 schema 的名称。

table_name

要显示的表的名称。

示例

以下是表 sales 的 SHOW TABLE 输出示例。

```
show table sales;
```

```
CREATE TABLE public.sales (
salesid integer NOT NULL ENCODE az64,
listid integer NOT NULL ENCODE az64 distkey,
sellerid integer NOT NULL ENCODE az64,
buyerid integer NOT NULL ENCODE az64,
eventid integer NOT NULL ENCODE az64,
dateid smallint NOT NULL,
qtysold smallint NOT NULL ENCODE az64,
pricepaid numeric(8,2) ENCODE az64,
commission numeric(8,2) ENCODE az64,
saletime timestamp without time zone ENCODE az64
)
DISTSTYLE KEY SORTKEY ( dateid );
```

以下是 schema public 中的表 category 的 SHOW TABLE 输出示例。

```
show table public.category;
```

```
CREATE TABLE public.category (
```

```
catid smallint NOT NULL distkey,
catgroup character varying(10) ENCODE lzo,
catname character varying(10) ENCODE lzo,
catdesc character varying(50) ENCODE lzo
) DISTSTYLE KEY SORTKEY ( catid );
```

以下示例将使用主键创建表 foo。

```
create table foo(a int PRIMARY KEY, b int);
```

SHOW TABLE 结果将显示 create 语句，以及 foo 表的所有属性。

```
show table foo;
```

```
CREATE TABLE public.foo ( a integer NOT NULL ENCODE az64, b integer ENCODE az64,
 PRIMARY KEY (a) ) DISTSTYLE AUTO;
```

SHOW TABLES

显示架构中表的列表以及一些表属性。

每个输出行都包含数据库名称、架构名称、表名称、表类型、表 ACL 和备注。有关这些属性的更多信息，请参阅 SVV_ALL_TABLES。

如果由 SHOW TABLES 命令生成的表超过 10000 个，则会返回错误。

语法

```
SHOW TABLES FROM SCHEMA database_name.schema_name [LIKE 'filter_pattern']
 [LIMIT row_limit ]
```

参数

database_name

包含要列出的表的数据库的名称。

要显示 AWS Glue Data Catalog 中的表，请指定（awsdatacatalog）作为数据库名称，并确保系统配置 data_catalog_auto_mount 设置为 true。有关更多信息，请参阅 ALTER SYSTEM。

schema_name

包含要列出的表的架构的名称。

要显示 AWS Glue Data Catalog 表，请提供 AWS Glue 数据库名称作为架构名称。

filter_pattern

一个有效的 UTF-8 字符表达式，具有与表名称匹配的模式。LIKE 选项执行区分大小写的匹配，支持以下模式匹配元字符：

元字符	描述
%	匹配任意序列的零个或多个字符。
_	匹配任何单个字符。

如果 filter_pattern 不包含元字符，则模式仅表示字符串本身；在此情况下，LIKE 的行为与等于运算符相同。

row_limit

要返回的最大行数。row_limit 可以是 0–10000。

示例

以下示例显示了名为 dev 的 Amazon Redshift 数据库中的表，这些表位于架构 public 中。

```
SHOW TABLES FROM SCHEMA dev.public;

 database_name | schema_name | table_name | table_type | table_acl | remarks
---------------+-------------+------------+------------+-----------+---------
 dev           | public      | tb         | TABLE      |           |
 dev           | public      | tb2        | TABLE      |           |
 dev           | public      | tb3        | TABLE      |           |
```

以下示例显示了名为 awsdatacatalog 的 AWS Glue Data Catalog 数据库中的表，这些表位于架构 batman 中。

```
SHOW TABLES FROM SCHEMA awsdatacatalog.batman;
```

```
 database_name  | schema_name |    table_name    | table_type | table_acl | remarks
----------------+-------------+------------------+------------+-----------+---------
 awsdatacatalog | batman      | nation           | EXTERNAL   |           |
 awsdatacatalog | batman      | part             | EXTERNAL   |           |
 awsdatacatalog | batman      | partsupp         | EXTERNAL   |           |
 awsdatacatalog | batman      | region           | EXTERNAL   |           |
 awsdatacatalog | batman      | supplier         | EXTERNAL   |           |
 awsdatacatalog | batman      | automount_nation | EXTERNAL   |           |
```

SHOW VIEW

显示视图的定义，包括实体化视图和后期绑定视图的定义。您可以使用 SHOW VIEW 语句的输出来重新创建视图。

语法

```
SHOW VIEW [schema_name.]view_name
```

参数

schema_name

（可选）相关 schema 的名称。

view_name

要显示的视图的名称。

示例

以下是视图 LA_Venues_v 的视图定义。

```
create view LA_Venues_v as select * from venue where venuecity='Los Angeles';
```

以下是前面定义的视图的 SHOW VIEW 命令和输出示例。

```
show view LA_Venues_v;
```

```
SELECT venue.venueid,
venue.venuename,
```

```
venue.venuecity,
venue.venuestate,
venue.venueseats
FROM venue WHERE ((venue.venuecity)::text = 'Los Angeles'::text);
```

以下是 schema public 中的视图 public.Sports_v 的视图定义。

```
create view public.Sports_v as select * from category where catgroup='Sports';
```

以下是前面定义的视图的 SHOW VIEW 命令和输出示例。

```
show view public.Sports_v;
```

```
SELECT category.catid,
category.catgroup,
category.catname,
category.catdesc
FROM category WHERE ((category.catgroup)::text = 'Sports'::text);
```

START TRANSACTION

BEGIN 函数的同义词。

请参阅 BEGIN。

TRUNCATE

删除表中的所有行，而不执行表扫描：此操作是非限定的 DELETE 操作的替代方法，其速度更快。要运行 TRUNCATE 命令，您必须拥有 TRUNCATE TABLE 权限，是表的所有者，或者是超级用户。要授予截断表的权限，请使用 GRANT 命令。

TRUNCATE 的效率要比 DELETE 高很多，不需要 VACUUM 和 ANALYZE。不过请注意，TRUNCATE 在其运行的事务中提交事务。

语法

```
TRUNCATE [ TABLE ] table_name
```

该命令也适用于实体化视图。

```
TRUNCATE materialized_view_name
```

参数

TABLE

可选关键字。

table_name

一个临时或永久表。只有表所有者或超级用户可以截断表。

您可以截断任何表，包括在外键约束中引用的表。

在截断表后，无需对表执行 vacuum 操作。

materialized_view_name

实体化视图。

您可以截断用于串流摄取的实体化视图。

使用说明

TRUNCATE 命令提交运行该命令的事务；因此，您无法回滚 TRUNCATE 操作，TRUNCATE 命令可能在提交自身时提交其他操作。

示例

使用 TRUNCATE 命令可以删除 CATEGORY 表中的所有行：

```
truncate category;
```

尝试回滚 TRUNCATE 操作：

```
begin;

truncate date;

rollback;

select count(*) from date;
```

```
count
-------
0
(1 row)
```

DATE 表在 ROLLBACK 命令后保留为空，因为已自动提交 TRUNCATE 命令。

以下示例使用 TRUNCATE 命令从实体化视图中删除所有行。

```
truncate my_materialized_view;
```

此命令会删除实体化视图中的所有记录，并保持实体化视图及其架构不变。在查询中，实体化视图名称是一个示例。

UNLOAD

使用 Amazon S3 服务器端加密 (SSE-S3) 功能将查询结果卸载到 Amazon S3 上的一个或多个文本文件、JSON 或 Apache Parquet 文件中。您还可以指定利用 AWS Key Management Service 密钥进行服务器端加密 (SSE-KMS)，或利用客户管理的密钥进行客户端加密。

预设情况下，卸载文件的格式为管道符分隔（|）的文本。

您可以通过设置 MAXFILESIZE 参数来管理 Amazon S3 上文件的大小，并可通过扩展对文件数进行管理。确保将 S3 IP 范围添加到您的允许列表中。要了解有关所需 S3 IP 范围的更多信息，请参阅网络隔离。

现在，您可以将 Amazon Redshift 查询的结果卸载到 Apache Parquet 中的 Amazon S3 数据湖，Apache Parquet 是一种用于分析的高效开放列式存储格式。与文本格式相比，Parquet 格式的卸载速度提高了 2 倍，并且在 Amazon S3 中耗用的存储量减少了 6 倍。这使您能够以开放格式将在 Amazon S3 中完成的数据转换和数据扩充保存到 Amazon S3 数据湖中。然后，您可以使用 Redshift Spectrum 和其他 AWS 服务（例如 Amazon Athena、Amazon EMR 和 Amazon SageMaker）分析您的数据。

有关使用 UNLOAD 命令的更多信息和示例场景，请参阅卸载数据。

所需的特权和权限

要成功执行 UNLOAD 命令，至少需要对数据库中的数据具有 SELECT 权限以及写入 Amazon S3 位置的权限。所需的权限与 COPY 命令类似。有关 COPY 命令权限的信息，请参阅访问其他 AWS 资源的权限。

语法

```
UNLOAD ('select-statement')
TO 's3://object-path/name-prefix'
authorization
[ option, ...]

where authorization is
IAM_ROLE { default | 'arn:aws:iam::<AWS ##-id-1>:role/<role-name>[,arn:aws:iam::<AWS #
#-id-2>:role/<role-name>][,...]' }

where option is
| [ FORMAT [ AS ] ] CSV | PARQUET | JSON
| PARTITION BY ( column_name [, ... ] ) [ INCLUDE ]
| MANIFEST [ VERBOSE ]
| HEADER
| DELIMITER [ AS ] 'delimiter-char'
| FIXEDWIDTH [ AS ] 'fixedwidth-spec'
| ENCRYPTED [ AUTO ]
| BZIP2
| GZIP
| ZSTD
| ADDQUOTES
| NULL [ AS ] 'null-string'
| ESCAPE
| ALLOWOVERWRITE
| CLEANPATH
| PARALLEL [ { ON | TRUE } | { OFF | FALSE } ]
| MAXFILESIZE [AS] max-size [ MB | GB ]
| ROWGROUPSIZE [AS] size [ MB | GB ]
| REGION [AS] 'aws-region' }
| EXTENSION 'extension-name'
```

参数

('select-statement')

SELECT 查询。卸载查询的结果。大多数情况下，通过在查询中指定 ORDER BY 子句来按排序顺序卸载数据是值得的。这种方法可以节省重新加载数据时对数据进行排序所需的时间。

必须用单引号将查询括起来，如下所示：

```
('select * from venue order by venueid')
```

> **Note**
>
> 如果查询包含引号（例如，用引号将文本值引起来），则将文本放在两组单引号之间—您还必须用单引号将查询引起来：
>
> ```
> ('select * from venue where venuestate=''NV''')
> ```

TO 's3://object-path/name-prefix'

Amazon S3 上的位置的完整路径（包括桶名称），Amazon Redshift 会将输出文件对象（如果指定 MANIFEST，则包括清单文件）写入到该位置。对象名将带有 name-prefix 前缀。如果使用 PARTITION BY，则会根据需要自动将正斜杠 (/) 添加到 name-prefix 值的末尾。为了增强安全性，UNLOAD 使用 HTTPS 连接来连接到 Amazon S3。默认情况下，UNLOAD 为每个切片写入一个或多个文件。UNLOAD 会将一个分片编号和一个分段编号附加到指定名称前缀，如下所示：

<object-path>/*<name-prefix>*<*slice-number*>_part_<*part-number*>.

如果指定了 MANIFEST，则会按以下格式写入清单文件：

<object_path>/*<name_prefix>*manifest.

如果将 PARALLEL 指定为 OFF，则按如下方式写入数据文件：

<object_path>/*<name_prefix>*<*part-number*>.

UNLOAD 会使用 Amazon S3 服务器端加密 (SSE) 自动创建加密文件，如果使用 MANIFEST，还将包括清单文件。COPY 命令在加载操作期间会自动读取服务器端加密文件。您可以使用 Amazon S3 控制台或 API 以透明方式从桶下载服务器端加密文件。有关更多信息，请参阅使用服务器端加密保护数据。

要使用 Amazon S3 客户端加密，请指定 ENCRYPTED 选项。

> **Important**
>
> 当 Amazon S3 桶与 Amazon Redshift 数据库不在同一个 AWS 区域时，需要 REGION。

授权

UNLOAD 命令需要向 Amazon S3 写入数据的授权。UNLOAD 命令的授权参数与 COPY 命令相同。有关更多信息，请参阅 COPY 命令语法参考中的 授权参数。

IAM_ROLE { default | 'arn:aws:iam::*<AWS ##-id-1>*:role/*<role-name>*'

使用默认关键字让 Amazon Redshift 使用 IAM 角色，该角色设置为默认值并在 UNLOAD 命令运行时与集群关联。

使用 IAM 角色的 Amazon 资源名称（ARN），您的集群使用该角色进行身份验证和授权。如果您指定 IAM_ROLE，则无法使用 ACCESS_KEY_ID 和 SECRET_ACCESS_KEY、SESSION_TOKEN 或 CREDENTIALS。IAM_ROLE 可以链接起来。有关更多信息，请参阅《Amazon Redshift 管理指南》中的链接 IAM 角色。

[FORMAT [AS]] CSV | PARQUET | JSON

用于指定卸载格式以覆盖默认格式的关键词。

使用 CSV 时，使用逗号 (,) 字符作为默认分隔符卸载到 CSV 格式的文本文件。如果某个字段包含分隔符、双引号、换行符或回车符，则卸载文件中的该字段会用双引号引起来。数据字段中的双引号由一个额外的双引号转义。当卸载零行时，Amazon Redshift 可能会写入空的 Amazon S3 对象。

使用 PARQUET 时，卸载到 Apache Parquet 1.0 版本格式的文件。默认情况下，每个行组都使用 SNAPPY 压缩进行压缩。有关 Apache Parquet 格式的更多信息，请参阅 Parquet。

当 JSON 卸载到 JSON 文件时，其中每行都包含一个 JSON 对象，表示查询结果中的完整记录。当查询结果包含 SUPER 列时，Amazon Redshift 支持编写嵌套 JSON。要创建有效的 JSON 对象，查询中每个列的名称必须唯一。在 JSON 文件中，布尔值将卸载为 `t` 或 `f`，并且 NULL 值卸载为 `null`。当卸载零行时，Amazon Redshift 不写入 Amazon S3 对象。

关键字 FORMAT 和 AS 是可选的。不能将 CSV 与 FIXEDWIDTH 或 ADDQUOTES 一起使用。您不能将 PARQUET 与 DELIMITER、FIXEDWIDTH、ADDQUOTES、ESCAPE、NULL AS、HEADER、GZIP、BZIP2 或 ZSTD 一起使用。只有使用 AWS Key Management Service 密钥的服务器端加密 (SSE-KMS) 支持带有 ENCRYPTED 的 PARQUET。您不能将 JSON 与 DELIMITER、HEADER、FIXEDWIDTH、ADDQUOTES、ESCAPE 或 NULL AS 一起使用。

PARTITION BY (column_name [, ...]) [INCLUDE]

指定卸载操作的分区键。UNLOAD 按照 Apache Hive 约定，根据分区键值自动将输出文件分区到分区文件夹中。例如，属于 2019 分区年和 9 月份的 Parquet 文件具有以下前缀：`s3://my_bucket_name/my_prefix/year=2019/month=September/000.parquet`。

column_name 的值必须是正在卸载的查询结果中的列。

如果指定 PARTITION BY 与 INCLUDE 选项，则不会从卸载的文件中删除分区列。

Amazon Redshift 不支持 PARTITION BY 子句中的字符串文本。

MANIFEST [VERBOSE]

创建一个清单文件，其中明确列出由 UNLOAD 过程创建的数据文件的详细信息。清单是一个 JSON 格式的文本文件，其中列出写入到 Amazon S3 的每个文件的 URL。

如果使用 VERBOSE 选项指定 MANIFEST，则清单包含以下详细信息：

- 列名和数据类型，对于 CHAR、VARCHAR 或 NUMERIC 数据类型，还包含每列的维度。对于 CHAR 和 VARCHAR 数据类型，维度是长度。对于 DECIMAL 或 NUMERIC 数据类型，维度是精度和小数位数。
- 已卸载到每个文件的行计数。如果指定了 HEADER 选项，则行计数包括标题行。
- 卸载的所有文件的总文件大小以及卸载到所有文件的总行数。如果指定了 HEADER 选项，则行计数包括标题行。
- 作者。作者始终是“Amazon Redshift”。

您只能在 MANIFEST 之后指定 VERBOSE。

清单文件将采用与卸载文件相同的 Amazon S3 路径前缀，按照 `<object_path_prefix>manifest` 的格式写入。例如，如果 UNLOAD 指定 Amazon S3 路径前缀“`s3://mybucket/venue_`”，则清单文件位置为“`s3://mybucket/venue_manifest`”。

HEADER

在每个输出文件的顶部添加包含列名称的标题行。文本转换选项（如 CSV、DELIMITER、ADDQUOTES 和 ESCAPE）也适用于标题行。您不能将 HEADER 与 FIXEDWIDTH 一起使用。

DELIMITER AS 'delimiter_character'

指定用于在输出文件中分隔字段的单个 ASCII 字符，如管道字符 (|)、逗号 (,) 或制表符 (\t)。文本文件的默认分隔符是竖线字符。CSV 文件的默认分隔符是逗号字符。AS 关键字是可选的。您不能将 DELIMITER 与 FIXEDWIDTH 一起使用。如果数据包含分隔符，您需要指定 ESCAPE 选项来转义分隔符，或者使用 ADDQUOTES 用双引号将数据括起来。或者，指定一个数据中不包含的分隔符。

FIXEDWIDTH 'fixedwidth_spec'

将数据卸载到一个文件，其中每个列的宽度均为固定长度，而不是由分隔符隔开。fixedwidth_spec 是一个字符串，用于指定列数和列宽。AS 关键字是可选的。由于 FIXEDWIDTH 不会截断数据，因此 UNLOAD 语句中每个列的规格必须至少为该列最长条目的长度。fixedwidth_spec 的格式如下：

```
'colID1:colWidth1,colID2:colWidth2, ...'
```

您不能将 FIXEDWIDTH 与 DELIMITER 或 HEADER 一起使用。

ENCRYPTED [AUTO] (加密 [自动])

指定 Amazon S3 上的输出文件的加密方法是 Amazon S3 服务器端加密还是客户端加密。如果指定了 MANIFEST，也会加密清单文件。有关更多信息，请参阅 卸载加密的数据文件。如果不指定 ENCRYPTED 参数，UNLOAD 会使用 AWS 托管的加密密钥进行 Amazon S3 服务器端加密 (SSE-S3)，自动创建加密文件。

对于 ENCRYPTED，您可能希望使用 AWS KMS 密钥进行服务器端加密 (SSE-KMS) 卸载到 Amazon S3。如果是这样，请使用 KMS_KEY_ID 参数提供密钥 ID。CREDENTIALS 参数不能与 KMS_KEY_ID 参数配合使用。如果使用 KMS_KEY_ID 对数据运行 UNLOAD 命令，则可以对同一数据执行 COPY 操作，而无需指定密钥。

要利用客户提供的对称密钥进行客户端加密来卸载到 Amazon S3，请通过以下两种方式之一提供密钥。要提供密钥，请使用 MASTER_SYMMETRIC_KEY 参数或 `master_symmetric_key` 凭证字符串的 CREDENTIALS 部分。如果使用根对称密钥卸载数据，请确保在对加密数据执行 COPY 操作时提供相同的密钥。

UNLOAD 不支持使用客户提供的密钥（SSE-C）的 Amazon S3 服务器端加密。

如果使用 ENCRYPTED AUTO，则 UNLOAD 命令将提取目标 Amazon S3 桶属性上的默认 AWS KMS 加密密钥，并使用 AWS KMS 密钥加密写入 Amazon S3 的文件。如果桶没有默认的 AWS KMS 加密密钥，UNLOAD 会使用 AWS 托管的加密密钥进行 Amazon Redshift 服务器端加密 (SSE-S3) 以自动创建加密文件。您不能将此选项与 KMS_KEY_ID、MASTER_SYMMETRIC_KEY 或包含 master_symmetric_key 的 CREDENTIALS 一起使用。

KMS_KEY_ID 'key-id'

指定用于在 Amazon S3 上加密数据文件的 AWS Key Management Service（AWS KMS）密钥的密钥 ID。有关更多信息，请参阅什么是 AWS Key Management Service？ 如果指定 KMS_KEY_ID，您还必须指定 ENCRYPTED 参数。如果指定 KMS_KEY_ID，则不能

使用 CREDENTIALS 参数进行身份验证，而应使用 IAM_ROLE 或 ACCESS_KEY_ID and SECRET_ACCESS_KEY。

MASTER_SYMMETRIC_KEY 'root_key'

指定用于在 Amazon S3 上加密数据文件的根对称密钥。如果指定 MASTER_SYMMETRIC_KEY，您还必须指定 ENCRYPTED 参数。您不能将 MASTER_SYMMETRIC_KEY 与 CREDENTIALS 参数结合使用。有关更多信息，请参阅 从 Amazon S3 中加载加密的数据文件。

BZIP2

对于每个切片，将数据卸载到一个或多个 bzip2 压缩的文件。将为每个生成的文件附加 `.bz2` 扩展名。

GZIP

对于每个切片，将数据卸载到一个或多个 gzip 压缩的文件。将为每个生成的文件附加 `.gz` 扩展名。

ZSTD

对于每个切片，将数据卸载到一个或多个 Zstandard 压缩的文件。将为每个生成的文件附加 `.zst` 扩展名。

ADDQUOTES

使用引号将每个卸载的数据字段括起来，以便 Amazon Redshift 能够自行卸载包含分隔符自身的数据值。例如，如果分隔符为逗号，则可以成功卸载并重新加载以下数据：

```
"1","Hello, World"
```

如果不添加引号，则将字符串 `Hello, World` 解析为两个单独的字段。

某些输出格式不支持 ADDQUOTES。

在使用 ADDQUOTES 的情况下，如果重新加载数据，则必须在 COPY 中指定 REMOVEQUOTES。

NULL AS 'null-string'

指定表示卸载文件中的 null 值的字符串。如果使用此选项，所有输出文件将包含指定字符串来替代在选定数据中找到的所有 null 值。如果未指定此选项，则 null 值将卸载为：

- 零长度字符串（对于分隔的输出）
- 空格字符串（对于固定宽度输出）

如果为固定宽度的卸载指定 null 字符串，并且输出列的宽度小于 null 字符串的宽度，则将发生以下行为：

- 使用空字段作为非字符列的输出
- 针对字符列报告错误

与用户定义的字符串表示 null 值的其他数据类型不同，Amazon Redshift 使用 JSON 格式导出 SUPER 数据列，并根据 JSON 格式将其表示为 null。因此，SUPER 数据列会忽略 UNLOAD 命令中使用的 NULL [AS] 选项。

ESCAPE

对于分隔的卸载文件中的 CHAR 和 VARCHAR 列，将在每次出现的以下字符之前放置一个转义字符 (\)：

- 换行：\n
- 回车：\r
- 为卸载的数据指定的分隔符。
- 转义字符：\
- 引号字符：" 或 '（如果在 UNLOAD 命令中同时指定 ESCAPE 和 ADDQUOTES）。

> **Important**
>
> 如果已将 COPY 与 ESCAPE 选项结合使用来加载数据，则还必须在 UNLOAD 命令中指定 ESCAPE 选项以生成反向输出文件。同样，如果您使用 ESCAPE 选项执行 UNLOAD 命令，则在您对相同数据执行 COPY 操作时将需要使用 ESCAPE 选项。

ALLOWOVERWRITE

默认情况下，如果 UNLOAD 找到可能会覆盖的文件，则该命令将失败。如果指定 ALLOWOVERWRITE，则 UNLOAD 将覆盖现有文件（包括清单文件）。

CLEANPATH

CLEANPATH 选项会删除位于 TO 子句中指定的 Amazon S3 路径中的现有文件，然后再将文件卸载到指定位置。

如果包含 PARTITION BY 子句，则只会从分区文件夹中删除现有文件，以接收 UNLOAD 操作生成的新文件。

您必须拥有 Amazon S3 桶的 `s3:DeleteObject` 权限。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的 Amazon S3 中的策略和权限。您使用 CLEANPATH 选项删除的文件将永久删除并且无法恢复。

如果您指定了 ALLOWOVERWRITE 选项，则无法指定 CLEANPATH 选项。

PARALLEL

默认情况下，UNLOAD 根据集群中切片的数量将数据并行写入到多个文件。默认选项为 ON 或 TRUE。如果 PARALLEL 为 OFF 或 FALSE，UNLOAD 将按顺序写入到一个或多个数据文件，并完全根据 ORDER BY 子句（如果已使用）进行排序。数据文件的最大大小为 6.2 GB。例如，如果您卸载 13.4GB 数据，则 UNLOAD 将创建以下三个文件。

```
s3://mybucket/key000    6.2 GB
s3://mybucket/key001    6.2 GB
s3://mybucket/key002    1.0 GB
```

> **Note**
>
> UNLOAD 命令旨在使用并行处理。对于大多数情况，特别是文件将用于通过 COPY 命令加载表时，我们建议保留 PARALLEL 为启用状态。

MAXFILESIZE [AS] 最大大小 [MB | GB]

指定 UNLOAD 在 Amazon S3 中创建的文件的最大大小。可以指定 5 MB 到 6.2 GB 之间的十进制值。AS 关键字是可选的。默认单位为 MB。如果未指定 MAXFILESIZE，则默认最大文件大小为 6.2 GB。清单文件（如果使用）的大小不受 MAXFILESIZE 的影响。

ROWGROUPSIZE [AS] size [MB | GB]

指定行组的大小。选择更大的容量可以减少行组的数量，从而减少网络通信量。指定介于 32MB 到 128MB 之间的整数值。AS 关键字是可选的。默认单位为 MB。

如果未指定 ROWGROUPSIZE，则默认大小为 32MB。要使用此参数，存储格式必须是 Parquet，节点类型必须是 ra3.4xlarge、ra3.16xlarge 或 dc2.8xlarge。

REGION [AS] 'aws-region'

指定目标 Amazon S3 桶所在的 AWS 区域。当 UNLOAD 的目标 Amazon S3 桶与 Amazon Redshift 数据库不在同一个 AWS 区域时，需要 REGION。

aws_region 的值必须与《AWS 一般参考》的 Amazon Redshift 区域和端点中列出的 AWS 区域匹配。

默认情况下，UNLOAD 假定目标 Amazon S3 桶位于 Amazon Redshift 数据库所在的 AWS 区域。

EXTENSION ‘extension-name'

指定要附加到卸载文件的名称后的文件扩展名。Amazon Redshift 不运行任何验证，因此您必须验证指定的文件扩展名是否正确。如果您使用的是 GZIP 等压缩方法，则仍然需要在扩展参数中指定 `.gz`。如果您不提供任何扩展名，则 Amazon Redshift 不会向文件名添加任何内容。如果您指定压缩方法而不提供扩展名，则 Amazon Redshift 只会将压缩方法的扩展名添加到文件名。

使用说明

将 ESCAPE 用于所有分隔的文本 UNLOAD 操作

在使用分隔符执行 UNLOAD 时，您的数据可能包含该分隔符或 ESCAPE 选项描述中列出的任意字符。在这种情况下，必须将 ESCAPE 选项与 UNLOAD 语句一起使用。如果不将 ESCAPE 选项与 UNLOAD 结合使用，则使用卸载的数据执行的后续 COPY 操作可能会失败。

> **Important**
>
> 我们强烈建议您始终将 ESCAPE 与 UNLOAD 和 COPY 语句一起使用。例外情况是您确定您的数据不包含任何分隔符或可能需要转义的其他字符。

丢失浮点精度

您可能遇到连续卸载并重新加载的浮点数据的精度丢失的情况。

Limit 子句

SELECT 查询无法在外部 SELECT 中使用 LIMIT 子句。例如，以下 UNLOAD 语句将失败。

```
unload ('select * from venue limit 10')
to 's3://mybucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole';
```

应改用嵌套的 LIMIT 子句，如下例所示。

```
unload ('select * from venue where venueid in
(select venueid from venue order by venueid desc limit 10)')
to 's3://mybucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole';
```

您也可以使用 LIMIT 子句通过 SELECT…INTO 或 CREATE TABLE AS 来填充表，然后从该表卸载。

卸载 GEOMETRY 数据类型的列

您只能将 GEOMETRY 列卸载为文本或 CSV 格式。无法使用 FIXEDWIDTH 选项卸载 GEOMETRY 数据。以扩展的已知二进制 (EWKB) 格式的十六进制形式卸载数据。如果 EWKB 数据的大小大于 4 MB，则会出现警告，因为以后无法将数据加载到表中。

卸载 HLLSKETCH 数据类型

您只能将 HLLSKETCH 列卸载为文本或 CSV 格式。无法使用 FIXEDWIDTH 选项卸载 HLLSKETCH 数据。对于密集的 HyperLogLog 草图，数据将以 Base64 格式卸载，对于稀疏 HyperLogLog 草图，则以 JSON 格式卸载数据。有关更多信息，请参阅 HyperLogLog 函数。

以下示例将包含 HLLSKETCH 列的表导出到文件中。

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2),
 (5,2), (6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

UNLOAD ('select * from hll_table') TO 's3://mybucket/unload/'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole' NULL AS 'null' ALLOWOVERWRITE
 CSV;
```

卸载 VARBYTE 数据类型的列

您只能将 VARBYTE 列卸载为文本或 CSV 格式。以十六进制形式卸载数据。无法使用 FIXEDWIDTH 选项卸载 VARBYTE 数据。不支持 ADDQUOTES 选项“卸载到 CSV”。VARBYTE 列不得为 PARTITIONED BY 列。

FORMAT AS PARQUET 子句

使用 FORMAT AS PARQUET 时，请注意以下注意事项：

- 卸载到 Parquet 不使用文件级压缩。每个行组都使用 SNAPPY 进行压缩。
- 如果未指定 MAXFILESIZE，则默认最大文件大小为 6.2 GB。您可以使用 MAXFILESIZE 指定 5 MB–6.2 GB 的文件大小。写入文件时，实际文件大小是近似值，因此它可能不完全等于您指定的数字。

  要最大限度地提高扫描性能，Amazon Redshift 尝试创建包含相同大小的 32 MB 行组的 Parquet 文件。您指定的 MAXFILESIZE 值会自动向下舍入为 32 MB 的最接近倍数。例如，如果您指定 MAXFILESIZE 200 MB，则卸载的每个 Parquet 文件大约为 192 MB（32 MB 行组 x 6 = 192 MB）。
- 如果列使用 TIMESTAMPTZ 数据格式，则只卸载时间戳值。未卸载时区信息。
- 不要指定以下划线 (_) 或句点 (.) 字符开头的文件名前缀。Redshift Spectrum 将以这些字符开头的文件视为隐藏文件并忽略它们。

PARTITION BY 子句

使用 PARTITION BY 时，请注意以下注意事项：

- 分区列不包含在输出文件中。
- 请确保在 UNLOAD 语句中使用的 SELECT 查询中包含分区列。您可以在 UNLOAD 命令中指定任意数量的分区列。但有一个限制，即至少应有一个非分区列作为文件的一部分。
- 如果分区键值为 null，Amazon Redshift 会自动将该数据卸载到名为 `partition_column=__HIVE_DEFAULT_PARTITION__` 的默认分区中。
- UNLOAD 命令不会对外部目录进行任何调用。要将新分区注册为现有外部表的一部分，请使用单独的 ALTER TABLE ... ADD PARTITION ... 命令。或者，您可以运行 CREATE EXTERNAL TABLE 命令将卸载的数据注册为新的外部表。也可以使用 AWS Glue 爬网程序填充您的 Data Catalog。有关更多信息，请参阅《AWS Glue 开发人员指南》中的定义爬网程序。
- 如果使用 MANIFEST 选项，则 Amazon Redshift 在根 Amazon S3 文件夹中仅生成一个清单文件。
- 可以用作分区键的列数据类型为：SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、BOOLEAN、CHAR、VARCHAR、DATE 和 TIMESTAMP。

使用 ASSUMEROLE 权限授予对 UNLOAD 操作的 IAM 角色的访问权限

要为特定用户和组提供对 UNLOAD 操作的 IAM 角色的访问权限，超级用户可以向用户和组授予 IAM 角色的 ASSUMEROLE 权限。有关信息，请参阅 GRANT。

UNLOAD 不支持 Amazon S3 接入点别名

不得将 Amazon S3 接入点别名与 UNLOAD 命令一起使用。

示例

有关如何使用 UNLOAD 命令的示例，请参阅 UNLOAD 示例。

UNLOAD 示例

这些示例演示了 UNLOAD 命令的各种参数。许多示例使用 TICKIT 示例数据集。有关更多信息，请参阅 示例数据库。

> **Note**
>
> 为便于阅读，这些示例包含换行符。请不要在您的 credentials-args 字符串中包含换行符或空格。

将 VENUE 卸载到竖线分隔的文件（默认分隔符）

以下示例卸载 VENUE 表并将数据写入到 s3://mybucket/unload/：

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

默认情况下，UNLOAD 为每个切片写入一个或多个文件。假定一个双节点集群中的每个节点有两个切片，上一个示例在 mybucket 中会创建这些文件：

```
unload/0000_part_00
unload/0001_part_00
unload/0002_part_00
unload/0003_part_00
```

为了更好地区分输出文件，可在位置中包含前缀。以下示例卸载 VENUE 表并将数据写入到 s3://mybucket/unload/venue_pipe_：

```
unload ('select * from venue')
to 's3://mybucket/unload/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

最后在 unload 文件夹中生成四个文件，再次假定有四个切片。

```
venue_pipe_0000_part_00
venue_pipe_0001_part_00
venue_pipe_0002_part_00
venue_pipe_0003_part_00
```

将 LINEITEM 表卸载到分区的 Parquet 文件

以下示例以 Parquet 格式卸载 LINEITEM 表，按 l_shipdate 列进行分区。

```
unload ('select * from lineitem')
to 's3://mybucket/lineitem/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
PARQUET
PARTITION BY (l_shipdate);
```

假设有四个切片，生成的 Parquet 文件会动态分区到不同文件夹中。

```
s3://mybucket/lineitem/l_shipdate=1992-01-02/0000_part_00.parquet
                                             0001_part_00.parquet
                                             0002_part_00.parquet
                                             0003_part_00.parquet
s3://mybucket/lineitem/l_shipdate=1992-01-03/0000_part_00.parquet
                                             0001_part_00.parquet
                                             0002_part_00.parquet
                                             0003_part_00.parquet
s3://mybucket/lineitem/l_shipdate=1992-01-04/0000_part_00.parquet
                                             0001_part_00.parquet
                                             0002_part_00.parquet
                                             0003_part_00.parquet
...
```

> **Note**
>
> 在某些情况下，UNLOAD 命令使用的是 INCLUDE 选项，如以下 SQL 语句所示。
>
> ```
> unload ('select * from lineitem')
> to 's3://mybucket/lineitem/'
> iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
> PARQUET
> ```

```
PARTITION BY (l_shipdate) INCLUDE;
```

在这些情况下，l_shipdate 列也在 Parquet 文件中的数据中。否则，l_shipdate 列数据不在 Parquet 文件中。

将 VENUE 表卸载到 JSON 文件

以下示例卸载了 VENUE 表并将 JSON 格式的数据写入到 s3://mybucket/unload/。

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
JSON;
```

以下是 VENUE 表中的示例行。

```
venueid | venuename                  | venuecity       | venuestate | venueseats
--------+----------------------------+-----------------+------------+-----------
      1 | Pinewood Racetrack         | Akron           | OH         | 0
      2 | Columbus "Crew" Stadium    | Columbus        | OH         | 0
      4 | Community, Ballpark, Arena | Kansas City     | KS         | 0
```

卸载到 JSON 后，文件的格式类似于以下形式。

```
{"venueid":1,"venuename":"Pinewood
 Racetrack","venuecity":"Akron","venuestate":"OH","venueseats":0}
{"venueid":2,"venuename":"Columbus \"Crew\" Stadium
 ","venuecity":"Columbus","venuestate":"OH","venueseats":0}
{"venueid":4,"venuename":"Community, Ballpark, Arena","venuecity":"Kansas
 City","venuestate":"KS","venueseats":0}
```

将 VENUE 上传到 CSV 文件

以下示例卸载 VENUE 表并将 CSV 格式的数据写入到 s3://mybucket/unload/。

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
CSV;
```

假设 VENUE 表包含以下行。

```
venueid | venuename                  | venuecity       | venuestate | venueseats
--------+----------------------------+-----------------+------------+-----------
      1 | Pinewood Racetrack         | Akron           | OH         | 0
      2 | Columbus "Crew" Stadium    | Columbus        | OH         | 0
      4 | Community, Ballpark, Arena | Kansas City     | KS         | 0
```

卸载文件类似于以下内容。

```
1,Pinewood Racetrack,Akron,OH,0
2,"Columbus ""Crew"" Stadium",Columbus,OH,0
4,"Community, Ballpark, Arena",Kansas City,KS,0
```

将 VENUE 卸载到使用分隔符的 CSV 文件

以下示例卸载 VENUE 表并使用竖线字符 (|) 作为分隔符以 CSV 格式写入数据。卸载的文件将写入到 s3://mybucket/unload/。此示例中的 VENUE 表在第一行 (Pinewood Race|track) 的值中包含竖线字符。这样做是为了表明结果中的值已用双引号引起来。双引号由双引号进行转义，整个字段用双引号引起来。

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
CSV DELIMITER AS '|';
```

假设 VENUE 表包含以下行。

```
venueid | venuename                  | venuecity       | venuestate | venueseats
--------+----------------------------+-----------------+------------+-------------
      1 | Pinewood Race|track        | Akron           | OH         | 0
      2 | Columbus "Crew" Stadium    | Columbus        | OH         | 0
      4 | Community, Ballpark, Arena | Kansas City     | KS         | 0
```

卸载文件类似于以下内容。

```
1|"Pinewood Race|track"|Akron|OH|0
2|"Columbus ""Crew"" Stadium"|Columbus|OH|0
```

```
4|Community, Ballpark, Arena|Kansas City|KS|0
```

使用清单文件卸载 VENUE

要创建清单文件，请包含 MANIFEST 选项。以下示例卸载 VENUE 表，并将清单文件与数据文件一起写入到 s3://mybucket/venue_pipe_：

> **Important**
>
> 如果使用 MANIFEST 选项卸载文件，则应在加载文件时将 MANIFEST 选项与 COPY 命令结合使用。如果您使用相同的前缀来加载文件且不指定 MANIFEST 选项，则 COPY 将失败，因为它假定清单文件是数据文件。

```
unload ('select * from venue')
to 's3://mybucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole'
manifest;
```

最后生成下面的 5 个文件：

```
s3://mybucket/venue_pipe_0000_part_00
s3://mybucket/venue_pipe_0001_part_00
s3://mybucket/venue_pipe_0002_part_00
s3://mybucket/venue_pipe_0003_part_00
s3://mybucket/venue_pipe_manifest
```

下面显示了清单文件的内容。

```
{
  "entries": [
    {"url":"s3://mybucket/tickit/venue_0000_part_00"},
    {"url":"s3://mybucket/tickit/venue_0001_part_00"},
    {"url":"s3://mybucket/tickit/venue_0002_part_00"},
    {"url":"s3://mybucket/tickit/venue_0003_part_00"}
  ]
}
```

使用 MANIFEST VERBOSE 卸载 VENUE

指定 MANIFEST VERBOSE 选项时，清单文件包含以下部分：

- entries 部分列出每个文件的 Amazon S3 路径、文件大小和行数。
- schema 部分列出每列的列名、数据类型和维度。
- meta 部分显示所有文件的总文件大小和行数。

以下示例使用 MANIFEST VERBOSE 选项卸载 VENUE 表。

```
unload ('select * from venue')
to 's3://mybucket/unload_venue_folder/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest verbose;
```

下面显示了清单文件的内容。

```
{
  "entries": [
    {"url":"s3://mybucket/venue_pipe_0000_part_00", "meta": { "content_length": 32295,
 "record_count": 10 }},
    {"url":"s3://mybucket/venue_pipe_0001_part_00", "meta": { "content_length": 32771,
 "record_count": 20 }},
    {"url":"s3://mybucket/venue_pipe_0002_part_00", "meta": { "content_length": 32302,
 "record_count": 10 }},
    {"url":"s3://mybucket/venue_pipe_0003_part_00", "meta": { "content_length": 31810,
 "record_count": 15 }}
  ],
  "schema": {
    "elements": [
      {"name": "venueid", "type": { "base": "integer" }},
      {"name": "venuename", "type": { "base": "character varying", 25 }},
      {"name": "venuecity", "type": { "base": "character varying", 25 }},
      {"name": "venuestate", "type": { "base": "character varying", 25 }},
      {"name": "venueseats", "type": { "base": "character varying", 25 }}
    ]
  },
  "meta": {
    "content_length": 129178,
    "record_count": 55
  },
  "author": {
    "name": "Amazon Redshift",
    "version": "1.0.0"
  }
```

```
}
```

使用标题卸载 VENUE

以下示例使用标题行卸载 VENUE。

```
unload ('select * from venue where venueseats > 75000')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
header
parallel off;
```

下面显示了带标题行的输出文件的内容。

```
venueid|venuename|venuecity|venuestate|venueseats
6|New York Giants Stadium|East Rutherford|NJ|80242
78|INVESCO Field|Denver|CO|76125
83|FedExField|Landover|MD|91704
79|Arrowhead Stadium|Kansas City|MO|79451
```

将 VENUE 卸载到较小的文件

默认情况下，文件的最大大小为 6.2 GB。如果卸载数据大于 6.2GB，UNLOAD 将为每个 6.2GB 数据段创建一个新文件。要创建较小的文件，请包括 MAXFILESIZE 参数。假设上一个示例中的数据大小为 20 GB，则下面的 UNLOAD 命令将创建 20 个文件，每个文件的大小为 1 GB。

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
maxfilesize 1 gb;
```

连续卸载 VENUE

要连续卸载，请指定 PARALLEL OFF。然后，UNLOAD 将一次写入一个文件，每个文件的大小最多为 6.2 GB。

以下示例连续卸载 VENUE 表并将数据写入到 s3://mybucket/unload/。

```
unload ('select * from venue')
to 's3://mybucket/unload/venue_serial_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
parallel off;
```

最后生成一个名为 venue_serial_000 的文件。

如果卸载数据大于 6.2GB，UNLOAD 将为每个 6.2GB 数据段创建一个新文件。以下示例连续卸载 LINEORDER 表并将数据写入到 s3://mybucket/unload/。

```
unload ('select * from lineorder')
to 's3://mybucket/unload/lineorder_serial_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off gzip;
```

最后生成下面的一系列文件。

```
lineorder_serial_0000.gz
lineorder_serial_0001.gz
lineorder_serial_0002.gz
lineorder_serial_0003.gz
```

为了更好地区分输出文件，可在位置中包含前缀。以下示例卸载 VENUE 表并将数据写入到 s3://mybucket/venue_pipe_：

```
unload ('select * from venue')
to 's3://mybucket/unload/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

最后在 unload 文件夹中生成四个文件，再次假定有四个切片。

```
venue_pipe_0000_part_00
venue_pipe_0001_part_00
venue_pipe_0002_part_00
venue_pipe_0003_part_00
```

从卸载文件中加载 VENUE

要从一组卸载文件加载表，只需使用 COPY 命令反向执行该过程即可。以下示例创建一个名为 LOADVENUE 的新表，并从上一个示例中创建的数据文件加载该表。

```
create table loadvenue (like venue);
```

```
copy loadvenue from 's3://mybucket/venue_pipe_' iam_role
 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

如果您使用 MANIFEST 选项通过卸载文件创建清单文件，则可使用相同的清单文件加载数据。您可以使用带有 MANIFEST 选项的 COPY 命令执行此操作。以下示例使用清单文件加载数据。

```
copy loadvenue
from 's3://mybucket/venue_pipe_manifest' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole'
manifest;
```

将 VENUE 卸载到加密文件

以下示例使用 AWS KMS 密钥将 VENUE 表卸载到一组加密文件。如果您使用 ENCRYPTED 选项来指定清单文件，清单文件也将加密。有关更多信息，请参阅 卸载加密的数据文件。

```
unload ('select * from venue')
to 's3://mybucket/venue_encrypt_kms'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
kms_key_id '1234abcd-12ab-34cd-56ef-1234567890ab'
manifest
encrypted;
```

以下示例使用根对称密钥将 VENUE 表卸载到一组加密文件。

```
unload ('select * from venue')
to 's3://mybucket/venue_encrypt_cmk'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key 'EXAMPLEMASTERKEYtkbjk/OpCwtYSx/M4/t7DMCDIK722'
encrypted;
```

从加密文件中加载 VENUE

要从使用带有 ENCRYPT 选项的 UNLOAD 创建的一组文件中加载表，请使用 COPY 命令逆向执行该过程。对于该命令，应使用 ENCRYPTED 选项并指定用于 UNLOAD 命令的相同根对称密钥。以下示例从上一个示例创建的加密数据文件中加载 LOADVENUE 表。

```
create table loadvenue (like venue);

copy loadvenue
```

```
from 's3://mybucket/venue_encrypt_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key 'EXAMPLEMASTERKEYtkbjk/OpCwtYSx/M4/t7DMCDIK722'
manifest
encrypted;
```

将 VENUE 数据卸载到制表符分隔的文件

```
unload ('select venueid, venuename, venueseats from venue')
to 's3://mybucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t';
```

输出数据文件如下所示：

```
1 Toyota Park Bridgeview IL 0
2 Columbus Crew Stadium Columbus OH 0
3 RFK Stadium Washington DC 0
4 CommunityAmerica Ballpark Kansas City KS 0
5 Gillette Stadium Foxborough MA 68756
...
```

将 VENUE 卸载到固定宽度的数据文件

```
unload ('select * from venue')
to 's3://mybucket/venue_fw_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth as 'venueid:3,venuename:39,venuecity:16,venuestate:2,venueseats:6';
```

输出数据文件类似于以下内容。

```
1  Toyota Park              Bridgeview  IL0
2  Columbus Crew Stadium    Columbus    OH0
3  RFK Stadium              Washington  DC0
4  CommunityAmerica BallparkKansas City KS0
5  Gillette Stadium         Foxborough  MA68756
...
```

将 VENUE 卸载到一组制表符分隔的 GZIP 压缩文件

```
unload ('select * from venue')
```

```
to 's3://mybucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t'
gzip;
```

将 VENUE 卸载到 GZIP 压缩文本文件

```
unload ('select * from venue')
to 's3://mybucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
extension 'txt.gz'
gzip;
```

卸载包含分隔符的数据

此示例使用 ADDQUOTES 选项卸载逗号分隔的数据，其中一些实际数据字段中包含逗号。

首先，创建一个包含引号的表。

```
create table location (id int, location char(64));

insert into location values (1,'Phoenix, AZ'),(2,'San Diego, CA'),(3,'Chicago, IL');
```

然后，使用 ADDQUOTES 选项卸载数据。

```
unload ('select id, location from location')
to 's3://mybucket/location_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter ',' addquotes;
```

卸载的数据文件如下所示：

```
1,"Phoenix, AZ"
2,"San Diego, CA"
3,"Chicago, IL"
...
```

卸载联接查询的结果

以下示例卸载包含窗口函数的联接查询的结果。

```
unload ('select venuecity, venuestate, caldate, pricepaid,
sum(pricepaid) over(partition by venuecity, venuestate
order by caldate rows between 3 preceding and 3 following) as winsum
from sales join date on sales.dateid=date.dateid
join event on event.eventid=sales.eventid
join venue on event.venueid=venue.venueid
order by 1,2')
to 's3://mybucket/tickit/winsum'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

输出文件如下所示：

```
Atlanta|GA|2008-01-04|363.00|1362.00
Atlanta|GA|2008-01-05|233.00|2030.00
Atlanta|GA|2008-01-06|310.00|3135.00
Atlanta|GA|2008-01-08|166.00|8338.00
Atlanta|GA|2008-01-11|268.00|7630.00
...
```

使用 NULL AS 进行卸载

默认情况下，UNLOAD 将 null 值作为空字符串输出。以下示例说明如何使用 NULL AS 来将文本字符串替换为 null 值。

对于这些示例，我们将向 VENUE 表添加几个 null 值。

```
update venue set venuestate = NULL
where venuecity = 'Cleveland';
```

从 VENUESTATE 为 null 的 VENUE 中选择，以验证列是否包含 NULL。

```
select * from venue where venuestate is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
---------+--------------------------+-----------+------------+------------
      22 | Quicken Loans Arena      | Cleveland |            |          0
     101 | Progressive Field        | Cleveland |            |      43345
      72 | Cleveland Browns Stadium | Cleveland |            |      73200
```

现在，使用 NULL AS 选项对 VENUE 表执行 UNLOAD 以便将 null 值替换为字符串“fred”。

```
unload ('select * from venue')
to 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
null as 'fred';
```

以下来自卸载文件的示例说明已将 null 值替换为 fred。这证明 VENUESEATS 的一些值也为 null，并且已替换为 fred。即使 VENUESEATS 的数据类型为整数，UNLOAD 也会将值转换为卸载文件中的文本，然后 COPY 会将其转换回整数。如果卸载到固定宽度的文件，则 NULL AS 字符串不得大于字段宽度。

```
248|Charles Playhouse|Boston|MA|0
251|Paris Hotel|Las Vegas|NV|fred
258|Tropicana Hotel|Las Vegas|NV|fred
300|Kennedy Center Opera House|Washington|DC|0
306|Lyric Opera House|Baltimore|MD|0
308|Metropolitan Opera|New York City|NY|0
  5|Gillette Stadium|Foxborough|MA|5
 22|Quicken Loans Arena|Cleveland|fred|0
101|Progressive Field|Cleveland|fred|43345
...
```

要从卸载文件加载表，请使用带相同 NULL AS 选项的 COPY 命令。

> **Note**
>
> 如果您尝试将 null 加载到定义为 NOT NULL 的列中，则 COPY 命令将失败。

```
create table loadvenuenulls (like venue);

copy loadvenuenulls from 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
null as 'fred';
```

要确认列包含 null 而不仅仅是包含空字符串，请从 LOADVENUENULLS 中选择并针对 null 进行筛选。

```
select * from loadvenuenulls where venuestate is null or venueseats is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
```

```
---------+--------------------------+-----------+------------+------------
      72 | Cleveland Browns Stadium | Cleveland |            |      73200
     253 | Mirage Hotel             | Las Vegas | NV         |
     255 | Venetian Hotel           | Las Vegas | NV         |
      22 | Quicken Loans Arena      | Cleveland |            |          0
     101 | Progressive Field        | Cleveland |            |      43345
     251 | Paris Hotel              | Las Vegas | NV         |

...
```

您可以使用默认 NULL AS 行为对包含 null 的表执行 UNLOAD 操作，然后使用默认 NULL AS 行为对数据执行 COPY 以复制回表中；不过，目标表中的任何非数字字段将包含空字符串而不是 null。默认情况下，UNLOAD 将 null 转换为空字符串（空格或零长度）。对于数字列，COPY 会将空字符串转换为 NULL，但会将空字符串插入非数字列中。以下示例说明如何在执行 COPY 后使用默认 NULL AS 行为执行 UNLOAD。

```
unload ('select * from venue')
to 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' allowoverwrite;

truncate loadvenuenulls;
copy loadvenuenulls from 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

在本示例中，当您针对 null 进行筛选时，仅显示那些 VENUESEATS 包含 null 的行。其中，VENUESTATE 在表 (VENUE) 中包含 null，目标表 (LOADVENUENULLS) 中的 VENUESTATE 包含空字符串。

```
select * from loadvenuenulls where venuestate is null or venueseats is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
---------+--------------------------+-----------+------------+------------
     253 | Mirage Hotel             | Las Vegas | NV         |
     255 | Venetian Hotel           | Las Vegas | NV         |
     251 | Paris Hotel              | Las Vegas | NV         |
...
```

要将空字符串作为 NULL 加载到非数字列，请包含 EMPTYASNULL 或 BLANKSASNULL 选项。可以同时使用这两个选项。

```
unload ('select * from venue')
```

```
to 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' allowoverwrite;

truncate loadvenuenulls;
copy loadvenuenulls from 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' EMPTYASNULL;
```

要确认列包含 NULL，而不只是包含空格或空字符串，请从 LOADVENUENULLS 中选择并针对 null 进行筛选。

```
select * from loadvenuenulls where venuestate is null or venueseats is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
---------+--------------------------+-----------+------------+------------
      72 | Cleveland Browns Stadium | Cleveland |            |      73200
     253 | Mirage Hotel             | Las Vegas | NV         |
     255 | Venetian Hotel           | Las Vegas | NV         |
      22 | Quicken Loans Arena      | Cleveland |            |          0
     101 | Progressive Field        | Cleveland |            |      43345
     251 | Paris Hotel              | Las Vegas | NV         |
     ...
```

使用 ALLOWOVERWRITE 参数卸载

默认情况下，UNLOAD 不会覆盖目标桶中的现有文件。例如，如果您运行同一个 UNLOAD 语句两次，而不修改目标桶中的文件，则第二次运行 UNLOAD 时将失败。要覆盖现有文件（包括清单文件），请指定 ALLOWOVERWRITE 选项。

```
unload ('select * from venue')
to 's3://mybucket/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest allowoverwrite;
```

使用 PARALLEL 和 MANIFEST 参数卸载 EVENT 表

您可以并行卸载表并生成清单文件。Amazon S3 数据文件都是在同一级别创建的，且名称以模式 `0000_part_00` 为后缀。清单文件与数据文件位于同一文件夹级别，并以文本 `manifest` 为后缀。以下 SQL 卸载 EVENT 表并使用基本名称 `parallel` 创建文件

```
unload ('select * from mytickit1.event')
to 's3://my-s3-bucket-name/parallel'
```

```
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
parallel on
manifest;
```

Amazon S3 文件列表与以下内容类似。

```
 Name                        Last modified                          Size
 parallel0000_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 52.1 KB
 parallel0001_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 53.4 KB
 parallel0002_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 52.1 KB
 parallel0003_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 51.1 KB
 parallel0004_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 54.6 KB
 parallel0005_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 53.4 KB
 parallel0006_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 54.1 KB
 parallel0007_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 55.9 KB
 parallelmanifest       -   August 2, 2023, 14:54:39 (UTC-07:00) 886.0 B
```

`parallelmanifest` 文件内容类似于以下内容。

```
{
  "entries": [
    {"url":"s3://my-s3-bucket-name/parallel0000_part_00", "meta": { "content_length":
 53316 }},
    {"url":"s3://my-s3-bucket-name/parallel0001_part_00", "meta": { "content_length":
 54704 }},
    {"url":"s3://my-s3-bucket-name/parallel0002_part_00", "meta": { "content_length":
 53326 }},
    {"url":"s3://my-s3-bucket-name/parallel0003_part_00", "meta": { "content_length":
 52356 }},
    {"url":"s3://my-s3-bucket-name/parallel0004_part_00", "meta": { "content_length":
 55933 }},
    {"url":"s3://my-s3-bucket-name/parallel0005_part_00", "meta": { "content_length":
 54648 }},
    {"url":"s3://my-s3-bucket-name/parallel0006_part_00", "meta": { "content_length":
 55436 }},
    {"url":"s3://my-s3-bucket-name/parallel0007_part_00", "meta": { "content_length":
 57272 }}
  ]
}
```

使用 PARALLEL OFF 和 MANIFEST 参数卸载 EVENT 表

您可以串行卸载表（PARALLEL OFF）并生成清单文件。Amazon S3 数据文件都是在同一级别创建的，且名称以模式 0000 为后缀。清单文件与数据文件位于同一文件夹级别，并以文本 manifest 为后缀。

```
unload ('select * from mytickit1.event')
to 's3://my-s3-bucket-name/serial'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
parallel off
manifest;
```

Amazon S3 文件列表与以下内容类似。

```
 Name                         Last modified                          Size
 serial0000             -     August 2, 2023, 15:54:39 (UTC-07:00) 426.7 KB
 serialmanifest         -     August 2, 2023, 15:54:39 (UTC-07:00) 120.0 B
```

serialmanifest 文件内容类似于以下内容。

```
{
  "entries": [
    {"url":"s3://my-s3-bucket-name/serial000", "meta": { "content_length": 436991 }}
  ]
}
```

使用 PARTITION BY 和 MANIFEST 参数卸载 EVENT 表

您可以按分区卸载表并生成清单文件。在 Amazon S3 中创建了一个包含子分区文件夹的新文件夹，子文件夹中的数据文件的名称模式类似于 0000_par_00。清单文件与子文件夹位于同一文件夹级别，其名称为 manifest。

```
unload ('select * from mytickit1.event')
to 's3://my-s3-bucket-name/partition'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
partition by (eventname)
manifest;
```

Amazon S3 文件列表与以下内容类似。

```
Name                    Type     Last modified                     Size

partition            Folder
```

在 partition 文件夹中是具有分区名称的子文件夹和清单文件。下面显示的是 partition 文件夹中文件夹列表的底部，类似于下面的内容。

```
Name                    Type      Last modified                      Size

...
eventname=Zucchero/    Folder
eventname=Zumanity/    Folder
eventname=ZZ Top/      Folder
manifest               -     August 2, 2023, 15:54:39 (UTC-07:00) 467.6 KB
```

在 eventname=Zucchero/ 文件夹中是类似于以下内容的数据文件。

```
Name               Last modified                       Size
0000_part_00 -   August 2, 2023, 15:59:19 (UTC-07:00) 70.0 B
0001_part_00 -   August 2, 2023, 15:59:16 (UTC-07:00) 106.0 B
0002_part_00 -   August 2, 2023, 15:59:15 (UTC-07:00) 70.0 B
0004_part_00 -   August 2, 2023, 15:59:17 (UTC-07:00) 141.0 B
0006_part_00 -   August 2, 2023, 15:59:16 (UTC-07:00) 35.0 B
0007_part_00 -   August 2, 2023, 15:59:19 (UTC-07:00) 108.0 B
```

manifest 文件内容的底部类似于以下内容。

```
{
  "entries": [
    ...
    {"url":"s3://my-s3-bucket-name/partition/eventname=Zucchero/0007_part_00", "meta":
 { "content_length": 108 }},
    {"url":"s3://my-s3-bucket-name/partition/eventname=Zumanity/0007_part_00", "meta":
 { "content_length": 72 }}
```

```
  ]
}
```

使用 MAXFILESIZE、ROWGROUPSIZE 和 MANIFEST 参数卸载 EVENT 表

您可以并行卸载表并生成清单文件。Amazon S3 数据文件都是在同一级别创建的，且名称以模式 `0000_part_00` 为后缀。生成的 Parquet 数据文件限制为 256MB，行组大小限制为 128MB。清单文件与数据文件位于同一文件夹级别，并以 `manifest` 为后缀。

```
unload ('select * from mytickit1.event')
to 's3://my-s3-bucket-name/eventsize'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
maxfilesize 256 MB
rowgroupsize 128 MB
parallel on
parquet
manifest;
```

Amazon S3 文件列表与以下内容类似。

```
 Name                              Type      Last modified                         Size
 eventsize0000_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.5 KB
 eventsize0001_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.8 KB
 eventsize0002_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.4 KB
 eventsize0003_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.0 KB
 eventsize0004_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 25.3 KB
 eventsize0005_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.8 KB
 eventsize0006_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 25.0 KB
 eventsize0007_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 25.6 KB
 eventsizemanifest                  -       August 2, 2023, 17:35:21 (UTC-07:00) 958.0 B
```

`eventsizemanifest` 文件内容类似于以下内容。

```
{
  "entries": [
    {"url":"s3://my-s3-bucket-name/eventsize0000_part_00.parquet", "meta":
 { "content_length": 25130 }},
    {"url":"s3://my-s3-bucket-name/eventsize0001_part_00.parquet", "meta":
 { "content_length": 25428 }},
```

```
    {"url":"s3://my-s3-bucket-name/eventsize0002_part_00.parquet", "meta":
 { "content_length": 25025 }},
    {"url":"s3://my-s3-bucket-name/eventsize0003_part_00.parquet", "meta":
 { "content_length": 24554 }},
    {"url":"s3://my-s3-bucket-name/eventsize0004_part_00.parquet", "meta":
 { "content_length": 25918 }},
    {"url":"s3://my-s3-bucket-name/eventsize0005_part_00.parquet", "meta":
 { "content_length": 25362 }},
    {"url":"s3://my-s3-bucket-name/eventsize0006_part_00.parquet", "meta":
 { "content_length": 25647 }},
    {"url":"s3://my-s3-bucket-name/eventsize0007_part_00.parquet", "meta":
 { "content_length": 26256 }}
  ]
}
```

UPDATE

主题

- 语法
- 参数
- 使用说明
- UPDATE 语句的示例

如果满足条件，则更新一个或多个表列中的值。

> **Note**
>
> 单个 SQL 语句的最大大小为 16MB。

语法

```
[ WITH [RECURSIVE] common_table_expression [, common_table_expression , ...] ]
            UPDATE table_name [ [ AS ] alias ] SET column = { expression | DEFAULT }
 [,...]

[ FROM fromlist ]
[ WHERE condition ]
```

参数

WITH 子句

可选子句，指定一个或多个 common-table-expressions。请参阅 WITH 子句。

table_name

一个临时或永久表。只有表所有者或对表具有 UPDATE 权限的用户可以更新行。如果您使用 FROM 子句或从表达式或条件中的表进行选择，则必须对这些表具有 SELECT 权限。您不能在此处为表提供别名；不过可以在 FROM 子句中指定别名。

> **Note**
>
> Amazon Redshift Spectrum 外部表为只读。您无法对外部表进行 UPDATE。

alias

目标表的临时备用名称。别名是可选的。AS 关键字始终是可选的。

SET column =

要修改的一个或多个列。未列出的列将保留其当前值。不要将表名包含在目标列的规范中。例如，`UPDATE tab SET tab.col = 1` 是无效的。

expression

一个定义指定列的新值的表达式。

DEFAULT

使用 CREATE TABLE 语句中分配给列的默认值更新列。

FROM tablelist

您可以通过引用其他表中的信息来更新表。在 FROM 子句中列出其他表，或者将子查询用作 WHERE 条件的一部分。FROM 子句中列出的表可以具有别名。如果您需要在列表中包含 UPDATE 语句的目标表，请使用别名。

WHERE condition

一个可选子句，用于限制对符合条件的行的更新。当条件返回 `true` 时，将更新指定的 SET 列。条件可以是列上的简单谓词，也可以是基于子查询的结果的条件。

可以在子查询中命名任何表，包括 UPDATE 的目标表。

使用说明

在更新表中的大量行后：

- 对表执行 Vacuum 操作，以回收存储空间并对行重新排序。
- 分析表以更新查询计划程序的统计数据。

UPDATE 语句的 FROM 子句中不支持左、右和完整外部联接；它们将返回以下错误：

```
ERROR: Target table must be part of an equijoin predicate
```

如果需要指定外部联接，请在 UPDATE 语句的 WHERE 子句中使用子查询。

如果您的 UPDATE 语句需要自联接到目标表，您需要指定联接条件以及限定更新操作的行的 WHERE 子句条件。通常，在将目标表联接到自身或其他表时，最佳实践是使用可明确地将联接条件与限定要更新的行的条件分隔的子查询。

每行具有多个匹配项的 UPDATE 查询会在配置参数 `error_on_nondeterministic_update` 被设置为 true 时引发错误。有关更多信息，请参阅 error_on_nondeterministic_update。

您可以更新 GENERATED BY DEFAULT AS IDENTITY 列。您可以使用您提供的值来更新定义为 GENERATED BY DEFAULT AS IDENTITY 的列。有关更多信息，请参阅 GENERATED BY DEFAULT AS IDENTITY。

UPDATE 语句的示例

有关以下示例中使用的表的更多信息，请参阅示例数据库。

TICKIT 数据库中的 CATEGORY 表包含以下行：

```
+-------+----------+-----------+--------------------------------------------+
| catid | catgroup |  catname  |                  catdesc                   |
+-------+----------+-----------+--------------------------------------------+
| 5     | Sports   | MLS       | Major League Soccer                        |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |
| 1     | Sports   | MLB       | Major League Baseball                      |
| 6     | Shows    | Musicals  | Musical theatre                            |
| 3     | Sports   | NFL       | National Football League                   |
```

```
| 8     | Shows    | Opera     | All opera and light opera                  |
| 2     | Sports   | NHL       | National Hockey League                     |
| 9     | Concerts | Pop       | All rock and pop music concerts            |
| 4     | Sports   | NBA       | National Basketball Association            |
| 7     | Shows    | Plays     | All non-musical theatre                    |
| 10    | Concerts | Jazz      | All jazz singers and bands                 |
+-------+----------+-----------+--------------------------------------------+
```

根据一系列值更新表

基于 CATID 列中的一系列值更新 CATGROUP 列。

```
UPDATE category
SET catgroup='Theatre'
WHERE catid BETWEEN 6 AND 8;

SELECT * FROM category
WHERE catid BETWEEN 6 AND 8;

+-------+----------+----------+---------------------------+
| catid | catgroup | catname  |          catdesc          |
+-------+----------+----------+---------------------------+
| 6     | Theatre  | Musicals | Musical theatre           |
| 7     | Theatre  | Plays    | All non-musical theatre   |
| 8     | Theatre  | Opera    | All opera and light opera |
+-------+----------+----------+---------------------------+
```

根据当前值更新表

基于 CATNAME 和 CATDESC 列的当前 CATGROUP 值更新这两个列：

```
UPDATE category
SET catdesc=default, catname='Shows'
WHERE catgroup='Theatre';

SELECT * FROM category
WHERE catname='Shows';

+-------+----------+---------+---------+
| catid | catgroup | catname | catdesc |
+-------+----------+---------+---------+
| 6     | Theatre  | Shows   | NULL    |
| 7     | Theatre  | Shows   | NULL    |
```

```
| 8     | Theatre  | Shows   | NULL    |
+-------+----------+---------+---------+)
```

在本示例中，CATDESC 列已设置为 null，因为创建表时未定义默认值。

运行以下命令可将 CATEGORY 表数据设置回原始值：

```
TRUNCATE category;

COPY category
FROM 's3://redshift-downloads/tickit/category_pipe.txt'
DELIMITER '|'
IGNOREHEADER 1
REGION 'us-east-1'
IAM_ROLE default;
```

根据 WHERE 子句子查询结果更新表

基于 WHERE 子句中子查询的结果更新 CATEGORY 表：

```
UPDATE category
SET catdesc='Broadway Musical'
WHERE category.catid IN
(SELECT category.catid FROM category
JOIN event ON category.catid = event.catid
JOIN venue ON venue.venueid = event.venueid
JOIN sales ON sales.eventid = event.eventid
WHERE venuecity='New York City' AND catname='Musicals');
```

查看更新后的表：

```
SELECT * FROM category ORDER BY catid;

+-------+----------+-----------+--------------------------------------------+
| catid | catgroup |  catname  |                  catdesc                   |
+-------+----------+-----------+--------------------------------------------+
| 2     | Sports   | NHL       | National Hockey League                     |
| 3     | Sports   | NFL       | National Football League                   |
| 4     | Sports   | NBA       | National Basketball Association            |
| 5     | Sports   | MLS       | Major League Soccer                        |
| 6     | Shows    | Musicals  | Broadway Musical                           |
| 7     | Shows    | Plays     | All non-musical theatre                    |
```

```
| 8     | Shows    | Opera     | All opera and light opera                    |
| 9     | Concerts | Pop       | All rock and pop music concerts              |
| 10    | Concerts | Jazz      | All jazz singers and bands                   |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |
+-------+----------+-----------+----------------------------------------------+
```

根据 WITH 子句子查询结果更新表

要使用 WITH 子句根据子查询的结果更新 CATEGORY 表，请使用以下示例。

```
WITH u1 as (SELECT catid FROM event ORDER BY catid DESC LIMIT 1)
UPDATE category SET catid='200' FROM u1 WHERE u1.catid=category.catid;

SELECT * FROM category ORDER BY catid DESC LIMIT 1;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
| 200   | Concerts | Pop     | All rock and pop music concerts |
+-------+----------+---------+---------------------------------+
```

根据联接条件结果更新表

基于 EVENT 表中匹配的 CATID 行更新 CATEGORY 表中的原始 11 个行：

```
UPDATE category SET catid=100
FROM event
WHERE event.catid=category.catid;

SELECT * FROM category ORDER BY catid;

+-------+----------+-----------+----------------------------------------------+
| catid | catgroup |  catname  |                  catdesc                     |
+-------+----------+-----------+----------------------------------------------+
| 2     | Sports   | NHL       | National Hockey League                       |
| 3     | Sports   | NFL       | National Football League                     |
| 4     | Sports   | NBA       | National Basketball Association              |
| 5     | Sports   | MLS       | Major League Soccer                          |
| 10    | Concerts | Jazz      | All jazz singers and bands                   |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |
| 100   | Concerts | Pop       | All rock and pop music concerts              |
| 100   | Shows    | Plays     | All non-musical theatre                      |
| 100   | Shows    | Opera     | All opera and light opera                    |
```

```
| 100   | Shows    | Musicals  | Broadway Musical                          |
+-------+----------+-----------+-------------------------------------------+
```

请注意，在 FROM 子句中列出 EVENT 表，并且在 WHERE 子句中定义目标表的联接条件。只有 4 行符合更新条件。对于这 4 行，其原始 CATID 值为 6、7、8 和 9；EVENT 表中仅呈现这 4 个类别：

```
SELECT DISTINCT catid FROM event;

+-------+
| catid |
+-------+
| 6     |
| 7     |
| 8     |
| 9     |
+-------+
```

通过扩展上一个示例并向 WHERE 子句添加其他条件来更新 CATEGORY 表中的原始 11 个行。由于 CATGROUP 列的限制，只有一行符合更新条件（虽然这 4 行都符合联接条件）。

```
UPDATE category SET catid=100
FROM event
WHERE event.catid=category.catid
AND catgroup='Concerts';

SELECT * FROM category WHERE catid=100;

+-------+----------+---------+----------------------------------+
| catid | catgroup | catname |             catdesc              |
+-------+----------+---------+----------------------------------+
| 100   | Concerts | Pop     | All rock and pop music concerts |
+-------+----------+---------+----------------------------------+
```

编写此示例的另一种方法如下：

```
UPDATE category SET catid=100
FROM event JOIN category cat ON event.catid=cat.catid
WHERE cat.catgroup='Concerts';
```

这种方法的好处是，明确地将联接条件与限定要更新的行的任何其他条件分隔开。请注意，在 FROM 子句中使用了 CATEGORY 表的别名 CAT。

在 FROM 子句中使用外部联接进行更新

上一个示例显示了 UPDATE 语句的 FROM 子句中指定的内部联接。以下示例返回一个错误，因为 FROM 子句不支持与目标表的外部联接：

```
UPDATE category SET catid=100
FROM event LEFT JOIN category cat ON event.catid=cat.catid
WHERE cat.catgroup='Concerts';
ERROR:  Target table must be part of an equijoin predicate
```

如果 UPDATE 语句需要外部联接，您可以将外部联接语法移到子查询中：

```
UPDATE category SET catid=100
FROM
(SELECT event.catid FROM event LEFT JOIN category cat ON event.catid=cat.catid)
 eventcat
WHERE category.catid=eventcat.catid
AND catgroup='Concerts';
```

在 SET 子句中使用另一个表中的列进行更新

要使用 sales 表中的值更新 TICKIT 示例数据库中的 listing 表，请使用以下示例。

```
SELECT listid, numtickets FROM listing WHERE sellerid = 1 ORDER BY 1 ASC LIMIT 5;

+--------+------------+
| listid | numtickets |
+--------+------------+
| 100423 | 4          |
| 108334 | 24         |
| 117150 | 4          |
| 135915 | 20         |
| 205927 | 6          |
+--------+------------+

UPDATE listing
SET numtickets = sales.sellerid
FROM sales
WHERE sales.sellerid = 1 AND listing.sellerid = sales.sellerid;

SELECT listid, numtickets FROM listing WHERE sellerid = 1 ORDER BY 1 ASC LIMIT 5;
```

```
+--------+------------+
| listid | numtickets |
+--------+------------+
| 100423 | 1          |
| 108334 | 1          |
| 117150 | 1          |
| 135915 | 1          |
| 205927 | 1          |
+--------+------------+
```

VACUUM

对行重新排序，并回收指定表或当前数据库中所有表的空间。

> **Note**
>
> 只有拥有必要的表权限的用户才能有效地对表执行 vacuum 操作。如果在没有必需的表权限的情况下运行 VACUUM 操作，该操作将成功完成，但不起任何作用。有关能有效运行 VACUUM 操作的有效表权限列表，请参阅以下“必需权限”部分。

Amazon Redshift 自动对数据进行排序，并在后台运行 VACUUM DELETE。这减少了运行 VACUUM 命令的需要。有关更多信息，请参阅 对表执行 vacuum 操作。

默认情况下，当任意表中有 95% 的行已有序时，VACUUM 会为该表跳过排序阶段。跳过排序阶段能够显著提高 VACUUM 的性能。要更改某个表的默认排序或删除阈值，请在运行 VACUUM 时包含表名称和 TO threshold PERCENT 参数。

在对表执行 vacuum 操作时，用户可以访问表。您可以在对表执行 vacuum 操作的同时执行查询和写入操作，但如果数据操作语言 (DML) 命令和 vacuum 操作同时运行，则二者可能花费更长时间。如果您在 vacuum 操作期间执行 UPDATE 和 DELETE 语句，则系统性能可能会降低。VACUUM DELETE 会临时阻止更新和删除操作。

Amazon Redshift 在后台自动执行 DELETE ONLY vacuum 操作。当用户运行数据定义语言 (DDL) 操作（如 ALTER TABLE）时，自动 vacuum 操作将暂停。

> **Note**
>
> Amazon Redshift VACUUM 命令语法和行为与 PostgreSQL VACUUM 操作的语法和行为大不相同。例如，Amazon Redshift 中的默认 VACUUM 操作是 VACUUM FULL，该操作可回收磁

盘空间并对所有行进行重新排序。相比之下，PostgreSQL 中的默认 VACUUM 操作只能回收空间并使其可供重复使用。

有关更多信息，请参阅 对表执行 vacuum 操作。

所需的权限

以下是 VACUUM 所需的权限：

- Superuser
- 具有 VACUUM 权限的用户
- 表拥有者
- 向其共享表的数据库拥有者

语法

```
VACUUM [ FULL | SORT ONLY | DELETE ONLY | REINDEX | RECLUSTER ]
[ [ table_name ] [ TO threshold PERCENT ] [ BOOST ] ]
```

参数

FULL

对指定的表（或当前数据库中的所有表）进行排序，并回收由前面的 UPDATE 和 DELETE 操作标记为删除的行所占用的磁盘空间。VACUUM FULL 是默认值。

完全 vacuum 操作不会为交错的表重建索引。要在执行完全 vacuum 操作后对交错的表重建索引，请使用 VACUUM REINDEX 选项。

如果任意表已至少 95% 有序，则默认情况下 VACUUM FULL 会为其跳过排序阶段。如果 VACUUM 能够跳过排序阶段，它将执行 DELETE ONLY 并在删除阶段回收空间，这样，至少有 95% 的剩余行不会被标记为删除。

如果未达到排序阈值（例如，如果对 90% 的行进行了排序）且 VACUUM 执行了完全排序，则 VACUUM 也会执行一次彻底删除操作，从而从 100% 的已删除行恢复空间。

您只能为单个表更改默认 vacuum 阈值。要更改某个表的默认 vacuum 阈值，请包含表名称和 TO threshold PERCENT 参数。

SORT ONLY

对指定的表（或当前数据库中的所有表）进行排序，而不回收由删除的表所释放的空间。当回收磁盘空间不重要但对新行进行重新排序很重要时，此选项非常有用。当未排序的区域不包含大量已删除行且不跨整个已排序区域时，SORT ONLY vacuum 将减少 vacuum 操作的运行时间。如果应用程序不具有磁盘空间限制，但依赖于要求表行保持排序状态的查询优化，则可从此类 vacuum 操作获益。

默认情况下，VACUUM SORT ONLY 将跳过任何至少有 95% 的行已排序的表。要更改某个表的默认排序阈值，请在运行 VACUUM 时包含表名称和 TO threshold PERCENT 参数。

DELETE ONLY

Amazon Redshift 在后台自动执行 DELETE ONLY vacuum 操作，因此，您很少需要运行 DELETE ONLY vacuum。

VACUUM DELETE 回收由前面的 UPDATE 和 DELETE 操作标记为删除的行所占用的磁盘空间，并压缩表以释放占用的空间。DELETE ONLY vacuum 操作不对表数据进行排序。

当回收磁盘空间很重要但对新行进行重新排序不重要时，此选项可减少 vacuum 操作的运行时间。此外，当查询性能已处于最佳状态，且不要求对行重新排序来优化查询性能时，此选项也很有用。

默认情况下，VACUUM DELETE ONLY 将回收空间，这样，至少有 95% 的剩余行不会被标记为删除。要更改某个表的默认删除阈值，请在运行 VACUUM 时包含表名称和 TO threshold PERCENT 参数。

一些操作（例如 `ALTER TABLE APPEND`）可能会导致对表进行分片。当您使用 `DELETE ONLY`子句时，vacuum 操作将从分片的表中回收空间。95% 的同一阈值适用于碎片整理操作。

REINDEX tablename

分析交错排序键列中的值的分配，然后执行完全 VACUUM 操作。如果使用 REINDEX，则表名称是必需的。

VACUUM REINDEX 所需的时间显著大于 VACUUM FULL 所需的时间，因为它需要执行额外的过程来分析交错排序键。对于交错表，排序和合并操作所花费的时间更长，因为与复合排序相比，交错排序需要重新排列的行可能更多。

如果 VACUUM REINDEX 操作在完成前终止，则下一个 VACUUM 将恢复重建索引操作，然后执行完全 vacuum 操作。

VACUUM REINDEX 不支持 TO threshold PERCENT。

table_name

要执行 vacuum 操作的表的名称。如果不指定表名称，vacuum 操作将应用于当前数据库中的所有表。您可以指定任何永久或临时的用户创建的表。此命令对于其他对象（例如，视图和系统表）没有用处。

如果包含 TO threshold PERCENT 参数，则必须指定表名称。

RECLUSTER tablename

对表中未排序的部分进行排序。表中已按自动表排序进行排序的部分保持不变。此命令不会将新排序的数据与排序区域合并。也不会回收标记为删除的所有空间。完成此命令后，表可能不会显示完全排序，如 SVV_TABLE_INFO 中的 `unsorted` 字段所示。

我们建议您对具有频繁摄入和仅访问最新数据的查询操作的大型表使用 VACUUM RECLUSTER。

VACUUM RECLUSTER 不支持 TO threshold PERCENT。如果使用 RECLUSTER，则表名称是必需的。

对于具有交错排序键的表和具有 ALL 分配方式的表，不支持 VACUUM RECLUSTER。

table_name

要执行 vacuum 操作的表的名称。您可以指定任何永久或临时的用户创建的表。此命令对于其他对象（例如，视图和系统表）没有用处。

TO threshold PERCENT

一个子句，指定超过时 VACUUM 将跳过排序阶段的阈值和删除阶段中回收空间的目标阈值。排序阈值 是执行 vacuum 操作之前在指定表中已排序的行占总行数的百分比。 删除阈值 是执行 vacuum 操作之后未标记为删除的行占总行数的最小百分比。

由于 VACUUM 仅会在表中已排序行所占的百分比低于排序阈值时对行进行重新排序，Amazon Redshift 经常可以大幅减少 VACUUM 次数。同样，当 VACUUM 不受限于从 100% 的标记为删除的行回收空间时，它通常能够跳过重写仅包含几个已删除行的数据块的过程。

例如，如果您为 threshold 指定 75，则当表中 75% 或以上的行已有序时，VACUUM 会跳过排序阶段。对于删除阶段，VACUUMS 将设置一个回收磁盘空间的目标，这样，在执行 vacuum 操作之后，表中至少有 75% 的行不会被标记为删除。threshold 值必须为介于 0 到 100 之间的整数。默认值为 95。如果您指定了值 100，VACUUM 将始终对表进行排序（除非它已完全排序）并从标记为删除的所有行回收空间。如果您指定的值为 0，则 VACUUM 将从不对表进行排序，也从不会回收空间。

如果您包含 TO threshold PERCENT 参数，则必须同时指定表名称。如果忽略表名称，则 VACUUM 操作将失败。

您不能将 TO threshold PERCENT 参数用于 REINDEX。

BOOST

使用可用的其他资源（例如内存和磁盘空间）运行 VACUUM 命令。利用 BOOST 选项，VACUUM 在一个窗口中运行，并在 VACUUM 操作期间阻止并发删除和更新。使用 BOOST 选项运行会争用系统资源，这可能会影响查询性能。当系统负载很小时（例如，在维护操作期间），运行 VACUUM BOOST。

在使用 BOOST 选项时，注意以下事项：

- 在指定 BOOST 时，需要 table_name 值。
- 不支持将 BOOST 与 REINDEX 结合使用。
- BOOST 与 DELETE ONLY 结合使用时将被忽略。

使用说明

对于大多数 Amazon Redshift 应用程序，建议执行完全 vacuum 操作。有关更多信息，请参阅 对表执行 vacuum 操作。

在运行 vacuum 操作之前，请注意以下行为：

- 您不能在事务数据块 (BEGIN ... END) 中运行 VACUUM。有关事务的更多信息，请参阅 可序列化的隔离。
- 在任何给定时间，您只能在一个集群上运行一个 VACUUM 命令。如果您尝试同时运行多个 vacuum 操作，Amazon Redshift 将返回错误。
- 对表执行 vacuum 操作时，表可能会出现一定量的增长。当没有可回收空间的已删除行时，或者表的新排序顺序导致数据压缩率降低时，这是预期行为。
- 在 vacuum 操作期间，会出现一定程度的查询性能降级。一旦 vacuum 操作完成，性能就将恢复正常。
- 并发写入操作可在 vacuum 操作期间执行，但我们不建议在进行 vacuum 操作时执行写入操作。更高效的方法是，先完成写入操作，然后再运行 vacuum 操作。此外，不能对 vacuum 操作开始后写入的任何数据执行此操作。在这种情况下，有必要执行另一个 vacuum 操作。
- 如果加载或插入操作已在进行中，则 vacuum 操作可能无法开始。Vacuum 操作临时需要表的独占访问权限才能开始。要求此独占访问权限的时间非常短，vacuum 操作不会阻止任何重要时段内的并发加载和插入操作。

- 当不需要对特定表执行 vacuum 操作时，将跳过 vacuum 操作；不过，发现可跳过该操作会产生一些开销。如果您知道某个表是原始表或未达到 vacuum 阈值，则不要对其运行 vacuum 操作。
- 对小型表执行 DELETE ONLY vacuum 操作可能不会减少用于存储数据的块数，特别是当表具有大量列或集群为每个节点使用大量切片时。这些 vacuum 操作会按每个切片、每个列增加一个块，来支持对表执行的并行插入；相比通过回收磁盘空间减少块计数带来的好处，产生的开销可能会得不偿失。例如，如果在执行 vacuum 操作之前，一个 8 节点集群中包含 10 列的表占用了 1000 个块，则 vacuum 不会减少实际块计数，除非由于已删除的行导致回收 80 个以上的磁盘空间块。（每个数据块使用 1MB。）

在满足下列任意条件时，自动 vacuum 操作暂停：

- 用户运行数据定义语言 (DDL) 操作（例如 ALTER TABLE），该操作需要自动 vacuum 当前正在处理的表上的独占锁定。
- 用户触发了集群中任意表上的 VACUUM（一次只能运行一个 VACUUM）。
- 高集群负载期间。

示例

根据默认的 95% vacuum 阈值回收所有表中的空间和数据库并对这些表中的行进行重新排序。

```
vacuum;
```

根据默认的 95% 阈值回收 SALES 表中的空间并对该表中的行进行重新排序。

```
vacuum sales;
```

始终回收 SALES 表中的空间并对该表中的行进行重新排序。

```
vacuum sales to 100 percent;
```

仅当 SALES 表中有序行的比例低于 75% 时对该表中的行进行重新排序。

```
 vacuum sort only sales to 75 percent;
```

回收 SALES 表中的空间，以使至少 75% 的剩余行不会在 vacuum 操作之后被标记为删除。

```
vacuum delete only sales to 75 percent;
```

为 LISTING 表重建索引，然后对该表执行 vacuum 操作。

```
vacuum reindex listing;
```

以下命令会返回错误。

```
vacuum reindex listing to 75 percent;
```

重建集群，然后对 LISTING 表执行 vacuum 操作。

```
vacuum recluster listing;
```

重建集群，然后使用 BOOST 选项对 LISTING 表执行 vacuum 操作。

```
vacuum recluster listing boost;
```

SQL 函数参考

主题

- 仅领导节点函数
- 仅计算节点函数
- 聚合函数
- 数组函数
- 按位聚合函数
- 条件表达式
- 数据类型格式设置函数
- 日期和时间函数
- 哈希函数
- HyperLogLog 函数
- JSON 函数
- 机器学习函数。

- 数学函数
- 对象函数
- 空间函数
- 字符串函数
- SUPER 类型信息函数
- VARBYTE 函数和运算符
- 窗口函数
- 系统管理函数
- 系统信息函数

Amazon Redshift 支持大量作为 SQL 标准的扩展的函数以及标准聚合函数、标量函数和窗口函数。

> **Note**
>
> Amazon Redshift 基于 PostgreSQL。Amazon Redshift 和 PostgreSQL 之间的差别非常大，您在设计和开发数据仓库应用程序时必须注意这一点。有关 Amazon Redshift SQL 与 PostgreSQL 之间的差异的更多信息，请参阅Amazon Redshift 和 PostgreSQL。

仅领导节点函数

一些 Amazon Redshift 查询是在计算节点上分发和运行的；而另一些查询仅在领导节点上运行。

当查询引用用户创建的表或系统表（具有 STL 或 STV 前缀的表和具有 SVL 或 SVV 前缀的系统视图）时，领导节点就会将 SQL 分发到计算节点。仅引用目录表（具有 PG 前缀的表（如 PG_TABLE_DEF））或不引用任何表的查询在领导节点上以独占方式运行。

部分 Amazon Redshift SQL 函数仅在领导节点上受支持，在计算节点上不受支持。使用领导节点函数的查询必须在领导节点上而不是计算节点上以独占方式执行，否则它将返回错误。

每个仅领导节点函数的文档均包含一个注释，指示该函数将在引用用户定义的表或 Amazon Redshift 系统表时返回错误。

有关更多信息，请参阅 在领导节点上支持的 SQL 函数。

以下 SQL 函数为仅领导节点函数且在计算节点上不受支持：

系统信息函数

- CURRENT_SCHEMA
- CURRENT_SCHEMAS
- HAS_DATABASE_PRIVILEGE
- HAS_SCHEMA_PRIVILEGE
- HAS_TABLE_PRIVILEGE

字符串函数

- SUBSTR

数学函数

- FACTORIAL()

以下仅领导节点函数已被弃用，不再受支持：

日期函数

- AGE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- LOCALTIME
- ISFINITE
- NOW

字符串函数

- GETBIT
- GET_BYTE
- SET_BIT
- SET_BYTE
- TO_ASCII

仅计算节点函数

某些 Amazon Redshift 查询必须只在计算节点上运行。如果查询引用用户创建的表，则 SQL 在计算节点上运行。

仅引用目录表（具有 PG 前缀的表（如 PG_TABLE_DEF））或不引用任何表的查询在领导节点上以独占方式运行。

如果使用计算节点函数的查询不引用用户定义的表，否则 Amazon Redshift 系统表会返回以下错误。

```
[Amazon](500310) Invalid operation: One or more of the used functions must be applied
 on at least one user created table.
```

每个仅计算节点函数的文档均包含一个注释，指示函数将在查询引用用户定义的表或 Amazon Redshift 系统表时返回错误。

以下 SQL 函数是仅计算节点函数：

- LISTAGG
- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC 和 APPROXIMATE PERCENTILE_DISC

聚合函数

主题

- ANY_VALUE 函数
- APPROXIMATE PERCENTILE_DISC 函数
- AVG 函数
- COUNT 函数
- LISTAGG 函数
- MAX 函数
- MEDIAN 函数
- MIN 函数
- PERCENTILE_CONT 函数
- STDDEV_SAMP 和 STDDEV_POP 函数

- SUM 函数
- VAR_SAMP 和 VAR_POP 函数

聚合函数从一组输入值计算单个结果值。

使用聚合函数的 SELECT 语句可以包含两个可选子句：GROUP BY 和 HAVING。这些子句的语法如下（使用 COUNT 函数作为示例）：

```
SELECT count (*) expression FROM table_reference
WHERE condition [GROUP BY expression ] [ HAVING condition]
```

GROUP BY 子句聚合结果并按指定的一列或多列中的唯一值对结果进行分组。HAVING 子句限制返回到满足某个特定聚合条件（如 count (*) > 1）的行中的结果。HAVING 子句的使用方式与 WHERE 用来基于列值限制行的方式相同。有关这些附加子句的示例，请参阅COUNT。

聚合函数不接受嵌套聚合函数或窗口函数作为参数。

ANY_VALUE 函数

ANY_VALUE 函数以非确定方式返回输入表达式值中的任何值。如果输入表达式未导致任何行被返回，则此函数返回 NULL。如果输入表达式中有 NULL 值，则此函数也可以返回 NULL。

语法

```
ANY_VALUE( [ DISTINCT | ALL ] expression )
```

参数

DISTINCT | ALL

指定 DISTINCT 或 ALL 以从输入表达式值中返回任何值。DISTINCT 参数没有任何效果，将被忽略。

expression

对其执行函数的目标列或表达式。表达式为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT

- DECIMAL
- REAL
- DOUBLE PRECISON
- BOOLEAN
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND
- VARBYTE
- SUPER
- HLLSKETCH
- GEOMETRY
- GEOGRAPHY

返回值

返回与 expression 相同的数据类型。

使用说明

如果为列指定 ANY_VALUE 函数的语句也包含第二列引用，则第二列必须出现在 GROUP BY 子句中或包含在聚合函数中。

示例

这些示例使用在《Amazon Redshift 入门指南》的步骤 4：从 Amazon S3 中加载示例数据中创建的事件表。以下示例返回事件名称为 Eagles. 的任何日期 ID 的实例。

```
select any_value(dateid) as dateid, eventname from event where eventname ='Eagles'
 group by eventname;
```

以下是结果。

```
dateid | eventname
-------+---------------
 1878  | Eagles
```

以下示例返回事件名称为 Eagles 或 Cold War Kids 的任何日期 ID 的实例。

```
select any_value(dateid) as dateid, eventname from event where eventname in('Eagles',
 'Cold War Kids') group by eventname;
```

以下是结果。

```
dateid | eventname
-------+---------------
 1922  | Cold War Kids
 1878  | Eagles
```

APPROXIMATE PERCENTILE_DISC 函数

APPROXIMATE PERCENTILE_DISC 是一种假定离散分布模型的逆分布函数。该函数具有一个百分比值和一个排序规范，并返回给定集合中的元素。近似值可以大幅加快函数运行速度，相对错误率较低，约为 0.5%。

如果给定百分位数值，APPROXIMATE PERCENTILE_DISC 会使用分位数摘要算法估计 ORDER BY 子句中的表达式的离散百分位数。APPROXIMATE PERCENTILE_DISC 返回的值具有最小的积累分布值（针对同一排序规范），该值大于或等于百分位数。

APPROXIMATE PERCENTILE_DISC 是仅计算节点函数。如果查询不引用用户定义的表或 Amazon Redshift 系统表，该函数将返回错误。

语法

```
APPROXIMATE  PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
```

参数

percentile

介于 0 和 1 之间的数字常数。计算中将忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定用于排序和计算百分比的数字或日期/时间值的子句。

返回值

与 WITHIN GROUP 子句中的 ORDER BY 表达式相同的数据类型。

使用说明

如果 APPROXIMATE PERCENTILE_DISC 语句包括 GROUP BY 子句，结果集将受限。这一限制将根据节点类型和节点数量发生变化。如果超出限制，函数将失败并会返回以下错误。

```
GROUP BY limit for approximate percentile_disc exceeded.
```

如果您需要评估的组数量超出了限制，请考虑使用 PERCENTILE_CONT 函数。

示例

以下示例返回销售数量、销售总额和排名前十位日期的第五十个百分位数。

```
select top 10 date.caldate,
count(totalprice), sum(totalprice),
approximate percentile_disc(0.5)
within group (order by totalprice)
from listing
join date on listing.dateid = date.dateid
group by date.caldate
order by 3 desc;

caldate    | count | sum        | percentile_disc
-----------+-------+------------+----------------
2008-01-07 |   658 | 2081400.00 |         2020.00
2008-01-02 |   614 | 2064840.00 |         2178.00
2008-07-22 |   593 | 1994256.00 |         2214.00
2008-01-26 |   595 | 1993188.00 |         2272.00
2008-02-24 |   655 | 1975345.00 |         2070.00
2008-02-04 |   616 | 1972491.00 |         1995.00
2008-02-14 |   628 | 1971759.00 |         2184.00
2008-09-01 |   600 | 1944976.00 |         2100.00
2008-07-29 |   597 | 1944488.00 |         2106.00
2008-07-23 |   592 | 1943265.00 |         1974.00
```

AVG 函数

AVG 函数返回输入表达式值的平均值（算术平均值）。AVG 函数使用数值并忽略 NULL 值。

语法

```
AVG ( [ DISTINCT | ALL ] expression )
```

参数

expression

对其执行函数的目标列或表达式。表达式为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT
- NUMERIC
- DECIMAL
- REAL
- DOUBLE PRECISON
- SUPER

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算平均值之前消除指定表达式中的所有重复值。利用参数 ALL，该函数可保留表达式中的所有重复值以计算平均值。ALL 是默认值。

数据类型

AVG 函数支持的参数类型为
SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL、DOUBLE PRECISION 和 SUPER。

AVG 函数支持的返回类型为：

- 适用于任何整数类型参数的 BIGINT
- 适用于浮点型参数的 DOUBLE PRECISION
- 返回与任何其他参数类型的表达式相同的数据类型。

带有 NUMERIC 或 DECIMAL 参数的 AVG 函数结果的默认精度为 38。结果的小数位数与参数的小数位数相同。例如，DEC(5,2) 列的 AVG 返回 DEC(38,2) 数据类型。

示例

从 SALES 表中查找每笔交易所售的平均产品数：

```
select avg(qtysold)from sales;

avg
-----
2
(1 row)
```

查找所有列表中列出的平均总价：

```
select avg(numtickets*priceperticket) as avg_total_price from listing;

avg_total_price
-----------------
3034.41
(1 row)
```

查找平均支付价格 (以降序顺序按月分组)：

```
select avg(pricepaid) as avg_price, month
from sales, date
where sales.dateid = date.dateid
group by month
order by avg_price desc;

avg_price | month
-----------+-------
659.34 | MAR
655.06 | APR
645.82 | JAN
643.10 | MAY
642.72 | JUN
642.37 | SEP
640.72 | OCT
640.57 | DEC
635.34 | JUL
```

```
635.24 | FEB
634.24 | NOV
632.78 | AUG
(12 rows)
```

COUNT 函数

COUNT 函数对由表达式定义的行计数。

COUNT 函数具有以下变体。

- COUNT (*) 对目标表中的所有行计数，无论它们是否包含 null 值。
- COUNT (expression) 计算某个特定列或表达式中带非 NULL 值的行的数量。
- COUNT (DISTINCT expression) 计算某个列或表达式中非重复的非 NULL 值的数量。
- APPROXIMATE COUNT DISTINCT 估算某个列或表达式中非重复的非 NULL 值的数量。

语法

```
COUNT( * | expression )
```

```
COUNT ( [ DISTINCT | ALL ] expression )
```

```
APPROXIMATE COUNT ( DISTINCT expression )
```

参数

expression

对其执行函数的目标列或表达式。COUNT 函数支持所有参数数据类型。

DISTINCT | ALL

利用参数 DISTINCT，该函数可在执行计数之前消除指定表达式中的所有重复值。利用参数 ALL，该函数可保留表达式中的所有重复值以进行计数。ALL 是默认值。

APPROXIMATE

与 APPROXIMATE 结合使用时，COUNT DISTINCT 函数使用 HyperLogLog 算法来估算某个列或表达式中非重复的非 NULL 值的数量。使用 APPROXIMATE 关键字的查询，运行速度会快得多，

错误率相对较低，约为 2%。返回大量非重复值的查询可提供近似值，每个查询或组（如果有按子句划分的组）中有几百万或更多非重复值。对于较少数量（以千为单位）的非重复值，近似值计算可能慢于精确计数。APPROXIMATE 只能与 COUNT DISTINCT 结合使用。

返回类型

COUNT 函数返回 BIGINT。

示例

对来自佛罗里达州的所有用户计数：

```
select count(*) from users where state='FL';

count
-------
510
```

对 EVENT 表中的所有事件名称计数：

```
select count(eventname) from event;

count
-------
8798
```

对 EVENT 表中的所有事件名称计数：

```
select count(all eventname) from event;

count
-------
8798
```

对 EVENT 表中的所有唯一场地 ID 计数：

```
select count(distinct venueid) as venues from event;

venues
--------
```

```
204
```

计算每个卖家列出 4 张以上门票出售的批次的次数。按卖家 ID 对结果进行分组：

```
select count(*), sellerid from listing
where numtickets > 4
group by sellerid
order by 1 desc, 2;

count | sellerid
------+----------
12    |    6386
11    |    17304
11    |    20123
11    |    25428
...
```

以下示例比较 COUNT 和 APPROXIMATE COUNT 的返回值与执行时间。

```
select  count(distinct pricepaid) from sales;

count
-------
  4528


Time: 48.048 ms


select approximate count(distinct pricepaid) from sales;

count
-------
  4553


Time: 21.728 ms
```

LISTAGG 函数

对于查询中的每个组，LISTAGG 聚合函数根据 ORDER BY 表达式对该组的行进行排序，然后将值串联成一个字符串。

LISTAGG 是仅计算节点函数。如果查询不引用用户定义的表或 Amazon Redshift 系统表，该函数将返回错误。有关更多信息，请参阅 查询目录表。

语法

```
LISTAGG( [DISTINCT] aggregate_expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
```

参数

DISTINCT

用于在串联之前消除指定表达式中重复值的子句。将忽略尾随空格。例如，字符串 'a' 和 'a ' 视为重复项。LISTAGG 将使用遇到的第一个值。有关更多信息，请参阅 尾部空格的意义。

aggregate_expression

提供要聚合的值的任何有效表达式，如列名称。忽略 NULL 值和空字符串。

分隔符

用于分隔串联的值的字符串常数。默认值为 NULL。

WITHIN GROUP (ORDER BY order_list)

用于指定聚合值的排序顺序的子句。

返回值

VARCHAR(MAX)。如果结果集大于最大 VARCHAR 大小，则 LISTAGG 返回以下错误：

```
Invalid operation: Result size exceeds LISTAGG limit
```

使用说明

- 如果语句包含多个使用 WITHIN GROUP 子句的 LISTAGG 函数，则每个 WITHIN GROUP 子句必须使用相同的 ORDER BY 值。

  例如，以下语句将返回错误。

  ```
  SELECT LISTAGG(sellerid)
  WITHIN GROUP (ORDER BY dateid) AS sellers,
  ```

```
LISTAGG(dateid)
WITHIN GROUP (ORDER BY sellerid) AS dates
FROM sales;
```

以下各条语句将成功运行。

```
SELECT LISTAGG(sellerid)
WITHIN GROUP (ORDER BY dateid) AS sellers,
LISTAGG(dateid)
WITHIN GROUP (ORDER BY dateid) AS dates
FROM sales;

SELECT LISTAGG(sellerid)
WITHIN GROUP (ORDER BY dateid) AS sellers,
LISTAGG(dateid) AS dates
FROM sales;
```

示例

以下示例聚合卖家 ID（按卖家 ID 进行排序）。

```
SELECT LISTAGG(sellerid, ', ')
WITHIN GROUP (ORDER BY sellerid)
FROM sales
WHERE eventid = 4337;

listagg

-------------------------------------------------------------------------------------------------
380, 380, 1178, 1178, 1178, 2731, 8117, 12905, 32043, 32043, 32043, 32432, 32432,
 38669, 38750, 41498, 45676, 46324, 47188, 47188, 48294
```

以下示例使用 DISTINCT 返回唯一卖家 ID 的列表。

```
SELECT LISTAGG(DISTINCT sellerid, ', ')
WITHIN GROUP (ORDER BY sellerid)
FROM sales
WHERE eventid = 4337;

listagg
```

```
-------------------------------------------------------------------------------------------------
380, 1178, 2731, 8117, 12905, 32043, 32432, 38669, 38750, 41498, 45676, 46324, 47188,
 48294
```

以下示例按日期顺序聚合卖家 ID。

```
SELECT LISTAGG(sellerid, ', ')
WITHIN GROUP (ORDER BY dateid)
FROM sales
WHERE eventid = 4337;

   listagg
-------------------------------------------------------------------------------------------------
 41498, 47188, 47188, 1178, 1178, 1178, 380, 45676, 46324, 48294, 32043, 32043, 32432,
 12905, 8117, 38750, 2731, 32432, 32043, 380, 38669
```

以下示例返回 ID 为 660 的买家的销售日期的竖线分隔的列表。

```
SELECT LISTAGG(
    (SELECT caldate FROM date WHERE date.dateid=sales.dateid), ' | '
)
WITHIN GROUP (ORDER BY sellerid DESC, salesid ASC)
FROM sales
WHERE buyerid = 660;

             listagg
-------------------------------------------------
2008-07-16 | 2008-07-09 | 2008-01-01 | 2008-10-26
```

以下示例返回买家 ID 660、661 和 662 的销售 ID 的逗号分隔列表。

```
SELECT buyerid,
LISTAGG(salesid,', ')
WITHIN GROUP (ORDER BY salesid) AS sales_id
FROM sales
WHERE buyerid BETWEEN 660 AND 662
GROUP BY buyerid
ORDER BY buyerid;

buyerid |                sales_id
--------+-------------------------------------------------
660     | 32872, 33095, 33514, 34548
```

```
661     | 19951, 20517, 21695, 21931
662     | 3318, 3823, 4215, 51980, 53202, 55908, 57832, 171603
```

MAX 函数

MAX 函数返回一组行中的最大值。可以使用 DISTINCT 或 ALL，但不会影响结果。

语法

```
MAX ( [ DISTINCT | ALL ] expression )
```

参数

expression

对其执行函数的目标列或表达式。表达式为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISON
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算最大值之前消除指定表达式中的所有重复值。利用参数 ALL，该函数可保留表达式中的所有重复值以计算最大值。ALL 是默认值。

数据类型

返回与 expression 相同的数据类型。MIN 函数的等效布尔值为 BOOL_AND 函数，MAX 函数的等效布尔值为 BOOL_OR 函数。

示例

从所有销售中查找支付的最高价格：

```
select max(pricepaid) from sales;

max
----------
12624.00
(1 row)
```

从所有销售中查找每张门票的已支付最高价格：

```
select max(pricepaid/qtysold) as max_ticket_price
from sales;

max_ticket_price
-----------------
2500.00000000
(1 row)
```

MEDIAN 函数

计算一系列值的中值。此范围内的 NULL 值将被忽略。

MEDIAN 是一种假定连续分布模型的逆分布函数。

MEDIAN 是 PERCENTILE_CONT 的特殊情况。

MEDIAN 是仅计算节点函数。如果查询不引用用户定义的表或 Amazon Redshift 系统表，该函数将返回错误。

语法

```
MEDIAN(median_expression)
```

参数

median_expression

对其执行函数的目标列或表达式。

数据类型

返回类型由 median_expression 的数据类型确定。下表显示了每种 median_expression 数据类型的返回类型。

输入类型	返回类型
INT2, INT4, INT8, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

使用说明

如果 median_expression 参数是使用 38 位最大精度定义的 DECIMAL 数据类型，则 MEDIAN 可能将返回不准确的结果或错误。如果 MEDIAN 函数的返回值超过 38 位，则结果将截断以符合规范，这将导致精度降低。如果在插值期间，中间结果超出最大精度，则会发生数值溢出且函数会返回错误。要避免这些情况，建议使用具有较低精度的数据类型或将 median_expression 参数转换为较低精度。

如果语句包括对基于排序的聚合函数 (LISTAGG、PERCENTILE_CONT 或 MEDIAN) 的多个调用，则它们必须全都使用相同的 ORDER BY 值。请注意，MEDIAN 对表达式值应用隐式排序依据。

例如，以下语句将返回错误。

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;
```

```
An error occurred when executing the SQL command:
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

ERROR: within group ORDER BY clauses for aggregate functions must be the same
```

以下语句将成功运行。

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(salesid)
FROM sales
GROUP BY salesid, pricepaid;
```

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

以下示例显示，MEDIAN 生成和 PERCENTILE_CONT(0.5) 相同的结果。

```
SELECT TOP 10 DISTINCT sellerid, qtysold,
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qtysold),
MEDIAN(qtysold)
FROM sales
GROUP BY sellerid, qtysold;

+----------+---------+-----------------+--------+
| sellerid | qtysold | percentile_cont | median |
+----------+---------+-----------------+--------+
|        2 |       2 |               2 |      2 |
|       26 |       1 |               1 |      1 |
|       33 |       1 |               1 |      1 |
|       38 |       1 |               1 |      1 |
|       43 |       1 |               1 |      1 |
|       48 |       2 |               2 |      2 |
|       48 |       3 |               3 |      3 |
|       77 |       4 |               4 |      4 |
|       85 |       4 |               4 |      4 |
|       95 |       2 |               2 |      2 |
```

```
+----------+---------+-----------------+---------+
```

以下示例查找每个 sellerid 的销售量中值。

```
SELECT sellerid,
MEDIAN(qtysold)
FROM sales
GROUP BY sellerid
ORDER BY sellerid
LIMIT 10;

+----------+--------+
| sellerid | median |
+----------+--------+
|        1 |    1.5 |
|        2 |      2 |
|        3 |      2 |
|        4 |      2 |
|        5 |      1 |
|        6 |      1 |
|        7 |    1.5 |
|        8 |      1 |
|        9 |      4 |
|       12 |      2 |
+----------+--------+
```

要验证第一个 sellerid 的上一次查询的结果，请使用以下示例。

```
SELECT qtysold
FROM sales
WHERE sellerid=1;

+---------+
| qtysold |
+---------+
|       2 |
|       1 |
+---------+
```

MIN 函数

MIN 函数返回一组行中的最小值。可以使用 DISTINCT 或 ALL，但不会影响结果。

语法

```
MIN ( [ DISTINCT | ALL ] expression )
```

参数

expression

对其执行函数的目标列或表达式。表达式为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISON
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算最小值之前消除指定表达式中的所有重复值。利用参数 ALL，该函数可保留表达式中的所有重复值以计算最小值。ALL 是默认值。

数据类型

返回与 expression 相同的数据类型。MIN 函数的等效布尔值为 BOOL_AND 函数，MAX 函数的等效布尔值为 BOOL_OR 函数。

示例

从所有销售中查找支付的最低价格：

```
select min(pricepaid) from sales;

min
-------
20.00
(1 row)
```

从所有销售中查找每张门票的已支付最低价格：

```
select min(pricepaid/qtysold)as min_ticket_price
from sales;

min_ticket_price
------------------
20.00000000
(1 row)
```

PERCENTILE_CONT 函数

PERCENTILE_CONT 是一种假定连续分布模型的逆分布函数。该函数具有一个百分比值和一个排序规范，并返回一个在有关排序规范的给定百分比值范围内的内插值。

PERCENTILE_CONT 在对值进行排序后计算值之间的线性内插。通过在聚合组中使用百分比值 (P) 和非 null 行数 (N)，该函数会在根据排序规范对行进行排序后计算行号。根据公式 (RN) 计算此行号 RN = (1+ (P*(N-1))。聚合函数的最终结果通过行号 CRN = CEILING(RN) 和 FRN = FLOOR(RN) 的行中的值之间的线性内插计算。

最终结果将如下所示。

如果 (CRN = FRN = RN)，则结果为 (value of expression from row at RN)

否则，结果将如下所示：

(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN).

PERCENTILE_CONT 是仅计算节点函数。如果查询不引用用户定义的表或 Amazon Redshift 系统表，该函数将返回错误。

语法

```
PERCENTILE_CONT(percentile)
WITHIN GROUP(ORDER BY expr)
```

参数

percentile

介于 0 和 1 之间的数值常数。计算中将忽略 NULL 值。

expr

指定用于排序和计算百分比的数字或日期/时间值。

返回值

返回类型由 WITHIN GROUP 子句中的 ORDER BY 表达式的数据类型决定。下表显示了每种个 ORDER BY 表达式数据类型的返回类型。

输入类型	返回类型
INT2, INT4, INT8, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

使用说明

如果 ORDER BY 表达式是使用 38 位最大精度定义的 DECIMAL 数据类型，则 PERCENTILE_CONT 可能将返回不准确的结果或错误。如果 PERCENTILE_CONT 函数的返回值超过 38 位，结果将截断以符合规范，这将导致精度降低。如果在插值期间，中间结果超出最大精度，则会发生数值溢出且函数会返回错误。要避免这些情况，建议使用具有较低精度的数据类型或将 ORDER BY 表达式转换为较低精度。

如果语句包括对基于排序的聚合函数 (LISTAGG、PERCENTILE_CONT 或 MEDIAN) 的多个调用，则它们必须全都使用相同的 ORDER BY 值。请注意，MEDIAN 对表达式值应用隐式排序依据。

例如，以下语句将返回错误。

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

An error occurred when executing the SQL command:
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

ERROR: within group ORDER BY clauses for aggregate functions must be the same
```

以下语句将成功运行。

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(salesid)
FROM sales
GROUP BY salesid, pricepaid;
```

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

以下示例显示 PERCENTILE_CONT(0.5) 生成与 MEDIAN 相同的结果。

```
SELECT TOP 10 DISTINCT sellerid, qtysold,
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qtysold),
MEDIAN(qtysold)
FROM sales
GROUP BY sellerid, qtysold;

+----------+---------+-----------------+--------+
| sellerid | qtysold | percentile_cont | median |
+----------+---------+-----------------+--------+
```

```
|        2 |       2 |                2 |      2 |
|       26 |       1 |                1 |      1 |
|       33 |       1 |                1 |      1 |
|       38 |       1 |                1 |      1 |
|       43 |       1 |                1 |      1 |
|       48 |       2 |                2 |      2 |
|       48 |       3 |                3 |      3 |
|       77 |       4 |                4 |      4 |
|       85 |       4 |                4 |      4 |
|       95 |       2 |                2 |      2 |
+----------+---------+------------------+--------+
```

以下示例为 SALES 表中每个 sellerid 的销售量找到 PERCENTILE_CONT(0.5) 和 PERCENTILE_CONT(0.75)。

```
SELECT sellerid,
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qtysold) as pct_05,
PERCENTILE_CONT(0.75) WITHIN GROUP(ORDER BY qtysold) as pct_075
FROM sales
GROUP BY sellerid
ORDER BY sellerid
LIMIT 10;

+----------+--------+---------+
| sellerid | pct_05 | pct_075 |
+----------+--------+---------+
|        1 |    1.5 |    1.75 |
|        2 |      2 |    2.25 |
|        3 |      2 |       3 |
|        4 |      2 |       2 |
|        5 |      1 |     1.5 |
|        6 |      1 |       1 |
|        7 |    1.5 |    1.75 |
|        8 |      1 |       1 |
|        9 |      4 |       4 |
|       12 |      2 |    3.25 |
+----------+--------+---------+
```

要验证第一个 sellerid 的上一次查询的结果，请使用以下示例。

```
SELECT qtysold
FROM sales
WHERE sellerid=1;
```

```
+---------+
| qtysold |
+---------+
|       2 |
|       1 |
+---------+
```

STDDEV_SAMP 和 STDDEV_POP 函数

STDDEV_SAMP 和 STDDEV_POP 函数返回一组数值（整数、小数或浮点）的样本标准差和总体标准差。STDDEV_SAMP 函数的结果等于同一组值的样本方差的平方根。

STDDEV_SAMP 和 STDDEV 是同一函数的同义词。

语法

```
STDDEV_SAMP | STDDEV ( [ DISTINCT | ALL ] expression)
STDDEV_POP ( [ DISTINCT | ALL ] expression)
```

表达式必须具有整数、小数或浮点数据类型。无论表达式的数据类型如何，此函数的返回类型都是双精度数。

> **Note**
>
> 使用浮点算法计算标准偏差，其计算结果可能会稍微不准确。

使用说明

当计算包含一个值的表达式的样本标准差（STDDEV 或 STDDEV_SAMP）时，函数的结果为 NULL 而不是 0。

示例

以下查询返回 VENUE 表的 VENUESEATS 列中各值的平均数，后跟同一组值的样本标准差和总体标准差。VENUESEATS 是一个 INTEGER 列。结果的小数位数已减少至 2 位。

```
select avg(venueseats),
cast(stddev_samp(venueseats) as dec(14,2)) stddevsamp,
cast(stddev_pop(venueseats) as dec(14,2)) stddevpop
from venue;
```

```
avg  | stddevsamp | stddevpop
-------+------------+-----------
17503 |   27847.76 |  27773.20
(1 row)
```

以下查询返回 SALES 表中 COMMISSION 列的样本标准差。COMMISSION 是一个 DECIMAL 列。结果的小数位数已减少至 10 位。

```
select cast(stddev(commission) as dec(18,10))
from sales;

stddev
----------------
130.3912659086
(1 row)
```

以下查询将 COMMISSION 列的样本标准差转换为整数。

```
select cast(stddev(commission) as integer)
from sales;

stddev
--------
130
(1 row)
```

以下查询返回 COMMISSION 列的样本标准差和样本方差的平方根。这些计算的结果相同。

```
select
cast(stddev_samp(commission) as dec(18,10)) stddevsamp,
cast(sqrt(var_samp(commission)) as dec(18,10)) sqrtvarsamp
from sales;

stddevsamp   |  sqrtvarsamp
----------------+----------------
130.3912659086 | 130.3912659086
(1 row)
```

SUM 函数

SUM 函数返回输入列值或表达式值的和。SUM 函数使用数值并忽略 NULL 值。

语法

```
SUM ( [ DISTINCT | ALL ] expression )
```

参数

expression

对其执行函数的目标列或表达式。表达式为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT
- NUMERIC
- DECIMAL
- REAL
- DOUBLE PRECISON
- SUPER

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算和之前消除指定表达式中的所有重复值。利用参数 ALL，该函数可保留表达式中的所有重复值以计算和。ALL 是默认值。

数据类型

SUM 函数支持的参数类型为
SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL、DOUBLE PRECISION 和 SUPER。

SUM 函数支持的返回类型为

- 适用于 BIGINT、SMALLINT 和 INTEGER 参数的 BIGINT
- 适用于 NUMERIC 参数的 NUMERIC
- 适用于浮点参数的 DOUBLE PRECISION
- 返回与任何其他参数类型的表达式相同的数据类型。

带有 NUMERIC 或 DECIMAL 参数的 SUM 函数结果的默认精度为 38。结果的小数位数与参数的小数位数相同。例如，DEC(5,2) 列的 SUM 返回 DEC(38,2) 数据类型。

示例

从 SALES 表中查找所有已付佣金的和：

```
select sum(commission) from sales;

sum
-------------
16614814.65
(1 row)
```

查找佛罗里达州的所有场地的座位数：

```
select sum(venueseats) from venue
where venuestate = 'FL';

sum
--------
250411
(1 row)
```

查找 5 月份售出的座位数：

```
select sum(qtysold) from sales, date
where sales.dateid = date.dateid and date.month = 'MAY';

sum
-------
32291
(1 row)
```

VAR_SAMP 和 VAR_POP 函数

VAR_SAMP 和 VAR_POP 函数返回一组数值（整数、小数或浮点）的样本方差和总体方差。VAR_SAMP 函数的结果等于同一组值的样本标准差的平方。

VAR_SAMP 和 VARIANCE 是同一函数的同义词。

语法

```
VAR_SAMP | VARIANCE ( [ DISTINCT | ALL ] expression)
VAR_POP ( [ DISTINCT | ALL ] expression)
```

表达式必须具有整数、小数或浮点数据类型。无论表达式的数据类型如何，此函数的返回类型都是双精度数。

> **Note**
>
> 这些函数的结果可能跨数据仓库集群而异，具体取决于每个案例中集群的配置。

使用说明

当计算包含一个值的表达式的样本方差（VARIANCE 或 VAR_SAMP）时，函数的结果为 NULL 而不是 0。

示例

以下查询返回 LISTING 表中 NUMTICKETS 列的已取整样本方差和总体方差。

```
select avg(numtickets),
round(var_samp(numtickets)) varsamp,
round(var_pop(numtickets)) varpop
from listing;

avg | varsamp | varpop
-----+---------+--------
10 |      54 |     54
(1 row)
```

以下查询运行相同的计算但将结果转换为小数值。

```
select avg(numtickets),
cast(var_samp(numtickets) as dec(10,4)) varsamp,
cast(var_pop(numtickets) as dec(10,4)) varpop
from listing;

avg | varsamp | varpop
-----+---------+---------
10 | 53.6291 | 53.6288
```

```
(1 row)
```

数组函数

下面，您可以找到 Amazon Redshift 支持用于访问和操作数组的 SQL 数组函数的描述。

主题

数组函数

创建 SUPER 数据类型的数组。

语法

```
ARRAY( [ expr1 ] [ , expr2 [ , ... ] ] )
```

参数

expr1、expr2

除日期和时间类型以外的任何 Amazon Redshift 数据类型的表达式，因为 Amazon Redshift 不会将日期和时间类型转换为 SUPER 数据类型。参数不需要为相同的数据类型。

返回类型

数组函数返回 SUPER 数据类型。

示例

以下示例显示了一个数值数组和一个不同数据类型的数组。

```
--an array of numeric values
select array(1,50,null,100);
```

```
      array
------------------
 [1,50,null,100]
(1 row)

--an array of different data types
select array(1,'abc',true,3.14);
        array
----------------------
 [1,"abc",true,3.14]
(1 row)
```

array_concat 函数

array_concat 函数连接两个数组来创建一个数组，该数组包含第一个数组中的所有元素，然后包含第二个数组中的所有元素。这两个参数必须是有效的数组。

语法

```
array_concat( super_expr1,  super_expr2 )
```

参数

super_expr1

指定要连接的两个数组中的第一个数组的值。

super_expr2

指定要连接的两个数组中的第二个数组的值。

返回类型

array_concat 函数返回一个 SUPER 数据值。

示例

以下示例显示了相同类型的两个数组的连接以及两个不同类型的数组的连接。

```
-- concatenating two arrays
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY(10003,10004));
              array_concat
-----------------------------------
```

```
 [10001,10002,10003,10004]
(1 row)

-- concatenating two arrays of different types
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY('ab','cd'));
          array_concat
------------------------------
 [10001,10002,"ab","cd"]
(1 row)
```

array_flatten 函数

将多个数组合并为 SUPER 类型的单个数组。

语法

```
array_flatten( super_expr1,super_expr2,.. )
```

参数

super_expr1、super_expr2

数组形式的有效 SUPER 表达式。

返回类型

array_flatten 函数返回一个 SUPER 数据值。

示例

以下示例显示 array_flatten 函数。

```
SELECT ARRAY_FLATTEN(ARRAY(ARRAY(1,2,3,4),ARRAY(5,6,7,8),ARRAY(9,10)));
     array_flatten
-----------------------
 [1,2,3,4,5,6,7,8,9,10]
(1 row)
```

get_array_length 函数

返回指定数组的长度。GET_ARRAY_LENGTH 函数在给定对象或数组路径的情况下返回 SUPER 数组的长度。

语法

```
get_array_length( super_expr )
```

参数

super_expr

数组形式的有效 SUPER 表达式。

返回类型

get_array_length 函数返回 BIGINT。

示例

以下示例显示 get_array_length 函数。

```
SELECT GET_ARRAY_LENGTH(ARRAY(1,2,3,4,5,6,7,8,9,10));
 get_array_length
----------------------
            10
(1 row)
```

split_to_array 函数

将分隔符用作可选参数。如果不存在分隔符，则默认值为逗号。

语法

```
split_to_array( string,delimiter )
```

参数

string

要拆分的输入字符串。

分隔符

输入字符串将在其上拆分的可选值。默认值为逗号。

返回类型

split_to_array 函数返回一个 SUPER 数据值。

示例

以下示例显示 split_to_array 函数。

```
SELECT SPLIT_TO_ARRAY('12|345|6789', '|');
     split_to_array
------------------------
 ["12","345","6789"]
(1 row)
```

子数组函数

操作数组以返回输入数组的子集。

语法

```
SUBARRAY( super_expr, start_position, length )
```

参数

super_expr

数组形式的有效 SUPER 表达式。

start_position

数组中开始提取的位置，从索引位置 0 开始。负位置从数组的末尾向后计数。

length

要提取的元素的数量（子字符串的长度）。

返回类型

子数组函数返回一个 SUPER 数据值。

示例

以下是子数组函数输出的示例。

```
 SELECT SUBARRAY(ARRAY('a', 'b', 'c', 'd', 'e', 'f'), 2, 3);
   subarray
---------------
 ["c","d","e"]
(1 row)
```

按位聚合函数

按位聚合函数会进行位运算，以聚合整数列和可转换或舍入为整数值的列。

主题

在按位聚合中使用 NULL

在将按位函数应用于可为 null 的列时，在计算函数结果之前将消除任何 NULL 值。如果没有行符合聚合资格，则按位函数将返回 NULL。相同的行为适用于常规的聚合函数。以下为示例。

```
select sum(venueseats), bit_and(venueseats) from venue
where venueseats is null;

sum  | bit_and
------+---------
null |    null
(1 row)
```

按位聚合的 DISTINCT 支持

与其他聚合函数相同的是，按位函数也支持 DISTINCT 关键字。

但是，将 DISTINCT 用于这些函数不会影响结果。值的第一个实例足以满足按位 AND 或 OR 操作。如果重复值出现在正在计算的表达式中，不会造成任何差异。

由于 DISTINCT 处理可能会产生一些查询执行开销，我们建议不要将 DISTINCT 用于按位函数。

按位函数的概述示例

下面，您可以找到一些概述示例，说明如何使用按位函数。您还可以通过每个函数描述找到具体的代码示例。

按位函数的示例以 TICKIT 示例数据库为基础。TICKIT 示例数据库中的 USERS 表包含多个布尔列，该类列指示每个用户是否喜欢各种类型的活动，如运动、戏剧、歌剧等。下面是一个示例。

```
select userid, username, lastname, city, state,
likesports, liketheatre
from users limit 10;

userid | username | lastname  |     city     | state | likesports | liketheatre
-------+----------+-----------+--------------+-------+------------+-------------
1 | JSG99FHE | Taylor    | Kent         | WA    | t          | t
9 | MSD36KVR | Watkins   | Port Orford  | MD    | t          | f
```

假定新版本的 USERS 表是以不同的方式构建的。在此新版本中，包含一个整数列，该列定义（以二进制格式）每个用户喜欢或不喜欢的 8 种类型的事件。在此设计中，每个位位置均代表一类活动。某个喜欢全部 8 种类型的用户已将全部 8 个位设置为 1（如下表的第一行中所示）。不喜欢任何这些活动的用户已将全部 8 个位设置为 0（请见第二行）。仅喜欢运动和爵士乐的用户显示在以下第三行中：

	SPORTS	THEATRE	JAZZ	OPERA	ROCK	VEGAS	BROADW	CLASSICAL
用户 1	1	1	1	1	1	1	1	1
用户 2	0	0	0	0	0	0	0	0
用户 3	1	0	1	0	0	0	0	0

在数据库表中，这些二进制值可作为整数存储在一个 LIKES 列中，如下所示。

User	二进制值	存储的值（整数）
用户 1	11111111	255
用户 2	00000000	0

User	二进制值	存储的值（整数）
用户 3	10100000	160

BIT_AND 函数

BIT_AND 函数会对单个整数列或表达式中的所有值运行按位 AND 运算。此函数会聚合与表达式中的每个整数值对应的每个二进制值的每个位。

如果所有值中没有设置为 1 的位，则 BIT_AND 函数将返回结果 0。如果所有值中的一个或多个位设置为 1，该函数返回一个整数值。此整数是对应于这些位的二进制值的数字。

例如，表中的一个列包含 4 个整数值：3、7、10 和 22。这些整数用二进制格式表示，如下所示：

整数	二进制值
3	11
7	111
10	1010
22	10110

针对此数据集的 BIT_AND 操作发现所有位仅在倒数第二的位置设置为 1。结果是一个二进制值 00000010，它表示整数值 2。因此，BIT_AND 函数返回 2。

语法

```
BIT_AND ( [DISTINCT | ALL] expression )
```

参数

expression

对其执行函数的目标列或表达式。此表达式必须具有 INT、INT2 或 INT8 数据类型。该函数返回等同的 INT、INT2 或 INT8 数据类型。

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算结果之前消除指定表达式的所有重复值。利用参数 ALL，该函数可保留所有重复值。ALL 是默认值。有关更多信息，请参阅 按位聚合的 DISTINCT 支持。

示例

假定有意义的商业信息存储在整数列中，您可使用按位函数提取和聚合该信息。以下查询将 BIT_AND 函数应用于名为“USERLIKES”的表的 LIKES 列中并按 CITY 列对结果进行分组。

```
select city, bit_and(likes) from userlikes group by city
order by city;
city          | bit_and
--------------+---------
Los Angeles   |       0
Sacramento    |       0
San Francisco |       0
San Jose      |      64
Santa Barbara |     192
(5 rows)
```

您可以将这些结果解释如下：

- Santa Barbara 的整数值 192 转换为二进制值 11000000。换句话说，此城市中的所有用户都喜欢运动和戏剧，但并非所有用户都喜欢任何其他类型的活动。
- 整数 64 转换为 01000000。因此，对于圣荷西的所有用户来说，他们喜欢的唯一活动类型是戏剧。
- 其他三个城市的值 0 表示这些城市中的所有用户没有共同的“喜好”。

BIT_OR 函数

BIT_OR 函数会对单个整数列或表达式中的所有值运行按位 OR 运算。此函数会聚合与表达式中的每个整数值对应的每个二进制值的每个位。

例如，假设您的表中的一个列包含 4 个整数值：3、7、10 和 22。这些整数用二进制格式表示，如下所示。

整数	二进制值
3	11

整数	二进制值
7	111
10	1010
22	10110

如果将 BIT_OR 函数应用于整数值集合，则该操作将查找 1 在每个位置找到的任何值。在这种情况下，1 存在于至少一个值的后 5 个位置，从而生成二进制结果 00011111；因此，该函数将返回 31（或 16 + 8 + 4 + 2 + 1）。

语法

```
BIT_OR ( [DISTINCT | ALL] expression )
```

参数

expression

对其执行函数的目标列或表达式。此表达式必须具有 INT、INT2 或 INT8 数据类型。该函数返回等同的 INT、INT2 或 INT8 数据类型。

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算结果之前消除指定表达式的所有重复值。利用参数 ALL，该函数可保留所有重复值。ALL 是默认值。有关更多信息，请参阅 按位聚合的 DISTINCT 支持。

示例

以下查询会将 BIT_OR 函数应用于名为“USERLIKES”的表的 LIKES 列并按 CITY 列对结果进行分组。

```
select city, bit_or(likes) from userlikes group by city
order by city;
city          | bit_or
--------------+--------
Los Angeles   |    127
Sacramento    |    255
San Francisco |    255
San Jose      |    255
```

```
Santa Barbara |    255
(5 rows)
```

对于所列的四个城市，至少有一个用户喜欢全部活动类型 (255=11111111)。对于洛杉矶，至少有一个用户喜欢除运动之外的所有活动类型 (127=01111111)。

BOOL_AND 函数

BOOL_AND 函数在单个布尔/整数列或表达式上运行。此函数将类似的逻辑应用于 BIT_AND 和 BIT_OR 函数。对于此函数，返回类型为布尔值 (`true` 或 `false`) 。

如果集合中的所有值为 true，则 BOOL_AND 函数返回 `true` (`t`)。如果任何值为 false，该函数返回 `false` (`f`)。

语法

```
BOOL_AND ( [DISTINCT | ALL] expression )
```

参数

expression

对其执行函数的目标列或表达式。此表达式必须具有 BOOLEAN 或整数数据类型。该函数的返回类型为 BOOLEAN。

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算结果之前消除指定表达式的所有重复值。利用参数 ALL，该函数可保留所有重复值。ALL 是默认值。有关更多信息，请参阅 [按位聚合的 DISTINCT 支持]。

示例

您可以对布尔表达式或整数表达式使用布尔函数。例如，以下查询从 TICKIT 数据库中的标准 USERS 表返回结果，该表包含多个布尔列。

BOOL_AND 函数对所有 5 个行返回 `false`。并非每个州的所有用户都喜欢运动。

```
select state, bool_and(likesports) from users
group by state order by state limit 5;

state | bool_and
------+---------
```

```
AB    | f
AK    | f
AL    | f
AZ    | f
BC    | f
(5 rows)
```

BOOL_OR 函数

BOOL_OR 函数在单个布尔/整数列或表达式上运行。此函数将类似的逻辑应用于 BIT_AND 和 BIT_OR 函数。对于此函数，返回类型为布尔值（`true`、`false` 或 NULL）。

如果集合中的一个或多个值为 `true`，则 BOOL_OR 函数返回 `true`（`t`）。如果集合中的所有值为 `false`，则函数返回 `false`（`f`）。如果值未知，则可以返回 NULL。

语法

```
BOOL_OR ( [DISTINCT | ALL] expression )
```

参数

expression

对其执行函数的目标列或表达式。此表达式必须具有 BOOLEAN 或整数数据类型。该函数的返回类型为 BOOLEAN。

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算结果之前消除指定表达式的所有重复值。利用参数 ALL，该函数可保留所有重复值。ALL 是默认值。请参阅 按位聚合的 DISTINCT 支持。

示例

您可以将布尔函数与布尔表达式或整数表达式结合使用。例如，以下查询从 TICKIT 数据库中的标准 USERS 表返回结果，该表包含多个布尔列。

BOOL_OR 函数对所有 5 个行返回 `true`。每个州中至少有一个用户喜欢运动。

```
select state, bool_or(likesports) from users
group by state order by state limit 5;

state | bool_or
```

```
------+--------
AB    | t
AK    | t
AL    | t
AZ    | t
BC    | t
(5 rows)
```

以下示例返回 NULL。

```
SELECT BOOL_OR(NULL = '123')
               bool_or
------
NULL
```

条件表达式

主题

- CASE 条件表达式
- DECODE 函数
- GREATEST 和 LEAST 函数
- NVL 和 COALESCE 函数
- NVL2 函数
- NULLIF 函数

Amazon Redshift 支持 SQL 标准的扩展的一些条件表达式。

CASE 条件表达式

CASE 表达式是一种条件表达式，类似于其他语言中发现的 if/then/else 语句。CASE 用于指定存在多个条件时的结果。在 SQL 表达式有效的情况下使用 CASE，例如在 SELECT 命令中。

有两种类型的 CASE 表达式：简单和搜索。

- 在简单 CASE 表达式中，将一个表达式与一个值比较。在找到匹配项时，将应用 THEN 子句中的指定操作。如果未找到匹配项，则应用 ELSE 子句中的操作。
- 在搜索 CASE 表达式中，基于布尔表达式计算每个 CASE，而且 CASE 语句会返回第一个匹配的 CASE。如果在 WHEN 子句中未找到匹配，则返回 ELSE 子句中的操作。

语法

用于匹配条件的简单 CASE 语句：

```
CASE expression
  WHEN value THEN result
  [WHEN...]
  [ELSE result]
END
```

用于计算每个条件的搜索 CASE 语句：

```
CASE
  WHEN condition THEN result
  [WHEN ...]
  [ELSE result]
END
```

参数

expression

一个列名称或任何有效的表达式。

值

与该表达式比较的值，如数字常数或字符串。

result

计算表达式或布尔条件时返回的目标值或表达式。所有结果表达式的数据类型必须可转换为单一输出类型。

condition

计算结果为 true 或 false 的 Boolean 表达式。如果 condition 为 true，则 CASE 表达式的值是符合条件的结果，不处理 CASE 表达式的其余部分。如果 condition 为 false，则计算任何后续的 WHEN 子句。如果没有 WHEN 条件结果为 true，则 CASE 表达式的值是 ELSE 子句的结果。如果没有 ELSE 子句且没有条件为 true，则结果为 null。

示例

以下示例使用 TICKIT 样本数据中的 VENUE 表和 SALES 表。有关更多信息，请参阅 示例数据库。

使用简单 CASE 表达式在针对 VENUE 表的查询中将 New York City 替换为 Big Apple。将所有其他城市名称替换为 other。

```
select venuecity,
  case venuecity
    when 'New York City'
    then 'Big Apple' else 'other'
  end
from venue
order by venueid desc;

venuecity        |   case
-----------------+-----------
Los Angeles      | other
New York City    | Big Apple
San Francisco    | other
Baltimore        | other
...
```

使用搜索 CASE 表达式来基于单个门票销售的 PRICEPAID 值分配组编号：

```
select pricepaid,
  case when pricepaid <10000 then 'group 1'
    when pricepaid >10000 then 'group 2'
    else 'group 3'
  end
from sales
order by 1 desc;

pricepaid |  case
----------+---------
12624     | group 2
10000     | group 3
10000     | group 3
9996      | group 1
9988      | group 1
...
```

DECODE 函数

DECODE 表达式将一个特定值替换为另一个特定值或默认值，具体取决于等式条件的结果。此操作等同于简单 CASE 表达式或 IF-THEN-ELSE 语句的操作。

语法

```
DECODE ( expression, search, result [, search, result ]... [ ,default ] )
```

此类表达式对于将存储在表中的缩写或代码替换为报表所需的有意义的业务值非常有用。

参数

expression

要比较的值的源，如表中的列。

搜索

与源表达式比较的目标值，如数字值或字符串。搜索表达式的计算结果必须为一个固定值。您无法指定计算结果为值范围（如 `age between 20 and 29`）的表达式；您需要为要替换的每个值指定单独的搜索/结果对。

搜索表达式的所有实例的数据类型必须相同或兼容。expression 和 search 参数也必须兼容。

result

查询在表达式匹配搜索值时返回的替换值。您必须在 DECODE 表达式中包含至少一个搜索/结果对。

结果表达式的所有实例的数据类型必须相同或兼容。result 和 default 参数也必须兼容。

默认值

搜索条件失败时用于案例的可选默认值。如果您未指定默认值，则 DECODE 表达式返回 NULL。

使用说明

如果 expression 值和 search 值都为 NULL，则 DECODE 结果为对应的 result 值。有关此函数的使用的说明，请参阅“示例”部分。

以此方式使用时，DECODE 类似于 NVL2 函数，但存在一些区别。有关这些区别的说明，请参阅 NVL2 使用说明。

示例

当值 `2008-06-01` 存在于 datetable 的 caldate 列中时，以下示例会将该值替换为 `June 1st, 2008`。此示例将所有其他 caldate 值替换为 NULL。

```
select decode(caldate, '2008-06-01', 'June 1st, 2008')
from datetable where month='JUN' order by caldate;

case
----------------
June 1st, 2008

...
(30 rows)
```

以下示例使用 DECODE 表达式将 CATEGORY 表中的 5 个缩写的 CATNAME 列转换为全名并将该列中的其他值转换为 Unknown。

```
select catid, decode(catname,
'NHL', 'National Hockey League',
'MLB', 'Major League Baseball',
'MLS', 'Major League Soccer',
'NFL', 'National Football League',
'NBA', 'National Basketball Association',
'Unknown')
from category
order by catid;

catid  | case
-------+---------------------------------
1      | Major League Baseball
2      | National Hockey League
3      | National Football League
4      | National Basketball Association
5      | Major League Soccer
6      | Unknown
7      | Unknown
8      | Unknown
9      | Unknown
10     | Unknown
11     | Unknown
(11 rows)
```

使用 DECODE 表达式在 VENUESEATS 列中查找带 NULL 值的科罗拉多州和内华达州的 5 个场地；将这些 NULL 值转换为零。如果 VENUESEATS 列不为 NULL，则返回 1 作为结果。

```
select venuename, venuestate, decode(venueseats,null,0,1)
```

```
from venue
where venuestate in('NV','CO')
order by 2,3,1;

venuename               | venuestate     | case
-----------------------------+----------------+-----------
Coors Field                  | CO         |   1
Dick's Sporting Goods Park   | CO         |   1
Ellie Caulkins Opera House   | CO         |   1
INVESCO Field         | CO         |   1
Pepsi Center         | CO         |   1
Ballys Hotel         | NV         |   0
Bellagio Hotel               | NV         |   0
Caesars Palace               | NV         |   0
Harrahs Hotel                | NV         |   0
Hilton Hotel                 | NV         |   0
...
(20 rows)
```

GREATEST 和 LEAST 函数

从包含任何数量的表达式的列表中返回最大值或最小值。

语法

```
GREATEST (value [, ...])
LEAST (value [, ...])
```

参数

expression_list

表达式的逗号分隔的列表，如列名称。这些表达式都必须可转换为常见数据类型。忽略该列表中的 NULL 值。如果所有表达式的计算结果为 NULL，则结果为 NULL。

返回值

从所提供的表达式列表中返回最大值（对于 GREATEST）或最小值（对于 LEAST）。

示例

以下示例按字母顺序返回 `firstname` 或 `lastname` 的最高值。

```
select firstname, lastname, greatest(firstname,lastname) from users
where userid < 10
order by 3;

 firstname | lastname  | greatest
-----------+-----------+-----------
 Lars      | Ratliff   | Ratliff
 Reagan    | Hodge     | Reagan
 Colton    | Roy       | Roy
 Barry     | Roy       | Roy
 Tamekah   | Juarez    | Tamekah
 Rafael    | Taylor    | Taylor
 Victor    | Hernandez | Victor
 Vladimir  | Humphrey  | Vladimir
 Mufutau   | Watkins   | Watkins
(9 rows)
```

NVL 和 COALESCE 函数

返回表达式系列中不为 null 的第一个表达式的值。当找到非 null 值时，将不计算该列表中的剩余表达式。

NVL 与 COALESCE 相同。它们是同义词。本主题说明了其语法，并提供这两者的示例。

语法

```
NVL( expression, expression, ... )
```

用于 COALESCE 的语法是相同的：

```
COALESCE( expression, expression, ... )
```

如果所有表达式为 null，则结果为 null。

如果您要在主要值缺失或为 null 时返回次要值，则这些函数非常有用。例如，一个查询可能会返回前三个可用电话号码中的第一个：手机、家庭或工作号码。函数中表达式的顺序决定了计算结果的顺序。

参数

expression

一个要针对 null 状态进行计算的表达式，如列名称。

返回类型

Amazon Redshift 根据输入表达式确定返回值的数据类型。如果输入表达式的数据类型不是通用类型，则会返回错误。

示例

如果列表包含整数表达式，则该函数返回一个整数。

```
SELECT COALESCE(NULL, 12, NULL);

coalesce
--------------
12
```

此示例与前面的示例相同（不同之处在于它使用 NVL），返回相同的结果。

```
SELECT NVL(NULL, 12, NULL);

coalesce
--------------
12
```

以下示例返回字符串类型。

```
SELECT COALESCE(NULL, 'Amazon Redshift', NULL);

coalesce
--------------
Amazon Redshift
```

以下示例会导致错误，因为表达式列表中的数据类型有变化。在这种情况下，列表中既有字符串类型，也有数字类型。

```
SELECT COALESCE(NULL, 'Amazon Redshift', 12);
ERROR: invalid input syntax for integer: "Amazon Redshift"
```

对于本例，您创建一个包含 START_DATE 和 END_DATE 列的表，插入几个包含 null 值的行，然后将 NVL 表达式应用于这两个列。

```
create table datetable (start_date date, end_date date);
```

```
insert into datetable values ('2008-06-01','2008-12-31');
insert into datetable values (null,'2008-12-31');
insert into datetable values ('2008-12-31',null);
```

```
select nvl(start_date, end_date)
from datetable
order by 1;

coalesce
------------
2008-06-01
2008-12-31
2008-12-31
```

NVL 表达式的默认列名称为 COALESCE。以下查询将返回相同的结果：

```
select coalesce(start_date, end_date)
from datetable
order by 1;
```

在以下示例查询中，您创建一个包含示例酒店预订信息的表，然后插入几行。一些记录包含 null 值。

```
create table booking_info (booking_id int, booking_code character(8), check_in date,
 check_out date, funds_collected numeric(12,2));
```

插入以下示例数据。一些记录没有 check_out 日期或 funds_collected 数量。

```
insert into booking_info values (1, 'OCEAN_WV', '2023-02-01','2023-02-03',100.00);
insert into booking_info values (2, 'OCEAN_WV', '2023-04-22','2023-04-26',120.00);
insert into booking_info values (3, 'DSRT_SUN', '2023-03-13','2023-03-16',125.00);
insert into booking_info values (4, 'DSRT_SUN', '2023-06-01','2023-06-03',140.00);
insert into booking_info values (5, 'DSRT_SUN', '2023-07-10',null,null);
insert into booking_info values (6, 'OCEAN_WV', '2023-08-15',null,null);
```

以下查询返回日期列表。如果 check_out 日期不可用，它会列出 check_in 日期。

```
select coalesce(check_out, check_in)
from booking_info
order by booking_id;
```

结果如下。请注意，最后两条记录显示了 check_in 日期。

```
coalesce
------------
2023-02-03
2023-04-26
2023-03-16
2023-06-03
2023-07-10
2023-08-15
```

如果您希望查询为特定函数或列返回 null 值，则可使用 NVL 表达式将这些 null 值替换为其他一些值。例如，聚合函数（如 SUM）在没有要计算的行时会返回 null 值而不是零。您可以使用 NVL 表达式将这些 null 值替换为 700.0。对 funds_collected 求和的结果不是 485，而是 1885，因为值为 null 的两行被替换为 700。

```
select sum(nvl(funds_collected, 700.0)) as sumresult from booking_info;

sumresult
------
 1885
```

NVL2 函数

根据指定表达式的计算结果是 NULL 还是 NOT NULL 返回这两个值之一。

语法

```
NVL2 ( expression, not_null_return_value, null_return_value )
```

参数

expression

一个要针对 null 状态进行计算的表达式，如列名称。

not_null_return_value

在 expression 的计算结果为 NOT NULL 时返回的值。not_null_return_value 值必须具有与 expression 相同的数据类型或可隐式转换为该数据类型。

null_return_value

在 expression 的计算结果为 NULL 时返回的值。null_return_value 值必须具有与 expression 相同的数据类型或可隐式转换为该数据类型。

返回类型

按以下方式确定 NVL2 返回类型：

- 如果 not_null_return_value 或 null_return_value 为 null，则返回非 null 表达式的数据类型。

如果 not_null_return_value 和 null_return_value 都不为 null：

- 如果 not_null_return_value 和 null_return_value 具有相同的数据类型，则返回该数据类型。
- 如果 not_null_return_value 和 null_return_value 具有不同的数字数据类型，则返回最小的可兼容数字数据类型。
- 如果 not_null_return_value 和 null_return_value 具有不同的日期时间数据类型，则返回时间戳数据类型。
- 如果 not_null_return_value 和 null_return_value 具有不同的字符数据类型，则返回 not_null_return_value 的数据类型。
- 如果 not_null_return_value 和 null_return_value 具有混合的数字和非数字数据类型，则返回 not_null_return_value 的数据类型。

> **Important**
>
> 在最后两个示例中（其中返回 not_null_return_value 的数据类型），null_return_value 将隐式转换为该数据类型。如果数据类型不兼容，则该函数将失败。

使用说明

当 expression 和 search 参数均为 null 时，DECODE 函数 可以通过与 NVL2 类似的方式使用。区别在于：在 DECODE 中，返回内容将同时具有 result 参数的值和数据类型。相反，在 NVL2 中，返回内容将具有 not_null_return_value 或 null_return_value 参数的值（不管函数选择哪一个），但将具有 not_null_return_value 的数据类型。

例如，假定 column1 为 NULL，则以下查询将返回相同的值。但是，DECODE 返回值数据类型将为 INTEGER，NVL2 返回值数据类型将为 VARCHAR。

```
select decode(column1, null, 1234, '2345');
select nvl2(column1, '2345', 1234);
```

示例

以下示例修改一些示例数据，然后计算两个字段以为用户提供相应的联系人信息：

```
update users set email = null where firstname = 'Aphrodite' and lastname = 'Acevedo';

select (firstname + ' ' + lastname) as name,
nvl2(email, email, phone) AS contact_info
from users
where state = 'WA'
and lastname  like 'A%'
order by lastname, firstname;

name        contact_info
--------------------+--------------------------------------------
Aphrodite Acevedo (906) 632-4407
Caldwell Acevedo  Nunc.sollicitudin@Duisac.ca
Quinn Adams    vel@adipiscingligulaAenean.com
Kamal Aguilar  quis@vulputaterisusa.com
Samson Alexander  hendrerit.neque@indolorFusce.ca
Hall Alford    ac.mattis@vitaediamProin.edu
Lane Allen     et.netus@risusDonec.org
Xander Allison    ac.facilisis.facilisis@Infaucibus.com
Amaya Alvarado    dui.nec.tempus@eudui.edu
Vera Alvarez   at.arcu.Vestibulum@pellentesque.edu
Yetta Anthony  enim.sit@risus.org
Violet Arnold  ad.litora@at.com
August Ashley  consectetuer.euismod@Phasellus.com
Karyn Austin   ipsum.primis.in@Maurisblanditenim.org
Lucas Ayers    at@elitpretiumet.com
```

NULLIF 函数

语法

NULLIF 表达式比较两个参数并在两个参数相等时返回 null。如果两个参数不相等，则返回第一个参数。此表达式为 NVL 或 COALESCE 表达式的反向表达式。

```
NULLIF ( expression1, expression2 )
```

参数

expression1，expression2

所比较的目标列或表达式。返回类型与第一个表达式的类型相同。NULLIF 结果的默认列名称为第一个表达式的列名称。

示例

在以下示例中，查询返回字符串 first，因为参数不相等。

```
SELECT NULLIF('first', 'second');

case
-------
first
```

在以下示例中，查询返回字符串 NULL，因为字符串文本参数相等。

```
SELECT NULLIF('first', 'first');

case
-------
NULL
```

在以下示例中，查询返回 1，因为整数参数不相等。

```
SELECT NULLIF(1, 2);

case
-------
1
```

在以下示例中，查询返回 NULL，因为整数参数相等。

```
SELECT NULLIF(1, 1);

case
-------
NULL
```

在以下示例中，查询在 LISTID 和 SALESID 值匹配时返回 null：

```
select nullif(listid,salesid), salesid
from sales where salesid<10 order by 1, 2 desc;

listid  | salesid
--------+---------
     4  |       2
     5  |       4
     5  |       3
     6  |       5
     10 |       9
     10 |       8
     10 |       7
     10 |       6
        |       1
(9 rows)
```

您可以使用 NULLIF 来确保始终将空字符串返回为 null。在以下示例中，NULLIF 表达式返回一个 null 值或一个包含至少一个字符的字符串。

```
insert into category
values(0,'','Special','Special');

select nullif(catgroup,'') from category
where catdesc='Special';

catgroup
----------
null
(1 row)
```

NULLIF 忽略尾随空格。如果一个字符串不为空但包含空格，则 NULLIF 仍返回 null：

```
create table nulliftest(c1 char(2), c2 char(2));

insert into nulliftest values ('a','a ');

insert into nulliftest values ('b','b');


select nullif(c1,c2) from nulliftest;
```

```
c1
------
null
null
(2 rows)
```

数据类型格式设置函数

主题

- CAST 函数
- CONVERT 函数
- TO_CHAR
- TO_DATE 函数
- TO_NUMBER
- TEXT_TO_INT_ALT
- TEXT_TO_NUMERIC_ALT
- 日期时间格式字符串
- 数字格式字符串
- 数字数据的 Teradata 类格式字符

数据类型格式设置函数提供了将值从一种数据类型转换为另一种数据类型的简单方式。对于这些函数，第一个参数始终是要进行格式设置的参数，第二个参数包含新格式的模板。Amazon Redshift 支持若干数据类型格式设置函数。

CAST 函数

CAST 函数将一种数据类型转换为另一种兼容的数据类型。例如，您可以将字符串转换为日期，或将数值类型转换为字符串。CAST 执行运行时转换，这意味着转换不会更改源表中值的数据类型。仅在查询上下文中对其进行更改。

CAST 函数与 the section called “CONVERT” 非常相似，它们都将一种数据类型转换为另一种数据类型，但它们的调用方式不同。

某些数据类型需要使用 CAST 或 CONVERT 函数显式转换为其他数据类型。其他数据类型可进行隐式转换（作为另一个命令的一部分），无需使用 CAST 或 CONVERT。请参阅 类型兼容性和转换。

语法

使用以下两个等效的语法形式，将表达式从一种数据类型强制转换为另一种数据类型。

```
CAST ( expression AS type )
expression :: type
```

参数

expression

计算结果为一个或多个值的表达式，如列名称或文本。转换 null 值将返回 null。表达式不能包含空白或空字符串。

type

支持的 数据类型 之一。

返回类型

CAST 返回 type 参数指定的数据类型。

> **Note**
>
> 如果您尝试执行有问题的转换（例如，会导致精度损失的 DECIMAL 转换），Amazon Redshift 将返回错误。有问题的转换示例如下：
>
> ```
> select 123.456::decimal(2,1);
> ```
>
> 或导致溢出的 INTEGER 转换：
>
> ```
> select 12345678::smallint;
> ```

示例

某些示例使用示例 TICKIT 数据库。有关设置示例数据的更多信息，请参阅加载数据。

以下两个查询是等效的。它们都将小数值转换为整数：

```
select cast(pricepaid as integer)
from sales where salesid=100;

pricepaid
-----------
162
(1 row)
```

```
select pricepaid::integer
from sales where salesid=100;

pricepaid
-----------
162
(1 row)
```

以下内容会产生类似的结果。它不需要示例数据即可运行：

```
select cast(162.00 as integer) as pricepaid;

pricepaid
-----------
162
(1 row)
```

在此示例中，时间戳列中的值将强制转换为日期，这会导致从每个结果中删除时间：

```
select cast(saletime as date), salesid
from sales order by salesid limit 10;

 saletime  | salesid
-----------+---------
2008-02-18 |       1
2008-06-06 |       2
2008-06-06 |       3
2008-06-09 |       4
2008-08-31 |       5
2008-07-16 |       6
2008-06-26 |       7
2008-07-10 |       8
2008-07-22 |       9
```

```
2008-08-06 |      10
(10 rows)
```

如果您没有像前一个示例中所示的那样使用 CAST，则结果将包括时间：2008-02-18 02:36:48。

以下查询将可变字符数据强制转换为日期。它不需要示例数据即可运行。

```
select cast('2008-02-18 02:36:48' as date) as mysaletime;

mysaletime
--------------------
2008-02-18
(1 row)
```

在此示例中，日期列中的值将强制转换为时间戳：

```
select cast(caldate as timestamp), dateid
from date order by dateid limit 10;

      caldate       | dateid
--------------------+--------
2008-01-01 00:00:00 |   1827
2008-01-02 00:00:00 |   1828
2008-01-03 00:00:00 |   1829
2008-01-04 00:00:00 |   1830
2008-01-05 00:00:00 |   1831
2008-01-06 00:00:00 |   1832
2008-01-07 00:00:00 |   1833
2008-01-08 00:00:00 |   1834
2008-01-09 00:00:00 |   1835
2008-01-10 00:00:00 |   1836
(10 rows)
```

在与前面的示例类似的情况下，您可以使用 TO_CHAR 进一步控制输出格式。

在此示例中，整数将强制转换为字符串：

```
select cast(2008 as char(4));

bpchar
--------
```

```
2008
```

在此示例中，DECIMAL(6,3) 值将强制转换为 DECIMAL(4,1) 值：

```
select cast(109.652 as decimal(4,1));

numeric
---------
109.7
```

此示例显示了一个更复杂的表达式。SALES 表中的 PRICEPAID 列（DECIMAL(8,2) 列）将转换为 DECIMAL(38,2) 列，并且值将乘以 100000000000000000000：

```
select salesid, pricepaid::decimal(38,2)*100000000000000000000
as value from sales where salesid<10 order by salesid;


 salesid |           value
---------+----------------------------
       1 | 728000000000000000000000.00
       2 |  760000000000000000000000.00
       3 | 350000000000000000000000.00
       4 | 175000000000000000000000.00
       5 | 154000000000000000000000.00
       6 | 394000000000000000000000.00
       7 | 788000000000000000000000.00
       8 | 197000000000000000000000.00
       9 | 591000000000000000000000.00
(9 rows)
```

> **Note**
>
> 无法对 GEOMETRY 数据类型执行 CAST 或 CONVERT 操作来将其更改为其他数据类型。不过，您可以提供扩展的已知二进制 (EWKB) 格式的字符串文字的十六进制表示形式，作为接受 GEOMETRY 参数的函数的输入。例如，下面的 ST_AsText 函数需要 GEOMETRY 数据类型。
>
> ```
> SELECT ST_AsText('010100000000000000000001C400000000000002040');
> ```
>
> ```
> st_astext
> ------------
> ```

```
 POINT(7 8)
```

您也可以明确指定 GEOMETRY 数据类型。

```
SELECT ST_AsText('010100000000000000000014400000000000001840'::geometry);
```

```
st_astext
------------
 POINT(5 6)
```

CONVERT 函数

与 CAST 函数相似，CONVERT 函数将一种数据类型转换为另一种兼容的数据类型。例如，您可以将字符串转换为日期，或将数值类型转换为字符串。CONVERT 执行运行时转换，这意味着转换不会更改源表中值的数据类型。仅在查询上下文中对其进行更改。

某些数据类型需要使用 CONVERT 函数显式转换为其他数据类型。其他数据类型可进行隐式转换（作为另一个命令的一部分），无需使用 CAST 或 CONVERT。请参阅 类型兼容性和转换。

语法

```
CONVERT ( type, expression )
```

参数

type

支持的 数据类型 之一。

expression

计算结果为一个或多个值的表达式，如列名称或文本。转换 null 值将返回 null。表达式不能包含空白或空字符串。

返回类型

CONVERT 返回 type 参数指定的数据类型。

> **Note**
>
> 如果您尝试执行有问题的转换（例如，会导致精度损失的 DECIMAL 转换），Amazon Redshift 将返回错误。有问题的转换示例如下：
>
> ```
> SELECT CONVERT(decimal(2,1), 123.456);
> ```
>
> 或导致溢出的 INTEGER 转换：
>
> ```
> SELECT CONVERT(smallint, 12345678);
> ```

示例

某些示例使用示例 TICKIT 数据库。有关设置示例数据的更多信息，请参阅加载数据。

以下查询使用 CONVERT 函数将一些小数列转换为整数

```
SELECT CONVERT(integer, pricepaid)
FROM sales WHERE salesid=100;
```

此示例将整数转换为字符串。

```
SELECT CONVERT(char(4), 2008);
```

在此示例中，当前日期和时间转换为可变字符数据类型：

```
SELECT CONVERT(VARCHAR(30), GETDATE());

getdate
---------
2023-02-02 04:31:16
```

此示例将 saletime 列仅转换只包括时间，删除每行中的日期。

```
SELECT CONVERT(time, saletime), salesid
FROM sales order by salesid limit 10;
```

有关将时间戳从一个时区转换为另一个时区的信息，请参阅 CONVERT_TIMEZONE 函数。有关其他日期和时间函数，请参阅日期和时间函数。

以下示例将可变字符数据转换为日期时间对象。

```
SELECT CONVERT(datetime, '2008-02-18 02:36:48') as mysaletime;
```

> **Note**
>
> 无法对 GEOMETRY 数据类型执行 CAST 或 CONVERT 操作来将其更改为其他数据类型。不过，您可以提供扩展的已知二进制 (EWKB) 格式的字符串文字的十六进制表示形式，作为接受 GEOMETRY 参数的函数的输入。例如，下面的 ST_AsText 函数需要 GEOMETRY 数据类型。
>
> ```
> SELECT ST_AsText('010100000000000000000001C40000000000000002040');
> ```
>
> ```
> st_astext
> ------------
>  POINT(7 8)
> ```
>
> 您也可以明确指定 GEOMETRY 数据类型。
>
> ```
> SELECT ST_AsText('010100000000000000000001440000000000000001840'::geometry);
> ```
>
> ```
> st_astext
> ------------
>  POINT(5 6)
> ```

TO_CHAR

TO_CHAR 将时间戳或数值表达式转换为字符串数据格式。

语法

```
TO_CHAR (timestamp_expression | numeric_expression , 'format')
```

参数

timestamp_expression

一个表达式，用于生成 TIMESTAMP 或 TIMESTAMPTZ 类型值或可隐式强制转换为时间戳的值。

numeric_expression

一个表达式，用于生成数字数据类型值或可隐式强制转换为数字类型的值。有关更多信息，请参阅数字类型。TO_CHAR 在数字串左侧插入空格。

> **Note**
>
> TO_CHAR 不支持 128 位 DECIMAL 值。

format

新值的格式。有关有效格式，请参阅日期时间格式字符串和数字格式字符串。

返回类型

VARCHAR

示例

以下示例将时间戳转换为一个具有日期和时间的值，格式为月份名称填充为九个字符、星期几的名称和当月的日期编号。

```
select to_char(timestamp '2009-12-31 23:15:59', 'MONTH-DY-DD-YYYY HH12:MIPM');

to_char
-------------------------
DECEMBER -THU-31-2009 11:15PM
```

以下示例将时间戳转换为具有这一年中日期编号的值。

```
select to_char(timestamp '2009-12-31 23:15:59', 'DDD');

to_char
-------------------------
365
```

以下示例将时间戳转换为这一周的 ISO 日期编号。

```
select to_char(timestamp '2022-05-16 23:15:59', 'ID');

to_char
-------------------------
1
```

以下示例从日期中提取月份名称。

```
select to_char(date '2009-12-31', 'MONTH');

to_char
-------------------------
DECEMBER
```

以下示例将 EVENT 表中的每个 STARTTIME 值转换为由小时、分钟和秒组成的字符串。

```
select to_char(starttime, 'HH12:MI:SS')
from event where eventid between 1 and 5
order by eventid;

to_char
----------
02:30:00
08:00:00
02:30:00
02:30:00
07:00:00
```

以下示例将整个时间戳值转换为不同的格式。

```
select starttime, to_char(starttime, 'MON-DD-YYYY HH12:MIPM')
from event where eventid=1;

      starttime      |       to_char
---------------------+---------------------
 2008-01-25 14:30:00 | JAN-25-2008 02:30PM
```

以下示例将时间戳文本转换为字符串。

```
select to_char(timestamp '2009-12-31 23:15:59','HH24:MI:SS');
```

```
to_char
----------
23:15:59
```

以下示例将十进制数转换为字符串。

```
select to_char(125.8, '999.99');

to_char
---------
125.80
```

以下示例将十进制数转换为字符串。

```
select to_char(125.8, '999D99');

to_char
---------
125.80
```

以下示例将数字转换为具有一个前导零的字符串。

```
select to_char(125.8, '0999D99');

to_char
---------
0125.80
```

以下示例将一个数字转换为末尾带负号的字符串。

```
select to_char(-125.8, '999D99S');

to_char
---------
125.80-
```

以下示例将数字转换为在指定位置带有正号或负号的字符串。

```
select to_char(125.8, '999D99SG');
```

```
to_char
---------
125.80+
```

以下示例将数字转换为在指定位置带有正号的字符串。

```
select to_char(125.8, 'PL999D99');

to_char
---------
+ 125.80
```

以下示例将一个数字转换为带货币符号的字符串。

```
select to_char(-125.88, '$S999D99');

to_char
---------
$-125.88
```

以下示例将数字转换为在指定位置带货币符号的字符串。

```
select to_char(-125.88, 'S999D99L');

to_char
---------
-125.88$
```

以下示例使用千位（逗号）分隔符将数字转换为字符串。

```
select to_char(1125.8, '9,999.99');

to_char
---------
1,125.80
```

以下示例将一个数字转换为用尖括号将负数括起来的字符串。

```
select to_char(-125.88, '$999D99PR');

to_char
```

```
---------
$<125.88>
```

以下示例将一个数字转换为罗马数字字符串。

```
select to_char(125, 'RN');

to_char
---------
   CXXV
```

以下示例将日期转换为世纪代码。

```
select to_char(date '2020-12-31', 'CC');

to_char
---------
21
```

以下示例显示一周中的某天。

```
SELECT to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS');

to_char
-----------------------
Wednesday, 31 09:34:26
```

以下示例显示数字的序数后缀。

```
SELECT to_char(482, '999th');

to_char
-----------------------
 482nd
```

以下示例将销售表中支付的价格减去佣金。差随后将向上舍入并转换为罗马数字，如 to_char 列中所示：

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'rn') from sales
group by sales.pricepaid, sales.commission, salesid
```

```
order by salesid limit 10;

 salesid | pricepaid | commission | difference |     to_char
---------+-----------+------------+------------+-----------------
       1 |    728.00 |     109.20 |     618.80 |           dcxix
       2 |     76.00 |      11.40 |      64.60 |             lxv
       3 |    350.00 |      52.50 |     297.50 |        ccxcviii
       4 |    175.00 |      26.25 |     148.75 |           cxlix
       5 |    154.00 |      23.10 |     130.90 |           cxxxi
       6 |    394.00 |      59.10 |     334.90 |         cccxxxv
       7 |    788.00 |     118.20 |     669.80 |           dclxx
       8 |    197.00 |      29.55 |     167.45 |          clxvii
       9 |    591.00 |      88.65 |     502.35 |             dii
      10 |     65.00 |       9.75 |      55.25 |              lv
```

以下示例向 to_char 列中显示的差值添加货币符号：

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'l99999D99') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;

salesid | pricepaid | commission | difference |  to_char
--------+-----------+------------+------------+------------
      1 |    728.00 |     109.20 |     618.80 | $   618.80
      2 |     76.00 |      11.40 |      64.60 | $    64.60
      3 |    350.00 |      52.50 |     297.50 | $   297.50
      4 |    175.00 |      26.25 |     148.75 | $   148.75
      5 |    154.00 |      23.10 |     130.90 | $   130.90
      6 |    394.00 |      59.10 |     334.90 | $   334.90
      7 |    788.00 |     118.20 |     669.80 | $   669.80
      8 |    197.00 |      29.55 |     167.45 | $   167.45
      9 |    591.00 |      88.65 |     502.35 | $   502.35
     10 |     65.00 |       9.75 |      55.25 | $    55.25
```

以下示例列出了完成每次销售的世纪。

```
select salesid, saletime, to_char(saletime, 'cc') from sales
order by salesid limit 10;

 salesid |      saletime       | to_char
---------+---------------------+---------
       1 | 2008-02-18 02:36:48 | 21
```

```
       2 | 2008-06-06 05:00:16 | 21
       3 | 2008-06-06 08:26:17 | 21
       4 | 2008-06-09 08:38:52 | 21
       5 | 2008-08-31 09:17:02 | 21
       6 | 2008-07-16 11:59:24 | 21
       7 | 2008-06-26 12:56:06 | 21
       8 | 2008-07-10 02:12:36 | 21
       9 | 2008-07-22 02:23:17 | 21
      10 | 2008-08-06 02:51:55 | 21
```

以下示例将 EVENT 表中的每个 STARTTIME 值转换为由小时、分钟、秒和时区组成的字符串：

```
select to_char(starttime, 'HH12:MI:SS TZ')
from event where eventid between 1 and 5
order by eventid;

to_char
----------
02:30:00 UTC
08:00:00 UTC
02:30:00 UTC
02:30:00 UTC
07:00:00 UTC
```

以下示例显示了秒、毫秒和微秒的格式设置。

```
select sysdate,
to_char(sysdate, 'HH24:MI:SS') as seconds,
to_char(sysdate, 'HH24:MI:SS.MS') as milliseconds,
to_char(sysdate, 'HH24:MI:SS:US') as microseconds;

timestamp           | seconds  | milliseconds | microseconds
--------------------+----------+--------------+----------------
2015-04-10 18:45:09 | 18:45:09 | 18:45:09.325 | 18:45:09:325143
```

TO_DATE 函数

TO_DATE 会将以字符串形式表示的日期转换为 DATE 数据类型。

语法

```
TO_DATE(string, format)
```

```
TO_DATE(string, format, is_strict)
```

参数

string

要转换的字符串。

format

一个字符串文本，用于定义其日期部分中的输入字符串格式。有关有效日期、月份和年份格式的列表，请参阅日期时间格式字符串。

is_strict

一个可选的布尔值，它指定在输入日期值超出范围时是否返回错误。当 is_strict 被设置为 TRUE 时，如果存在超出范围的值，则返回错误。当 is_strict 被设置为 FALSE（默认值）时，则接受溢出值。

返回类型

TO_DATE 将根据 format 值返回 DATE。

如果转换为格式 失败，则返回错误。

示例

以下 SQL 语句将日期 02 Oct 2001 转换为日期数据类型。

```
select to_date('02 Oct 2001', 'DD Mon YYYY');

to_date
------------
2001-10-02
(1 row)
```

以下 SQL 语句将字符串 20010631 转换为日期。

```
select to_date('20010631', 'YYYYMMDD', FALSE);
```

结果是 2001 年 7 月 1 日，因为六月份只有 30 天。

```
to_date
------------
2001-07-01
```

以下 SQL 语句将字符串 20010631 转换为日期：

```
to_date('20010631', 'YYYYMMDD', TRUE);
```

结果产生错误，因为 6 月只有 30 天。

```
ERROR:  date/time field date value out of range: 2001-6-31
```

TO_NUMBER

TO_NUMBER 将字符串转换为数字（小数）值。

语法

```
to_number(string, format)
```

参数

string

要转换的字符串。格式必须是文本值。

format

第二个参数是指示应如何分析字符串以创建数字值的格式字符串。例如，格式 '99D999' 指定要转换的字符串包含五位数，其中小数点在第三位。例如，to_number('12.345','99D999') 将 12.345 作为数字值返回。有关有效格式的列表，请参阅 数字格式字符串。

返回类型

TO_NUMBER 返回 DECIMAL 数。

如果转换为格式 失败，则返回错误。

示例

以下示例将字符串 12,454.8- 转换为数字：

```
select to_number('12,454.8-', '99G999D9S');

to_number
-----------
-12454.8
```

以下示例将字符串 $ 12,454.88 转换为数字：

```
select to_number('$ 12,454.88', 'L 99G999D99');

to_number
-----------
12454.88
```

以下示例将字符串 $ 2,012,454.88 转换为数字：

```
select to_number('$ 2,012,454.88', 'L 9,999,999.99');

to_number
-----------
2012454.88
```

TEXT_TO_INT_ALT

TEXT_TO_INT_ALT 使用 Teradata 类格式将字符串转换为整数。结果中的小数位数将被截断。

语法

```
TEXT_TO_INT_ALT (expression [ , 'format'])
```

参数

expression

结果产生一个或多个 CHAR 或 VARCHAR 值的表达式，如列名称或文本字符串。转换 null 值将返回 null。该函数将空白或空字符串转换为 0。

format

一个字符串文本，用于定义输入表达式的格式。有关您可以指定的格式字符的更多信息，请参阅数字数据的 Teradata 类格式字符。

返回类型

TEXT_TO_INT_ALT 返回一个 INTEGER 值。

转换结果的小数部分将被截断。

如果转换为您指定的 format 短语不成功，Amazon Redshift 将返回一个错误。

示例

以下示例将输入 expression 字符串“123-”转换为整数 -123。

```
select text_to_int_alt('123-');
```

```
text_to_int_alt
----------
      -123
```

以下示例将输入 expression 字符串“2147483647+”转换为整数 2147483647。

```
select text_to_int_alt('2147483647+');
```

```
text_to_int_alt
----------
2147483647
```

以下示例将指数输入 expression 字符串“-123E-2”转换为整数 -1。

```
select text_to_int_alt('-123E-2');
```

```
text_to_int_alt
----------
        -1
```

以下示例将输入 expression 字符串“2147483647+”转换为整数 2147483647。

```
select text_to_int_alt('2147483647+');
```

```
text_to_int_alt
```

```
----------
2147483647
```

以下示例使用 format 短语“999S”将输入 expression 字符串“123{”转换为整数 1230。S 字符表示带符号的分区十进制。有关更多信息，请参阅 数字数据的 Teradata 类格式字符。

```
text_to_int_alt('123{', '999S');
```

```
text_to_int_alt
----------
      1230
```

以下示例使用 format 短语“C9(I)”将输入 expression 字符串“USD123”转换为整数 123。请参阅 数字数据的 Teradata 类格式字符。

```
text_to_int_alt('USD123', 'C9(I)');
```

```
text_to_int_alt
----------
       123
```

以下示例指定一个表列作为输入 expression。

```
select text_to_int_alt(a), text_to_int_alt(b) from t_text2int order by 1;
```

```
 text_to_int_alt | text_to_int_alt
-----------------+-----------------
            -123 |            -123
            -123 |            -123
             123 |             123
             123 |             123
```

以下是此示例的表定义和 insert 语句。

```
create table t_text2int (a varchar(200), b char(200));
```

```
insert into t_text2int VALUES('123', '123'),('123.123', '123.123'), ('-123', '-123'),
 ('123-', '123-');
```

TEXT_TO_NUMERIC_ALT

TEXT_TO_NUMERIC_ALT 执行一种 Teradata 类的转换操作，以将字符串转换为数字数据格式。

语法

```
TEXT_TO_NUMERIC_ALT (expression [, 'format'] [, precision, scale])
```

参数

expression

计算结果为一个或多个 CHAR 或 VARCHAR 值的表达式，如列名称或文本。转换 null 值将返回 null。空白或空字符串将转换为 0。

format

一个字符串文本，用于定义输入表达式的格式。有关更多信息，请参阅 数字数据的 Teradata 类格式字符。

精度

数字结果中的位数。默认值为 38。

小数位数

数字结果中小数点右侧的位数。默认值为 0。

返回类型

TEXT_TO_NUMERIC_ALT 返回一个 DECIMAL 数。

如果转换为您指定的 format 短语不成功，Amazon Redshift 将返回一个错误。

Amazon Redshift 使用您在精度选项中为该类型指定的最高精度将输入 expression 字符串转换为数字类型。如果数值的长度超过您为精度指定的值，Amazon Redshift 将根据以下规则对数值进行四舍五入：

- 如果转换结果的长度超出了您在 format 短语中指定的长度，Amazon Redshift 会返回一个错误。
- 如果将结果转换为数值，则结果将四舍五入到最接近的值。如果小数部分正好位于上下转换结果的中间位置，则结果将四舍五入到最接近的偶数值。

示例

以下示例将输入 expression 字符串“1.5”转换为数值“2”。因为语句没有指定小数位数，小数位数默认值为 0，并且转换结果不包含分数结果。因为 .5 介于 1 和 2 之间的中间，所以转换结果四舍五入为偶数值 2。

```
select text_to_numeric_alt('1.5');
```

```
 text_to_numeric_alt
---------------------
                   2
```

以下示例将输入 expression 字符串“2.51”转换为数值 3。因为语句没有指定小数位数，小数位数默认值为 0，并且转换结果不包含分数结果。因为 .51 接近 3 而不是 2，所以转换结果四舍五入到值 3。

```
select text_to_numeric_alt('2.51');
```

```
 text_to_numeric_alt
---------------------
                   3
```

以下示例以 10 为精度，2 为小数位数，将输入 expression 字符串 123.52501 转换为数值 123.53。

```
select text_to_numeric_alt('123.52501', 10, 2);
```

```
 text_to_numeric_alt
---------------------
              123.53
```

以下示例指定一个表列作为输入 expression。

```
select text_to_numeric_alt(a), text_to_numeric_alt(b) from t_text2numeric order by 1;
```

```
          text_to_numeric_alt           |           text_to_numeric_alt
----------------------------------------+----------------------------------------
 -99999999999999999999999999999999999999 | -99999999999999999999999999999999999999
                                  -12300 |                                  -12300
                                     123 |                                     123
                                     123 |                                     123
```

```
 999999999999999999999999999999999999 |  999999999999999999999999999999999999
```

以下是此示例的表定义和 insert 语句。

```
create table  t_text2numeric (a varchar(200), b char(200));
```

```
insert into  t_text2numeric values
('123', '123'),
('+123.456', '+123.456'),
('-' || repeat('9', 38), '-' || repeat('9', 38)),
(repeat('9', 38) || '+', repeat('9', 38) || '+'),
('-123E2', '-123E2');
```

日期时间格式字符串

您可以找到日期时间格式字符串遵循的引用。

以下格式字符串适用于 TO_CHAR 之类的函数。这些字符串可包含日期时间分隔符（如 '-'、'/' 或 ':'）以及下面的日期部分和时间部分。

日期部分或时间部分	意义
BC 或 B.C.、AD 或 A.D.、b.c. 或 bc、ad 或 a.d。	大写和小写的纪元指示符
CC	2 位世纪数字
YYYY、YYY、YY、Y	4 位、3 位、2 位、1 位年数字
Y,YYY	带逗号的 4 位年数
IYYY、IYY、IY、I	4 位、3 位、2 位、1 位国际标准化组织 (ISO) 年数
Q	季度数（1 至 4）
MONTH、Month、month	月名称（大写、大小写混合、小写，空格填补为 9 个字符）

日期部分或时间部分	意义
MON、Mon、mon	缩写的月份名称（大写、大小写混合、小写，空格填补至 3 个字符）
MM	月数 (01-12)
RM、rm	使用罗马数字的月数（I–XII，I 代表 1 月，大小写均可）
W	一个月中的周（1–5，第一周从当月的第一天开始。）
WW	一年的周数（1–53，第一周从一年的第一天开始。）
IW	一年的 ISO 周数（新的一年的第一个星期四算在第 1 周。）
DAY、Day、day	日名称（大写、大小写混合、小写，空格填补为 9 个字符）
DY、Dy、dy	缩写的日期名称（大写、大小写混合、小写，空格填补为 3 个字符）
DDD	一年中的日 (001–366)
IDDD	ISO 8601 按周编号的年中的日期（001-371；每年的第一天是 ISO 第一周的周一）
DD	用数字表示的一个月中的日 (01–31)

日期部分或时间部分	意义
D	一周中的日（1–7；星期日为 1） **Note** D 日期部分的运行方式与用于日期时间功能的 DATE_PART 和 EXTRACT 的一周中的日 (DOW) 日期部分不同。DOW 基于 0–6 的整数，其中星期日是 0。有关更多信息，请参阅 日期或时间戳函数的日期部分。
ID	ISO 8601 星期几，周一 (1) 至周日 (7)
J	儒略日（公元前 4712 年 1 月 1 日起的日期）
HH24	小时（24 小时制，00–23）
HH 或 HH12	小时（12 小时制，01–12）
MI	分钟数 (00—59)
SS	秒数 (00—59)
MS	毫秒 (.000)
US	微秒 (.000000)
AM 或 PM；A.M. 或 P.M.；a.m. 或 p.m.；am 或 pm	大写和小写的子午线指示符（适用于 12 小时制）
TZ、tz	大小写时区缩写；仅对 TIMESTAMPTZ 有效。
OF	从 UTC 的偏移；仅对 TIMESTAMPTZ 有效。

> **Note**
>
> 您必须用单引号将 datetime 分隔符（如 '-'、'/' 或 ':'）括起，但您必须用双引号将上表中列出的 "dateparts" 和 "timeparts" 括起。

示例

有关将日期格式化为字符串的示例，请参阅TO_CHAR。

数字格式字符串

在下面，您可以找到数字格式字符串的引用。

以下格式字符串适用于 TO_NUMBER 和 TO_CHAR 之类的函数。

- 有关将字符串格式化为数字的示例，请参阅TO_NUMBER。
- 有关将数字格式化为字符串的示例，请参阅TO_CHAR。

格式	描述
9	具有指定位数的数字值。
0	包含前导零的数字值。
.（句点），D	小数点。
,（逗号）	千位分隔符。
CC	世纪代码。例如，21 世纪从 2001-01-01 开始（仅受 TO_CHAR 支持）。
FM	填充模式。隐藏填补空格和零。
PR	尖括号中的负值。
S	锚定数字的符号。
L	指定位置中的货币符号。

格式	描述
G	组分隔符。
MI	小于 0 的数字的指定位置中的减号。
PL	大于 0 的数字的指定位置中的加号。
SG	指定位置中的加号或减号。
RN	1 到 3999 之间的罗马数字（仅受 TO_CHAR 支持）。
TH 或 th	序号后缀。不会转换小于零的分数或值。

数字数据的 Teradata 类格式字符

下面，您可以找到 TEXT_TO_INT_ALT 和 TEXT_TO_NUMERIC_ALT 函数如何解释输入 expression 字符串中的字符。您还可以找到您可以在 format 短语中指定的字符列表。此外，您还可以找到 Teradata 类格式和 Amazon Redshift 之间的格式选项差异的说明。

格式	描述
G	不支持在输入 expression 字符串中作为组分隔符。您不能在 format 短语中指定此字符。
D	基数符号。您可以在 format 短语中指定此字符。此字符等效于. (句点)。 基数符号不能出现在包含以下任意字符的 format 短语中： • . （句点） • S（大写“s”） • V（大写“v”）
/ , : %	插入字符/（正斜杠）、逗号 (,)、:（冒号）和 %（百分号）。

格式	描述
	您不能在 format 短语中包含这些字符。 Amazon Redshift 会忽略输入 expression 字符串中的这些字符。
.	表示基数字符的句点，即小数点。 此字符不能出现在包含以下任意字符的 format 短语中： • D（大写“d”） • S（大写“s”） • V（大写“v”）
B	不能将空格字符 (B) 包含在 format 短语中。在输入 expression 字符串中，忽略前导空格和尾随空格，并且不允许数字之间存在空格。
+ -	您不能在 format 短语中包含加号 (+) 或减号 (-)。但是，如果加号 (+) 和减号 (-) 出现在输入 expression 字符串，则将它们隐式解析为数值的一部分。
V	小数点位置指示器。 此字符不能出现在包含以下任意字符的 format 短语中： • D（大写“d”） • .（句点）
Z	抑零十进制数字。Amazon Redshift 修剪前导零。Z 字符不能跟随字符 9。如果分数部分包含字符 9，则 Z 字符必须位于基数字符的左侧。
9	十进制数字。

格式	描述
CHAR(n)	对于此格式，您可以指定以下各项： • CHAR 由 Z 或 9 字符组成。Amazon Redshift 不支持 CHAR 值中的 + (加号) 或 - (减号)。 • n 是整数常量 I 或 F。对于 I，它是显示数字或整数数据的整数部分所需的字符数。对于 F，它是显示数字数据的小数部分所需的字符数。
-	连字符 (-) 字符。 您不能在 format 短语中包含此字符。 Amazon Redshift 会忽略输入 expression 字符串中的此字符。
S	带符号的分区时进度。S 字符必须跟随 format 短语中的最后一位十进制数字。输入 expression 字符串的最后一个字符和相应的数字转换列在带符号的分区十进制、Teradata 类数字数据格式的数据格式字符 中。 S 字符不能出现在包含以下任意字符的 format 短语中： • + (加号) • . (句点) • D (大写“d”) • Z (大写“z”) • F (大写“f”) • E (大写“e”)
E	指数表示法。输入 expression 字符串可以包含指数字符。您不能在 format 短语中将 E 指定为指数字符。

格式	描述
FN9	在 Amazon Redshift 中不受支持。
FNE	在 Amazon Redshift 中不受支持。
$、USD、US Dollars	美元符号 ($)、ISO 货币符号 (USD) 和货币名称 US Dollars。 ISO 货币符号 USD 和货币名称 US Dollars 区分大小写。Amazon Redshift 仅支持 USD 货币。输入 expression 字符串可以在 USD 货币符号和数值之间包含空格，例如“$ 123E2”或“123E2 $”。
L	货币符号。此货币符号字符只能在 format 短语中出现一次。您不能指定重复的货币符号字符。
C	ISO 货币符号。此货币符号字符只能在 format 短语中出现一次。您不能指定重复的货币符号字符。
否	完整货币名称。此货币符号字符只能在 format 短语中出现一次。您不能指定重复的货币符号字符。
O	双币符号。您不能在 format 短语中指定此字符。
U	双 ISO 货币符号。您不能在 format 短语中指定此字符。
A	完整的双币名称。您不能在 format 短语中指定此字符。

带符号的分区十进制、Teradata 类数字数据格式的数据格式字符

您可以在 TEXT_TO_INT_ALT 和 TEXT_TO_NUMERIC_ALT 函数的 format 短语中将以下字符用于带符号的分区十进制值。

输入字符串的最后一个字符	数字转换
{ 或 0	n … 0
A 或 1	n … 1
B 或 2	n … 2
C 或 3	n … 3
D 或 4	n … 4
E 或 5	n … 5
F 或 6	n … 6
G 或 7	n … 7
H 或 8	n … 8
I 或 9	n … 9
}	-n … 0
J	-n … 1
K	-n … 2
L	-n … 3
M	-n … 4
否	-n … 5
O	-n … 6
P	-n … 7
Q	-n … 8
R	-n … 9

日期和时间函数

在本部分中，您可以找到 Amazon Redshift 支持的日期和时间标量函数的相关信息。

主题

- TIMESTAMPTZ_CMP 函数
- TIMESTAMPTZ_CMP_DATE 函数
- TIMESTAMPTZ_CMP_TIMESTAMP 函数
- TIMEZONE 函数
- TO_TIMESTAMP 函数
- TRUNC 函数
- 日期或时间戳函数的日期部分

日期和时间函数摘要

函数	语法	返回值
+ (串联) 运算符 将日期连接到 + 符号任一侧的时间，并返回 TIMESTAMP 或 TIMESTAMPTZ。	date + time	TIMESTAMP 或者 TIMESTAMPZ
ADD_MONTHS 将指定的月数添加到日期或时间戳。	ADD_MONTHS ({date|timestamp}, integer)	TIMESTAMP
AT TIME ZONE 指定要与 TIMESTAMP 或 TIMESTAMPTZ 表达式一起使用的时区。	AT TIME ZONE 'timezone'	TIMESTAMP 或者 TIMESTAMPZ
CONVERT_TIMEZONE 将一个时区的时间戳转换为另一个时区的时间戳。	CONVERT_TIMEZONE (['timezone',] 'timezone', timestamp)	TIMESTAMP
CURRENT_DATE 返回当前事务开始时的当前会话时区 (预设情况下为 UTC) 中的日期。	CURRENT_DATE	DATE
DATE_CMP	DATE_CMP (date1, date2)	INTEGER

函数	语法	返回值
比较两个日期并在日期相同时返回 0，date1 较大时返回 1，date2 较大时返回 -1。		
DATE_CMP_TIMESTAMP 将日期与时间进行比较，并在值相同时返回 0，日期较大时返回 1，时间戳较大时返回 -1。	DATE_CMP_TIMESTAMP (date, timestamp)	INTEGER
DATE_CMP_TIMESTAMPTZ 将日期与带有时区的时间戳进行比较，并在值相同时返回 0，日期较大时返回 1，timestamptz 较大时返回 -1。	DATE_CMP_TIMESTAMPTZ (date, timestamptz)	INTEGER
DATE_PART_YEAR 从日期中提取年份。	DATE_PART_YEAR (date)	INTEGER
DATEADD 按指定的时间间隔递增日期或时间。	DATEADD (datepart, interval, {date|time|timetz|timestamp})	TIMESTAMP 、TIME 或 TIMETZ
DATEDIFF 返回给定日期部分（如一天或月）的两个日期或时间之间的差值。	DATEDIFF (datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp})	BIGINT
DATE_PART 从日期或时间中提取日期部分值。	DATE_PART (datepart, {date|timestamp})	DOUBLE
DATE_TRUNC 基于日期部分截断时间戳。	DATE_TRUNC ('datepart', timestamp)	TIMESTAMP
EXTRACT 从 timestamp、timestamptz、time 或 timetz 中提取日期或时间部分。	EXTRACT (datepart FROM source)	INTEGER or DOUBLE

函数	语法	返回值
GETDATE 返回当前会话时区（预设情况下为 UTC）中的当前日期和时间。括号为必填项。	GETDATE()	TIMESTAMP
INTERVAL_CMP 比较两个时间间隔并在时间间隔相等时返回 0，interval1 较大时返回 1，interval2 较大时返回 -1。	INTERVAL_CMP (interval1, interval2)	INTEGER
LAST_DAY 返回该月最后一天的日期，该日期包含 date。	LAST_DAY(date)	DATE
MONTHS_BETWEEN 返回两个日期之间相隔的月数。	MONTHS_BETWEEN (date, date)	FLOAT8
NEXT_DAY 返回比 date 晚的日期的第一个实例的日期。	NEXT_DAY (date, day)	DATE
SYSDATE 返回当前事务开始的日期和时间 (UTC)。	SYSDATE	TIMESTAMP
TIMEOFDAY 以字符串值形式返回当前会话时区（预设情况下为 UTC）中的当前工作日、日期和时间。	TIMEOFDAY()	VARCHAR
TIMESTAMP_CMP 将两个时间戳值进行比较，并在值相同时返回 0，在 timestamp1 较大时返回 1，在 timestamp 2 较大时返回 -1。	TIMESTAMP_CMP (timestamp1, timestamp2)	INTEGER

函数	语法	返回值
TIMESTAMP_CMP_DATE 将时间戳与日期进行比较，并在值相同时返回 0，时间戳较大时返回 1，日期较大时返回 -1。	TIMESTAMP_CMP_DATE (timestamp, date)	INTEGER
TIMESTAMP_CMP_TIMESTAMPTZ 将时间戳与带有时区的时间戳进行比较，并在值相同时返回 0，时间戳较大时返回 1，timestamptz 较大时返回 -1。	TIMESTAMP_CMP_TIMESTAMPTZ (timestamp, timestamptz)	INTEGER
TIMESTAMPTZ_CMP 将两个带有时区的时间戳值进行比较，并在值相同时返回 0，timestamptz1 较大时返回 1，timestamptz2 较大时返回 -1。	TIMESTAMPTZ_CMP (timestamptz1, timestamptz2)	INTEGER
TIMESTAMPTZ_CMP_DATE 将带有时区的时间戳值与日期进行比较，并在值相同时返回 0，timestamptz 较大时返回 1，日期较大时返回 -1。	TIMESTAMPTZ_CMP_DATE (timestamptz, date)	INTEGER
TIMESTAMPTZ_CMP_TIMESTAMP 将带有时区的时间戳与时间戳进行比较，并在值相同时返回 0，timestamptz 较大时返回 1，时间戳较大时返回 -1。	TIMESTAMPTZ_CMP_TIMESTAMP (timestamptz, timestamp)	INTEGER
TIMEZONE 返回指定时区和时间戳值的一个时间戳。	TIMEZONE ('timezone' { timestamp | timestamptz)	TIMESTAMP 或者 TIMESTAMPTZ
TO_TIMESTAMP 返回指定时间戳和时区格式的一个带有时区的时间戳。	TO_TIMESTAMP ('timestamp', 'format')	TIMESTAMPTZ

函数	语法	返回值
TRUNC 截断时间戳并返回日期。	TRUNC(timestamp)	DATE

> **Note**
>
> 在经过时间计算中不考虑闰秒。

事务中的日期和时间函数

当您在事务块（BEGIN … END）中运行以下函数时，该函数将返回当前事务的开始日期或时间，而不是当前语句的开始时间。

- SYSDATE
- TIMESTAMP
- CURRENT_DATE

以下函数始终返回当前语句的开始日期或时间，即使它们在事务数据块中也是如此。

- GETDATE
- TIMEOFDAY

弃用的仅领导节点函数

以下日期函数因为仅在领导节点上运行而遭到弃用。有关更多信息，请参阅 仅领导节点函数。

- AGE。请改用 DATEDIFF 函数。
- CURRENT_TIME。改用 GETDATE 函数 或 SYSDATE。
- CURRENT_TIMESTAMP。改用 GETDATE 函数 或 SYSDATE。
- LOCALTIME。改用 GETDATE 函数 或 SYSDATE。
- LOCALTIMESTAMP。改用 GETDATE 函数 或 SYSDATE。
- ISFINITE

- NOW。改用 GETDATE 函数 或 SYSDATE。

+ (串联) 运算符

将 DATE 连接到 + 符号任一侧的 TIME 或 TIMETZ，并返回 TIMESTAMP 或 TIMESTAMPTZ。

语法

```
date + {time | timetz}
```

参数的顺序可以反转。例如，time + date。

参数

date

数据类型为 DATE 的列，或一个隐式计算结果为 DATE 类型的表达式。

time

数据类型为 TIME 的列，或一个隐式计算结果为 TIME 类型的表达式。

timetz

数据类型为 TIMETZ 的列，或一个隐式计算结果为 TIMETZ 类型的表达式。

返回类型

如果输入为 date + time，则为 TIMESTAMP。

如果输入为 date + timetz，则为 TIMESTAMPTZ。

示例

示例设置

要设置示例中使用的 TIME_TEST 和 TIMETZ_TEST 表，请使用以下命令。

```
create table time_test(time_val time);

insert into time_test values
('20:00:00'),
```

```
('00:00:00.5550'),
('00:58:00');

create table timetz_test(timetz_val timetz);

insert into timetz_test values
('04:00:00+00'),
('00:00:00.5550+00'),
('05:58:00+00');
```

具有时间列的示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

以下示例串联一个日期文本和一个 TIME_VAL 列。

```
select date '2000-01-02' + time_val as ts from time_test;

ts
---------------------
2000-01-02 20:00:00
2000-01-02 00:00:00.5550
2000-01-02 00:58:00
```

以下示例串联一个日期文本和一个时间文本。

```
select date '2000-01-01' + time '20:00:00' as ts;

         ts
---------------------
 2000-01-01 20:00:00
```

以下示例串联一个时间文本和一个日期文本。

```
select time '20:00:00' + date '2000-01-01' as ts;

         ts
---------------------
 2000-01-01 20:00:00
```

具有 TIMETZ 列的示例

下面的示例表 TIMETZ_TEST 具有一个列 TIMETZ_VAL（类型 TIMETZ），其中插入了三个值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

以下示例串联一个日期文本和一个 TIMETZ_VAL 列。

```
select date '2000-01-01' + timetz_val as ts from timetz_test;
ts
---------------------
2000-01-01 04:00:00+00
2000-01-01 00:00:00.5550+00
2000-01-01 05:58:00+00
```

以下示例串联一个 TIMETZ_VAL 列和一个日期文本。

```
select timetz_val + date '2000-01-01' as ts from timetz_test;
ts
---------------------
2000-01-01 04:00:00+00
2000-01-01 00:00:00.5550+00
2000-01-01 05:58:00+00
```

以下示例串联一个 DATE 文本和一个 TIMETZ 文本。该示例返回 TIMESTAMPTZ，默认情况下它采用 UTC 时区。UTC 比 PST 早 8 小时，因此结果比输入时间早 8 小时。

```
select date '2000-01-01' + timetz '20:00:00 PST' as ts;
```

```
          ts
------------------------
 2000-01-02 04:00:00+00
```

ADD_MONTHS 函数

ADD_MONTHS 会将指定的月数添加到日期或时间戳值或表达式中。DATEADD 函数提供了类似的功能。

语法

```
ADD_MONTHS( {date | timestamp}, integer)
```

参数

date | timestamp

数据类型为 DATE 或 TIMESTAMP 的列，或一个隐式计算结果为 DATE 或 TIMESTAMP 类型的表达式。如果日期是该月的最后一天，或者如果产生的月份较短，则函数在结果中返回该月的最后一天。对于其他日期，结果包含与日期表达式相同的日期编号。

integer

数据类型 INTEGER 的值。使用负数从日期中减去月份。

返回类型

TIMESTAMP

示例

以下查询使用 TRUNC 函数内的 ADD_MONTHS 函数。TRUNC 函数从 ADD_MONTHS 的结果中删除一天中的时间。ADD_MONTHS 函数会为 CALDATE 列中的每个值添加 12 个月。CALDATE 列中的值是日期。

```
select distinct trunc(add_months(caldate, 12)) as calplus12,
trunc(caldate) as cal
from date
order by 1 asc;
```

```
 calplus12  |    cal
------------+------------
 2009-01-01 | 2008-01-01
 2009-01-02 | 2008-01-02
 2009-01-03 | 2008-01-03
...
(365 rows)
```

以下示例使用 ADD_MONTHS 函数给时间戳 加上 1 个月。

```
select add_months('2008-01-01 05:07:30', 1);

add_months
---------------------
2008-02-01 05:07:30
```

以下示例演示 ADD_MONTHS 函数在具有不同天数的月份的日期上运行时的行为。此示例说明该函数如何处理在 3 月 31 日的基础上再加上 1 个月以及在 4 月 30 日的基础上再加上 1 个月的情况。4 月有 30 天，所以在 3 月 31 日的基础上再加上 1 个月，结果是 4 月 30 日。5 月有 31 天，所以在 4 月 30 日的基础上再加上 1 个月，结果是 5 月 31 日。

```
select add_months('2008-03-31',1);

add_months
---------------------
2008-04-30 00:00:00

select add_months('2008-04-30',1);

add_months
---------------------
2008-05-31 00:00:00
```

AT TIME ZONE 函数

AT TIME ZONE 指定要与 TIMESTAMP 或 TIMESTAMPTZ 表达式一起使用的时区。

语法

```
AT TIME ZONE 'timezone'
```

参数

timezone

返回值的 TIMEZONE。该时区可以指定为时区名称（例如 **'Africa/Kampala'** 或者 **'Singapore'**）或作为时区缩写（例如 **'UTC'** 或者 **'PDT'**）。

要查看支持的时区名称的列表，请执行以下命令。

```
select pg_timezone_names();
```

要查看支持的时区缩写的列表，请执行以下命令。

```
select pg_timezone_abbrevs();
```

有关更多信息以及示例，请参阅 时区使用说明。

返回类型

与 TIMESTAMP 表达式一起使用时的 TIMESTAMPTZ。与 TIMESTAMPTZ 表达式一起使用时的 TIMESTAMP。

示例

以下示例转换不带时区的时间戳值，并将其解释为 MST 时间（POSIX 中的 UTC+7）。该示例返回 UTC 时区的数据类型为 TIMESTAMPTZ 的值。如果将默认时区配置为 UTC 以外的时区，可能会出现不同的结果。

```
SELECT TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE 'MST';

timezone
------------------------
2001-02-17 03:38:40+00
```

以下示例采用带有时区值的输入时间戳，其中指定时区为 EST（POSIX 中的 UTC+5），然后将其转换为 MST（POSIX 中的 UTC+7）。该示例返回一个数据类型为 TIMESTAMP 的值。

```
SELECT TIMESTAMPTZ '2001-02-16 20:38:40-05' AT TIME ZONE 'MST';
```

```
timezone
------------------------
2001-02-16 18:38:40
```

CONVERT_TIMEZONE 函数

CONVERT_TIMEZONE 将一个时区的时间戳转换为另一个时区的时间戳。该函数会自动根据夏令时调整。

语法

```
CONVERT_TIMEZONE( ['source_timezone',] 'target_timezone', 'timestamp')
```

参数

source_timezone

（可选）当前时间戳的时区。默认值为 UTC。有关更多信息，请参阅 时区使用说明。

target_timezone

新时间戳的时区。有关更多信息，请参阅 时区使用说明。

timestamp

时间戳列或隐式转换为时间戳的表达式。

返回类型

TIMESTAMP

时区使用说明

可指定 source_timezone 或 target_timezone 作为时区名称（如“非洲/坎帕拉”或“新加坡”）或作为时区缩写（如“UTC”或“PDT”）。您不必将时区名称转换为其他名称，也不必将缩写转换为其他缩写。例如，您可以从源时区名称“新加坡”中选择一个时间戳，然后将其转换为时区缩写“PDT”中的时间戳。

> **Note**
>
> 使用时区名称或时区缩写的结果可能会因当地季节性时间（如夏令时）而有所不同。

使用时区名称

要查看当前的完整时区名称列表，请运行以下命令。

```
select pg_timezone_names();
```

每行包含一个以逗号分隔的字符串，其中包含时区名称、缩写、UTC 偏移量以及用于指示时区是否遵守夏令时的指示符（t 或 f）。例如，以下代码段显示了两个生成的行。第一行是时区 Europe/Paris、缩写 CET、UTC 偏移量 01:00:00，以及用于指明它不遵守夏令时的 f。第二行是时区 Israel、缩写 IST、UTC 偏移量 02:00:00，以及用于指明它不遵守夏令时的 f。

```
pg_timezone_names
-----------------
(Europe/Paris,CET,01:00:00,f)
(Israel,IST,02:00:00,f)
```

运行 SQL 语句以获取整个列表并找到时区名称。返回大约 600 行。虽然部分返回的时区名称是大写的首字母缩略词（例如，GB、PRC、ROK），但 CONVERT_TIMEZONE 函数将它们视为时区名称，而不是时区缩写。

如果您使用时区名称指定时区，CONVERT_TIMEZONE 会根据夏令时（DST）或在 timestamp 指定的日期和时间期间为该时区实行的任何其他当地季节性协议，如夏令时、标准时间或冬令时，自动进行调整。例如，“欧洲/伦敦”在冬季表示 UTC，在夏季加一小时。

使用时区缩写

要查看当前的完整时区缩写列表，请运行以下命令。

```
select pg_timezone_abbrevs();
```

结果包含一个以逗号分隔的字符串，其中包含时区缩写、UTC 偏移量以及用于指示时区是否遵守夏令时的指示符（t 或 f）。例如，以下代码段显示了两个生成的行。第一行包含太平洋夏令时的缩写 PDT、UTC 偏移量 -07:00:00，以及用于指示它遵守夏令时的 t。第二行包含太平洋标准时间的缩写 PST、UTC 偏移量 -08:00:00，以及用于指示它不遵守夏令时的 f。

```
pg_timezone_abbrevs
--------------------
(PDT,-07:00:00,t)
(PST,-08:00:00,f)
```

运行 SQL 语句以获取整个列表，并根据其偏移量和夏令时指示符查找缩写。返回大约 200 行。

时区缩写表示与 UTC 的固定偏移量。如果您使用时区缩写指定时区，CONVERT_TIMEZONE 将使用与 UTC 的固定偏移量，并且不会针对任何本地季节性协议进行调整。

使用 POSIX 样式格式

POSIX 样式的时区规范采用 STDoffset 或 STDoffsetDST 的形式，其中 STD 是时区缩写，offset 是从 UTC 向西的小时数偏移，而 DST 是可选的夏令时区缩写。夏令时时间假定为比给定的偏移提前一个小时。

POSIX 样式的时区格式使用格林威治以西的正偏移，而 ISO-8601 约定则使用格林威治以东的正偏移量。

以下是 POSIX 样式时区的示例：

- PST8
- PST8PDT
- EST5
- EST5EDT

> **Note**
>
> Amazon Redshift 不验证 POSIX 样式时区规格，因此可能将时区设置为无效值。例如，即使将时区设置成了无效值，以下命令也没有返回错误。
>
> ```
> set timezone to ‘xxx36’;
> ```

示例

许多示例使用 TICKIT 样本数据集。有关更多信息，请参阅示例数据库。

以下示例将时间戳值从默认的 UTC 时区转换为 PST。

```
select convert_timezone('PST', '2008-08-21 07:23:54');

 convert_timezone
```

```
------------------------
2008-08-20 23:23:54
```

以下示例将 LISTTIME 列中的时间戳值从默认 UTC 时区转换为 PST。尽管时间戳在夏令时间段内，但它会转换为标准时间，因为目标时区被指定为缩写 (PST)。

```
select listtime, convert_timezone('PST', listtime) from listing
where listid = 16;

     listtime       |   convert_timezone
--------------------+-------------------
2008-08-24 09:36:12     2008-08-24 01:36:12
```

以下示例将 LISTTIME 列中的时间戳从默认 UTC 时区转换为美国/太平洋时区。目标时区使用时区名称，时间戳位于夏令时间段内，因此函数返回夏令时。

```
select listtime, convert_timezone('US/Pacific', listtime) from listing
where listid = 16;

     listtime       |   convert_timezone
--------------------+---------------------
2008-08-24 09:36:12 | 2008-08-24 02:36:12
```

以下示例将时间戳字符串从 EST 转换为 PST：

```
select convert_timezone('EST', 'PST', '20080305 12:25:29');

 convert_timezone
-------------------
2008-03-05 09:25:29
```

以下示例将时间戳转换为美国东部标准时间，因为目标时区使用时区名称 (America/New_York)，并且时间戳在标准时间段内。

```
select convert_timezone('America/New_York', '2013-02-01 08:00:00');

 convert_timezone
---------------------
2013-02-01 03:00:00
(1 row)
```

以下示例将时间戳转换为美国东部夏令时，因为目标时区使用时区名称 (America/New_York)，并且时间戳在夏令时时间段内。

```
select convert_timezone('America/New_York', '2013-06-01 08:00:00');

 convert_timezone
---------------------
2013-06-01 04:00:00
(1 row)
```

以下示例演示了偏移的用法。

```
SELECT CONVERT_TIMEZONE('GMT','NEWZONE +2','2014-05-17 12:00:00') as newzone_plus_2,
CONVERT_TIMEZONE('GMT','NEWZONE-2:15','2014-05-17 12:00:00') as newzone_minus_2_15,
CONVERT_TIMEZONE('GMT','America/Los_Angeles+2','2014-05-17 12:00:00') as la_plus_2,
CONVERT_TIMEZONE('GMT','GMT+2','2014-05-17 12:00:00') as gmt_plus_2;

   newzone_plus_2    | newzone_minus_2_15  |      la_plus_2      |     gmt_plus_2
---------------------+---------------------+---------------------+---------------------
2014-05-17 10:00:00 | 2014-05-17 14:15:00 | 2014-05-17 10:00:00 | 2014-05-17 10:00:00
(1 row)
```

CURRENT_DATE 函数

CURRENT_DATE 以默认格式 YYYY-MM-DD 返回当前会话时区（预设情况下为 UTC）中的日期。

> **Note**
>
> CURRENT_DATE 返回当前事务的开始日期，而不是当前语句的开始日期。考虑这样的场景，即您在 2008 年 1 月 10 日 23:59 开始一个包含多个语句的事务，而包含 CURRENT_DATE 的语句在 2008 年 2 月 10 日 00:00 运行。CURRENT_DATE 返回 10/01/08，而不是 10/02/08。

语法

```
CURRENT_DATE
```

返回类型

DATE

示例

以下示例返回当前日期（在运行函数的 AWS 区域）。

```
select current_date;

   date
------------
2008-10-01
```

以下示例创建一个表，插入一行，其中列 todays_date 的默认设置是 CURRENT_DATE，然后选择表中的所有行。

```
CREATE TABLE insert_dates(
    label varchar(128) NOT NULL,
    todays_date DATE DEFAULT CURRENT_DATE);

INSERT INTO insert_dates(label)
VALUES('Date row inserted');

SELECT * FROM insert_dates;


 label            | todays_date
------------------+-------------
Date row inserted | 2023-05-10
```

DATE_CMP 函数

DATE_CMP 将比较两个日期。如果日期相同，函数将返回 0，date1 较大时返回 1，date2 较大时返回 -1。

语法

```
DATE_CMP(date1, date2)
```

参数

date1

数据类型 DATE 的列，或一个计算结果为 DATE 类型的表达式。

date2

数据类型 DATE 的列，或一个计算结果为 DATE 类型的表达式。

返回类型

INTEGER

示例

以下查询将 CALDATE 列中的 DATE 值与日期 2008 年 1 月 4 日进行比较，并返回 CALDATE 中的值是在 2008 年 1 月 4 日之前（-1）、等于这一天（0），还是在这一天之后（1）：

```
select caldate, '2008-01-04',
date_cmp(caldate,'2008-01-04')
from date
order by dateid
limit 10;

 caldate   |  ?column?  | date_cmp
-----------+------------+----------
2008-01-01 | 2008-01-04 |       -1
2008-01-02 | 2008-01-04 |       -1
2008-01-03 | 2008-01-04 |       -1
2008-01-04 | 2008-01-04 |        0
2008-01-05 | 2008-01-04 |        1
2008-01-06 | 2008-01-04 |        1
2008-01-07 | 2008-01-04 |        1
2008-01-08 | 2008-01-04 |        1
2008-01-09 | 2008-01-04 |        1
2008-01-10 | 2008-01-04 |        1
(10 rows)
```

DATE_CMP_TIMESTAMP 函数

DATE_CMP_TIMESTAMP 将日期与时间戳进行比较，并在值相同时返回 0，date 按时间顺序较大时返回 1，timestamp 较大时返回 -1。

语法

```
DATE_CMP_TIMESTAMP(date, timestamp)
```

参数

date

数据类型 DATE 的列，或一个计算结果为 DATE 类型的表达式。

timestamp

数据类型 TIMESTAMP 的列，或一个计算结果为 TIMESTAMP 类型的表达式。

返回类型

INTEGER

示例

以下示例将日期 2008-06-18 与 LISTTIME 进行比较。LISTTIME 列的值是时间戳。在此日期之前创建的清单返回 1；此日期之后创建的清单返回 -1。

```
select listid, '2008-06-18', listtime,
date_cmp_timestamp('2008-06-18', listtime)
from listing
order by 1, 2, 3, 4
limit 10;

 listid |  ?column?  |      listtime       | date_cmp_timestamp
--------+------------+---------------------+--------------------
      1 | 2008-06-18 | 2008-01-24 06:43:29 |                  1
      2 | 2008-06-18 | 2008-03-05 12:25:29 |                  1
      3 | 2008-06-18 | 2008-11-01 07:35:33 |                 -1
      4 | 2008-06-18 | 2008-05-24 01:18:37 |                  1
      5 | 2008-06-18 | 2008-05-17 02:29:11 |                  1
      6 | 2008-06-18 | 2008-08-15 02:08:13 |                 -1
      7 | 2008-06-18 | 2008-11-15 09:38:15 |                 -1
      8 | 2008-06-18 | 2008-11-09 05:07:30 |                 -1
      9 | 2008-06-18 | 2008-09-09 08:03:36 |                 -1
     10 | 2008-06-18 | 2008-06-17 09:44:54 |                  1
(10 rows)
```

DATE_CMP_TIMESTAMPTZ 函数

DATE_CMP_TIMESTAMPTZ 将日期与带有时区的时间戳进行比较，并在值相同时返回 0，date 按时间顺序较大时返回 1，timestamptz 较大时返回 -1。

语法

```
DATE_CMP_TIMESTAMPTZ(date, timestamptz)
```

参数

date

数据类型为 DATE 的列，或一个隐式计算结果为 DATE 类型的表达式。

timestamptz

数据类型为 TIMESTAMPTZ 的列，或一个隐式计算结果为 TIMESTAMPTZ 类型的表达式。

返回类型

INTEGER

示例

以下示例将日期 2008-06-18 与 LISTTIME 进行比较。在此日期之前创建的清单返回 1；此日期之后创建的清单返回 -1。

```
select listid, '2008-06-18', CAST(listtime AS timestamptz),
date_cmp_timestamptz('2008-06-18', CAST(listtime AS timestamptz))
from listing
order by 1, 2, 3, 4
limit 10;

 listid |  ?column?  |      timestamptz       | date_cmp_timestamptz
--------+------------+------------------------+----------------------
      1 | 2008-06-18 | 2008-01-24 06:43:29+00 |                    1
      2 | 2008-06-18 | 2008-03-05 12:25:29+00 |                    1
      3 | 2008-06-18 | 2008-11-01 07:35:33+00 |                   -1
      4 | 2008-06-18 | 2008-05-24 01:18:37+00 |                    1
      5 | 2008-06-18 | 2008-05-17 02:29:11+00 |                    1
      6 | 2008-06-18 | 2008-08-15 02:08:13+00 |                   -1
      7 | 2008-06-18 | 2008-11-15 09:38:15+00 |                   -1
      8 | 2008-06-18 | 2008-11-09 05:07:30+00 |                   -1
      9 | 2008-06-18 | 2008-09-09 08:03:36+00 |                   -1
     10 | 2008-06-18 | 2008-06-17 09:44:54+00 |                    1
(10 rows)
```

DATEADD 函数

按指定的时间间隔递增 DATE、TIME、TIMETZ 或 TIMESTAMP 值。

语法

```
DATEADD( datepart, interval, {date|time|timetz|timestamp} )
```

参数

datepart

函数操作的日期部分（例如年、月、日或小时）。有关更多信息，请参阅 日期或时间戳函数的日期部分。

interval

指定要添加到目标表达式的时间间隔（例如天数）的整数。负整数减去时间间隔。

date|time|timetz|timestamp

DATE、TIME、TIMETZ 或 TIMESTAMP 列或隐式转换为 DATE、TIME、TIMETZ 或 TIMESTAMP 的表达式。DATE、TIME、TIMETZ 或 TIMESTAMP 表达式必须包含指定的日期部分。

返回类型

TIMESTAMP 或 TIME 或 TIMETZ，具体取决于输入数据类型。

具有 DATE 列的示例

以下示例为 DATE 表中存在的 11 月中的每个日期添加 30 天。

```
select dateadd(day,30,caldate) as novplus30
from date
where month='NOV'
order by dateid;

novplus30
---------------------
2008-12-01 00:00:00
2008-12-02 00:00:00
2008-12-03 00:00:00
```

```
...
(30 rows)
```

以下示例将 18 个月添加到文本日期值。

```
select dateadd(month,18,'2008-02-28');

date_add
---------------------
2009-08-28 00:00:00
(1 row)
```

DATEADD 函数的默认列名称为 DATE_ADD。日期值的默认时间戳为 00:00:00。

以下示例向未指定时间戳的日期值添加 30 分钟。

```
select dateadd(m,30,'2008-02-28');

date_add
---------------------
2008-02-28 00:30:00
(1 row)
```

您可以用全名或缩写来命名日期部分。在此情况下，m 代表几分钟，而不是几个月。

具有 TIME 列的示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

以下示例为 TIME_TEST 表中的每个 TIME_VAL 添加 5 分钟。

```
select dateadd(minute,5,time_val) as minplus5 from time_test;

minplus5
```

```
---------------
20:05:00
00:05:00.5550
01:03:00
```

以下示例为文本时间值添加 8 小时。

```
select dateadd(hour, 8, time '13:24:55');

date_add
---------------
21:24:55
```

以下示例显示时间何时超过 24:00:00 或低于 00:00:00。

```
select dateadd(hour, 12, time '13:24:55');

date_add
---------------
01:24:55
```

具有 TIMETZ 列的示例

这些示例中的输出值以 UTC 为默认时区。

下面的示例表 TIMETZ_TEST 具有一个列 TIMETZ_VAL（类型 TIMETZ），其中插入了三个值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

下面的示例为 TIMETZ_TEST 表中的每个 TIMETZ_VAL 添加 5 分钟。

```
select dateadd(minute,5,timetz_val) as minplus5_tz from timetz_test;

minplus5_tz
---------------
04:05:00+00
```

```
00:05:00.5550+00
06:03:00+00
```

以下示例将 2 小时添加到一个文本 timetz 值。

```
select dateadd(hour, 2, timetz '13:24:55 PST');

date_add
---------------
23:24:55+00
```

具有 TIMESTAMP 列的示例

这些示例中的输出值以 UTC 为默认时区。

下面的示例表 TIMESTAMP_TEST 具有一个列 TIMESTAMP_VAL（类型为 TIMESTAMP），其中插入了三个值。

```
SELECT timestamp_val FROM timestamp_test;

timestamp_val
------------------
1988-05-15 10:23:31
2021-03-18 17:20:41
2023-06-02 18:11:12
```

以下示例仅向 TIMESTAMP_TEST 中 2000 年之前的 TIMESTAMP_VAL 值增加 20 年。

```
SELECT dateadd(year,20,timestamp_val)
FROM timestamp_test
WHERE timestamp_val < to_timestamp('2000-01-01 00:00:00', 'YYYY-MM-DD HH:MI:SS');

date_add
---------------
2008-05-15 10:23:31
```

以下示例向不带秒指示器的文本时间戳值增加 5 秒。

```
SELECT dateadd(second, 5, timestamp '2001-06-06');

date_add
```

```
---------------
2001-06-06 00:00:05
```

使用说明

DATEADD(month, ...) 和 ADD_MONTHS 函数以不同的方式处理位于月末的日期：

- ADD_MONTHS：如果添加到的日期是该月的最后一天，则无论该月有多少天，结果始终是结果月份的最后一天。例如，4 月 30 日 + 1 个月是 5 月 31 日。

```
select add_months('2008-04-30',1);

add_months
---------------------
2008-05-31 00:00:00
(1 row)
```

- DATEADD：如果添加到的日期中的天数少于结果月份，则结果是结果月份的对应日期，而不是该月的最后一天。例如，4 月 30 日 + 1 个月是 5 月 30 日。

```
select dateadd(month,1,'2008-04-30');

date_add
---------------------
2008-05-30 00:00:00
(1 row)
```

当使用 dateadd(month, 12,...) 或 dateadd(year, 1, ...) 时，DATEADD 函数以不同方式处理闰年日期 02-29。

```
select dateadd(month,12,'2016-02-29');

date_add
---------------------
2017-02-28 00:00:00

select dateadd(year, 1, '2016-02-29');

date_add
---------------------
2017-03-01 00:00:00
```

DATEDIFF 函数

DATEDIFF 返回两个日期或时间表达式的日期部分之间的差异。

语法

```
DATEDIFF( datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp} )
```

参数

datepart

该函数运行所依据的日期或时间值的特定部分（年、月或日、小时、分钟、秒、毫秒或微秒）。有关更多信息，请参阅 日期或时间戳函数的日期部分。

具体而言，DATEDIFF 确定在两个表达式之间交叉的日期部分边界的数量。例如，假设您计算两个日期 12-31-2008 与 01-01-2009 之间的年份差异。在这种情况下，函数返回 1 年，尽管这些日期仅相隔一天。如果您发现两个时间戳 01-01-2009 8:30:00 与 01-01-2009 10:00:00 之间存在小时差，则结果为 2 小时。如果您发现两个时间戳 8:30:00 与 10:00:00 之间存在小时差，则结果为 2 小时。

date|time|timetz|timestamp

DATE、TIME、TIMETZ 或 TIMESTAMP 列或隐式转换为 DATE、TIME、TIMETZ 或 TIMESTAMP 的表达式。表达式必须同时包含指定的日期或时间部分。如果第二个日期或时间晚于第一个日期或时间，则结果为正值。如果第二个日期或时间早于第一个日期或时间，则结果为负值。

返回类型

BIGINT

具有 DATE 列的示例

以下示例查找两个文本日期值之间的差异（以周数为单位）。

```
select datediff(week,'2009-01-01','2009-12-31') as numweeks;

numweeks
----------
52
```

```
(1 row)
```

以下示例查找两个文本日期值之间的差异，以小时为单位。如果您没有为日期提供时间值，则默认为 00:00:00。

```
select datediff(hour, '2023-01-01', '2023-01-03 05:04:03');

date_diff
----------
53
(1 row)
```

以下示例查找两个文本 TIMESTAMETZ 值之间的差异，以天为单位。

```
Select datediff(days, 'Jun 1,2008  09:59:59 EST', 'Jul 4,2008  09:59:59 EST')

date_diff
----------
33
```

以下示例查找表中同一行的两个日期之间的差异，以天为单位。

```
select * from date_table;

start_date |   end_date
-----------+-----------
2009-01-01 | 2009-03-23
2023-01-04 | 2024-05-04
(2 rows)

select datediff(day, start_date, end_date) as duration from date_table;

duration
---------
      81
     486
(2 rows)
```

以下示例查找过去日期和今天日期中的文本值之间的差异（以季度数为单位）。此示例假定当前日期为 2008 年 6 月 5 日。您可以可以用全名或缩写来命名日期部分。DATEDIFF 函数的默认列名称为 DATE_DIFF。

```
select datediff(qtr, '1998-07-01', current_date);

date_diff
-----------
40
(1 row)
```

以下示例将 SALES 和 LISTING 表联接，以计算它们列出后多少天清单 1000 到 1005 的所有票证被售出。这些清单的最长销售等待时间为 15 天，最短等待时间不到一天（0 天）。

```
select priceperticket,
datediff(day, listtime, saletime) as wait
from sales, listing where sales.listid = listing.listid
and sales.listid between 1000 and 1005
order by wait desc, priceperticket desc;

priceperticket | wait
---------------+------
 96.00         |   15
 123.00        |   11
 131.00        |    9
 123.00        |    6
 129.00        |    4
 96.00         |    4
 96.00         |    0
(7 rows)
```

此示例计算卖家等待所有票证销售的平均小时数。

```
select avg(datediff(hours, listtime, saletime)) as avgwait
from sales, listing
where sales.listid = listing.listid;

avgwait
---------
465
(1 row)
```

具有 TIME 列的示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

以下示例查找 TIME_VAL 列与时间文本之间的小时数差异。

```
select datediff(hour, time_val, time '15:24:45') from time_test;

 date_diff
-----------
        -5
        15
        15
```

以下示例查找两个文本时间值之间的分钟数差异。

```
select datediff(minute, time '20:00:00', time '21:00:00') as nummins;

nummins
----------
60
```

具有 TIMETZ 列的示例

下面的示例表 TIMETZ_TEST 具有一个列 TIMETZ_VAL（类型 TIMETZ），其中插入了三个值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

以下示例查找 TIMETZ 文本与 timetz_val 之间的小时数差异。

```
select datediff(hours, timetz '20:00:00 PST', timetz_val) as numhours from timetz_test;
```

```
numhours
----------
0
-4
1
```

以下示例查找两个文本 TIMETZ 值之间的小时数差异。

```
select datediff(hours, timetz '20:00:00 PST', timetz '00:58:00 EST') as numhours;

numhours
----------
1
```

DATE_PART 函数

DATE_PART 从表达式中提取日期部分值。DATE_PART 是 PGDATE_PART 函数的同义词。

语法

```
DATE_PART(datepart, {date|timestamp})
```

参数

datepart

函数所操作的日期值的特定部分（例如年、月或日）的标识符文本或字符串。有关更多信息，请参阅 日期或时间戳函数的日期部分。

{date|timestamp}

日期列、时间戳列或隐式转换为日期或时间戳的表达式。date 或 timestamp 的列或表达式必须包含 datepart 中指定的日期部分。

返回类型

DOUBLE

示例

DATE_PART 函数的默认列名是 pgdate_part。

有关以下示例中使用的数据的更多信息，请参阅示例数据库。

以下示例从时间戳文本中查找分钟。

```
SELECT DATE_PART(minute, timestamp '20230104 04:05:06.789');

pgdate_part
-----------
          5
```

以下示例从时间戳文本中查找周编号。周编号计算遵循 ISO 8601 标准。有关更多信息，请参阅 Wikipedia 中的 ISO 8601。

```
SELECT DATE_PART(week, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         18
```

以下示例从时间戳文本中查找月份中的某个日期。

```
SELECT DATE_PART(day, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
          2
```

下面的示例从时间戳文本中查找星期几信息。星期几的计算是 0-6 之间的整数，从星期日开始。

```
SELECT DATE_PART(dayofweek, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
          1
```

以下示例从时间戳文本中查找世纪。世纪计算遵循 ISO 8601 标准。有关更多信息，请参阅 Wikipedia 中的 ISO 8601。

```
SELECT DATE_PART(century, timestamp '20220502 04:05:06.789');

pgdate_part
```

```
-----------
         21
```

以下示例从时间戳文本中查找千禧年。千禧年计算遵循 ISO 8601 标准。有关更多信息，请参阅 Wikipedia 中的 ISO 8601。

```
SELECT DATE_PART(millennium, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         3
```

以下示例从时间戳文本中查找微秒。微秒计算遵循 ISO 8601 标准。有关更多信息，请参阅 Wikipedia 中的 ISO 8601。

```
SELECT DATE_PART(microsecond, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
    789000
```

以下示例从日期文本中查找月份。

```
SELECT DATE_PART(month, date '20220502');

pgdate_part
-----------
         5
```

以下示例将 DATE_PART 函数应用于表中的列。

```
SELECT date_part(w, listtime) AS weeks, listtime
FROM listing
WHERE listid=10


weeks |      listtime
------+---------------------
 25   | 2008-06-17 09:44:54
(1 row)
```

您可以用全名或缩写来命名日期部分；在这种情况下，w 代表星期数。

星期日期部分返回一个从 0-6 整数，从星期日开始。将 DATE_PART 与 dow (DAYOFWEEK) 结合使用以查看星期六的活动。

```
SELECT date_part(dow, starttime) AS dow, starttime
FROM event
WHERE date_part(dow, starttime)=6
ORDER BY 2,1;

 dow |      starttime
-----+---------------------
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
...
(1147 rows)
```

DATE_PART_YEAR 函数

DATE_PART_YEAR 函数从日期中提取年份。

语法

```
DATE_PART_YEAR(date)
```

参数

date

数据类型为 DATE 的列，或一个隐式计算结果为 DATE 类型的表达式。

返回类型

INTEGER

示例

以下示例从日期文本中查找年份。

```
SELECT DATE_PART_YEAR(date '20220502 04:05:06.789');
```

```
date_part_year
---------------
2022
```

以下示例从 CALDATE 列中提取年份。CALDATE 列中的值是日期。有关此示例中使用的数据的更多信息，请参阅示例数据库。

```
select caldate, date_part_year(caldate)
from date
order by
dateid limit 10;

 caldate   | date_part_year
-----------+----------------
2008-01-01 |           2008
2008-01-02 |           2008
2008-01-03 |           2008
2008-01-04 |           2008
2008-01-05 |           2008
2008-01-06 |           2008
2008-01-07 |           2008
2008-01-08 |           2008
2008-01-09 |           2008
2008-01-10 |           2008
(10 rows)
```

DATE_TRUNC 函数

DATE_TRUNC 函数根据您指定的日期部分（如小时、天或月）截断时间戳表达式或文字。

语法

```
DATE_TRUNC('datepart', timestamp)
```

参数

datepart

截断时间戳值的日期部分。输入时间戳被截断为输入 datepart 的精度。例如，month 截断至每月的第一天。有效格式如下所示：

- microsecond、microseconds
- millisecond、milliseconds
- second、seconds
- minute、minutes
- hour、hours
- day、days
- week、weeks
- month、months
- quarter、quarters
- year、years
- decade、decades
- century、centuries
- millennium、millennia

有关某些格式的缩写的更多信息，请参阅日期或时间戳函数的日期部分

timestamp

时间戳列或隐式转换为时间戳的表达式。

返回类型

TIMESTAMP

示例

将输入时间戳截断至秒。

```
SELECT DATE_TRUNC('second', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:05:06
```

将输入时间戳截断至分钟。

```
SELECT DATE_TRUNC('minute', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:05:00
```

将输入时间戳截断至小时。

```
SELECT DATE_TRUNC('hour', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:00:00
```

将输入时间戳截断至天。

```
SELECT DATE_TRUNC('day', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 00:00:00
```

将输入时间戳截断至一个月的第一天。

```
SELECT DATE_TRUNC('month', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

将输入时间戳截断至一个季度的第一天。

```
SELECT DATE_TRUNC('quarter', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

将输入时间戳截断至一年的第一天。

```
SELECT DATE_TRUNC('year', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-01-01 00:00:00
```

将输入时间戳截断至一个世纪的第一天。

```
SELECT DATE_TRUNC('millennium', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2001-01-01 00:00:00
```

将输入时间戳截断至某周的星期一。

```
select date_trunc('week', TIMESTAMP '20220430 04:05:06.789');
date_trunc
```

```
2022-04-25 00:00:00
```

在以下示例中，DATE_TRUNC 函数使用“周”日期部分返回每周星期一的日期。

```
select date_trunc('week', saletime), sum(pricepaid) from sales where
saletime like '2008-09%' group by date_trunc('week', saletime) order by 1;

date_trunc  |    sum
------------+-------------
2008-09-01  | 2474899
2008-09-08  | 2412354
2008-09-15  | 2364707
2008-09-22  | 2359351
2008-09-29  |  705249
```

EXTRACT 函数

EXTRACT 函数返回 TIMESTAMP、TIMESTAMPTZ、TIME、TIMETZ、INTERVAL YEAR TO MONTH 或 INTERVAL DAY TO SECOND 值中的日期或时间部分。示例包括时间戳中的日、月、年、小时、分钟、秒、毫秒或微秒。

语法

```
EXTRACT(datepart FROM source)
```

参数

datepart

要提取的日期或时间的子字段，例如日、月、年、小时、分钟、毫秒或微秒。有关可能的值，请参阅日期或时间戳函数的日期部分。

source

计算结果为 TIMESTAMP、TIMESTAMPTZ、TIME、TIMETZ、INTERVAL YEAR TO MONTH 或 INTERVAL DAY TO SECOND 数据类型的列或表达式。

返回类型

如果 source 值的计算结果为数据类型 TIMESTAMP、TIME、TIMETZ、INTERVAL YEAR TO MONTH 或 INTERVAL DAY TO SECOND，则为 INTEGER。

如果 source 值的计算结果为数据类型 TIMESTAMPTZ，则为 DOUBLE PRECISION。

TIMESTAMP 示例

以下示例确定支付价格为 10000 美元或更高的销售周数。此示例使用 TICKIT 数据。有关更多信息，请参阅 示例数据库。

```
select salesid, extract(week from saletime) as weeknum
from sales
where pricepaid > 9999
order by 2;

salesid | weeknum
--------+---------
 159073 |       6
 160318 |       8
 161723 |      26
```

以下示例从文本时间戳值返回分钟值。

```
select extract(minute from timestamp '2009-09-09 12:08:43');

date_part
-----------
8
```

以下示例从文本时间戳值返回毫秒值。

```
select extract(ms from timestamp '2009-09-09 12:08:43.101');

date_part
-----------
101
```

TIMESTAMPTZ 示例

以下示例从文本 timestamptz 值返回年份值。

```
select extract(year from timestamptz '1.12.1997 07:37:16.00 PST');

date_part
-----------
```

```
1997
```

TIME 示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

```
select time_val from time_test;

time_val
--------------------
20:00:00
00:00:00.5550
00:58:00
```

以下示例从每个 time_val 中提取分钟数。

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

以下示例从每个 time_val 中提取小时数。

```
select extract(hour from time_val) as hours from time_test;

hours
-----------
         20
         0
         0
```

以下示例从文本值中提取毫秒。

```
select extract(ms from time '18:25:33.123456');

 date_part
-----------
     123
```

TIMETZ 示例

下面的示例表 TIMETZ_TEST 具有一个列 TIMETZ_VAL（类型 TIMETZ），其中插入了三个值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

以下示例从每个 timetz_val 中提取小时数。

```
select extract(hour from timetz_val) as hours from time_test;

hours
-----------
         4
         0
         5
```

以下示例从文本值中提取毫秒。在处理提取之前，文本不会转换为 UTC。

```
select extract(ms from timetz '18:25:33.123456 EST');

 date_part
-----------
     123
```

以下示例从文本 timetz 值返回与 UTC 的时区偏移小时数。

```
select extract(timezone_hour from timetz '1.12.1997 07:37:16.00 PDT');

date_part
-----------
-7
```

INTERVAL YEAR TO MONTH 和 INTERVAL DAY TO SECOND 示例

以下示例从定义 36 小时（即 1 天 12 小时）的 INTERVAL DAY TO SECOND 中提取 1 的日部分。

```
select EXTRACT('days' from INTERVAL '36 hours' DAY TO SECOND)

 date_part
------------------
 1
```

以下示例从定义 15 个月（即 1 年 3 个月）的 YEAR TO MONTH 中提取 3 的月份部分。

```
select EXTRACT('month' from INTERVAL '15 months' YEAR TO MONTH)

 date_part
------------------
 3
```

以下示例从 30 个月（即 2 年 6 个月）中提取 6 的月份部分。

```
select EXTRACT('month' from INTERVAL '30' MONTH)

 date_part
------------------
 6
```

以下示例从 50 个小时（即 2 天 2 小时）中提取 2 的小时部分。

```
select EXTRACT('hours' from INTERVAL '50' HOUR)

 date_part
------------------
 2
```

以下示例从 1 小时 11 分 11.123 秒中提取 11 的分钟部分。

```
select EXTRACT('minute' from INTERVAL '70 minutes 70.123 seconds' MINUTE TO SECOND)

 date_part
------------------
 11
```

以下示例从 1 天 1 小时 1 分 1.11 秒中提取 1.11 的秒部分。

```
select EXTRACT('seconds' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND)

 date_part
------------------
 1.11
```

以下示例提取 INTERVAL 中的总小时数。提取每个部分并将其添加到总数中。

```
select EXTRACT('days' from INTERVAL '50' HOUR) * 24 + EXTRACT('hours' from INTERVAL
 '50' HOUR)

 ?column?
------------------
 50
```

以下示例提取 INTERVAL 中的总秒数。提取每个部分并将其添加到总数中。

```
select EXTRACT('days' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND) * 86400 +
       EXTRACT('hours' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND) * 3600 +
       EXTRACT('minutes' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND) * 60 +
       EXTRACT('seconds' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND)

 ?column?
------------------
 90061.11
```

GETDATE 函数

GETDATE 返回当前会话时区（预设情况下为 UTC）中的当前日期和时间。它返回当前语句的开始日期或时间，即使它在事务块中也是如此。

语法

```
GETDATE()
```

括号为必填项。

返回类型

TIMESTAMP

示例

以下示例使用 GETDATE 函数返回当前日期的完整时间戳。

```
select getdate();

timestamp
---------------------
2008-12-04 16:10:43
```

以下示例使用 TRUNC 函数内的 GETDATE 函数来返回没有时间的当前日期。

```
select trunc(getdate());

trunc
------------
2008-12-04
```

INTERVAL_CMP 函数

INTERVAL_CMP 比较两个时间间隔并在第一个时间间隔较大时返回 1，在第二个时间间隔较大时返回 -1，并在时间间隔相等时返回 0。有关更多信息，请参阅 不带限定词语法的间隔文字示例。

语法

```
INTERVAL_CMP(interval1, interval2)
```

参数

interval1

时间间隔文本值。

interval2

时间间隔文本值。

返回类型

INTEGER

示例

以下示例将 3 days 的值与 1 year 进行比较。

```
select interval_cmp('3 days','1 year');

interval_cmp
--------------
-1
```

此示例比较了值 7 days 与 1 week。

```
select interval_cmp('7 days','1 week');

interval_cmp
--------------
0
```

以下示例将 1 year 的值与 3 days 进行比较。

```
select interval_cmp('1 year','3 days');

interval_cmp
--------------
1
```

LAST_DAY 函数

LAST_DAY 返回该月最后一天的日期，该日期包含 date。无论 date 参数的数据类型如何，返回类型始终为 DATE。

有关检索特定日期部分的更多信息，请参阅DATE_TRUNC 函数。

语法

```
LAST_DAY( { date | timestamp } )
```

参数

date | timestamp

数据类型为 DATE 或 TIMESTAMP 的列，或一个隐式计算结果为 DATE 或 TIMESTAMP 类型的表达式。

返回类型

DATE

示例

以下示例返回当前月份最后一天的日期。

```
select last_day(sysdate);

  last_day
------------
 2014-01-31
```

以下示例返回该月最后 7 天每天售出的票证数量。SALETIME 列中的值是时间戳。

```
select datediff(day, saletime, last_day(saletime)) as "Days Remaining", sum(qtysold)
from sales
where datediff(day, saletime, last_day(saletime)) < 7
group by 1
order by 1;

days remaining |  sum
---------------+-------
             0 | 10140
             1 | 11187
             2 | 11515
             3 | 11217
             4 | 11446
             5 | 11708
             6 | 10988
(7 rows)
```

MONTHS_BETWEEN 函数

MONTHS_BETWEEN 确定两个日期之间相隔的月数。

如果第一个日期晚于第二个日期，则结果为正；否则，结果为负数。

如果任一参数为 null，则结果为 NULL。

语法

```
MONTHS_BETWEEN( date1, date2 )
```

参数

date1

数据类型为 DATE 的列，或一个隐式计算结果为 DATE 类型的表达式。

date2

数据类型为 DATE 的列，或一个隐式计算结果为 DATE 类型的表达式。

返回类型

FLOAT8

结果的整数部分基于日期的年份和月份值之间的差值。结果的小数部分根据日期的日期和时间戳值计算，假定一个月为 31 天。

如果 date1 和 date2 都包含一个月内的相同日期（例如，1/15/14 和 2/15/14）或该月的最后一天（例如 8/31/14 和 9/30/14），则结果是基于日期的年份和月份值的整数，无论时间戳部分是否匹配（如果存在）。

示例

以下示例返回 1969 年 1 月 18 日至 1969 年 3 月 18 日之间的月份。

```
select months_between('1969-01-18', '1969-03-18')
as months;

months
----------
-2
```

以下示例返回 1969 年 1 月 18 日至 1969 年 1 月 18 日之间的月数。

```
select months_between('1969-01-18', '1969-01-18')
as months;

months
```

```
----------
0
```

以下示例返回事件的第一个和最后一个展示之间的月份。

```
select eventname,
min(starttime) as first_show,
max(starttime) as last_show,
months_between(max(starttime),min(starttime)) as month_diff
from event
group by eventname
order by eventname
limit 5;

eventname         first_show             last_show              month_diff
-------------------------------------------------------------------------
.38 Special       2008-01-21 19:30:00.0  2008-12-25 15:00:00.0  11.12
3 Doors Down      2008-01-03 15:00:00.0  2008-12-01 19:30:00.0  10.94
70s Soul Jam      2008-01-16 19:30:00.0  2008-12-07 14:00:00.0  10.7
A Bronx Tale      2008-01-21 19:00:00.0  2008-12-15 15:00:00.0  10.8
A Catered Affair  2008-01-08 19:30:00.0  2008-12-19 19:00:00.0  11.35
```

NEXT_DAY 函数

NEXT_DAY 返回比给定日期晚的指定日期的第一个实例的日期。

如果 day 值与给定日期为一个星期中的同一天，则会返回当天的下一个匹配项。

语法

```
NEXT_DAY( { date | timestamp }, day )
```

参数

date | timestamp

数据类型为 DATE 或 TIMESTAMP 的列，或一个隐式计算结果为 DATE 或 TIMESTAMP 类型的表达式。

day

一个包含任何日期的名称的字符串。大小写不重要。

有效的值如下所示。

天	值
Sunday	Su、Sun、Sunday
Monday	M、Mo、Mon、Monday
星期二	Tu、Tue、Tues、Tuesday
星期三	W、We、Wed、Wednesday
星期四	Th、Thu、Thurs、Thursday
Friday	F、Fr、Fri、Friday
Saturday	Sa、Sat、Saturday

返回类型

DATE

示例

以下示例返回 2014 年 8 月 20 日之后第一个星期二的日期。

```
select next_day('2014-08-20','Tuesday');

next_day
-----------
2014-08-26
```

以下示例返回 2008 年 1 月 1 日之后的第一个星期二的日期，时间为 5:54:44。

```
select listtime, next_day(listtime, 'Tue') from listing limit 1;

listtime            | next_day
--------------------+-----------
2008-01-01 05:54:44 | 2008-01-08
```

以下示例获取第三季度的目标营销日期。

```
select username, (firstname ||' '|| lastname) as name,
eventname, caldate, next_day (caldate, 'Monday') as marketing_target
from sales, date, users, event
where sales.buyerid = users.userid
and sales.eventid = event.eventid
and event.dateid = date.dateid
and date.qtr = 3
order by marketing_target, eventname, name;

username  |     name          |     eventname        |    caldate   |
 marketing_target
----------+-------------------+----------------------+--------------
+-------------------
MBO26QSG  |   Callum Atkinson | .38 Special          |  2008-07-06  | 2008-07-07
WCR50YIU  |   Erasmus Alvarez | A Doll's House       |  2008-07-03  | 2008-07-07
CKT70OIE  |   Hadassah Adkins | Ana Gabriel          |  2008-07-06  | 2008-07-07
VVG07OUO  |   Nathan Abbott   | Armando Manzanero    |  2008-07-04  | 2008-07-07
GEW77SII  |   Scarlet Avila   | August: Osage County |  2008-07-06  | 2008-07-07
ECR71CVS  |   Caryn Adkins    | Ben Folds            |  2008-07-03  | 2008-07-07
KUW82CYU  |   Kaden Aguilar   | Bette Midler         |  2008-07-01  | 2008-07-07
WZE78DJZ  |   Kay Avila       | Bette Midler         |  2008-07-01  | 2008-07-07
HXY04NVE  |   Dante Austin    | Britney Spears       |  2008-07-02  | 2008-07-07
URY81YWF  |   Wilma Anthony   | Britney Spears       |  2008-07-02  | 2008-07-07
```

SYSDATE 函数

SYSDATE 返回当前会话时区（预设情况下为 UTC）中的当前日期和时间。

> **Note**
>
> SYSDATE 返回当前事务的开始日期和时间，而不是当前语句的开始日期和时间。

语法

```
SYSDATE
```

此函数不需要任何参数。

返回类型

TIMESTAMP

示例

以下示例使用 SYSDATE 函数返回当前日期的完整时间戳。

```
select sysdate;

timestamp
----------------------------
2008-12-04 16:10:43.976353
```

以下示例使用 TRUNC 函数内的 SYSDATE 函数来返回没有时间的当前日期。

```
select trunc(sysdate);

trunc
------------
2008-12-04
```

以下查询返回介于发出查询的日期和 120 天之前的任何日期之间的日期的销售信息。

```
select salesid, pricepaid, trunc(saletime) as saletime, trunc(sysdate) as now
from sales
where saletime between trunc(sysdate)-120 and trunc(sysdate)
order by saletime asc;

 salesid | pricepaid |  saletime  |    now
---------+-----------+------------+------------
91535    |    670.00 | 2008-08-07 | 2008-12-05
91635    |    365.00 | 2008-08-07 | 2008-12-05
91901    |   1002.00 | 2008-08-07 | 2008-12-05
...
```

TIMEOFDAY 函数

TIMEOFDAY 是一个特殊的别名，用于将工作日、日期和时间作为字符串值返回。它返回当前语句的时间字符串，即使它在事务块内也是如此。

语法

```
TIMEOFDAY()
```

返回类型

VARCHAR

示例

以下示例通过使用 TIMEOFDAY 函数返回当前日期和时间。

```
select timeofday();

timeofday
------------
Thu Sep 19 22:53:50.333525 2013 UTC
```

TIMESTAMP_CMP 函数

比较两个字符串的值并返回整数。如果时间戳相同，此函数返回 0。如果第一个时间戳较大，则函数返回 1。如果第二个时间戳较大，则函数返回 -1。

语法

```
TIMESTAMP_CMP(timestamp1, timestamp2)
```

参数

timestamp1

数据类型为 TIMESTAMP 的列，或一个隐式计算结果为 TIMESTAMP 类型的表达式。

timestamp2

数据类型为 TIMESTAMP 的列，或一个隐式计算结果为 TIMESTAMP 类型的表达式。

返回类型

INTEGER

示例

以下示例比较了时间戳并显示了比较结果。

```
SELECT TIMESTAMP_CMP('2008-01-24 06:43:29', '2008-01-24 06:43:29'),
 TIMESTAMP_CMP('2008-01-24 06:43:29', '2008-02-18 02:36:48'), TIMESTAMP_CMP('2008-02-18
 02:36:48', '2008-01-24 06:43:29');
```

```
timestamp_cmp  | timestamp_cmp | timestamp_cmp
---------------+---------------+---------------
             0 |            -1 |             1
```

以下示例比较清单的 LISTTIME 和 SALETIME。对于所有上架项，TIMESTAMP_CMP 的值都是 -1，因为销售的时间戳都在上架时间戳之后。

```
select listing.listid, listing.listtime,
sales.saletime, timestamp_cmp(listing.listtime, sales.saletime)
from listing, sales
where listing.listid=sales.listid
order by 1, 2, 3, 4
limit 10;

 listid |      listtime       |      saletime       | timestamp_cmp
--------+---------------------+---------------------+---------------
      1 | 2008-01-24 06:43:29 | 2008-02-18 02:36:48 |            -1
      4 | 2008-05-24 01:18:37 | 2008-06-06 05:00:16 |            -1
      5 | 2008-05-17 02:29:11 | 2008-06-06 08:26:17 |            -1
      5 | 2008-05-17 02:29:11 | 2008-06-09 08:38:52 |            -1
      6 | 2008-08-15 02:08:13 | 2008-08-31 09:17:02 |            -1
     10 | 2008-06-17 09:44:54 | 2008-06-26 12:56:06 |            -1
     10 | 2008-06-17 09:44:54 | 2008-07-10 02:12:36 |            -1
     10 | 2008-06-17 09:44:54 | 2008-07-16 11:59:24 |            -1
     10 | 2008-06-17 09:44:54 | 2008-07-22 02:23:17 |            -1
     12 | 2008-07-25 01:45:49 | 2008-08-04 03:06:36 |            -1
(10 rows)
```

此示例显示 TIMESTAMP_CMP 对于相同的时间戳返回 0：

```
select listid, timestamp_cmp(listtime, listtime)
from listing
order by 1 , 2
limit 10;

 listid | timestamp_cmp
--------+---------------
      1 |             0
      2 |             0
      3 |             0
      4 |             0
      5 |             0
```

```
      6 |             0
      7 |             0
      8 |             0
      9 |             0
     10 |             0
(10 rows)
```

TIMESTAMP_CMP_DATE 函数

TIMESTAMP_CMP_DATE 将时间戳和日期的值进行比较。如果时间戳和日期值相同，则此函数返回 0。如果时间戳按时间顺序较大，则函数返回 1。如果日期较大，则函数返回 -1。

语法

```
TIMESTAMP_CMP_DATE(timestamp, date)
```

参数

timestamp

数据类型为 TIMESTAMP 的列，或一个隐式计算结果为 TIMESTAMP 类型的表达式。

date

数据类型为 DATE 的列，或一个隐式计算结果为 DATE 类型的表达式。

返回类型

INTEGER

示例

以下示例将 LISTTIME 与日期 2008-06-18 进行比较。在此日期之后创建的清单返回 1；此日期之前创建的清单返回 -1。LISTTIME 值是时间戳。

```
select listid, listtime,
timestamp_cmp_date(listtime, '2008-06-18')
from listing
order by 1, 2, 3
limit 10;
```

```
 listid |      listtime       | timestamp_cmp_date
--------+---------------------+--------------------
      1 | 2008-01-24 06:43:29 |                 -1
      2 | 2008-03-05 12:25:29 |                 -1
      3 | 2008-11-01 07:35:33 |                  1
      4 | 2008-05-24 01:18:37 |                 -1
      5 | 2008-05-17 02:29:11 |                 -1
      6 | 2008-08-15 02:08:13 |                  1
      7 | 2008-11-15 09:38:15 |                  1
      8 | 2008-11-09 05:07:30 |                  1
      9 | 2008-09-09 08:03:36 |                  1
     10 | 2008-06-17 09:44:54 |                 -1
(10 rows)
```

TIMESTAMP_CMP_TIMESTAMPTZ 函数

TIMESTAMP_CMP_TIMESTAMPTZ 将时间戳表达式的值与带有时区的时间戳表达式进行比较。如果时间戳与带有时区的时间戳值相同，则函数返回 0。如果时间戳按时间顺序较大，则函数返回 1。如果具有时区的时间戳较大，则函数返回 -1。

语法

```
TIMESTAMP_CMP_TIMESTAMPTZ(timestamp, timestamptz)
```

参数

timestamp

数据类型为 TIMESTAMP 的列，或一个隐式计算结果为 TIMESTAMP 类型的表达式。

timestamptz

数据类型为 TIMESTAMPTZ 的列，或一个隐式计算结果为 TIMESTAMPTZ 类型的表达式。

返回类型

INTEGER

示例

以下示例将时间戳与带有时区的时间戳进行比较，并显示比较结果。

```
SELECT TIMESTAMP_CMP_TIMESTAMPTZ('2008-01-24 06:43:29', '2008-01-24 06:43:29+00'),
 TIMESTAMP_CMP_TIMESTAMPTZ('2008-01-24 06:43:29', '2008-02-18 02:36:48+00'),
 TIMESTAMP_CMP_TIMESTAMPTZ('2008-02-18 02:36:48', '2008-01-24 06:43:29+00');

timestamp_cmp_timestamptz  | timestamp_cmp_timestamptz | timestamp_cmp_timestamptz
---------------------------+---------------------------+--------------------------
             0             |            -1             |             1
```

TIMESTAMPTZ_CMP 函数

TIMESTAMPTZ_CMP 比较两个时间戳的值与时区值并返回整数。如果时间戳相同，此函数返回 0。如果第一个时间戳按时间顺序较大，则函数返回 1。如果第二个时间戳较大，则函数返回 -1。

语法

```
TIMESTAMPTZ_CMP(timestamptz1, timestamptz2)
```

参数

timestamptz1

数据类型为 TIMESTAMPTZ 的列，或一个隐式计算结果为 TIMESTAMPTZ 类型的表达式。

timestamptz2

数据类型为 TIMESTAMPTZ 的列，或一个隐式计算结果为 TIMESTAMPTZ 类型的表达式。

返回类型

INTEGER

示例

以下示例比较了带有时区的时间戳，并显示比较的结果。

```
SELECT TIMESTAMPTZ_CMP('2008-01-24 06:43:29+00', '2008-01-24 06:43:29+00'),
 TIMESTAMPTZ_CMP('2008-01-24 06:43:29+00', '2008-02-18 02:36:48+00'),
 TIMESTAMPTZ_CMP('2008-02-18 02:36:48+00', '2008-01-24 06:43:29+00');

timestamptz_cmp  | timestamptz_cmp | timestamptz_cmp
-----------------+-----------------+----------------
        0        |       -1        |       1
```

TIMESTAMPTZ_CMP_DATE 函数

TIMESTAMPTZ_CMP_DATE 将时间戳和日期的值进行比较。如果时间戳和日期值相同，则此函数返回 0。如果时间戳按时间顺序较大，则函数返回 1。如果日期较大，则函数返回 -1。

语法

```
TIMESTAMPTZ_CMP_DATE(timestamptz, date)
```

参数

timestamptz

数据类型为 TIMESTAMPTZ 的列，或一个隐式计算结果为 TIMESTAMPTZ 类型的表达式。

date

数据类型为 DATE 的列，或一个隐式计算结果为 DATE 类型的表达式。

返回类型

INTEGER

示例

以下示例将 LISTTIME 作为带有时区的时间戳与日期 2008-06-18 进行比较。在此日期之后创建的清单返回 1；此日期之前创建的清单返回 -1。

```
select listid, CAST(listtime as timestamptz) as tstz,
timestamp_cmp_date(tstz, '2008-06-18')
from listing
order by 1, 2, 3
limit 10;


 listid |          tstz          | timestamptz_cmp_date
--------+------------------------+----------------------
      1 | 2008-01-24 06:43:29+00 |                   -1
      2 | 2008-03-05 12:25:29+00 |                   -1
      3 | 2008-11-01 07:35:33+00 |                    1
      4 | 2008-05-24 01:18:37+00 |                   -1
      5 | 2008-05-17 02:29:11+00 |                   -1
```

```
      6 | 2008-08-15 02:08:13+00 |               1
      7 | 2008-11-15 09:38:15+00 |               1
      8 | 2008-11-09 05:07:30+00 |               1
      9 | 2008-09-09 08:03:36+00 |               1
     10 | 2008-06-17 09:44:54+00 |              -1
(10 rows)
```

TIMESTAMPTZ_CMP_TIMESTAMP 函数

TIMESTAMPTZ_CMP_TIMESTAMP 将带有时区的时间戳表达式的值与时间戳表达式进行比较。如果带有时区的时间戳与时间戳值相同，则函数返回 0。如果带有时区的时间戳按时间顺序较大，则函数返回 1。如果时间戳较大，则函数返回 -1。

语法

```
TIMESTAMPTZ_CMP_TIMESTAMP(timestamptz, timestamp)
```

参数

timestamptz

数据类型为 TIMESTAMPTZ 的列，或一个隐式计算结果为 TIMESTAMPTZ 类型的表达式。

timestamp

数据类型为 TIMESTAMP 的列，或一个隐式计算结果为 TIMESTAMP 类型的表达式。

返回类型

INTEGER

示例

以下示例将带有时区的时间戳与时间戳进行比较，并显示比较结果。

```
SELECT TIMESTAMPTZ_CMP_TIMESTAMP('2008-01-24 06:43:29+00', '2008-01-24 06:43:29'),
 TIMESTAMPTZ_CMP_TIMESTAMP('2008-01-24 06:43:29+00', '2008-02-18 02:36:48'),
 TIMESTAMPTZ_CMP_TIMESTAMP('2008-02-18 02:36:48+00', '2008-01-24 06:43:29');

timestamptz_cmp_timestamp  | timestamptz_cmp_timestamp | timestamptz_cmp_timestamp
---------------------------+---------------------------+--------------------------
             0             |            -1             |             1
```

TIMEZONE 函数

TIMEZONE 返回指定时区的一个时间戳和时间戳值。

有关如何设置时区的信息和示例，请参阅timezone。

有关如何转换时区的信息和示例，请参阅CONVERT_TIMEZONE。

语法

```
TIMEZONE('timezone', { timestamp | timestamptz })
```

参数

timezone

返回值的时区。该时区可以指定为时区名称（例如 **'Africa/Kampala'** 或者 **'Singapore'**）或作为时区缩写（例如 **'UTC'** 或者 **'PDT'**）。要查看支持的时区名称的列表，请执行以下命令。

```
select pg_timezone_names();
```

要查看支持的时区缩写的列表，请执行以下命令。

```
select pg_timezone_abbrevs();
```

有关更多信息以及示例，请参阅 时区使用说明。

timestamp | timestamptz

一个结果是 TIMESTAMP 或 TIMESTAMPTZ 类型的表达式，或可隐式强制转换为时间戳或带有时区的时间戳的值。

返回类型

与 TIMESTAMP 表达式一起使用时的 TIMESTAMPTZ。

与 TIMESTAMPTZ 表达式一起使用时的 TIMESTAMP。

示例

以下示例使用 PST 时区中的时间戳 `2008-06-17 09:44:54` 返回 UTC 时区的时间戳。

```
SELECT TIMEZONE('PST', '2008-06-17 09:44:54');

timezone
-----------------------
2008-06-17 17:44:54+00
```

以下示例使用带有 UTC 时区的时间戳 2008-06-17 09:44:54+00 返回 PST 时区的时间戳。

```
SELECT TIMEZONE('PST', timestamptz('2008-06-17 09:44:54+00'));

timezone
-----------------------
2008-06-17 01:44:54
```

TO_TIMESTAMP 函数

TO_TIMESTAMP 将 TIMESTAMP 字符串转换为 TIMESTAMPTZ。有关适用于 Amazon Redshift 的其他日期和时间函数的列表，请参阅日期和时间函数。

语法

```
to_timestamp(timestamp, format)
```

```
to_timestamp (timestamp, format, is_strict)
```

参数

timestamp

以 format 指定的格式表示时间戳值的字符串。如果将此参数留为空，则时间戳值默认为 0001-01-01 00:00:00。

format

一个字符串文本，用于定义 timestamp 值的格式。包含时区的格式（**TZ**、**tz**，或者 **OF**）不支持作为输入。有关有效的时间戳格式，请参阅日期时间格式字符串。

is_strict

一个可选的布尔值，它指定在输入时间戳值超出范围时是否返回错误。当 is_strict 被设置为 TRUE 时，如果存在超出范围的值，则返回错误。当 is_strict 被设置为 FALSE（默认值）时，则接受溢出值。

返回类型

TIMESTAMPTZ

示例

以下示例演示使用 TO_TIMESTAMP 函数将 TIMESTAMP 字符串转换为 TIMESTAMPTZ。

```
select sysdate, to_timestamp(sysdate, 'YYYY-MM-DD HH24:MI:SS') as second;

timestamp                  | second
---------------------------   ---------------------
2021-04-05 19:27:53.281812 | 2021-04-05 19:27:53+00
```

可以传递日期的 TO_TIMESTAMP 部分。其余日期部分设置为默认值。时间包括在输出中：

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp
---------------------------
2017-01-01 00:00:00+00
```

以下 SQL 语句将字符串“2011-12-18 24:38:15”转换为 TIMESTAMPTZ。得到的结果是第二天的 TIMESTAMPTZ，因为小时数超过 24 小时：

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS');

to_timestamp
----------------------
2011-12-19 00:38:15+00
```

以下 SQL 语句将字符串“2011-12-18 24:38:15”转换为 TIMESTAMPTZ。结果产生错误，因为时间戳中的时间值超过 24 小时：

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS', TRUE);

ERROR:  date/time field time value out of range: 24:38:15.0
```

TRUNC 函数

截断 TIMESTAMP 并返回 DATE。

此函数也可以截断数值。有关更多信息，请参阅 TRUNC 函数。

语法

```
TRUNC(timestamp)
```

参数

timestamp

数据类型为 TIMESTAMP 的列，或一个隐式计算结果为 TIMESTAMP 类型的表达式。

要返回以 00:00:00 作为时间的时间戳值，请将函数结果强制转换为 TIMESTAMP。

返回类型

DATE

示例

以下示例返回 SYSDATE 函数（返回时间戳）的结果的日期部分。

```
SELECT SYSDATE;

+----------------------------+
|         timestamp          |
+----------------------------+
| 2011-07-21 10:32:38.248109 |
+----------------------------+

SELECT TRUNC(SYSDATE);

+------------+
|   trunc    |
+------------+
| 2011-07-21 |
+------------+
```

以下示例将 TRUNC 函数应用于 TIMESTAMP 列。返回类型为日期。

```
SELECT TRUNC(starttime) FROM event
ORDER BY eventid LIMIT 1;
```

```
+------------+
|   trunc    |
+------------+
| 2008-01-25 |
+------------+
```

以下示例通过将 TRUNC 函数结果强制转换为 TIMESTAMP 来返回时间戳值，其中 00:00:00 为时间。

```
SELECT CAST((TRUNC(SYSDATE)) AS TIMESTAMP);

+---------------------+
|        trunc        |
+---------------------+
| 2011-07-21 00:00:00 |
+---------------------+
```

日期或时间戳函数的日期部分

下表标识了作为以下函数参数接受的日期部分和时间部分的名称和缩写：

- DATEADD
- DATEDIFF
- DATE_PART
- EXTRACT

日期部分或时间部分	缩写
millennium、millennia	mil、mils
century、centuries	c、cent、cents
decade、decades	dec、decs
纪元	epoch（由 EXTRACT 提供支持）
year、years	y、yr、yrs
quarter、quarters	qtr、qtrs

日期部分或时间部分	缩写
month、months	mon,、mons
week、weeks	w
星期几	dayofweek、dow、dw、weekday（由 DATE_PART 和 EXTRACT 函数 提供支持） 返回 0–6 的整数（星期日是第一个数）。 Note DOW 日期部分的运行方式与用于日期时间格式字符串的星期 (D) 日期部分不同。D 是基于 1–7 的整数，其中星期日是 1。有关更多信息，请参阅 日期时间格式字符串。
一年中的日期	dayofyear、doy、dy、yearday（由 EXTRACT提供支持）
day、days	d
hour、hours	h、hr、hrs
minute、minutes	m、min、mins
second、seconds	s、sec、secs
millisecond、milliseconds	ms、msec、msecs、msecond、mseconds、millisec、millisecs、millisecon
microsecond、microseconds	microsec、microsecs、microsecond、usecond、useconds、us、usec、usecs
timezone、timezone_hour、timezone_minute	由 EXTRACT 支持，仅用于带有时区的时间戳 (TIMESTAMPTZ)。

秒、毫秒和微秒导致的结果差异

当不同的日期函数指定秒、毫秒或微秒作为日期部分时，查询结果会出现细微差异：

- EXTRACT 函数仅返回指定日期部分的整数，忽略较高级别和较低级别的日期部分。如果指定的日期部分为秒，则结果中不包括毫秒和微秒。如果指定的日期部分为毫秒，则不包括秒和微秒。如果指定的日期部分为微秒，则不包括秒和毫秒。
- DATE_PART 函数返回时间戳的完整秒部分，无论指定的日期部分是什么，从而根据需要返回十进制值或整数。

例如，比较以下查询的结果：

```
create table seconds(micro timestamp);

insert into seconds values('2009-09-21 11:10:03.189717');

select extract(sec from micro) from seconds;

date_part
-----------
3

select date_part(sec, micro) from seconds;

pgdate_part
-------------
3.189717
```

CENTURY、EPOCH、DECADE 和 MIL 说明

CENTURY 或 CENTURIES

Amazon Redshift 将 CENTURY 解释为开始于 ###1 年并结束于 ###0 年：

```
select extract (century from timestamp '2000-12-16 12:21:13');
date_part
-----------
20

select extract (century from timestamp '2001-12-16 12:21:13');
date_part
-----------
21
```

EPOCH

Amazon Redshift 的 EPOCH 实施与独立于集群所在的时区的 1970-01-01 00:00:00.000000 相关。根据集群所在的时区，您可能需要按小时差来抵消结果。

以下示例演示了以下操作：

1. 基于 EVENT 表创建名为 EVENT_EXAMPLE 的表。此 CREATE AS 命令使用 DATE_PART 函数创建日期列（预设情况下称为 PGDATE_PART），以存储每个事件的纪元值。
2. 从 PG_TABLE_DEF 中选择 EVENT_EXAMPLE 的列和数据类型。
3. 从 EVENT_EXAMPLE 表中选择 EVENTNAME、STARTTIME 和 PGDATE_PART，以查看不同的日期和时间格式。
4. 按原样从 EVENT EXAMPLE 中选择 EVENTNAME 和 STARTTIME。使用 1 秒的时间间隔将 PGDATE_PART 中的周期值转换为不带时区的时间戳，并在名为 CONVERTED_TIMESTAMP 的列中返回结果。

```
create table event_example
as select eventname, starttime, date_part(epoch, starttime) from event;

select "column", type from pg_table_def where tablename='event_example';

     column    |            type
---------------+-----------------------------
 eventname     | character varying(200)
 starttime     | timestamp without time zone
 pgdate_part   | double precision
(3 rows)
```

```
select eventname, starttime, pgdate_part from event_example;

   eventname          |      starttime      | pgdate_part
----------------------+---------------------+-------------
 Mamma Mia!           | 2008-01-01 20:00:00 |  1199217600
 Spring Awakening     | 2008-01-01 15:00:00 |  1199199600
 Nas                  | 2008-01-01 14:30:00 |  1199197800
 Hannah Montana       | 2008-01-01 19:30:00 |  1199215800
 K.D. Lang            | 2008-01-01 15:00:00 |  1199199600
 Spamalot             | 2008-01-02 20:00:00 |  1199304000
 Macbeth              | 2008-01-02 15:00:00 |  1199286000
 The Cherry Orchard   | 2008-01-02 14:30:00 |  1199284200
 Macbeth              | 2008-01-02 19:30:00 |  1199302200
```

```
 Demi Lovato          | 2008-01-02 19:30:00 |  1199302200


select eventname,
starttime,
timestamp with time zone 'epoch' + pgdate_part * interval '1 second' AS
 converted_timestamp
from event_example;

       eventname      |      starttime      | converted_timestamp
----------------------+---------------------+---------------------
 Mamma Mia!           | 2008-01-01 20:00:00 | 2008-01-01 20:00:00
 Spring Awakening     | 2008-01-01 15:00:00 | 2008-01-01 15:00:00
 Nas                  | 2008-01-01 14:30:00 | 2008-01-01 14:30:00
 Hannah Montana       | 2008-01-01 19:30:00 | 2008-01-01 19:30:00
 K.D. Lang            | 2008-01-01 15:00:00 | 2008-01-01 15:00:00
 Spamalot             | 2008-01-02 20:00:00 | 2008-01-02 20:00:00
 Macbeth              | 2008-01-02 15:00:00 | 2008-01-02 15:00:00
 The Cherry Orchard   | 2008-01-02 14:30:00 | 2008-01-02 14:30:00
 Macbeth              | 2008-01-02 19:30:00 | 2008-01-02 19:30:00
 Demi Lovato          | 2008-01-02 19:30:00 | 2008-01-02 19:30:00
 ...
```

DECADE 或 DECADES

Amazon Redshift 根据公历解释 DECADE 或 DECADES DATEPART。例如，由于公历从第一年开始，因此第一个十年（第 1 个十年）是 0001-01-01 到 0009-12-31，而第二个十年（第 2 个十年）是 0010-01-01 到 0019-12-31。例如，十年 201 为 2000-01-01 - 2009-12-31：

```
select extract(decade from timestamp '1999-02-16 20:38:40');
date_part
-----------
200

select extract(decade from timestamp '2000-02-16 20:38:40');
date_part
-----------
201

select extract(decade from timestamp '2010-02-16 20:38:40');
date_part
-----------
202
```

MIL 或 MILS

Amazon Redshift 将 MIL 解释为开始于 #001 年的第一天并结束于 #000 年的最后一天：

```
select extract (mil from timestamp '2000-12-16 12:21:13');
date_part
-----------
2

select extract (mil from timestamp '2001-12-16 12:21:13');
date_part
-----------
3
```

哈希函数

主题

- CHECKSUM 函数
- farmFingerprint64 函数
- FUNC_SHA1 函数
- FNV_HASH 函数
- MD5 函数
- SHA 函数
- SHA1 函数
- SHA2 函数
- MURMUR3_32_HASH

哈希函数是将数值输入值转换为另一个值的数学函数。

CHECKSUM 函数

计算用于建立哈希索引的校验和值。

语法

```
CHECKSUM(expression)
```

参数

expression

输入表达式必须是 VARCHAR、INTEGER 或 DECIMAL 数据类型。

返回类型

CHECKSUM 函数返回整数。

示例

以下示例计算 COMMISSION 列的校验和值：

```
select checksum(commission)
from sales
order by salesid
limit 10;

checksum
----------
10920
1140
5250
2625
2310
5910
11820
2955
8865
975
(10 rows)
```

farmFingerprint64 函数

使用 `Fingerprint64` 函数计算输入参数的 farmhash 值。

语法

```
farmFingerprint64(expression)
```

参数

expression

输入表达式必须为 VARCHAR 或 VARBYTE 数据类型。

返回类型

farmFingerprint64 函数返回 BIGINT。

示例

以下示例返回 Amazon Redshift 的作为 VARCHAR 数据类型输入的 farmFingerprint64 值。

```
SELECT farmFingerprint64('Amazon Redshift');
```

```
  farmfingerprint64
---------------------
 8085098817162212970
```

以下示例返回 Amazon Redshift 的作为 VARBYTE 数据类型输入的 farmFingerprint64 值。

```
SELECT farmFingerprint64('Amazon Redshift'::varbyte);
```

```
  farmfingerprint64
---------------------
 8085098817162212970
```

FUNC_SHA1 函数

SHA1 函数的同义词。

请参阅 SHA1 函数。

FNV_HASH 函数

计算所有基本数据类型的 64 位 FNV-1a 非加密哈希函数。

语法

```
FNV_HASH(value [, seed])
```

参数

值

要进行哈希处理的输入值。Amazon Redshift 使用值的二进制表示形式来对输入值进行哈希处理；例如，使用 4 个字节对 INTEGER 值进行哈希处理，使用 8 个字节对 BIGINT 值进行哈希处理。此外，对 CHAR 和 VARCHAR 输入进行哈希处理不会忽略尾随空格。

种子

哈希函数的 BIGINT 种子是可选的。如果未给定，Amazon Redshift 使用默认 FNV 种子。这样可以组合多个列的哈希，而无需任何转换或联接。

返回类型

BIGINT

示例

以下示例返回数字的 FNV 哈希值、字符串“Amazon Redshift”以及两者的联接。

```
select fnv_hash(1);
        fnv_hash
----------------------
 -5968735742475085980
(1 row)
```

```
select fnv_hash('Amazon Redshift');
      fnv_hash
---------------------
 7783490368944507294
(1 row)
```

```
select fnv_hash('Amazon Redshift', fnv_hash(1));
       fnv_hash
----------------------
 -2202602717770968555
```

```
(1 row)
```

使用说明

- 要计算具有多列的表的哈希，可以计算第一列的 FNV 哈希，并将其作为种子传递给第二列的哈希。然后，它将第二列的 FNV 哈希作为种子传递给第三列的哈希。

  以下示例创建种子来对包含多列的表进行哈希处理。

```
select fnv_hash(column_3, fnv_hash(column_2, fnv_hash(column_1))) from sample_table;
```

- 同一个属性可用于计算字符串联接的哈希。

```
select fnv_hash('abcd');
       fnv_hash
---------------------
 -281581062704388899
(1 row)
```

```
select fnv_hash('cd', fnv_hash('ab'));
      fnv_hash
---------------------
 -281581062704388899
(1 row)
```

- 哈希函数使用输入的类型来确定要进行哈希处理的字节数。如有必要，使用强制转换来强制特定类型。

  以下示例使用不同类型的输入来生成不同的结果。

```
select fnv_hash(1::smallint);
      fnv_hash
--------------------
 589727492704079044
(1 row)
```

```
select fnv_hash(1);
       fnv_hash
----------------------
 -5968735742475085980
```

```
(1 row)
```

```
select fnv_hash(1::bigint);
       fnv_hash
----------------------
 -8517097267634966620
(1 row)
```

MD5 函数

使用 MD5 加密哈希函数将长度可变的字符串转换为以 128 位校验和的十六进制值的文本表示形式表示的 32 字符字符串。

语法

```
MD5(string)
```

参数

string

一个长度可变的字符串。

返回类型

MD5 函数返回以 128 位校验和的十六进制值的文本表示形式表示的 32 字符字符串。

示例

以下示例显示了字符串“Amazon Redshift”的 128 位值：

```
select md5('Amazon Redshift');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

SHA 函数

SHA1 函数的同义词。

请参阅 SHA1 函数。

SHA1 函数

SHA1 函数使用 SHA1 加密哈希函数将长度可变的字符串转换为以 160 位校验和的十六进制值的文本表示形式表示的 40 个字符的字符串。

语法

SHA1 是 SHA 函数 和 FUNC_SHA1 函数 的同义词。

```
SHA1(string)
```

参数

string

一个长度可变的字符串。

返回类型

SHA1 函数返回以 160 位校验和的十六进制值的文本表示形式表示的 40 个字符的字符串。

示例

以下示例返回单词“Amazon Redshift”的 160 位值：

```
select sha1('Amazon Redshift');
```

SHA2 函数

SHA2 函数使用 SHA2 加密哈希函数将长度可变的字符串转换为一个字符串。该字符串是具有指定位数的校验和的十六进制值的文本表示形式。

语法

```
SHA2(string, bits)
```

参数

string

一个长度可变的字符串。

integer

哈希函数中的位数。有效值为 0（与 256 相同）、224、256、384 和 512。

返回类型

SHA2 函数返回一个字符串，该字符串是校验和的十六进制值的文本表示形式；如果位数无效，此函数将返回一个空字符串。

示例

以下示例返回单词“Amazon Redshift”的 256 位值：

```
select sha2('Amazon Redshift', 256);
```

MURMUR3_32_HASH

MURMUR3_32_HASH 函数计算包括数值和字符串类型在内的所有常见数据类型的 32 位 Murmur3A 非加密哈希。

语法

```
MURMUR3_32_HASH(value [, seed])
```

参数

值

要进行哈希处理的输入值。Amazon Redshift 对输入值的二进制表示进行哈希处理。此行为类似于 FNV_HASH 函数，但值会转换为由 Apache Iceberg 32 位 Murmur3 哈希规范指定的二进制表示形式。

种子

哈希函数的 INT 种子。此参数是可选的。如果未给定，Amazon Redshift 使用默认种子 0。这样可以组合多个列的哈希，而无需任何转换或联接。

返回类型

此函数返回 INT。

示例

以下示例分别返回数字、字符串“Amazon Redshift”以及两者的联接的 Murmur3 哈希值。

```
select MURMUR3_32_HASH(1);

    MURMUR3_32_HASH
----------------------
 -5968735742475085980
(1 row)
```

```
select MURMUR3_32_HASH('Amazon Redshift');

    MURMUR3_32_HASH
----------------------
 7783490368944507294
(1 row)
```

```
select MURMUR3_32_HASH('Amazon Redshift', MURMUR3_32_HASH(1));

    MURMUR3_32_HASH
----------------------
 -2202602717770968555
(1 row)
```

使用说明

要计算具有多列的表的哈希，可以计算第一列的 Murmur3 哈希，并将其作为种子传递给第二列的哈希。然后，它将第二列的 Murmur3 哈希作为种子传递给第三列的哈希。

以下示例创建种子来对包含多列的表进行哈希处理。

```
select MURMUR3_32_HASH(column_3, MURMUR3_32_HASH(column_2, MURMUR3_32_HASH(column_1)))
 from sample_table;
```

同一个属性可用于计算字符串联接的哈希。

```
select MURMUR3_32_HASH('abcd');
```

```
   MURMUR3_32_HASH
---------------------
 -281581062704388899
(1 row)
```

```
select MURMUR3_32_HASH('cd', MURMUR3_32_HASH('ab'));

   MURMUR3_32_HASH
---------------------
 -281581062704388899
(1 row)
```

哈希函数使用输入的类型来确定要进行哈希处理的字节数。如有必要，使用强制转换来强制特定类型。

以下示例使用不同的输入类型来生成不同的结果。

```
select MURMUR3_32_HASH(1::smallint);

   MURMUR3_32_HASH
--------------------
 589727492704079044
(1 row)
```

```
select MURMUR3_32_HASH(1);

   MURMUR3_32_HASH
----------------------
 -5968735742475085980
(1 row)
```

```
select MURMUR3_32_HASH(1::bigint);

   MURMUR3_32_HASH
----------------------
 -8517097267634966620
(1 row)
```

HyperLogLog 函数

接下来，您可以找到 Amazon Redshift 支持的 HyperLogLog 函数的描述。

主题

HLL 函数

HLL 函数返回输入表达式值的 HyperLogLog 基数。HLL 函数适用于除 HLLSKETCH 数据类型之外的任何数据类型。HLL 函数将忽略 NULL 值。如果表中没有行或所有行均为 NULL，则生成的基数为 0。

语法

```
HLL (aggregate_expression)
```

参数

aggregate_expression

将值提供给聚合的任何有效表达式（如列名称）。此函数支持除 HLLSKETCH、GEOMETRY、GEOGRAPHY 和 VARBYTE 之外的任何数据类型作为输入。

返回类型

HLL 函数返回一个 BIGINT 或 INT8 值。

示例

以下示例返回表 a_table 中列 an_int 的基数。

```
CREATE TABLE a_table(an_int INT);
INSERT INTO a_table VALUES (1), (2), (3), (4);

SELECT hll(an_int) AS cardinality FROM a_table;
cardinality
-------------
4
```

HLL_CREATE_SKETCH 函数

HLL_CREATE_SKETCH 函数返回封装输入表达式值的 HLLSKETCH 数据类型。HLL_CREATE_SKETCH 函数适用于任何数据类型，并忽略 NULL 值。如果表中没有行或所有行都为 NULL，则生成的草图没有 `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}` 之类的索引值对。

语法

```
HLL_CREATE_SKETCH (aggregate_expression)
```

参数

aggregate_expression

将值提供给聚合的任何有效表达式（如列名称）。将忽略 NULL 值。此函数支持除 HLLSKETCH、GEOMETRY、GEOGRAPHY 和 VARBYTE 之外的任何数据类型作为输入。

返回类型

HLL_CREATE_SKETCH 函数返回一个 HLLSKETCH 值。

示例

以下示例返回表 `a_table` 中列 `an_int` 的 HLLSKETCH 类型。在导入、导出或打印草图时，JSON 对象用于表示稀疏的 HyperLogLog 草图。字符串表示形式（Base64 格式）用于表示密集的 HyperLogLog 草图。

```
CREATE TABLE a_table(an_int INT);
INSERT INTO a_table VALUES (1), (2), (3), (4);

SELECT hll_create_sketch(an_int) AS sketch FROM a_table;
sketch
----------------------------------------------------------------------------------------------------
{"version":1,"logm":15,"sparse":{"indices":
[20812342,20850007,22362299,47158030],"values":[1,2,1,1]}}
(1 row)
```

HLL_CARDINALITY 函数

HLL_CARDINALITY 函数返回输入 HLLSKETCH 数据类型的基数。

语法

```
HLL_CARDINALITY (hllsketch_expression)
```

参数

hllsketch_expression

计算结果为 HLLSKETCH 类型的任何有效表达式（如列名称）。输入值为 HLLSKETCH 数据类型。

返回类型

HLL_CARDINALITY 函数返回一个 BIGINT 或 INT8 值。

示例

以下示例返回表 hll_table 中列 sketch 的基数。

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2),
 (5,2), (6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

SELECT hll_cardinality(sketch) AS cardinality FROM hll_table;
cardinality
-------------
6
4
(2 rows)
```

HLL_COMBINE 函数

HLL_COMBINE 聚合函数返回一个 HLLSKETCH 数据类型，该数据类型将所有的输入 HLLSKETCH 值合并。

两个或多个 HyperLogLog 草图的组合是一个新的 HLLSKETCH，它封装了有关每个输入草图所表示的不同值的并集的信息。合并草图后，Amazon Redshift 会提取两个或多个数据集的并集的基数。有关如何合并多个草图的更多信息，请参阅示例：通过合并多个草图返回 HyperLogLog 草图。

语法

```
HLL_COMBINE (hllsketch_expression)
```

参数

hllsketch_expression

计算结果为 HLLSKETCH 类型的任何有效表达式（如列名称）。输入值为 HLLSKETCH 数据类型。

返回类型

HLL_COMBINE 函数返回一个 HLLSKETCH 类型。

示例

以下示例返回表 hll_table 中的合并 HLLSKETCH 值。

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2),
 (5,2), (6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

SELECT hll_combine(sketch) AS sketches FROM hll_table;
sketches
--------------------------------------------------------------------------------------------------------
{"version":1,"logm":15,"sparse":{"indices":
[20812342,20850007,22362299,40314817,42650774,47158030],"values":[1,2,1,3,2,1]}}
(1 row)
```

HLL_COMBINE_SKETCHES 函数

HLL_COMBINE_SKETCHES 是一个标量函数，它将两个 HLLSKETCH 值作为输入，并将它们合并为单个 HLLSKETCH 值。

两个或多个 HyperLogLog 草图的组合是一个新的 HLLSKETCH，它封装了有关每个输入草图所表示的不同值的并集的信息。

语法

```
HLL_COMBINE_SKETCHES (hllsketch_expression1, hllsketch_expression2)
```

参数

hllsketch_expression1 和 hllsketch_expression2

计算结果为 HLLSKETCH 类型的任何有效表达式（如列名称）。

返回类型

HLL_COMBINE_SKETCHES 函数返回一个 HLLSKETCH 类型。

示例

以下示例返回表 hll_table 中的合并 HLLSKETCH 值。

```
WITH tbl1(x, y)
     AS (SELECT Hll_create_sketch(1),
                Hll_create_sketch(2)
         UNION ALL
         SELECT Hll_create_sketch(3),
                Hll_create_sketch(4)
         UNION ALL
         SELECT Hll_create_sketch(5),
                Hll_create_sketch(6)
         UNION ALL
         SELECT Hll_create_sketch(7),
                Hll_create_sketch(8)),
     tbl2(x, y)
     AS (SELECT Hll_create_sketch(9),
                Hll_create_sketch(10)
         UNION ALL
         SELECT Hll_create_sketch(11),
                Hll_create_sketch(12)
         UNION ALL
         SELECT Hll_create_sketch(13),
                Hll_create_sketch(14)
         UNION ALL
         SELECT Hll_create_sketch(15),
                Hll_create_sketch(16)
```

```
        UNION ALL
        SELECT Hll_create_sketch(NULL),
               Hll_create_sketch(NULL)),
     tbl3(x, y)
     AS (SELECT *
         FROM   tbl1
         UNION ALL
         SELECT *
         FROM   tbl2)
SELECT Hll_combine_sketches(x, y)
FROM   tbl3;
```

JSON 函数

主题

- IS_VALID_JSON 函数
- IS_VALID_JSON_ARRAY 函数
- JSON_ARRAY_LENGTH 函数
- JSON_EXTRACT_ARRAY_ELEMENT_TEXT 函数
- JSON_EXTRACT_PATH_TEXT 函数
- JSON_PARSE 函数
- CAN_JSON_PARSE 函数
- JSON_SERIALIZE 函数
- JSON_SERIALIZE_TO_VARBYTE 函数

当您需要存储相对较小的一组键值对时，您可以通过以 JSON 格式存储数据来节省空间。由于 JSON 字符串可存储在单个列中，因此使用 JSON 可能比以表格格式存储数据更高效。例如，假设您有一个稀疏表，在此表中，您需要设置多个列来完整表示所有可能的属性，但大多数列值对任何给定行或任何给定列为 NULL。通过将 JSON 用于存储，您可能能够将行的数据以键值对的形式存储在单个 JSON 字符串中并删除稀疏填充的表列。

此外，您还可以轻松修改 JSON 字符串以存储其他键值对，而无需向表添加列。

我们建议慎用 JSON。若要存储较大的数据集，JSON 不是一个好的选择，因为将分散的数据存储在单个列中后，JSON 不会利用 Amazon Redshift 的列存储架构。虽然 Amazon Redshift 支持跨 CHAR 和 VARCHAR 列的 JSON 函数，但我们建议使用 SUPER 来处理 JSON 序列化格式的数据。SUPER

使用可以有效查询分层数据的后解析无 schema 表示。有关 SUPER 数据类型的更多信息，请参阅在 Amazon Redshift 中摄取和查询半结构化数据。

JSON 使用 UTF-8 编码的文本字符串，因此 JSON 字符串可存储为 CHAR 或 VARCHAR 数据类型。如果字符串包含多字节字符，则使用 VARCHAR。

JSON 字符串必须是根据以下规则正确设置格式的 JSON：

- 根级别的 JSON 可以是 JSON 对象或 JSON 数组。JSON 对象是用大括号括起的一组无序的键值对（由逗号分隔）。

  例如，`{"one":1, "two":2}`
- JSON 数组是用方括号括起的一组有序值（由逗号分隔）。

  以下是示例：`["first", {"one":1}, "second", 3, null]`
- JSON 数组使用从零开始的索引；数组中的第一个元素位于位置 0。在 JSON 键:值对中，键是用双引号括起的字符串。
- JSON 值可能为以下任一值：
  - JSON 对象
  - JSON 数组
  - 用双引号括起的字符串
  - 数字（整数和浮点）
  - 布尔值
  - null
- 空对象和空数组是有效的 JSON 值。
- JSON 字段区分大小写。
- 将忽略 JSON 结构元素之间的空格（如 `{ }, [ ]`）。

Amazon Redshift JSON 函数和 Amazon Redshift COPY 命令使用相同的方法处理 JSON 格式的数据。有关使用 JSON 的更多信息，请参阅从 JSON 格式数据执行的 COPY 操作。

IS_VALID_JSON 函数

IS_VALID_JSON 函数用于验证 JSON 字符串。如果字符串是格式正确的 JSON 字符串，则该函数返回布尔值 `true`；如果字符串格式不正确，函数将返回 `false`。要验证 JSON 数组，请使用 IS_VALID_JSON_ARRAY 函数

有关更多信息，请参阅 JSON 函数。

语法

```
IS_VALID_JSON('json_string')
```

参数

json_string

计算结果为 JSON 字符串的字符串或表达式。

返回类型

BOOLEAN

示例

要创建一个表并插入 JSON 字符串进行测试，请使用以下示例。

```
CREATE TABLE test_json(id int IDENTITY(0,1), json_strings VARCHAR);

-- Insert valid JSON strings --
INSERT INTO test_json(json_strings) VALUES
('{"a":2}'),
('{"a":{"b":{"c":1}}}'),
('{"a": [1,2,"b"]}');

-- Insert invalid JSON strings --
INSERT INTO test_json(json_strings) VALUES
('{{}}'),
('{1:"a"}'),
('[1,2,3]');
```

要验证前面示例中的字符串，请使用下面的示例。

```
SELECT id, json_strings, IS_VALID_JSON(json_strings)
FROM test_json
ORDER BY id;

+----+--------------------+---------------+
| id |    json_strings    | is_valid_json |
+----+--------------------+---------------+
```

```
|  0 | {"a":2}             | true          |
|  4 | {"a":{"b":{"c":1}}} | true          |
|  8 | {"a": [1,2,"b"]}    | true          |
| 12 | {{}}                | false         |
| 16 | {1:"a"}             | false         |
| 20 | [1,2,3]             | false         |
+----+---------------------+---------------+
```

IS_VALID_JSON_ARRAY 函数

IS_VALID_JSON_ARRAY 函数用于验证 JSON 数组。如果数组是格式正确的 JSON 数组，则该函数返回布尔值 `true`；如果数组格式不正确，函数将返回 `false`。要验证 JSON 字符串，请使用 IS_VALID_JSON 函数

有关更多信息，请参阅 JSON 函数。

语法

```
IS_VALID_JSON_ARRAY('json_array')
```

参数

json_array

计算结果为 JSON 数组的字符串或表达式。

返回类型

BOOLEAN

示例

要创建一个表并插入 JSON 字符串进行测试，请使用以下示例。

```
CREATE TABLE test_json_arrays(id int IDENTITY(0,1), json_arrays VARCHAR);

-- Insert valid JSON array strings --
INSERT INTO test_json_arrays(json_arrays)
VALUES('[]'),
('["a","b"]'),
('["a",["b",1,["c",2,3,null]]]');

-- Insert invalid JSON array strings --
```

```
INSERT INTO test_json_arrays(json_arrays)
VALUES('{"a":1}'),
('a'),
('[1,2,]');
```

要验证前面示例中的字符串，请使用下面的示例。

```
SELECT json_arrays, IS_VALID_JSON_ARRAY(json_arrays)
FROM test_json_arrays ORDER BY id;

+------------------------------+---------------------+
|         json_arrays          | is_valid_json_array |
+------------------------------+---------------------+
| []                           | true                |
| ["a","b"]                    | true                |
| ["a",["b",1,["c",2,3,null]]] | true                |
| {"a":1}                      | false               |
| a                            | false               |
| [1,2,]                       | false               |
+------------------------------+---------------------+
```

JSON_ARRAY_LENGTH 函数

JSON_ARRAY_LENGTH 函数返回 JSON 字符串的外部数组中的元素的数量。如果 null_if_invalid 参数设置为 `true` 并且 JSON 字符串无效，函数将返回 NULL 而不是返回错误。

有关更多信息，请参阅 JSON 函数。

语法

```
JSON_ARRAY_LENGTH('json_array' [, null_if_invalid ] )
```

参数

json_array

格式正确的 JSON 数组。

null_if_invalid

（可选）一个 BOOLEAN 值，指定在输入 JSON 字符串无效时是否返回 NULL，而不返回错误。要在 JSON 无效时返回 NULL，请指定 `true`（`t`）。要在 JSON 无效时返回错误，请指定 `false`（f）。默认为 `false`。

返回类型

INTEGER

示例

要返回数组中的元素数，请使用以下示例。

```
SELECT JSON_ARRAY_LENGTH('[11,12,13,{"f1":21,"f2":[25,26]},14]');

+-------------------+
| json_array_length |
+-------------------+
|                 5 |
+-------------------+
```

要因为 JSON 无效而返回错误，请使用以下示例。

```
SELECT JSON_ARRAY_LENGTH('[11,12,13,{"f1":21,"f2":[25,26]},14');

ERROR: invalid json array object [11,12,13,{"f1":21,"f2":[25,26]},14
```

要将 null_if_invalid 设置为 true，以便语句返回 NULL，而不是在 JSON 无效时返回错误，请使用以下示例。

```
SELECT JSON_ARRAY_LENGTH('[11,12,13,{"f1":21,"f2":[25,26]},14',true);

+-------------------+
| json_array_length |
+-------------------+
| NULL              |
+-------------------+
```

JSON_EXTRACT_ARRAY_ELEMENT_TEXT 函数

JSON_EXTRACT_ARRAY_ELEMENT_TEXT 函数返回 JSON 字符串的最外侧数组中的 JSON 数组元素（使用从零开始的索引）。数组中的第一个元素位于位置 0。如果索引为负或超出界限，JSON_EXTRACT_ARRAY_ELEMENT_TEXT 将返回空字符串。如果 null_if_invalid 参数设置为 `true` 并且 JSON 字符串无效，函数将返回 NULL 而不是返回错误。

有关更多信息，请参阅 JSON 函数。

语法

```
JSON_EXTRACT_ARRAY_ELEMENT_TEXT('json string', pos [, null_if_invalid ] )
```

参数

json_string

格式正确的 JSON 字符串。

pos

一个 INTEGER，表示要返回的数组元素的索引（使用从零开始的数组索引）。

null_if_invalid

（可选）一个 BOOLEAN 值，指定在输入 JSON 字符串无效时是否返回 NULL，而不返回错误。要在 JSON 无效时返回 NULL，请指定 true（t）。要在 JSON 无效时返回错误，请指定 false (f)。默认为 false。

返回类型

VARCHAR

表示 pos 引用的 JSON 数组元素的 VARCHAR 字符串。

示例

要返回位置 2 的数组元素（它是从零开始的数组索引的第三个元素），请使用以下示例：

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('[111,112,113]', 2);

+---------------------------------+
| json_extract_array_element_text |
+---------------------------------+
|                             113 |
+---------------------------------+
```

要因为 JSON 无效而返回错误，请使用以下示例。

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('["a",["b",1,["c",2,3,null,]]]',1);

ERROR: invalid json array object ["a",["b",1,["c",2,3,null,]]]
```

要将 null_if_invalid 设置为 true，以便语句返回 NULL，而不是在 JSON 无效时返回错误，请使用以下示例。

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('["a",["b",1,["c",2,3,null,]]]',1,true);

+---------------------------------+
| json_extract_array_element_text |
+---------------------------------+
| NULL                            |
+---------------------------------+
```

JSON_EXTRACT_PATH_TEXT 函数

JSON_EXTRACT_PATH_TEXT 函数返回 JSON 字符串中的一系列路径元素引用的键/值对的值。JSON 路径最深可嵌套至 5 层。路径元素区分大小写。如果 JSON 字符串中不存在路径元素，JSON_EXTRACT_PATH_TEXT 将返回 NULL。

如果 null_if_invalid 参数设置为 `true` 并且 JSON 字符串无效，函数将返回 NULL 而不是返回错误。

有关其他 JSON 函数的信息，请参阅 JSON 函数。有关使用 JSON 的更多信息，请参阅从 JSON 格式数据执行的 COPY 操作。

语法

```
JSON_EXTRACT_PATH_TEXT('json_string', 'path_elem' [,'path_elem'[, …] ]
 [, null_if_invalid ] )
```

参数

json_string

格式正确的 JSON 字符串。

path_elem

JSON 字符串中的路径元素。需要一个路径元素。可指定额外的路径元素，最深五层。

null_if_invalid

（可选）一个 BOOLEAN 值，指定在输入 JSON 字符串无效时是否返回 NULL，而不返回错误。要在 JSON 无效时返回 NULL，请指定 true（t）。要在 JSON 无效时返回错误，请指定 false (f)。默认为 false。

在 JSON 字符串中，Amazon Redshift 将 \n 识别为换行符，将 \t 识别为制表符。要加载反斜杠，请使用反斜杠 (\\ ) 对其进行转义。有关更多信息，请参阅 在 JSON 中转义字符。

返回类型

VARCHAR

表示路径元素引用的 JSON 值的 VARCHAR 字符串。

示例

要返回路径 'f4', 'f6' 的值，请使用以下示例。

```
SELECT JSON_EXTRACT_PATH_TEXT('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}','f4', 'f6');

+------------------------+
| json_extract_path_text |
+------------------------+
| star                   |
+------------------------+
```

要因为 JSON 无效而返回错误，请使用以下示例。

```
SELECT JSON_EXTRACT_PATH_TEXT('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4', 'f6');

ERROR: invalid json object {"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}
```

要将 null_if_invalid 设置为 true，以便语句在 JSON 无效返回 NULL，而不是返回错误，请使用以下示例。

```
SELECT JSON_EXTRACT_PATH_TEXT('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4',
 'f6',true);
```

```
+------------------------+
| json_extract_path_text |
+------------------------+
| NULL                   |
+------------------------+
```

要返回路径 'farm', 'barn', 'color' 的值，其中检索到的值位于第三级，请使用以下示例。为更便于阅读，此示例使用 JSON lint 工具进行格式化。

```
SELECT JSON_EXTRACT_PATH_TEXT('{
    "farm": {
        "barn": {
            "color": "red",
            "feed stocked": true
        }
    }
}', 'farm', 'barn', 'color');
+------------------------+
| json_extract_path_text |
+------------------------+
| red                    |
+------------------------+
```

要因为缺少 'color' 元素而返回 NULL，请使用以下示例。此示例使用 JSON lint 工具进行格式化。

```
SELECT JSON_EXTRACT_PATH_TEXT('{
    "farm": {
        "barn": {}
    }
}', 'farm', 'barn', 'color');

+------------------------+
| json_extract_path_text |
+------------------------+
| NULL                   |
+------------------------+
```

如果 JSON 有效，则尝试提取缺失的元素将返回 NULL。

要返回路径 'house', 'appliances', 'washing machine', 'brand' 的值，请使用以下示例。

```
SELECT JSON_EXTRACT_PATH_TEXT('{
  "house": {
    "address": {
      "street": "123 Any St.",
      "city": "Any Town",
      "state": "FL",
      "zip": "32830"
    },
    "bathroom": {
      "color": "green",
      "shower": true
    },
    "appliances": {
      "washing machine": {
        "brand": "Any Brand",
        "color": "beige"
      },
      "dryer": {
        "brand": "Any Brand",
        "color": "white"
      }
    }
  }
}', 'house', 'appliances', 'washing machine', 'brand');

+------------------------+
| json_extract_path_text |
+------------------------+
| Any Brand              |
+------------------------+
```

以下示例创建一个示例表并用 SUPER 值填充该表，然后在两行中均返回路径 'f2' 的值。

```
CREATE TABLE json_example(id INT, json_text SUPER);

INSERT INTO json_example VALUES
(1, JSON_PARSE('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}')),
(2, JSON_PARSE('{
    "farm": {
        "barn": {
            "color": "red",
            "feed stocked": true
        }
```

```
    }
}'));

SELECT * FROM json_example;
id          | json_text
------------+--------------------------------------------
1           | {"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}
2           | {"farm":{"barn":{"color":"red","feed stocked":true}}}


SELECT id, JSON_EXTRACT_PATH_TEXT(JSON_SERIALIZE(json_text), 'f2') FROM json_example;

id          | json_text
------------+--------------------------------------------
1           | {"f3":1}
2           |
```

JSON_PARSE 函数

JSON_PARSE 函数以 JSON 格式解析数据并将其转换为 SUPER 表示形式。

要使用 INSERT 或 UPDATE 命令摄取到 SUPER 数据类型，请使用 JSON_PARSE 函数。当您使用 JSON_PARSE () 将 JSON 字符串解析为 SUPER 值时，某些限制适用。有关更多信息，请参阅 解析 SUPER 的选项。

语法

```
JSON_PARSE( {json_string | binary_value} )
```

参数

json_string

以 VARBYTE 或 VARCHAR 类型返回序列化 JSON 的表达式。

binary_value

VARBYTE 类型的二进制值。

返回类型

SUPER

示例

要将 JSON 数组 [10001,10002,"abc"] 转换为 SUPER 数据类型，请使用以下示例。

```
SELECT JSON_PARSE('[10001,10002,"abc"]');

+---------------------+
|     json_parse      |
+---------------------+
| [10001,10002,"abc"] |
+---------------------+
```

为了确保函数将 JSON 数组转换为 SUPER 数据类型，请使用以下示例。有关更多信息，请参阅 JSON_TYPEOF 函数

```
SELECT JSON_TYPEOF(JSON_PARSE('[10001,10002,"abc"]'));

+-------------+
| json_typeof |
+-------------+
| array       |
+-------------+
```

CAN_JSON_PARSE 函数

CAN_JSON_PARSE 函数以 JSON 格式解析数据，如果可以使用 JSON_PARSE 函数将结果转换为 SUPER 值，则返回 `true`。

语法

```
CAN_JSON_PARSE( {json_string | binary_value} )
```

参数

json_string

以 VARBYTE 或 VARCHAR 形式返回序列化 JSON 的表达式。

binary_value

VARBYTE 类型的二进制值。

返回类型

BOOLEAN

示例

要查看是否可以将 JSON 数组 [10001,10002,"abc"] 转换为 SUPER 数据类型，请使用以下示例。

```
SELECT CAN_JSON_PARSE('[10001,10002,"abc"]');

+----------------+
| can_json_parse |
+----------------+
| true           |
+----------------+
```

JSON_SERIALIZE 函数

JSON_SERIALIZE 函数将 SUPER 表达式序列化为文本 JSON 表示形式，以遵守 RFC 8259。有关该 RFC 的更多信息，请参阅 JavaScript Object Notation (JSON) 数据交换格式。

SUPER 大小限制与数据块限制大致相同，并且 VARCHAR 限制小于 SUPER 大小限制。因此，当 JSON 格式超出系统的 varchar 限制时，JSON_SERIALIZE 函数会返回一个错误。要检查 SUPER 表达式的大小，请参阅 JSON_SIZE 函数。

语法

```
JSON_SERIALIZE(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

VARCHAR

示例

要将 SUPER 值序列化为字符串，请使用以下示例。

```
SELECT JSON_SERIALIZE(JSON_PARSE('[10001,10002,"abc"]'));

+---------------------+
|   json_serialize    |
+---------------------+
| [10001,10002,"abc"] |
+---------------------+
```

JSON_SERIALIZE_TO_VARBYTE 函数

JSON_SERIALIZE_TO_VARBYTE 函数将 SUPER 值转换为类似于 JSON_SERIALIZE() 的 JSON 字符串，但存储在 VARBYTE 值中。

语法

```
JSON_SERIALIZE_TO_VARBYTE(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

VARBYTE

示例

要序列化 SUPER 值并以 VARBYTE 格式返回结果，请使用以下示例。

```
SELECT JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'));

+----------------------------------------+
|       json_serialize_to_varbyte        |
+----------------------------------------+
| 5b31303030312c31303030322c22616263225d |
+----------------------------------------+
```

要序列化 SUPER 值并将结果强制转换为 VARCHAR 格式，请使用以下示例。有关更多信息，请参阅 CAST 函数。

```
SELECT CAST((JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'))) AS VARCHAR);

+---------------------------+
| json_serialize_to_varbyte |
+---------------------------+
| [10001,10002,"abc"]       |
+---------------------------+
```

机器学习函数。

通过使用 Amazon Redshift 机器学习（ML），您可以使用 SQL 语句训练机器学习模型，并在 SQL 查询中调用它们以进行预测。Amazon Redshift 模型可解释性包括功能重要性值，可帮助您了解训练数据中的每个属性对预测结果的贡献程度。

接下来，您可以找到对于 Amazon Redshift 支持的 SQL 机器学习函数的描述。

主题

- EXPLAIN_MODEL 函数

EXPLAIN_MODEL 函数

EXPLAIN_MODEL 函数返回一个 SUPER 数据类型，其中以 JSON 格式提供了模型可解释性报告。可解释性报告中包含有关所有模型功能的 Shapley 值的信息。

EXPLAIN_MODEL 函数目前仅支持 AUTO ON 或 AUTO OFF XGBoost 模型。

如果未提供可解释性报告，函数将返回显示模型进度的状态。这包括 `Waiting for training job to complete`、`Waiting for processing job to complete` 和 `Processing job failed`。

运行 CREATE MODEL 语句时，解释状态将变为 `Waiting for training job to complete`。当模型经过训练并发送解释请求后，解释状态变为 `Waiting for processing job to complete`。成功完成模型解释后，即可获得完整的可解释性报告。否则，状态将变为 `Processing job failed`。

运行 CREATE MODEL 语句时，可以使用可选的 `MAX_RUNTIME` 参数，以指定训练应花费的最大时间量。一旦模型创建时间达到该时间，Amazon Redshift 就会停止创建模型。如果您在创建自动驾驶模型时达到该时间限制，Amazon Redshift 将返回到该时间为止最好的模型。一旦模型训练完成，模型可解

释性就变为可用，因此，如果 MAX_RUNTIME 设置为较短的时间，可解释性报告可能不可用。训练时间各不相同，具体取决于模型复杂度、数据大小和其他因素。

语法

```
EXPLAIN_MODEL ('schema_name.model_name')
```

参数

schema_name

架构的名称。如果未指定 schema_name，则会选择当前架构。

model_name

模型的名称。schema 中的模型名称必须是唯一的。

返回类型

EXPLAIN_MODEL 函数会返回 SUPER 数据类型，如下所示。

```
{"version":"1.0","explanations":{"kernel_shap":{"label0":{"global_shap_values":
{"x0":0.05,"x1":0.10,"x2":0.30,"x3":0.15},"expected_value":0.50}}}}
```

示例

以下示例返回了解释状态 waiting for training job to complete。

```
select explain_model('customer_churn_auto_model');
                 explain_model
------------------------------------------------------------
{"explanations":"waiting for training job to complete"}
(1 row)
```

成功完成模型解释后，即可获得完整的可解释性报告，如下所示。

```
select explain_model('customer_churn_auto_model');
                                        explain_model
-------------------------------------------------------------------------------------------------
{"version":"1.0","explanations":{"kernel_shap":{"label0":{"global_shap_values":
{"x0":0.05386043365892927,"x1":0.10801289723274592,"x2":0.23227865827017378,"x3":0.067668513394
(1 row)
```

由于 EXPLAIN_MODEL 函数返回了 SUPER 数据类型，因此您可以查询可解释性报告。这样，您可以提取 global_shap_values、expected_value，或特定于功能的 Shapley 值。

以下示例提取了模型的 global_shap_values。

```
select json_table.report.explanations.kernel_shap.label0.global_shap_values from
 (select explain_model('customer_churn_auto_model') as report) as json_table;
                                                        global_shap_values
-----------------------------------------------------------------------------------------------------------
{"state":0.10983770427197151,"account_length":0.1772441398408543,"area_code":0.0862682396863959
(1 row)
```

以下示例提取了功能 x0 的 global_shap_values 。

```
select json_table.report.explanations.kernel_shap.label0.global_shap_values.x0 from
 (select explain_model('customer_churn_auto_model') as report) as json_table;
          x0
------------------------
  0.05386043365892927
(1 row)
```

如果模型是在特定架构中创建的，且您有权访问所创建的模型，则可以查询模型解释，如下所示。

```
-- Check the current schema
SHOW search_path;
   search_path
-----------------
  $user, public
(1 row)
-- If you have the privilege to access the model explanation
-- in `test_schema`
SELECT explain_model('test_schema.test_model_name');
                       explain_model
-----------------------------------------------------------
{"explanations":"waiting for training job to complete"}
(1 row)
```

数学函数

主题

- 数学运算符符号

- ABS 函数
- ACOS 函数
- ASIN 函数
- ATAN 函数
- ATAN2 函数
- CBRT 函数
- CEILING（或 CEIL）函数
- COS 函数
- COT 函数
- DEGREES 函数
- DEXP 函数
- DLOG1 函数
- DLOG10 函数
- EXP 函数
- FLOOR 函数
- LN 函数
- LOG 函数
- MOD 函数
- PI 函数
- POWER 函数
- RADIANS 函数
- RANDOM 函数
- ROUND 函数
- SIN 函数
- SIGN 函数
- SQRT 函数
- TAN 函数
- TRUNC 函数

本部分描述 Amazon Redshift 中支持的数学运算符和函数。

数学运算符符号

下表列出了支持的数学运算符。

支持的运算符

操作符	描述	示例	结果
+	加	2 + 3	5
-	减	2 - 3	–1
*	乘	2 * 3	6
/	除	4 / 2	2
%	取模	5 % 4	1
^	幂	2.0 ^ 3.0	8
|/	平方根	| / 25.0	5
||/	立方根	|| / 27.0	3
@	绝对值	@ -5.0	5
<<	按位左移	1 << 4	16
>>	按位右移	8 >> 2	2
&	按位和	8 & 2	0

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要为给定的交易计算支付的佣金加 2.00 美元手续费，请使用以下示例。

```
SELECT
    commission,
    (commission + 2.00) AS comm
```

```
FROM
    sales
WHERE
    salesid = 10000;

+------------+-------+
| commission | comm  |
+------------+-------+
|      28.05 | 30.05 |
+------------+-------+
```

要为给定的交易计算销售价格的 20%，请使用以下示例。

```
SELECT pricepaid, (pricepaid * .20) as twentypct
FROM sales
WHERE salesid=10000;

+-----------+-----------+
| pricepaid | twentypct |
+-----------+-----------+
|       187 |      37.4 |
+-----------+-----------+
```

要根据持续增长模式预测票的销售量，请使用以下示例。在此示例中，子查询将返回 2008 年销售的票数。在此后 10 年，该结果将以 5% 的连续增长率呈指数增长。

```
SELECT (SELECT SUM(qtysold) FROM sales, date
WHERE sales.dateid=date.dateid AND year=2008)^((5::float/100)*10) AS qty10years;

+------------------+
|    qty10years    |
+------------------+
| 587.664019657491 |
+------------------+
```

要查找日期 ID 大于或等于 2000 的销售的总支付价格和总佣金，请使用以下示例。然后将总支付价格减去总佣金。

```
SELECT SUM(pricepaid) AS sum_price, dateid,
SUM(commission) AS sum_comm, (SUM(pricepaid) - SUM(commission)) AS value
FROM sales
WHERE dateid >= 2000
```

```
GROUP BY dateid
ORDER BY dateid
LIMIT 10;

+-----------+--------+----------+-----------+
| sum_price | dateid | sum_comm |   value   |
+-----------+--------+----------+-----------+
|    305885 |   2000 | 45882.75 | 260002.25 |
|    316037 |   2001 | 47405.55 | 268631.45 |
|    358571 |   2002 | 53785.65 | 304785.35 |
|    366033 |   2003 | 54904.95 | 311128.05 |
|    307592 |   2004 |  46138.8 |  261453.2 |
|    333484 |   2005 |  50022.6 |  283461.4 |
|    317670 |   2006 |  47650.5 |  270019.5 |
|    351031 |   2007 | 52654.65 | 298376.35 |
|    313359 |   2008 | 47003.85 | 266355.15 |
|    323675 |   2009 | 48551.25 | 275123.75 |
+-----------+--------+----------+-----------+
```

ABS 函数

ABS 用于计算数字的绝对值，该数字可以是文本或计算结果为数字的表达式。

语法

```
ABS(number)
```

参数

number

数字或计算结果为数字的表达式。它可以是 SMALLINT、INTEGER、BIGINT、DECIMAL、FLOAT4、FLOAT8 或 SUPER 类型。

返回类型

ABS 返回与其参数相同的数据类型。

示例

要计算 -38 的绝对值，请使用以下示例：

```
SELECT ABS(-38);

+-----+
| abs |
+-----+
|  38 |
+-----+
```

要计算 (14-76) 的绝对值，请使用以下示例：

```
SELECT ABS(14-76);

+-----+
| abs |
+-----+
|  62 |
+-----+
```

ACOS 函数

ACOS 是返回数字的反余弦的三角函数。返回值采用弧度形式且介于 0 和 PI 之间。

语法

```
ACOS(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 -1 的反余弦，请使用以下示例。

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

要将 .5 的反余弦转换为等效的度数，请使用以下示例。

```
SELECT (ACOS(.5) * 180/(SELECT PI())) AS degrees;

+-------------------+
|      degrees      |
+-------------------+
| 60.00000000000001 |
+-------------------+
```

ASIN 函数

ASIN 是返回数字的反正弦的三角函数。返回值采用弧度形式且介于 PI/2 和 -PI/2 之间。

语法

```
ASIN(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 1 的反正弦，请使用以下示例。

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

要将 .5 的反正弦转换为等效的度数，请使用以下示例。

```
SELECT (ASIN(.5) * 180/(SELECT PI())) AS degrees;

+--------------------+
|      degrees       |
+--------------------+
| 30.000000000000004 |
+--------------------+
```

ATAN 函数

ATAN 是返回数字的反正切的三角函数。返回值采用弧度形式且介于 -PI 和 PI 之间。

语法

```
ATAN(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 1 的反正切并将其乘以 4，请使用以下示例。

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

要将 1 的反正切转换为等效的度数，请使用以下示例。

```
SELECT (ATAN(1) * 180/(SELECT PI())) AS degrees;

+---------+
| degrees |
+---------+
|      45 |
+---------+
```

ATAN2 函数

ATAN2 是一个三角函数，它返回两个数值相除的结果的反正切。返回值采用弧度形式且介于 PI/2 和 -PI/2 之间。

语法

```
ATAN2(number1, number2)
```

参数

number1

DOUBLE PRECISION 数值。

number2

DOUBLE PRECISION 数值。

返回类型

DOUBLE PRECISION

示例

要返回 2/2 的反正切并将其乘以 4，请使用以下示例。

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

要将 1/0（计算结果为 0）的反正切转换为等效的度数，请使用以下示例。

```
SELECT (ATAN2(1,0) * 180/(SELECT PI())) AS degrees;

+---------+
| degrees |
+---------+
|      90 |
+---------+
```

CBRT 函数

CBRT 函数是计算给定数值的立方根的数学函数。

语法

```
CBRT(number)
```

参数

CBRT 以 DOUBLE PRECISION 数值作为参数：

返回类型

DOUBLE PRECISION

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要计算为给定交易支付的佣金的立方根，请使用以下示例。

```
SELECT CBRT(commission) FROM sales WHERE salesid=10000;

+--------------------+
|        cbrt        |
+--------------------+
| 3.0383953904884344 |
+--------------------+
```

CEILING（或 CEIL）函数

CEILING 或 CEIL 函数用于将数字向上舍入到下一个整数。（FLOOR 函数将数字向下舍入到下一个整数）

语法

```
{CEIL | CEILING}(number)
```

参数

number

数字或计算结果为数字的表达式。它可以是 SMALLINT、INTEGER、BIGINT、DECIMAL、FLOAT4、FLOAT8 或 SUPER 类型。

返回类型

CEILING 和 CEIL 返回与其参数相同的数据类型。

当输入为 SUPER 类型时，输出将保留与输入相同的动态类型，而静态类型仍为 SUPER 类型。当 SUPER 的动态类型不是数值时，Amazon Redshift 将返回 null。

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要计算为给定的销售交易支付的佣金的上限，请使用以下示例。

```
SELECT CEILING(commission) FROM sales
```

```
WHERE salesid=10000;

+---------+
| ceiling |
+---------+
|      29 |
+---------+
```

COS 函数

COS 是返回数字的余弦的三角函数。返回值采用弧度形式且介于 -1 和 1 之间（含）。

语法

```
COS(double_precision)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

COS 函数返回 DOUBLE PRECISION 数值。

示例

要返回 0 的余弦，请使用以下示例。

```
SELECT COS(0);

+-----+
| cos |
+-----+
|   1 |
+-----+
```

要返回 pi 的余弦，请使用以下示例。

```
SELECT COS(PI());
```

```
+-----+
| cos |
+-----+
|  -1 |
+-----+
```

COT 函数

COT 是返回数字的余切的三角函数。输入参数必须为非零。

语法

```
COT(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 1 的余切，请使用以下示例。

```
SELECT COT(1);

+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

DEGREES 函数

将用弧度表示的角度转换用度表示。

语法

```
DEGREES(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 0.5 弧度的等效度数，请使用以下示例。

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
+-------------------+
```

要将 PI 弧度转换为度数，请使用以下示例。

```
SELECT DEGREES(pi());

+---------+
| degrees |
+---------+
|     180 |
+---------+
```

DEXP 函数

DEXP 函数返回双精度数的采用科学表示法的指数值。DEXP 函数和 EXP 函数的唯一区别在于 DEXP 的参数必须为 DOUBLE PRECISION。

语法

```
DEXP(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

使用 DEXP 函数根据持续增长模式预测票的销售量。在此示例中，子查询将返回 2008 年销售的票数。该结果将乘以 DEXP 函数的结果（指定了在接下来 10 年保持 7% 的持续增长率）。

```
SELECT (SELECT SUM(qtysold)
FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * DEXP((7::FLOAT/100)*10) qty2010;

+-------------------+
|      qty2010      |
+-------------------+
| 695447.4837722216 |
+-------------------+
```

DLOG1 函数

DLOG1 函数返回输入参数的自然对数。LN 函数的同义词。

DLOG10 函数

DLOG10 返回输入参数的以 10 为底的对数。

LOG 函数的同义词。

语法

```
DLOG10(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回数值 100 的以 10 为底的对数，请使用以下示例。

```
SELECT DLOG10(100);

+--------+
| dlog10 |
+--------+
|      2 |
+--------+
```

EXP 函数

EXP 函数实施数值表达式的指数函数，即以自然对数 e 为底数，对表达式求次方。EXP 函数是 LN 函数的反函数。

语法

```
EXP(expression)
```

参数

expression

表达式必须为 INTEGER、DECIMAL 或 DOUBLE PRECISION 数据类型。

返回类型

DOUBLE PRECISION

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

使用 EXP 函数根据持续增长模式预测票的销售量。在此示例中，子查询将返回 2008 年销售的票数。该结果将乘以 EXP 函数的结果（指定了在接下来 10 年保持 7% 的持续增长率）。

```
SELECT (SELECT SUM(qtysold)
FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * EXP((7::FLOAT/100)*10) qty2018;

+-------------------+
|      qty2018      |
+-------------------+
| 695447.4837722216 |
+-------------------+
```

FLOOR 函数

FLOOR 函数将数字向下舍入到下一个整数。

语法

```
FLOOR(number)
```

参数

number

数字或计算结果为数字的表达式。它可以是 SMALLINT、INTEGER、BIGINT、DECIMAL、FLOAT4、FLOAT8 或 SUPER 类型。

返回类型

FLOOR 返回与其参数相同的数据类型。

当输入为 SUPER 类型时，输出将保留与输入相同的动态类型，而静态类型仍保留 SUPER 类型。当 SUPER 的动态类型不是数值时，Amazon Redshift 将返回 NULL。

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要显示在使用 FLOOR 函数之前和之后为给定的销售交易支付的佣金值，请使用以下示例。

```
SELECT commission
FROM sales
WHERE salesid=10000;

+------------+
| commission |
+------------+
|      28.05 |
+------------+

SELECT FLOOR(commission)
FROM sales
WHERE salesid=10000;

+-------+
| floor |
+-------+
|    28 |
+-------+
```

LN 函数

返回输入参数的自然对数。

DLOG1 函数的同义词。

语法

```
LN(expression)
```

参数

expression

对其执行函数的目标列或表达式。

> **Note**
>
> 如果表达式引用了 Amazon Redshift 用户创建的表或者引用了 Amazon Redshift STL 或 STV 系统表，此函数将对某些数据类型返回错误。

具有以下数据类型的表达式在引用了用户创建的表或系统表时将产生错误。具有这些数据类型的表达式专用于在领导节点上运行：

- BOOLEAN
- CHAR
- DATE
- DECIMAL 或 NUMERIC
- TIMESTAMP
- VARCHAR

具有以下数据类型的表达式可在用户创建的表以及 STL 或 STV 系统表上成功运行：

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

返回类型

LN 函数返回与输入 expression 相同的类型。

示例

要返回数值 2.718281828 的自然对数或以 e 为底的对数，请使用以下示例。

```
SELECT LN(2.718281828);

+--------------------+
|         ln         |
+--------------------+
| 0.9999999998311267 |
+--------------------+
```

请注意，结果约等于 1。

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要返回 USERS 表 userid 列中的值的自然对数，请使用以下示例。

```
SELECT username, LN(userid) FROM users ORDER BY userid LIMIT 10;

+----------+--------------------+
| username |         ln         |
+----------+--------------------+
| JSG99FHE |                  0 |
| PGL08LJI | 0.6931471805599453 |
| IFT66TXU | 1.0986122886681098 |
| XDZ38RDD | 1.3862943611198906 |
| AEB55QTM | 1.6094379124341003 |
| NDQ15VBM |  1.791759469228055 |
| OWY35QYB | 1.9459101490553132 |
| AZG78YIP | 2.0794415416798357 |
| MSD36KVR | 2.1972245773362196 |
| WKW41AIW |  2.302585092994046 |
+----------+--------------------+
```

LOG 函数

返回数值的对数。

如果您使用这个函数来计算以 10 为底的对数，则也可以使用 DLOG10 函数。

语法

```
LOG([base, ]argument)
```

参数

base

（可选）对数函数的底。此数值必须为正数且不能等于 1。如果省略此参数，Amazon Redshift 将计算 argument 的以 10 为底的对数。

argument

对数函数的参数。此数值必须为正数。如果 argument 值为 1，则函数返回 0。

返回类型

LOG 函数返回 DOUBLE PRECISION 数值。

示例

要查找 100 的以 2 为底的对数，请使用以下示例。

```
SELECT LOG(2, 100);
+-------------------+
|        log        |
+-------------------+
| 6.643856189774725 |
+-------------------+
```

要查找 100 的以 10 为底的对数，请使用以下示例。请注意，如果您省略底参数，则 Amazon Redshift 假设底为 10。

```
SELECT LOG(100);

+-----+
| log |
+-----+
|   2 |
+-----+
```

MOD 函数

返回两个数字的余数，也称为取模 运算。将第一个参数除以第二个参数来计算结果。

语法

```
MOD(number1, number2)
```

参数

number1

第一个输入参数是 INTEGER、SMALLINT、BIGINT 或 DECIMAL 数值。如果任一参数是 DECIMAL 类型，则另一参数也必须是 DECIMAL 类型。如果任一参数是 INTEGER，则另一参数可以是 INTEGER、SMALLINT 或 BIGINT。两个参数也可以都是 SMALLINT 或 BIGINT，但如果一个参数是 BIGINT，则另一个参数不能是 SMALLINT。

number2

第二个参数是 INTEGER、SMALLINT、BIGINT 或 DECIMAL 数值。相同的数据类型规则与 number1 一样适用于 number2。

返回类型

如果两个参数属于相同的类型，MOD 函数的返回类型是与输入参数相同的数值类型。但是，如果任一输入参数是 INTEGER，返回类型也将是 INTEGER。有效的返回类型为 DECIMAL、INT、SMALLINT 和 BIGINT。

使用说明

您可以使用 % 作为取模运算符。

示例

要返回一个数值除以另一个数值后的余数，请使用以下示例。

```
SELECT MOD(10, 4);

+-----+
| mod |
+-----+
|   2 |
+-----+
```

要在使用 MOD 函数时返回 DECIMAL 结果，请使用以下示例。

```
SELECT MOD(10.5, 4);

+-----+
| mod |
+-----+
| 2.5 |
+-----+
```

要在运行 MOD 函数之前强制转换数值，请使用以下示例。有关更多信息，请参阅 CAST 函数。

```
SELECT MOD(CAST(16.4 AS INTEGER), 5);
```

```
+-----+
| mod |
+-----+
|   1 |
+-----+
```

要通过将第一个参数除以 2 来检查该参数是否为偶数，请使用以下示例。

```
SELECT mod(5,2) = 0 AS is_even;

+---------+
| is_even |
+---------+
| false   |
+---------+
```

要使用 % 作为取模运算符，请使用以下示例。

```
SELECT 11 % 4 as remainder;

 +-----------+
| remainder |
+-----------+
|         3 |
+-----------+
```

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要返回 CATEGORY 表中奇数类别的信息，请使用以下示例。

```
SELECT catid, catname
FROM category
WHERE MOD(catid,2)=1
ORDER BY 1,2;

+-------+-----------+
| catid |  catname  |
+-------+-----------+
|     1 | MLB       |
|     3 | NFL       |
|     5 | MLS       |
|     7 | Plays     |
```

```
|     9 | Pop       |
|    11 | Classical |
+-------+-----------+
```

PI 函数

PI 函数返回 14 个小数位的 pi 值。

语法

```
PI()
```

返回类型

DOUBLE PRECISION

示例

要返回 pi 的值，请使用以下示例。

```
SELECT PI();

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

POWER 函数

POWER 函数是让一个数值表达式自乘到另一个数值表达式的幂的指数函数。例如，2 的三次幂的计算公式为 POWER(2,3)，结果为 8。

语法

```
{POW | POWER}(expression1, expression2)
```

参数

expression1

要自乘的数值表达式。必须是 INTEGER、DECIMAL 或 FLOAT 数据类型。

expression2

让 expression1 自乘到的幂。必须是 INTEGER、DECIMAL 或 FLOAT 数据类型。

返回类型

DOUBLE PRECISION

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

在以下示例中，POWER 函数用于根据 2008 年销售的票的数量（子查询的结果）预测将来 10 年的票销售情况。在此示例中，增长率设置为每年 7%。

```
SELECT (SELECT SUM(qtysold) FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * POW((1+7::FLOAT/100),10) qty2010;

+-------------------+
|      qty2010      |
+-------------------+
| 679353.7540885945 |
+-------------------+
```

以下示例是上一个示例的变体，其增长率为每年 7%，但间隔设置为月（10 年中的 120 个月）。

```
SELECT (SELECT SUM(qtysold) FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * POW((1+7::FLOAT/100/12),120) qty2010;

+-----------------+
|     qty2010     |
+-----------------+
| 694034.54678046 |
+-----------------+
```

RADIANS 函数

RADIANS 函数将用度表示的角度转换为用弧度表示。

语法

```
RADIANS(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 180 度的等效弧度，请使用以下示例。

```
SELECT RADIANS(180);

+-------------------+
|      radians      |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

RANDOM 函数

RANDOM 函数生成介于 0.0（含）和 1.0（不含）之间的随机值。

语法

```
RANDOM()
```

返回类型

DOUBLE PRECISION

使用说明

在使用 SET 命令设置种子值后调用 RANDOM 以使 RANDOM 生成可预测序列中的数。

示例

要计算 0 到 99 之间的随机值，请使用以下示例。如果随机数为 0 - 1，此查询将生成 0 - 100 的随机数。

```
SELECT CAST(RANDOM() * 100 AS INT);

+------+
| int4 |
+------+
|   59 |
+------+
```

此示例使用 SET 命令设置一个 SEED 值，以使 RANDOM 生成可预测的数字序列。

要返回三个 RANDOM 整数而不设置 SEED 值，请使用以下示例。

```
SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|    6 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   68 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   56 |
+------+
```

要将 SEED 值设置为 .25，然后返回三个 RANDOM 数值，请使用以下示例。

```
SET SEED TO .25;
SELECT CAST(RANDOM() * 100 AS INT);
```

```
+------+
| int4 |
+------+
|   21 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   79 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   12 |
+------+
```

要将 SEED 值重置为 .25，并验证 RANDOM 是否返回与前三个调用相同的结果，请使用以下示例。

```
SET SEED TO .25;
SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   21 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   79 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   12 |
+------+
```

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要从 SALES 表中检索 10 个项目的统一随机样本，请使用以下示例。

```
SELECT *
FROM sales
ORDER BY RANDOM()
LIMIT 10;

+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
|   45422 |  51114 |     5983 |   24482 |    4369 |   2118 |       1 |       195 |
 29.25 | 2008-10-19 05:20:07 |
|   42481 |  47638 |     4573 |    6198 |    6479 |   1987 |       4 |      1140 |
   171 | 2008-06-10 09:39:19 |
|   31494 |  34759 |    18895 |    4719 |    7753 |   2090 |       4 |      1024 |
 153.6 | 2008-09-21 03:44:26 |
|  119388 | 136685 |    21815 |   41905 |    2071 |   1884 |       1 |       359 |
 53.85 | 2008-02-27 10:43:10 |
|  166990 | 225037 |    18529 |    7628 |     746 |   2113 |       1 |      2009 |
 301.35 | 2008-10-14 10:07:44 |
|   11146 |  12096 |    42685 |    6619 |    1876 |   2123 |       1 |        29 |
  4.35 | 2008-10-24 06:23:54 |
|  148537 | 172056 |    15102 |   11787 |    6122 |   1923 |       2 |       480 |
    72 | 2008-04-07 03:58:23 |
|   68945 |  78387 |     7359 |   18323 |    6636 |   1910 |       1 |       457 |
 68.55 | 2008-03-25 08:31:03 |
|   52796 |  59576 |     9909 |   15102 |    7958 |   1951 |       1 |       479 |
 71.85 | 2008-05-05 02:25:08 |
|   90684 | 103522 |    38052 |   21549 |    7384 |   2117 |       1 |       313 |
 46.95 | 2008-10-18 05:43:11 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

要检索 10 个项目的随机样本，但选择与其价格成比例的项目，请使用以下示例。例如，价格是另一个项目的两倍的项目在查询结果中出现的可能性是两倍。

```
SELECT *
FROM sales
```

```
ORDER BY -LOG(RANDOM()) / pricepaid
LIMIT 10;

+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
|  158340 | 208208 |    17082 |   42018 |    1211 |   2160 |       4 |      6852 |
 1027.8 | 2008-11-30 12:21:43 |
|   53250 |  60069 |    12644 |    7066 |    7942 |   1838 |       4 |      1528 |
 229.2 | 2008-01-12 11:24:56 |
|   22929 |  24938 |    47314 |    6503 |     179 |   2000 |       3 |       741 |
 111.15 | 2008-06-23 08:04:50 |
|  164980 | 221181 |     1949 |   19670 |    1471 |   1906 |       1 |      1330 |
 199.5 | 2008-03-21 07:59:51 |
|  159641 | 211179 |    44897 |   16652 |    7458 |   2128 |       1 |      1019 |
 152.85 | 2008-10-29 02:02:15 |
|   73143 |  83439 |     5716 |    5727 |    7314 |   1903 |       1 |       248 |
  37.2 | 2008-03-18 11:07:42 |
|   84778 |  96749 |    46608 |   32980 |    3883 |   1999 |       2 |       958 |
 143.7 | 2008-06-22 12:13:31 |
|  171096 | 232929 |    43683 |    8536 |    8353 |   1870 |       1 |       929 |
 139.35 | 2008-02-13 01:36:36 |
|   74212 |  84697 |    39809 |   15569 |    5525 |   2105 |       2 |       896 |
 134.4 | 2008-10-06 11:47:50 |
|  158011 | 207556 |    25399 |   16881 |     232 |   2088 |       2 |      2526 |
 378.9 | 2008-09-19 06:00:26 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

ROUND 函数

ROUND 函数将数字舍入到最近的整数或小数。

ROUND 函数可以选择性地以 INTEGER 形式包含另一个参数，以指示在任意方向舍入到的小数位数。当您不提供第二个参数时，函数会舍入到最接近的整数。指定第二个参数 integer 时，函数将舍入为最接近的数值，其中精度为 integer 个小数位。

语法

```
ROUND(number [ , integer ] )
```

参数

number

数字或计算结果为数字的表达式。它可以是 DECIMAL、FLOAT8 或 SUPER 类型。Amazon Redshift 可以隐式转换其他数值数据类型。

integer

（可选）一个 INTEGER，指示以任意方向四舍五入的小数位数。此参数不支持 SUPER 数据类型。

返回类型

ROUND 返回与输入 number 相同的数值数据类型。

当输入为 SUPER 类型时，输出将保留与输入相同的动态类型，而静态类型仍保留 SUPER 类型。当 SUPER 的动态类型不是数值时，Amazon Redshift 将返回 NULL。

示例

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要将为给定交易支付的佣金舍入到最近的整数，请使用以下示例。

```
SELECT commission, ROUND(commission)
FROM sales WHERE salesid=10000;

+------------+-------+
| commission | round |
+------------+-------+
|      28.05 |    28 |
+------------+-------+
```

要将为给定交易支付的佣金舍入到第一个小数位，请使用以下示例。

```
SELECT commission, ROUND(commission, 1)
FROM sales WHERE salesid=10000;

+------------+-------+
| commission | round |
+------------+-------+
|      28.05 |  28.1 |
```

```
+------------+-------+
```

要以与上一个示例相反的方向扩展精度，请使用以下示例。

```
SELECT commission, ROUND(commission, -1)
FROM sales WHERE salesid=10000;

+------------+-------+
| commission | round |
+------------+-------+
|      28.05 |    30 |
+------------+-------+
```

SIN 函数

SIN 是返回数字的正弦的三角函数。返回值介于 -1 与 1 之间。

语法

```
SIN(number)
```

参数

number

以弧度表示的 DOUBLE PRECISION 数值。

返回类型

DOUBLE PRECISION

示例

要返回 -PI 的正弦，请使用以下示例。

```
SELECT SIN(-PI());

+-------------------------+
|           sin           |
+-------------------------+
```

```
| -0.00000000000000012246 |
+-------------------------+
```

SIGN 函数

SIGN 函数返回数字的符号（正或负）。如果参数为正，则 SIGN 函数的结果为 1；如果参数为负，则结果为 -1；或者，如果参数为 0，则结果为 0。

语法

```
SIGN(number)
```

参数

number

数字或计算结果为数字的表达式。它可以是 DECIMAL、FLOAT8、或 SUPER 类型。Amazon Redshift 可根据隐式转换规则转换其他数据类型。

返回类型

SIGN 返回与输入参数相同的数值数据类型。如果输入为 DECIMAL，则输出为 DECIMAL(1,0)。

当输入为 SUPER 类型时，输出将保留与输入相同的动态类型，而静态类型仍保留 SUPER 类型。当 SUPER 的动态类型不是数值时，Amazon Redshift 将返回 NULL。

示例

以下示例显示，由于输入是 DOUBLE PRECISION，表 t2 中的列 d 将 DOUBLE PRECISION 作为其类型；而由于输入是 NUMERIC，表 t2 中的列 n 将 NUMERIC(1,0) 作为输出。

```
CREATE TABLE t1(d DOUBLE PRECISION, n NUMERIC(12, 2));
INSERT INTO t1 VALUES (4.25, 4.25), (-4.25, -4.25);
CREATE TABLE t2 AS SELECT SIGN(d) AS d, SIGN(n) AS n FROM t1;
SELECT table_name, column_name, data_type FROM SVV_REDSHIFT_COLUMNS WHERE
 table_name='t1' OR table_name='t2';

+------------+-------------+----------------------+
| table_name | column_name |       data_type      |
+------------+-------------+----------------------+
| t1         | d           | double precision     |
```

```
| t1         | n           | numeric(12,2)         |
| t2         | d           | double precision      |
| t2         | n           | numeric(1,0)          |
| t1         | col1        | character varying(20) |
+------------+-------------+-----------------------+
```

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要从 SALES 表中确定为给定交易支付的佣金的符号，请使用以下示例。

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

SQRT 函数

SQRT 函数返回 NUMERIC 值的平方根。平方根是一个乘以自身以得到给定值的数字。

语法

```
SQRT(expression)
```

参数

expression

表达式必须具有 INTEGER、DECIMAL 或 FLOAT 数据类型，或隐式转换为这些数据类型的数据类型。expression 可以包含函数。

返回类型

DOUBLE PRECISION

示例

要返回 16 的平方根，请使用以下示例。

```
SELECT SQRT(16);

+------+
| sqrt |
+------+
|    4 |
+------+
```

要使用隐式类型转换返回字符串 16 的平方根，请使用以下示例。

```
SELECT SQRT('16');

+------+
| sqrt |
+------+
|    4 |
+------+
```

要在使用 ROUND 函数后返回 16.4 的平方根，请使用以下示例。

```
SELECT SQRT(ROUND(16.4));

+------+
| sqrt |
+------+
|    4 |
+------+
```

要返回给定圆面积时的半径长度，请使用以下示例。例如，当给定以平方英寸为单位的面积时，它以英寸为单位计算半径。示例中的面积为 20。

```
SELECT SQRT(20/PI()) AS radius;

+--------------------+
|      radius        |
+--------------------+
| 2.5231325220201604 |
+--------------------+
```

以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要返回 SALES 表中 COMMISSION 值的平方根，请使用以下示例。COMMISSION 列是 DECIMAL 列。此示例说明如何在具有更复杂条件逻辑的查询中使用该函数。

```
SELECT SQRT(commission)
FROM sales WHERE salesid < 10 ORDER BY salesid;

+--------------------+
|        sqrt        |
+--------------------+
| 10.449880382090505 |
| 3.3763886032268267 |
|  7.245688373094719 |
|  5.123475382979799 |
|  4.806245936279167 |
|  7.687652437513028 |
| 10.871982339941507 |
| 5.4359911699707535 |
|   9.41541289588513 |
+--------------------+
```

要返回同一组 COMMISSION 值的平方根的舍入值，请使用以下示例。

```
SELECT ROUND(SQRT(commission))
FROM sales WHERE salesid < 10 ORDER BY salesid;

+-------+
| round |
+-------+
|    10 |
|     3 |
|     7 |
|     5 |
|     5 |
|     8 |
|    11 |
|     5 |
|     9 |
+-------+
```

TAN 函数

TAN 是返回数字的正切的三角函数。输入参数是一个数值（用弧度表示）。

语法

```
TAN(number)
```

参数

number

DOUBLE PRECISION 数值。

返回类型

DOUBLE PRECISION

示例

要返回 0 的正切，请使用以下示例。

```
SELECT TAN(0);

+-----+
| tan |
+-----+
|   0 |
+-----+
```

TRUNC 函数

TRUNC 函数将数字截断为前一个整数或小数。

TRUNC 函数可以选择性地以 INTEGER 形式包含另一个参数，以指示在任意方向舍入到的小数位数。当您不提供第二个参数时，函数会舍入到最接近的整数。指定第二个参数 integer 时，函数将舍入为最接近的数值，其中精度为 integer 个小数位。

这个函数也可以截断 TIMESTAMP 并返回 DATE。有关更多信息，请参阅 TRUNC 函数。

语法

```
TRUNC(number [ , integer ])
```

参数

number

数值或计算结果为数值的表达式。它可以是 DECIMAL、FLOAT8 或 SUPER 类型。Amazon Redshift 可根据隐式转换规则转换其他数据类型。

integer

（可选）一个 INTEGER，指示精度在任意方向的小数位数。如果未提供 integer，数值将作为整数截断；如果指定了 integer，数值将截断到指定的小数位。SUPER 数据类型不支持此功能。

返回类型

TRUNC 返回与输入 number 相同的数据类型。

当输入为 SUPER 类型时，输出将保留与输入相同的动态类型，而静态类型仍保留 SUPER 类型。当 SUPER 的动态类型不是数值时，Amazon Redshift 将返回 NULL。

示例

以下某些示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

要截断为给定的销售交易支付的佣金，请使用以下示例。

```
SELECT commission, TRUNC(commission)
FROM sales WHERE salesid=784;

+------------+-------+
| commission | trunc |
+------------+-------+
|     111.15 |   111 |
+------------+-------+
```

要将同一佣金值截断到第一个小数位，请使用以下示例。

```
SELECT commission, TRUNC(commission,1)
FROM sales WHERE salesid=784;

+------------+-------+
| commission | trunc |
+------------+-------+
|     111.15 | 111.1 |
```

```
+------------+-------+
```

要为第二个参数使用负值截断佣金，请使用以下示例。请注意，111.15 向下四舍五入为 110。

```
SELECT commission, TRUNC(commission,-1)
FROM sales WHERE salesid=784;

+------------+-------+
| commission | trunc |
+------------+-------+
|     111.15 |   110 |
+------------+-------+
```

对象函数

以下是 SQL 对象函数，Amazon Redshift 支持这些函数以创建 SUPER 类型对象：

主题

- LOWER_ATTRIBUTE_NAMES 函数
- OBJECT 函数
- OBJECT_TRANSFORM 函数
- UPPER_ATTRIBUTE_NAMES 函数

LOWER_ATTRIBUTE_NAMES 函数

使用与 LOWER 函数相同的大小写转换例程，将 SUPER 值中的所有适用属性名称转换为小写。LOWER_ATTRIBUTE_NAMES 支持 UTF-8 多字节字符，每个字符最多可以有 4 个字节。

要将 SUPER 属性名称转换为大写，请使用 UPPER_ATTRIBUTE_NAMES 函数。

语法

```
LOWER_ATTRIBUTE_NAMES(super_expression)
```

参数

super_expression

SUPER 表达式。

返回类型

SUPER

使用说明

在 Amazon Redshift 中，列标识符传统上不区分大小写且转换为小写。如果您从区分大小写的数据格式（例如 JSON）中提取数据，则数据可能包含大小写混合的属性名称。

考虑以下示例。

```
CREATE TABLE t1 (s) AS SELECT JSON_PARSE('{"AttributeName": "Value"}');


SELECT s.AttributeName FROM t1;

attributename
-------------
NULL


SELECT s."AttributeName" FROM t1;

attributename
-------------
NULL
```

Amazon Redshift 对这两个查询都返回 NULL。要查询 `AttributeName`，请使用 LOWER_ATTRIBUTE_NAMES 将数据的属性名称转换为小写。考虑以下示例。

```
CREATE TABLE t2 (s) AS SELECT LOWER_ATTRIBUTE_NAMES(s) FROM t1;


SELECT s.attributename FROM t2;

attributename
-------------
"Value"


SELECT s.AttributeName FROM t2;

attributename
```

```
-------------
"Value"


SELECT s."attributename" FROM t2;

attributename
-------------
"Value"


SELECT s."AttributeName" FROM t2;

attributename
-------------
"Value"
```

用于处理大小写混合的对象属性名称的一个相关选项是 enable_case_sensitive_super_attribute 配置选项，它让 Amazon Redshift 可以识别 SUPER 属性名称中的大小写。这可能是使用 LOWER_ATTRIBUTE_NAMES 的替代解决方案。有关 enable_case_sensitive_super_attribute 的更多信息，请转至 enable_case_sensitive_super_attribute。

示例

将 SUPER 属性名称转换为小写

以下示例使用 LOWER_ATTRIBUTE_NAMES 来转换表中所有 SUPER 值的属性名称。

```
-- Create a table and insert several SUPER values.
CREATE TABLE t (i INT, s SUPER);

INSERT INTO t VALUES
  (1, NULL),
  (2, 'A'::SUPER),
  (3, JSON_PARSE('{"AttributeName": "B"}')),
  (4, JSON_PARSE(
     '[{"Subobject": {"C": "C"},
       "Subarray": [{"D": "D"}, "E"]
      }]'));

-- Convert all attribute names to lowercase.
UPDATE t SET s = LOWER_ATTRIBUTE_NAMES(s);
```

```
SELECT i, s FROM t ORDER BY i;

 i |                         s
---+-----------------------------------------------------
 1 | NULL
 2 | "A"
 3 | {"attributename":"B"}
 4 | [{"subobject":{"c":"C"},"subarray":[{"d":"D"}, "E"]}]
```

观察 LOWER_ATTRIBUTE_NAMES 的工作原理。

- NULL 值和标量 SUPER 值（例如 "A"）保持不变。
- 在 SUPER 对象中，所有属性名称都更改为小写，但诸如 "B" 之类的属性值保持不变。
- LOWER_ATTRIBUTE_NAMES 以递归方式应用于嵌套在 SUPER 数组或其它对象内的任何 SUPER 对象。

在具有重复属性名称的 SUPER 对象上使用 LOWER_ATTRIBUTE_NAMES

如果 SUPER 对象包含的属性的名称仅在大小写上有所不同，则 LOWER_ATTRIBUTE_NAMES 将引发错误。考虑以下示例。

```
SELECT LOWER_ATTRIBUTE_NAMES(JSON_PARSE('{"A": "A", "a": "a"}'));

error:   Invalid input
code:    8001
context: SUPER value has duplicate attributes after case conversion.
```

OBJECT 函数

创建 SUPER 数据类型的对象。

语法

```
OBJECT ( [ key1, value1 ], [ key2, value2 ...] )
```

参数

key1, key2

计算结果为 VARCHAR 类型字符串的表达式。

value1, value2

除日期时间类型以外的任何 Amazon Redshift 数据类型的表达式，因为 Amazon Redshift 不会将日期时间类型强制转换为 SUPER 数据类型。有关日期时间类型的更多信息，请参阅日期时间类型。

对象中的 value 表达式无需为同一数据类型。

返回类型

SUPER

示例

```
-- Creates an empty object.
select object();

object
--------
{}
(1 row)

-- Creates objects with different keys and values.
select object('a', 1, 'b', true, 'c', 3.14);

object
---------------------------
{"a":1,"b":true,"c":3.14}
(1 row)

select object('a', object('aa', 1), 'b', array(2,3), 'c', json_parse('{}'));

object
---------------------------------
{"a":{"aa":1},"b":[2,3],"c":{}}
(1 row)

-- Creates objects using columns from a table.
create table bar (k varchar, v super);
insert into bar values ('k1', json_parse('[1]')), ('k2', json_parse('{}'));
select object(k, v) from bar;

object
------------
```

```
{"k1":[1]}
{"k2":{}}
(2 rows)

-- Errors out because DATE type values can't be converted to SUPER type.
select object('k', '2008-12-31'::date);

ERROR:  OBJECT could not convert type date to super
```

OBJECT_TRANSFORM 函数

转换 SUPER 对象。

语法

```
OBJECT_TRANSFORM(
  input
  [KEEP path1, ...]
  [SET
    path1, value1,
    ...,  ...
  ]
)
```

参数

input

解析为 SUPER 类型对象的表达式。

KEEP

此子句中指定的所有路径值都将保留并传递到输出对象。

此子句是可选的。

path1、path2...

常量字符串文本，格式为双引号路径组成部分，以句点分隔。例如，`'"a"."b"."c"'` 是一个有效的路径值。这适用于 KEEP 和 SET 子句中的路径参数。

SET

路径和值对，可修改现有路径或添加新路径，并在输出对象中设置该路径的值。

此子句是可选的。

value1、value2...

解析为 SUPER 类型值的表达式。请注意，数值、文本和布尔类型可解析为 SUPER。

返回类型

SUPER

使用说明

OBJECT_TRANSFORM 返回一个 SUPER 类型对象，其中包含在 KEEP 中指定的输入路径值以及在 SET 中指定的路径和值对。

如果 KEEP 和 SET 都为空，则 OBJECT_TRANSFORM 返回输入。

如果输入不是 SUPER 类型对象，则无论 KEEP 或 SET 值如何，OBJECT_TRANSFORM 都会返回输入。

示例

以下示例将一个 SUPER 对象转换为另一个 SUPER 对象。

```
CREATE TABLE employees (
    col_person SUPER
);

INSERT INTO employees
VALUES
    (
        json_parse('
            {
                "name": {
                    "first": "John",
                    "last": "Doe"
                },
                "age": 25,
                "ssn": "111-22-3333",
                "company": "Company Inc.",
                "country": "U.S."
            }
        ')
    ),
    (
        json_parse('
```

```
            {
                "name": {
                    "first": "Jane",
                    "last": "Appleseed"
                },
                "age": 34,
                "ssn": "444-55-7777",
                "company": "Organization Org.",
                "country": "Ukraine"
            }
        ')
    )
;

SELECT
    OBJECT_TRANSFORM(
        col_person
        KEEP
            '"name"."first"',
            '"age"',
            '"company"',
            '"country"'
        SET
            '"name"."first"', UPPER(col_person.name.first::TEXT),
            '"age"', col_person.age + 5,
            '"company"', 'Amazon'
    ) AS col_person_transformed
FROM employees;

--This result is formatted for ease of reading.
                  col_person_transformed
-------------------------------------------------------------
{
    "name": {
        "first": "JOHN"
    },
    "age": 30,
    "company": "Amazon",
    "country": "U.S."
}
{
    "name": {
        "first": "JANE"
    },
```

```
    "age": 39,
    "company": "Amazon",
    "country": "Ukraine"
}
```

UPPER_ATTRIBUTE_NAMES 函数

使用与 UPPER 函数相同的大小写转换例程，将 SUPER 值中的所有适用属性名称转换为大写。UPPER_ATTRIBUTE_NAMES 支持 UTF-8 多字节字符，每个字符最多可以有 4 个字节。

要将 SUPER 属性名称转换为小写，请使用 LOWER_ATTRIBUTE_NAMES 函数。

语法

```
UPPER_ATTRIBUTE_NAMES(super_expression)
```

参数

super_expression

SUPER 表达式。

返回类型

SUPER

示例

将 SUPER 属性名称转换为大写

以下示例使用 UPPER_ATTRIBUTE_NAMES 来转换表中所有 SUPER 值的属性名称。

```
-- Create a table and insert several SUPER values.
CREATE TABLE t (i INT, s SUPER);

INSERT INTO t VALUES
  (1, NULL),
  (2, 'a'::SUPER),
  (3, JSON_PARSE('{"AttributeName": "b"}')),
  (4, JSON_PARSE(
     '[{"Subobject": {"c": "c"},
```

```
        "Subarray": [{"d": "d"}, "e"]
      }]'));

-- Convert all attribute names to uppercase.
UPDATE t SET s = UPPER_ATTRIBUTE_NAMES(s);

SELECT i, s FROM t ORDER BY i;

 i |                         s
---+-----------------------------------------------------
 1 | NULL
 2 | "a"
 3 | {"ATTRIBUTENAME":"B"}
 4 | [{"SUBOBJECT":{"C":"c"},"SUBARRAY":[{"D":"d"}, "e"]}]
```

观察 UPPER_ATTRIBUTE_NAMES 的工作原理。

- NULL 值和标量 SUPER 值（例如 "a"）保持不变。
- 在 SUPER 对象中，所有属性名称都更改为大写，但诸如 "b" 之类的属性值保持不变。
- UPPER_ATTRIBUTE_NAMES 以递归方式应用于嵌套在 SUPER 数组或其它对象内的任何 SUPER 对象。

在具有重复属性名称的 SUPER 对象上使用 UPPER_ATTRIBUTE_NAMES

如果 SUPER 对象包含的属性的名称仅在大小写上有所不同，则 UPPER_ATTRIBUTE_NAMES 将引发错误。考虑以下示例。

```
SELECT UPPER_ATTRIBUTE_NAMES(JSON_PARSE('{"A": "A", "a": "a"}'));

error:   Invalid input
code:    8001
context: SUPER value has duplicate attributes after case conversion.
```

空间函数

几何对象之间的关系基于维度扩展的九交模型 (DE-9IM)。此模型定义了诸如等于、包含和覆盖之类的谓词。有关空间关系的定义的更多信息，请参阅 Wikipedia 中的 DE-9IM。

有关如何在 Amazon Redshift 中使用空间数据的更多信息，请参阅在 Amazon Redshift 中查询空间数据。

Amazon Redshift 提供了可以使用 GEOMETRY 和 GEOGRAPHY 数据类型的空间函数。下面列出了支持 GEOGRAPHY 数据类型的函数：

- ST_Area
- ST_AsEWKT
- ST_AsGeoJSON
- ST_AsHexEWKB
- ST_AsHexWKB
- ST_AsText
- ST_Distance
- ST_GeogFromText
- ST_GeogFromWKB
- ST_Length
- ST_NPoints
- ST_Perimeter

下面列出了 Amazon Redshift 支持的全套空间函数。

主题

- AddBBox
- DropBBox
- GeometryType
- H3_FromLongLat
- H3_FromPoint
- H3_Polyfill
- ST_AddPoint
- ST_Angle
- ST_Area
- ST_AsBinary
- ST_AsEWKB
- ST_AsEWKT

- ST_AsGeoJSON
- ST_AsHexWKB
- ST_AsHexEWKB
- ST_AsText
- ST_Azimuth
- ST_Boundary
- ST_Buffer
- ST_Centroid
- ST_Collect
- ST_Contains
- ST_ContainsProperly
- ST_ConvexHull
- ST_CoveredBy
- ST_Covers
- ST_Crosses
- ST_Dimension
- ST_Disjoint
- ST_Distance
- ST_DistanceSphere
- ST_DWithin
- ST_EndPoint
- ST_Envelope
- ST_Equals
- ST_ExteriorRing
- ST_Force2D
- ST_Force3D
- ST_Force3DM
- ST_Force3DZ
- ST_Force4D

- ST_GeoHash
- ST_GeogFromText
- ST_GeogFromWKB
- ST_GeometryN
- ST_GeometryType
- ST_GeomFromEWKB
- ST_GeomFromEWKT
- ST_GeomFromGeoHash
- ST_GeomFromGeoJSON
- ST_GeomFromGeoSquare
- ST_GeomFromText
- ST_GeomFromWKB
- ST_GeoSquare
- ST_InteriorRingN
- ST_Intersects
- ST_Intersection
- ST_IsPolygonCCW
- ST_IsPolygonCW
- ST_IsClosed
- ST_IsCollection
- ST_IsEmpty
- ST_IsRing
- ST_IsSimple
- ST_IsValid
- ST_Length
- ST_LengthSphere
- ST_Length2D
- ST_LineFromMultiPoint
- ST_LineInterpolatePoint

- ST_M
- ST_MakeEnvelope
- ST_MakeLine
- ST_MakePoint
- ST_MakePolygon
- ST_MemSize
- ST_MMax
- ST_MMin
- ST_Multi
- ST_NDims
- ST_NPoints
- ST_NRings
- ST_NumGeometries
- ST_NumInteriorRings
- ST_NumPoints
- ST_Perimeter
- ST_Perimeter2D
- ST_Point
- ST_PointN
- ST_Points
- ST_Polygon
- ST_RemovePoint
- ST_Reverse
- ST_SetPoint
- ST_SetSRID
- ST_Simplify
- ST_SRID
- ST_StartPoint
- ST_Touches

- ST_Transform
- ST_Union
- ST_Within
- ST_X
- ST_XMax
- ST_XMin
- ST_Y
- ST_YMax
- ST_YMin
- ST_Z
- ST_ZMax
- ST_ZMin
- SupportsBBox

AddBBox

AddBBox 返回支持使用预先计算的边界框进行编码的输入几何体的副本。有关对边界框的支持的更多信息，请参阅边界框。

语法

```
AddBBox(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回支持使用边界框编码的输入多边形几何体的副本。

```
SELECT ST_AsText(AddBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
 st_astext
----------
 POLYGON((0 0,1 0,0 1,0 0))
```

DropBBox

DropBBox 返回不支持使用预先计算的边界框进行编码的输入几何体的副本。有关对边界框的支持的更多信息，请参阅边界框。

语法

```
DropBBox(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回不支持使用边界框编码的输入多边形几何体的副本。

```
SELECT ST_AsText(DropBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
 st_astext
----------
 POLYGON((0 0,1 0,0 1,0 0))
```

GeometryType

GeometryType 以字符串形式返回输入几何体的子类型。

语法

```
GeometryType(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

VARCHAR，表示 geom 的子类型。

如果 geom 为 null，则返回 null。

返回的值如下所示。

返回了 2D、3DZ、4D 几何体的字符串值	返回了 3DM 几何体的字符串值	几何体子类型
POINT	POINTM	在 geom 为 POINT 子类型时返回
LINESTRING	LINESTRINGM	在 geom 为 LINESTRING 子类型时返回
POLYGON	POLYGONM	在 geom 为 POLYGON 子类型时返回
MULTIPOINT	MULTIPOINTM	在 geom 为 MULTIPOINT 子类型时返回

返回了 2D、3DZ、4D 几何体的字符串值	返回了 3DM 几何体的字符串值	几何体子类型
MULTILINESTRING	MULTILINESTRINGM	在 geom 为 MULTILINESTRING 子类型时返回
MULTIPOLYGON	MULTIPOLYGONM	在 geom 为 MULTIPOLYGON 子类型时返回
GEOMETRYCOLLECTION	GEOMETRYCOLLECTIONM	在 geom 为 GEOMETRYCOLLECTION 子类型时返回

示例

以下 SQL 转换多边形的已知文本 (WKT) 表示形式，并以字符串形式返回 GEOMETRY 子类型。

```
SELECT GeometryType(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
geometrytype
-------------
 POLYGON
```

H3_FromLongLat

H3_FromLongLat 根据输入的经度、纬度和分辨率返回相应的 H3 单元格 ID。有关 H3 索引的信息，请参阅H3。

语法

```
H3_FromLongLat(longitude, lattitude, resolution)
```

参数

longitude

一个 DOUBLE PRECISION 数据类型的值，或一个计算结果为 DOUBLE PRECISION 类型的表达式。

latitude

一个 DOUBLE PRECISION 数据类型的值，或一个计算结果为 DOUBLE PRECISION 类型的表达式。

resolution

一个 INTEGER 数据类型的值，或一个计算结果为 INTEGER 类型的表达式。该值表示 H3 网格系统的分辨率。该值必须是 0-15 之间的整数，包括 0 和 15。0 表示最粗糙，15 表示最精细。

返回类型

BIGINT – 表示 H3 单元格 ID。

如果 resolution 超出范围，则返回错误信息。

示例

下面的 SQL 根据经度 0、纬度 0 和分辨率 10 返回 H3 单元格 ID。

```
SELECT H3_FromLongLat(0, 0, 10);
```

```
 h3_fromlonglat
------------------
 623560421467684863
```

H3_FromPoint

H3_FromPoint 根据输入的几何点和分辨率返回相应的 H3 单元格 ID。有关 H3 索引的信息，请参阅H3。

语法

```
H3_FromPoint(geom, resolution)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。geom 必须是 POINT。

resolution

一个 INTEGER 数据类型的值，或一个计算结果为 INTEGER 类型的表达式。该值表示 H3 网格系统的分辨率。该值必须是 0-15 之间的整数，包括 0 和 15。0 表示最粗糙，15 表示最精细。

返回类型

BIGINT – 表示 H3 单元格 ID。

如果 geom 不是 POINT，则返回一个错误。

如果 resolution 超出范围，则返回错误信息。

如果 geom 为空，则返回 NULL。

示例

下面的 SQL 根据点 0,0 和分辨率 10 返回 H3 单元格 ID。

```
SELECT H3_FromPoint(ST_GeomFromText('POINT(0 0)'), 10);
```

```
 h3_frompoint
--------------------
 623560421467684863
```

H3_Polyfill

H3_Polyfill 返回与给定分辨率的输入多边形中所含六边形和五边形相对应的 H3 单元格 ID。有关 H3 索引的信息，请参阅H3。

语法

```
H3_Polyfill(geom, resolution)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。geom 必须是 POLYGON。

resolution

一个 INTEGER 数据类型的值，或一个计算结果为 INTEGER 类型的表达式。该值表示 H3 网格系统的分辨率。该值必须是 0-15 之间的整数，包括 0 和 15。0 表示最粗糙，15 表示最精细。

返回类型

SUPER – 表示 H3 单元格 ID 的列表。

如果 geom 不是 POLYGON，则返回一个错误。

如果 resolution 超出范围，则返回错误信息。

如果 geom 为空，则返回 NULL。

示例

下面的 SQL 根据多边形和分辨率 4 返回一个 SUPER 数据类型数组，其中包含 H3 单元格 ID。

```
SELECT H3_Polyfill(ST_GeomFromText('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'), 4);
```

```
 h3_polyfill
-------------------------------------------------------------------------------------------------
 [596538848238895103,596538805289222143,596538856828829695,596538813879156735,5965379205259591
```

ST_AddPoint

ST_AddPoint 返回一个线串几何体，它与添加了点的输入几何体相同。如果提供了索引，则在索引位置添加点。如果索引为 -1 或未提供，则在线串后面附加点。

索引是从零开始的。结果的空间参考系统标识符 (SRID) 与输入几何体的相同。

返回的几何体的维度与 geom1 值的相同。如果 geom1 和 geom2 具有不同的维度，则 geom2 会投影到 geom1 的维度。

语法

```
ST_AddPoint(geom1, geom2)
```

```
ST_AddPoint(geom1, geom2, index)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 LINESTRING。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 POINT。该点可以是空点。

index

一个 INTEGER 数据类型的值，表示从零开始的索引的位置。

返回类型

GEOMETRY

如果 geom1、geom2 或 index 为 null，则返回 null。

如果 geom2 是空点，则会返回 geom1 的副本。

如果 geom1 不是 LINESTRING，则返回一个错误。

如果 geom2 不是 POINT，则返回一个错误。

如果 index 超出范围，则返回一个错误。索引位置的有效值为 -1 或为一个介于 0 和 ST_NumPoints(geom1) 之间的值。

示例

以下 SQL 向线串添加点以使其成为闭合线串。

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_AddPoint(g, ST_StartPoint(g))) FROM tmp;
```

```
 st_asewkt
-----------------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 5,0 5,0 0)
```

以下 SQL 将点添加到线串中的特定位置。

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_AddPoint(g, ST_SetSRID(ST_Point(5, 10), 4326), 3)) FROM tmp;
```

```
 st_asewkt
-----------------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 10,5 5,0 5)
```

ST_Angle

ST_Angle 返回顺时针方向测量的点之间的角度（以弧度为单位），如下所示：

- 如果输入三个点，则测量返回的角度 P1-P2-P3，就好像通过围绕 P2 顺时针从 P1 旋转到 P3 获得角度一样。
- 如果输入四个点，则返回有向线段 P1-P2 和 P3-P4 形成的返回顺时针角度。如果输入为退化情况（即，P1 等于 P2，或 P3 等于 P4），则返回 null。

返回值以弧度为单位且在 [0, 2π) 范围内。

ST_Angle 对输入几何体的 2D 投影进行操作。

语法

```
ST_Angle(geom1, geom2, geom3)
```

```
ST_Angle(geom1, geom2, geom3, geom4)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 POINT。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 POINT。

geom3

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 POINT。

geom4

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 POINT。

返回类型

DOUBLE PRECISION.

如果 geom1 等于 geom2，或者 geom2 等于 geom3，则会返回 null。

如果 geom1、geom2、geom3 或 geom4 为 null，则会返回 null。

如果 geom1、geom2、geom3 或 geom4 中的任何一个为空点，则返回一个错误。

如果 geom1、geom2、geom3 和 geom4 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

示例

以下 SQL 返回转换为三个输入点度数的角度。

```
SELECT ST_Angle(ST_Point(1,1), ST_Point(0,0), ST_Point(1,0)) / Pi() * 180.0 AS angle;
```

```
 angle
---------------
    45
```

以下 SQL 返回转换为四个输入点度数的角度。

```
SELECT ST_Angle(ST_Point(1,1), ST_Point(0,0), ST_Point(1,0), ST_Point(2,0)) / Pi() *
 180.0 AS angle;
```

```
 angle
---------------
   225
```

ST_Area

对于输入几何体，ST_Area 返回 2D 投影的笛卡尔面积。面积单位与用于表示输入几何体坐标的单位相同。对于点、线串、多点和多线串，此函数返回 0。对于几何体集合，它返回集合中几何体的面积之和。

对于输入地理，ST_Area 返回由 SRID 确定的在椭球体上计算的输入平面地理的 2D 投影的测地线面积。长度以平方米为单位。对于点、多点和线性地理，此函数返回零 (0)。当输入为几何体集合时，此函数返回集合中的平面地理面积之和。

语法

```
ST_Area(geo)
```

参数

geo

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。

返回类型

DOUBLE PRECISION

如果 geo 为 null，则返回 null。

示例

以下 SQL 返回多边形的笛卡尔面积。

```
SELECT ST_Area(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((10 0,20 0,20 10,10
 0)))'));
```

```
 st_area
---------
     100
```

以下 SQL 返回地理中多边形的面积。

```
SELECT ST_Area(ST_GeogFromText('polygon((34 35, 28 30, 25 34, 34 35))'));
```

```
     st_area
------------------
 201824655743.383
```

以下 SQL 对线性地理返回零。

```
SELECT ST_Area(ST_GeogFromText('multipoint(0 0, 1 1, -21.32 121.2)'));
```

```
 st_area
---------
       0
```

ST_AsBinary

ST_AsBinary 返回输入几何体的十六进制已知二进制 (WKB) 表示形式。对于 3DZ、3DM 和 4D 几何体，ST_AsBinary 使用开放地理空间联盟 (OGC) 标准值作为几何体类型。

语法

```
ST_AsBinary(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

VARBYTE

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回多边形的十六进制 WKB 表示形式。

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_asbinary
--------------------------------
0103000000010000000500000000000000000000000000000000000000000000000000F03F000000
```

ST_AsEWKB

ST_AsEWKB 返回输入几何体的扩展的已知二进制 (EWKB) 表示形式。对于 3DZ、3DM 和 4D 几何体，ST_AsEWKB 使用开放地理空间联盟 (OGC) 标准值作为几何体类型。

语法

```
ST_AsEWKB(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

VARBYTE

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回多边形的十六进制 EWKB 表示形式。

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_asewkb
--------------------------------
0103000020E6100000010000000500000000000000000000000000000000000000000000000000F0
```

ST_AsEWKT

ST_AsEWKT 返回输入几何体或地理的扩展的已知文本 (EWKT) 表示形式。对于 3DZ、3DM 和 4D 几何体，ST_AsEWKT 会将 Z、M 或 ZM 附加到几何类型的 WKT 值。

语法

```
ST_AsEWKT(geo)
```

```
ST_AsEWKT(geo, precision)
```

参数

geo

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。

精度

一个 数据类型的 值INTEGER 对于几何体，geo 的坐标使用指定的精度 1–20 显示。如果未指定精度，则默认值为 15。对于地理，geo 的坐标使用指定的精度显示。如果未指定精度，则默认值为 15。

返回类型

VARCHAR

如果 geo 为 null，则返回 null。

如果精度 为 null，则返回 null。

如果结果大于 64-KB VARCHAR，则将返回一个错误。

示例

以下 SQL 返回线串的 EWKT 表示形式。

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326));
```

```
st_asewkt
---------------------------------
 SRID=4326;LINESTRING(3.14159265358979 -6.28318530717959,2.71828182845905
 -1.41421356237309)
```

以下 SQL 返回线串的 EWKT 表示形式。使用六位数精度显示几何体的坐标。

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326), 6);
```

```
st_asewkt
---------------------------------
 SRID=4326;LINESTRING(3.14159 -6.28319,2.71828 -1.41421)
```

以下 SQL 返回地理的 EWKT 表示形式。

```
SELECT ST_AsEWKT(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
                  st_asewkt
-----------------------------------------------
 SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

ST_AsGeoJSON

ST_AsGeoJSON 返回输入几何体或地理的 GeoJSON 表示形式。有关 GeoJSON 的更多信息，请参阅 Wikipedia 中的 GeoJSON。

对于 3DZ 和 4D 几何体，输出几何体是输入 3DZ 或 4D 几何体的 3DZ 投影。也就是说，x、y, 和 z 坐标存在于输出中。对于 3DZ 几何体，输出几何体是输入 3DM 几何体的 2D 投影。也就是说，只有 x 和 y 坐标存在于输出中。

对于输入地理，ST_AsGeoJSON 返回输入地理的 GeoJSON 表示形式。地理的坐标使用指定的精度显示。

语法

```
ST_AsGeoJSON(geo)
```

```
ST_AsGeoJSON(geo, precision)
```

参数

geo

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。

精度

一个 数据类型的 值INTEGER 对于几何体，geo 的坐标使用指定的精度 1–20 显示。如果未指定精度，则默认值为 15。对于地理，geo 的坐标使用指定的精度显示。如果未指定精度，则默认值为 15。

返回类型

VARCHAR

如果 geo 为 null，则返回 null。

如果精度 为 null，则返回 null。

如果结果大于 64-KB VARCHAR，则将返回一个错误。

示例

以下 SQL 返回线串的 GeoJSON 表示形式。

```
SELECT ST_AsGeoJSON(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)'));
```

```
st_asgeojson
-----------------------------------------------------------------------------------------------------------
 {"type":"LineString","coordinates":[[3.14159265358979,-6.28318530717959],
[2.71828182845905,-1.4142135623709]]}
```

以下 SQL 返回线串的 GeoJSON 表示形式。使用六位数精度显示几何体的坐标。

```
SELECT ST_AsGeoJSON(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)'), 6);
```

```
st_asgeojson
-------------------------------------------------------------------------
  {"type":"LineString","coordinates":[[3.14159,-6.28319],[2.71828,-1.41421]]}
```

以下 SQL 返回地理的 GeoJSON 表示形式。

```
SELECT ST_AsGeoJSON(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
                            st_asgeojson
-------------------------------------------------------------------
 {"type":"LineString","coordinates":[[110,40],[2,3],[-10,80],[-7,9]]}
```

ST_AsHexWKB

ST_AsHexWKB 使用 ASCII 十六进制字符（0–9、A–F）返回输入几何体或地理的十六进制已知二进制 (WKB) 表示形式。对于 3DZ、3DM 和 4D 几何体或地理，ST_AsHexWKB 使用开放地理空间联盟 (OGC) 标准值作为几何体或地理类型。

语法

```
ST_AsHexWKB(geo)
```

参数

geo

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。

返回类型

VARCHAR

如果 geo 为 null，则返回 null。

如果结果大于 64-KB VARCHAR，则将返回一个错误。

示例

以下 SQL 返回几何体中多边形的十六进制 WKB 表示形式。

```
SELECT ST_AsHexWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_ashexwkb
--------------------------------
0103000000010000000500000000000000000000000000000000000000000000000000F03F000000
```

以下 SQL 返回地理中多边形的十六进制 WKB 表示形式。

```
SELECT ST_AsHexWKB(ST_GeogFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))'));
```

```
st_ashexwkb
--------------------------------
0103000000010000000500000000000000000000000000000000000000000000000000F03F000000
```

ST_AsHexEWKB

ST_AsHexEWKB 使用 ASCII 十六进制字符（0–9、A–F）返回输入几何体或地理的扩展的已知二进制 (WKB) 表示形式。对于 3DZ、3DM 和 4D 几何体或地理，ST_AsHexEWKB 使用 PostGIS 扩展 WKB 值作为几何或地理类型。

语法

```
ST_AsHexEWKB(geo)
```

参数

geo

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。

返回类型

VARCHAR

如果 geo 为 null，则返回 null。

如果结果大于 64-KB VARCHAR，则将返回一个错误。

示例

以下 SQL 返回几何体中多边形的十六进制 EWKB 表示形式。

```
SELECT ST_AsHexEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_ashexewkb
---------------------------------
0103000020E6100000010000000500000000000000000000000000000000000000000000000000F
```

以下 SQL 返回地理中多边形的十六进制 EWKB 表示形式。

```
SELECT ST_AsHexEWKB(ST_GeogFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))'));
```

```
st_ashexewkb
---------------------------------
0103000020E6100000010000000500000000000000000000000000000000000000000000000000F
```

ST_AsText

ST_AsText 返回输入几何体或地理的已知文本 (WKT) 表示形式。对于 3DZ、3DM 和 4D 几何体或地理，ST_AsEWKT 会将 Z、M 或 ZM 附加到几何或地理类型的 WKT 值。

语法

```
ST_AsText(geo)
```

```
ST_AsText(geo, precision)
```

参数

geo

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。

精度

一个 数据类型的 值INTEGER 对于几何体，geo 的坐标使用指定的精度 1–20 显示。如果未指定精度，则默认值为 15。对于地理，geo 的坐标使用指定的精度显示。如果未指定精度，则默认值为 15。

返回类型

VARCHAR

如果 geo 为 null，则返回 null。

如果精度 为 null，则返回 null。

如果结果大于 64-KB VARCHAR，则将返回一个错误。

示例

以下 SQL 返回线串的 WKT 表示形式。

```
SELECT ST_AsText(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326));
```

```
st_astext
-------------------------------
LINESTRING(3.14159265358979 -6.28318530717959,2.71828182845905 -1.41421356237309)
```

以下 SQL 返回线串的 WKT 表示形式。使用六位数精度显示几何体的坐标。

```
SELECT ST_AsText(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326), 6);
```

```
st_astext
-----------------------------------------------
 LINESTRING(3.14159 -6.28319,2.71828 -1.41421)
```

以下 SQL 返回地理的 WKT 表示形式。

```
SELECT ST_AsText(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
             st_astext
-----------------------------------
 LINESTRING(110 40,2 3,-10 80,-7 9)
```

ST_Azimuth

ST_Azimuth 使用两个输入点的 2D 投影返回基于北向的笛卡尔方位。

语法

```
ST_Azimuth(point1, point2)
```

参数

point1

一个 POINT 数据类型的 GEOMETRY 值。point1 的空间参考系统标识符 (SRID) 必须与 point2 的 SRID 匹配。

point2

一个 POINT 数据类型的 GEOMETRY 值。point2 的 SRID 必须与 point1 的 SRID 匹配。

返回类型

一个数字，它表示 DOUBLE PRECISION 数据类型的角度（以弧度为单位）。值的范围从 0（包含）到 2 pi（不包含）。

如果 point1 或 point2 是空点，则返回一个错误。

如果 point1 或 point2 为 null，则返回 null。

如果 point1 和 point2 相等，则返回 null。

如果 point1 或 point2 不是点，则返回一个错误。

如果 point1 和 point2 不具有空间参考系统标识符 (SRID) 的值，则返回一个错误。

示例

以下 SQL 返回输入点的方位。

```
SELECT ST_Azimuth(ST_Point(1,2), ST_Point(5,6));
```

```
st_azimuth
--------------------
 0.7853981633974483
```

ST_Boundary

ST_Boundary 返回输入几何体的边界，如下所示：

- 如果输入几何体为空（即，它不包含点），则按原样返回。
- 如果输入几何是点或非空多点，则返回空几何体集合。
- 如果输入是线串或多线串，则返回包含边界上所有点的多点。多点可能为空）。
- 如果输入是一个没有任何内环的面，则返回一个表示其边界的闭合线串。
- 如果输入是具有内环的面，或者是多面，则返回多线串。多线串包含平面几何体中所有环的所有边界，作为闭合线串。

为了确定点相等性，ST_Boundary 会对输入几何体的 2D 投影进行操作。如果输入几何体为空，则返回该几何体的维度与输入相同的副本。对于非空 3DM 和 4D 几何体，其 m 坐标会被删除。在 3DZ 和

4D 多线串的特殊情况下，多线串边界点的 z 坐标被计算为具有相同 2D 投影的线串边界点的不同 z 值的平均值。

语法

```
ST_Boundary(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY

如果 geom 为 null，则返回 null。

如果 geom 为 GEOMETRYCOLLECTION，则返回一个错误。

示例

以下 SQL 返回输入多边形的边界，作为多线串。

```
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(1 1,1
 2,2 1,1 1))')));
```

```
st_asewkt
--------------------
 MULTILINESTRING((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 1,1 1))
```

ST_Buffer

ST_Buffer 返回 2D 几何体，该几何表示与 xy 笛卡尔平面上投影的输入几何体之间的距离小于或等于输入距离的所有点。

语法

```
ST_Buffer(geom, distance)
```

```
ST_Buffer(geom, distance, number_of_segments_per_quarter_circle)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

distance

数据类型 DOUBLE PRECISION 的值表示缓冲区的距离（或半径）。

number_of_segments_per_quarter_circle

数据类型 INTEGER 的值。此值确定了围绕输入几何体的每个顶点大致四分之一圆的点数。负值默认为零。默认值为 8。

返回类型

GEOMETRY

ST_Buffer 函数返回 xy 笛卡尔平面中的二维 (2D) 几何体。

如果 geom 为 GEOMETRYCOLLECTION，则返回一个错误。

示例

以下 SQL 返回输入线串的缓冲区。

```
SELECT ST_AsEwkt(ST_Buffer(ST_GeomFromText('LINESTRING(1 2,5 2,5 8)'), 2));
```

```
               st_asewkt
POLYGON((-1 2,-0.96157056080646 2.39018064403226,-0.847759065022573
 2.76536686473018,-0.662939224605089 3.11114046603921,-0.414213562373093
 3.4142135623731,-0.111140466039201 3.66293922460509,0.234633135269824
 3.84775906502257,0.609819355967748 3.96157056080646,1 4,3 4,3 8,3.03842943919354
 8.39018064403226,3.15224093497743 8.76536686473018,3.33706077539491
 9.11114046603921,3.58578643762691 9.4142135623731,3.8888595339608
 9.66293922460509,4.23463313526982 9.84775906502257,4.60981935596775
 9.96157056080646,5 10,5.39018064403226 9.96157056080646,5.76536686473018
 9.84775906502257,6.11114046603921 9.66293922460509,6.4142135623731
```

```
 9.41421356237309,6.66293922460509 9.1111404660392,6.84775906502258
 8.76536686473017,6.96157056080646 8.39018064403225,7 8,7 2,6.96157056080646
 1.60981935596774,6.84775906502257 1.23463313526982,6.66293922460509
 0.888859533960796,6.41421356237309 0.585786437626905,6.1111404660392
 0.33706077539491,5.76536686473018 0.152240934977427,5.39018064403226
 0.0384294391935391,5 0,1 0,0.609819355967744 0.0384294391935391,0.234633135269821
 0.152240934977427,-0.111140466039204 0.337060775394909,-0.414213562373095
 0.585786437626905,-0.662939224605091 0.888859533960796,-0.847759065022574
 1.23463313526982,-0.961570560806461 1.60981935596774,-1 2))
```

以下 SQL 返回输入点几何体的缓冲区，该缓冲区大致为一个圆。由于该命令没有指定每个四分之一圆的区段数，因此该函数使用八个区段的默认值表示约四分之一圆。

```
SELECT ST_AsEwkt(ST_Buffer(ST_GeomFromText('POINT(3 4)'), 2));
```

```
               st_asewkt
POLYGON((1 4,1.03842943919354 4.39018064403226,1.15224093497743
 4.76536686473018,1.33706077539491 5.11114046603921,1.58578643762691
 5.4142135623731,1.8888595339608 5.66293922460509,2.23463313526982
 5.84775906502257,2.60981935596775 5.96157056080646,3 6,3.39018064403226
 5.96157056080646,3.76536686473019 5.84775906502257,4.11114046603921
 5.66293922460509,4.4142135623731 5.41421356237309,4.66293922460509
 5.1111404660392,4.84775906502258 4.76536686473017,4.96157056080646 4.39018064403225,5
 4,4.96157056080646 3.60981935596774,4.84775906502257 3.23463313526982,4.66293922460509
 2.8888595339608,4.41421356237309 2.58578643762691,4.1111404660392
 2.33706077539491,3.76536686473018 2.15224093497743,3.39018064403226 2.03842943919354,3
 2,2.60981935596774 2.03842943919354,2.23463313526982 2.15224093497743,1.8888595339608
 2.33706077539491,1.58578643762691 2.58578643762691,1.33706077539491
 2.8888595339608,1.15224093497743 3.23463313526982,1.03842943919354 3.60981935596774,1
 4))
```

以下 SQL 返回输入点几何体的缓冲区，该缓冲区大致为一个圆。由于该命令指定 3 作为每个四分之一圆的区段数，因此该函数使用三个区段表示约四分之一圆。

```
SELECT ST_AsEwkt(ST_Buffer(ST_GeomFromText('POINT(3 4)'), 2, 3));
```

```
               st_asewkt
POLYGON((1 4,1.26794919243112 5,2 5.73205080756888,3 6,4
 5.73205080756888,4.73205080756888 5,5 4,4.73205080756888 3,4 2.26794919243112,3 2,2
 2.26794919243112,1.26794919243112 3,1 4))
```

ST_Centroid

ST_Centroid 返回一个表示几何体质心的点，如下所示：

- 对于 POINT 几何体，它返回坐标为几何体中点坐标平均值的点。
- 对于 LINESTRING 几何体，它返回坐标是几何体分段中点的加权平均值的点，其中权重是几何体分段的长度。
- 对于 POLYGON 几何体，它返回坐标是平面几何体的三角测量的质心的加权平均值的点，其中权重是三角测量中三角形的面积。
- 对于几何体集合，它返回几何体集合中具有最大拓扑维度的几何体的质心的加权平均值。

语法

```
ST_Centroid(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY

如果 geom 为 null，则返回 null。

如果 geom 为空，则返回 null。

示例

以下 SQL 从输入线串返回中心点。

```
SELECT ST_AsEWKT(ST_Centroid(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9, -22
 -33)', 4326)))
```

```
                    st_asewkt
```

```
----------------------------------------------------
 SRID=4326;POINT(15.6965103455214 27.0206782881905)
```

ST_Collect

ST_Collect 有两个变体。一个接受两个几何体，一个接受聚合表达式。

ST_Collect 的第一个变体从输入几何体中创建几何体。输入几何体的顺序将保留。此变体的原理如下：

- 如果两个输入几何体都是点，则返回具有两个点的 MULTIPOINT。
- 如果两个输入几何体都是线串，则返回具有两个线串的 MULTILINESTRING。
- 如果两个输入几何体都是面，则返回具有两个面的 MULTIPOLYGON。
- 否则，返回具有两个输入几何体的 GEOMETRYCOLLECTION。

ST_Collect 的第二个变体从几何体列中的几何体创建几何体。没有确定的几何体返回顺序。指定 WITHIN GROUP (ORDER BY ...) 子句以指定返回几何体的顺序。此变体的原理如下：

- 如果输入聚合表达式中的所有非 NULL 行都是点，则返回包含聚合表达式中所有点的多点。
- 如果聚合表达式中的所有非 NULL 行都是线串，则返回包含聚合表达式中所有线串的多线串。
- 如果聚合表达式中的所有非 NULL 行都是面，则返回一个包含聚合表达式中所有面的多面。
- 否则，返回包含聚合表达式中的所有几何体的 GEOMETRYCOLLECTION。

ST_Collect 返回与输入几何体具有相同维度的几何体。所有的输入几何体必须具有相同的维度。

语法

```
ST_Collect(geom1, geom2)
```

```
ST_Collect(aggregate_expression)  [WITHIN GROUP (ORDER BY sort_expression1 [ASC | DESC]
 [, sort_expression2 [ASC | DESC] ...])]
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

aggregate_expression

数据类型 GEOMETRY 的列，或一个计算结果为 GEOMETRY 类型的表达式。

[WITHIN GROUP (ORDER BY sort_expression1 [ASC | DESC] [, sort_expression2 [ASC | DESC] ...])]

用于指定聚合值的排序顺序的可选子句。ORDER BY 子句包含排序表达式的列表。排序表达式是类似于查询选择列表中的有效排序表达式（如列名）的表达式。您可以指定升序 (ASC) 或降序 (DESC) 顺序。默认为 ASC。

返回类型

子类型 MULTIPOINT、MULTILINESTRING、MULTIPOLYGON 或 GEOMETRYCOLLECTION的 GEOMETRY。

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom1 或 geom2 均为 null，则返回 null。

如果 aggregate_expression 的所有行均为 null，则返回 null。

如果 geom1 为 null，则返回 geom2 的副本。同样，如果 geom2 为 null，则返回 geom1 的副本。

如果 geom1 和 geom2 具有不同的 SRID 值，则返回一个错误。

如果 aggregate_expression 中的两个几何体具有不同的 SRID 值，则返回一个错误。

如果返回的几何体大于 GEOMETRY 最大大小，则返回一个错误。

如果 geom1 和 geom2 具有不同的维度，则返回一个错误。

如果 aggregate_expression 中的两个几何体具有不同的维度，则返回一个错误。

示例

以下 SQL 返回包含两个输入几何体的几何体集合。

```
SELECT ST_AsText(ST_Collect(ST_GeomFromText('LINESTRING(0 0,1 1)'),
 ST_GeomFromText('POLYGON((10 10,20 10,10 20,10 10))')));
```

```
st_astext
-----------
 GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),POLYGON((10 10,20 10,10 20,10 10)))
```

以下 SQL 将表中的所有几何体收集到几何体集合中。

```
WITH tbl(g) AS (SELECT ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT NULL::geometry UNION ALL
SELECT ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326))
SELECT ST_AsEWKT(ST_Collect(g)) FROM tbl;
```

```
st_astext
-----------
 SRID=4326;GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(0 0,10 0),MULTIPOINT((13 4),(8 5),
(4 4)),POLYGON((0 0,10 0,0 10,0 0)))
```

以下 SQL 收集表中按 id 列分组并按此 ID 排序的所有几何体。在此示例中，生成的几何体按 ID 分组，如下所示：

- id 1 – 多点钟的点。
- id 2 – 多线串中的线串。
- id 3 – 几何体集合中的混合子类型。
- id 4 – 多面中的面。
- id 5 – null，且结果为 null。

```
WITH tbl(id, g) AS (SELECT 1, ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT 1, ST_GeomFromText('POINT(4 5)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(10 0,20 -5)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5))', 4326) UNION
 ALL
SELECT 4, ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((20 20,20 30,30 20,20 20))', 4326) UNION ALL
SELECT 1, NULL::geometry UNION ALL SELECT 2, NULL::geometry UNION ALL
```

```
SELECT 5, NULL::geometry UNION ALL SELECT 5, NULL::geometry)
SELECT id, ST_AsEWKT(ST_Collect(g)) FROM tbl GROUP BY id ORDER BY id;
```

```
 id |                                                st_asewkt

----
+-----------------------------------------------------------------------------------------------
  1 | SRID=4326;MULTIPOINT((1 2),(4 5))
  2 | SRID=4326;MULTILINESTRING((0 0,10 0),(10 0,20 -5))
  3 | SRID=4326;GEOMETRYCOLLECTION(MULTIPOINT((13 4),(8 5),(4 4)),MULTILINESTRING((-1
 -1,-2 -2),(-3 -3,-5 -5)))
  4 | SRID=4326;MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((20 20,20 30,30 20,20 20)))
  5 |
```

以下 SQL 将表中的所有几何体收集到几何体集合中。结果根据 id 按降序排序，然后根据它们的最小和最大 x 坐标按字典顺序排序。

```
WITH tbl(id, g) AS (
SELECT 1, ST_GeomFromText('POINT(4 5)', 4326) UNION ALL
SELECT 1, ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(10 0,20 -5)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5))', 4326) UNION
 ALL
SELECT 4, ST_GeomFromText('POLYGON((20 20,20 30,30 20,20 20))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326) UNION ALL
SELECT 1, NULL::geometry UNION ALL SELECT 2, NULL::geometry UNION ALL
SELECT 5, NULL::geometry UNION ALL SELECT 5, NULL::geometry)
SELECT ST_AsEWKT(ST_Collect(g) WITHIN GROUP (ORDER BY id DESC, ST_XMin(g), ST_XMax(g)))
 FROM tbl;
```

```
                          st_asewkt

-----------------------------------------------------------------------------------------------
 SRID=4326;GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0 0)),POLYGON((20 20,20 30,30
 20,20 20)),MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5)),MULTIPOINT((13 4),(8 5),(4
 4)),LINESTRING(0 0,10 0),LINESTRING(10 0,20 -5),POINT(1 2),POINT(4 5)
```

ST_Contains

如果第一个输入几何体的 2D 投影包含第二个输入几何体的 2D 投影，则 ST_Contains 返回 true。如果 A 中的每个点均为 B 中的一个点，并且其内部有非空相交区域，则几何体 B 包含几何体 A。

ST_Contains(A, B) 与 ST_Within(B, A) 等效。

语法

```
ST_Contains(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。此值将与 geom1 进行比较以确定它是否包含在 geom1 中。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

以下 SQL 检查第一个多边形是否包含第二个多边形。

```
SELECT ST_Contains(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_contains
-----------
```

```
 false
```

ST_ContainsProperly

如果两个输入几何体都为非空，并且第二个几何体的 2D 投影的所有点都是第一个几何体的 2D 投影的内部点，则 ST_ContainsProperly 返回 true。

语法

```
ST_ContainsProperly(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型不能为 GEOMETRYCOLLECTION。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型不能为 GEOMETRYCOLLECTION。将此值与 geom1 进行比较，以确定其所有点是否都是 geom1 的内点。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

在输入线串与输入面的内部和边界（但不是其外部）相交的位置，以下 SQL 返回 ST_Contains 和 ST_ContainsProperly 的值。面包含线串，但没有正确包含线串。

```
WITH tmp(g1, g2)
AS (SELECT ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('LINESTRING(5 5,10 5,10 6,5 5)')) SELECT ST_Contains(g1, g2),
 ST_ContainsProperly(g1, g2)
```

```
FROM tmp;
```

```
 st_contains | st_containsproperly
-------------+---------------------
 t           | f
```

ST_ConvexHull

ST_ConvexHull 返回一个几何体，该几何体表示输入几何体中包含的非空点的凸包。

对于空输入，生成的几何体与输入几何体相同。对于所有非空输入，该函数在输入几何体的 2D 投影上运行。但是，输出几何体的维度取决于输入几何体的维度。更具体地说，当输入几何体为非空 3DM 或 3D 几何体时，m 坐标将被删除。也就是说，返回的几何体的维度分别为 2D 或 3DZ。如果输入为非空 2D 或 3DZ 几何体，则生成的几何体具有相同的维度。

语法

```
ST_ConvexHull(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom 为 null，则返回 null。

返回的值如下所示。

凸包上的点数	几何体子类型
0	返回 geom 的副本。

凸包上的点数	几何体子类型
1	返回子类型 POINT。
2	返回子类型 LINESTRING 。返回线串的两个点按字典顺序排序。
3 或更大	返回不具有内环的 POLYGON 子类型。面是顺时针方向的，外环的第一个点是按字典顺序排列的环的最小点。

示例

以下 SQL 返回线串的扩展已知文本 (EWKT) 表示形式。在本例中，返回的凸包是面。

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('LINESTRING(0 0,1 0,0 1,1 1,0.5 0.5)')))
 as output;
```

```
output
-------------
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

以下 SQL 返回线串的 EWKT 表示形式。在本例中，返回的凸包是线串。

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('LINESTRING(0 0,1 1,0.2 0.2,0.6 0.6,0.5
 0.5)'))) as output;
```

```
output
-------------
LINESTRING(0 0,1 1)
```

以下 SQL 返回多点的 EWKT 表示形式。在本例中，返回的凸包是点。

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('MULTIPOINT(0 0,0 0,0 0)'))) as output;
```

```
output
-------------
POINT(0 0)
```

ST_CoveredBy

如果第一个输入几何体的 2D 投影被第二个输入几何体的 2D 投影覆盖，则 ST_CoveredBy 返回 true。如果几何体 A 和几何体 B 都是非空的，并且 A 中的每个点均为 B 中的一个点，则前者被后者覆盖。

ST_CoveredBy(A, B) 与 ST_Covers(B, A) 等效。

语法

```
ST_CoveredBy(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。此值将与 geom2 进行比较以确定它是否被 geom2 覆盖。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

以下 SQL 检查第一个多边形是否被第二个多边形覆盖。

```
SELECT ST_CoveredBy(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_coveredby
-----------
 true
```

ST_Covers

如果第一个输入几何体的 2D 投影被第二个输入几何体的 2D 投影覆盖，则 ST_Covers 返回 true。如果几何体 A 和几何体 B 都是非空的，并且 B 中的每个点均为 A 中的一个点，则前者覆盖了后者。

ST_Covers(A, B) 与 ST_CoveredBy(B, A) 等效。

语法

```
ST_Covers(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。此值将与 geom1进行比较以确定它是否覆盖了 geom1。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

以下 SQL 检查第一个多边形是否覆盖了第二个多边形。

```
SELECT ST_Covers(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_covers
-----------
 false
```

ST_Crosses

如果两个输入几何体的 2D 投影相互交叉，ST_Crosses 将返回 true。

语法

```
ST_Crosses(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

示例

以下 SQL 检查第一个面是否与第二个多点相交。在此示例中，多点与面的内部和外部相交，这就是 ST_Crosses返回 true 的原因。

```
SELECT ST_Crosses (ST_GeomFromText('polygon((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('multipoint(5 5,0 0,-1 -1)'));
```

```
st_crosses
-------------
 true
```

以下 SQL 检查第一个面是否与第二个多点相交。在此示例中，多点与面的外部相交，而不是其内部相交，这就是 ST_Crosses 返回 false 的原因。

```
SELECT ST_Crosses (ST_GeomFromText('polygon((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('multipoint(0 0,-1 -1)'));
```

```
st_crosses
-------------
 false
```

ST_Dimension

ST_Dimension 返回输入几何体的固有维度。固有维度 是几何体中定义的子类型的维度值。

对于 3DM、3DZ 和 4D 几何体输入，ST_Dimension 返回的结果与 2D 几何体输入的结果相同。

语法

```
ST_Dimension(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

INTEGER，表示 geom 的固有维度。

如果 geom 为 null，则返回 null。

返回的值如下所示。

返回的值	几何体子类型
0	在 geom 为 POINT 或 MULTIPOINT 子类型时返回
1	在 geom 为 LINESTRING 或 MULTILINESTRING 子类型时返回。
2	在 geom 为 POLYGON 或 MULTIPOLYGON 子类型时返回
0	在 geom 为空 GEOMETRYCOLLECTION 子类型时返回
集合组件的最大维度	在 geom 为 GEOMETRYCOLLECTION 子类型时返回

示例

以下 SQL 将四点 LINESTRING 的已知文本 (WKT) 表示形式转换为 GEOMETRY 对象，并返回线串的维度。

```
SELECT ST_Dimension(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'));
```

```
st_dimension
-------------
1
```

ST_Disjoint

如果两个输入几何体的 2D 投影没有共同点，则 ST_Disjoint 返回 true。

语法

```
ST_Disjoint(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

以下 SQL 检查第一个多边形是否与第二个多边形不相交。

```
SELECT ST_Disjoint(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,2 5,5 5,5
 2,2 2))'), ST_Point(4, 4));
```

```
st_disjoint
-----------
 true
```

ST_Distance

对于输入几何体，ST_Distance 返回两个输入几何体值的 2D 投影之间的最小欧氏距离。

对于 3DM、3DZ、4D 几何体，ST_Distance 返回两个输入几何体值的 2D 投影之间的欧氏距离。

对于输入地理，ST_Distance 返回两个 2D 点的测地线距离。距离以米为单位。对于点和空点以外的地理，将返回错误。

语法

```
ST_Distance(geo1, geo2)
```

参数

geo1

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。geo1 的数据类型必须与 geo2 相同。

geo2

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。geo2 的数据类型必须与 geo1 相同。

返回类型

DOUBLE PRECISION，采用与输入几何体或地理相同的单位。

如果 geo1 或 geo2 为 null 或为空，则返回 null。

如果 geo1 和 geo2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geo1 或 geo2 为几何体集合，则返回一个错误。

示例

以下 SQL 返回两个多边形之间的距离。

```
SELECT ST_Distance(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 -3,-2 -1,0 -3,-1 -3))'));
```

```
  st_distance
-----------
1.4142135623731
```

以下 SQL 使用 GEOGRAPHY 数据类型返回勃兰登堡门和柏林国会大厦之间的距离（以米为单位）。

```
SELECT ST_Distance(ST_GeogFromText('POINT(13.37761826722198 52.516411678282445)'),
 ST_GeogFromText('POINT(13.377950831464005 52.51705102546893)'));
```

```
   st_distance
------------------
 74.64129172609631
```

ST_DistanceSphere

ST_DistanceSphere 返回位于球体上的两个点几何体之间的距离。

语法

```
ST_DistanceSphere(geom1, geom2)
```

```
ST_DistanceSphere(geom1, geom2, radius)
```

参数

geom1

位于球体上的数据类型 GEOMETRY 的点值（以度为单位）。该点的第一个坐标是经度值。该点的第二个坐标是纬度值。对于 3DZ、3DM 或 4D 几何体，仅使用前两个坐标。

geom2

位于球体上的数据类型 GEOMETRY 的点值（以度为单位）。该点的第一个坐标是经度值。该点的第二个坐标是纬度值。对于 3DZ、3DM 或 4D 几何体，仅使用前两个坐标。

radius

数据类型 DOUBLE PRECISION 的球体半径。如果未提供 radius，则球体默认为地球，并且半径是根据椭球体的世界大地测量系统 (WGS) 84 表示形式来计算的。

返回类型

DOUBLE PRECISION，采用与半径相同的单位。如果未提供半径，则距离以米为单位。

如果 geom1 或 geom2 为 null 或为空，则返回 null。

如果未提供 radius，则结果是沿地球表面的米数。

如果 radius 是负数，则返回一个错误。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 不是点，则返回一个错误。

示例

以下示例 SQL 计算地球上两点之间的距离（以千米为单位）。

```
SELECT ROUND(ST_DistanceSphere(ST_Point(-122, 47), ST_Point(-122.1, 47.1))/ 1000, 0);
```

```
  round
-----------
 13
```

以下示例 SQL 计算位于德国的三个机场位置之间的距离（以公里为单位）：Berlin Tegel (TXL)、Munich International (MUC) 和 Frankfurt International (FRA)。

```
WITH airports_raw(code,lon,lat) AS (
(SELECT 'MUC', 11.786111, 48.353889) UNION
(SELECT 'FRA', 8.570556, 50.033333) UNION
(SELECT 'TXL', 13.287778, 52.559722)),
airports1(code,location) AS (SELECT code, ST_Point(lon, lat) FROM airports_raw),
airports2(code,location) AS (SELECT * from airports1)
SELECT (airports1.code || ' <-> ' || airports2.code) AS airports,
round(ST_DistanceSphere(airports1.location, airports2.location) / 1000, 0) AS
 distance_in_km
FROM airports1, airports2 WHERE airports1.code < airports2.code ORDER BY 1;
```

```
  airports   | distance_in_km
-------------+----------------
 FRA <-> MUC |            299
 FRA <-> TXL |            432
 MUC <-> TXL |            480
```

ST_DWithin

如果两个输入几何体值的 2D 投影之间的欧氏距离不大于阈值，则 ST_DWithin 返回 true。

语法

```
ST_DWithin(geom1, geom2, threshold)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

threshold

一个 数据类型的 值DOUBLE PRECISION 该值用输入参数的单位表示。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回 null。

如果 threshold 为负数，则返回一个错误。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

以下 SQL 检查两个多边形之间的距离是否在 5 个单位内。

```
SELECT ST_DWithin(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'),5);
```

```
st_dwithin
-----------
 true
```

ST_EndPoint

ST_EndPoint 返回输入线串的最后一个点。结果的空间参考系统标识符 (SRID) 值与输入几何体的相同。返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_EndPoint(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 LINESTRING。

返回类型

GEOMETRY

如果 geom 为 null，则返回 null。

如果 geom 为空，则返回 null。

如果 geom 不是 LINESTRING，则返回 null。

示例

以下 SQL 将四点 LINESTRING 的扩展的已知文本 (EWKT) 表示形式返回到 GEOMETRY 对象，并返回线串的终点。

```
SELECT ST_AsEWKT(ST_EndPoint(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0
 5)',4326)));
```

```
st_asewkt
-------------
 SRID=4326;POINT(0 5)
```

ST_Envelope

ST_Envelope 返回输入几何体的最小边界框，如下所示：

- 如果输入几何体为空，则返回的几何体是输入几何体的副本。
- 如果输入几何体的最小边界框退化为一个点，则返回的几何体是一个点。

- 如果输入几何体的最小边界框是一维的，则返回两点线串。
- 如果上述条件都不成立，则函数将返回一个顺时针方向的多边形，其顶点为最小边界框的角。

返回的几何体的空间参考系统标识符 (SRID) 与输入几何体的相同。

对于所有非空输入，该函数在输入几何体的 2D 投影上运行。

语法

```
ST_Envelope(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY

如果 geom 为 null，则返回 null。

示例

以下 SQL 将四点 LINESTRING 的已知文本 (WKT) 表示形式转换为 GEOMETRY 对象，并返回其顶点为最小边界框的角的多边形。

```
SELECT ST_AsText(ST_Envelope(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0
 10,0 0)),LINESTRING(20 10,20 0,10 0))')));
```

```
    st_astext
-----------------------------------
  POLYGON((0 0,0 10,20 10,20 0,0 0))
```

ST_Equals

如果输入几何体的 2D 投影在几何上相等，则 ST_Equals 返回 true。如果几何体具有相等的点集且其内部具有非空相交区域，则将几何体视为在几何上相等。

语法

```
ST_Equals(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。此值将与 geom1进行比较以确定它是否等于 geom1。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回一个错误。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

以下 SQL 检查两个多边形在几何上是否相等。

```
SELECT ST_Equals(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_equals
-----------
 false
```

以下 SQL 检查两个线串在几何上是否相等。

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(1 0,10 0)'), ST_GeomFromText('LINESTRING(1
 0,5 0,10 0)'));
```

```
st_equals
-----------
 true
```

ST_ExteriorRing

ST_ExteriorRing 返回一个表示输入面外环的闭合线串。返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_ExteriorRing(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY子类型 的 LINESTRING。

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom 为 null，则返回 null。

如果 geom 不是多边形，则返回 null。

如果 geom 为空，则返回一个空的面。

示例

以下 SQL 以闭合线串形式返回面的外环。

```
SELECT ST_AsEWKT(ST_ExteriorRing(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17
 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12
 14,15 14,13 11,12 14))')));
```

```
st_asewkt
-----------
 LINESTRING(7 9,8 7,11 6,15 8,16 6,17 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9)
```

ST_Force2D

ST_Force2D 返回输入几何体的 2D 几何体。对于 2D 几何体，将返回输入的副本。对于 3DZ、3DM 和 4D 几何体，ST_Force2D 将几何体投影到 xy 笛卡尔平面。输入几何体中的空点仍然是输出几何体中的空点。

语法

```
ST_Force2D(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY.

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom 为 null，则返回 null。

如果 geom 为空，则返回一个空的几何体。

示例

以下 SQL 从 3DZ 几何体返回 2D 几何体。

```
SELECT ST_AsEWKT(ST_Force2D(ST_GeomFromText('MULTIPOINT Z(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```
st_asewkt
-----------
  MULTIPOINT((0 1),EMPTY,(2 3),(5 6))
```

ST_Force3D

ST_Force3D 是 ST_Force3DZ 的别名。有关更多信息，请参阅 ST_Force3DZ。

ST_Force3DM

ST_Force3DM 返回输入几何体的 3DM 几何体。对于 2D 几何体，输出几何体中非空点的 m 坐标全部设置为 0。对于 3DM 几何体，将返回输入几何体的副本。对于 3DZ 几何体，几何体将投影到 xy 笛卡尔平面，而输出几何体中的非空点的 m 坐标将全部设置为 0。对于 4D 几何体，几何体将投影到 xym 笛卡尔空间。输入几何体中的空点仍然是输出几何体中的空点。

语法

```
ST_Force3DM(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY.

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom 为 null，则返回 null。

如果 geom 为空，则返回一个空的几何体。

示例

以下 SQL 从 3DZ 几何体中返回 3DM 几何体。

```
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromText('MULTIPOINT Z(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```
st_asewkt
-----------
  MULTIPOINT M ((0 1 0),EMPTY,(2 3 0),(5 6 0))
```

ST_Force3DZ

ST_Force3DZ 从输入几何体返回 3DZ 几何体。对于 2D 几何体，输出几何体中非空点的 z 坐标全部设置为 0。对于 3DM 几何体，几何体投影在 xy 笛卡尔平面上，输出几何体中非空点的 z 坐标全部设置为 0。对于 3DZ 几何体，将返回输入几何体的副本。对于 4D 几何体，几何体将投影到 xyz 笛卡尔空间。输入几何体中的空点仍然是输出几何体中的空点。

语法

```
ST_Force3DZ(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY.

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom 为 null，则返回 null。

如果 geom 为空，则返回一个空的几何体。

示例

以下 SQL 从 3DM 几何体返回 3DZ 几何体。

```
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromText('MULTIPOINT M(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```
st_asewkt
```

```
-----------
  MULTIPOINT Z ((0 1 0),EMPTY,(2 3 0),(5 6 0))
```

ST_Force4D

ST_Force4D 返回输入几何体的 4D 几何体。对于 2D 几何体，输出几何体中非空点的 z 和 m 坐标全部设置为 0。对于 3DM 几何体，输出几何体中非空点的 z 坐标全部设置为 0。对于 3DZ 几何体，输出几何体中非空点的 m 坐标全部设置为 0。对于 4D 几何体，将返回输入几何体的副本。输入几何体中的空点仍然是输出几何体中的空点。

语法

```
ST_Force4D(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY.

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom 为 null，则返回 null。

如果 geom 为空，则返回一个空的几何体。

示例

以下 SQL 从 3DM 几何体返回 4D 几何体。

```
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromText('MULTIPOINT M(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```
st_asewkt
-----------
  MULTIPOINT ZM ((0 1 0 2),EMPTY,(2 3 0 4),(5 6 0 7))
```

ST_GeoHash

ST_GeoHash 以指定精度返回输入点的 geohash 表示形式。默认精度值为 20。有关 geohash 的定义的更多信息，请参阅维基百科中的 Geohash。

语法

```
ST_GeoHash(geom)
```

```
ST_GeoHash(geom, precision)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

精度

一个 数据类型的 值INTEGER 默认值为 20。

返回类型

GEOMETRY

该函数返回输入点的 geohash 表示形式。

如果输入点为空，该函数将返回 null。

如果输入几何体不是一个点，该函数将返回错误。

示例

以下 SQL 返回输入点的 geohash 表示形式。

```
SELECT ST_GeoHash(ST_GeomFromText('POINT(45 -45)'), 25) AS geohash;
```

```
          geohash
---------------------------
 m0000000000000000000000gzz
```

以下 SQL 返回 null，因为输入点为空。

```
SELECT ST_GeoHash(ST_GeomFromText('POINT EMPTY'), 10) IS NULL AS result;
```

```
 result
---------
 true
```

ST_GeogFromText

ST_GeogFromText 从输入几何体的已知文本 (WKT) 或扩展的已知文本 (EWKT) 表示形式构造几何体对象。

语法

```
ST_GeogFromText(wkt_string)
```

参数

wkt_string

一个 VARCHAR 数据类型的值，它是地理的 WKT 或 EWKT 表示形式。

返回类型

GEOGRAPHY

如果 SRID 值已设置为输入中提供的值。如果未提供 SRID，则将其设置为 4326。

如果 wkt_string 为 null，则返回 null。

如果 wkt_string 无效，则返回一个错误。

示例

以下 SQL 从地理对象构造一个具有 SRID 值的多边形。

```
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4324;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))'));
```

```
  st_asewkt
-----------------------------------------------
 SRID=4324;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))
```

以下 SQL 从地理对象构造一个多边形。将 SRID 值设置为 4326。

```
SELECT ST_AsEWKT(ST_GeogFromText('POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))'));
```

```
 st_asewkt
-----------------------------------------------
 SRID=4326;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))
```

ST_GeogFromWKB

ST_GeogFromWKB 从输入地理的十六进制已知二进制 (WKB) 表示形式构造地理对象。

语法

```
ST_GeogFromWKB(wkb_string)
```

参数

wkb_string

一个 VARCHAR 数据类型的值，它是地理的十六进制 WKB 表示形式。

返回类型

GEOGRAPHY

如果提供了 SRID 值，则将其设置为所提供的值。如果未提供 SRID，则将其设置为 4326。

如果 wkb_string 为 null，则返回 null。

如果 wkb_string 无效，则返回一个错误。

示例

以下 SQL 从十六进制 WKB 值构造了一个地理。

```
SELECT
 ST_AsEWKT(ST_GeogFromWKB('0103000000010000000500000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'
```

```
 st_asewkt
-------------------------------------------
 SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
```

ST_GeometryN

ST_GeometryN 返回由输入几何体的输入索引指向的几何体，如下所示：

- 当输入是点、线串或多边形时，如果索引等于一 (1)，则按原样返回几何体；如果索引不是一 (1)，则返回 null。
- 如果输入是多点、多线串、多边形或几何体集合，则返回由输入索引指向的点、线串、多边形或几何体集合。

索引是从 1 开始的。结果的空间参考系统标识符 (SRID) 与输入几何体的相同。返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_GeometryN(geom, index)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

index

一个 INTEGER 数据类型的值，表示从 1 开始的索引的位置。

返回类型

GEOMETRY

如果 geom 或 index 为 null，则返回 null。

如果 index 超出范围，则返回一个错误。

示例

以下 SQL 返回几何体集合中的几何体。

```
WITH tmp1(idx) AS (SELECT 1 UNION SELECT 2),
tmp2(g) AS (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0
 0)),LINESTRING(20 10,20 0,10 0))'))
SELECT idx, ST_AsEWKT(ST_GeometryN(g, idx)) FROM tmp1, tmp2 ORDER BY idx;
```

```
 idx |          st_asewkt
-----+-----------------------------
   1 | POLYGON((0 0,10 0,0 10,0 0))
   2 | LINESTRING(20 10,20 0,10 0)
```

ST_GeometryType

ST_GeometryType 以字符串形式返回输入几何体的子类型。

对于 3DM、3DZ 和 4D 几何体输入，ST_GeometryType 返回的结果与 2D 几何体输入的结果相同。

语法

```
ST_GeometryType(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

VARCHAR，表示 geom 的子类型。

如果 geom 为 null，则返回 null。

返回的值如下所示。

返回的字符串值	几何体子类型
ST_Point	在 geom 为 POINT 子类型时返回
ST_LineString	在 geom 为 LINESTRING 子类型时返回
ST_Polygon	在 geom 为 POLYGON 子类型时返回
ST_MultiPoint	在 geom 为 MULTIPOINT 子类型时返回
ST_MultiLineString	在 geom 为 MULTILINESTRING 子类型时返回
ST_MultiPolygon	在 geom 为 MULTIPOLYGON 子类型时返回
ST_GeometryCollection	在 geom 为 GEOMETRYCOLLECTION 子类型时返回

示例

以下 SQL 返回输入线串几何体的子类型。

```
SELECT ST_GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'));
```

```
st_geometrytype
-------------
 ST_LineString
```

ST_GeomFromEWKB

ST_GeomFromEWKB 从输入几何体的扩展的已知二进制 (EWKB) 表示形式构造几何体对象。

ST_GeomFromEWKB 接受以 WKB 和 EWKB 十六进制格式编写的 3DZ、3DM 和 4D 几何体。

语法

```
ST_GeomFromEWKB(ewkb_string)
```

参数

ewkb_string

一个 VARCHAR 数据类型的值，它是几何体的十六进制 EWKB 表示形式。

返回类型

GEOMETRY

如果 ewkb_string 为 null，则返回 null。

如果 ewkb_string 无效，则返回一个错误。

示例

以下 SQL 从一个 EWKB 值构造一个多边形，并返回一个多边形的 EWKT 表示形式。

```
SELECT
 ST_AsEWKT(ST_GeomFromEWKB('0103000020E6100000010000000500000000000000000000000000000000000000000
```

```
 st_asewkt
--------------------------------
 SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
```

ST_GeomFromEWKT

ST_GeomFromEWKT 从输入几何体的扩展的已知文本 (EWKT) 表示形式构造几何体对象。

ST_GeomFromEWKT 接受 3DZ、3DM 和 4D，其中几何类型分别以 Z、M 或 ZM 作为前缀。

语法

```
ST_GeomFromEWKT(ewkt_string)
```

参数

ewkt_string

数据类型 VARCHAR 的或计算结果为 VARCHAR 类型的表达式，即几何体的 EWKT 表示形式。

您可以使用 WKT 关键字 EMPTY 指定一个空点、一个带有空点的多点或一个带有空点的几何体集合。以下示例将创建空点。

```
ST_GeomFromEWKT('SRID=4326;POINT EMPTY');
```

返回类型

GEOMETRY

如果 ewkt_string 为 null，则返回 null。

如果 ewkt_string 无效，则返回一个错误。

示例

以下 SQL 通过 EWKT 值构造多线串并返回几何体。它还返回几何体的 ST_AsEWKT 结果。

```
SELECT ST_GeomFromEWKT('SRID=4326;MULTILINESTRING((1 0,1 0),(2 0,3 0),(4 0,5 0,6 0))')
 as geom, ST_AsEWKT(geom);
```

```
                                                    geom

                                          |                          st_asewkt

-----------------------------------------------------------------------------------------------
+-----------------------------------------------------------
 0105000020E6100000030000000102000000020000000000000000F03F000000000000000000000000000000F03F00
 | SRID=4326;MULTILINESTRING((1 0,1 0),(2 0,3 0),(4 0,5 0,6 0))
```

ST_GeomFromGeoHash

ST_GeomFromGeoHash 从输入几何体的 geohash 表示形式构造几何体对象。ST_GeomFromGeoHash 返回空间参考标识符 (SRID) 为零 (0) 的二维 (2D) 几何体。有关 geohash 格式的更多信息，请参阅 Wikipedia 中的 Geohash。

语法

```
ST_GeomFromGeoHash(geohash_string)
```

```
ST_GeomFromGeoHash(geohash_string, precision)
```

参数

geohash_string

数据类型为 VARCHAR 的值或计算结果为 VARCHAR 类型的表达式，即几何体的 geohash 表示形式。

精度

数据类型 INTEGER 的值，表示 geohash 的精度。该值是要用作精度的 geohash 的字符数。如果未指定该值、值小于零或大于 geohash_string 长度，则使用 geohash_string 长度。

返回类型

GEOMETRY

如果 geohash_string 为 null，则返回 null。

如果 geohash_string 无效，则返回一个错误。

示例

以下 SQL 返回具有高精度的多边形。

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0'));
```

```
 st_asewkt
-----------------------
 POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816
 36.114646,-115.172816 36.114646))
```

以下 SQL 返回具有高精度的点。

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz00'));
```

```
 st_asewkt
-----------------------
 POINT(-115.172816 36.114646)
```

以下 SQL 返回具有低精度的多边形。

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qq'));
```

```
 st_asewkt
-----------------------
 POLYGON((-115.3125 35.15625,-115.3125 36.5625,-113.90625 36.5625,-113.90625
 35.15625,-115.3125 35.15625))
```

以下 SQL 返回精度为 3 的多边形。

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 3));
```

```
 st_asewkt
-----------------------
 POLYGON((-115.3125 35.15625,-115.3125 36.5625,-113.90625 36.5625,-113.90625
 35.15625,-115.3125 35.15625))
```

ST_GeomFromGeoJSON

ST_GeomFromGeoJSON 从输入几何体的 GeoJSON 表示形式构造几何体对象。有关 GeoJSON 格式的更多信息，请参阅 Wikipedia 中的 GeoJSON。

如果至少有一个点具有三个或更多坐标，则生成的几何体为 3DZ，其中只有两个坐标的点的 Z 分量为零。如果输入 GeoJSON 中的所有点都包含两个坐标或为空，则 ST_GeomFromGeoJSON 将返回 2D 几何体。返回的几何体的空间参考标识符 (SRID) 始终为 4326。

语法

```
ST_GeomFromGeoJSON(geojson_string)
```

参数

geojson_string

数据类型为 VARCHAR 的值或计算结果为 VARCHAR 类型的表达式，即几何体的 GeoJSON 表示形式。

返回类型

GEOMETRY

如果 geojson_string 为 null，则返回 null。

如果 geojson_string 无效，则返回一个错误。

示例

以下 SQL 返回以输入 GeoJSON 表示的 2D 几何体。

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[1,2]}'));
```

```
 st_asewkt
-----------------------
 SRID=4326;POINT(1 2)
```

以下 SQL 返回以输入 GeoJSON 表示的 3DZ 几何体。

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"LineString","coordinates":[[1,2,3],
[4,5,6],[7,8,9]]}'));
```

```
 st_asewkt
-----------------------------------------
 SRID=4326;LINESTRING Z (1 2 3,4 5 6,7 8 9)
```

当输入 GeoJSON 中只有一个点具有三个坐标，而所有其他点具有两个坐标时，以下 SQL 返回 3DZ 几何体。

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"Polygon","coordinates":[[[0, 0],[0, 1,
 8],[1, 0],[0, 0]]]}'));
```

```
 st_asewkt
-------------------------------------------------
 SRID=4326;POLYGON Z ((0 0 0,0 1 8,1 0 0,0 0 0))
```

ST_GeomFromGeoSquare

ST_GeomFromGeoSquare 返回一个几何体，该几何体覆盖由输入 geosquare 值表示的区域。返回的几何体始终是二维的。要计算 geosquare 值，请参阅 ST_GeoSquare。

语法

```
ST_GeomFromGeoSquare(geosquare)
```

```
ST_GeomFromGeoSquare(geosquare, max_depth)
```

参数

geosquare

一个数据类型为 BIGINT 的值或一个计算结果为 BIGINT 类型的 geosquare 值的表达式，该值描述了为达到目标正方形，而在初始域上进行细分的顺序。此值由 ST_GeoSquare 计算。

max_depth

一个数据类型为 INTEGER 的值，表示在初始域上进行的最大域细分次数。此值必须等于或大于 1。

返回类型

GEOMETRY

如果 geosquare 无效，则该函数会返回错误。

如果输入 max_depth 不在范围内，则该函数会返回错误。

示例

以下 SQL 从 geosquare 值返回几何体。

```
SELECT ST_AsText(ST_GeomFromGeoSquare(797852));
```

```
 st_astext
-----------------------------------------------------------------------------------------------------------
 POLYGON((13.359375 52.3828125,13.359375 52.734375,13.7109375 52.734375,13.7109375
 52.3828125,13.359375 52.3828125))
```

以下 SQL 从 geosquare 值返回几何体，最大深度为 3。

```
SELECT ST_AsText(ST_GeomFromGeoSquare(797852, 3));
```

```
 st_astext
--------------------------------------
 POLYGON((0 45,0 90,45 90,45 45,0 45))
```

以下 SQL 首先通过将 x 坐标指定为经度，将 y 坐标指定为纬度 (-122.3, 47.6) 来计算西雅图的 geosquare 值，然后为 geosquare 返回多边形。尽管输出是二维几何体，但它可用于根据经度和纬度计算空间数据。

```
SELECT ST_AsText(ST_GeomFromGeoSquare(ST_GeoSquare(ST_Point(-122.3, 47.6))));
```

```
 st_astext
-----------------------------------------------------------------------------------------------------------
POLYGON((-122.335167014971 47.6080129947513,-122.335167014971
 47.6080130785704,-122.335166931152 47.6080130785704,-122.335166931152
 47.6080129947513,-122.335167014971 47.6080129947513))
```

ST_GeomFromText

ST_GeomFromText 从输入几何体的已知文本 (WKT) 表示形式构造几何体对象。

ST_GeomFromText 接受 3DZ、3DM 和 4D，其中几何类型分别以 Z、M 或 ZM 作为前缀。

语法

```
ST_GeomFromText(wkt_string)
```

```
ST_GeomFromText(wkt_string, srid)
```

参数

wkt_string

一个 VARCHAR 数据类型的值，它是几何体的 WKT 表示形式。

您可以使用 WKT 关键字 EMPTY 指定一个空点、一个带有空点的多点或一个带有空点的几何体集合。以下示例创建一个多点，其中包括一个空点和一个非空点。

```
ST_GeomFromEWKT('MULTIPOINT(1 0,EMPTY)');
```

srid

一个 INTEGER 数据类型的值，它是空间参考标识符 (SRID)。如果提供了一个 SRID 值，则返回的几何体将具有此 SRID 值。否则，返回的几何体的 SRID 值将设置为零 (0)。

返回类型

GEOMETRY

如果 wkt_string 或 srid 为 null，则返回 null。

如果 srid 为负，则返回 null。

如果 wkt_string 无效，则返回一个错误。

如果 srid 无效，则返回一个错误。

示例

以下 SQL 从 WKT 表示形式和 SRID 值构造几何体对象。

```
SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326);
```

```
st_geomfromtext
--------------------------------
0103000020E6100000010000000500000000000000000000000000000000000000000000000000F0
```

ST_GeomFromWKB

ST_GeomFromWKB 从输入几何体的十六进制已知二进制 (WKB) 表示形式构造几何体对象。

ST_GeomFromWKB 接受以 WKB 十六进制格式编写的 3DZ、3DM 和 4D 几何体。

语法

```
ST_GeomFromWKB(wkb_string)
```

```
ST_GeomFromWKB(wkb_string, srid)
```

参数

wkb_string

一个 VARCHAR 数据类型的值，它是几何体的十六进制 WKB 表示形式。

srid

一个 INTEGER 数据类型的值，它是空间参考标识符 (SRID)。如果提供了一个 SRID 值，则返回的几何体将具有此 SRID 值。否则，返回的几何体的 SRID 值将设置为 0。

返回类型

GEOMETRY

如果 wkb_string 或 srid 为 null，则返回 null。

如果 srid 为负，则返回 null。

如果 wkb_string 无效，则返回一个错误。

如果 srid 无效，则返回一个错误。

示例

以下 SQL 从一个 WKB 值构造一个多边形，并返回一个多边形的 WKT 表示形式。

```
SELECT
 ST_AsText(ST_GeomFromWKB('010300000001000000050000000000000000000000000000000000000000000000
```

```
 st_astext
-------------------------------
 POLYGON((0 0,0 1,1 1,1 0,0 0))
```

ST_GeoSquare

ST_GeoSquare 以递归方式将域 ([-180, 180], [-90, 90]) 细分为相等的正方形区域，称为 geosquare，直至指定深度。细分基于所提供的一个点的位置。包含该点的 geosquare 在每一步都被细分，直至达到最大深度。这个 geosquare 的选择是稳定的，即，该函数的结果仅取决于输入参数。该函数将返回一个唯一值，用于标识该点所在的最终 geosquare。

ST_GeoSquare 接受一个 POINT 值，其 x 坐标表示经度，y 坐标表示纬度。经度和纬度分别限制为 [-180, 180] 和 [-90, 90]。ST_GeoSquare 的输出可以用作 ST_GeomFromGeoSquare 函数的输入。

地球沿赤道一周的弧线是 360°，且地球被分为两个半球（东半球和西半球），每个半球的经线（子午线）都是从 0°到 180°。按照惯例，投影到笛卡尔平面上的 x 轴时，东经线为“+”（正）坐标，西经线为“-”（负）坐标。在地球的 0°赤道线以北和以南分别有 90°纬线，每条纬线都与地球的 0°赤道线平行。按照惯例，投影到笛卡尔平面时，北纬线与“+”（正）y 轴相交，南纬线与“-”（负）y 轴相交。由经线和纬线相交形成的球形网格被转换为投影到笛卡尔平面上的网格，在笛卡尔平面上具有标准的正负 x 坐标和正负 y 坐标。

ST_GeoSquare 的目的是用相等的代码值标记相近的点。位于同一 geosquare 中的点将获得相同的代码值。geosquare 用于将地理坐标（纬度和经度）编码为整数。较大的区域被划分为网格，以在地图上以不同的分辨率描绘一个区域。geosquare 可用于空间索引、空间分组、邻近搜索、位置搜索和创建唯一的地点标识符。ST_GeoHash 函数遵循类似的过程，将区域划分为网格，但编码不同。

语法

```
ST_GeoSquare(geom)
```

```
ST_GeoSquare(geom, max_depth)
```

参数

geom

一个数据类型为 GEOMETRY 的 POINT 值，或一个计算结果为 POINT 子类型的表达式。该点的 x 坐标（经度）必须在以下范围内：-180 – 180。该点的 y 坐标（纬度）必须在以下范围内：-90 – 90。

max_depth

数据类型 INTEGER 的值。包含该点的域被以递归方式细分的最大次数。该值必须是介于 1 到 32 之间的整数。默认值为 32。细分的实际最终次数小于或等于指定的 max_depth。

返回类型

BIGINT

该函数返回一个唯一值，用于标识输入点所在的最终 geosquare。

如果输入 geom 不是点，则该函数将返回错误。

如果输入点为空，则返回值不是 ST_GeomFromGeoSquare 函数的有效输入。使用 ST_IsEmpty 函数可防止使用空点调用 ST_GeoSquare。

如果输入点不在范围内，则该函数将返回错误。

如果输入 max_depth 超出范围，则该函数将返回错误。

示例

以下 SQL 从输入点返回 geosquare。

```
SELECT ST_GeoSquare(ST_Point(13.5, 52.5));
```

```
  st_geosquare
-----------------------
 -4410772491521635895
```

以下 SQL 从输入点返回 geosquare，最大深度为 10。

```
SELECT ST_GeoSquare(ST_Point(13.5, 52.5), 10);
```

```
 st_geosquare
--------------
 797852
```

ST_InteriorRingN

ST_InteriorRingN 返回与索引位置处输入面的内环相对应的闭合线串。返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_InteriorRingN(geom, index)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

index

一个 INTEGER 数据类型的值，表示从 1 开始的索引环的位置。

返回类型

GEOMETRY子类型 的 LINESTRING。

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom 或 index 为 null，则返回 null。

如果 index 超出范围，则返回 null。

如果 geom 不是多边形，则返回 null。

如果 geom 为空面，则返回 null。

示例

以下 SQL 以闭合线串形式返回面的第二个环。

```
SELECT ST_AsEWKT(ST_InteriorRingN(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17
 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12
 14,15 14,13 11,12 14))'),2));
```

```
st_asewkt
-----------
 LINESTRING(12 14,15 14,13 11,12 14)
```

ST_Intersects

如果两个输入几何体的 2D 投影至少有一个共同点，则 ST_Intersects 返回 true。

语法

```
ST_Intersects(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

以下 SQL 检查第一个多边形是否与第二个多边形相交。

```
SELECT ST_Intersects(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,2 5,5 5,5
 2,2 2))'), ST_GeomFromText('MULTIPOINT((4 4),(6 6))'));
```

```
st_intersects
-------------
 true
```

ST_Intersection

ST_Interction 返回一个表示两个几何体的点集交集的几何体。也就是说，它返回两个输入几何体间共享的部分。

语法

```
ST_Intersection(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY

如果 geom1 和 geom2 不共享任何空间（它们不相交），则返回一个空的几何体。

如果 geom1 或 geom2 为空，则返回一个空的几何体。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

如果 geom1 或 geom2 非二维 (2D) 几何体，则返回一个错误。

示例

以下 SQL 返回表示两个输入几何体交集的非空几何体。

```
SELECT ST_AsEWKT(ST_Intersection(ST_GeomFromText('polygon((0 0,100 100,0 200,0 0))'),
 ST_GeomFromText('polygon((0 0,10 0,0 10,0 0))')));
```

```
        st_asewkt
-------------------------
 POLYGON((0 0,0 10,5 5,0 0))
```

传递不相交（无交集）输入几何体时，以下 SQL 返回一个空几何体。

```
SELECT ST_AsEWKT(ST_Intersection(ST_GeomFromText('linestring(0 100,0 0)'),
 ST_GeomFromText('polygon((1 0,10 0,1 10,1 0))')));
```

```
    st_asewkt
------------------
 LINESTRING EMPTY
```

ST_IsPolygonCCW

如果输入面或多面的 2D 投影是逆时针方向，则 ST_IsPolygonCCW 返回 true。如果输入几何体是点、线串、多点或多线串，则返回 true。对于几何体集合，如果集合中的所有几何体都是逆时针方向，则 ST_IsPolygonCCW 返回 true。

语法

```
ST_IsPolygonCCW(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom 为 null，则返回 null。

示例

以下 SQL 检查面是否为逆时针方向。

```
SELECT ST_IsPolygonCCW(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18
 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13
 11,12 14))'));
```

```
 st_ispolygonccw
----------
 true
```

ST_IsPolygonCW

如果输入面或多面的 2D 投影是顺时针方向，则 ST_IsPolygonCW 会返回 true。如果输入几何体是点、线串、多点或多线串，则返回 true。对于几何体集合，如果集合中的所有几何体都是顺时针方向，则 ST_IsPolygonCW 返回 true。

语法

```
ST_IsPolygonCW(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom 为 null，则返回 null。

示例

以下 SQL 检查面是否为顺时针方向。

```
SELECT ST_IsPolygonCW(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18
 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13
 11,12 14))'));
```

```
 st_ispolygonccw
----------
 true
```

ST_IsClosed

如果输入几何体的 2D 投影已闭合，则 ST_IsClosed 返回 true。以下规则定义闭合的几何体：

- 输入的几何体是一个点或一个多点。
- 输入几何体是一个线串，并且该线串的起点和终点是重合的。
- 输入几何体是一个非空的多线串，并且其所有线串均已闭合。
- 输入几何体是一个非空多边形，所有多边形的环都是非空的，并且所有环的起点和终点都是重合的。
- 输入几何体是一个非空的多边形集合，并且其所有多边形均已闭合。
- 输入几何体是一个非空几何体集合，并且其所有组件均已闭合。

语法

```
ST_IsClosed(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom 是一个空点，则返回 false。

如果 geom 为 null，则返回 null。

示例

以下 SQL 检查多边形是否已闭合。

```
SELECT ST_IsClosed(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_isclosed
-----------
 true
```

ST_IsCollection

如果输入几何体为下列子类型之一，则 ST_IsCollection 返回 true：GEOMETRYCOLLECTION、MULTIPOINT、MULTILINESTRING 或 MULTIPOLYGON。

语法

```
ST_IsCollection(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom 为 null，则返回 null。

示例

以下 SQL 检查多边形是否为一个集合。

```
SELECT ST_IsCollection(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_iscollection
-----------
 false
```

ST_IsEmpty

如果输入几何体是空的，则 ST_IsEmpty 返回 true。如果几何体至少包含一个非空点，则该几何体不为空。

如果输入几何体至少有一个非空点，ST_IsEmpty 返回 true。

语法

```
ST_IsEmpty(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom 为 null，则返回 null。

示例

以下 SQL 检查指定的多边形是否为空。

```
SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_isempty
-----------
 false
```

ST_IsRing

如果输入线串为环形，则 ST_IsRing 返回 true。如果线串闭合且简单，则为环形。

语法

```
ST_IsRing(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。几何体必须是 LINESTRING。

返回类型

BOOLEAN

如果 geom 不是 LINESTRING，则返回一个错误。

示例

以下 SQL 检查指定的线串是否为环形。

```
SELECT ST_IsRing(ST_GeomFromText('linestring(0 0, 1 1, 1 2, 0 0)'));
```

```
st_isring
-----------
 true
```

ST_IsSimple

如果输入几何体的 2D 投影很简单，则 ST_IsSimple 返回 true。有关简单几何体的定义的更多信息，请参阅几何简单性。

语法

```
ST_IsSimple(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom 为 null，则返回 null。

示例

以下 SQL 检查指定的线串是否简单。在本例中，它并不简单，因为它具有自交集。

```
SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(0 0,10 0,5 5,5 -5)'));
```

```
 st_issimple
-----------
 false
```

ST_IsValid

如果输入几何体的 2D 投影有效，ST_IsValid 返回 true。有关有效几何体的定义的更多信息，请参阅几何有效性。

语法

```
ST_IsValid(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom 为 null，则返回 null。

示例

以下 SQL 检查指定的面是否有效。在此示例中，面无效，因为面的内部并非简单地连接。

```
SELECT ST_IsValid(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 0,10 5,5 10,0
 5,5 0))'));
```

```
 st_isvalid
------------
 false
```

ST_Length

对于线性几何体，ST_Length 返回 2D 投影的笛卡尔长度。长度单位与用于表示输入几何体坐标的单位相同。对于点、多点和平面几何体，此函数返回零 (0)。当输入为几何体集合时，此函数返回集合中的几何体长度之和。

对于地理，ST_Length 返回由 SRID 确定的在椭球体上计算的输入线性地理的 2D 投影的测地线长度。长度以米为单位。对于点、多点和平面地理，此函数返回零 (0)。当输入为几何体集合时，此函数返回集合中的地理长度之和。

语法

```
ST_Length(geo)
```

参数

geo

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。

返回类型

DOUBLE PRECISION

如果 geo 为 null，则返回 null。

如果找不到 SRID 值，则返回一个错误。

示例

以下 SQL 返回多线串的笛卡尔长度。

```
SELECT ST_Length(ST_GeomFromText('MULTILINESTRING((0 0,10 0,0 10),(10 0,20 0,20
 10))'));
```

```
st_length
--------------------------------
  44.142135623731
```

以下 SQL 返回几何体中线串的长度。

```
SELECT ST_Length(ST_GeogFromText('SRID=4326;LINESTRING(5 0,6 0,4 0)'));
```

```
 st_length
------------------
 333958.472379804
```

以下 SQL 返回几何体中点的长度。

```
SELECT ST_Length(ST_GeogFromText('SRID=4326;POINT(4 5)'));
```

```
 st_length
-----------
 0
```

ST_LengthSphere

ST_LengthSphere 返回线性几何体的长度（以米为单位）。对于点、多点和平面几何体，ST_LengthSphere 返回 0。对于几何体集合，ST_LengthSphere 将返回集合中线性几何体的总长度（以米为单位）。

ST_LengthSphere 将输入几何体的每个点的坐标解释为经度和纬度（以度为单位）。对于 3DZ、3DM 或 4D 几何体，仅使用前两个坐标。

语法

```
ST_LengthSphere(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

DOUBLE PRECISION 长度（以米为单位）。长度计算基于地球的球状模型，其半径为地球世界大地测量系统 (WGS) 84 椭球模型的地球平均半径。

如果 geom 为 null，则返回 null。

示例

以下示例 SQL 计算线串的长度（以米为单位）。

```
SELECT ST_LengthSphere(ST_GeomFromText('LINESTRING(10 10,45 45)'));
```

```
 st_lengthsphere
------------------
 5127736.08292556
```

ST_Length2D

ST_Length2D 是 ST_Length 的别名。有关更多信息，请参阅 ST_Length。

ST_LineFromMultiPoint

ST_LineFromMultiPoint 返回输入多点几何体中的线串。点的顺序将保留。返回的几何体的空间参考系统标识符 (SRID) 与输入几何体的相同。返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_LineFromMultiPoint(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 MULTIPOINT。

返回类型

GEOMETRY

如果 geom 为 null，则返回 null。

如果 geom 为空，则返回空的 LINESTRING。

如果 geom 包含空点，则忽略这些空点。

如果 geom 不是 MULTIPOINT，则返回错误。

示例

以下 SQL 从多点创建线串。

```
SELECT ST_AsEWKT(ST_LineFromMultiPoint(ST_GeomFromText('MULTIPOINT(0 0,10 0,10 10,5 5,0
 5)',4326)));
```

```
 st_asewkt
--------------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 5,0 5)
```

ST_LineInterpolatePoint

ST_LineInterpolatePoint 返回一条直线上距离该线起点一小段距离的一个点。

要确定点相等性，ST_LineInterpolatePoint 对输入几何体的 2D 投影进行操作。如果输入几何体为空，则返回该几何体的维度与输入相同的副本。对于 3DZ、3DM 和 4D 几何体，z 或 m 坐标点所在线段的 z 或 m 坐标的平均值。

语法

```
ST_LineInterpolatePoint(geom, fraction)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型为 LINESTRING。

fraction

数据类型 DOUBLE PRECISION 的一个值，表示点沿着线的线串的位置。该值是介于 0—1 范围内的一个分数，包含首尾。

返回类型

GEOMETRY子类型 的 POINT。

如果 geom 或 fraction 为 null，则返回 null。

如果 geom 为空，则返回一个空点。

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 fraction 超出范围，则返回一个错误。

如果 geom 不是线串，则返回一个错误。

示例

以下 SQL 返回沿线串中途的点。

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(ST_GeomFromText('LINESTRING(0 0, 5 5, 7 7, 10
 10)'), 0.50));
```

```
st_asewkt
-----------
 POINT(5 5)
```

以下 SQL 返回沿线串行程 90% 的点。

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(ST_GeomFromText('LINESTRING(0 0, 5 5, 7 7, 10
 10)'), 0.90));
```

```
st_asewkt
-----------
 POINT(9 9)
```

ST_M

ST_M 返回输入点的 m 坐标。

语法

```
ST_M(point)
```

参数

point

一个 POINT 数据类型的 GEOMETRY 值。

返回类型

m 坐标的 DOUBLE PRECISION 值。

如果 point 为 null，则返回 null。

如果 point 是 2D 或 3DZ 点，则返回 null。

如果 point 是空点，则返回 null。

如果 point 不是 POINT，则返回一个错误。

示例

以下 SQL 返回 3DM 几何体中一个点的 m 坐标。

```
SELECT ST_M(ST_GeomFromEWKT('POINT M (1 2 3)'));
```

```
st_m
-----------
 3
```

以下 SQL 返回 4D 几何体中一个点的 m 坐标。

```
SELECT ST_M(ST_GeomFromEWKT('POINT ZM (1 2 3 4)'));
```

```
st_m
-----------
 4
```

ST_MakeEnvelope

ST_MakeEnvelope 返回一个几何体，如下所示：

- 如果输入坐标指定一个点，则返回的几何体是一个点。
- 如果输入坐标指定一条线，则返回的几何为线串。
- 否则，返回的几何体位面，其中输入坐标指定框的左下角和右上角。

返回的几何体的空间参考系统标识符 (SRID) 值（如有提供）将设置为输入 SRID 值。

语法

```
ST_MakeEnvelope(xmin, ymin, xmax, ymax)
```

```
ST_MakeEnvelope(xmin, ymin, xmax, ymax, srid)
```

参数

xmin

一个 数据类型的 值DOUBLE PRECISION 此值是框左下角的第一个坐标。

ymin

一个 数据类型的 值DOUBLE PRECISION 此值是框左下角的第二个坐标。

xmax

一个 数据类型的 值DOUBLE PRECISION 此值是框右上角的第一个坐标。

ymax

一个 数据类型的 值DOUBLE PRECISION 此值是框右上角的第二个坐标。

srid

数据类型 INTEGER 的一个值，它表示空间参考系统标识符 (SRID)。如果未提供 SRID 值，则它会设置为零。

返回类型

子类型 POINT、LINESTRING 或 POLYGON 的 GEOMETRY。

返回的几何体的 SRID 将设置为 srid，如果未设置 srid，则设置为零。

如果 xmin、ymin、xmax、ymax 或 srid 为 null，则返回 null。

如果 srid 为负，则返回一个错误。

示例

以下 SQL 会返回一个面，表示由四个输入坐标值定义的信封。

```
SELECT ST_AsEWKT(ST_MakeEnvelope(2,4,5,7));
```

```
 st_astext
---------------
 POLYGON((2 4,2 7,5 7,5 4,2 4))
```

以下 SQL 会返回一个面，表示由四个输入坐标值和一个 SRID 值定义的信封。

```
SELECT ST_AsEWKT(ST_MakeEnvelope(2,4,5,7,4326));
```

```
 st_astext
-----------------------------------
 SRID=4326;POLYGON((2 4,2 7,5 7,5 4,2 4))
```

ST_MakeLine

ST_MakeLine 从输入几何体创建线串。

返回的几何体的维度与输入几何体的维度相同。两个输入几何体必须具有相同的维度。

语法

```
ST_MakeLine(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 POINT、LINESTRING 或 MULTIPOINT。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 POINT、LINESTRING 或 MULTIPOINT。

返回类型

GEOMETRY子类型 的 LINESTRING。

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 和 geom2 为空点或包含空点，则忽略这些空点。

如果 geom1 和 geom2 为空，则返回空 LINESTRING。

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom1 和 geom2 具有不同的 SRID 值，则返回一个错误。

如果 geom1 或 geom2 不为 POINT、LINESTRING 或 MULTIPOINT，则返回一个错误。

如果 geom1 和 geom2 具有不同的维度，则返回一个错误。

示例

以下 SQL 从两个输入线串构造一个线串。

```
SELECT ST_MakeLine(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'), ST_GeomFromText('LINESTRING(88.29 39.07,88.42 39.26,88.27
 39.31,88.29 39.07)'));
```

```
st_makeline
-----------
 0102000000080000000C3F5285C8F52534052B81E85EB113D407B14AE47E15A5340C3F5285C8F423D40E17A14AE4751
```

ST_MakePoint

ST_MakePoint 返回其坐标值为输入值的点几何体。

语法

```
ST_MakePoint(x, y)
```

```
ST_MakePoint(x, y, z)
```

```
ST_MakePoint(x, y, z, m)
```

参数

x

一个 DOUBLE PRECISION 数据类型的值，该值表示第一个坐标。

y

一个 DOUBLE PRECISION 数据类型的值，该值表示第二个坐标。

z

数据类型 DOUBLE PRECISION 的一个值，该值表示第三个坐标。

m

数据类型 DOUBLE PRECISION 的一个值，该值表示第四个坐标。

返回类型

GEOMETRY子类型 的 POINT。

返回的几何体的空间参考系统标识符 (SRID) 值将设置为 0。

如果 x、y、z 或 m 为 null，则返回 null。

示例

以下 SQL 返回子类型 GEOMETRY 的 POINT 类型以及提供的坐标。

```
SELECT ST_AsText(ST_MakePoint(1,3));
```

```
st_astext
-----------
 POINT(1 3)
```

以下 SQL 返回子类型 GEOMETRY 的 POINT 类型以及提供的坐标。

```
SELECT ST_AsEWKT(ST_MakePoint(1, 2, 3));
```

```
st_asewkt
----------------
 POINT Z (1 2 3)
```

以下 SQL 返回子类型 GEOMETRY 的 POINT 类型以及提供的坐标。

```
SELECT ST_AsEWKT(ST_MakePoint(1, 2, 3, 4));
```

```
st_asewkt
-------------------
 POINT ZM (1 2 3 4)
```

ST_MakePolygon

ST_MakePolygon 具有两个可返回面的变体。一个采用单个几何体，另一个采用两个几何体。

- 第一个变体的输入是定义输出面的外环的线串。
- 第二个变体的输入是一个线串和一个多线串。两个都是空的或闭合的。

输出面外环的边界是输入线串，而面内环的边界是输入多线串中的线串。如果输入线串为空，则返回空面。多线串中的空线串将被忽略。生成的几何体的空间参考系统标识符 (SRID) 是两个输入几何体的共同 SRID。

返回的几何体的维度与输入几何体的维度相同。外环和内环必须具有相同维度。

语法

```
ST_MakePolygon(geom1)
```

```
ST_MakePolygon(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 LINESTRING。linestring 值必须是闭合的或为空。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 MULTILINESTRING。

返回类型

GEOMETRY子类型 的 POLYGON。

返回的几何体的空间参考系统标识符 (SRID) 等于输入的 SRID。

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 不是线串，则返回一个错误。

如果 geom2 不是多线串，则返回一个错误。

如果 geom1 不是闭合的，则返回一个错误。

如果 geom1 是单个点或者不是闭合的，则返回一个错误。

如果 geom2 至少包含一个具有单个点或未闭合的线串，则返回一个错误。

如果 geom1 和 geom2 具有不同的 SRID 值，则返回一个错误。

如果 geom1 和 geom2 具有不同的维度，则返回一个错误。

示例

以下 SQL 从输入线串返回多边形。

```
SELECT ST_AsText(ST_MakePolygon(ST_GeomFromText('LINESTRING(77.29 29.07,77.42
 29.26,77.27 29.31,77.29 29.07)')));
```

```
 st_astext
---------------
POLYGON((77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07))
```

以下 SQL 根据闭合线串和闭合多线串创建面。线串用于面的外环。多线串中的线串用于面的内环。

```
SELECT ST_AsEWKT(ST_MakePolygon(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,0 10,0 0)'),
 ST_GeomFromText('MULTILINESTRING((1 1,1 2,2 1,1 1),(3 3,3 4,4 3,3 3))')));
```

```
 st_astext
---------------------------------
POLYGON((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 1,1 1),(3 3,3 4,4 3,3 3))
```

ST_MemSize

ST_MemSize 返回输入几何体所使用的内存空间量（以字节为单位）。此大小取决于几何体的 Amazon Redshift 内部表示形式，因此，如果内部表示形式发生变化，则大小会发生变化。可以将此大小用作 Amazon Redshift 中几何体对象的相对大小的指示。

语法

```
ST_MemSize(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

INTEGER，表示 geom 的固有维度。

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回几何体集合的内存大小。

```
SELECT ST_MemSize(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0
 0)),LINESTRING(20 10,20 0,10 0))')::varchar + ' bytes';
```

```
 ?column?
-----------
 172 bytes
```

ST_MMax

ST_MMax 返回输入几何体的最大的 m 坐标。

语法

```
ST_MMax(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

最大的 m 坐标的 DOUBLE PRECISION 值。

如果 geom 为空，则返回 null。

如果 geom 为 null，则返回 null。

如果 geom 是 2D 或 3DZ 几何体，则返回 null。

示例

以下 SQL 返回 3DM 几何体中一个线串的最大 m 坐标。

```
SELECT ST_MMax(ST_GeomFromEWKT('LINESTRING M (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_mmax
-----------
  8
```

以下 SQL 返回 4D 几何体中一个线串的最大 m 坐标。

```
SELECT ST_MMax(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_mmax
-----------
  11
```

ST_MMin

ST_MMin 返回输入几何体的最小 m 坐标。

语法

```
ST_MMin(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

最小 m 坐标的 DOUBLE PRECISION 值。

如果 geom 为空，则返回 null。

如果 geom 为 null，则返回 null。

如果 geom 是 2D 或 3DZ 几何体，则返回 null。

示例

以下 SQL 返回 3DM 几何体中一个线串的最小 m 坐标。

```
SELECT ST_MMin(ST_GeomFromEWKT('LINESTRING M (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_mmin
-----------
  2
```

以下 SQL 返回 4D 几何体中一个线串的最小 m 坐标。

```
SELECT ST_MMin(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_mmin
-----------
  3
```

ST_Multi

ST_Multi 将几何体转换为相应的多类型。如果输入几何体已经是多类型或几何体集合，则返回其副本。如果输入几何体是点、线串或面，则返回包含输入几何体的多点、多线串或多面。

语法

```
ST_Multi(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

带子类型 MULTIPOINT、MULTILINESTRING、MULTIPOLYGON 或 GEOMETRYCOLLECTION 的 GEOMETRY。

返回的几何体的空间参考系统标识符 (SRID) 与输入几何体的相同。

如果 geom 为 null，则返回 null。

示例

以下 SQL 从输入多点返回多点。

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('MULTIPOINT((1 2),(3 4))', 4326)));
```

```
    st_asewkt
-----------------------------------
  SRID=4326;MULTIPOINT((1 2),(3 4))
```

以下 SQL 从输入点返回多点。

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('POINT(1 2)', 4326)));
```

```
    st_asewkt
-----------------------------------
  SRID=4326;MULTIPOINT((1 2))
```

以下 SQL 返回输入几何体集合中返回几何体集合。

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('GEOMETRYCOLLECTION(POINT(1 2),MULTIPOINT((1
 2),(3 4)))', 4326)));
```

```
    st_asewkt
-----------------------------------
  SRID=4326;GEOMETRYCOLLECTION(POINT(1 2),MULTIPOINT((1 2),(3 4)))
```

ST_NDims

ST_NDims 返回几何体的坐标维度。ST_NDims 不考虑几何体的拓扑维度。相反，它会根据几何体的维度返回一个常量值。

语法

```
ST_NDims(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

INTEGER，表示 geom 的固有维度。

如果 geom 为 null，则返回 null。

返回的值如下所示。

返回的值	输入几何体的维度
2	2D
3	3DZ 或 3DM
4	4D

示例

以下 SQL 返回 2D 线串的维数。

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING(0 0,1 1,2 2,0 0)'));
```

```
st_ndims
-------------
 2
```

以下 SQL 返回 3DZ 线串的维数。

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING Z(0 0 3,1 1 3,2 2 3,0 0 3)'));
```

```
st_ndims
-------------
 3
```

以下 SQL 返回 3DM 线串的维数。

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING M(0 0 4,1 1 4,2 2 4,0 0 4)'));
```

```
st_ndims
-------------
 3
```

以下 SQL 返回 4D 线串的维数。

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING ZM(0 0 3 4,1 1 3 4,2 2 3 4,0 0 3 4)'));
```

```
st_ndims
-------------
 4
```

ST_NPoints

ST_NPoints 返回输入几何体或地理中的非空点数量。

语法

```
ST_NPoints(geo)
```

参数

geo

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。

返回类型

INTEGER

如果 geo 为空点，则返回 0。

如果 geo 为 null，则返回 null。

示例

以下 SQL 返回线串中的点数。

```
SELECT ST_NPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_npoints
-------------
 4
```

以下 SQL 返回几何体中线串中的点数。

```
SELECT ST_NPoints(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
st_npoints
-------------
 4
```

ST_NRings

ST_NRings 返回输入几何体中的环形数。

语法

```
ST_NRings(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

INTEGER

如果 geom 为 null，则返回 null。

返回的值如下所示。

返回的值	几何体子类型
0	在 geom 为 POINT、LINESTRING 、MULTIPOINT 或 MULTILINESTRING 子类型时返回
环形的数量。	在 geom 为 POLYGON 或 MULTIPOLYGON 子类型时返回
所有组件中的环形数	在 geom 为 GEOMETRYCOLLECTION 子类型时返回

示例

以下 SQL 返回多边形集合中的环形数。

```
SELECT ST_NRings(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((0 0,-10 0,0
 -10,0 0)))'));
```

```
 st_nrings
-------------
 2
```

ST_NumGeometries

ST_NumGeometries 返回输入几何体中的几何体数。

语法

```
ST_NumGeometries(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

INTEGER，表示 geom 中的几何体数。

如果 geom 为 null，则返回 null。

如果 geom 是单个空几何体，则返回 0。

如果 geom 是单个非空几何体，则返回 1。

如果 geom 为 GEOMETRYCOLLECTION 或 MULTI 子类型，则返回几何体数。

示例

以下 SQL 返回输入线串集合中的几何体数量。

```
SELECT ST_NumGeometries(ST_GeomFromText('MULTILINESTRING((0 0,1 0,0 5),(3 4,13 26))'));
```

```
st_numgeometries
-------------
 2
```

ST_NumInteriorRings

ST_NumInteriorRings 返回输入多边形几何体中的环形数。

语法

```
ST_NumInteriorRings(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

INTEGER

如果 geom 为 null，则返回 null。

如果 geom 不是多边形，则返回 null。

示例

以下 SQL 返回输入多边形中的内环数。

```
SELECT ST_NumInteriorRings(ST_GeomFromText('POLYGON((0 0,100 0,100 100,0 100,0 0),(1
 1,1 5,5 1,1 1),(7 7,7 8,8 7,7 7))'));
```

```
 st_numinteriorrings
-------------
 2
```

ST_NumPoints

ST_NumPoints 返回输入几何体中的点数。

语法

```
ST_NumPoints(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

INTEGER

如果 geom 为 null，则返回 null。

如果 geom 不属于子类型 LINESTRING，则返回 null。

示例

以下 SQL 返回输入线串中的点数。

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'));
```

```
st_numpoints
-------------
4
```

以下 SQL 返回 null，因为输入 geom 不属于子类型 LINESTRING。

```
SELECT ST_NumPoints(ST_GeomFromText('MULTIPOINT(1 2,3 4)'));
```

```
st_numpoints
-------------

```

ST_Perimeter

对于输入平面几何体，ST_Perimeter 返回 2D 投影的笛卡尔周长（边界长度）。周长单位与用于表示输入几何体坐标的单位相同。对于点、多点和线性几何体，此函数返回零 (0)。当输入为几何体集合时，此函数返回集合中的几何体周长之和。

对于输入平面地理，ST_Piameter 返回由 SRID 确定的在椭球体上计算的输入平面地理的 2D 投影的测地线周长（边界长度）。周长以米为单位。对于点、多点和线性地理，此函数返回零 (0)。当输入为几何体集合时，此函数返回集合中的地理周长之和。

语法

```
ST_Perimeter(geo)
```

参数

geo

一个 GEOMETRY 或 GEOGRAPHY 数据类型的值，或一个计算结果为 GEOMETRY 或 GEOGRAPHY 类型的表达式。

返回类型

DOUBLE PRECISION

如果 geo 为 null，则返回 null。

如果找不到 SRID 值，则返回一个错误。

示例

以下 SQL 返回多边形的笛卡尔周长。

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((10 0,20 0,20
 10,10 0)))'));
```

```
 st_perimeter
------------------------------
    68.2842712474619
```

以下 SQL 返回多边形的笛卡尔周长。

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((10 0,20 0,20
 10,10 0)))'));
```

```
 st_perimeter
-------------------------------
    68.2842712474619
```

以下 SQL 返回地理中的多边形周长。

```
SELECT ST_Perimeter(ST_GeogFromText('SRID=4326;POLYGON((0 0,1 0,0 1,0 0))'));
```

```
 st_perimeter
------------------
 378790.428393693
```

以下 SQL 返回地理中线串的周长。

```
SELECT ST_Perimeter(ST_GeogFromText('SRID=4326;LINESTRING(5 0,10 0)'));
```

```
 st_perimeter
--------------
 0
```

ST_Perimeter2D

ST_Perimeter2D 是 ST_Perimeter 的别名。有关更多信息，请参阅 ST_Perimeter。

ST_Point

ST_Point 从输入坐标值返回点几何体。

语法

```
ST_Point(x, y)
```

参数

x

一个 DOUBLE PRECISION 数据类型的值，该值表示第一个坐标。

y

一个 DOUBLE PRECISION 数据类型的值，该值表示第二个坐标。

返回类型

GEOMETRY子类型 的 POINT。

返回的几何体的空间参考系统标识符 (SRID) 值将设置为 0。

如果 x 或 y 为 null，则返回 null。

示例

以下 SQL 从输入坐标构造点几何体。

```
SELECT ST_AsText(ST_Point(5.0, 7.0));
```

```
st_astext
-------------
POINT(5 7)
```

ST_PointN

ST_PointN 返回由索引值指定的线串中的点。负索引值从线串的末尾开始倒计数，因此 -1 是最后一个点。

返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_PointN(geom, index)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 LINESTRING。

index

一个 INTEGER 数据类型的值，表示线串中的点的索引。

返回类型

GEOMETRY子类型 的 POINT。

返回的几何体的空间参考系统标识符 (SRID) 值将设置为 0。

如果 geom 或 index 为 null，则返回 null。

如果 index 超出范围，则返回 null。

如果 geom 为空，则返回 null。

如果 geom 不为 LINESTRING，则返回 null。

示例

以下 SQL 将六点 LINESTRING 的扩展的已知文本 (EWKT) 表示形式返回到 GEOMETRY 对象，并返回线串的索引 5 的点。

```
SELECT ST_AsEWKT(ST_PointN(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5,0
 0)',4326), 5));
```

```
st_asewkt
-------------
 SRID=4326;POINT(0 5)
```

ST_Points

ST_Points 返回包含输入几何体中所有非空点的多点几何体。ST_Points 不会移除输入中重复的点，包括环形几何体的起点和终点。

语法

```
ST_Points(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY子类型 的 MULTIPOINT。

返回的几何体的空间参考系统标识符 (SRID) 值与 geom 相同。

如果 geom 为 null，则返回 null。

如果 geom 为空，则返回一个空的多点。

示例

以下 SQL 示例根据输入几何体构建多点几何体。结果产生包含输入几何体中的非空点的多点几何体。

```
SELECT ST_AsEWKT(ST_Points(ST_SetSRID(ST_GeomFromText('LINESTRING(1 0,2 0,3 0)'),
 4326)));
```

```
st_asewkt
-------------
SRID=4326;MULTIPOINT((1 0),(2 0),(3 0))
```

```
SELECT ST_AsEWKT(ST_Points(ST_SetSRID(ST_GeomFromText('MULTIPOLYGON(((0 0,1 0,0 1,0
 0)))'), 4326)));
```

```
st_asewkt
-------------
SRID=4326;MULTIPOINT((0 0),(1 0),(0 1),(0 0))
```

ST_Polygon

ST_Polygon 返回一个多边形几何体，其外部环形是输入线串，其值是空间参考系统标识符 (SRID) 的输入值。

返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_Polygon(linestring, srid)
```

参数

linestring

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是表示线串的 LINESTRING。linestring 值必须是闭合的。

srid

一个 INTEGER 数据类型的值，该值表示一个 SRID。

返回类型

GEOMETRY子类型 的 POLYGON。

返回的几何体的 SRID 值将设置为 srid。

如果 linestring 或 srid 为 null，则返回 null。

如果 linestring 不是线串，则返回一个错误。

如果 linestring 未闭合，则返回一个错误。

如果 srid 为负，则返回一个错误。

示例

以下 SQL 构造一个具有 SRID 值的多边形。

```
SELECT ST_AsEWKT(ST_Polygon(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'),4356));
```

```
st_asewkt
-------------
 SRID=4356;POLYGON((77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07))
```

ST_RemovePoint

ST_RemovePoint 返回一个线串几何体，该几何体已删除输入几何体在索引位置的点。

索引是从零开始的。结果的空间参考系统标识符 (SRID) 与输入几何体的相同。返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_RemovePoint(geom, index)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 LINESTRING。

index

一个 INTEGER 数据类型的值，表示从零开始的索引的位置。

返回类型

GEOMETRY

如果 geom 或 index 为 null，则返回 null。

如果 geom 不是子类型 LINESTRING，则返回错误。

如果 index 超出范围，则返回一个错误。索引位置的有效值为 0 到 ST_NumPoints(geom) 减 1。

示例

以下 SQL 删除线串中的最后一个点。

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_RemovePoint(g, ST_NumPoints(g) - 1)) FROM tmp;
```

```
   st_asewkt
----------------------------------------
```

```
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 5)
```

ST_Reverse

ST_Reverse 可反转线性几何体和平面几何体的顶点顺序。对于点或多点几何体，将返回原始几何体的副本。对于几何体集合，ST_Reverse 将反转集合中每个几何体的顶点顺序。

返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_Reverse(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY

返回的几何体的空间参考系统标识符 (SRID) 与输入几何体的相同。

如果 geom 为 null，则返回 null。

示例

以下 SQL 反转线串中点的顺序。

```
SELECT ST_AsEWKT(ST_Reverse(ST_GeomFromText('LINESTRING(1 0,2 0,3 0,4 0)', 4326)));
```

```
    st_asewkt
-----------------------------------
  SRID=4326;LINESTRING(4 0,3 0,2 0,1 0)
```

ST_SetPoint

ST_SetPoint 返回一个线串，该线串具有相对于索引指定的输入线串位置的更新坐标。新坐标是输入点的坐标。

返回的几何体的维度与 geom1 值的相同。如果 geom1 和 geom2 具有不同的维度，则 geom2 会投影到 geom1 的维度。

语法

```
ST_SetPoint(geom1, index, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 LINESTRING。

index

数据类型 INTEGER 的一个值，表示索引的位置。0 是指从左边开始的线串的第一个点，1 指的是第二点，依此类推。索引可以是负值。-1 是指从右边开始的线串的第一个点，-2 指的是从右边开始的线串的第二点，依此类推。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 POINT。

返回类型

GEOMETRY

如果 geom2 是空点，则会返回 geom1。

如果 geom1、geom2 或 index 为 null，则返回 null。

如果 geom1 不是线串，则返回一个错误。

如果 index 不在有效的索引范围内，则返回一个错误。

如果 geom2 不是点，则返回一个错误。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

示例

以下 SQL 返回一个新的线串，其中我们用指定的点设置了输入线串的第二个点。

```
SELECT ST_AsText(ST_SetPoint(ST_GeomFromText('LINESTRING(1 2, 3 2, 5 2, 1 2)'), 2,
 ST_GeomFromText('POINT(7 9)')));
```

```
st_astext
-------------
 LINESTRING(1 2,3 2,7 9,1 2)
```

以下 SQL 示例返回一个新的线串，其中我们用指定的点设置了线串右起第三个点（索引为负数）。

```
SELECT ST_AsText(ST_SetPoint(ST_GeomFromText('LINESTRING(1 2, 3 2, 5 2, 1 2)'), -3,
 ST_GeomFromText('POINT(7 9)')));
```

```
st_astext
-------------
 LINESTRING(1 2,7 9,5 2,1 2)
```

ST_SetSRID

ST_SetSRID 返回一个与输入几何体相同的几何体，只不过使用空间参考系统标识符 (SRID) 的输入值进行了更新。

语法

```
ST_SetSRID(geom, srid)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

srid

一个 INTEGER 数据类型的值，该值表示一个 SRID。

返回类型

GEOMETRY

返回的几何体的 SRID 值将设置为 srid。

如果 geom 或 srid 为 null，则返回 null。

如果 srid 为负，则返回一个错误。

示例

以下 SQL 设置线串的 SRID 值。

```
SELECT ST_AsEWKT(ST_SetSRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'),50));
```

```
st_asewkt
-------------
 SRID=50;LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)
```

ST_Simplify

ST_Simplify 使用带有给定容差的 Ramer-Douglas-Peucker 算法返回输入几何体的简化副本。输入几何体的拓扑结构可能不会保留。有关该算法的更多信息，请参阅 Wikipedia 中的 Ramer–Douglas–Peucker 算法。

当 ST_Simplify 计算距离以简化几何体时，ST_Simplify 会对输入几何体的 2D 投影进行操作。

语法

```
ST_Simplify(geom, tolerance)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

tolerance

数据类型 DOUBLE PRECISION 的一个值，表示 Ramer-Douglas-Peucker 算法的容差水平。如果 tolerance 是负数，则使用零。

返回类型

GEOMETRY.

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

返回的几何体的维度与输入几何体的维度相同。

如果 geom 为 null，则返回 null。

示例

以下 SQL 通过 Ramer-Douglas-Peucker 算法，使用欧几里得距离容差 1 简化了输入线串。距离的单位与几何体坐标的单位相同。

```
SELECT ST_AsEWKT(ST_Simplify(ST_GeomFromText('LINESTRING(0 0,1 2,1 1,2 2,2 1)'), 1));
```

```
 st_asewkt
-----------
LINESTRING(0 0,1 2,2 1)
```

ST_SRID

ST_SRID 返回输入几何体的空间参考系统标识符 (SRID)。有关 SRID 的更多信息，请参阅在 Amazon Redshift 中查询空间数据。

语法

```
ST_SRID(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

INTEGER，表示 geom 的 SRID 值。

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回线串的 SRID 值，该线串设置为 SRID 4326。

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)',4326));
```

```
st_srid
-------------
 4326
```

以下 SQL 返回线串的 SRID 值，该线串在构造时未设置。这导致 SRID 值为 0。

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_srid
-------------
 0
```

ST_StartPoint

ST_StartPoint 返回输入线串的第一个点。结果的空间参考系统标识符 (SRID) 值与输入几何体的相同。返回的几何体的维度与输入几何体的维度相同。

语法

```
ST_StartPoint(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。子类型必须是 LINESTRING。

返回类型

GEOMETRY

如果 geom 为 null，则返回 null。

如果 geom 为空，则返回 null。

如果 geom 不是 LINESTRING，则返回 null。

示例

以下 SQL 将四点 LINESTRING 的扩展的已知文本 (EWKT) 表示形式返回到 GEOMETRY 对象，并返回线串的起点。

```
SELECT ST_AsEWKT(ST_StartPoint(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0
 5)',4326)));
```

```
st_asewkt
-------------
 SRID=4326;POINT(0 0)
```

ST_Touches

如果两个输入几何体的 2D 投影接触，则 ST_Touches 返回 true。如果两个几何体是非空的、相交并且没有共同的内部点，则它们是接触的。

语法

```
ST_Touches(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

以下 SQL 检查多边形是否与线串接触。

```
SELECT ST_Touches(ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))'),
 ST_GeomFromText('LINESTRING(20 10,20 0,10 0)'));
```

```
 st_touches
-------------
 t
```

ST_Transform

ST_Transform 返回一个新的几何体，坐标在由输入空间参考系统标识符 (SRID) 定义的空间参考系统中转换。

语法

```
ST_Transform(geom, srid)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

srid

一个 INTEGER 数据类型的值，该值表示一个 SRID。

返回类型

GEOMETRY.

返回的几何体的 SRID 值将设置为 srid。

如果 geom 或 srid 为 null，则返回 null。

如果与输入 geom 关联的 SRID 值不存在，则返回错误。

如果 srid 不存在，则返回一个错误。

示例

以下 SQL 转换空几何体集合的 SRID。

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('GEOMETRYCOLLECTION EMPTY', 3857),
 4326));
```

```
             st_asewkt
-----------------------------------
 SRID=4326;GEOMETRYCOLLECTION EMPTY
```

以下 SQL 转换线串的 SRID。

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9,
 -22 -33)', 4326), 26918));
```

```
     st_asewkt
-----------------------------------------------------------------------------------------------
 SRID=26918;LINESTRING(73106.6977300955 15556182.9688576,14347201.5059964
 1545178.32934967,1515090.41262989 9522193.25115316,10491250.83295
 2575457.28410878,5672303.72135968 -5233682.61176205)
```

以下 SQL 转换多边形的 SRID。

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('POLYGON Z ((-10 10 -7, -65 10 -6, -10 64
 -5, -10 10 -7), (-11 11 5, -11 12 6, -12 11 7, -11 11 5))', 6989), 6317));
```

```
                                        st_asewkt
-------------------------------------------------------------------------------------------------
 SRID=6317;POLYGON Z ((6186430.2771091 -1090834.57212608
 1100247.33216237,2654831.67853801 -5693304.90741276 1100247.50581055,2760987.41750022
 -486836.575101877 5709710.44137268,6186430.2771091 -1090834.57212608
 1100247.33216237),(6146675.25029258 -1194792.63532103 1209007.1115113,6125027.87562215
 -1190584.81194058 1317403.77865723,6124888.99555252 -1301885.3455052
 1209007.49312929,6146675.25029258 -1194792.63532103 1209007.1115113))
```

ST_Union

ST_Union 返回一个表示两个几何体联合的几何体。也就是说，它合并输入几何体，以生成一个没有重叠的结果几何体。

语法

```
ST_Union(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

GEOMETRY

返回的几何体的空间参考系统标识符 (SRID) 值是输入几何体的 SRID 值。

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 或 geom2 为空，则返回一个空的几何体。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合、线串或多线串，则返回一个错误。

如果 geom1 或 geom2 非二维 (2D) 几何体，则返回一个错误。

示例

以下 SQL 返回表示两个输入几何体的联合的非空几何体。

```
SELECT ST_AsEWKT(ST_Union(ST_GeomFromText('POLYGON((0 0,100 100,0 200,0 0))'),
 ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))')));
```

```
        st_asewkt
-------------------------
 POLYGON((0 0,0 200,100 100,5 5,10 0,0 0))
```

ST_Within

如果第一个输入几何体的 2D 投影在第二个输入几何体的 2D 投影中，则 ST_Within 返回 true。

例如，如果几何体 A 中的每个点均为几何体 B 中的一个点，并且其内部有非空相交区域，则几何体 A 在几何体 B 中。

ST_Within(A, B) 与 ST_Contains(B, A) 等效。

语法

```
ST_Within(geom1, geom2)
```

参数

geom1

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。此值将与 geom2 进行比较以确定它是否在 geom2 中。

geom2

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom1 或 geom2 为 null，则返回 null。

如果 geom1 和 geom2 不具有相同的空间参考系统标识符 (SRID) 值，则返回一个错误。

如果 geom1 或 geom2 为几何体集合，则返回一个错误。

示例

以下 SQL 检查第一个多边形是否在第二个多边形中。

```
SELECT ST_Within(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_within
-----------
 true
```

ST_X

ST_X 返回输入点的第一个坐标。

语法

```
ST_X(point)
```

参数

point

一个 POINT 数据类型的 GEOMETRY 值。

返回类型

DOUBLE PRECISION第一个坐标的 值。

如果 point 为 null，则返回 null。

如果 point 是空点，则返回 null。

如果 point 不是 POINT，则返回一个错误。

示例

以下 SQL 返回点的第一个坐标。

```
SELECT ST_X(ST_Point(1,2));
```

```
st_x
-----------
 1.0
```

ST_XMax

ST_XMax 返回输入几何体的最大的第一个坐标。

语法

```
ST_XMax(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

DOUBLE PRECISION最大的第一个坐标的 值。

如果 geom 为空，则返回 null。

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回线串的最大的第一个坐标。

```
SELECT ST_XMax(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_xmax
-----------
 77.42
```

ST_XMin

ST_XMin 返回输入几何体的最小的第一个坐标。

语法

```
ST_XMin(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

DOUBLE PRECISION最小的第一个坐标的 值。

如果 geom 为空，则返回 null。

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回线串的最小的第一个坐标。

```
SELECT ST_XMin(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_xmin
-----------
 77.27
```

ST_Y

ST_Y 返回输入点的第二个坐标。

语法

```
ST_Y(point)
```

参数

point

一个 POINT 数据类型的 GEOMETRY 值。

返回类型

DOUBLE PRECISION第二个坐标的 值。

如果 point 为 null，则返回 null。

如果 point 是空点，则返回 null。

如果 point 不是 POINT，则返回一个错误。

示例

以下 SQL 返回点的第二个坐标。

```
SELECT ST_Y(ST_Point(1,2));
```

```
st_y
-----------
 2.0
```

ST_YMax

ST_YMax 返回输入几何体的最大的第二个坐标。

语法

```
ST_YMax(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

DOUBLE PRECISION最大的第二个坐标的 值。

如果 geom 为空，则返回 null。

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回线串的最大的第二个坐标。

```
SELECT ST_YMax(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_ymax
-----------
 29.31
```

ST_YMin

ST_YMin 返回输入几何体的最小的第二个坐标。

语法

```
ST_YMin(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

DOUBLE PRECISION最小的第二个坐标的 值。

如果 geom 为空，则返回 null。

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回线串的最小的第二个坐标。

```
SELECT ST_YMin(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_ymin
-----------
 29.07
```

ST_Z

ST_Z 返回输入点的 z 坐标。

语法

```
ST_Z(point)
```

参数

point

一个 POINT 数据类型的 GEOMETRY 值。

返回类型

m 坐标的 DOUBLE PRECISION 值。

如果 point 为 null，则返回 null。

如果 point 是 2D 或 3DM 点，则返回 null。

如果 point 是空点，则返回 null。

如果 point 不是 POINT，则返回一个错误。

示例

以下 SQL 返回 3DZ 几何体中一个点的 z 坐标。

```
SELECT ST_Z(ST_GeomFromEWKT('POINT Z (1 2 3)'));
```

```
st_z
-----------
 3
```

以下 SQL 返回 4D 几何体中一个点的 z 坐标。

```
SELECT ST_Z(ST_GeomFromEWKT('POINT ZM (1 2 3 4)'));
```

```
st_z
-----------
 3
```

ST_ZMax

ST_ZMax 返回输入几何体的最大的 z 坐标。

语法

```
ST_ZMax(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

最大的 z 坐标的 DOUBLE PRECISION 值。

如果 geom 为空，则返回 null。

如果 geom 为 null，则返回 null。

如果 geom 是 2D 或 3DM 几何体，则返回 null。

示例

以下 SQL 返回 3DZ 几何体中一个线串的最大 z 坐标。

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING Z (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_zmax
-----------
  8
```

以下 SQL 返回 4D 几何体中一个线串的最大 z 坐标。

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_zmax
-----------
  10
```

ST_ZMin

ST_ZMin 返回输入几何体的最小的 z 坐标。

语法

```
ST_ZMin(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

最小 z 坐标的 DOUBLE PRECISION 值。

如果 geom 为空，则返回 null。

如果 geom 为 null，则返回 null。

如果 geom 是 2D 或 3DM 几何体，则返回 null。

示例

以下 SQL 返回 3DZ 几何体中一个线串的最小 z 坐标。

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING Z (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_zmin
-----------
  2
```

以下 SQL 返回 4D 几何体中一个线串的最小 z 坐标。

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_zmin
-----------
  2
```

SupportsBBox

如果输入几何体支持使用预先计算的边界框进行编码，则 SupportsBBox 返回 true。有关对边界框的支持的更多信息，请参阅边界框。

语法

```
SupportsBBox(geom)
```

参数

geom

一个 GEOMETRY 数据类型的值，或一个计算结果为 GEOMETRY 类型的表达式。

返回类型

BOOLEAN

如果 geom 为 null，则返回 null。

示例

以下 SQL 返回 true，因为输入点几何体支持使用边界框进行编码。

```
SELECT SupportsBBox(AddBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
supportsbbox
--------------
t
```

以下 SQL 返回 false，因为输入点几何体不支持使用边界框编码。

```
SELECT SupportsBBox(DropBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
supportsbbox
--------------
f
```

字符串函数

主题

- || (串联) 运算符
- ASCII 函数
- BPCHARCMP 函数
- BTRIM 函数
- BTTEXT_PATTERN_CMP 函数
- CHAR_LENGTH 函数
- CHARACTER_LENGTH 函数

- CHARINDEX 函数
- CHR 函数
- COLLATE 函数
- CONCAT 函数
- CRC32 函数
- DIFFERENCE 函数
- INITCAP 函数
- LEFT 和 RIGHT 函数
- LEN 函数
- LENGTH 函数
- LOWER 函数
- LPAD 和 RPAD 函数
- LTRIM 函数
- OCTETINDEX 函数
- OCTET_LENGTH 函数
- POSITION 函数
- QUOTE_IDENT 函数
- QUOTE_LITERAL 函数
- REGEXP_COUNT 函数
- REGEXP_INSTR 函数
- REGEXP_REPLACE 函数
- REGEXP_SUBSTR 函数
- REPEAT 函数
- REPLACE 函数
- REPLICATE 函数
- REVERSE 函数
- RTRIM 函数
- SOUNDEX 函数
- SPLIT_PART 函数

- STRPOS 函数
- STRTOL 函数
- SUBSTRING 函数
- TEXTLEN 函数
- TRANSLATE 函数
- TRIM 函数
- UPPER 函数

字符串函数用于处理和操作字符串或计算结果为字符串的表达式。当这些函数中的 string 参数为文本值时，该参数必须括在单引号中。支持的数据类型包括 CHAR 和 VARCHAR。

以下部分提供了支持的函数的函数名称、语法和描述。对字符串的所有偏移都从 1 开始。

弃用的仅领导节点函数

以下字符串函数因为仅在领导节点上运行而遭到弃用。有关更多信息，请参阅仅领导节点函数

- GET_BYTE
- SET_BIT
- SET_BYTE
- TO_ASCII

|| (串联) 运算符

联接位于 || 符号的任意一侧的两个表达式并返回联接后的表达式。

与 CONCAT 函数相似。

> **Note**
>
> 如果一个表达式为 Null，或两个表达式都为 Null，则联接的结果为 NULL。

语法

```
expression1 || expression2
```

参数

expression1

CHAR 字符串、VARCHAR 字符串、二进制表达式或者其计算结果为这些类型之一的表达式。

expression2

CHAR 字符串、VARCHAR 字符串、二进制表达式或者其计算结果为这些类型之一的表达式。

返回类型

字符串的返回类型与输入参数的类型相同。例如，联接两个类型为 VARCHAR 的字符串会返回一个类型为 VARCHAR 的字符串。

示例

以下示例使用 TICKIT 示例数据库中的 USERS 表和 VENUE 表。有关更多信息，请参阅 示例数据库。

要联接示例数据库的 USERS 表中的 FIRSTNAME 和 LASTNAME 字段，请使用以下示例。

```
SELECT (firstname || ' ' || lastname) as fullname
FROM users
ORDER BY 1
LIMIT 10;

+-----------------+
|    fullname     |
+-----------------+
| Aaron Banks     |
| Aaron Booth     |
| Aaron Browning  |
| Aaron Burnett   |
| Aaron Casey     |
| Aaron Cash      |
| Aaron Castro    |
| Aaron Dickerson |
| Aaron Dixon     |
| Aaron Dotson    |
+-----------------+
```

要联接可能包含 null 值的列，请使用 NVL 和 COALESCE 函数 表达式。以下示例使用 NVL 以在遇到 NULL 时返回 0。

```
SELECT (venuename || ' seats ' || NVL(venueseats, 0)) as seating
FROM venue
WHERE venuestate = 'NV' or venuestate = 'NC'
ORDER BY 1
LIMIT 10;

+-------------------------------------+
|               seating               |
+-------------------------------------+
| Ballys Hotel seats 0                |
| Bank of America Stadium seats 73298 |
| Bellagio Hotel seats 0              |
| Caesars Palace seats 0              |
| Harrahs Hotel seats 0               |
| Hilton Hotel seats 0                |
| Luxor Hotel seats 0                 |
| Mandalay Bay Hotel seats 0          |
| Mirage Hotel seats 0                |
| New York New York seats 0           |
+-------------------------------------+
```

ASCII 函数

ASCII 函数返回指定字符串中第一个字符的 ASCII 代码或 Unicode 代码点。如果字符串为空，该函数返回 0。如果字符串为 null，则返回 NULL。

语法

```
ASCII('string')
```

参数

string

CHAR 字符串或 VARCHAR 字符串。

返回类型

INTEGER

示例

要返回 NULL，请使用以下示例。如果两个参数相同，NULLIF 函数将返回 NULL，因此 ASCII 函数的输入参数为 NULL。有关更多信息，请参阅 NULLIF 函数。

```
SELECT ASCII(NULLIF('',''));

+-------+
| ascii |
+-------+
|  NULL |
+-------+
```

要返回 ASCII 代码 0，请使用以下示例。

```
SELECT ASCII('');

+-------+
| ascii |
+-------+
|     0 |
+-------+
```

要返回单词 amazon 的第一个字母的 ASCII 代码 97，请使用以下示例。

```
SELECT ASCII('amazon');

+-------+
| ascii |
+-------+
|    97 |
+-------+
```

要返回单词 Amazon 的第一个字母的 ASCII 代码 65，请使用以下示例。

```
SELECT ASCII('Amazon');

+-------+
| ascii |
+-------+
|    65 |
```

```
+-------+
```

BPCHARCMP 函数

比较两个字符串的值并返回整数。如果字符串相同，此函数返回 0。如果按字母顺序，第一个字符串更靠后，则函数返回 1。如果第二个字符串较大，则函数返回 -1。

对于多字节字符，该比较基于字节编码。

BTTEXT_PATTERN_CMP 函数的同义词。

语法

```
BPCHARCMP(string1, string2)
```

参数

string1

CHAR 字符串或 VARCHAR 字符串。

string2

CHAR 字符串或 VARCHAR 字符串。

返回类型

INTEGER

示例

以下示例使用 TICKIT 示例数据库中的 USERS 表。有关更多信息，请参阅 示例数据库。

要确定 USERS 表中前十个条目的用户的名字在字母顺序上是否比其姓氏更靠后，请使用以下示例。对于 FIRSTNAME 的字符串在字母顺序上比 LASTNAME 的字符串更靠后的条目，函数返回 1。如果 LASTNAME 在字母顺序上比 FIRSTNAME 更靠后，此函数将返回 -1。

```
SELECT userid, firstname, lastname, BPCHARCMP(firstname, lastname)
FROM users
ORDER BY 1, 2, 3, 4
LIMIT 10;
```

```
+--------+-----------+-----------+-----------+
| userid | firstname | lastname  | bpcharcmp |
+--------+-----------+-----------+-----------+
|      1 | Rafael    | Taylor    |        -1 |
|      2 | Vladimir  | Humphrey  |         1 |
|      3 | Lars      | Ratliff   |        -1 |
|      4 | Barry     | Roy       |        -1 |
|      5 | Reagan    | Hodge     |         1 |
|      6 | Victor    | Hernandez |         1 |
|      7 | Tamekah   | Juarez    |         1 |
|      8 | Colton    | Roy       |        -1 |
|      9 | Mufutau   | Watkins   |        -1 |
|     10 | Naida     | Calderon  |         1 |
+--------+-----------+-----------+-----------+
```

要返回 USERS 表中该函数返回 0 的所有条目，请使用以下示例。当 FIRSTNAME 与 LASTNAME 相同时，该函数返回 0。

```
SELECT userid, firstname, lastname,
BPCHARCMP(firstname, lastname)
FROM users
WHERE BPCHARCMP(firstname, lastname)=0
ORDER BY 1, 2, 3, 4;

+--------+-----------+----------+-----------+
| userid | firstname | lastname | bpcharcmp |
+--------+-----------+----------+-----------+
|     62 | Chase     | Chase    |         0 |
|   4008 | Whitney   | Whitney  |         0 |
|  12516 | Graham    | Graham   |         0 |
|  13570 | Harper    | Harper   |         0 |
|  16712 | Cooper    | Cooper   |         0 |
|  18359 | Chase     | Chase    |         0 |
|  27530 | Bradley   | Bradley  |         0 |
|  31204 | Harding   | Harding  |         0 |
+--------+-----------+----------+-----------+
```

BTRIM 函数

BTRIM 函数通过删除前导空格和尾随空格或删除与可选的指定字符串匹配的前导字符和尾随字符来剪裁字符串。

语法

```
BTRIM(string [, trim_chars ] )
```

参数

string

要剪裁的输入 VARCHAR 字符串。

trim_chars

该 VARCHAR 字符串包含要匹配的字符。

返回类型

BTRIM 函数返回 VARCHAR 字符串。

示例

以下示例从字符串 ' abc ' 中剪裁前导和尾随空格：

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

以下示例从字符串 'xyzaxyzbxyzcxyz' 中删除前导和尾随 'xyz' 字符串。将删除前导和尾随的 'xyz'，但不会删除字符串内部的匹配字符。

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

以下示例从字符串 'setuphistorycassettes' 中删除与 trim_chars 列表 'tes' 中的任何字符相匹配的开头和结尾部分。在输入字符串开头或结尾部分，在 trim_chars 列表中未包含的其他字符之前出现的任何 t、e 或 s 都将被删除。

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
-----------------
 uphistoryca
```

BTTEXT_PATTERN_CMP 函数

BPCHARCMP 函数的同义词。

有关更多信息，请参阅 BPCHARCMP 函数。

CHAR_LENGTH 函数

LEN 函数的同义词。

请参阅 LEN 函数。

CHARACTER_LENGTH 函数

LEN 函数的同义词。

请参阅 LEN 函数。

CHARINDEX 函数

返回指定子字符串在字符串中的位置。

有关类似的函数，请参阅POSITION 函数和STRPOS 函数。

语法

```
CHARINDEX( substring, string )
```

参数

substring

要在 string 中搜索的子字符串。

string

要搜索的字符串或列。

返回类型

INTEGER

CHARINDEX 函数返回与子字符串的位置对应的 INTEGER（从 1 开始，而不是从 0 开始）。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。如果在字符串未找到子字符串，CHARINDEX 将返回 0。

示例

要显示字符串 fish 在单词 dog 中的位置，请使用以下示例。

```
SELECT CHARINDEX('fish', 'dog');

+-----------+
| charindex |
+-----------+
|         0 |
+-----------+
```

要显示字符串 fish 在单词 dogfish 中的位置，请使用以下示例。

```
SELECT CHARINDEX('fish', 'dogfish');

+-----------+
| charindex |
+-----------+
|         4 |
+-----------+
```

以下示例使用 TICKIT 示例数据库中的 SALES 表。有关更多信息，请参阅 示例数据库。

要返回 SALES 表中佣金超过 999.00 的不同销售交易的数量，请使用以下示例。此命令通过检查小数点距离佣金值开头是否超过 4 位数来计算大于 999.00 的佣金。

```
SELECT DISTINCT CHARINDEX('.', commission), COUNT (CHARINDEX('.', commission))
FROM sales
WHERE CHARINDEX('.', commission) > 4
GROUP BY CHARINDEX('.', commission)
ORDER BY 1,2;
```

```
+-----------+-------+
| charindex | count |
+-----------+-------+
|         5 |   629 |
+-----------+-------+
```

CHR 函数

CHR 函数返回与输入参数指定的 ASCII 码位值匹配的字符。

语法

```
CHR(number)
```

参数

number

输入参数是表示 ASCII 码位值的 INTEGER。

返回类型

CHAR

如果 ASCII 字符与输入值匹配，CHR 函数将返回 CHAR 字符串。如果输入数值没有匹配的 ASCII 字符，该函数将返回 NULL。

示例

要返回与 ASCII 码位 0 对应的字符，请使用以下示例。请注意，对于输入 0，CHR 函数会返回 NULL。

```
SELECT CHR(0);

+-----+
| chr |
+-----+
|     |
+-----+
```

要返回与 ASCII 码位 65 对应的字符，请使用以下示例。

```
SELECT CHR(65);

+-----+
| chr |
+-----+
| A   |
+-----+
```

要返回以大写字母 A（ASCII 码位 65）开头的独特的事件名称，请使用以下示例。以下示例使用 TICKIT 示例数据库中的 EVENT 表。有关更多信息，请参阅 示例数据库。

```
SELECT DISTINCT eventname FROM event
WHERE SUBSTRING(eventname, 1, 1)=CHR(65) LIMIT 5;

+-----------------------+
|       eventname       |
+-----------------------+
| A Catered Affair      |
| As You Like It        |
| A Man For All Seasons |
| Alan Jackson          |
| Armando Manzanero     |
+-----------------------+
```

COLLATE 函数

COLLATE 函数覆盖字符串列或表达式的排序规则。

有关如何使用数据库排序规则创建表的信息，请参阅CREATE TABLE。

有关如何使用数据库排序规则创建数据库的信息，请参阅CREATE DATABASE。

语法

```
COLLATE( string, 'case_sensitive' | 'case_insensitive');
```

参数

string

要覆盖的字符串列或表达式。

'case_sensitive' | 'case_insensitive'

排序规则名称的字符串常量。Amazon Redshift 仅支持 case_sensitive 或 case_insensitive。

返回类型

COLLATE 函数根据第一个输入表达式类型返回 VARCHAR 或 CHAR。此函数仅更改第一个输入参数的排序规则，不会更改其输出值。

示例

要创建表 T 并将表 T 中的 col1 定义为 case_sensitive，请使用以下示例。

```
CREATE TABLE T ( col1 Varchar(20) COLLATE case_sensitive );

INSERT INTO T VALUES ('john'),('JOHN');
```

当您运行第一个查询时，Amazon Redshift 仅返回 john。在 col1 上运行 COLLATE 函数后，排序规则变成 case_insensitive。第二个查询同时返回 john 和 JOHN。

```
SELECT * FROM T WHERE col1 = 'john';

+------+
| col1 |
+------+
| john |
+------+

SELECT * FROM T WHERE COLLATE(col1, 'case_insensitive') = 'john';

+------+
| col1 |
+------+
| john |
| JOHN |
+------+
```

要创建表 A 并将表 A 中的 col1 定义为 case_insensitive，请使用以下示例。

```
CREATE TABLE A ( col1 Varchar(20) COLLATE case_insensitive );
```

```
INSERT INTO A VALUES ('john'),('JOHN');
```

当您运行第一个查询时，Amazon Redshift 同时返回 john 和 JOHN。在 col1 上运行 COLLATE 函数后，排序规则变成 case_sensitive。第二个查询仅返回 john。

```
SELECT * FROM A WHERE col1 = 'john';

+------+
| col1 |
+------+
| john |
| JOHN |
+------+

SELECT * FROM A WHERE COLLATE(col1, 'case_sensitive') = 'john';

+------+
| col1 |
+------+
| john |
+------+
```

CONCAT 函数

CONCAT 函数将联接两个表达式并返回生成的表达式。要联接两个以上的表达式，请使用嵌套 CONCAT 函数。在两个表达式之间使用联接运算符（||）将生成与 CONCAT 函数相同的结果。

语法

```
CONCAT ( expression1, expression2 )
```

参数

expression1、expression2

两个参数可以是固定长度字符串、可变长度字符串、二进制表达式或计算结果为其中一个输入的表达式。

返回类型

CONCAT 返回一个表达式。表达式的数据类型与输入参数的数据类型相同。

如果输入表达式的类型不同，Amazon Redshift 会尝试隐式转换其中一个表达式类型。如果值无法转换，则会返回一个错误。

使用说明

- 对于 CONCAT 函数和联接运算符，如果一个或多个表达式为 null，则联接的结果也为 null。

示例

以下示例联接两个字符文本：

```
SELECT CONCAT('December 25, ', '2008');

concat
-------------------
December 25, 2008
(1 row)
```

以下查询（使用 || 运算符而不是 CONCAT）将生成相同的结果：

```
SELECT 'December 25, '||'2008';

?column?
-------------------
December 25, 2008
(1 row)
```

以下示例使用一个 CONCAT 函数中的另一个嵌套 CONCAT 函数来串联三个字符串：

```
SELECT CONCAT('Thursday, ', CONCAT('December 25, ', '2008'));

concat
----------------------------
Thursday, December 25, 2008
(1 row)
```

要串联可能包含 NULL 的列，请使用 NVL 和 COALESCE 函数，它会在遇到 NULL 时返回给定值。以下示例使用 NVL 在遇到 NULL 时返回 0。

```
SELECT CONCAT(venuename, CONCAT(' seats ', NVL(venueseats, 0))) AS seating
FROM venue WHERE venuestate = 'NV' OR venuestate = 'NC'
```

```
ORDER BY 1
LIMIT 5;

seating
-----------------------------------
Ballys Hotel seats 0
Bank of America Stadium seats 73298
Bellagio Hotel seats 0
Caesars Palace seats 0
Harrahs Hotel seats 0
(5 rows)
```

以下查询联接 VENUE 表中的 CITY 和 STATE 值：

```
SELECT CONCAT(venuecity, venuestate)
FROM venue
WHERE venueseats > 75000
ORDER BY venueseats;

concat
-------------------
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

以下查询使用嵌套 CONCAT 函数。该查询将联接 VENUE 表中的 CITY 和 STATE 值，但会使用逗号和空格分隔生成的字符串：

```
SELECT CONCAT(CONCAT(venuecity,', '),venuestate)
FROM venue
WHERE venueseats > 75000
ORDER BY venueseats;

concat
---------------------
Denver, CO
Kansas City, MO
East Rutherford, NJ
Landover, MD
(4 rows)
```

以下示例联接了两个二进制表达式。其中 abc 是一个二进制值（具有一个 616263 的十六进制表示形式），def 是一个二进制值（具有一个 646566 的十六进制表示形式）。结果会自动显示为二进制值的十六进制表示形式。

```
SELECT CONCAT('abc'::VARBYTE, 'def'::VARBYTE);

concat
-------------------
616263646566
```

CRC32 函数

CRC32 是一个用于错误检测的函数。此函数使用 CRC32 算法来检测源数据和目标数据之间的变化。CRC32 函数会将长度可变的字符串转换为以 32 位二进制序列的十六进制值的文本表示形式表示的 8 字符字符串。要检测源数据和目标数据之间的变化，请对源数据使用 CRC32 函数并存储输出。然后，对目标数据使用 CRC32 函数，并将该输出与来自源数据的输出进行比较。如果数据未经修改，则输出将相同；如果修改了数据，则输出将有所不同。

语法

```
CRC32(string)
```

参数

string

CHAR 字符串、VARCHAR 字符串或隐式计算为 CHAR 或 VARCHAR 类型的表达式。

返回类型

CRC32 函数返回以 32 位二进制序列的十六进制值的文本表示形式表示的 8 字符字符串。Amazon Redshift CRC32 函数基于 CRC-32C 多项式。

示例

显示字符串 Amazon Redshift 的 8 位值。

```
SELECT CRC32('Amazon Redshift');

+----------+
```

```
|  crc32   |
+----------+
| f2726906 |
+----------+
```

DIFFERENCE 函数

DIFFERENCE 函数比较两个字符串的美国 Soundex 代码。该函数返回 INTEGER，以指示 Soundex 代码之间匹配的字符数。

Soundex 代码是一个长度为四个字符的字符串。Soundex 代码表示单词的发音方式，而不是其拼写方式。例如，Smith 和 Smyth 具有相同的 Soundex 代码。

语法

```
DIFFERENCE(string1, string2)
```

参数

string1

CHAR 字符串、VARCHAR 字符串或隐式计算为 CHAR 或 VARCHAR 类型的表达式。

string2

CHAR 字符串、VARCHAR 字符串或隐式计算为 CHAR 或 VARCHAR 类型的表达式。

返回类型

INTEGER

DIFFERENCE 函数返回 0–4 之间的一个 INTEGER 值，该值计算两个字符串的美国 Soundex 代码中匹配字符的数量。Soundex 代码具有 4 个字符，因此，当字符串的所有 4 个字符的美国 Soundex 代码值都相同时，DIFFERENCE 函数返回 4。如果两个字符串中有一个为空，则 DIFFERENCE 返回 0。如果两个字符串都不包含有效字符，则此函数返回 1。DIFFERENCE 函数仅转换英文字母小写或大写 ASCII 字符，包括 a–z 和 A–Z。DIFFERENCE 将忽略其他字符。

示例

要比较字符串 % 和 @ 的 Soundex 值，请使用以下示例。因为这两个字符串都不包含有效字符，所以此函数返回 1。

```
SELECT DIFFERENCE('%', '@');

+------------+
| difference |
+------------+
|          1 |
+------------+
```

要比较 Amazon 和一个空字符串的 Soundex 值，请使用以下示例。因为两个字符串中有一个是空的，所以该函数返回 0。

```
SELECT DIFFERENCE('Amazon', '');

+------------+
| difference |
+------------+
|          0 |
+------------+
```

要比较字符串 Amazon 和 Ama 的 Soundex 值，请使用以下示例。因为字符串的 Soundex 值中有 2 个字符是相同的，所以该函数返回 2。

```
SELECT DIFFERENCE('Amazon', 'Ama');

+------------+
| difference |
+------------+
|          2 |
+------------+
```

要比较字符串 Amazon 和 +-*/%Amazon 的 Soundex 值，请使用以下示例。因为字符串的 Soundex 值中所有 4 个字符都是相同的，所以该函数返回 4。请注意，该函数会忽略第二个字符串中的无效字符 +-*/%。

```
SELECT DIFFERENCE('Amazon', '+-*/%Amazon');

+------------+
| difference |
+------------+
|          4 |
```

```
+------------+
```

要比较字符串 AC/DC 和 Ay See Dee See 的 Soundex 值，请使用以下示例。因为字符串的 Soundex 值中所有 4 个字符都是相同的，所以该函数返回 4。

```
SELECT DIFFERENCE('AC/DC', 'Ay See Dee See');

+------------+
| difference |
+------------+
|          4 |
+------------+
```

INITCAP 函数

将指定字符串中的每个单词的第一个字母大写。INITCAP 支持 UTF-8 多字节字符，并且每个字符最多可以有 4 个字节。

语法

```
INITCAP(string)
```

参数

string

CHAR 字符串、VARCHAR 字符串或隐式计算为 CHAR 或 VARCHAR 类型的表达式。

返回类型

VARCHAR

使用说明

INITCAP 函数会将字符串中的每个单词的第一个字母大写，并将所有后续字母小写。因此，务必了解哪些字符（空格字符除外）充当分隔符。文字分隔符 字符是任何非字母数字字符，包括标点符号、普通符号和控制字符。所有以下字符都是文字分隔符：

```
! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~
```

制表符、换行符、换页符和回车也是文字分隔符。

示例

以下示例使用 TICKIT 示例数据库的 CATEGORY 表和 USERS 表中的数据。有关更多信息，请参阅示例数据库。

要使 CATDESC 列中每个单词的首字母大写，请使用以下示例。

```
SELECT catid, catdesc, INITCAP(catdesc)
FROM category
ORDER BY 1, 2, 3;

+-------+--------------------------------------------
+--------------------------------------------+
| catid |                  catdesc                   |                  initcap
          |
+-------+--------------------------------------------
+--------------------------------------------+
|     1 | Major League Baseball                      | Major League Baseball
          |
|     2 | National Hockey League                     | National Hockey League
          |
|     3 | National Football League                   | National Football League
          |
|     4 | National Basketball Association            | National Basketball Association
          |
|     5 | Major League Soccer                        | Major League Soccer
          |
|     6 | Musical theatre                            | Musical Theatre
          |
|     7 | All non-musical theatre                    | All Non-Musical Theatre
          |
|     8 | All opera and light opera                  | All Opera And Light Opera
          |
|     9 | All rock and pop music concerts            | All Rock And Pop Music Concerts
          |
|    10 | All jazz singers and bands                 | All Jazz Singers And Bands
          |
|    11 | All symphony, concerto, and choir concerts | All Symphony, Concerto, And
 Choir Concerts |
+-------+--------------------------------------------
+--------------------------------------------+
```

要显示在大写字符不作为单词的首字母时 INITCAP 函数不保留这些字符，请使用以下示例。例如，字符串 MLB 变成 Mlb。

```
SELECT INITCAP(catname)
FROM category
ORDER BY catname;

+-----------+
|  initcap  |
+-----------+
| Classical |
| Jazz      |
| Mlb       |
| Mls       |
| Musicals  |
| Nba       |
| Nfl       |
| Nhl       |
| Opera     |
| Plays     |
| Pop       |
+-----------+
```

要显示除空格以外的非字母数字字符用作单词分隔符，请使用以下示例。每个字符串中的几个字母将为大写。

```
SELECT email, INITCAP(email)
FROM users
ORDER BY userid DESC LIMIT 5;

+------------------------------------+------------------------------------+
|               email                |              initcap               |
+------------------------------------+------------------------------------+
| urna.Ut@egetdictumplacerat.edu     | Urna.Ut@Egetdictumplacerat.Edu     |
| nibh.enim@egestas.ca               | Nibh.Enim@Egestas.Ca               |
| in@Donecat.ca                      | In@Donecat.Ca                      |
| sodales@blanditviverraDonec.ca     | Sodales@Blanditviverradonec.Ca     |
| sociis.natoque.penatibus@vitae.org | Sociis.Natoque.Penatibus@Vitae.Org |
+------------------------------------+------------------------------------+
```

LEFT 和 RIGHT 函数

这些函数返回指定数量的位于字符串最左侧或最右侧的字符。

该数量基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。

语法

```
LEFT( string,  integer )

RIGHT( string,  integer )
```

参数

string

CHAR 字符串、VARCHAR 字符串或任何计算为 CHAR 或 VARCHAR 字符串的表达式。

integer

一个正整数。

返回类型

VARCHAR

示例

以下示例使用 TICKIT 示例数据库的 EVENT 表中的数据。有关更多信息，请参阅 示例数据库。

要返回事件 ID 在 1000 和 1005 之间的事件名称中最左侧的 5 个字符和最右侧的 5 个字符，请使用以下示例。

```
SELECT eventid, eventname,
LEFT(eventname,5) AS left_5,
RIGHT(eventname,5) AS right_5
FROM event
WHERE eventid BETWEEN 1000 AND 1005
ORDER BY 1;

+---------+----------------+--------+---------+
| eventid |   eventname    | left_5 | right_5 |
```

```
+---------+----------------+--------+---------+
|    1000 | Gypsy          | Gypsy  | Gypsy   |
|    1001 | Chicago        | Chica  | icago   |
|    1002 | The King and I | The K  | and I   |
|    1003 | Pal Joey       | Pal J  | Joey    |
|    1004 | Grease         | Greas  | rease   |
|    1005 | Chicago        | Chica  | icago   |
+---------+----------------+--------+---------+
```

LEN 函数

以字符数形式返回指定字符串的长度。

语法

LEN 是 LENGTH 函数、CHAR_LENGTH 函数、CHARACTER_LENGTH 函数和 TEXTLEN 函数的同义词。

```
LEN(expression)
```

参数

expression

CHAR 字符串、VARCHAR 字符串、VARBYTE 表达式或隐式计算为 CHAR、VARCHAR 或 VARBYTE 类型的表达式。

返回类型

INTEGER

LEN 函数返回一个整数，表示输入字符串中的字符的数量。

如果输入的是字符串，LEN 函数将返回多字节字符串中的字符的实际数量，而不是字节的数量。例如，存储 3 个 4 字节中文字符需要 VARCHAR(12) 列。LEN 函数将对同一字符串返回 3。要获取字符串长度（以字节为单位），请使用 OCTET_LENGTH 函数。

使用说明

如果 expression 为 CHAR 字符串，则不计算尾随空格。

如果 expression 为 VARCHAR 字符串，则计算尾随空格。

示例

要返回字符串 `français` 中的字节数和字符数，请使用以下示例。

```
SELECT OCTET_LENGTH('français'),
LEN('français');

+--------------+-----+
| octet_length | len |
+--------------+-----+
|            9 |   8 |
+--------------+-----+
```

要在不使用 OCTET_LENGTH 函数的情况下返回字符串 `français` 中的字节数和字符数，请使用以下示例。有关更多信息，请参阅 CAST 函数。

```
SELECT LEN(CAST('français' AS VARBYTE)) as bytes, LEN('français');

+-------+-----+
| bytes | len |
+-------+-----+
|     9 |   8 |
+-------+-----+
```

要返回字符串 `cat`（没有尾随空格）、`cat `（有三个尾随空格）、`cat `（有三个尾随空格，强制转换为长度为 6 的 CHAR）以及 `cat `（有三个尾随空格，强制转换为长度为 6 的 VARCHAR）中的字符数，请使用以下示例。请注意，该函数不计算 CHAR 字符串的尾随空格，但的确计算 VARCHAR 字符串的尾随空格。

```
SELECT LEN('cat'), LEN('cat   '), LEN(CAST('cat   ' AS CHAR(6))) AS len_char,
 LEN(CAST('cat   ' AS VARCHAR(6))) AS len_varchar;

+-----+-----+----------+-------------+
| len | len | len_char | len_varchar |
+-----+-----+----------+-------------+
|   3 |   6 |        3 |           6 |
+-----+-----+----------+-------------+
```

以下示例使用 TICKIT 示例数据库的 VENUE 表中的数据。有关更多信息，请参阅 示例数据库。

要返回 VENUE 表中最长的 10 个场地名称，请使用以下示例。

```
SELECT venuename, LEN(venuename)
FROM venue
ORDER BY 2 DESC, 1
LIMIT 10;

+----------------------------------------+-----+
|                venuename               | len |
+----------------------------------------+-----+
| Saratoga Springs Performing Arts Center |  39 |
| Lincoln Center for the Performing Arts  |  38 |
| Nassau Veterans Memorial Coliseum       |  33 |
| Jacksonville Municipal Stadium          |  30 |
| Rangers BallPark in Arlington           |  29 |
| University of Phoenix Stadium           |  29 |
| Circle in the Square Theatre            |  28 |
| Hubert H. Humphrey Metrodome            |  28 |
| Oriole Park at Camden Yards             |  27 |
| Dick's Sporting Goods Park              |  26 |
+----------------------------------------+-----+
```

LENGTH 函数

LEN 函数的同义词。

请参阅 LEN 函数。

LOWER 函数

将字符串转换为小写。LOWER 支持 UTF-8 多字节字符，并且每个字符最多可以有 4 个字节。

语法

```
LOWER(string)
```

参数

string

VARCHAR 字符串或任何计算结果为 VARCHAR 类型的表达式。

返回类型

字符串

LOWER 函数返回与输入字符串具有相同数据类型的字符串。例如，如果输入是 CHAR 字符串，该函数将返回 CHAR 字符串。

示例

以下示例使用 TICKIT 示例数据库的 CATEGORY 表中的数据。有关更多信息，请参阅 示例数据库。

要将 CATNAME 列中的 VARCHAR 字符串转换为小写，请使用以下示例。

```
SELECT catname, LOWER(catname) FROM category ORDER BY 1,2;

+-----------+-----------+
|  catname  |   lower   |
+-----------+-----------+
| Classical | classical |
| Jazz      | jazz      |
| MLB       | mlb       |
| MLS       | mls       |
| Musicals  | musicals  |
| NBA       | nba       |
| NFL       | nfl       |
| NHL       | nhl       |
| Opera     | opera     |
| Plays     | plays     |
| Pop       | pop       |
+-----------+-----------+
```

LPAD 和 RPAD 函数

这些函数根据指定长度在字符串前面或后面追加字符。

语法

```
LPAD(string1, length, [ string2 ])
```

```
RPAD(string1, length, [ string2 ])
```

参数

string1

CHAR 字符串、VARCHAR 字符串或隐式计算为 CHAR 或 VARCHAR 类型的表达式。

length

一个用于定义函数结果的长度的整数。字符串的长度基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。如果 string1 的长度超过指定长度，它将被截断（在右侧）。如果 length 为零或负数，则函数的结果将为空字符串。

string2

（可选）追加到 string1 前面或后面的一个或多个字符。如果未指定此参数，则使用空格。

返回类型

VARCHAR

示例

以下示例使用 TICKIT 示例数据库的 EVENT 表中的数据。有关更多信息，请参阅 示例数据库。

要将指定的一组事件名称截断到 20 个字符，并在短于此长度的名称前面追加空格，请使用以下示例。

```
SELECT LPAD(eventname, 20) FROM event
WHERE eventid BETWEEN 1 AND 5 ORDER BY 1;

+----------------------+
|         lpad         |
+----------------------+
|              Salome |
|        Il Trovatore |
|       Boris Godunov |
|     Gotterdammerung |
|La Cenerentola (Cind |
+----------------------+
```

要将相同的一组事件名称截断到 20 个字符，并在短于此长度的名称后面追加 0123456789，请使用以下示例。

```
SELECT RPAD(eventname, 20,'0123456789') FROM event
```

```
WHERE eventid BETWEEN 1 AND 5 ORDER BY 1;

+----------------------+
|         rpad         |
+----------------------+
| Boris Godunov0123456 |
| Gotterdammerung01234 |
| Il Trovatore01234567 |
| La Cenerentola (Cind |
| Salome01234567890123 |
+----------------------+
```

LTRIM 函数

从字符串的开头剪裁几个字符。删除只包含剪裁字符列表中的字符的最长字符串。当输入字符串中没有了剪裁字符时，剪裁即告完成。

语法

```
LTRIM( string [, trim_chars] )
```

参数

string

要剪裁的字符串列、表达式或字符串文本。

trim_chars

表示要从 string 的开头剪裁的字符的字符串列、表达式或字符串文本。如果未指定，则使用空格作为剪裁字符。

返回类型

LTRIM 函数返回与输入字符串（CHAR 或 VARCHAR）具有相同数据类型的字符串。

示例

以下示例从 `listime` 列中剪裁掉年份。字符串文本中的剪裁字符 `'2008-'` 表示要从左侧剪裁的字符。如果您使用剪裁字符 `'028-'`，则会获得相同的结果。

```
select listid, listtime, ltrim(listtime, '2008-')
from listing
```

```
order by 1, 2, 3
limit 10;

listid |      listtime       |     ltrim
-------+---------------------+---------------
     1 | 2008-01-24 06:43:29 | 1-24 06:43:29
     2 | 2008-03-05 12:25:29 | 3-05 12:25:29
     3 | 2008-11-01 07:35:33 | 11-01 07:35:33
     4 | 2008-05-24 01:18:37 | 5-24 01:18:37
     5 | 2008-05-17 02:29:11 | 5-17 02:29:11
     6 | 2008-08-15 02:08:13 | 15 02:08:13
     7 | 2008-11-15 09:38:15 | 11-15 09:38:15
     8 | 2008-11-09 05:07:30 | 11-09 05:07:30
     9 | 2008-09-09 08:03:36 | 9-09 08:03:36
    10 | 2008-06-17 09:44:54 | 6-17 09:44:54
```

当 trim_chars 中的任意字符出现在 string 的开头时，LTRIM 都会予以删除。以下示例从 VENUENAME（VARCHAR 列）的开头剪裁字符“C”、“D”和“G”。

```
select venueid, venuename, ltrim(venuename, 'CDG')
from venue
where venuename like '%Park'
order by 2
limit 7;

venueid | venuename                  | btrim
--------+----------------------------+--------------------------
    121 | ATT Park                   | ATT Park
    109 | Citizens Bank Park         | itizens Bank Park
    102 | Comerica Park              | omerica Park
      9 | Dick's Sporting Goods Park | ick's Sporting Goods Park
     97 | Fenway Park                | Fenway Park
    112 | Great American Ball Park   | reat American Ball Park
    114 | Miller Park                | Miller Park
```

以下示例使用从 venueid 列中检索到的剪裁字符 2。

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
```

```
008-01-24 06:43:29
```

以下示例不剪裁任何字符，因为在 '0' 剪裁字符之前找到了 2。

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

以下示例使用默认的空格剪裁字符，从字符串的开头剪裁掉两个空格。

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

OCTETINDEX 函数

OCTETINDEX 函数以字节数形式返回子字符串在字符串中的位置。

语法

```
OCTETINDEX(substring, string)
```

参数

substring

CHAR 字符串、VARCHAR 字符串或隐式计算为 CHAR 或 VARCHAR 类型的表达式。

string

CHAR 字符串、VARCHAR 字符串或隐式计算为 CHAR 或 VARCHAR 类型的表达式。

返回类型

INTEGER

OCTETINDEX 函数会以字节数的形式返回一个 INTEGER 值，该值与 substring 在 string 中的位置相对应，其中 string 中的第一个字符被计数为 1。如果 string 不包含多字节字符，则结果等于

CHARINDEX 函数的结果。如果 string 不包含 substring，则该函数返回 0。如果 substring 为空，该函数返回 1。

示例

要返回字符串 Amazon Redshift 中子字符串 q 的位置，请使用以下示例。因为 substring 不在 string 中，所以此示例返回 0。

```
SELECT OCTETINDEX('q', 'Amazon Redshift');

+------------+
| octetindex |
+------------+
|          0 |
+------------+
```

要返回空子字符串在字符串 Amazon Redshift 中的位置，请使用以下示例。因为 substring 为空，所以此示例返回 1。

```
SELECT OCTETINDEX('', 'Amazon Redshift');

+------------+
| octetindex |
+------------+
|          1 |
+------------+
```

要返回字符串 Amazon Redshift 中子字符串 Redshift 的位置，请使用以下示例。因为 substring 从 string 的第八个字节开始，所以此示例返回 8。

```
SELECT OCTETINDEX('Redshift', 'Amazon Redshift');

+------------+
| octetindex |
+------------+
|          8 |
+------------+
```

要返回字符串 Amazon Redshift 中子字符串 Redshift 的位置，请使用以下示例。因为 string 的前六个字符是双字节字符，所以此示例返回 21。

```
SELECT OCTETINDEX('Redshift', 'Άμαζον Amazon Redshift');

+------------+
| octetindex |
+------------+
|         21 |
+------------+
```

OCTET_LENGTH 函数

以字节数形式返回指定字符串的长度。

语法

```
OCTET_LENGTH(expression)
```

参数

expression

CHAR 字符串、VARCHAR 字符串、VARBYTE 表达式或隐式计算为 CHAR、VARCHAR 或 VARBYTE 类型的表达式。

返回类型

INTEGER

OCTET_LENGTH 函数返回一个整数，表示输入字符串中的字节数。

如果输入的是字符串，LEN 函数将返回多字节字符串中字符的实际数量，而不是字节的数量。例如，存储 3 个 4 字节中文字符需要 VARCHAR(12) 列。OCTET_LENGTH 函数对于该字符串将返回 12，而 LEN 函数将对于该同一个字符串将返回 3。

使用说明

如果 expression 是 CHAR 字符串，则此函数返回 CHAR 字符串的长度。例如，CHAR(6) 输入的输出为 CHAR(6)。

如果 expression 为 VARCHAR 字符串，则计算尾随空格。

示例

当带有三个尾随空格的字符串 francais 强制转换为 CHAR 和 VARCHAR 类型时，要返回字节数，请使用以下示例。有关更多信息，请参阅 CAST 函数。

```
SELECT OCTET_LENGTH(CAST('francais   ' AS CHAR(15))) AS octet_length_char,
 OCTET_LENGTH(CAST('francais   ' AS VARCHAR(15))) AS octet_length_varchar;

+------------------+----------------------+
| octet_length_char | octet_length_varchar |
+------------------+----------------------+
|                15 |                   11 |
+------------------+----------------------+
```

要返回字符串 français 中的字节数和字符数，请使用以下示例。

```
SELECT OCTET_LENGTH('français'), LEN('français');

+--------------+-----+
| octet_length | len |
+--------------+-----+
|            9 |   8 |
+--------------+-----+
```

要在字符串 français 强制转换为 VARBYTE 时返回字节数，请使用以下示例。

```
SELECT OCTET_LENGTH(CAST('français' AS VARBYTE));

+--------------+
| octet_length |
+--------------+
|            9 |
+--------------+
```

POSITION 函数

返回指定子字符串在字符串中的位置。

有关类似的函数，请参阅CHARINDEX 函数和STRPOS 函数。

语法

```
POSITION(substring IN string )
```

参数

substring

要在 string 中搜索的子字符串。

string

要搜索的字符串或列。

返回类型

POSITION 函数返回与子字符串的位置对应的 INTEGER（从 1 开始，而不是从 0 开始）。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。如果在字符串中未找到子字符串，POSITION 将返回 0。

示例

要显示字符串 fish 在单词 dog 中的位置，请使用以下示例。

```
SELECT POSITION('fish' IN 'dog');

+-----------+
|  position |
+-----------+
|         0 |
+-----------+
```

要显示字符串 fish 在单词 dogfish 中的位置，请使用以下示例。

```
SELECT POSITION('fish' IN 'dogfish');

+-----------+
|  position |
+-----------+
|         4 |
+-----------+
```

以下示例使用 TICKIT 示例数据库中的 SALES 表。有关更多信息，请参阅 示例数据库。

要返回 SALES 表中佣金超过 999.00 的不同销售交易的数量，请使用以下示例。此命令通过检查小数点距离佣金值开头是否超过 4 位数来计算大于 999.00 的佣金。

```
SELECT DISTINCT POSITION('.' IN commission), COUNT (POSITION('.' IN commission))
FROM sales
WHERE POSITION('.' IN commission) > 4
GROUP BY POSITION('.' IN commission)
ORDER BY 1,2;

+-----------+-------+
|  position | count |
+-----------+-------+
|         5 |   629 |
+-----------+-------+
```

QUOTE_IDENT 函数

QUOTE_IDENT 函数将指定的字符串作为一个带前导双引号和尾随双引号的字符串返回。此函数输出可用作 SQL 语句中的标识符。此函数适当地在任何嵌入式双引号之外再加上一对双引号。

QUOTE_IDENT 仅在需要时（当字符串包含非标识符字符或会转换为小写时）添加双引号，从而创建有效的标识符。要始终返回一个单引号字符串，请使用 QUOTE_LITERAL。

语法

```
QUOTE_IDENT(string)
```

参数

string

CHAR 或 VARCHAR 字符串。

返回类型

QUOTE_IDENT 函数返回与输入 string 相同类型的字符串。

示例

要返回带双引号的字符串 "CAT"，请使用以下示例。

```
SELECT QUOTE_IDENT('"CAT"');

+-------------+
| quote_ident |
+-------------+
| """CAT"""   |
+-------------+
```

以下示例使用 TICKIT 示例数据库的 CATEGORY 表中的数据。有关更多信息，请参阅 示例数据库。

要返回用引号括起的 CATNAME 列，请使用以下示例。

```
SELECT catid, QUOTE_IDENT(catname)
FROM category
ORDER BY 1,2;

+-------+-------------+
| catid | quote_ident |
+-------+-------------+
|     1 | "MLB"       |
|     2 | "NHL"       |
|     3 | "NFL"       |
|     4 | "NBA"       |
|     5 | "MLS"       |
|     6 | "Musicals"  |
|     7 | "Plays"     |
|     8 | "Opera"     |
|     9 | "Pop"       |
|    10 | "Jazz"      |
|    11 | "Classical" |
+-------+-------------+
```

QUOTE_LITERAL 函数

QUOTE_LITERAL 函数以单引号字符串的形式返回指定字符串，以便此字符串可用作 SQL 语句中的字符串文本。如果输入参数为数字，则 QUOTE_LITERAL 会将其视为字符串。请适当地在任何嵌入式单引号和反斜杠之外再加上一对双引号。

语法

```
QUOTE_LITERAL(string)
```

参数

string

CHAR 或 VARCHAR 字符串。

返回类型

QUOTE_LITERAL 函数返回与输入 string 相同数据类型的 CHAR 或 VARCHAR 字符串。

示例

要返回带单引号的字符串 ''CAT''，请使用以下示例。

```
SELECT QUOTE_LITERAL('''CAT''');

+---------------+
| quote_literal |
+---------------+
| '''CAT'''     |
+---------------+
```

以下示例使用 TICKIT 示例数据库的 CATEGORY 表中的数据。有关更多信息，请参阅 示例数据库。

要返回用单引号括起的 CATNAME 列，请使用以下示例。

```
SELECT catid, QUOTE_LITERAL(catname)
FROM category
ORDER BY 1,2;

+-------+---------------+
| catid | quote_literal |
+-------+---------------+
|     1 | 'MLB'         |
|     2 | 'NHL'         |
|     3 | 'NFL'         |
|     4 | 'NBA'         |
|     5 | 'MLS'         |
|     6 | 'Musicals'    |
|     7 | 'Plays'       |
|     8 | 'Opera'       |
|     9 | 'Pop'         |
```

```
|    10 | 'Jazz'        |
|    11 | 'Classical'   |
+-------+---------------+
```

要返回用单引号括起的 CATID 列，请使用以下示例。

```
SELECT QUOTE_LITERAL(catid), catname
FROM category
ORDER BY 1,2;

+---------------+-----------+
| quote_literal |  catname  |
+---------------+-----------+
| '1'           | MLB       |
| '10'          | Jazz      |
| '11'          | Classical |
| '2'           | NHL       |
| '3'           | NFL       |
| '4'           | NBA       |
| '5'           | MLS       |
| '6'           | Musicals  |
| '7'           | Plays     |
| '8'           | Opera     |
| '9'           | Pop       |
+---------------+-----------+
```

REGEXP_COUNT 函数

在字符串中搜索正则表达式模式，并返回指示指定模式在字符串中出现的次数的整数。如果未找到匹配项，此函数将返回 0。有关正则表达式的更多信息，请参阅 POSIX 运算符和 Wikipedia 中的 Regular expression。

语法

```
REGEXP_COUNT( source_string, pattern [, position [, parameters ] ] )
```

参数

source_string

CHAR 或 VARCHAR 字符串。

pattern

表示正则表达式模式的 UTF-8 字符串文本。有关更多信息，请参阅 POSIX 运算符。

position

（可选）指示在 source_string 中开始搜索的位置的正 INTEGER。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。默认为 1。如果 position 小于 1，则搜索从 source_string 的第一个字符开始。如果 position 大于 source_string 中字符的数量，则结果为 0。

参数

（可选）一个或多个字符串文本，指示函数与模式的匹配方式。可能的值包括：

- c – 执行区分大小写的匹配。默认情况下，使用区分大小写的匹配。
- i – 执行不区分大小写的匹配。
- p – 使用 Perl 兼容正则表达式 (PCRE) 方言解释模式。有关 PCRE 的更多信息，请参阅 Wikipedia 中的 Perl Compatible Regular Expressions。

返回类型

INTEGER

示例

要计算三个字母序列出现的次数，请使用以下示例。

```
SELECT REGEXP_COUNT('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');

+--------------+
| regexp_count |
+--------------+
|            8 |
+--------------+
```

要使用不区分大小写的匹配计算字符串 FOX 的出现次数，请使用以下示例。

```
SELECT REGEXP_COUNT('the fox', 'FOX', 1, 'i');

+--------------+
| regexp_count |
+--------------+
|            1 |
```

```
+--------------+
```

要使用以 PCRE 方言编写的模式来定位至少包含一个数值和一个小写字母的单词，请使用以下示例。此示例使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例使用区分大小写的匹配计算此类单词的出现次数。

```
SELECT REGEXP_COUNT('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

+--------------+
| regexp_count |
+--------------+
|            2 |
+--------------+
```

要使用以 PCRE 方言编写的模式来定位至少包含一个数值和一个小写字母的单词，请使用以下示例。它使用 ?= 运算符，它在 PCRE 中具有特定的含义。此示例计算此类单词的出现次数，但与前面的示例不同，因为它使用了不区分大小写的匹配。

```
SELECT REGEXP_COUNT('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');

+--------------+
| regexp_count |
+--------------+
|            3 |
+--------------+
```

以下示例使用 TICKIT 示例数据库的 USERS 表中的数据。有关更多信息，请参阅 示例数据库。

要计算顶级域名为 org 或 edu 的次数，请使用以下示例。

```
SELECT email, REGEXP_COUNT(email,'@[^.]*\.(org|edu)') FROM users
ORDER BY userid LIMIT 4;

+-----------------------------------------------+--------------+
|                     email                     | regexp_count |
+-----------------------------------------------+--------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu |            1 |
| Suspendisse.tristique@nonnisiAenean.edu       |            1 |
| amet.faucibus.ut@condimentumegetvolutpat.ca   |            0 |
```

```
| sed@lacusUtnec.ca                              |           0 |
+------------------------------------------------+-------------+
```

REGEXP_INSTR 函数

在字符串中搜索正则表达式模式并返回指示匹配子字符串的开始位置的整数。如果未找到匹配项，此函数将返回 0。REGEXP_INSTR 与 函数相似，只不过前者可让您在字符串中搜索正则表达式模式。有关正则表达式的更多信息，请参阅 POSIX 运算符和 Wikipedia 中的 Regular expression。

语法

```
REGEXP_INSTR( source_string, pattern [, position [, occurrence] [, option [, parameters
 ] ] ] ] )
```

参数

source_string

要搜索的字符串表达式（如列名称）。

pattern

表示正则表达式模式的 UTF-8 字符串文本。有关更多信息，请参阅 POSIX 运算符。

position

（可选）指示在 source_string 中开始搜索的位置的正 INTEGER。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。默认为 1。如果 position 小于 1，则搜索从 source_string 的第一个字符开始。如果 position 大于 source_string 中字符的数量，则结果为 0。

出现

（可选）一个正 INTEGER，指示要使用的模式的哪一次出现。REGEXP_INSTR 会跳过第一个 *occurrence*-1 匹配项。默认为 1。如果 occurrence 小于 1 或大于 source_string 中的字符数量，则会忽略搜索且结果为 0。

option

（可选）一个值，指示是返回匹配项的第一个字符的位置（0），还是匹配项结尾后面第一个字符的位置（1）。非零值与 1 相同。默认值为 0。

参数

（可选）一个或多个字符串文本，指示函数与模式的匹配方式。可能的值包括：

- c – 执行区分大小写的匹配。默认情况下，使用区分大小写的匹配。
- i – 执行不区分大小写的匹配。
- e – 使用子表达式提取子字符串。

  如果 pattern 包含一个子表达式，REGEXP_INSTR 会使用 pattern 中的第一个子表达式来匹配子字符串。REGEXP_INSTR 仅考虑第一个子表达式；其他子表达式会被忽略。如果模式没有子表达式，REGEXP_INSTR 会忽略“e”参数。
- p – 使用 Perl 兼容正则表达式 (PCRE) 方言解释模式。有关 PCRE 的更多信息，请参阅 Wikipedia 中的 Perl Compatible Regular Expressions。

返回类型

整数

示例

以下示例使用 TICKIT 示例数据库的 USERS 表中的数据。有关更多信息，请参阅 示例数据库。

要搜索作为域名开头的 @ 字符并返回第一个匹配项的开始位置，请使用以下示例。

```
SELECT email, REGEXP_INSTR(email, '@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

+----------------------------------------------+--------------+
|                     email                    | regexp_instr |
+----------------------------------------------+--------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu |           21 |
| Suspendisse.tristique@nonnisiAenean.edu       |           22 |
| amet.faucibus.ut@condimentumegetvolutpat.ca   |           17 |
| sed@lacusUtnec.ca                             |            4 |
+----------------------------------------------+--------------+
```

要搜索单词 Center 的变体并返回第一个匹配项的开始位置，请使用以下示例。

```
SELECT venuename, REGEXP_INSTR(venuename,'[cC]ent(er|re)$')
FROM venue
WHERE REGEXP_INSTR(venuename,'[cC]ent(er|re)$') > 0
ORDER BY venueid LIMIT 4;
```

```
+----------------------+--------------+
|       venuename      | regexp_instr |
+----------------------+--------------+
| The Home Depot Center |           16 |
| Izod Center          |            6 |
| Wachovia Center      |           10 |
| Air Canada Centre    |           12 |
+----------------------+--------------+
```

要使用不区分大小写的匹配逻辑找到字符串 FOX 第一次出现的起始位置，请使用以下示例。

```
SELECT REGEXP_INSTR('the fox', 'FOX', 1, 1, 0, 'i');

+--------------+
| regexp_instr |
+--------------+
|            5 |
+--------------+
```

要使用以 PCRE 方言编写的模式来定位至少包含一个数值和一个小写字母的单词，请使用以下示例。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例查找第二个此类单词的起始位置。

```
SELECT REGEXP_INSTR('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

+--------------+
| regexp_instr |
+--------------+
|           21 |
+--------------+
```

要使用以 PCRE 方言编写的模式来定位至少包含一个数值和一个小写字母的单词，请使用以下示例。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。本示例查找第二个此类单词的起始位置，但与前面的示例不同，因为它使用了不区分大小写的匹配。

```
SELECT REGEXP_INSTR('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');

+--------------+
| regexp_instr |
+--------------+
```

```
|           15 |
+--------------+
```

REGEXP_REPLACE 函数

在字符串中搜索正则表达式模式并将该模式的每个匹配项替换为指定字符串。REGEXP_REPLACE 与 REPLACE 函数相似，只不过前者可让您在字符串中搜索正则表达式模式。有关正则表达式的更多信息，请参阅 POSIX 运算符和 Wikipedia 中的 Regular expression。

REGEXP_REPLACE 与 TRANSLATE 函数和 REPLACE 函数相似，只不过 TRANSLATE 进行多次单字符替换，REPLACE 一次性将整个字符串替换为其他字符串，而 REGEXP_REPLACE 可让您在字符串中搜索正则表达式模式。

语法

```
REGEXP_REPLACE( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

参数

source_string

要搜索的 CHAR 或 VARCHAR 字符串表达式（如列名称）。

pattern

表示正则表达式模式的 UTF-8 字符串文本。有关更多信息，请参阅 POSIX 运算符。

replace_string

（可选）将替换模式的每次出现的 CHAR 或 VARCHAR 字符串表达式（如列名称）。默认值是空字符串 ("")。

position

（可选）指示在 source_string 中开始搜索的位置的正整数。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。默认为 1。如果 position 小于 1，则搜索从 source_string 的第一个字符开始。如果 position 大于 source_string 中的字符数量，则结果为 source_string。

参数

（可选）一个或多个字符串文本，指示函数与模式的匹配方式。可能的值包括：

- c – 执行区分大小写的匹配。默认情况下，使用区分大小写的匹配。
- i – 执行不区分大小写的匹配。
- p – 使用 Perl 兼容正则表达式 (PCRE) 方言解释模式。有关 PCRE 的更多信息，请参阅 Wikipedia 中的 Perl Compatible Regular Expressions。

返回类型

VARCHAR

如果 pattern 或 replace_string 为 NULL，则返回 NULL。

示例

要使用不区分大小写的匹配替换字符串 FOX 在值 quick brown fox 内的所有出现，请使用以下示例。

```
SELECT REGEXP_REPLACE('the fox', 'FOX', 'quick brown fox', 1, 'i');

+----------------------+
|   regexp_replace     |
+----------------------+
| the quick brown fox |
+----------------------+
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。要将此单词的每次出现替换为值 [hidden]，请使用以下示例。

```
SELECT REGEXP_REPLACE('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

+--------------------------------+
|        regexp_replace          |
+--------------------------------+
| [hidden] plain A1234 [hidden] |
+--------------------------------+
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。要将此单词的每次出现替换为值 [hidden]，但与前面的示例不同，它使用不区分大小写的匹配，请使用以下示例。

```
SELECT REGEXP_REPLACE('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

+----------------------------------+
|          regexp_replace          |
+----------------------------------+
| [hidden] plain [hidden] [hidden] |
+----------------------------------+
```

以下示例使用 TICKIT 示例数据库的 USERS 表中的数据。有关更多信息，请参阅 示例数据库。

要删除电子邮件地址中的 @ 和域名，请使用以下示例。

```
SELECT email, REGEXP_REPLACE(email, '@.*\\.(org|gov|com|edu|ca)$')
FROM users
ORDER BY userid LIMIT 4;

+----------------------------------------------+----------------------+
|                    email                     |    regexp_replace    |
+----------------------------------------------+----------------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu | Etiam.laoreet.libero  |
| Suspendisse.tristique@nonnisiAenean.edu       | Suspendisse.tristique |
| amet.faucibus.ut@condimentumegetvolutpat.ca   | amet.faucibus.ut      |
| sed@lacusUtnec.ca                             | sed                   |
+----------------------------------------------+----------------------+
```

要使用 internal.company.com 替换电子邮件地址的域名，请使用以下示例。

```
SELECT email, REGEXP_REPLACE(email, '@.*\\.[[:alpha:]]{2,3}','@internal.company.com')
FROM users
ORDER BY userid LIMIT 4;

+----------------------------------------------
+-------------------------------------------+
|                    email                     |               regexp_replace
      |
+----------------------------------------------
+-------------------------------------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu |
 Etiam.laoreet.libero@internal.company.com  |
| Suspendisse.tristique@nonnisiAenean.edu       |
 Suspendisse.tristique@internal.company.com |
```

```
| amet.faucibus.ut@condimentumegetvolutpat.ca   | amet.faucibus.ut@internal.company.com
      |
| sed@lacusUtnec.ca                             | sed@internal.company.com
      |
+-----------------------------------------------
+-------------------------------------------+
```

REGEXP_SUBSTR 函数

通过在字符串中搜索正则表达式模式，返回字符串中的字符。REGEXP_SUBSTR 与 SUBSTRING 函数 函数相似，只不过前者可让您在字符串中搜索正则表达式模式。如果函数无法将正则表达式与字符串中的任何字符匹配，则返回一个空字符串。有关正则表达式的更多信息，请参阅 POSIX 运算符和 Wikipedia 中的 Regular expression。

语法

```
REGEXP_SUBSTR( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

参数

source_string

要搜索的字符串表达式。

pattern

表示正则表达式模式的 UTF-8 字符串文本。有关更多信息，请参阅 POSIX 运算符。

position

指示在 source_string 中开始搜索的位置的正整数。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。默认值为 1。如果 position 小于 1，则搜索从 source_string 的第一个字符开始。如果 position 大于 source_string 中的字符数量，则结果为空字符串 ("")。

出现

一个正整数，指示要使用的模式的匹配项。REGEXP_SUBSTR 会跳过第一个 occurrence -1 匹配项。默认值为 1。如果 occurrence 小于 1 或大于 source_string 中的字符串，则会忽略搜索，并且结果为 NULL。

参数

一个或多个字符串，指示函数与模式的匹配方式。可能的值包括：

- c – 执行区分大小写的匹配。默认情况下，使用区分大小写的匹配。
- i – 执行不区分大小写的匹配。
- e – 使用子表达式提取子字符串。

  如果 pattern 包含一个子表达式，REGEXP_SUBSTR 会使用 pattern 中的第一个子表达式来匹配子字符串。子表达式是模式中用括号括起的表达式。例如，模式 `'This is a (\\w+)'` 将第一个表达式与字符串 `'This is a '` 后接一个字词进行匹配。此时不返回模式，带 e 参数的 REGEXP_SUBSTR 仅返回子表达式内的字符串。

  REGEXP_SUBSTR 仅考虑第一个子表达式；其他子表达式会被忽略。如果模式没有子表达式，REGEXP_SUBSTR 会忽略“e”参数。
- p – 使用 Perl 兼容正则表达式 (PCRE) 方言解释模式。有关 PCRE 的更多信息，请参阅 Wikipedia 中的 Perl Compatible Regular Expressions。

返回类型

VARCHAR

示例

以下示例返回电子邮件地址中 @ 字符和域扩展名之间的部分。所查询的 `users` 数据来自 Amazon Redshift 示例数据。有关更多信息，请参阅 示例数据库。

```
SELECT email, regexp_substr(email,'@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

                     email                      |      regexp_substr
------------------------------------------------+--------------------------
 Suspendisse.tristique@nonnisiAenean.edu        | @nonnisiAenean
 amet.faucibus.ut@condimentumegetvolutpat.ca    | @condimentumegetvolutpat
 sed@lacusUtnec.ca                              | @lacusUtnec
 Cum@accumsan.com                               | @accumsan
```

以下示例使用不区分大小写的匹配返回与字符串 FOX 的第一次出现相对应的输入部分。

```
SELECT regexp_substr('the fox', 'FOX', 1, 1, 'i');

 regexp_substr
```

```
---------------
 fox
```

以下示例使用不区分大小写的匹配返回与字符串 FOX 的第二次出现相对应的输入部分。结果为 NULL（空），因为没有第二次出现。

```
SELECT regexp_substr('the fox', 'FOX', 1, 2, 'i');

 regexp_substr
---------------

```

以下示例返回以小写字母开头的输入的第一部分。这在功能上与不带 c 参数的同一 SELECT 语句相同。

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');

 regexp_substr
---------------
 abc
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例返回与第二个此类单词相对应的输入部分。

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例返回与第二个此类单词相对应的输入部分，但与前面的示例不同，因为它使用了不区分大小写的匹配。

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
```

```
---------------
 A1234
```

以下示例使用子表达式，通过不区分大小写的匹配来查找与模式 'this is a (\\w+)' 匹配的第二个字符串。它返回括号内的子表达式。

```
SELECT regexp_substr(
               'This is a cat, this is a dog. This is a mouse.',
               'this is a (\\w+)', 1, 2, 'ie');

 regexp_substr
---------------
 dog
```

REPEAT 函数

将字符串重复指定的次数。如果输入参数为数字，REPEAT 会将其视为字符串。

REPLICATE 函数的同义词。

语法

```
REPEAT(string, integer)
```

参数

string

第一个输入参数是要重复的字符串。

integer

第二个参数是指示字符串重复次数的 INTEGER。

返回类型

VARCHAR

示例

以下示例使用 TICKIT 示例数据库的 CATEGORY 表中的数据。有关更多信息，请参阅 示例数据库。

要重复 CATEGORY 表中 CATID 列的值三次，请使用以下示例。

```
SELECT catid, REPEAT(catid,3)
FROM category
ORDER BY 1,2;

+-------+--------+
| catid | repeat |
+-------+--------+
|     1 |    111 |
|     2 |    222 |
|     3 |    333 |
|     4 |    444 |
|     5 |    555 |
|     6 |    666 |
|     7 |    777 |
|     8 |    888 |
|     9 |    999 |
|    10 | 101010 |
|    11 | 111111 |
+-------+--------+
```

REPLACE 函数

将现有字符串中一组字符的所有匹配项替换为其他指定字符。

REPLACE 与 TRANSLATE 函数和 REGEXP_REPLACE 函数相似，只不过 TRANSLATE 进行多次单字符替换，REGEXP_REPLACE 可让您在字符串中搜索正则表达式模式，而 REPLACE 一次性将整个字符串替换为其他字符串。

语法

```
REPLACE(string, old_chars, new_chars)
```

参数

string

要搜索的 CHAR 或 VARCHAR 字符串

old_chars

要替换的 CHAR 或 VARCHAR 字符串。

new_chars

用于替换 old_string 的新 CHAR 或 VARCHAR 字符串。

返回类型

VARCHAR

如果 old_chars 或 new_chars 为 NULL，则将返回 NULL。

示例

以下示例使用 TICKIT 示例数据库的 CATEGORY 表中的数据。有关更多信息，请参阅 示例数据库。

要将 CATGROUP 字段中的字符串 Shows 转换为 Theatre，请使用以下示例。

```
SELECT catid, catgroup, REPLACE(catgroup, 'Shows', 'Theatre')
FROM category
ORDER BY 1,2,3;

+-------+----------+----------+
| catid | catgroup | replace  |
+-------+----------+----------+
|     1 | Sports   | Sports   |
|     2 | Sports   | Sports   |
|     3 | Sports   | Sports   |
|     4 | Sports   | Sports   |
|     5 | Sports   | Sports   |
|     6 | Shows    | Theatre  |
|     7 | Shows    | Theatre  |
|     8 | Shows    | Theatre  |
|     9 | Concerts | Concerts |
|    10 | Concerts | Concerts |
|    11 | Concerts | Concerts |
+-------+----------+----------+
```

REPLICATE 函数

REPEAT 函数的同义词。

请参阅 REPEAT 函数。

REVERSE 函数

REVERSE 函数对字符串运行并以反向顺序返回字符。例如，`reverse('abcde')` 将返回 edcba。此函数适用于数字和日期数据类型以及字符数据类型；但在大多数情况下，它对于字符串具有实用价值。

语法

```
REVERSE( expression )
```

参数

expression

一个表达式，带有表示字符反转目标的字符、日期、时间戳或数字数据类型。所有表达式都隐式转换为 VARCHAR 字符串。CHAR 字符串中的尾部空白会被忽略：

返回类型

VARCHAR

示例

以下示例使用 TICKIT 示例数据库的 USERS 表和 SALES 表中的数据。有关更多信息，请参阅 [示例数据库](#)。

要从 USERS 表中选择 5 个不同的城市名称及其对应的反转名称，请使用以下示例。

```
SELECT DISTINCT city AS cityname, REVERSE(cityname)
FROM users
ORDER BY city LIMIT 5;

+----------+----------+
| cityname | reverse  |
+----------+----------+
| Aberdeen | needrebA |
| Abilene  | enelibA  |
| Ada      | adA      |
| Agat     | tagA     |
| Agawam   | mawagA   |
```

```
+----------+----------+
```

要选择 5 个销售 ID 及其对应的反转 ID（已强制转换为字符串），请使用以下示例。

```
SELECT salesid, REVERSE(salesid)
FROM sales
ORDER BY salesid DESC LIMIT 5;

+---------+---------+
| salesid | reverse |
+---------+---------+
|  172456 |  654271 |
|  172455 |  554271 |
|  172454 |  454271 |
|  172453 |  354271 |
|  172452 |  254271 |
+---------+---------+
```

RTRIM 函数

RTRIM 函数从字符串的末尾剪裁指定的一组字符。删除只包含剪裁字符列表中的字符的最长字符串。当输入字符串中没有了剪裁字符时，剪裁即告完成。

语法

```
RTRIM( string, trim_chars )
```

参数

string

要剪裁的字符串列、表达式或字符串文本。

trim_chars

表示要从 string 的结尾剪裁的字符的字符串列、表达式或字符串文本。如果未指定，则使用空格作为剪裁字符。

返回类型

与 string 参数具有相同的数据类型的字符串。

示例

以下示例从字符串 ' abc ' 中剪裁前导和尾随空格：

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

以下示例从字符串 'xyzaxyzbxyzcxyz' 中删除尾随字符串 'xyz'。将删除尾随的 'xyz'，但不会删除字符串内部的匹配字符。

```
select 'xyzaxyzbxyzcxyz' as untrim,
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

以下示例从字符串 'setuphistorycassettes' 中删除与 trim_chars 列表 'tes' 中的任何字符相匹配的结尾部分。在输入字符串结尾部分，在 trim_chars 列表中未包含的其他字符之前出现的任何 t、e 或 s 都将被删除。

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

以下示例从 VENUENAME 的末尾剪裁字符“Park”（如果有）：

```
select venueid, venuename, rtrim(venuename, 'Park')
from venue
order by 1, 2, 3
limit 10;

venueid |         venuename          |          rtrim
--------+----------------------------+-------------------------
      1 | Toyota Park                | Toyota
      2 | Columbus Crew Stadium      | Columbus Crew Stadium
```

```
      3 | RFK Stadium                | RFK Stadium
      4 | CommunityAmerica Ballpark  | CommunityAmerica Ballp
      5 | Gillette Stadium           | Gillette Stadium
      6 | New York Giants Stadium    | New York Giants Stadium
      7 | BMO Field                  | BMO Field
      8 | The Home Depot Center      | The Home Depot Cente
      9 | Dick's Sporting Goods Park | Dick's Sporting Goods
     10 | Pizza Hut Park             | Pizza Hut
```

请注意，当字符 P、a、r 或 k 中的任意一个出现在 VENUENAME 的末尾时，RTRIM 都会予以删除。

SOUNDEX 函数

SOUNDEX 函数返回美国 Soundex 值，其中包括输入字符串的第一个字母，后跟一个 3 位数字的声音编码，该编码表示您指定的字符串的英语发音。例如，Smith 和 Smyth 具有相同的 Soundex 值。

语法

```
SOUNDEX(string)
```

参数

string

您可以指定要转换为美国 Soundex 代码值的 CHAR 或 VARCHAR 字符串。

返回类型

VARCHAR(4)

使用说明

SOUNDEX 函数仅转换英文字母小写或大写 ASCII 字符，包括 a–z 和 A–Z。SOUNDEX 将忽略其他字符。对于由空格分隔的多个单词组成的字符串，SOUNDEX 返回单个 Soundex 值。

```
SELECT SOUNDEX('AWS Amazon');

+---------+
| soundex |
+---------+
| A252    |
```

```
+---------+
```

如果输入字符串不包含任何英文字母，SOUNDEX 将返回一个空字符串。

```
SELECT SOUNDEX('+-*/%');

+---------+
| soundex |
+---------+
|         |
+---------+
```

示例

要返回 Amazon 的 Soundex 值，请使用以下示例。

```
SELECT SOUNDEX('Amazon');

+---------+
| soundex |
+---------+
| A525    |
+---------+
```

要返回 smith 和 smyth 的 Soundex 值，请使用以下示例。请注意，Soundex 值是相同的。

```
SELECT SOUNDEX('smith'), SOUNDEX('smyth');

+-------+-------+
| smith | smyth |
+-------+-------+
| S530  | S530  |
+-------+-------+
```

SPLIT_PART 函数

用指定的分隔符拆分字符串，并返回指定位置的部分内容。

语法

```
SPLIT_PART(string, delimiter, position)
```

参数

string

要拆分的字符串列、表达式或字符串文本。字符串可以是 CHAR 或 VARCHAR。

分隔符

分隔符字符串指示输入 string 的部分。

如果 delimiter 是文本，则将其括在单引号中。

position

要返回的 string 部分的位置（从 1 算起）。必须是大于 0 的整数。如果 position 大于字符串部分的数量，SPLIT_PART 将返回空字符串。如果在字符串中未找到分隔符，则返回的值包含指定部分的内容，它可能是整个字符串或一个空值。

返回类型

CHAR 或 VARCHAR 字符串，与 string 参数相同。

示例

以下示例使用 $ 分隔符，将字符串文本拆分为多个部分，并返回第二部分。

```
select split_part('abc$def$ghi','$',2)

split_part
----------
def
```

以下示例使用 $ 分隔符，将字符串文本拆分为多个部分。它返回一个空字符串，因为找不到部分 4。

```
select split_part('abc$def$ghi','$',4)

split_part
----------

```

以下示例使用 # 分隔符，将字符串文本拆分为多个部分。它返回整个字符串，也就是第一部分，因为找不到分隔符。

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

以下示例将时间戳字段 LISTTIME 拆分为年、月和日组成部分。

```
select listtime, split_part(listtime,'-',1) as year,
split_part(listtime,'-',2) as month,
split_part(split_part(listtime,'-',3),' ',1) as day
from listing limit 5;

      listtime       | year | month | day
---------------------+------+-------+------
 2008-03-05 12:25:29 | 2008 | 03    | 05
 2008-09-09 08:03:36 | 2008 | 09    | 09
 2008-09-26 05:43:12 | 2008 | 09    | 26
 2008-10-04 02:00:30 | 2008 | 10    | 04
 2008-01-06 08:33:11 | 2008 | 01    | 06
```

以下示例选择 LISTTIME 时间戳字段并在 '-' 字符处拆分它以获取月（LISTTIME 字符串的第二部分），然后计算每个月的条目数：

```
select split_part(listtime,'-',2) as month, count(*)
from listing
group by split_part(listtime,'-',2)
order by 1, 2;

 month | count
-------+-------
    01 | 18543
    02 | 16620
    03 | 17594
    04 | 16822
    05 | 17618
    06 | 17158
    07 | 17626
    08 | 17881
    09 | 17378
    10 | 17756
    11 | 12912
```

```
    12 | 4589
```

STRPOS 函数

返回子字符串在指定字符串中的位置。

有关类似的函数，请参阅CHARINDEX 函数和POSITION 函数。

语法

```
STRPOS(string, substring )
```

参数

string

第一个输入参数是要在其中进行搜索的 CHAR 或 VARCHAR 字符串。

substring

第二个参数是要在 string 中搜索的子字符串。

返回类型

INTEGER

STRPOS 函数返回与 substring 的位置对应的 INTEGER（从 1 开始，而不是从 0 开始）。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。

使用说明

如果在 string 中未找到 substring，STRPOS 将返回 0。

```
SELECT STRPOS('dogfish', 'fist');

+--------+
| strpos |
+--------+
|      0 |
+--------+
```

示例

要显示 fish 在 dogfish 内的位置，请使用以下示例。

```
SELECT STRPOS('dogfish', 'fish');

+--------+
| strpos |
+--------+
|      4 |
+--------+
```

以下示例使用 TICKIT 示例数据库的 SALES 表中的数据。有关更多信息，请参阅 示例数据库。

要返回 SALES 表中 COMMISSION 超过 999.00 的销售交易的数量，请使用以下示例。

```
SELECT DISTINCT STRPOS(commission, '.'),
COUNT (STRPOS(commission, '.'))
FROM sales
WHERE STRPOS(commission, '.') > 4
GROUP BY STRPOS(commission, '.')
ORDER BY 1, 2;

+--------+-------+
| strpos | count |
+--------+-------+
|      5 |   629 |
+--------+-------+
```

STRTOL 函数

将由一些指定基数组成的字符串表达式转换为等效的整数值。已转换的值必须在有符号 64 位范围中。

语法

```
STRTOL(num_string, base)
```

参数

num_string

要转换的数字的字符串表达式。如果 num_string 为空（ '' ）或以 null 字符（ '\0' ）开头，则转换后的值为 0。如果 num_string 是包含 NULL 值的列，STRTOL 将返回 NULL。字符串能够以任意数

量的空格开头，也可以后跟加号“+”或减号“-”以表示正负。默认值为“+”。如果 base 为 16，则字符串可以“0x”开头。

base

INTEGER 介于 2 和 36 之间。

返回类型

BIGINT

如果 num_string 为 null，则函数返回 NULL。

示例

要将字符串和基数值对转换为整数，请使用以下示例。

```
SELECT STRTOL('0xf',16);

+--------+
| strtol |
+--------+
|     15 |
+--------+

SELECT STRTOL('abcd1234',16);

+------------+
|   strtol   |
+------------+
| 2882343476 |
+------------+

SELECT STRTOL('1234567', 10);

+---------+
| strtol  |
+---------+
| 1234567 |
+---------+

SELECT STRTOL('1234567', 8);
```

```
+--------+
| strtol |
+--------+
| 342391 |
+--------+

SELECT STRTOL('110101', 2);

+--------+
| strtol |
+--------+
|     53 |
+--------+

SELECT STRTOL('\0', 2);

+--------+
| strtol |
+--------+
|      0 |
+--------+
```

SUBSTRING 函数

按指定的开始位置返回子字符串子集。

如果输入的是字符串，字符的开始位置和数量基于字符数而不是字节数，这是为了将多字节字符作为单个字符计数。如果输入的是二进制表达式，则开始位置和提取的子字符串基于字节。您无法指定负长度，但可指定负开始位置。

语法

```
SUBSTRING(character_string FROM start_position [ FOR number_characters ] )
```

```
SUBSTRING(character_string, start_position, number_characters )
```

```
SUBSTRING(binary_expression, start_byte, number_bytes )
```

```
SUBSTRING(binary_expression, start_byte )
```

参数

character_string

要搜索的字符串。非字符数据类型将视为字符串。

start_position

字符串中开始提取的位置，从 1 开始。start_position 基于字符数而不是字节数，这是为了将多字节字符作为单个字符计数。此数字可以为负。

number_characters

要提取的字符的数量（子字符串的长度）。number_characters 基于字符数而不是字节数，这是为了将多字节字符作为单个字符计数。此数字不能为负。

binary_expression

要搜索的数据类型为 VARBYTE 的 binary_expression。

start_byte

二进制表达式中开始提取的位置，从 1 开始。此数字可以为负。

number_bytes

要提取的字节的数量（子字符串的长度）。此数字不能为负。

返回类型

根据输入选择 VARCHAR 或 VARBYTE。

使用说明

以下是一些示例，说明如何使用 start_position 和 number_characters 从字符串的不同位置提取子字符串。

以下示例返回以第 6 个字符开头的 4 字符字符串。

```
select substring('caterpillar',6,4);
substring
-----------
pill
```

```
(1 row)
```

如果 start_position + number_characters 超过 string 的长度，SUBSTRING 将返回从 start_position 开始到此字符串末尾的子字符串。例如：

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

如果 start_position 为负或 0，SUBSTRING 函数将返回从长度为 start_position + number_characters -1 的字符串的第一个字符开始的子字符串。例如：

```
select substring('caterpillar',-2,6);
substring
-----------
cat
(1 row)
```

如果 start_position + number_characters -1 小于或等于零，SUBSTRING 将返回空字符串。例如：

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

示例

以下示例返回 LISTING 表的 LISTTIME 字符串中的月份：

```
select listid, listtime,
substring(listtime, 6, 2) as month
from listing
order by 1, 2, 3
limit 10;

 listid |      listtime       | month
```

```
--------+---------------------+-------
      1 | 2008-01-24 06:43:29 | 01
      2 | 2008-03-05 12:25:29 | 03
      3 | 2008-11-01 07:35:33 | 11
      4 | 2008-05-24 01:18:37 | 05
      5 | 2008-05-17 02:29:11 | 05
      6 | 2008-08-15 02:08:13 | 08
      7 | 2008-11-15 09:38:15 | 11
      8 | 2008-11-09 05:07:30 | 11
      9 | 2008-09-09 08:03:36 | 09
     10 | 2008-06-17 09:44:54 | 06
(10 rows)
```

以下示例与上述示例相同，但使用 FROM...FOR 选项：

```
select listid, listtime,
substring(listtime from 6 for 2) as month
from listing
order by 1, 2, 3
limit 10;

 listid |      listtime       | month
--------+---------------------+-------
      1 | 2008-01-24 06:43:29 | 01
      2 | 2008-03-05 12:25:29 | 03
      3 | 2008-11-01 07:35:33 | 11
      4 | 2008-05-24 01:18:37 | 05
      5 | 2008-05-17 02:29:11 | 05
      6 | 2008-08-15 02:08:13 | 08
      7 | 2008-11-15 09:38:15 | 11
      8 | 2008-11-09 05:07:30 | 11
      9 | 2008-09-09 08:03:36 | 09
     10 | 2008-06-17 09:44:54 | 06
(10 rows)
```

您无法使用 SUBSTRING 以可预测的方式提取可能包含多字节字符的字符串的前缀，因为您需要根据字节数（而不是字符数）指定多字节字符串的长度。要基于以字节为单位的长度提取字符串的开始部分，您可将字符串强制转换为 VARCHAR(byte_length) 以截断字符串，其中 byte_length 是必需长度。以下示例提取字符串 'Fourscore and seven' 的前 5 个字节。

```
select cast('Fourscore and seven' as varchar(5));
```

```
varchar
-------
Fours
```

以下示例显示了二进制值 abc 的负开始位置。由于开始位置为 -3，所以子字符串从二进制值的开头进行提取。结果会自动显示为二进制子字符串的十六进制表示形式。

```
select substring('abc'::varbyte, -3);

 substring
-----------
 616263
```

以下示例显示二进制值 abc 的开始位置为 1。因为没有指定长度，所以将字符串从字符串开始位置提取到末尾。结果会自动显示为二进制子字符串的十六进制表示形式。

```
select substring('abc'::varbyte, 1);

 substring
-----------
 616263
```

以下示例显示二进制值 abc 的开始位置为 3。因为没有指定长度，所以将字符串从字符串开始位置提取到末尾。结果会自动显示为二进制子字符串的十六进制表示形式。

```
select substring('abc'::varbyte, 3);

 substring
-----------
 63
```

以下示例显示二进制值 abc 的开始位置为 2。字符串从开始位置提取到位置 10，但字符串的末尾位于位置 3。结果会自动显示为二进制子字符串的十六进制表示形式。

```
select substring('abc'::varbyte, 2, 10);

 substring
-----------
 6263
```

以下示例显示二进制值 abc 的开始位置为 2。字符串从开始位置提取 1 个字节。结果会自动显示为二进制子字符串的十六进制表示形式。

```
select substring('abc'::varbyte, 2, 1);

 substring
-----------
 62
```

以下示例返回出现在输入字符串 Silva, Ana 中最后一个空格之后的名字 Ana。

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))

 reverse
-----------
 Ana
```

TEXTLEN 函数

LEN 函数的同义词。

请参阅 LEN 函数。

TRANSLATE 函数

对于给定表达式，将指定字符的所有匹配项替换为指定替代项。现有字符将按其在 characters_to_replace 和 characters_to_substitute 参数中的位置映射到替换字符。如果在 characters_to_replace 参数中指定的字符多于在 characters_to_substitute 参数中指定的字符，返回值中将省略 characters_to_replace 参数中的额外字符。

TRANSLATE 与 REPLACE 函数和 REGEXP_REPLACE 函数相似，只不过 REPLACE 将整个字符串替换为其他字符串，REGEXP_REPLACE 可让您在字符串中搜索正则表达式模式，而 TRANSLATE 进行多次单字符替换。

如果任何参数为 null，则返回 NULL。

语法

```
TRANSLATE( expression, characters_to_replace, characters_to_substitute )
```

参数

expression

要转换的表达式。

characters_to_replace

一个包含要替换的字符的字符串。

characters_to_substitute

一个字符串，其中包含要替换其他字符的字符。

返回类型

VARCHAR

示例

要替换字符串中的多个字符，请使用以下示例。

```
SELECT TRANSLATE('mint tea', 'inea', 'osin');

+-----------+
| translate |
+-----------+
| most tin  |
+-----------+
```

以下示例使用 TICKIT 示例数据库的 USERS 表中的数据。有关更多信息，请参阅 示例数据库。

要将列中所有值的 at（@）符号替换为句点，请使用以下示例。

```
SELECT email, TRANSLATE(email, '@', '.') as obfuscated_email
FROM users LIMIT 10;

+----------------------------------------+----------------------------------------+
|                 email                  |           obfuscated_email             |
+----------------------------------------+----------------------------------------+
| Cum@accumsan.com                       | Cum.accumsan.com                       |
| lorem.ipsum@Vestibulumante.com         | lorem.ipsum.Vestibulumante.com         |
| non.justo.Proin@ametconsectetuer.edu   | non.justo.Proin.ametconsectetuer.edu   |
| non.ante.bibendum@porttitortellus.org | non.ante.bibendum.porttitortellus.org |
```

```
| eros@blanditatnisi.org               | eros.blanditatnisi.org               |
| augue@Donec.ca                       | augue.Donec.ca                       |
| cursus@pedeacurna.edu                | cursus.pedeacurna.edu                |
| at@Duis.com                          | at.Duis.com                          |
| quam@facilisisvitaeorci.ca           | quam.facilisisvitaeorci.ca           |
| mi.lorem@nunc.edu                    | mi.lorem.nunc.edu                    |
+--------------------------------------+--------------------------------------+
```

要将空格替换为下划线并去掉列中所有值的句点，请使用以下示例。

```
SELECT city, TRANSLATE(city, ' .', '_')
FROM users
WHERE city LIKE 'Sain%' OR city LIKE 'St%'
GROUP BY city
ORDER BY city;

+----------------+---------------+
|      city      |   translate   |
+----------------+---------------+
| Saint Albans   | Saint_Albans  |
| Saint Cloud    | Saint_Cloud   |
| Saint Joseph   | Saint_Joseph  |
| Saint Louis    | Saint_Louis   |
| Saint Paul     | Saint_Paul    |
| St. George     | St_George     |
| St. Marys      | St_Marys      |
| St. Petersburg | St_Petersburg |
| Stafford       | Stafford      |
| Stamford       | Stamford      |
| Stanton        | Stanton       |
| Starkville     | Starkville    |
| Statesboro     | Statesboro    |
| Staunton       | Staunton      |
| Steubenville   | Steubenville  |
| Stevens Point  | Stevens_Point |
| Stillwater     | Stillwater    |
| Stockton       | Stockton      |
| Sturgis        | Sturgis       |
+----------------+---------------+
```

TRIM 函数

通过空白或指定的字符来剪裁字符串。

语法

```
TRIM( [ BOTH | LEADING | TRAILING ] [trim_chars FROM ] string )
```

参数

BOTH | LEADING | TRAILING

（可选）指定从何处剪裁字符。使用 BOTH 会删除前导和尾随字符，使用 LEADING 仅删除前导字符，使用 TRAILING 仅删除尾随字符。如果省略此参数，则会同时剪裁前导字符和尾随字符。

trim_chars

（可选）要从字符串剪裁的字符数。如果忽略此参数，则剪裁空白区域。

string

要剪裁的字符串。

返回类型

TRIM 函数返回 VARCHAR 或 CHAR 字符串。如果您将 TRIM 函数与 SQL 命令结合使用，Amazon Redshift 会将结果隐式转换为 VARCHAR。如果您在 SQL 函数的 SELECT 列表中使用 TRIM 函数，Amazon Redshift 不会隐式转换结果，您可能需要执行显式转换以避免数据类型不匹配错误。有关显式转换的信息，请参阅 CAST 函数和 CONVERT 函数 函数。

示例

要从字符串 dog 中剪裁前导和尾随空格，请使用以下示例。

```
SELECT TRIM('    dog    ');

+-------+
| btrim |
+-------+
| dog   |
+-------+
```

要从字符串 dog 中剪裁前导和尾随空格，请使用以下示例。

```
SELECT TRIM(BOTH FROM '    dog    ');
```

```
+-------+
| btrim |
+-------+
| dog   |
+-------+
```

要从字符串 "dog" 中删除前导双引号，请使用以下示例。

```
SELECT TRIM(LEADING '"' FROM'"dog"');

+-------+
| ltrim |
+-------+
| dog"  |
+-------+
```

要从字符串 "dog" 中删除尾随双引号，请使用以下示例。

```
SELECT TRIM(TRAILING '"' FROM'"dog"');

+-------+
| rtrim |
+-------+
| "dog  |
+-------+
```

当 trim_chars 中的任意字符出现在 string 的开头或结尾时，TRIM 都会予以删除。以下示例在字符“C”、“D”和“G”出现在 VENUENAME（即 VARCHAR 列）的开头或结尾时对其进行剪裁。有关更多信息，请参阅 VENUE 表。

```
SELECT venueid, venuename, TRIM('CDG' FROM venuename)
FROM venue
WHERE venuename LIKE '%Park'
ORDER BY 2
LIMIT 7;

+---------+----------------------------+----------------------------+
| venueid |         venuename          |           btrim            |
+---------+----------------------------+----------------------------+
|     121 | AT&T Park                  | AT&T Park                  |
```

```
|     109 | Citizens Bank Park         | itizens Bank Park         |
|     102 | Comerica Park              | omerica Park              |
|       9 | Dick's Sporting Goods Park | ick's Sporting Goods Park |
|      97 | Fenway Park                | Fenway Park               |
|     112 | Great American Ball Park   | reat American Ball Park   |
|     114 | Miller Park                | Miller Park               |
+---------+----------------------------+---------------------------+
```

UPPER 函数

将字符串转换为大写。UPPER 支持 UTF-8 多字节字符，并且每个字符最多可以有 4 个字节。

语法

```
UPPER(string)
```

参数

string

输入参数是 VARCHAR 字符串（或任何其他可隐式转换为 VARCHAR 的数据类型，如 CHAR）。

返回类型

UPPER 函数返回与输入字符串具有相同数据类型的字符串。例如，如果输入是 VARCHAR 字符串，该函数将返回 VARCHAR 字符串。

示例

以下示例使用 TICKIT 示例数据库的 CATEGORY 表中的数据。有关更多信息，请参阅 示例数据库。

要将 CATNAME 字段转换为大写，请使用以下内容。

```
SELECT catname, UPPER(catname)
FROM category
ORDER BY 1,2;

+-----------+-----------+
|  catname  |   upper   |
+-----------+-----------+
| Classical | CLASSICAL |
```

```
| Jazz       | JAZZ       |
| MLB        | MLB        |
| MLS        | MLS        |
| Musicals   | MUSICALS   |
| NBA        | NBA        |
| NFL        | NFL        |
| NHL        | NHL        |
| Opera      | OPERA      |
| Plays      | PLAYS      |
| Pop        | POP        |
+------------+------------+
```

SUPER 类型信息函数

接着，您可以找到 Amazon Redshift 支持的 SQL 的类型信息函数的描述，以从 SUPER 数据类型的输入中派生动态信息。

主题

- DECIMAL_PRECISION 函数
- DECIMAL_SCALE 函数
- IS_ARRAY 函数
- IS_BIGINT 函数
- IS_BOOLEAN 函数
- IS_CHAR 函数
- IS_DECIMAL 函数
- IS_FLOAT 函数
- IS_INTEGER 函数
- IS_OBJECT 函数
- IS_SCALAR 函数
- IS_SMALLINT 函数
- IS_VARCHAR 函数
- JSON_SIZE 函数
- JSON_TYPEOF 函数
- SIZE

DECIMAL_PRECISION 函数

检查要存储的最大小数位数总数的精度。此数字包括小数点的左侧和右侧数字。精度范围为 1 到 38，默认值为 38。

语法

```
DECIMAL_PRECISION(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

INTEGER

示例

要将 DECIMAL_PRECISION 函数应用于表 t，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_PRECISION(s) FROM t;

+-------------------+
| decimal_precision |
+-------------------+
|                 6 |
+-------------------+
```

DECIMAL_SCALE 函数

检查要存储在小数点右侧的小数位数。小数位数范围从 0 到精度点，默认值为 0。

语法

```
DECIMAL_SCALE(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

INTEGER

示例

要将 DECIMAL_SCALE 函数应用于表 t，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_SCALE(s) FROM t;

+---------------+
| decimal_scale |
+---------------+
|             5 |
+---------------+
```

IS_ARRAY 函数

检查变量是否为数组。如果变量是数组，则函数返回 `true`。该函数还包括空数组。否则，对于所有其他值，包括 null，函数返回 `false`。

语法

```
IS_ARRAY(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_ARRAY 函数检查 [1,2] 是否为数组，请使用以下示例。

```
SELECT IS_ARRAY(JSON_PARSE('[1,2]'));

+----------+
| is_array |
+----------+
| true     |
+----------+
```

IS_BIGINT 函数

检查某个值是否为 BIGINT。对于 64 位范围内的小数位数 0 的数量，IS_BIGINT 函数将返回 true。否则，对于所有其他值，包括 null 和浮点数，该函数将返回 false。

IS_BIGINT 函数是 IS_INTEGER 的超集。

语法

```
IS_BIGINT(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_BIGINT 函数检查 5 是否为 BIGINT，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_BIGINT(s) FROM t;

+---+-----------+
| s | is_bigint |
+---+-----------+
| 5 | true      |
+---+-----------+
```

IS_BOOLEAN 函数

检查某个值是否为 BOOLEAN。对于常量 JSON 布尔值，IS_BOOLEAN 函数返回 `true`。对于任何其他值，包括 null，该函数返回 `false`。

语法

```
IS_BOOLEAN(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_BOOLEAN 函数检查 TRUE 是否为 BOOLEAN，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (TRUE);

SELECT s, IS_BOOLEAN(s) FROM t;
```

```
+------+------------+
|  s   | is_boolean |
+------+------------+
| true | true       |
+------+------------+
```

IS_CHAR 函数

检查某个值是否为 CHAR。对于仅包含 ASCII 字符的字符串，IS_CHAR 函数返回 `true`，因为 CHAR 类型只能存储 ASCII 格式的字符。对于任何其他值，该函数返回 `false`。

语法

```
IS_CHAR(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_CHAR 函数检查 `t` 是否为 CHAR，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('t');

SELECT s, IS_CHAR(s) FROM t;

+-----+---------+
|  s  | is_char |
+-----+---------+
| "t" | true    |
+-----+---------+
```

IS_DECIMAL 函数

检查某个值是否为 DECIMAL。对于非浮点的数值，IS_DECIMAL 函数返回 true。对于任何其他值，包括 null，该函数返回 false。

IS_DECIMAL 函数是 IS_BIGINT 的超集。

语法

```
IS_DECIMAL(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_DECIMAL 函数检查 1.22 是否为 DECIMAL，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (1.22);

SELECT s, IS_DECIMAL(s) FROM t;

+------+------------+
|  s   | is_decimal |
+------+------------+
| 1.22 | true       |
+------+------------+
```

IS_FLOAT 函数

检查值是否为浮点数。对于浮点数（FLOAT4 和 FLOAT8），IS_FLOAT 函数返回 true。对于任何其他值，该函数返回 false。

IS_DECIMAL 集和 IS_FLOAT 集是不相交的。

语法

```
IS_FLOAT(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_FLOAT 函数检查 2.22::FLOAT 是否为 FLOAT，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES(2.22::FLOAT);

SELECT s, IS_FLOAT(s) FROM t;

+---------+----------+
|    s    | is_float |
+---------+----------+
| 2.22e+0 | true     |
+---------+----------+
```

IS_INTEGER 函数

对于 32 位范围内的小数位数 0 的数量，返回 true；对于其他任何值（包括 null 和浮点数），则返回 false。

IS_INTEGER 函数是 IS_SMALLINT 函数的超集。

语法

```
IS_INTEGER(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_INTEGER 函数检查 5 是否为 INTEGER，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_INTEGER(s) FROM t;

+---+------------+
| s | is_integer |
+---+------------+
| 5 | true       |
+---+------------+
```

IS_OBJECT 函数

检查变量是否为对象。对于包括空对象在内的对象，IS_OBJECT 函数返回 `true`。对于任何其他值，包括 null，该函数返回 `false`。

语法

```
IS_OBJECT(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_OBJECT 函数检查 {"name": "Joe"} 是否为对象，请使用以下示例。

```
CREATE TABLE t(s super);

INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));

SELECT s, IS_OBJECT(s) FROM t;

+----------------+-----------+
|       s        | is_object |
+----------------+-----------+
| {"name":"Joe"} | true      |
+----------------+-----------+
```

IS_SCALAR 函数

检查变量是否为标量。对于非数组或对象的任何值，IS_SCALAR 函数返回 `true`。对于任何其他值，包括 null，该函数返回 `false`。

IS_ARRAY、IS_OBJECT 和 IS_SCALAR 的集合覆盖除 null 之外的所有值。

语法

```
IS_SCALAR(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_SCALAR 函数检查 {"name": "Joe"} 是否为标量，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));

SELECT s, IS_SCALAR(s.name) FROM t;

+----------------+-----------+
|       s        | is_scalar |
+----------------+-----------+
| {"name":"Joe"} | true      |
+----------------+-----------+
```

IS_SMALLINT 函数

检查变量是否为 SMALLINT。对于 16 位范围内的小数位数 0 的数量，IS_SMALLINT 函数返回 true。对于任何其他值，包括 null 和浮点数，该函数返回 false。

语法

```
IS_SMALLINT(super_expression)
```

参数

super_expression

SUPER 表达式或列。

Return

BOOLEAN

示例

要使用 IS_SMALLINT 函数检查 5 是否为 SMALLINT，请使用以下示例。

```
CREATE TABLE t(s SUPER);
```

```
INSERT INTO t VALUES (5);

SELECT s, IS_SMALLINT(s) FROM t;

+---+-------------+
| s | is_smallint |
+---+-------------+
| 5 | true        |
+---+-------------+
```

IS_VARCHAR 函数

检查变量是否为 VARCHAR。对于所有字符串，IS_VARCHAR 函数返回 `true`。对于任何其他值，该函数返回 `false`。

IS_VARCHAR 函数是 IS_CHAR 函数的超集。

语法

```
IS_VARCHAR(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_VARCHAR 函数检查 abc 是否为 VARCHAR，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('abc');

SELECT s, IS_VARCHAR(s) FROM t;
```

```
+-------+------------+
|   s   | is_varchar |
+-------+------------+
| "abc" | true       |
+-------+------------+
```

JSON_SIZE 函数

当序列化为字符串时，JSON_SIZE 函数返回给定 SUPER 表达式中的字节数。

语法

```
JSON_SIZE(super_expression)
```

参数

super_expression

SUPER 常量或表达式。

返回类型

INTEGER

JSON_SIZE 函数返回一个 INTEGER，表示输入字符串中的字节数。此值不同于字符数。例如，UTF-8 字符 #（黑点）的大小为 3 字节，即使它是 1 个字符也是如此。

使用说明

JSON_SIZE(x) 在功能上与 OCTET_LENGTH(JSON_SERIALIZE) 相同。但请注意，如果提供的 SUPER 表达式在序列化时超过系统的 VARCHAR 限制，JSON_SERIALIZE 会返回错误。JSON_SIZE 没有这个限制。

示例

要返回序列化为字符串的 SUPER 值的长度，请使用以下示例。

```
SELECT JSON_SIZE(JSON_PARSE('[10001,10002,"#"]'));

+-----------+
```

```
| json_size |
+-----------+
|        19 |
+-----------+
```

请注意，提供的 SUPER 表达式长度为 17 个字符，但 # 为 3 字节字符，因此 JSON_SIZE 返回 19。

JSON_TYPEOF 函数

根据 SUPER 值的动态类型，JSON_TYPEOF 标量函数返回具有布尔值、数值、字符串、对象、数组或 null 的 VARCHAR。

语法

```
JSON_TYPEOF(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

VARCHAR

示例

要使用 JSON_TYPEOF 函数检查数组 [1,2] 的 JSON 类型，请使用以下示例。

```
SELECT JSON_TYPEOF(ARRAY(1,2));

+-------------+
| json_typeof |
+-------------+
| array       |
+-------------+
```

要使用 JSON_TYPEOF 函数检查对象 {"name":"Joe"} 的 JSON 类型，请使用以下示例。

```
SELECT JSON_TYPEOF(JSON_PARSE('{"name":"Joe"}'));
```

```
+-------------+
| json_typeof |
+-------------+
| object      |
+-------------+
```

SIZE

以 INTEGER 形式返回 SUPER 类型常量或表达式的二进制内存大小。

语法

```
SIZE(super_expression)
```

参数

super_expression

SUPER 类型的常量或表达式。

返回类型

INTEGER

示例

要使用 SIZE 获取多个 SUPER 类型表达式的内存大小，请使用以下示例。

```
CREATE TABLE test_super_size(a SUPER);

INSERT INTO test_super_size
VALUES
  (null),
  (TRUE),
  (JSON_PARSE('[0,1,2,3]')),
  (JSON_PARSE('{"a":0,"b":1,"c":2,"d":3}'))
;

SELECT a, SIZE(a)
FROM test_super_size
ORDER BY 2, 1;
```

```
+---------------------------+------+
|             a             | size |
+---------------------------+------+
| true                      |    4 |
| NULL                      |    4 |
| [0,1,2,3]                 |   23 |
| {"a":0,"b":1,"c":2,"d":3} |   52 |
+---------------------------+------+
```

VARBYTE 函数和运算符

支持 VARBYTE 数据类型的 Amazon Redshift 函数和运算符包括：

- VARBYTE 运算符
- FROM_HEX
- FROM_VARBYTE
- GETBIT
- TO_HEX
- TO_VARBYTE
- CONCAT
- LEN
- LENGTH 函数
- OCTET_LENGTH
- SUBSTRING 函数

VARBYTE 运算符

下表列出了 VARBYTE 运算符。运算符使用数据类型为 VARBYTE 的二进制值。如果一个或两个输入为 null，则结果也为 null。

支持的运算符

操作符	描述	返回类型
<	Less than	BOOLEAN

操作符	描述	返回类型
<=	Less than or equal	BOOLEAN
=	Equal	BOOLEAN
>	Greater than	BOOLEAN
>=	Greater than or equal	BOOLEAN
!= 或 <>	不等于	BOOLEAN
||	联接	VARBYTE
+	联接	VARBYTE
~	逐位执行非运算	VARBYTE
&	按位和	VARBYTE
|	按位或	VARBYTE
#	按位异或	VARBYTE

示例

在以下示例中，'a'::VARBYTE 的值 是 61，'b'::VARBYTE 的值是 62。:: 将字符串强制转换为 VARBYTE 数据类型。有关强制转换数据类型的更多信息，请参阅CAST。

要使用 < 运算符比较 'a' 是否小于 'b'，请使用以下示例。

```
SELECT 'a'::VARBYTE < 'b'::VARBYTE AS less_than;

+-----------+
| less_than |
+-----------+
```

```
| true      |
+-----------+
```

要使用 = 运算符比较 'a' 是否等于 'b'，请使用以下示例。

```
SELECT 'a'::VARBYTE = 'b'::VARBYTE AS equal;

+-------+
| equal |
+-------+
| false |
+-------+
```

要使用 || 运算符连接两个二进制值，请使用以下示例。

```
SELECT 'a'::VARBYTE || 'b'::VARBYTE AS concat;

+--------+
| concat |
+--------+
|   6162 |
+--------+
```

要使用 + 运算符连接两个二进制值，请使用以下示例。

```
SELECT 'a'::VARBYTE + 'b'::VARBYTE AS concat;

+--------+
| concat |
+--------+
|   6162 |
+--------+
```

要使用 FROM_VARBYTE 函数否定输入二进制值的每个位，请使用以下示例。字符串 'a' 计算结果为 01100001。有关更多信息，请参阅FROM_VARBYTE。

```
SELECT FROM_VARBYTE(~'a'::VARBYTE, 'binary');

+--------------+
| from_varbyte |
+--------------+
```

```
|     10011110 |
+--------------+
```

要在两个输入二进制值上应用 & 运算符，请使用以下示例。字符串 'a' 计算结果为 01100001，而 'b' 计算结果为 01100010。

```
SELECT FROM_VARBYTE('a'::VARBYTE & 'b'::VARBYTE, 'binary');

+--------------+
| from_varbyte |
+--------------+
|     01100000 |
+--------------+
```

FROM_HEX 函数

FROM_HEX 将十六进制转换为二进制值。

语法

```
FROM_HEX(hex_string)
```

参数

hex_string

要转换的数据类型为 VARCHAR 或 TEXT 的十六进制字符串。格式必须是文本值。

返回类型

VARBYTE

示例

要将 '6162' 的十六进制表示形式转换为二进制值，请使用以下示例。结果会自动显示为二进制值的十六进制表示形式。

```
SELECT FROM_HEX('6162');

+----------+
```

```
| from_hex |
+----------+
|     6162 |
+----------+
```

FROM_VARBYTE 函数

FROM_VARBYTE 将二进制值转换为指定格式的字符串。

语法

```
FROM_VARBYTE(binary_value, format)
```

参数

binary_value

数据类型为 VARBYTE 的二进制值。

格式的日期和时间。

返回的字符串格式。不区分大小写的有效值为 hex、binary、utf8（还包括 utf-8 和 utf_8）和 base64。

返回类型

VARCHAR

示例

要将二进制值 'ab' 转换为十六进制，请使用以下示例。

```
SELECT FROM_VARBYTE('ab', 'hex');

+--------------+
| from_varbyte |
+--------------+
|         6162 |
+--------------+
```

要返回 '4d' 的二进制表示形式，请使用以下示例。'4d' 的二进制表示形式是字符串 01001101。

```
SELECT FROM_VARBYTE(FROM_HEX('4d'), 'binary');

+--------------+
| from_varbyte |
+--------------+
|     01001101 |
+--------------+
```

GETBIT 函数

GETBIT 返回指定索引处二进制值的位值。

语法

```
GETBIT(binary_value, index)
```

参数

binary_value

数据类型为 VARBYTE 的二进制值。

index

返回的二进制值中的位的索引号。二进制值是一个从 0 开始的位数组，它从最右边的位（最低有效位）索引到最左边的位（最高有效位）。

返回类型

INTEGER

示例

要返回二进制值 from_hex('4d') 在 2 索引处的位，请使用以下示例。'4d' 的二进制表示形式是 01001101。

```
SELECT GETBIT(FROM_HEX('4d'), 2);

+--------+
| getbit |
+--------+
```

```
|      1 |
+--------+
```

要返回由 from_hex('4d') 返回的二进制值在八个索引位置的位，请使用以下示例。'4d' 的二进制表示形式是 01001101。

```
SELECT GETBIT(FROM_HEX('4d'), 7), GETBIT(FROM_HEX('4d'), 6),
  GETBIT(FROM_HEX('4d'), 5), GETBIT(FROM_HEX('4d'), 4),
  GETBIT(FROM_HEX('4d'), 3), GETBIT(FROM_HEX('4d'), 2),
  GETBIT(FROM_HEX('4d'), 1), GETBIT(FROM_HEX('4d'), 0);

+--------+--------+--------+--------+--------+--------+--------+--------+
| getbit | getbit | getbit | getbit | getbit | getbit | getbit | getbit |
+--------+--------+--------+--------+--------+--------+--------+--------+
|      0 |      1 |      0 |      0 |      1 |      1 |      0 |      1 |
+--------+--------+--------+--------+--------+--------+--------+--------+
```

TO_HEX 函数

TO_HEX 将数字或二进制值转换为十六进制表示形式。

语法

```
TO_HEX(value)
```

参数

值

要转换的数值或二进制值（VARBYTE）。

返回类型

VARCHAR

示例

要将一个数值转换为其十六进制表示形式，请使用以下示例。

```
SELECT TO_HEX(2147676847);
```

```
+----------+
|  to_hex  |
+----------+
| 8002f2af |
+----------+
```

要将 'abc' 的 VARBYTE 表示形式转换为十六进制数值，请使用以下示例。

```
SELECT TO_HEX('abc'::VARBYTE);

+--------+
| to_hex |
+--------+
| 616263 |
+--------+
```

要创建一个表，将 'abc' 的 VARBYTE 表示形式插入到一个十六进制数值，然后选择具有该值的列，请使用以下示例。

```
CREATE TABLE t (vc VARCHAR);
INSERT INTO t SELECT TO_HEX('abc'::VARBYTE);
SELECT vc FROM t;

+--------+
|   vc   |
+--------+
| 616263 |
+--------+
```

要显示将 VARBYTE 值强制转换为 VARCHAR 时，格式为 UTF-8，请使用以下示例。

```
CREATE TABLE t (vc VARCHAR);
INSERT INTO t SELECT 'abc'::VARBYTE::VARCHAR;

SELECT vc FROM t;

+-----+
| vc  |
+-----+
| abc |
+-----+
```

TO_VARBYTE 函数

TO_VARBYTE 将指定格式的字符串转换为二进制值。

语法

```
TO_VARBYTE(string, format)
```

参数

string

CHAR 或 VARCHAR 字符串。

格式的日期和时间。

输入字符串的格式。不区分大小写的有效值为 hex、binary、utf8（还包括 utf-8 和 utf_8）和 base64。

返回类型

VARBYTE

示例

要将十六进制 6162 转换为二进制值，请使用以下示例。结果会自动显示为二进制值的十六进制表示形式。

```
SELECT TO_VARBYTE('6162', 'hex');

+------------+
| to_varbyte |
+------------+
|       6162 |
+------------+
```

要返回 4d 的二进制表示形式，请使用以下示例。“4d”的二进制表示形式是字符串 01001101。

```
SELECT TO_VARBYTE('01001101', 'binary');

+------------+
```

```
| to_varbyte |
+------------+
|         4d |
+------------+
```

要将 UTF-8 格式的字符串 'a' 转换为二进制值，请使用以下示例。结果会自动显示为二进制值的十六进制表示形式。

```
SELECT TO_VARBYTE('a', 'utf8');

+------------+
| to_varbyte |
+------------+
|         61 |
+------------+
```

要将十六进制的字符串 '4' 转换为二进制值，请使用以下示例。如果十六进制字符串长度为奇数，则需添加一个 0，形成一个有效的十六进制数字。

```
SELECT TO_VARBYTE('4', 'hex');

+------------+
| to_varbyte |
+------------+
|         04 |
+------------+
```

窗口函数

通过使用窗口函数，您可以更高效地创建分析业务查询。窗口函数运行于分区或结果集的“窗口”上，并为该窗口中的每个行返回一个值。相比之下，非窗口函数执行与结果集中的每个行相关的计算。与聚合结果行的分组函数不同，窗口函数在表的表达式中的保留所有行。

使用该窗口中的行集中的值计算返回的值。对于表中的每一行，窗口定义一组用于计算其他属性的行。窗口使用窗口规范（OVER 子句）进行定义并基于以下三个主要概念：

- 窗口分区，构成了行组（PARTITION 子句）
- 窗口排序，定义了每个分区中行的顺序或序列（ORDER BY 子句）
- 窗口框架，相对于每个行进行定义以进一步限制行集（ROWS 规范）

窗口函数是在查询中执行的最后一组操作（最后的 ORDER BY 子句除外）。所有联接和所有 WHERE、GROUP BY 和 HAVING 子句均在处理窗口函数前完成。因此，窗口函数只能显示在选择列表或 ORDER BY 子句中。您可以在一个具有不同框架子句的查询中使用多个窗口函数。您还可以在其他标量表达式（如 CASE）中使用窗口函数。

窗口函数语法摘要

窗口函数遵循标准语法，如下所示。

```
function (expression) OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list [ frame_clause ] ] )
```

其中，function 是本部分介绍的函数之一。

expr_list 如下所示。

```
expression | column_name [, expr_list ]
```

order_list 如下所示。

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

frame_clause 如下所示。

```
ROWS
{ UNBOUNDED PRECEDING | unsigned_value PRECEDING | CURRENT ROW } |

{ BETWEEN
{ UNBOUNDED PRECEDING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW}
AND
{ UNBOUNDED FOLLOWING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW }}
```

参数

函数

窗口函数。有关详细信息，请参阅各个函数描述。

OVER

定义窗口规范的子句。OVER 子句是窗口函数必需的，并可区分窗口函数与其他 SQL 函数。

PARTITION BY expr_list

（可选）PARTITION BY 子句将结果集细分为分区，与 GROUP BY 子句很类似。如果存在分区子句，则为每个分区中的行计算该函数。如果未指定任何分区子句，则一个分区包含整个表，并为整个表计算该函数。

排名函数 DENSE_RANK、NTILE、RANK 和 ROW_NUMBER 需要全局比较结果集中的所有行。使用 PARTITION BY 子句时，查询优化程序可通过根据分区跨多个切片分布工作负载来并行运行每个聚合。如果不存在 PARTITION BY 子句，则必须在一个切片上按顺序运行聚合步骤，这可能对性能产生显著的负面影响，特别是对于大型集群。

Amazon Redshift 不支持 PARTITION BY 子句中的字符串文本。

ORDER BY order_list

（可选）窗口函数将应用于每个分区中根据 ORDER BY 中的顺序规范排序的行。此 ORDER BY 子句与 frame_clause 中的 ORDER BY 子句不同且完全不相关。ORDER BY 子句可在没有 PARTITION BY 子句的情况下使用。

对于排名函数，ORDER BY 子句确定排名值的度量。对于聚合函数，分区的行必须在为每个框架计算聚合函数之前进行排序。有关窗口函数的更多信息，请参阅 窗口函数。

顺序列表中需要列标识符或计算结果为列标识符的表达式。常数和常数表达式都不可用作列名称的替代。

NULLS 值将被视为其自己的组，并根据 NULLS FIRST 或 NULLS LAST 选项进行排序和排名。默认情况下，按 ASC 顺序最后对 NULL 值进行排序和排名，按 DESC 顺序首先对 NULL 值进行排序和排名。

Amazon Redshift 不支持 ORDER BY 子句中的字符串文本。

如果省略 ORDER BY 子句，则行的顺序是不确定的。

> **Note**
>
> 在任何并行系统（如 Amazon Redshift）中，如果 ORDER BY 子句未生成数据的唯一排序和总排序，则行的顺序是不确定的。也就是说，如果 ORDER BY 表达式生成重复的值（部

分排序），则这些行的返回顺序可能会因 Amazon Redshift 的运行而异。反过来，窗口函数可能返回意外的或不一致的结果。有关更多信息，请参阅 窗口函数的唯一数据排序。

column_name

执行分区或排序操作所依据的列的名称。

ASC | DESC

一个定义表达式的排序顺序的选项，如下所示：

- ASC：升序（例如，按数值的从低到高的顺序和字符串的从 A 到 Z 的顺序）。如果未指定选项，则默认情况下将按升序对数据进行排序。
- DESC：降序（按数值的从高到低的顺序和字符串的从 Z 到 A 的顺序）。

NULLS FIRST | NULLS LAST

指定是应首先对 NULL 值进行排序（非 null 值之前）还是最后对 NULL 值进行排序（非 null 值之后）的选项。默认情况下，按 ASC 顺序最后对 NULLS 进行排序和排名，按 DESC 顺序首先对 NULLS 进行排序和排名。

frame_clause

对于聚合函数，框架子句在使用 ORDER BY 时进一步优化函数窗口中的行集。它使您可以包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。

frame 子句不适用于排名函数。同时，在聚合函数的 OVER 子句中未使用 ORDER BY 子句时不需要框架子句。如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。

未指定 ORDER BY 子句时，隐式框架是无界的：等同于 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING。

ROWS

此子句通过从当前行中指定物理偏移来定义窗口框架。

此子句指定当前行中的值将并入的当前窗口或分区中的行。它使用指定行位置的参数，行位置可位于当前行之前或之后。所有窗口框架的参考点为当前行。当窗口框架向前滑向分区中时，每个行会依次成为当前行。

框架可以是一组超过并包括当前行的行。

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

或者可以是两个边界之间的一组行。

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING 指示窗口从分区的第一行开始；offset PRECEDING 指示窗口开始于等同于当前行之前的偏移值的行数。UNBOUNDED PRECEDING 是默认值。

CURRENT ROW 指示窗口在当前行开始或结束。

UNBOUNDED FOLLOWING 指示窗口在分区的最后一行结束；offset FOLLOWING 指示窗口结束于等同于当前行之后的偏移值的行数。

offset 标识当前行之前或之后的实际行数。在这种情况下，offset 必须为计算结果为正数值的常数。例如，5 FOLLOWING 将在当前行之后的第 5 行结束框架。

其中，未指定 BETWEEN，框架受当前行隐式限制。例如，`ROWS 5 PRECEDING` 等于 `ROWS BETWEEN 5 PRECEDING AND CURRENT ROW`。同时，`ROWS UNBOUNDED FOLLOWING` 等于 `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`。

> **Note**
>
> 您无法指定起始边界大于结束边界的框架。例如，您无法指定以下任一框架。
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

窗口函数的唯一数据排序

如果窗口函数的 ORDER BY 子句不生成数据的唯一排序和总排序，则行的顺序是不确定的。如果 ORDER BY 表达式生成重复的值（部分排序），则这些行的返回顺序可能会在多次运行中有所不同。在这种情况下，窗口函数还可能返回意外的或不一致的结果。

例如，以下查询在多次运行中返回了不同的结果。出现这些不同的结果是因为 `order by dateid` 未生成 SUM 窗口函数的数据的唯一排序。

```
select dateid, pricepaid,
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid |   sumpaid
--------+-----------+-------------
1827 |   1730.00 |     1730.00
1827 |    708.00 |     2438.00
1827 |    234.00 |     2672.00
...

select dateid, pricepaid,
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid |   sumpaid
--------+-----------+-------------
1827 |    234.00 |      234.00
1827 |    472.00 |      706.00
1827 |    347.00 |     1053.00
...
```

在这种情况下，向该窗口函数添加另一个 ORDER BY 列可解决此问题。

```
select dateid, pricepaid,
sum(pricepaid) over(order by dateid, pricepaid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid | sumpaid
--------+-----------+---------
1827 |    234.00 |  234.00
1827 |    337.00 |  571.00
1827 |    347.00 |  918.00
...
```

支持的函数

Amazon Redshift 支持以下两种类型的窗口函数：聚合和排名。

以下是支持的聚合函数：

- AVG 窗口函数
- COUNT 窗口函数
- CUME_DIST 开窗函数
- DENSE_RANK 窗口函数
- FIRST_VALUE 窗口函数
- LAG 窗口函数
- LAST_VALUE 窗口函数
- LEAD 窗口函数
- LISTAGG 窗口函数
- MAX 窗口函数
- MEDIAN 开窗函数
- MIN 窗口函数
- NTH_VALUE 窗口函数
- PERCENTILE_CONT 开窗函数
- PERCENTILE_DISC 开窗函数
- RATIO_TO_REPORT 开窗函数
- STDDEV_SAMP 和 STDDEV_POP 窗口函数（STDDEV_SAMP 和 STDDEV 是同义词）
- SUM 窗口函数
- VAR_SAMP 和 VAR_POP 窗口函数（VAR_SAMP 和 VARIANCE 是同义词）

以下是支持的排名函数：

- DENSE_RANK 窗口函数
- NTILE 窗口函数
- PERCENT_RANK 开窗函数
- RANK 窗口函数
- ROW_NUMBER 窗口函数

窗口函数示例的示例表

您可以通过每个函数描述找到特定的窗口函数示例。其中一些示例使用一个名为 WINSALES 的表，该表包含 11 个行，如下所示。

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
30001	8/2/2003	3	B	10	10
10001	12/24/2003	1	C	10	10
10005	12/24/2003	1	A	30	
40001	1/9/2004	4	A	40	
10006	1/18/2004	1	C	10	
20001	2/12/2004	2	B	20	20
40005	2/12/2004	4	A	10	10
20002	2/16/2004	2	C	20	20
30003	4/18/2004	3	B	15	
30004	4/18/2004	3	B	20	
30007	9/7/2004	3	C	30	

以下脚创建并填充示例 WINSALES 表。

```
CREATE TABLE winsales(
  salesid int,
  dateid date,
  sellerid int,
  buyerid char(10),
  qty int,
  qty_shipped int);

INSERT INTO winsales VALUES
  (30001, '8/2/2003', 3, 'b', 10, 10),
  (10001, '12/24/2003', 1, 'c', 10, 10),
  (10005, '12/24/2003', 1, 'a', 30, null),
  (40001, '1/9/2004', 4, 'a', 40, null),
  (10006, '1/18/2004', 1, 'c', 10, null),
```

```
  (20001, '2/12/2004', 2, 'b', 20, 20),
  (40005, '2/12/2004', 4, 'a', 10, 10),
  (20002, '2/16/2004', 2, 'c', 20, 20),
  (30003, '4/18/2004', 3, 'b', 15, null),
  (30004, '4/18/2004', 3, 'b', 20, null),
  (30007, '9/7/2004', 3, 'c', 30, null);
```

AVG 窗口函数

AVG 窗口函数返回输入表达式值的平均值（算术平均值）。AVG 函数使用数值并忽略 NULL 值。

语法

```
AVG ( [ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值以进行计数。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 AVG 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

AVG 函数支持的参数类型为 SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 和 DOUBLE PRECISION。

AVG 函数支持的返回类型为：

- 适用于 SMALLINT 或 INTEGER 参数的 BIGINT
- 适用于 BIGINT 参数的 NUMERIC
- 适用于浮点参数的 DOUBLE PRECISION

示例

以下示例按日期计算销量的移动平均数；按日期 ID 和销售 ID 对结果进行排序：

```
select salesid, dateid, sellerid, qty,
avg(qty) over
(order by dateid, salesid rows unbounded preceding) as avg
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | avg
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
10005 | 2003-12-24 |        1 |  30 |  16
40001 | 2004-01-09 |        4 |  40 |  22
10006 | 2004-01-18 |        1 |  10 |  20
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  18
20002 | 2004-02-16 |        2 |  20 |  18
30003 | 2004-04-18 |        3 |  15 |  18
30004 | 2004-04-18 |        3 |  20 |  18
30007 | 2004-09-07 |        3 |  30 |  19
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

COUNT 窗口函数

COUNT 窗口函数对由表达式定义的行计数。

COUNT 函数具有两个变体。COUNT(*) 对目标表中的所有行计数，无论它们是否包含 null 值。COUNT(expression) 计算某个特定列或表达式中带非 NULL 值的行的数量。

语法

```
COUNT ( * | [ ALL ] expression) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值以进行计数。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 COUNT 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

COUNT 函数支持所有参数数据类型。

COUNT 函数支持的返回类型是 BIGINT。

示例

以下示例从数据窗口的开头显示销售 ID、数量和所有行的计数：

```
select salesid, qty,
count(*) over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | count
---------+-----+-----
10001 |  10 |   1
10005 |  30 |   2
10006 |  10 |   3
20001 |  20 |   4
20002 |  20 |   5
30001 |  10 |   6
30003 |  15 |   7
30004 |  20 |   8
30007 |  30 |   9
40001 |  40 |   10
40005 |  10 |   11
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例从数据窗口的开头显示销售 ID、数量和非 null 行的计数。（在 WINSALES 表中，QTY_SHIPPED 列包含一些 NULL 值。）

```
select salesid, qty, qty_shipped,
count(qty_shipped)
over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | qty_shipped | count
---------+-----+-------------+-------
```

```
10001 |  10 |          10 |   1
10005 |  30 |              |   1
10006 |  10 |              |   1
20001 |  20 |          20 |   2
20002 |  20 |          20 |   3
30001 |  10 |          10 |   4
30003 |  15 |              |   4
30004 |  20 |              |   4
30007 |  30 |              |   4
40001 |  40 |              |   4
40005 |  10 |          10 |   5
(11 rows)
```

CUME_DIST 开窗函数

计算某个窗口或分区中某个值的累积分布。假定升序排序，则使用以下公式确定累积分布：

```
count of rows with values <= x / count of rows in the window or partition
```

其中，x 等于 ORDER BY 子句中指定的列的当前行中的值。以下数据集说明了此公式的使用：

```
Row# Value   Calculation    CUME_DIST
1        2500    (1)/(5)    0.2
2        2600    (2)/(5)    0.4
3        2800    (3)/(5)    0.6
4        2900    (4)/(5)    0.8
5        3100    (5)/(5)    1.0
```

返回值范围介于 0 和 1（含 1）之间。

语法

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

参数

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口框架规范。

PARTITION BY partition_expression

可选。一个设置 OVER 子句中每个组的记录范围的表达式。

ORDER BY order_list

用于计算累积分布的表达式。该表达式必须具有数字数据类型或可隐式转换为 1。如果省略 ORDER BY，则所有行的返回值为 1。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅 窗口函数的唯一数据排序。

返回类型

FLOAT8

示例

以下示例计算每个卖家的销量的累积分布：

```
select sellerid, qty, cume_dist()
over (partition by sellerid order by qty)
from winsales;

sellerid   qty    cume_dist
-------------------------------------------------
1          10.00    0.33
1          10.64    0.67
1          30.37    1
3          10.04    0.25
3          15.15    0.5
3          20.75    0.75
3          30.55    1
2          20.09    0.5
2          20.12    1
4          10.12    0.5
4          40.23    1
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

DENSE_RANK 窗口函数

DENSE_RANK 窗口函数基于 OVER 子句中的 ORDER BY 表达式确定一组值中的一个值的排名。如果存在可选的 PARTITION BY 子句，则为每个行组重置排名。带符合排名标准的相同值的行接收相同的排名。DENSE_RANK 函数与 RANK 存在以下一点不同：如果两个或两个以上的行结合，则一系列排名的值之间没有间隔。例如，如果两个行的排名为 1，则下一个排名则为 2。

您可以在同一查询中包含带有不同的 PARTITION BY 和 ORDER BY 子句的排名函数。

语法

```
DENSE_RANK() OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

参数

( )

该函数没有参数，但需要空括号。

OVER

适用于 DENSE_RANK 函数的窗口子句。

PARTITION BY expr_list

（可选）一个或多个用于定义窗口的表达式。

ORDER BY order_list

（可选）排名值基于的表达式。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。如果省略 ORDER BY，则所有行的返回值为 1。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅 窗口函数的唯一数据排序。

返回类型

INTEGER

示例

以下示例使用窗口函数的示例表。有关更多信息，请参阅 窗口函数示例的示例表。

以下示例按销量对表进行排序，并将紧密排名和常规排名分配给每个行。在应用窗口函数结果后，对结果进行排序。

```
SELECT salesid, qty,
DENSE_RANK() OVER(ORDER BY qty DESC) AS d_rnk,
RANK() OVER(ORDER BY qty DESC) AS rnk
FROM winsales
ORDER BY 2,1;

+---------+-----+-------+-----+
| salesid | qty | d_rnk | rnk |
+---------+-----+-------+-----+
|   10001 |  10 |     5 |   8 |
|   10006 |  10 |     5 |   8 |
|   30001 |  10 |     5 |   8 |
|   40005 |  10 |     5 |   8 |
|   30003 |  15 |     4 |   7 |
|   20001 |  20 |     3 |   4 |
|   20002 |  20 |     3 |   4 |
|   30004 |  20 |     3 |   4 |
|   10005 |  30 |     2 |   2 |
|   30007 |  30 |     2 |   2 |
|   40001 |  40 |     1 |   1 |
+---------+-----+-------+-----+
```

在同一查询中一起使用 DENSE_RANK 和 RANK 函数时，记下已分配给同一组行的排名的差异。

以下示例按 sellerid 对表进行分区，按数量对每个分区进行排序，并为每个行分配紧密排名。在应用窗口函数结果后，对结果进行排序。

```
SELECT salesid, sellerid, qty,
DENSE_RANK() OVER(PARTITION BY sellerid ORDER BY qty DESC) AS d_rnk
FROM winsales
ORDER BY 2,3,1;

+---------+----------+-----+-------+
| salesid | sellerid | qty | d_rnk |
+---------+----------+-----+-------+
|   10001 |        1 |  10 |     2 |
```

```
|   10006 |        1 |  10 |     2 |
|   10005 |        1 |  30 |     1 |
|   20001 |        2 |  20 |     1 |
|   20002 |        2 |  20 |     1 |
|   30001 |        3 |  10 |     4 |
|   30003 |        3 |  15 |     3 |
|   30004 |        3 |  20 |     2 |
|   30007 |        3 |  30 |     1 |
|   40005 |        4 |  10 |     2 |
|   40001 |        4 |  40 |     1 |
+---------+----------+-----+-------+
```

要成功使用上一个示例，请使用以下命令在 WINSALES 表中插入一行。该行与另一行具有相同的 buyerid、sellerid 和 qtysold。这将导致上一个示例中的两行并列，从而显示 DENSE_RANK 和 RANK 函数之间的差异。

```
INSERT INTO winsales VALUES(30009, '2/2/2003', 3, 'b', 20, NULL);
```

以下示例按 buyerid 和 sellerid 对表进行分区，按数量对每个分区进行排序，并为每个行分配紧密排名和常规排名。在应用窗口函数后，对结果进行排序。

```
SELECT salesid, sellerid, qty, buyerid,
DENSE_RANK() OVER(PARTITION BY buyerid, sellerid ORDER BY qty DESC) AS d_rnk,
RANK() OVER (PARTITION BY buyerid, sellerid ORDER BY qty DESC) AS rnk
FROM winsales
ORDER BY rnk;

+---------+----------+-----+---------+-------+-----+
| salesid | sellerid | qty | buyerid | d_rnk | rnk |
+---------+----------+-----+---------+-------+-----+
|   20001 |        2 |  20 | b       |     1 |   1 |
|   30007 |        3 |  30 | c       |     1 |   1 |
|   10006 |        1 |  10 | c       |     1 |   1 |
|   10005 |        1 |  30 | a       |     1 |   1 |
|   20002 |        2 |  20 | c       |     1 |   1 |
|   30009 |        3 |  20 | b       |     1 |   1 |
|   40001 |        4 |  40 | a       |     1 |   1 |
|   30004 |        3 |  20 | b       |     1 |   1 |
|   10001 |        1 |  10 | c       |     1 |   1 |
|   40005 |        4 |  10 | a       |     2 |   2 |
|   30003 |        3 |  15 | b       |     2 |   3 |
|   30001 |        3 |  10 | b       |     3 |   4 |
```

```
+---------+----------+-----+---------+-------+-----+
```

FIRST_VALUE 窗口函数

在提供一组已排序行的情况下，FIRST_VALUE 返回有关窗口框架中的第一行的指定表达式的值。

有关选择框架中最后一行的信息，请参阅 LAST_VALUE 窗口函数。

语法

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

IGNORE NULLS

将此选项与 FIRST_VALUE 结合使用时，该函数返回不为 NULL 的框架中的第一个值（如果所有值为 NULL，则返回 NULL）。

RESPECT NULLS

指示 Amazon Redshift 应包含 null 值以确定要使用的行。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

引入函数的窗口子句。

PARTITION BY expr_list

依据一个或多个表达式定义函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY 子句，则 ORDER BY 对整个表进行排序。如果指定 ORDER BY 子句，则还必须指定 frame_clause。

FIRST_VALUE 函数的结果取决于数据的排序。在以下情况下，结果是不确定的：

- 当未指定 ORDER BY 子句且一个分区包含一个表达式的两个不同的值时
- 当表达式的计算结果为对应于 ORDER BY 列表中同一值的不同值时。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

返回类型

这些函数支持使用原始 Amazon Redshift 数据类型的表达式。返回类型与 expression 的数据类型相同。

示例

以下示例使用 TICKIT 样本数据中的 VENUE 表。有关更多信息，请参阅 示例数据库。

以下示例返回 VENUE 表中每个场地的座位数，同时按容量对结果进行排序（从高到低）。FIRST_VALUE 函数用于选择与框架中的第一行对应的场地的名称：在这种情况下，为座位数最多的行。按州对结果进行分区，以便当 VENUESTATE 值发生更改时，会选择一个新的第一个值。窗口框架是无界的，因此为每个分区中的每个行选择相同的第一个值。

对于加利福利亚，Qualcomm Stadium 具有最大座位数 (70561)，此名称是 CA 分区中所有行的第一个值。

```
select venuestate, venueseats, venuename,
first_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |           venuename            |         first_value
-----------+------------+--------------------------------
+-----------------------------
CA         |      70561 | Qualcomm Stadium               | Qualcomm Stadium
CA         |      69843 | Monster Park                   | Qualcomm Stadium
CA         |      63026 | McAfee Coliseum                | Qualcomm Stadium
CA         |      56000 | Dodger Stadium                 | Qualcomm Stadium
CA         |      45050 | Angel Stadium of Anaheim       | Qualcomm Stadium
```

```
CA         |      42445 | PETCO Park                     | Qualcomm Stadium
CA         |      41503 | AT&T Park                      | Qualcomm Stadium
CA         |      22000 | Shoreline Amphitheatre         | Qualcomm Stadium
CO         |      76125 | INVESCO Field                  | INVESCO Field
CO         |      50445 | Coors Field                    | INVESCO Field
DC         |      41888 | Nationals Park                 | Nationals Park
FL         |      74916 | Dolphin Stadium                | Dolphin Stadium
FL         |      73800 | Jacksonville Municipal Stadium | Dolphin Stadium
FL         |      65647 | Raymond James Stadium          | Dolphin Stadium
FL         |      36048 | Tropicana Field                | Dolphin Stadium
...
```

下面的示例介绍如何使用 IGNORE NULLS 选项，并且事先向 VENUE 表添加一个新行：

```
insert into venue values(2000,null,'Stanford','CA',90000);
```

此新行为 VENUENAME 列包含一个 NULL 值。现在，重复本部分中前面介绍的 FIRST_VALUE 查询：

```
select venuestate, venueseats, venuename,
first_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |         venuename          | first_value
-----------+------------+----------------------------+-------------
CA         |      90000 | NULL                       | NULL
CA         |      70561 | Qualcomm Stadium           | NULL
CA         |      69843 | Monster Park               | NULL
...
```

因为新行包含最高的 VENUESEATS 值 (90000) 且其 VENUENAME 为 NULL，所以 FIRST_VALUE 函数为 CA 分区返回 NULL。要在函数计算中忽略诸如此类的行，请向函数参数添加 IGNORE NULLS 选项：

```
select venuestate, venueseats, venuename,
first_value(venuename) ignore nulls
over(partition by venuestate
order by venueseats desc
```

```
rows between unbounded preceding and unbounded following)
from (select * from venue where venuestate='CA')
order by venuestate;

venuestate | venueseats |         venuename          |   first_value
-----------+------------+----------------------------+------------------
CA         |      90000 | NULL                       | Qualcomm Stadium
CA         |      70561 | Qualcomm Stadium           | Qualcomm Stadium
CA         |      69843 | Monster Park               | Qualcomm Stadium
...
```

LAG 窗口函数

LAG 窗口函数返回位于分区中当前行的上方（之前）的某个给定偏移量位置的行的值。

语法

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

参数

value_expr

对其执行函数的目标列或表达式。

offset

一个可选参数，该参数指定要返回其值的当前行前面的行数。偏移量可以是常量整数或计算结果为整数的表达式。如果您未指定偏移量，则 Amazon Redshift 使用 1 作为默认值。偏移量为 0 表示当前行。

IGNORE NULLS

一个可选规范，该规范指示 Amazon Redshift 应跳过 null 值以确定要使用的行。如果未列出 IGNORE NULLS，则包含 Null 值。

> **Note**
>
> 您可以使用 NVL 或 COALESCE 表达式将 null 值替换为另一个值。有关更多信息，请参阅 NVL 和 COALESCE 函数。

RESPECT NULLS

指示 Amazon Redshift 应包含 null 值以确定要使用的行。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

指定窗口分区和排序。OVER 子句不能包含窗口框架规范。

PARTITION BY window_partition

一个可选参数，该参数设置 OVER 子句中每个组的记录范围。

ORDER BY window_ordering

对每个分区中的行进行排序。

LAG 窗口函数支持使用任何 Amazon Redshift 数据类型的表达式。返回类型与 value_expr 的类型相同。

示例

以下示例显示已售给买家 ID 为 3 的买家的票数以及买家 3 的购票时间。要将每个销售与买家 3 的上一销售进行比较，查询要返回每个销售的上一销量。由于 1/16/2008 之前未进行购买，则第一个上一销量值为 null：

```
select buyerid, saletime, qtysold,
lag(qtysold,1) over (order by buyerid, saletime) as prev_qtysold
from sales where buyerid = 3 order by buyerid, saletime;

buyerid |      saletime       | qtysold | prev_qtysold
---------+---------------------+---------+--------------
3 | 2008-01-16 01:06:09 |       1 |
3 | 2008-01-28 02:10:01 |       1 |            1
3 | 2008-03-12 10:39:53 |       1 |            1
3 | 2008-03-13 02:56:07 |       1 |            1
3 | 2008-03-29 08:21:39 |       2 |            1
3 | 2008-04-27 02:39:01 |       1 |            2
3 | 2008-08-16 07:04:37 |       2 |            1
3 | 2008-08-22 11:45:26 |       2 |            2
3 | 2008-09-12 09:11:25 |       1 |            2
3 | 2008-10-01 06:22:37 |       1 |            1
3 | 2008-10-20 01:55:51 |       2 |            1
3 | 2008-10-28 01:30:40 |       1 |            2
```

```
(12 rows)
```

LAST_VALUE 窗口函数

在提供一组已排序行的情况下，LAST_VALUE 函数返回有关框架中最后一行的表达式的值。

有关选择框架中第一行的信息，请参阅 FIRST_VALUE 窗口函数。

语法

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

IGNORE NULLS

该函数返回不为 NULL 的框架中的最后一个值（如果所有值为 NULL，则返回 NULL）。

RESPECT NULLS

指示 Amazon Redshift 应包含 null 值以确定要使用的行。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

引入函数的窗口子句。

PARTITION BY expr_list

依据一个或多个表达式定义函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY 子句，则 ORDER BY 对整个表进行排序。如果指定 ORDER BY 子句，则还必须指定 frame_clause。

结果取决于数据的排序。在以下情况下，结果是不确定的：

- 当未指定 ORDER BY 子句且一个分区包含一个表达式的两个不同的值时

- 当表达式的计算结果为对应于 ORDER BY 列表中同一值的不同值时。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

返回类型

这些函数支持使用原始 Amazon Redshift 数据类型的表达式。返回类型与 expression 的数据类型相同。

示例

以下示例使用 TICKIT 样本数据中的 VENUE 表。有关更多信息，请参阅 示例数据库。

以下示例返回 VENUE 表中每个场地的座位数，同时按容量对结果进行排序（从高到低）。LAST_VALUE 函数用于选择与框架中的最后一行对应的场地的名称：在本例中，为座位数最少的行。按州对结果进行分区，以便当 VENUESTATE 值发生更改时，会选择一个新的最后一个值。窗口框架是无界的，因此为每个分区中的每个行选择相同的最后一个值。

对于加利福利亚，为该分区中的每个行返回 Shoreline Amphitheatre，因为它具有最小座位数 (22000)。

```
select venuestate, venueseats, venuename,
last_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |          venuename            |          last_value
-----------+------------+-------------------------------
+-----------------------------
CA         |      70561 | Qualcomm Stadium              | Shoreline Amphitheatre
CA         |      69843 | Monster Park                  | Shoreline Amphitheatre
CA         |      63026 | McAfee Coliseum               | Shoreline Amphitheatre
CA         |      56000 | Dodger Stadium                | Shoreline Amphitheatre
CA         |      45050 | Angel Stadium of Anaheim      | Shoreline Amphitheatre
CA         |      42445 | PETCO Park                    | Shoreline Amphitheatre
```

```
CA         |      41503 | AT&T Park                       | Shoreline Amphitheatre
CA         |      22000 | Shoreline Amphitheatre          | Shoreline Amphitheatre
CO         |      76125 | INVESCO Field                   | Coors Field
CO         |      50445 | Coors Field                     | Coors Field
DC         |      41888 | Nationals Park                  | Nationals Park
FL         |      74916 | Dolphin Stadium                 | Tropicana Field
FL         |      73800 | Jacksonville Municipal Stadium | Tropicana Field
FL         |      65647 | Raymond James Stadium           | Tropicana Field
FL         |      36048 | Tropicana Field                 | Tropicana Field
...
```

LEAD 窗口函数

LEAD 窗口函数返回位于分区中当前行的下方（之后）的某个给定偏移量位置的行的值。

语法

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

参数

value_expr

对其执行函数的目标列或表达式。

offset

一个可选参数，该参数指定要返回其值的当前行后面的行数。偏移量可以是常量整数或计算结果为整数的表达式。如果您未指定偏移量，则 Amazon Redshift 使用 1 作为默认值。偏移量为 0 表示当前行。

IGNORE NULLS

一个可选规范，该规范指示 Amazon Redshift 应跳过 null 值以确定要使用的行。如果未列出 IGNORE NULLS，则包含 Null 值。

> **Note**
>
> 您可以使用 NVL 或 COALESCE 表达式将 null 值替换为另一个值。有关更多信息，请参阅 NVL 和 COALESCE 函数。

RESPECT NULLS

指示 Amazon Redshift 应包含 null 值以确定要使用的行。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

指定窗口分区和排序。OVER 子句不能包含窗口框架规范。

PARTITION BY window_partition

一个可选参数，该参数设置 OVER 子句中每个组的记录范围。

ORDER BY window_ordering

对每个分区中的行进行排序。

LEAD 窗口函数支持使用任何 Amazon Redshift 数据类型的表达式。返回类型与 value_expr 的类型相同。

示例

以下示例提供了 SALES 表中于 2008 年 1 月 1 日与 1 月 2 日已售票的事件的佣金以及为后续销售中售票所付的佣金。以下示例使用 TICKIT 示例数据库。有关更多信息，请参阅 示例数据库。

```
SELECT eventid, commission, saletime, LEAD(commission, 1) over ( ORDER BY saletime ) AS
 next_comm
FROM sales
WHERE saletime BETWEEN '2008-01-09 00:00:00' AND '2008-01-10 12:59:59'
LIMIT 10;

+---------+------------+---------------------+-----------+
| eventid | commission |      saletime       | next_comm |
+---------+------------+---------------------+-----------+
|    1664 |       13.2 | 2008-01-09 01:00:21 |      69.6 |
|     184 |       69.6 | 2008-01-09 01:00:36 |     116.1 |
|    6870 |      116.1 | 2008-01-09 01:02:37 |      11.1 |
|    3718 |       11.1 | 2008-01-09 01:05:19 |     205.5 |
|    6772 |      205.5 | 2008-01-09 01:14:04 |      38.4 |
|    3074 |       38.4 | 2008-01-09 01:26:50 |     209.4 |
|    5254 |      209.4 | 2008-01-09 01:29:16 |      26.4 |
|    3724 |       26.4 | 2008-01-09 01:40:09 |      57.6 |
|    5303 |       57.6 | 2008-01-09 01:40:21 |      51.6 |
```

```
|    3678 |       51.6 | 2008-01-09 01:42:54 |      43.8 |
+---------+------------+---------------------+-----------+
```

LISTAGG 窗口函数

对于查询中的每个组，LISTAGG 窗口函数根据 ORDER BY 表达式对该组的行进行排序，然后将值串联成一个字符串。

LISTAGG 是仅计算节点函数。如果查询不引用用户定义的表或 Amazon Redshift 系统表，该函数将返回错误。有关更多信息，请参阅 查询目录表。

语法

```
LISTAGG( [DISTINCT] expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
OVER ( [PARTITION BY partition_expression] )
```

参数

DISTINCT

（可选）用于在串联之前消除指定表达式中重复值的子句。尾部空格将被忽略，因此会将字符串 'a' 和 'a ' 视为重复值。LISTAGG 将使用遇到的第一个值。有关更多信息，请参阅 尾部空格的意义。

aggregate_expression

提供要聚合的值的任何有效表达式（如列名称）。忽略 NULL 值和空字符串。

分隔符

（可选）将用于分隔串联的值的字符串常数。默认值为 NULL。

WITHIN GROUP (ORDER BY order_list)

（可选）用于指定聚合值的排序顺序的子句。仅在 ORDER BY 提供唯一排序时是确定性的。默认为聚合所有行并返回一个值。

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY partition_expression

（可选）设置 OVER 子句中每个组的记录范围。

返回值

VARCHAR(MAX)。如果结果集大于最大 VARCHAR 大小（64K – 1 或 65535），则 LISTAGG 返回以下错误：

```
Invalid operation: Result size exceeds LISTAGG limit
```

示例

以下示例使用 WINSALES 表。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例返回卖家 ID 的列表（按卖家 ID 排序）。

```
select listagg(sellerid)
within group (order by sellerid)
over() from winsales;

  listagg
------------
 11122333344
...
...
 11122333344
 11122333344
   (11 rows)
```

以下示例返回买家 B 的卖家 ID 的列表（按日期排序）。

```
select listagg(sellerid)
within group (order by dateid)
over () as seller
from winsales
where buyerid = 'b' ;

  seller
---------
    3233
    3233
```

```
    3233
    3233
```

以下示例返回买家 B 的销售日期的逗号分隔的列表。

```
select listagg(dateid,',')
within group (order by sellerid desc,salesid asc)
over () as dates
from winsales
where buyerid  = 'b';

            dates
----------------------------------------
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
```

以下示例使用 DISTINCT 返回买家 B 的唯一销售日期的列表。

```
select listagg(distinct dateid,',')
within group (order by sellerid desc,salesid asc)
over () as dates
from winsales
where buyerid  = 'b';

           dates
-------------------------------
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
```

以下示例返回每个买家 ID 的销售 ID 的逗号分隔的列表。

```
select buyerid,
listagg(salesid,',')
within group (order by salesid)
over (partition by buyerid) as sales_id
from winsales
order by buyerid;
```

```
+---------+-------------------------+
| buyerid |        sales_id         |
+---------+-------------------------+
| a       | 10005,40001,40005       |
| a       | 10005,40001,40005       |
| a       | 10005,40001,40005       |
| b       | 20001,30001,30003,30004 |
| b       | 20001,30001,30003,30004 |
| b       | 20001,30001,30003,30004 |
| b       | 20001,30001,30003,30004 |
| c       | 10001,10006,20002,30007 |
| c       | 10001,10006,20002,30007 |
| c       | 10001,10006,20002,30007 |
| c       | 10001,10006,20002,30007 |
+---------+-------------------------+
```

MAX 窗口函数

MAX 窗口函数返回最大输入表达式值。MAX 函数使用数值并忽略 NULL 值。

语法

```
MAX ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

一个指定聚合函数的窗口子句的子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 MAX 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

接受任何数据类型作为输入。返回与 expression 相同的数据类型。

示例

以下示例从数据窗口的开头显示销售 ID、数量和最大数量：

```
select salesid, qty,
max(qty) over (order by salesid rows unbounded preceding) as max
from winsales
order by salesid;

salesid | qty | max
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  30
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  30
30001 |  10 |  30
30003 |  15 |  30
30004 |  20 |  30
30007 |  30 |  30
40001 |  40 |  40
40005 |  10 |  40
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例在受限制的框架中显示销售 ID、数量和最大数量：

```
select salesid, qty,
```

```
max(qty) over (order by salesid rows between 2 preceding and 1 preceding) as max
from winsales
order by salesid;

salesid | qty | max
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  20
30001 |  10 |  20
30003 |  15 |  20
30004 |  20 |  15
30007 |  30 |  20
40001 |  40 |  30
40005 |  10 |  40
(11 rows)
```

MEDIAN 开窗函数

计算某个窗口或分区中值的范围的中间值。忽略该范围中的 NULL 值。

MEDIAN 是一种假定连续分布模型的逆分布函数。

MEDIAN 是仅计算节点函数。如果查询不引用用户定义的表或 Amazon Redshift 系统表，该函数将返回错误。

语法

```
MEDIAN ( median_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

参数

median_expression

一个提供要为其确定中间值的值的表达式（例如列名）。该表达式必须具有数字数据类型或日期时间数据类型或可隐式转换为 1。

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY partition_expression

可选。一个设置 OVER 子句中每个组的记录范围的表达式。

数据类型

返回类型由 median_expression 的数据类型确定。下表显示了每种 median_expression 数据类型的返回类型。

输入类型	返回类型
INT2、INT4、INT8、NUMERIC、DECIMAL	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE

使用说明

如果 median_expression 参数是使用 38 位最大精度定义的 DECIMAL 数据类型，则 MEDIAN 可能将返回不准确的结果或错误。如果 MEDIAN 函数的返回值超过 38 位，则结果将截断以符合规范，这将导致精度降低。如果在插值期间，中间结果超出最大精度，则会发生数值溢出且函数会返回错误。要避免这些情况，建议使用具有较低精度的数据类型或将 median_expression 参数转换为较低精度。

例如，带 DECIMAL 参数的 SUM 函数返回 38 位的默认精度。结果的小数位数与参数的小数位数相同。因此，例如，DECIMAL(5,2) 列的 SUM 返回 DECIMAL(38,2) 数据类型。

以下示例在 MEDIAN 函数的 median_expression 参数中使用 SUM 函数。PRICEPAID 列的数据类型是 DECIMAL (8,2)，因此 SUM 函数返回 DECIMAL(38,2)。

```
select salesid, sum(pricepaid), median(sum(pricepaid))
over() from sales where salesid < 10 group by salesid;
```

要避免潜在的精度降低或溢出错误，请将结果转换为具有较低精度的 DECIMAL 数据类型，如以下示例所示。

```
select salesid, sum(pricepaid), median(sum(pricepaid)::decimal(30,2))
over() from sales where salesid < 10 group by salesid;
```

示例

以下示例计算每个卖家的平均销售数量：

```
select sellerid, qty, median(qty)
over (partition by sellerid)
from winsales
order by sellerid;


sellerid qty median
----------------------------
1  10 10.0
1  10 10.0
1  30 10.0
2  20 20.0
2  20 20.0
3  10 17.5
3  15 17.5
3  20 17.5
3  30 17.5
4  10 25.0
4  40 25.0
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

MIN 窗口函数

MIN 窗口函数返回最小输入表达式值。MIN 函数使用数值并忽略 NULL 值。

语法

```
MIN ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 MIN 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

接受任何数据类型作为输入。返回与 expression 相同的数据类型。

示例

以下示例从数据窗口的开头显示销售 ID、数量和最小数量：

```
select salesid, qty,
min(qty) over
(order by salesid rows unbounded preceding)
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  10
```

```
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  10
40001 |  40 |  10
40005 |  10 |  10
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例在受限制的框架中显示销售 ID、数量和最小数量：

```
select salesid, qty,
min(qty) over
(order by salesid rows between 2 preceding and 1 preceding) as min
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  20
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  15
40001 |  40 |  20
40005 |  10 |  30
(11 rows)
```

NTH_VALUE 窗口函数

NTH_VALUE 窗口函数返回相对于窗口的第一行的窗口框架的指定行的表达式值。

语法

```
NTH_VALUE (expr, offset)
[ IGNORE NULLS | RESPECT NULLS ]
OVER
( [ PARTITION BY window_partition ]
```

```
[ ORDER BY window_ordering
                          frame_clause ] )
```

参数

expr

对其执行函数的目标列或表达式。

offset

确定相对于要为其返回表达式的窗口中的第一行的行号。offset 可以是常数或表达式，且必须为大于 0 的正整数。

IGNORE NULLS

一个可选规范，该规范指示 Amazon Redshift 应跳过 null 值以确定要使用的行。如果未列出 IGNORE NULLS，则包含 Null 值。

RESPECT NULLS

指示 Amazon Redshift 应包含 null 值以确定要使用的行。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

指定窗口分区、排序和窗口框架。

PARTITION BY window_partition

设置 OVER 子句中每个组的记录范围。

ORDER BY window_ordering

对每个分区中的行进行排序。如果忽略 ORDER BY，则默认框架将包含分区中的所有行。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

NTH_VALUE 窗口函数支持使用任何 Amazon Redshift 数据类型的表达式。返回类型与 expr 的类型相同。

示例

以下示例显示了加利福利亚、佛罗里达和纽约的第三大场地的座位数与这些州的其他场地的座位数的比较情况：

```
select venuestate, venuename, venueseats,
nth_value(venueseats, 3)
ignore nulls
over(partition by venuestate order by venueseats desc
rows between unbounded preceding and unbounded following)
as third_most_seats
from (select * from venue where venueseats > 0 and
venuestate in('CA', 'FL', 'NY'))
order by venuestate;

venuestate |           venuename            | venueseats | third_most_seats
-----------+--------------------------------+------------+------------------
CA         | Qualcomm Stadium               |      70561 |            63026
CA         | Monster Park                   |      69843 |            63026
CA         | McAfee Coliseum                |      63026 |            63026
CA         | Dodger Stadium                 |      56000 |            63026
CA         | Angel Stadium of Anaheim       |      45050 |            63026
CA         | PETCO Park                     |      42445 |            63026
CA         | AT&T Park                      |      41503 |            63026
CA         | Shoreline Amphitheatre         |      22000 |            63026
FL         | Dolphin Stadium                |      74916 |            65647
FL         | Jacksonville Municipal Stadium |      73800 |            65647
FL         | Raymond James Stadium          |      65647 |            65647
FL         | Tropicana Field                |      36048 |            65647
NY         | Ralph Wilson Stadium           |      73967 |            20000
NY         | Yankee Stadium                 |      52325 |            20000
NY         | Madison Square Garden          |      20000 |            20000
(15 rows)
```

NTILE 窗口函数

NTILE 窗口函数将分区中已排序的行划分为大小尽可能相等的指定数量的已排名组，并返回给定行所在的组。

语法

```
NTILE (expr)
OVER (
```

```
[ PARTITION BY expression_list ]
[ ORDER BY order_list ]
)
```

参数

expr

排名组的数目，并且必须为每个分区生成一个正整数值（大于零）。expr 参数不得可为 null。

OVER

一个指定窗口分区和排序的子句。OVER 子句不能包含窗口框架规范。

PARTITION BY window_partition

可选。OVER 子句中每个组的记录范围。

ORDER BY window_ordering

可选。一个对每个分区中的行进行排序的表达式。如果忽略 ORDER BY 子句，则排名行为相同。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅 窗口函数的唯一数据排序。

返回类型

BIGINT

示例

以下示例将于 2008 年 8 月 26 日购买 Hamlet 门票所付价格划分到四个排名组中。结果集为 17 个行，几乎均匀地划分到排名 1 到 4 中：

```
select eventname, caldate, pricepaid, ntile(4)
over(order by pricepaid desc) from sales, event, date
where sales.eventid=event.eventid and event.dateid=date.dateid and eventname='Hamlet'
and caldate='2008-08-26'
order by 4;

eventname |  caldate   | pricepaid | ntile
----------+------------+-----------+-------
Hamlet    | 2008-08-26 |   1883.00 |     1
Hamlet    | 2008-08-26 |   1065.00 |     1
Hamlet    | 2008-08-26 |    589.00 |     1
```

```
Hamlet    | 2008-08-26 |    530.00 |     1
Hamlet    | 2008-08-26 |    472.00 |     1
Hamlet    | 2008-08-26 |    460.00 |     2
Hamlet    | 2008-08-26 |    355.00 |     2
Hamlet    | 2008-08-26 |    334.00 |     2
Hamlet    | 2008-08-26 |    296.00 |     2
Hamlet    | 2008-08-26 |    230.00 |     3
Hamlet    | 2008-08-26 |    216.00 |     3
Hamlet    | 2008-08-26 |    212.00 |     3
Hamlet    | 2008-08-26 |    106.00 |     3
Hamlet    | 2008-08-26 |    100.00 |     4
Hamlet    | 2008-08-26 |     94.00 |     4
Hamlet    | 2008-08-26 |     53.00 |     4
Hamlet    | 2008-08-26 |     25.00 |     4
(17 rows)
```

PERCENT_RANK 开窗函数

计算给定行的百分比排名。使用以下公式确定百分比排名：

```
(x - 1) / (the number of rows in the window or partition - 1)
```

其中，x 为当前行的排名。以下数据集说明了此公式的使用：

```
Row# Value Rank Calculation PERCENT_RANK
1 15 1 (1-1)/(7-1) 0.0000
2 20 2 (2-1)/(7-1) 0.1666
3 20 2 (2-1)/(7-1) 0.1666
4 20 2 (2-1)/(7-1) 0.1666
5 30 5 (5-1)/(7-1) 0.6666
6 30 5 (5-1)/(7-1) 0.6666
7 40 7 (7-1)/(7-1) 1.0000
```

返回值范围介于 0 和 1（含 1）之间。任何集合中的第一行的 PERCENT_RANK 均为 0。

语法

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

参数

( )

该函数没有参数，但需要空括号。

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口框架规范。

PARTITION BY partition_expression

可选。一个设置 OVER 子句中每个组的记录范围的表达式。

ORDER BY order_list

可选。用于计算百分比排名的表达式。该表达式必须具有数字数据类型或可隐式转换为 1。如果省略 ORDER BY，则所有行的返回值为 0。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅 窗口函数的唯一数据排序。

返回类型

FLOAT8

示例

以下示例计算每个卖家的销售数量的百分比排名：

```
select sellerid, qty, percent_rank()
over (partition by sellerid order by qty)
from winsales;

sellerid qty  percent_rank
----------------------------------------
1  10.00  0.0
1  10.64  0.5
1  30.37  1.0
3  10.04  0.0
3  15.15  0.33
3  20.75  0.67
3  30.55  1.0
2  20.09  0.0
```

```
2  20.12  1.0
4  10.12  0.0
4  40.23  1.0
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

PERCENTILE_CONT 开窗函数

PERCENTILE_CONT 是一种假定连续分布模型的逆分布函数。该函数具有一个百分比值和一个排序规范，并返回一个在有关排序规范的给定百分比值范围内的内插值。

PERCENTILE_CONT 在对值进行排序后计算值之间的线性内插。通过在聚合组中使用百分比值 (P) 和非 null 行数 (N)，该函数会在根据排序规范对行进行排序后计算行号。根据公式 (RN) 计算此行号 RN = (1+ (P*(N-1))。聚合函数的最终结果通过行号 CRN = CEILING(RN) 和 FRN = FLOOR(RN) 的行中的值之间的线性内插计算。

最终结果将如下所示。

如果 (CRN = FRN = RN)，则结果为 (value of expression from row at RN)

否则，结果将如下所示：

(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN).

您在 OVER 子句中只能指定 PARTITION 子句。如果为每个行指定 PARTITION，则 PERCENTILE_CONT 会返回位于给定分区内一组值中的指定百分比范围内的值。

PERCENTILE_CONT 是仅计算节点函数。如果查询不引用用户定义的表或 Amazon Redshift 系统表，该函数将返回错误。

语法

```
PERCENTILE_CONT ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

参数

percentile

介于 0 和 1 之间的数字常数。计算中将忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定用于排序和计算百分比的数字或日期/时间值。

OVER

指定窗口分区。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY expr

设置 OVER 子句中每个组的记录范围的可选参数。

返回值

返回类型由 WITHIN GROUP 子句中的 ORDER BY 表达式的数据类型决定。下表显示了每种个 ORDER BY 表达式数据类型的返回类型。

输入类型	返回类型
INT2、INT4、INT8、NUMERIC、DECIMAL	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP

使用说明

如果 ORDER BY 表达式是使用 38 位最大精度定义的 DECIMAL 数据类型，则 PERCENTILE_CONT 可能将返回不准确的结果或错误。如果 PERCENTILE_CONT 函数的返回值超过 38 位，则结果将截断以符合规范，这将导致精度降低。如果在插值期间，中间结果超出最大精度，则会发生数值溢出且函数会返回错误。要避免这些情况，建议使用具有较低精度的数据类型或将 ORDER BY 表达式转换为较低精度。

例如，带 DECIMAL 参数的 SUM 函数返回 38 位的默认精度。结果的小数位数与参数的小数位数相同。因此，例如，DECIMAL(5,2) 列的 SUM 返回 DECIMAL(38,2) 数据类型。

以下示例在 PERCENTILE_CONT 函数的 ORDER BY 子句中使用 SUM 函数。PRICEPAID 列的数据类型是 DECIMAL (8,2)，因此 SUM 函数返回 DECIMAL(38,2)。

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid) desc) over()
from sales where salesid < 10 group by salesid;
```

要避免潜在的精度降低或溢出错误，请将结果转换为具有较低精度的 DECIMAL 数据类型，如以下示例所示。

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid)::decimal(30,2) desc) over()
from sales where salesid < 10 group by salesid;
```

示例

以下示例使用 WINSALES 表。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over() as median from winsales;

 sellerid | qty | median
----------+-----+--------
        1 |  10 |   20.0
        1 |  10 |   20.0
        3 |  10 |   20.0
        4 |  10 |   20.0
        3 |  15 |   20.0
        2 |  20 |   20.0
        3 |  20 |   20.0
        2 |  20 |   20.0
        3 |  30 |   20.0
        1 |  30 |   20.0
        4 |  40 |   20.0
(11 rows)
```

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

 sellerid | qty | median
----------+-----+--------
        2 |  20 |   20.0
```

```
        2 |  20 |   20.0
        4 |  10 |   25.0
        4 |  40 |   25.0
        1 |  10 |   10.0
        1 |  10 |   10.0
        1 |  30 |   10.0
        3 |  10 |   17.5
        3 |  15 |   17.5
        3 |  20 |   17.5
        3 |  30 |   17.5
(11 rows)
```

以下示例计算华盛顿州的卖家的门票销售的 PERCENTILE_CONT 和 PERCENTILE_DISC。

```
SELECT sellerid, state, sum(qtysold*pricepaid) sales,
percentile_cont(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over(),
percentile_disc(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over()
from sales s, users u
where s.sellerid = u.userid and state = 'WA' and sellerid < 1000
group by sellerid, state;

 sellerid | state |  sales  | percentile_cont | percentile_disc
----------+-------+---------+-----------------+-----------------
      127 | WA    | 6076.00 |         2044.20 |         1531.00
      787 | WA    | 6035.00 |         2044.20 |         1531.00
      381 | WA    | 5881.00 |         2044.20 |         1531.00
      777 | WA    | 2814.00 |         2044.20 |         1531.00
       33 | WA    | 1531.00 |         2044.20 |         1531.00
      800 | WA    | 1476.00 |         2044.20 |         1531.00
        1 | WA    | 1177.00 |         2044.20 |         1531.00
(7 rows)
```

PERCENTILE_DISC 开窗函数

PERCENTILE_DISC 是一种假定离散分布模型的逆分布函数。该函数具有一个百分比值和一个排序规范，并返回给定集合中的元素。

对于给定的百分比值 P，PERCENTILE_DISC 在 ORDER BY 子句中对表达式的值进行排序，并返回带有大于或等于 P 的最小累积分布值（相对于同一排序规范）的值。

您在 OVER 子句中只能指定 PARTITION 子句。

PERCENTILE_DISC 是仅计算节点函数。如果查询不引用用户定义的表或 Amazon Redshift 系统表，该函数将返回错误。

语法

```
PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

参数

percentile

介于 0 和 1 之间的数字常数。计算中将忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定用于排序和计算百分比的数字或日期/时间值。

OVER

指定窗口分区。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY expr

设置 OVER 子句中每个组的记录范围的可选参数。

返回值

与 WITHIN GROUP 子句中的 ORDER BY 表达式相同的数据类型。

示例

以下各示例使用 WINSALES 表。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

```
SELECT sellerid, qty, PERCENTILE_DISC(0.5)
WITHIN GROUP (ORDER BY qty)
OVER() AS MEDIAN FROM winsales;

+----------+-----+--------+
| sellerid | qty | median |
+----------+-----+--------+
| 3        | 10  | 20     |
| 1        | 10  | 20     |
```

```
| 1         | 10  | 20     |
| 4         | 10  | 20     |
| 3         | 15  | 20     |
| 2         | 20  | 20     |
| 2         | 20  | 20     |
| 3         | 20  | 20     |
| 1         | 30  | 20     |
| 3         | 30  | 20     |
| 4         | 40  | 20     |
+----------+-----+--------+

SELECT sellerid, qty, PERCENTILE_DISC(0.5)
WITHIN GROUP (ORDER BY qty)
OVER(PARTITION BY sellerid) AS MEDIAN FROM winsales;

+----------+-----+--------+
| sellerid | qty | median |
+----------+-----+--------+
| 4         | 10  | 10     |
| 4         | 40  | 10     |
| 3         | 10  | 15     |
| 3         | 15  | 15     |
| 3         | 20  | 15     |
| 3         | 30  | 15     |
| 2         | 20  | 20     |
| 2         | 20  | 20     |
| 1         | 10  | 10     |
| 1         | 10  | 10     |
| 1         | 30  | 10     |
+----------+-----+--------+
```

要在按卖家 ID 分区时为数量找到 PERCENTILE_DISC(0.25) 和 PERCENTILE_DISC(0.75)，请使用以下示例。

```
SELECT sellerid, qty, PERCENTILE_DISC(0.25)
WITHIN GROUP (ORDER BY qty)
OVER(PARTITION BY sellerid) AS quartile1 FROM winsales;

+----------+-----+-----------+
| sellerid | qty | quartile1 |
+----------+-----+-----------+
| 4         | 10  | 10        |
| 4         | 40  | 10        |
```

```
| 2         | 20  | 20         |
| 2         | 20  | 20         |
| 3         | 10  | 10         |
| 3         | 15  | 10         |
| 3         | 20  | 10         |
| 3         | 30  | 10         |
| 1         | 10  | 10         |
| 1         | 10  | 10         |
| 1         | 30  | 10         |
+-----------+-----+------------+

SELECT sellerid, qty, PERCENTILE_DISC(0.75)
WITHIN GROUP (ORDER BY qty)
OVER(PARTITION BY sellerid) AS quartile3 FROM winsales;

+-----------+-----+------------+
| sellerid | qty | quartile3 |
+-----------+-----+------------+
| 3         | 10  | 20         |
| 3         | 15  | 20         |
| 3         | 20  | 20         |
| 3         | 30  | 20         |
| 4         | 10  | 40         |
| 4         | 40  | 40         |
| 2         | 20  | 20         |
| 2         | 20  | 20         |
| 1         | 10  | 30         |
| 1         | 10  | 30         |
| 1         | 30  | 30         |
+-----------+-----+------------+
```

RANK 窗口函数

RANK 窗口函数基于 OVER 子句中的 ORDER BY 表达式确定一组值中的一个值的排名。如果存在可选的 PARTITION BY 子句，则为每个行组重置排名。带符合排名标准的相同值的行接收相同的排名。Amazon Redshift 将关联行的数目添加到关联排名以计算下一个排名，因此排名可能不是连续数。例如，如果两个行的排名为 1，则下一个排名则为 3。

RANK 与 DENSE_RANK 窗口函数存在以下一点不同：对于 DENSE_RANK 来说，如果两个或两个以上的行结合，则一系列排名的值之间没有间隔。例如，如果两个行的排名为 1，则下一个排名则为 2。

您可以在同一查询中包含带有不同的 PARTITION BY 和 ORDER BY 子句的排名函数。

语法

```
RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

参数

( )

该函数没有参数，但需要空括号。

OVER

适用于 RANK 函数的窗口子句。

PARTITION BY expr_list

可选。一个或多个定义窗口的表达式。

ORDER BY order_list

可选。定义排名值基于的列。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。如果省略 ORDER BY，则所有行的返回值为 1。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅 窗口函数的唯一数据排序。

返回类型

INTEGER

示例

以下示例按销量对表进行排序（预设情况下按升序顺序），并为每个行分配一个排名。排名值 1 为排名最高的值。在应用窗口函数结果后，对结果进行排序：

```
select salesid, qty,
rank() over (order by qty) as rnk
from winsales
```

```
order by 2,1;

salesid | qty | rnk
--------+-----+-----
10001 |  10 |  1
10006 |  10 |  1
30001 |  10 |  1
40005 |  10 |  1
30003 |  15 |  5
20001 |  20 |  6
20002 |  20 |  6
30004 |  20 |  6
10005 |  30 |  9
30007 |  30 |  9
40001 |  40 |  11
(11 rows)
```

请注意，本示例中的外部 ORDER BY 子句包括列 2 和列 1，以确保在每次运行查询时，Amazon Redshift 返回一致排序的结果。例如，销售 ID 为 10001 和 10006 的行具有相同的 QTY 和 RNK 值。按列 1 对最后的结果集进行排序可确保行 10001 始终在 10006 之前。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

在下面的示例中，将窗口函数的顺序倒转 (`order by qty desc`)。现在，最高排名值将应用于最大的 QTY 值。

```
select salesid, qty,
rank() over (order by qty desc) as rank
from winsales
order by 2,1;

 salesid | qty | rank
---------+-----+-----
   10001 |  10 |   8
   10006 |  10 |   8
   30001 |  10 |   8
   40005 |  10 |   8
   30003 |  15 |   7
   20001 |  20 |   4
   20002 |  20 |   4
   30004 |  20 |   4
   10005 |  30 |   2
   30007 |  30 |   2
   40001 |  40 |   1
```

```
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例按 SELLERID 对表进行分区，按数量对每个分区进行排序（按降序顺序），并为每个行分配排名。在应用窗口函数结果后，对结果进行排序。

```
select salesid, sellerid, qty, rank() over
(partition by sellerid
order by qty desc) as rank
from winsales
order by 2,3,1;

salesid | sellerid | qty | rank
--------+----------+-----+-----
  10001 |        1 |  10 |  2
  10006 |        1 |  10 |  2
  10005 |        1 |  30 |  1
  20001 |        2 |  20 |  1
  20002 |        2 |  20 |  1
  30001 |        3 |  10 |  4
  30003 |        3 |  15 |  3
  30004 |        3 |  20 |  2
  30007 |        3 |  30 |  1
  40005 |        4 |  10 |  2
  40001 |        4 |  40 |  1
(11 rows)
```

RATIO_TO_REPORT 开窗函数

计算某个窗口或分区中的某个值与所有值的和的比率。使用以下公式确定报表值的比率：

`value of` ratio_expression `argument for the current row / sum of` ratio_expression `argument for the window or partition`

以下数据集说明了此公式的使用：

```
Row# Value Calculation RATIO_TO_REPORT
1 2500 (2500)/(13900) 0.1798
2 2600 (2600)/(13900) 0.1870
3 2800 (2800)/(13900) 0.2014
4 2900 (2900)/(13900) 0.2086
```

```
5 3100 (3100)/(13900) 0.2230
```

返回值范围介于 0 和 1（含 1）之间。如果 ratio_expression 为 NULL，则返回值为 NULL。如果 partition_expression 中的值是唯一的，则函数将为该值返回 1。

语法

```
RATIO_TO_REPORT ( ratio_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

参数

ratio_expression

一个提供要为其确定比率的值的表达式（例如列名）。该表达式必须具有数字数据类型或可隐式转换为 1。

您无法在 ratio_expression 中使用任何其他分析函数。

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY partition_expression

可选。一个设置 OVER 子句中每个组的记录范围的表达式。

返回类型

FLOAT8

示例

以下各示例使用 WINSALES 表。有关如何创建 WINSALES 表的信息，请参阅窗口函数示例的示例表。

以下示例计算每行卖家数量占所有卖家总数量的比率值。

```
select sellerid, qty, ratio_to_report(qty)
over()
from winsales
order by sellerid;

sellerid  qty    ratio_to_report
```

```
---------------------------------------
1          30     0.13953488372093023
1          10     0.046511627906976744
1          10     0.046511627906976744
2          20     0.09302325581395349
2          20     0.09302325581395349
3          30     0.13953488372093023
3          20     0.09302325581395349
3          15     0.06976744186046512
3          10     0.046511627906976744
4          10     0.046511627906976744
4          40     0.18604651162790697
```

以下示例按分区计算每个卖家的销售数量的比率。

```
select sellerid, qty, ratio_to_report(qty)
over(partition by sellerid)
from winsales;

sellerid   qty    ratio_to_report
-----------------------------------------
2          20     0.5
2          20     0.5
4          40     0.8
4          10     0.2
1          10     0.2
1          30     0.6
1          10     0.2
3          10     0.13333333333333333
3          15     0.2
3          20     0.26666666666666666
3          30     0.4
```

ROW_NUMBER 窗口函数

根据 OVER 子句中的 ORDER BY 表达式，分配一组行中当前行的序号（从 1 开始计数）。如果存在可选的 PARTITION BY 子句，则为每组行重置序号。ORDER BY 表达式中具有相同值的行以非确定性的方式接收不同的行号。

语法

```
ROW_NUMBER() OVER(
```

```
  [ PARTITION BY expr_list ]
  [ ORDER BY order_list ]
)
```

参数

( )

该函数没有参数，但需要空括号。

OVER

适用于 ROW_NUMBER 函数的窗口函数子句。

PARTITION BY expr_list

可选。一个或多个将结果划分成行集的列表达式。

ORDER BY order_list

可选。一个或多个定义集合内的行顺序的列表达式。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

如果 ORDER BY 未生成唯一顺序或被省略，则行的顺序是不确定的。有关更多信息，请参阅 窗口函数的唯一数据排序。

返回类型

BIGINT

示例

下面的示例使用 WINSALES 表。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例按 QTY 对表进行排序（按升序），然后为每个行分配一个行号。在应用窗口函数结果后，对结果进行排序。

```
SELECT salesid, sellerid, qty,
ROW_NUMBER() OVER(
   ORDER BY qty ASC) AS row
FROM winsales
ORDER BY 4,1;
```

```
salesid   sellerid   qty   row
---------+----------+-----+-----
   30001 |        3 |  10 |   1
   10001 |        1 |  10 |   2
   10006 |        1 |  10 |   3
   40005 |        4 |  10 |   4
   30003 |        3 |  15 |   5
   20001 |        2 |  20 |   6
   20002 |        2 |  20 |   7
   30004 |        3 |  20 |   8
   10005 |        1 |  30 |   9
   30007 |        3 |  30 |  10
   40001 |        4 |  40 |  11
```

以下示例按 SELLERID 对表进行分区并按 QTY 对每个分区进行排序（按升序顺序），然后为每个行分配一个行号。在应用窗口函数结果后，对结果进行排序。

```
SELECT salesid, sellerid, qty,
ROW_NUMBER() OVER(
  PARTITION BY sellerid
  ORDER BY qty ASC) AS row_by_seller
FROM winsales
ORDER BY 2,4;

 salesid | sellerid | qty | row_by_seller
---------+----------+-----+-----
   10001 |        1 |  10 |   1
   10006 |        1 |  10 |   2
   10005 |        1 |  30 |   3
   20001 |        2 |  20 |   1
   20002 |        2 |  20 |   2
   30001 |        3 |  10 |   1
   30003 |        3 |  15 |   2
   30004 |        3 |  20 |   3
   30007 |        3 |  30 |   4
   40005 |        4 |  10 |   1
   40001 |        4 |  40 |   2
```

以下示例显示了不使用可选子句时的结果。

```
SELECT salesid, sellerid, qty, ROW_NUMBER() OVER() AS row
FROM winsales
```

```
ORDER BY 4,1;

salesid   sellerid   qty   row
---------+----------+-----+-----
   30001 |        3 |  10 |   1
   10001 |        1 |  10 |   2
   10005 |        1 |  30 |   3
   40001 |        4 |  40 |   4
   10006 |        1 |  10 |   5
   20001 |        2 |  20 |   6
   40005 |        4 |  10 |   7
   20002 |        2 |  20 |   8
   30003 |        3 |  15 |   9
   30004 |        3 |  20 |  10
   30007 |        3 |  30 |  11
```

STDDEV_SAMP 和 STDDEV_POP 窗口函数

STDDEV_SAMP 和 STDDEV_POP 窗口函数返回一组数值（整数、小数或浮点）的样本标准差和总体标准差。另请参阅 STDDEV_SAMP 和 STDDEV_POP 函数。

STDDEV_SAMP 和 STDDEV 是同一函数的同义词。

语法

```
STDDEV_SAMP | STDDEV | STDDEV_POP
( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

STDDEV 函数支持的参数类型包括 SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 和 DOUBLE PRECISION。

无论表达式的数据类型如何，STDDEV 函数的返回类型都是双精度数。

示例

以下示例说明如何使用 STDDEV_POP 和 VAR_POP 作为窗口函数。查询计算 SALES 表中 PRICEPAID 值的总体方差和总体标准差。

```
select salesid, dateid, pricepaid,
round(stddev_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as stddevpop,
round(var_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as varpop
from sales
order by 2,1;

salesid | dateid | pricepaid | stddevpop | varpop
--------+--------+-----------+-----------+---------
  33095 |   1827 |    234.00 |         0 |       0
  65082 |   1827 |    472.00 |       119 |   14161
  88268 |   1827 |    836.00 |       248 |   61283
```

```
  97197 |   1827 |    708.00 |       230 |   53019
 110328 |   1827 |    347.00 |       223 |   49845
 110917 |   1827 |    337.00 |       215 |   46159
 150314 |   1827 |    688.00 |       211 |   44414
 157751 |   1827 |   1730.00 |       447 |  199679
 165890 |   1827 |   4192.00 |      1185 | 1403323
...
```

样本标准差和样本标准方差函数可通过同一方式使用。

SUM 窗口函数

SUM 窗口函数返回输入列值或表达式值的和。SUM 函数使用数值并忽略 NULL 值。

语法

```
SUM ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 SUM 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

SUM 函数支持的参数类型为 SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 和 DOUBLE PRECISION。

SUM 函数支持的返回类型为：

- 适用于 SMALLINT 或 INTEGER 参数的 BIGINT
- 适用于 BIGINT 参数的 NUMERIC
- 适用于浮点参数的 DOUBLE PRECISION

示例

以下示例创建按日期和销售 ID 排序的销售数量的累积（移动）和：

```
select salesid, dateid, sellerid, qty,
sum(qty) over (order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  20
10005 | 2003-12-24 |        1 |  30 |  50
40001 | 2004-01-09 |        4 |  40 |  90
10006 | 2004-01-18 |        1 |  10 | 100
20001 | 2004-02-12 |        2 |  20 | 120
40005 | 2004-02-12 |        4 |  10 | 130
20002 | 2004-02-16 |        2 |  20 | 150
30003 | 2004-04-18 |        3 |  15 | 165
30004 | 2004-04-18 |        3 |  20 | 185
30007 | 2004-09-07 |        3 |  30 | 215
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例按日期创建销售数量的累积（移动）和，按卖家 ID 对结果进行分区，并按日期和销售 ID 对该分区中的结果进行排序：

```
select salesid, dateid, sellerid, qty,
sum(qty) over (partition by sellerid
order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
10005 | 2003-12-24 |        1 |  30 |  40
40001 | 2004-01-09 |        4 |  40 |  40
10006 | 2004-01-18 |        1 |  10 |  50
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  50
20002 | 2004-02-16 |        2 |  20 |  40
30003 | 2004-04-18 |        3 |  15 |  25
30004 | 2004-04-18 |        3 |  20 |  45
30007 | 2004-09-07 |        3 |  30 |  75
(11 rows)
```

对结果集中的所有行进行编号（按 SELLERID 和 SALESID 列进行排序）：

```
select salesid, sellerid, qty,
sum(1) over (order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid |  qty | rownum
--------+----------+------+--------
10001 |        1 |   10 |      1
10005 |        1 |   30 |      2
10006 |        1 |   10 |      3
20001 |        2 |   20 |      4
20002 |        2 |   20 |      5
30001 |        3 |   10 |      6
30003 |        3 |   15 |      7
30004 |        3 |   20 |      8
```

```
30007 |        3 |   30 |     9
40001 |        4 |   40 |    10
40005 |        4 |   10 |    11
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例对结果集中的所有行按顺序进行编号，按 SELLERID 对结果进行分区，并按 SELLERID 和 SALESID 对该分区中的结果进行排序：

```
select salesid, sellerid, qty,
sum(1) over (partition by sellerid
order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid | qty | rownum
---------+----------+-----+--------
10001 |        1 |  10 |      1
10005 |        1 |  30 |      2
10006 |        1 |  10 |      3
20001 |        2 |  20 |      1
20002 |        2 |  20 |      2
30001 |        3 |  10 |      1
30003 |        3 |  15 |      2
30004 |        3 |  20 |      3
30007 |        3 |  30 |      4
40001 |        4 |  40 |      1
40005 |        4 |  10 |      2
(11 rows)
```

VAR_SAMP 和 VAR_POP 窗口函数

VAR_SAMP 和 VAR_POP 窗口函数返回一组数值（整数、小数或浮点）的样本方差和总体方差。另请参阅 VAR_SAMP 和 VAR_POP 函数。

VAR_SAMP 和 VARIANCE 是同一函数的同义词。

语法

```
VAR_SAMP | VARIANCE | VAR_POP
( [ ALL ] expression ) OVER
```

```
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

VARIANCE 函数支持的参数类型包括
SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 和 DOUBLE PRECISION。

无论表达式的数据类型如何，VARIANCE 函数的返回类型都是双精度数。

系统管理函数

主题

- CHANGE_QUERY_PRIORITY

- CHANGE_SESSION_PRIORITY
- CHANGE_USER_PRIORITY
- CURRENT_SETTING
- PG_CANCEL_BACKEND
- PG_TERMINATE_BACKEND
- REBOOT_CLUSTER
- SET_CONFIG

Amazon Redshift 支持若干系统管理函数。

CHANGE_QUERY_PRIORITY

CHANGE_QUERY_PRIORITY 使超级用户能够修改在工作负载管理 (WLM) 中运行或等待的查询的优先级。

此功能使超级用户可以立即更改系统中任何查询的优先级。只有一个查询、用户或会话可以使用优先级 CRITICAL 运行。

语法

```
CHANGE_QUERY_PRIORITY(query_id, priority)
```

参数

query_id

更改其优先级的查询的查询标识符。需要 INTEGER 值。

priority

要分配给查询的新优先级。此参数必须是包含以下值的字符串：CRITICAL、HIGHEST、HIGH、NORMAL、LOW 或 LOWEST。

返回类型

无

示例

要显示 STV_WLM_QUERY_STATE 系统表中的列 query_priority，请使用以下示例。

```
SELECT query, service_class, query_priority, state
FROM stv_wlm_query_state WHERE service_class = 101;

+-------+---------------+----------------+---------+
| query | service_class | query_priority |  state  |
+-------+---------------+----------------+---------+
|  1076 |           101 | Lowest         | Running |
|  1075 |           101 | Lowest         | Running |
+-------+---------------+----------------+---------+
```

要显示运行函数 change_query_priority 以将优先级更改为 CRITICAL 的超级用户的结果，请使用以下示例。

```
SELECT CHANGE_QUERY_PRIORITY(1076, 'Critical');

+-------------------------------------------------------------------------------+
|                           change_query_priority                               |
+-------------------------------------------------------------------------------+
| Succeeded to change query priority. Priority changed from Lowest to Critical. |
+-------------------------------------------------------------------------------+
```

CHANGE_SESSION_PRIORITY

CHANGE_SESSION_PRIORITY 使超级用户可以立即更改系统中任何会话的优先级。只有一个会话、用户或查询可以使用优先级 CRITICAL 运行。

语法

```
CHANGE_SESSION_PRIORITY(pid, priority)
```

参数

pid

更改其优先级的会话的进程标识符。值 -1 指的是当前会话。需要 INTEGER 值。

priority

要分配给会话的新优先级。此参数必须是包含以下值的字符串：CRITICAL、HIGHEST、HIGH、NORMAL、LOW 或 LOWEST。

返回类型

无

示例

要返回处理当前会话的服务器进程的进程标识符，请使用以下示例。

```
SELECT pg_backend_pid();

+----------------+
| pg_backend_pid |
+----------------+
|          30311 |
+----------------+
```

在此示例中，将当前会话的优先级更改为 LOWEST。

```
SELECT CHANGE_SESSION_PRIORITY(30311, 'Lowest');

+-----------------------------------------------------------------------------------------
+
|                                change_session_priority
 |
+-----------------------------------------------------------------------------------------
+
| Succeeded to change session priority. Changed session (pid:30311) priority to lowest.
 |
+-----------------------------------------------------------------------------------------
+
```

在此示例中，将当前会话的优先级更改为 HIGH。

```
SELECT CHANGE_SESSION_PRIORITY(-1, 'High');

+-----------------------------------------------------------------------------------------
+
|                                    change_session_priority
           |
+-----------------------------------------------------------------------------------------
+
| Succeeded to change session priority. Changed session (pid:30311) priority from
 lowest to high. |
```

```
+-----------------------------------------------------------------------------------------------
+
```

要创建用于更改会话优先级的存储过程，请使用以下示例。运行此存储过程的权限授予数据库用户 test_user。

```
CREATE OR REPLACE PROCEDURE sp_priority_low(pid IN int, result OUT varchar)
AS $$
BEGIN
  SELECT CHANGE_SESSION_PRIORITY(pid, 'low') into result;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
GRANT EXECUTE ON PROCEDURE sp_priority_low(int) TO test_user;
```

然后，名为 test_user 的数据库用户调用该过程。

```
CALL sp_priority_low(pg_backend_pid());

+-----------------------------------------------------+
|                        result                       |
+-----------------------------------------------------+
| Success. Change session (pid:13155) priority to low. |
+-----------------------------------------------------+
```

CHANGE_USER_PRIORITY

CHANGE_USER_PRIORITY 使超级用户能够修改由在工作负载管理 (WLM) 中运行或等待的用户发出的所有查询的优先级。只有一个用户、会话或查询可以使用优先级 CRITICAL 运行。

语法

```
CHANGE_USER_PRIORITY(user_name, priority)
```

参数

user_name

查询优先级已更改的数据库用户名。

priority

user_name 要分配给所有查询的新优先级。此参数必须是包含以下值的字符串：CRITICAL、HIGHEST、HIGH、NORMAL、LOW、LOWEST 或 RESET。只有超级用户才能将优先级更改为 CRITICAL。将优先级更改为 RESET 会删除 user_name 的优先级设置。

返回类型

无

示例

要将用户 analysis_user 的优先级更改为 LOWEST，请使用以下示例。

```
SELECT CHANGE_USER_PRIORITY('analysis_user', 'lowest');

+-----------------------------------------------------------------------------------------+
|                                change_user_priority                                     |
+-----------------------------------------------------------------------------------------+
| Succeeded to change user priority. Changed user (analysis_user) priority to lowest. |
+-----------------------------------------------------------------------------------------+
```

要将优先级更改为 LOW，请使用以下示例。

```
SELECT CHANGE_USER_PRIORITY('analysis_user', 'low');

+-----------------------------------------------------------------------------------------------------------
+
|                                    change_user_priority
        |
+-----------------------------------------------------------------------------------------------------------
+
| Succeeded to change user priority. Changed user (analysis_user) priority from Lowest
 to low. |
+-----------------------------------------------------------------------------------------------------------
+
```

要重置优先级，请使用以下示例。

```
SELECT CHANGE_USER_PRIORITY('analysis_user', 'reset');
```

```
+-----------------------------------------------------+
|                 change_user_priority                |
+-----------------------------------------------------+
| Succeeded to reset priority for user (analysis_user). |
+-----------------------------------------------------+
```

CURRENT_SETTING

CURRENT_SETTING 返回指定配置参数的当前值。

此函数等效于 SHOW 命令。

语法

```
current_setting('parameter')
```

以下语句返回指定会话上下文变量的当前值。

```
current_setting('variable_name')
current_setting('variable_name'[, error_if_undefined])
```

参数

parameter

要显示的参数值。有关配置参数的列表，请参阅配置参考

variable_name

要显示的变量的名称。对于会话上下文变量，它必须是字符串常量。

error_if_undefined

（可选）一个布尔值，该值指定变量名不存在时的行为。当 error_if_undefined 设置为 TRUE（默认值）时，Amazon Redshift 会引发错误。当 error_if_undefined 设置为 FALSE 时，Amazon Redshift 会返回 NULL。Amazon Redshift 仅支持将 error_if_undefined 参数用于会话上下文变量。当输入为配置参数时，不能使用此项。

返回类型

返回 CHAR 或 VARCHAR 字符串。

示例

要返回 query_group 参数的当前设置，请使用以下示例。

```
SELECT CURRENT_SETTING('query_group');

+-----------------+
| current_setting |
+-----------------+
| unset           |
+-----------------+
```

要返回变量 app_context.user_id 的当前设置，请使用以下示例。

```
SELECT CURRENT_SETTING('app_context.user_id', FALSE);
```

PG_CANCEL_BACKEND

取消查询。PG_CANCEL_BACKEND 的功能等效于 CANCEL 命令。您可取消当前由您的用户运行的查询。超级用户可取消任何查询。

语法

```
pg_cancel_backend( pid )
```

参数

pid

要取消的查询的进程 ID (PID)。您不能通过指定查询 ID 来取消查询；您必须指定查询的进程 ID。需要 INTEGER 值。

返回类型

无

使用说明

如果多个会话中的查询在同一个表上保持锁定状态，您可以使用 PG_TERMINATE_BACKEND 函数终止其中一个会话，这将强制所终止会话中所有当前运行的事务释放锁定并回滚事务。查询

PG__LOCKS 目录表以查看当前持有的锁。如果某个查询由于位于事务块 (BEGIN … END) 中而无法取消，您可使用 PG_TERMINATE_BACKEND 函数终止在其中运行该查询的会话。

示例

要取消当前正在运行的查询，请先检索要取消的查询的进程 ID。要确定所有当前正在运行的查询的进程 ID，请运行以下命令。

```
SELECT pid, TRIM(starttime) AS start,
duration, TRIM(user_name) AS user,
SUBSTRING(query,1,40) AS querytxt
FROM stv_recents
WHERE status = 'Running';

+-----+-----------------------+----------+--------+-----------------------------+
| pid |       starttime       | duration |  user  |          querytxt           |
+-----+-----------------------+----------+--------+-----------------------------+
| 802 | 2013-10-14 09:19:03.55 |      132 | dwuser | select venuename from venue |
| 834 | 2013-10-14 08:33:49.47 |  1250414 | dwuser | select * from listing;      |
| 964 | 2013-10-14 08:30:43.29 |   326179 | dwuser | select sellerid from sales  |
+-----+-----------------------+----------+--------+-----------------------------+
```

要取消进程 ID 为 802 的查询，请使用以下示例。

```
SELECT PG_CANCEL_BACKEND(802);
```

PG_TERMINATE_BACKEND

终止会话。您可终止您的用户拥有的会话。超级用户可终止任何会话。

语法

```
pg_terminate_backend( pid )
```

参数

pid

要终止的会话的进程 ID。需要 INTEGER 值。

返回类型

无

使用说明

如果您即将达到并行连接的限制，请使用 PG_TERMINATE_BACKEND 终止空闲会话并释放连接。有关更多信息，请参阅 Amazon Redshift 限制。

如果多个会话中的查询锁定到了同一个表，您可以使用 PG_TERMINATE_BACKEND 终止其中一个会话，这将强制已终止会话中所有当前运行的事务释放锁定并回滚事务。查询 PG_LOCKS 目录表以查看当前持有的锁。

如果某个查询不在事务块 (BEGIN … END) 中，您可使用 CANCEL 命令或 PG_CANCEL_BACKEND 函数取消该查询。

示例

要查询 SVV_TRANSACTIONS 表以查看对当前事务生效的所有锁，请使用以下示例。

```
SELECT * FROM svv_transactions;

+-----------+--------+-------+------+---------------------+-----------------
+----------------------+----------+---------+
| txn_owner | txn_db |  xid  | pid  |      txn_start      |    lock_mode    |
 lockable_object_type | relation | granted |
+-----------+--------+-------+------+---------------------+-----------------
+----------------------+----------+---------+
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
           |    51940 | true    |
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
           |    52000 | true    |
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
           |   108623 | true    |
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | ExclusiveLock   |
 transactionid        |          | true    |
+-----------+--------+-------+------+---------------------+-----------------
+----------------------+----------+---------+
```

要终止持有锁的会话，请使用以下示例。

```
SELECT PG_TERMINATE_BACKEND(8585);
```

REBOOT_CLUSTER

在不关闭与集群的连接的情况下，重新启动 Amazon Redshift 集群。您必须是数据库超级用户才能执行此命令。

软重启完成后，Amazon Redshift 集群会向用户应用程序返回一个错误，并要求用户应用程序重新提交因软重启而中断的任何事务或查询。

语法

```
SELECT REBOOT_CLUSTER();
```

SET_CONFIG

将配置参数设为新的设置。

此函数等效于 SQL 中的 SET 命令。

语法

```
SET_CONFIG('parameter', 'new_value' , is_local)
```

以下语句将会话上下文变量设置为新设置。

```
set_config('variable_name', 'new_value' , is_local)
```

参数

parameter

要设置的参数。

variable_name

要设置的变量的名称。

new_value

参数的新值。

is_local

如果为 true，则参数值仅适用于当前事务。有效值为 `true` 或 `1` 以及 `false` 或 `0`。

返回类型

返回 CHAR 或 VARCHAR 字符串。

示例

要仅针对当前事务将 query_group 参数的值设置为 test，请使用以下示例。

```
SELECT SET_CONFIG('query_group', 'test', true);

+------------+
| set_config |
+------------+
| test       |
+------------+
```

要设置会话上下文变量，请使用以下示例。

```
SELECT SET_CONFIG('app.username', 'cuddy', FALSE);
```

系统信息函数

Amazon Redshift 支持很多系统信息函数。

主题

- CURRENT_AWS_ACCOUNT
- CURRENT_DATABASE
- CURRENT_NAMESPACE
- CURRENT_SCHEMA
- CURRENT_SCHEMAS
- CURRENT_USER
- CURRENT_USER_ID
- DEFAULT_IAM_ROLE
- HAS_ASSUMEROLE_PRIVILEGE
- HAS_DATABASE_PRIVILEGE
- HAS_SCHEMA_PRIVILEGE
- HAS_TABLE_PRIVILEGE

- LAST_USER_QUERY_ID
- PG_BACKEND_PID
- PG_GET_COLS
- PG_GET_GRANTEE_BY_IAM_ROLE
- PG_GET_IAM_ROLE_BY_USER
- PG_GET_LATE_BINDING_VIEW_COLS
- PG_GET_SESSION_ROLES
- PG_LAST_COPY_COUNT
- PG_LAST_COPY_ID
- PG_LAST_UNLOAD_ID
- PG_LAST_QUERY_ID
- PG_LAST_UNLOAD_COUNT
- SLICE_NUM 函数
- USER
- ROLE_IS_MEMBER_OF
- USER_IS_MEMBER_OF
- VERSION

CURRENT_AWS_ACCOUNT

返回与提交查询的 Amazon Redshift 集群关联的 AWS 账户。

语法

```
current_aws_account
```

返回类型

返回整数。

示例

以下查询返回当前数据库的名称。

```
select user, current_aws_account;
```

```
current_user | current_account
-------------+---------------
dwuser       | 987654321

(1 row)
```

CURRENT_DATABASE

返回您当前连接到的数据库的名称。

语法

```
current_database()
```

返回类型

返回 CHAR 或 VARCHAR 字符串。

示例

以下查询返回当前数据库的名称。

```
select current_database();

current_database
------------------
tickit
(1 row)
```

CURRENT_NAMESPACE

返回当前 Amazon Redshift 集群的集群命名空间。Amazon Redshift 集群命名空间是 Amazon Redshift 集群的唯一 ID。

语法

```
current_namespace
```

返回类型

返回 CHAR 或 VARCHAR 字符串。

示例

以下查询返回当前命名空间的名称。

```
select user, current_namespace;
current_user | current_namespace
-------------+--------------------------------------
dwuser       | 86b5169f-01dc-4a6f-9fbb-e2e24359e9a8

(1 row)
```

CURRENT_SCHEMA

返回位于搜索路径前部的 schema 的名称。此 schema 将用于在创建时未指定目标 schema 的任何表或其他命名对象。

语法

> **Note**
>
> 这是领导节点函数。如果此函数引用了用户创建的表、STL/STV 系统表或 SVV/SVL 系统视图，它将返回错误。

```
current_schema()
```

返回类型

CURRENT_SCHEMA 返回 CHAR 或 VARCHAR 字符串。

示例

以下查询将返回当前 schema：

```
select current_schema();

current_schema
---------------
public
(1 row)
```

CURRENT_SCHEMAS

返回当前搜索路径中任何 schemas 的名称的数组。当前搜索路径是在 search_path 参数中定义的。

语法

> **Note**
>
> 这是领导节点函数。如果此函数引用了用户创建的表、STL/STV 系统表或 SVV/SVL 系统视图，它将返回错误。

```
current_schemas(include_implicit)
```

参数

include_implicit

如果为 True，则指定搜索路径应包含任何隐式包含的系统 schemas。有效值为 `true` 和 `false`。通常，如果为 `true`，此参数将返回 `pg_catalog` schema 以及当前 schema。

返回类型

返回 CHAR 或 VARCHAR 字符串。

示例

以下示例返回当前搜索路径中的 schemas（不包括隐式包含的系统 schemas）的名称：

```
select current_schemas(false);

current_schemas
-----------------
{public}
(1 row)
```

以下示例返回当前搜索路径中的 schemas（包括隐式包含的系统 schemas）的名称：

```
select current_schemas(true);

current_schemas
```

```
---------------------
{pg_catalog,public}
(1 row)
```

CURRENT_USER

返回数据库的当前“有效”用户的用户名，视检查权限而定。通常，此用户名将与会话用户的相同；但是，此用户名偶尔可能被超级用户更改。

> **Note**
>
> 请勿在调用 CURRENT_USER 时使用尾随圆括号。

语法

```
current_user
```

返回类型

CURRENT_USER 返回 NAME 数据类型，可以将其转换为 CHAR 或 VARCHAR 字符串。

使用说明

如果存储过程是使用 CREATE_PROCEDURE 命令的 SECURITY DEFINER 选项创建的，则从存储过程内调用 CURRENT_USER 函数时，Amazon Redshift 会返回存储过程拥有者的用户名。

示例

以下查询返回当前数据库用户的名称：

```
select current_user;

current_user
--------------
dwuser
(1 row)
```

CURRENT_USER_ID

返回登录到当前会话的 Amazon Redshift 用户的唯一标识符。

语法

```
CURRENT_USER_ID
```

返回类型

CURRENT_USER_ID 函数返回整数。

示例

以下示例返回此会话的用户名和当前用户 ID：

```
select user, current_user_id;

 current_user | current_user_id
--------------+-----------------
   dwuser     |               1
(1 row)
```

DEFAULT_IAM_ROLE

返回当前与 Amazon Redshift 集群关联的默认 IAM 角色。如果没有任何关联的默认 IAM 角色，该函数将不返回任何内容。

语法

```
select default_iam_role();
```

返回类型

返回 VARCHAR 字符串。

示例

以下示例返回了当前与指定的 Amazon Redshift 集群关联的默认 IAM 角色，

```
select default_iam_role();
              default_iam_role
---------------------------------------------
 arn:aws:iam::123456789012:role/myRedshiftRole
(1 row)
```

HAS_ASSUMEROLE_PRIVILEGE

如果指定用户具有指定 IAM 角色并具有运行指定命令的权限，则返回 Boolean `true`（`t`）。如果用户不具有指定 IAM 角色但具有运行指定命令的权限，函数将返回 `false`（`f`）。有关权限的更多信息，请参阅 GRANT。

语法

```
has_assumerole_privilege( [ user, ] iam_role_arn, cmd_type)
```

参数

用户

要接受 IAM 角色权限检查的用户名称。默认为检查当前用户。超级用户和用户可以使用此函数。但是，用户只能查看自己的权限。

iam_role_arn

已被授予命令权限的 IAM 角色。

cmd_type

已授予访问权限的命令。有效值如下所示：

- COPY
- UNLOAD
- EXTERNAL FUNCTION
- CREATE MODEL

返回类型

BOOLEAN

示例

以下查询确认用户 `reg_user1` 具有运行 COPY 命令的 `Redshift-S3-Read` 角色的权限。

```
select has_assumerole_privilege('reg_user1', 'arn:aws:iam::123456789012:role/Redshift-
S3-Read', 'copy');
```

```
has_assumerole_privilege
------------------------
true
(1 row)
```

HAS_DATABASE_PRIVILEGE

如果用户对指定数据库具有指定特权，则返回 `true`。有关权限的更多信息，请参阅 GRANT。

语法

> **Note**
>
> 这是领导节点函数。如果此函数引用了用户创建的表、STL/STV 系统表或 SVV/SVL 系统视图，它将返回错误。

```
has_database_privilege( [ user, ] database, privilege)
```

参数

用户

要接受数据库权限检查的用户的名称。默认为检查当前用户。

数据库

与特权关联的数据库。

privilege

要检查的特权。有效值如下所示：

- CREATE
- TEMPORARY
- TEMP

返回类型

返回 CHAR 或 VARCHAR 字符串。

示例

以下查询确认 GUEST 用户对 TICKIT 数据库具有 TEMP 特权。

```
select has_database_privilege('guest', 'tickit', 'temp');

has_database_privilege
-----------------------
true
(1 row)
```

HAS_SCHEMA_PRIVILEGE

如果用户对指定 schema 具有指定特权，则返回 `true`。有关权限的更多信息，请参阅 GRANT。

语法

> **Note**
>
> 这是领导节点函数。如果此函数引用了用户创建的表、STL/STV 系统表或 SVV/SVL 系统视图，它将返回错误。

```
has_schema_privilege( [ user, ] schema, privilege)
```

参数

用户

要接受 schema 特权检查的用户的名称。默认为检查当前用户。

schema

与特权关联的 Schema。

privilege

要检查的特权。有效值如下所示：

- CREATE
- USAGE

返回类型

返回 CHAR 或 VARCHAR 字符串。

示例

以下查询确认 GUEST 用户对 PUBLIC schema 具有 CREATE 特权：

```
select has_schema_privilege('guest', 'public', 'create');

has_schema_privilege
---------------------
true
(1 row)
```

HAS_TABLE_PRIVILEGE

如果用户对指定的表具有指定特权，则返回 `true`，否则返回 `false`。

语法

> **Note**
>
> 这是领导节点函数。如果此函数引用了用户创建的表、STL/STV 系统表或 SVV/SVL 系统视图，它将返回错误。有关权限的更多信息，请参阅 GRANT。

```
has_table_privilege( [ user, ] table, privilege)
```

参数

用户

要接受表特权检查的用户的名称。默认为检查当前用户。

table

与特权关联的表。

privilege

要检查的特权。有效值如下所示：

- SELECT
- INSERT
- UPDATE
- 删除
- DROP
- REFERENCES

返回类型

BOOLEAN

示例

以下查询发现 GUEST 用户对 LISTING 表没有 SELECT 特权。

```
select has_table_privilege('guest', 'listing', 'select');

has_table_privilege
---------------------
false
```

以下查询使用 pg_tables 和 pg_user 目录表的输出来列出表权限，包括选择、插入、更新和删除。这只是一个示例。您可能需要指定数据库中的架构名称和表名。有关更多信息，请参阅 查询目录表。

```
SELECT
     tablename
     ,usename
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'select') AS sel
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'insert') AS ins
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'update') AS upd
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'delete') AS del
FROM
(SELECT * from pg_tables
WHERE schemaname = 'public' and tablename in ('event','listing')) as tables
,(SELECT * FROM pg_user) AS users;

tablename | usename   |  sel   |  ins  |  upd  | del
----------+-----------+--------+-------+-------+-------
event     |  john     |  true  | true  | true  | true
```

```
event     |  sally    |  false | false | false | false
event     |  elsa     |  false | false | false | false
listing   |  john     |  true  | true  | true  | true
listing   |  sally    |  false | false | false | false
listing   |  elsa     |  false | false | false | false
```

前一个查询还包含交叉联接。有关更多信息，请参阅 JOIN 示例。要查询不在 public 架构中的表，请从 WHERE 子句中删除 schemaname 条件，并在查询之前使用以下示例。

```
SET SEARCH_PATH to 'schema_name';
```

LAST_USER_QUERY_ID

返回当前会话中最近完成的用户查询的查询 ID。如果当前会话中尚未运行任何查询，last_user_query_id 将返回 -1。此函数不会返回以独占方式在领导节点上运行的查询的查询 ID。有关更多信息，请参阅 仅领导节点函数。

语法

```
last_user_query_id()
```

返回类型

返回整数。

示例

以下查询返回由用户在当前会话中运行的最近完成的查询的 ID。

```
select last_user_query_id();
```

结果如下。

```
last_user_query_id
-----------------------
    5437
(1 row)
```

以下查询返回由用户在当前会话中运行的最近完成的查询的查询 ID 和文本。

```
select query_id, query_text from sys_query_history where query_id =
 last_user_query_id();
```

结果如下。

```
 query_id, query_text
---------
+-----------------------------------------------------------------------------------------------------------------
 5556975 | select last_user_query_id() limit 100 --RequestID=<unique request ID>;
 TraceID=<unique trace ID>
```

PG_BACKEND_PID

返回处理当前会话的服务器进程的进程 ID (PID)。

> **Note**
>
> PID 不是全局唯一的。它可以在一段时间后重复使用。

语法

```
pg_backend_pid()
```

返回类型

返回整数。

示例

您可将 PG_BACKEND_PID 与日志表关联以检索当前会话的信息。例如，以下查询返回当前会话中完成的查询的查询 ID 和一部分查询文本。

```
select query, substring(text,1,40)
from stl_querytext
where pid =  PG_BACKEND_PID()
order by query desc;

 query |                substring
```

```
-------+----------------------------------------
 14831 | select query, substring(text,1,40) from
 14827 | select query, substring(path,0,80) as pa
 14826 | copy category from 's3://dw-tickit/manif
 14825 | Count rows in target table
 14824 | unload ('select * from category') to 's3
(5 rows)
```

您可将 PG_BACKEND_PID 与以下日志表中的 pid 列关联（圆括号中指明的除外）：

- STL_CONNECTION_LOG
- STL_DDLTEXT
- STL_ERROR
- STL_QUERY
- STL_QUERYTEXT
- STL_SESSIONS (process)
- STL_TR_CONFLICT
- STL_UTILITYTEXT
- STV_ACTIVE_CURSORS
- STV_INFLIGHT
- STV_LOCKS (lock_owner_pid)
- STV_RECENTS (process_id)

PG_GET_COLS

返回表或视图定义的列元数据。

语法

```
pg_get_cols('name')
```

参数

名称

Amazon Redshift 表或视图的名称。有关更多信息，请参阅 名称和标识符。

返回类型

VARCHAR

使用说明

PG_GET_COLS 函数为表或视图定义中的每个列返回一行内容。该行包含一个逗号分隔列表，其中包括 schema 名称、关系名称、列名、数据类型和列编号。SQL 结果的格式取决于所使用的 SQL 客户端。

示例

以下示例返回针对模式 public 中名为 SALES_VW 的视图和模式 mytickit1 中名为 sales 的表的结果，相应的视图和表由用户在连接的数据库 dev 中创建。

以下示例会返回名为 SALES_VW 的视图的列元数据。

```
select pg_get_cols('sales_vw');

pg_get_cols
---------------------------------------------------------------
(public,sales_vw,salesid,integer,1)
(public,sales_vw,listid,integer,2)
(public,sales_vw,sellerid,integer,3)
(public,sales_vw,buyerid,integer,4)
(public,sales_vw,eventid,integer,5)
(public,sales_vw,dateid,smallint,6)
(public,sales_vw,qtysold,smallint,7)
(public,sales_vw,pricepaid,"numeric(8,2)",8)
(public,sales_vw,commission,"numeric(8,2)",9)
(public,sales_vw,saletime,"timestamp without time zone",10)
```

以下示例会以表格式返回 SALES_VW 视图的列元数据。

```
select * from pg_get_cols('sales_vw')
cols(view_schema name, view_name name, col_name name, col_type varchar, col_num int);

view_schema | view_name | col_name   | col_type                    | col_num
------------+-----------+------------+-----------------------------+--------
public      | sales_vw  | salesid    | integer                     |       1
public      | sales_vw  | listid     | integer                     |       2
public      | sales_vw  | sellerid   | integer                     |       3
public      | sales_vw  | buyerid    | integer                     |       4
```

```
public      | sales_vw  | eventid    | integer                     |       5
public      | sales_vw  | dateid     | smallint                    |       6
public      | sales_vw  | qtysold    | smallint                    |       7
public      | sales_vw  | pricepaid  | numeric(8,2)                |       8
public      | sales_vw  | commission | numeric(8,2)                |       9
public      | sales_vw  | saletime   | timestamp without time zone |      10
```

以下示例会以表格式返回架构 mytickit1 中 SALES 表的列元数据。

```
select * from pg_get_cols('"mytickit1"."sales"')
cols(view_schema name, view_name name, col_name name, col_type varchar, col_num int);

view_schema | view_name | col_name   | col_type                    | col_num
------------+-----------+------------+-----------------------------+--------
mytickit1   | sales     | salesid    | integer                     |       1
mytickit1   | sales     | listid     | integer                     |       2
mytickit1   | sales     | sellerid   | integer                     |       3
mytickit1   | sales     | buyerid    | integer                     |       4
mytickit1   | sales     | eventid    | integer                     |       5
mytickit1   | sales     | dateid     | smallint                    |       6
mytickit1   | sales     | qtysold    | smallint                    |       7
mytickit1   | sales     | pricepaid  | numeric(8,2)                |       8
mytickit1   | sales     | commission | numeric(8,2)                |       9
mytickit1   | sales     | saletime   | timestamp without time zone |      10
```

PG_GET_GRANTEE_BY_IAM_ROLE

返回被授予指定 IAM 角色的所有用户和组。

语法

```
pg_get_grantee_by_iam_role('iam_role_arn')
```

参数

iam_role_arn

用于返回已被授予此角色的用户和组的 IAM 角色。

返回类型

VARCHAR

使用说明

PG_GET_GRANTEE_BY_IAM_ROLE 函数为每个用户或组返回一行。每行都包含被授权者名称、被授权者类型和授予的权限。被授权者类型的可能值如下：对于公有为 p，对于用户为 u，对于组为 g。

您必须是超级用户才能使用函数。

示例

以下示例指示 IAM 角色 Redshift-S3-Write 被授予给 group1 和 reg_user1。group_1 中的用户只能为 COPY 操作指定角色，用户 reg_user1 可以指定仅用于执行 UNLOAD 操作的角色。

```
select pg_get_grantee_by_iam_role('arn:aws:iam::123456789012:role/Redshift-S3-Write');
```

```
  pg_get_grantee_by_iam_role
-----------------------------
 (group_1,g,COPY)
 (reg_user1,u,UNLOAD)
```

下面的 PG_GET_GRANTEE_BY_IAM_ROLE 函数示例将结果格式化为表格。

```
select grantee, grantee_type, cmd_type FROM
 pg_get_grantee_by_iam_role('arn:aws:iam::123456789012:role/Redshift-S3-Write')
 res_grantee(grantee text, grantee_type text, cmd_type text) ORDER BY 1,2,3;
```

```
  grantee  | grantee_type | cmd_type
-----------+--------------+----------
 group_1   | g            | COPY
 reg_user1 | u            | UNLOAD
```

PG_GET_IAM_ROLE_BY_USER

返回授予用户的所有 IAM 角色和命令权限。

语法

```
pg_get_iam_role_by_user('name')
```

参数

名称

要返回 IAM 角色的用户的名称。

返回类型

VARCHAR

使用说明

PG_GET_IAM_ROLE_BY_USER 函数为每组角色和命令特权返回一行。该行包含一个逗号分隔列表，其中包括用户名、IAM 角色和命令。

结果中的值 `default` 表示用户可以指定任何可用角色来执行显示的命令。

您必须是超级用户才能使用函数。

示例

以下示例指示用户 `reg_user1` 可以指定任何可用的 IAM 角色来执行 COPY 操作。用户也可以指定 `Redshift-S3-Write` 角色进行 UNLOAD 操作。

```
select pg_get_iam_role_by_user('reg_user1');
```

```
                            pg_get_iam_role_by_user
-------------------------------------------------------------------------------
 (reg_user1,default,COPY)
 (reg_user1,arn:aws:iam::123456789012:role/Redshift-S3-Write,COPY|UNLOAD)
```

下面的 PG_GET_IAM_ROLE_BY_USER 函数示例将结果格式化为表格。

```
select username, iam_role, cmd FROM pg_get_iam_role_by_user('reg_user1')
 res_iam_role(username text, iam_role text, cmd text);
```

```
 username  |                    iam_role                     | cmd
-----------+-------------------------------------------------+------
 reg_user1 | default                                         | None
```

```
 reg_user1 | arn:aws:iam::123456789012:role/Redshift-S3-Read | COPY
```

PG_GET_LATE_BINDING_VIEW_COLS

返回数据库中所有后期绑定视图的列元数据。有关更多信息，请参阅后期绑定视图。

语法

```
pg_get_late_binding_view_cols()
```

返回类型

VARCHAR

使用说明

PG_GET_LATE_BINDING_VIEW_COLS 函数为后期绑定视图中的每个列返回一行内容。该行包含一个逗号分隔列表，其中包括 schema 名称、关系名称、列名、数据类型和列编号。

示例

以下示例返回所有后期绑定视图的列元数据。

```
select pg_get_late_binding_view_cols();

pg_get_late_binding_view_cols
------------------------------------------------------------
(public,myevent,eventname,"character varying(200)",1)
(public,sales_lbv,salesid,integer,1)
(public,sales_lbv,listid,integer,2)
(public,sales_lbv,sellerid,integer,3)
(public,sales_lbv,buyerid,integer,4)
(public,sales_lbv,eventid,integer,5)
(public,sales_lbv,dateid,smallint,6)
(public,sales_lbv,qtysold,smallint,7)
(public,sales_lbv,pricepaid,"numeric(8,2)",8)
(public,sales_lbv,commission,"numeric(8,2)",9)
(public,sales_lbv,saletime,"timestamp without time zone",10)
(public,event_lbv,eventid,integer,1)
(public,event_lbv,venueid,smallint,2)
(public,event_lbv,catid,smallint,3)
(public,event_lbv,dateid,smallint,4)
(public,event_lbv,eventname,"character varying(200)",5)
```

```
(public,event_lbv,starttime,"timestamp without time zone",6)
```

以下示例以表格式返回所有后期绑定视图的列元数据。

```
select * from pg_get_late_binding_view_cols() cols(view_schema name, view_name name,
 col_name name, col_type varchar, col_num int);
view_schema | view_name | col_name   | col_type                    | col_num
------------+-----------+------------+-----------------------------+--------
public      | sales_lbv | salesid    | integer                     |       1
public      | sales_lbv | listid     | integer                     |       2
public      | sales_lbv | sellerid   | integer                     |       3
public      | sales_lbv | buyerid    | integer                     |       4
public      | sales_lbv | eventid    | integer                     |       5
public      | sales_lbv | dateid     | smallint                    |       6
public      | sales_lbv | qtysold    | smallint                    |       7
public      | sales_lbv | pricepaid  | numeric(8,2)                |       8
public      | sales_lbv | commission | numeric(8,2)                |       9
public      | sales_lbv | saletime   | timestamp without time zone |      10
public      | event_lbv | eventid    | integer                     |       1
public      | event_lbv | venueid    | smallint                    |       2
public      | event_lbv | catid      | smallint                    |       3
public      | event_lbv | dateid     | smallint                    |       4
public      | event_lbv | eventname  | character varying(200)      |       5
public      | event_lbv | starttime  | timestamp without time zone |       6
```

PG_GET_SESSION_ROLES

返回当前登录的用户的会话角色。用户的会话角色是由身份提供者 (IdP) 为登录用户定义的组。例如，身份提供者 (IdP) (如 Microsoft Azure Active Directory (Azure AD)) 验证用户的身份，并在用户登录过程中提供该用户所属的所有外部组。这些外部组转换为 Amazon Redshift 角色，可在当前会话期间使用。这些角色称为会话角色。管理员可以向会话角色授予与其他 Amazon Redshift 角色类似的权限。有关使用角色的信息，请参阅基于角色的访问控制 (RBAC)。有关使用身份提供者 (IdP) 管理身份的信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 的原生身份提供者 (IdP) 联合身份验证。

要查看 Amazon Redshift 目录中定义的角色，请以管理员或超级用户身份连接到数据库，然后查询系统视图 SVV_ROLES。

语法

```
pg_get_session_roles()
```

返回类型

一组由两个值组成的行。第一个值由两部分组成，用冒号 (:) 分隔，其中包含 idp-namespace:role-name。idp-namespace 是身份提供者 (IdP) 的命名空间。role-name 是身份提供者 (IdP) 中的外部组的名称。第二个值包含作为角色标识符的 role-id。

使用说明

PG_GET_SESSION_ROLES 函数为每个返回的会话角色返回一行。

示例

以下示例为 Azure Active Directory IdP 中的每个角色返回一行。返回的列强制转换为 sess_roles，列为 name 和 roleid。每个 name 由 Azure Active Directory 命名空间和 Azure Active Directory 中的组名称组成。

```
SELECT * FROM pg_get_session_roles() AS sess_roles(name name, roleid integer);

name                   roleid
--------------------------------
my_aad:test_group_1   106204
my_aad:test_group_2   106205
my_aad:test_group_3   106206
my_aad:test_group_4   106207
my_aad:test_group_5   106208
```

以下示例为当前登录的 IAM 用户所属的每个 IAM 组返回一行。返回的列强制转换为 sess_roles，列为 name 和 roleid。每个 name 由 IAM 命名空间和 IAM 组名称组成。

```
SELECT * FROM pg_get_session_roles() AS sess_roles(name name, roleid integer);

name                   roleid
--------------------------------
IAM:myGroup           110332
```

PG_LAST_COPY_COUNT

返回由当前会话中运行的上一条 COPY 命令加载的行的数量。将使用上一个 COPY ID（开始加载过程的上一个 COPY 的查询 ID）更新 PG_LAST_COPY_COUNT，即使加载失败也是如此。当 COPY 命令开始加载过程时，将更新查询 ID 和 COPY ID。

如果 COPY 因语法错误或权限不足而失败，则不会更新 COPY ID 并且 PG_LAST_COPY_COUNT 将返回上一个 COPY 的计数。如果未在当前会话中运行任何 COPY 命令，或如果上一个 COPY 在加载过程中失败，PG_LAST_COPY_COUNT 将返回 0。有关更多信息，请参阅 PG_LAST_COPY_ID。

语法

```
pg_last_copy_count()
```

返回类型

返回 BIGINT。

示例

以下查询返回由当前会话中的最新 COPY 命令加载的行的数量。

```
select pg_last_copy_count();

pg_last_copy_count
--------------------
             192497
(1 row)
```

PG_LAST_COPY_ID

返回当前会话中最近完成的 COPY 命令的查询 ID。如果当前会话中尚未运行任何 COPY 命令，PG_LAST_COPY_ID 将返回 -1。

当 COPY 命令开始加载过程时，将更新 PG_LAST_COPY_ID 的值。如果 COPY 因加载数据无效而失败，则会更新 COPY ID，因此您可在查询 STL_LOAD_ERRORS 表时使用 PG_LAST_COPY_ID。如果回滚了 COPY 事务，则不会更新 COPY ID。

如果 COPY 命令因加载过程开始前出现的错误（如语法错误、访问错误、凭证无效或权限不足）而失败，则不会更新 COPY ID。如果 COPY 在分析压缩步骤（在成功连接之后、数据加载之前开始）中失败，则将不会更新 COPY ID。

语法

```
pg_last_copy_id()
```

返回类型

返回整数。

示例

以下查询返回当前会话中的最新 COPY 命令的查询 ID。

```
select pg_last_copy_id();

pg_last_copy_id
---------------
          5437
(1 row)
```

以下查询将 STL_LOAD_ERRORS 联接到 STL_LOADERROR_DETAIL 以查看当前会话中的最近一次加载过程中发生的错误的详细信息：

```
select d.query, substring(d.filename,14,20),
d.line_number as line,
substring(d.value,1,16) as value,
substring(le.err_reason,1,48) as err_reason
from stl_loaderror_detail d, stl_load_errors le
where d.query = le.query
and d.query = pg_last_copy_id();

 query |    substring      | line |  value   |                    err_reason
-------+-------------------+------+----------
+-------------------------------------------------
    558| allusers_pipe.txt |  251 | 251      | String contains invalid or unsupported
 UTF8 code
    558| allusers_pipe.txt |  251 | ZRU29FGR | String contains invalid or unsupported
 UTF8 code
    558| allusers_pipe.txt |  251 | Kaitlin  | String contains invalid or unsupported
 UTF8 code
    558| allusers_pipe.txt |  251 | Walter   | String contains invalid or unsupported
 UTF8 code
```

PG_LAST_UNLOAD_ID

返回当前会话中最近完成的 UNLOAD 命令的查询 ID。如果当前会话中未运行任何 UNLOAD 命令，则 PG_LAST_UNLOAD_ID 将返回 -1。

当 UNLOAD 命令开始加载过程时，将更新 PG_LAST_UNLOAD_ID 的值。如果 UNLOAD 因加载数据无效而失败，则将更新 UNLOAD ID，以便您能使用 UNLOAD ID 进行进一步调查。如果回滚 UNLOAD 事务，则不会更新 UNLOAD ID。

如果 UNLOAD 命令因加载过程开始前出现的错误（如语法错误、访问错误、凭证无效或权限不足）而失败，则不会更新 UNLOAD ID。

语法

```
PG_LAST_UNLOAD_ID()
```

返回类型

返回整数。

示例

以下查询返回当前会话中的最新 UNLOAD 命令的查询 ID。

```
select PG_LAST_UNLOAD_ID();

PG_LAST_UNLOAD_ID
---------------
          5437
(1 row)
```

PG_LAST_QUERY_ID

返回当前会话中最近完成的查询的查询 ID。如果当前会话中尚未运行任何查询，PG_LAST_QUERY_ID 将返回 -1。PG_LAST_QUERY_ID 不会返回以独占方式在领导节点上运行的查询的查询 ID。有关更多信息，请参阅 仅领导节点函数。

语法

```
pg_last_query_id()
```

返回类型

返回整数。

示例

以下查询返回当前会话中最近完成的查询的 ID。

```
select pg_last_query_id();
```

结果如下。

```
pg_last_query_id
----------------
           5437
(1 row)
```

以下查询返回当前会话中最近完成的查询的查询 ID 和文本。

```
select query, trim(querytxt) as sqlquery
from stl_query
where query = pg_last_query_id();
```

结果如下。

```
query | sqlquery
------+-------------------------------------------------
 5437 | select name, loadtime from stl_file_scan where loadtime > 1000000;
(1 rows)
```

PG_LAST_UNLOAD_COUNT

返回由当前会话中完成的上一条 UNLOAD 命令卸载的行的数量。将使用上一个 UNLOAD 的查询 ID 更新 PG_LAST_UNLOAD_COUNT，即使操作失败也是如此。将在完成 UNLOAD 时更新此查询 ID。如果 UNLOAD 因语法错误或权限不足而失败，PG_LAST_UNLOAD_COUNT 将返回上一个 UNLOAD 的计数。如果当前会话中尚未完成 UNLOAD 命令，或如果上一个 UNLOAD 在卸载操作期间失败，PG_LAST_UNLOAD_COUNT 将返回 0。

语法

```
pg_last_unload_count()
```

返回类型

返回 BIGINT。

示例

以下查询返回当前会话中的最新 UNLOAD 命令卸载的行的数量。

```
select pg_last_unload_count();

pg_last_unload_count
--------------------
              192497
(1 row)
```

SLICE_NUM 函数

返回一个整数，该整数对应于某个行的数据所在的集群中的切片数。SLICE_NUM 未采用任何参数。

语法

```
SLICE_NUM()
```

返回类型

SLICE_NUM 函数返回整数。

示例

以下示例显示包含 EVENTS 中前十个 EVENT 行的数据的切片：

```
select distinct eventid, slice_num() from event order by eventid limit 10;

 eventid | slice_num
---------+-----------
       1 |         1
       2 |         2
       3 |         3
       4 |         0
       5 |         1
       6 |         2
       7 |         3
       8 |         0
       9 |         1
      10 |         2
(10 rows)
```

以下示例将返回一个代码 (10000) 以演示没有 FROM 语句的查询是如何在领导节点上运行的：

```
select slice_num();
slice_num
-----------
10000
(1 row)
```

USER

CURRENT_USER 的同义词。请参阅 CURRENT_USER。

ROLE_IS_MEMBER_OF

如果角色是其他角色的成员，则返回 true。超级用户可以检查所有角色的成员身份。具有 ACCESS SYSTEM TABLE 权限的普通用户可以检查所有用户的成员身份。否则，普通用户只能检查其具有访问权限的角色。如果提供的角色不存在或当前用户无权访问该角色，则 Amazon Redshift 会返回错误。

语法

```
role_is_member_of( role_name,  granted_role_name)
```

参数

role_name

角色的名称。

granted_role_name

所授予角色的名称。

返回类型

返回一个布尔值。

示例

以下查询确认该角色不是 role1 的成员，也不是 role2 的成员。

```
SELECT role_is_member_of('role1', 'role2');
```

```
 role_is_member_of
-------------------
             False
```

USER_IS_MEMBER_OF

如果用户是某个角色或组的成员，则返回 true。超级用户可以检查所有用户的成员身份。属于 sy:secadmin 或 sys:superuser 角色成员的普通用户可以检查所有用户的成员身份。否则，普通用户只能检查自己的成员身份。如果提供的身份不存在或当前用户无权访问该角色，则 Amazon Redshift 会发出错误消息。

语法

```
user_is_member_of( user_name,  role_name | group_name)
```

参数

user_name

用户的名称。

role_name

角色的名称。

group_name

组的名称。

返回类型

返回一个布尔值。

示例

以下查询确认该用户不是 role1 的成员。

```
SELECT user_is_member_of('reguser', 'role1');

 user_is_member_of
-------------------
           False
```

VERSION

VERSION 函数返回有关当前安装版本的详细信息（在末尾有特定的 Amazon Redshift 版本信息）。

> **Note**
>
> 这是领导节点函数。如果此函数引用了用户创建的表、STL/STV 系统表或 SVV/SVL 系统视图，它将返回错误。

语法

```
VERSION()
```

返回类型

返回 CHAR 或 VARCHAR 字符串。

示例

以下示例显示了当前集群的集群版本信息：

```
select version();
```

```
 version
----------------------------------------------------------------------------------------------------------------

PostgreSQL 8.0.2 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 3.4.2 20041017 (Red
Hat 3.4.2-6.fc3), Redshift 1.0.12103
```

其中 1.0.12103 是集群版本号。

> **Note**
>
> 要强制您的集群更新到最新集群版本，请调整您的维护时段。

保留字

下面是 Amazon Redshift 保留关键字的列表。您可以通过分隔标识符（双引号）使用保留关键字。

> **Note**
>
> 虽然 START 和 CONNECT 不是保留字，但如果您在查询中使用 START 和 CONNECT 作为表别名，请使用分隔标识符或 AS，以避免在运行时失败。

有关更多信息，请参阅 名称和标识符。

```
AES128
AES256
ALL
ALLOWOVERWRITE
ANALYSE
ANALYZE
AND
ANY
ARRAY
AS
ASC
AUTHORIZATION
AZ64
BACKUP
BETWEEN
BINARY
BLANKSASNULL
BOTH
BYTEDICT
BZIP2
CASE
CAST
CHECK
COLLATE
COLUMN
CONSTRAINT
CREATE
CREDENTIALS
CROSS
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURRENT_USER_ID
DEFAULT
```

```
DEFERRABLE
DEFLATE
DEFRAG
DELTA
DELTA32K
DESC
DISABLE
DISTINCT
DO
ELSE
EMPTYASNULL
ENABLE
ENCODE
ENCRYPT
ENCRYPTION
END
EXCEPT
EXPLICIT
FALSE
FOR
FOREIGN
FREEZE
FROM
FULL
GLOBALDICT256
GLOBALDICT64K
GRANT
GROUP
GZIP
HAVING
IDENTITY
IGNORE
ILIKE
IN
INITIALLY
INNER
INTERSECT
INTERVAL
INTO
IS
ISNULL
JOIN
LEADING
LEFT
```

```
LIKE
LIMIT
LOCALTIME
LOCALTIMESTAMP
LUN
LUNS
LZO
LZOP
MINUS
MOSTLY16
MOSTLY32
MOSTLY8
NATURAL
NEW
NOT
NOTNULL
NULL
NULLS
OFF
OFFLINE
OFFSET
OID
OLD
ON
ONLY
OPEN
OR
ORDER
OUTER
OVERLAPS
PARALLEL
PARTITION
PERCENT
PERMISSIONS
PIVOT
PLACING
PRIMARY
RAW
READRATIO
RECOVER
REFERENCES
REJECTLOG
RESORT
RESPECT
```

```
RESTORE
RIGHT
SELECT
SESSION_USER
SIMILAR
SNAPSHOT
SOME
SYSDATE
SYSTEM
TABLE
TAG
TDES
TEXT255
TEXT32K
THEN
TIMESTAMP
TO
TOP
TRAILING
TRUE
TRUNCATECOLUMNS
UNION
UNIQUE
UNNEST
UNPIVOT
USER
USING
VERBOSE
WALLET
WHEN
WHERE
WITH
WITHOUT
```

系统表和视图参考

主题

系统表和视图

Amazon Redshift 有许多包含系统运行方式相关信息的系统表和视图。您可以像查询任何其他数据库表那样查询这些系统表和视图。本部分提供一些系统表查询示例并予以讲解：

- 生成的系统表和视图的类型有何不同
- 您可以从这些表中获得哪些类型的信息
- 如何将 Amazon Redshift 系统表联接到目录表
- 如何管理系统表日志文件的增长

某些系统表只能由 AWS 人员用于诊断目的。以下部分讨论可供系统管理员或其他数据库用户查询有用信息的系统表。

> **Note**
>
> 自动或手动集群备份（快照）中不包含系统表。STL 系统视图保留七天的日志历史记录。保留日志不要求客户执行任何操作，但如果需要将日志数据存储超过 7 天，则必须定期将日志数据复制到其他表，或将日志数据卸载到 Amazon S3。

系统表和视图类型

系统表和视图分为几种类型：

- SVV 视图包含有关数据库对象的信息，并引用了临时 STV 表。
- SYS 视图用于监控预置集群和无服务器工作组的查询和工作负载使用情况。
- STL 视图从长久保存到磁盘的用于提供系统历史记录的日志生成。
- STV 表是虚拟系统表，包含当前系统数据的快照。它们基于临时的内存数据，不会长久保存到基于磁盘的日志或常规表中。
- SVCS 视图提供了有关主集群和并发扩展集群上的查询的详细信息。
- SVL 视图提供有关主集群查询的详细信息。

系统表及视图不使用与常规表相同的一致性模型。在查询它们时，特别是查询 STV 表和 SVV 视图时，一定要注意这个问题。例如，对于包含列 c1 的常规表 t1，下面的查询不会返回任何行：

```
select * from t1
where c1 > (select max(c1) from t1)
```

但是，下面针对系统表的查询可能返回行：

```
select * from stv_exec_state
where currenttime > (select max(currenttime) from stv_exec_state)
```

该查询可能返回行的原因在于：currenttime 是临时的，查询中的两个引用在求值时可能返回不同的值。

另一方面，下面的查询也可能不返回行：

```
select * from stv_exec_state
```

```
where currenttime = (select max(currenttime) from stv_exec_state)
```

系统表和视图中的数据可见性

系统表及视图中的数据有两类可见性：对用户可见和对超级用户可见。

只有具有超级用户权限的用户才能看到属于超级用户可见类别的表中的数据。普通用户可以查看对用户可见的表中的数据。要使普通用户能够访问对超级用户可见的表，请向普通用户授予对该表的 SELECT 权限。有关更多信息，请参阅 GRANT。

默认情况下，在大多数对用户可见的表中，普通用户看不到其他用户生成的行。如果向普通用户授予 SYSLOG ACCESS UNRESTRICTED，则该用户可以查看用户可见表中的所有行，包括由其他用户生成的行。有关更多信息，请参阅ALTER USER或CREATE USER。SVV_TRANSACTIONS 中的所有行都对所有用户可见。有关数据可见性的更多信息，请参阅 AWS re:Post 知识库文章如何允许 Amazon Redshift 数据库普通用户查看我的集群中其他用户系统表中的数据？。

对于元数据视图，Amazon Redshift 不允许具有 SYSLOG ACCESS UNRESTRICTED 权限的用户查看。

> **Note**
>
> 如果向用户授予对系统表的无限制访问权限，用户便可以看到由其他用户生成的数据。例如，STL_QUERY 和 STL_QUERY_TEXT 包含 INSERT、UPDATE 和 DELETE 语句的完整文本 (其中可能包含敏感的用户生成数据)。

超级用户可以查看所有表中的所有行。要使普通用户能够访问对超级客户可见的表，GRANT请向普通用户授予对该表的 SELECT 权限。

筛选系统生成的查询

与查询有关的系统表和视图（如 SVL_QUERY_SUMMARY、SVL_QLOG 等）通常包含大量自动生成的语句，Amazon Redshift 使用这些语句监控数据库的状态。这些系统生成的查询对超级用户可见，但用处不大。从使用 `userid` 列的系统表或视图中进行选择时，如果要过滤掉它们，则可在 WHERE 子句中添加条件 `userid > 1`。例如：

```
select * from svl_query_summary where userid > 1
```

将仅预调配的查询迁移到 SYS 监控视图查询

从预调配集群迁移到 Amazon Redshift Serverless

如果您要将预调配集群迁移到 Amazon Redshift Serverless，则您可能具有使用以下系统视图的查询，而这些视图仅存储预调配集群中的数据。

- 所有 STL 视图
- 所有 STV 视图
- 所有 SVCS 视图
- 所有 SVL 视图
- 部分 SVV 视图
  - 有关 Amazon Redshift Serverless 中不支持的 SVV 视图的完整列表，请参阅《Amazon Redshift 管理指南》中的使用 Amazon Redshift Serverless 监控查询和工作负载底部的列表。

要继续使用查询，请将它们重新调整为使用在 SYS 监控视图中定义的列，这些列与仅预调配视图中的列相对应。要查看仅预调配视图和 SYS 监控视图之间的映射关系，请转至用于迁移到 SYS 监控视图的系统视图映射

停留在预调配集群上时更新查询

如果您不迁移到 Amazon Redshift Serverless，则可能仍需要更新现有查询。SYS 监控视图专为易于使用和降低复杂性而设计，并提供了一系列完整的指标来进行有效的监控和故障排除。使用 SYS 视图（如 SYS_QUERY_HISTORY 和 SYS_QUERY_DETAIL），其中整合了多个仅预调配视图的信息，可以简化查询。

使用 SYS 监控视图改进查询标识符跟踪

诸如 SYS_QUERY_HISTORY 和 SYS_QUERY_DETAIL 之类的 SYS 监控视图包含 query_id 列，该列包含用户的查询标识符。同样，诸如 STL_QUERY 和 SVL_QLOG 之类的仅预置视图包含查询列，该列还包含查询标识符。但是，SYS 系统视图中记录的查询标识符与仅预置视图中记录的查询标识符不同。

SYS 视图的 query_id 列值与仅预置视图的查询列值之间的区别如下：

- 在 SYS 视图中，query_id 列以原始形式记录用户提交的查询。为了提高性能，Amazon Redshift 优化器可能会将它们分解为子查询，但您运行的单个查询在 SYS_QUERY_HISTORY 中仍然只有一行。如果您想查看各个子查询，可以在 SYS_QUERY_DETAIL 中找到它们。
- 在仅预置视图中，查询列记录子查询级别的查询。如果 Amazon Redshift 优化器将您的原始查询重写为多个子查询，则对于您运行的单个查询，STL_QUERY 中将有多行具有不同的查询标识符值。

将监控和诊断查询从仅预置视图迁移到 SYS 视图时，请考虑这种差异并相应地编辑查询。有关 Amazon Redshift 如何处理查询的更多信息，请参阅查询计划和执行工作流程。

示例

有关 Amazon Redshift 在仅预调配视图和 SYS 监控视图中如何以不同的方式记录查询的示例，请参阅以下示例查询。这是按照在 Amazon Redshift 中运行的方式编写的查询。

```
SELECT
  s_name
  , COUNT(*) AS numwait
FROM
  supplier,
  lineitem l1,
  orders,
  nation
WHERE    s_suppkey = l1.l_suppkey
         AND o_orderkey = l1.l_orderkey
         AND o_orderstatus = 'F'
         AND l1.l_receiptdate > l1.l_commitdate
         AND EXISTS (SELECT
                       *
                     FROM
                       lineitem l2
                     WHERE  l2.l_orderkey = l1.l_orderkey
                            AND l2.l_suppkey <> l1.l_suppkey )
         AND NOT EXISTS (SELECT
                           *
                         FROM
                           lineitem l3
                         WHERE  l3.l_orderkey = l1.l_orderkey
                                AND l3.l_suppkey <> l1.l_suppkey
                                AND l3.l_receiptdate > l3.l_commitdate )
         AND s_nationkey = n_nationkey
         AND n_name = 'UNITED STATES'
```

```
GROUP BY
  s_name
ORDER BY
  numwait DESC
  , s_name LIMIT 100;
```

在后台，Amazon Redshift 查询优化器将上述用户提交的查询重写为 5 个子查询。

第一个子查询创建一个临时表来实现一个子查询。

```
CREATE TEMP TABLE volt_tt_606590308b512(l_orderkey
                                       , l_suppkey
                                       , s_name   ) AS SELECT
                                                       l1.l_orderkey
                                                       , l1.l_suppkey
                                                       , public.supplier.s_name
                                                     FROM
                                                       public.lineitem AS l1,
                                                       public.nation,
                                                       public.orders,
                                                       public.supplier
                                                     WHERE  l1.l_commitdate <
 l1.l_receiptdate
                                                            AND l1.l_orderkey =
 public.orders.o_orderkey
                                                            AND l1.l_suppkey =
 public.supplier.s_suppkey
                                                            AND public.nation.n_name
 = 'UNITED STATES'::CHAR(8)
                                                            AND
 public.nation.n_nationkey = public.supplier.s_nationkey
                                                            AND
 public.orders.o_orderstatus = 'F'::CHAR(1);
```

第二个子查询从临时表中收集统计数据。

```
padb_fetch_sample: select count(*) from volt_tt_606590308b512;
```

第三个子查询创建另一个临时表来实现另一个子查询，引用上面创建的临时表。

```
CREATE TEMP TABLE volt_tt_606590308c2ef(l_orderkey
                                       , l_suppkey) AS (SELECT
```

```
 volt_tt_606590308b512.l_orderkey
                                                              ,
 volt_tt_606590308b512.l_suppkey
                                                            FROM
                                                              public.lineitem AS l2,
                                                              volt_tt_606590308b512
                                                            WHERE  l2.l_suppkey <>
 volt_tt_606590308b512.l_suppkey
                                                                   AND l2.l_orderkey =
 volt_tt_606590308b512.l_orderkey)
                                                                   EXCEPT distinct (SELECT
 volt_tt_606590308b512.l_orderkey, volt_tt_606590308b512.l_suppkey
                                                                   FROM public.lineitem AS
 l3, volt_tt_606590308b512
                                                                   WHERE l3.l_commitdate <
 l3.l_receiptdate
                                                                     AND l3.l_suppkey <>
 volt_tt_606590308b512.l_suppkey
                                                                     AND l3.l_orderkey =
 volt_tt_606590308b512.l_orderkey);
```

第四个子查询再次收集临时表的统计数据。

```
padb_fetch_sample: select count(*) from volt_tt_606590308c2ef
```

最后一个子查询使用上面创建的临时表生成输出。

```
SELECT
  volt_tt_606590308b512.s_name AS s_name
  , COUNT(*) AS numwait
FROM
  volt_tt_606590308b512,
  volt_tt_606590308c2ef
WHERE    volt_tt_606590308b512.l_orderkey = volt_tt_606590308c2ef.l_orderkey
         AND volt_tt_606590308b512.l_suppkey = volt_tt_606590308c2ef.l_suppkey
GROUP BY
  1
ORDER BY
  2 DESC
  , 1 ASC LIMIT 100;
```

在仅预置系统视图 STL_QUERY 中，Amazon Redshift 在子查询级别记录了五行，如下所示：

```
SELECT userid, xid, pid, query, querytxt::varchar(100);
FROM stl_query
WHERE xid = 48237350
ORDER BY xid, starttime;

 userid |   xid    |    pid     |  query   |
    querytxt
--------+----------+------------+----------
+-----------------------------------------------------------------------------------------------------------
    101 | 48237350 | 1073840810 | 12058151 | CREATE TEMP TABLE
 volt_tt_606590308b512(l_orderkey, l_suppkey, s_name) AS SELECT l1.l_orderkey, l1.l
    101 | 48237350 | 1073840810 | 12058152 | padb_fetch_sample: select count(*) from
 volt_tt_606590308b512
    101 | 48237350 | 1073840810 | 12058156 | CREATE TEMP TABLE
 volt_tt_606590308c2ef(l_orderkey, l_suppkey) AS (SELECT volt_tt_606590308b512.l_or
    101 | 48237350 | 1073840810 | 12058168 | padb_fetch_sample: select count(*) from
 volt_tt_606590308c2ef
    101 | 48237350 | 1073840810 | 12058170 | SELECT s_name , COUNT(*) AS numwait FROM
 supplier, lineitem l1, orders, nation WHERE s_suppkey = l1.
(5 rows)
```

在 SYS 监控视图 SYS_QUERY_HISTORY 中，Amazon Redshift 按如下方式记录查询：

```
SELECT user_id, transaction_id, session_id, query_id, query_text::varchar(100)
FROM sys_query_history
WHERE transaction_id = 48237350
ORDER BY start_time;

 user_id | transaction_id | session_id | query_id |
          query_text
---------+----------------+------------+----------
+-----------------------------------------------------------------------------------------------------------
     101 |       48237350 | 1073840810 | 12058149 | SELECT s_name , COUNT(*) AS numwait
 FROM supplier, lineitem l1, orders, nation WHERE s_suppkey = l1.
```

在 SYS_QUERY_DETAIL 中，可以使用来自 SYS_QUERY_HISTORY 的 query_id 值查找子查询级详细信息。child_query_sequence 列显示子查询的执行顺序。有关 SYS_QUERY_DETAIL 中各列的更多信息，请参阅 SYS_QUERY_DETAIL。

```
select user_id,
       query_id,
       child_query_sequence,
```

```
       stream_id,
       segment_id,
       step_id,
       start_time,
       end_time,
       duration,
       blocks_read,
       blocks_write,
       local_read_io,
       remote_read_io,
       data_skewness,
       time_skewness,
       is_active,
       spilled_block_local_disk,
       spilled_block_remote_disk
from sys_query_detail
where query_id = 12058149
      and step_id = -1
order by query_id,
         child_query_sequence,
         stream_id,
         segment_id,
         step_id;

 user_id | query_id | child_query_sequence | stream_id | segment_id | step_id |
       start_time         |          end_time          | duration | blocks_read |
 blocks_write | local_read_io | remote_read_io | data_skewness | time_skewness |
 is_active | spilled_block_local_disk | spilled_block_remote_disk
---------+----------+----------------------+-----------+------------+---------
+----------------------------+----------------------------+----------+-------------
+--------------+---------------+----------------+---------------+---------------
+-----------+--------------------------+--------------------------
     101 | 12058149 |                    1 |         0 |          0 |      -1 |
 2023-09-27 15:40:38.512415 | 2023-09-27 15:40:38.533333 |    20918 |           0 |
         0 |             0 |              0 |             0 |            44 | f
 |                         0 |                           0
     101 | 12058149 |                    1 |         1 |          1 |      -1 |
 2023-09-27 15:40:39.931437 | 2023-09-27 15:40:39.972826 |    41389 |          12 |
         0 |            12 |              0 |             0 |            77 | f
 |                         0 |                           0
     101 | 12058149 |                    1 |         2 |          2 |      -1 |
 2023-09-27 15:40:40.584412 | 2023-09-27 15:40:40.613982 |    29570 |          32 |
         0 |            32 |              0 |             0 |            25 | f
 |                         0 |                           0
```

```
    101 | 12058149 |                    1 |         2 |          3 |      -1 |
2023-09-27 15:40:40.582038 | 2023-09-27 15:40:40.615758 |    33720 |           0 |
        0 |             0 |              0 |             0 |             1 | f
|                         0 |                            0
    101 | 12058149 |                    1 |         3 |          4 |      -1 |
2023-09-27 15:40:46.668766 | 2023-09-27 15:40:46.705456 |    36690 |          24 |
        0 |            15 |              0 |             0 |            17 | f
|                         0 |                            0
    101 | 12058149 |                    1 |         4 |          5 |      -1 |
2023-09-27 15:40:46.707209 | 2023-09-27 15:40:46.709176 |     1967 |           0 |
        0 |             0 |              0 |             0 |            18 | f
|                         0 |                            0
    101 | 12058149 |                    1 |         4 |          6 |      -1 |
2023-09-27 15:40:46.70656  | 2023-09-27 15:40:46.71289  |     6330 |           0 |
        0 |             0 |              0 |             0 |             0 | f
|                         0 |                            0
    101 | 12058149 |                    1 |         5 |          7 |      -1 |
2023-09-27 15:40:46.71405  | 2023-09-27 15:40:46.714343 |      293 |           0 |
        0 |             0 |              0 |             0 |             0 | f
|                         0 |                            0
    101 | 12058149 |                    2 |         0 |          0 |      -1 |
2023-09-27 15:40:52.083907 | 2023-09-27 15:40:52.087854 |     3947 |           0 |
        0 |             0 |              0 |             0 |            35 | f
|                         0 |                            0
    101 | 12058149 |                    2 |         1 |          1 |      -1 |
2023-09-27 15:40:52.089632 | 2023-09-27 15:40:52.091129 |     1497 |           0 |
        0 |             0 |              0 |             0 |            11 | f
|                         0 |                            0
    101 | 12058149 |                    2 |         1 |          2 |      -1 |
2023-09-27 15:40:52.089008 | 2023-09-27 15:40:52.091306 |     2298 |           0 |
        0 |             0 |              0 |             0 |             0 | f
|                         0 |                            0
    101 | 12058149 |                    3 |         0 |          0 |      -1 |
2023-09-27 15:40:56.882013 | 2023-09-27 15:40:56.897282 |    15269 |           0 |
        0 |             0 |              0 |             0 |            29 | f
|                         0 |                            0
    101 | 12058149 |                    3 |         1 |          1 |      -1 |
2023-09-27 15:40:59.718554 | 2023-09-27 15:40:59.722789 |     4235 |           0 |
        0 |             0 |              0 |             0 |            13 | f
|                         0 |                            0
    101 | 12058149 |                    3 |         2 |          2 |      -1 |
2023-09-27 15:40:59.800382 | 2023-09-27 15:40:59.807388 |     7006 |           0 |
        0 |             0 |              0 |             0 |            58 | f
|                         0 |                            0
```

```
    101 | 12058149 |                    3 |         3 |          3 |      -1 |
2023-09-27 15:41:06.488685 | 2023-09-27 15:41:06.493825 |     5140 |           0 |
        0 |             0 |              0 |             0 |           56 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         3 |          4 |      -1 |
2023-09-27 15:41:06.486206 | 2023-09-27 15:41:06.497756 |    11550 |           0 |
        0 |             0 |              0 |             0 |            2 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         4 |          5 |      -1 |
2023-09-27 15:41:06.499201 | 2023-09-27 15:41:06.500851 |     1650 |           0 |
        0 |             0 |              0 |             0 |           15 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         4 |          6 |      -1 |
2023-09-27 15:41:06.498609 | 2023-09-27 15:41:06.500949 |     2340 |           0 |
        0 |             0 |              0 |             0 |            0 | f
|                         0 |                           0
    101 | 12058149 |                    3 |         5 |          7 |      -1 |
2023-09-27 15:41:06.502945 | 2023-09-27 15:41:06.503282 |      337 |           0 |
        0 |             0 |              0 |             0 |            0 | f
|                         0 |                           0
    101 | 12058149 |                    4 |         0 |          0 |      -1 |
2023-09-27 15:41:06.62899  | 2023-09-27 15:41:06.631452 |     2462 |           0 |
        0 |             0 |              0 |             0 |           22 | f
|                         0 |                           0
    101 | 12058149 |                    4 |         1 |          1 |      -1 |
2023-09-27 15:41:06.632313 | 2023-09-27 15:41:06.63391  |     1597 |           0 |
        0 |             0 |              0 |             0 |           20 | f
|                         0 |                           0
    101 | 12058149 |                    4 |         1 |          2 |      -1 |
2023-09-27 15:41:06.631726 | 2023-09-27 15:41:06.633813 |     2087 |           0 |
        0 |             0 |              0 |             0 |            0 | f
|                         0 |                           0
    101 | 12058149 |                    5 |         0 |          0 |      -1 |
2023-09-27 15:41:12.571974 | 2023-09-27 15:41:12.584234 |    12260 |           0 |
        0 |             0 |              0 |             0 |           39 | f
|                         0 |                           0
    101 | 12058149 |                    5 |         0 |          1 |      -1 |
2023-09-27 15:41:12.569815 | 2023-09-27 15:41:12.585391 |    15576 |           0 |
        0 |             0 |              0 |             0 |            4 | f
|                         0 |                           0
    101 | 12058149 |                    5 |         1 |          2 |      -1 |
2023-09-27 15:41:13.758513 | 2023-09-27 15:41:13.76401  |     5497 |           0 |
        0 |             0 |              0 |             0 |           39 | f
|                         0 |                           0
```

```
    101 | 12058149 |                    5 |         1 |           3 |      -1 |
 2023-09-27 15:41:13.749    | 2023-09-27 15:41:13.772987 |    23987 |           0 |
         0 |             0 |              0 |             0 |            32 | f
 |                         0 |                            0
    101 | 12058149 |                    5 |         2 |           4 |      -1 |
 2023-09-27 15:41:13.799526 | 2023-09-27 15:41:13.813506 |    13980 |           0 |
         0 |             0 |              0 |             0 |            62 | f
 |                         0 |                            0
    101 | 12058149 |                    5 |         2 |           5 |      -1 |
 2023-09-27 15:41:13.798823 | 2023-09-27 15:41:13.813651 |    14828 |           0 |
         0 |             0 |              0 |             0 |             0 | f
 |                         0 |                            0
(28 rows)
```

系统表查询、进程和会话 ID

在对系统表中显示的查询、进程和会话 ID 进行分析时，请注意以下几点：

- 查询 ID 值（在 `query_id` 和 `query` 等列中）可在以后重复使用。
- 进程 ID 值或会话 ID 值（在 `process_id`、`pid` 和 `session_id` 等列中）可在以后重复使用。
- 事务 ID 值（在 `transaction_id` 和 `xid` 等列中）是唯一的。

SVV 元数据视图

SVV 视图是 Amazon Redshift 中的系统视图，其中包含有关数据库对象的信息。

> **Note**
>
> 如果数据库响应因任何原因失败，Amazon Redshift 会报告 WARNING 而不是 ERROR。当您在数据共享中查询对象时，Amazon Redshift 不会发送 ERROR 消息。

主题

- SVV_ACTIVE_CURSORS
- SVV_ALL_COLUMNS
- SVV_ALL_SCHEMAS
- SVV_ALL_TABLES

- SVV_ALTER_TABLE_RECOMMENDATIONS
- SVV_ATTACHED_MASKING_POLICY
- SVV_COLUMNS
- SVV_COLUMN_PRIVILEGES
- SVV_DATABASE_PRIVILEGES
- SVV_DATASHARE_PRIVILEGES
- SVV_DATASHARES
- SVV_DATASHARE_CONSUMERS
- SVV_DATASHARE_OBJECTS
- SVV_DEFAULT_PRIVILEGES
- SVV_DISKUSAGE
- SVV_EXTERNAL_COLUMNS
- SVV_EXTERNAL_DATABASES
- SVV_EXTERNAL_PARTITIONS
- SVV_EXTERNAL_SCHEMAS
- SVV_EXTERNAL_TABLES
- SVV_FUNCTION_PRIVILEGES
- SVV_GEOGRAPHY_COLUMNS
- SVV_GEOMETRY_COLUMNS
- SVV_IAM_PRIVILEGES
- SVV_IDENTITY_PROVIDERS
- SVV_INTEGRATION
- SVV_INTEGRATION_TABLE_STATE
- SVV_INTERLEAVED_COLUMNS
- SVV_LANGUAGE_PRIVILEGES
- SVV_MASKING_POLICY
- SVV_ML_MODEL_INFO
- SVV_ML_MODEL_PRIVILEGES
- SVV_MV_DEPENDENCY

- SVV_MV_INFO
- SVV_QUERY_INFLIGHT
- SVV_QUERY_STATE
- SVV_REDSHIFT_COLUMNS
- SVV_REDSHIFT_DATABASES
- SVV_REDSHIFT_FUNCTIONS
- SVV_REDSHIFT_SCHEMA_QUOTA
- SVV_REDSHIFT_SCHEMAS
- SVV_REDSHIFT_TABLES
- SVV_RELATION_PRIVILEGES
- SVV_RLS_APPLIED_POLICY
- SVV_RLS_ATTACHED_POLICY
- SVV_RLS_POLICY
- SVV_RLS_RELATION
- SVV_ROLE_GRANTS
- SVV_ROLES
- SVV_SCHEMA_PRIVILEGES
- SVV_SCHEMA_QUOTA_STATE
- SVV_SYSTEM_PRIVILEGES
- SVV_TABLE_INFO
- SVV_TABLES
- SVV_TRANSACTIONS
- SVV_USER_GRANTS
- SVV_USER_INFO
- SVV_VACUUM_PROGRESS
- SVV_VACUUM_SUMMARY

SVV_ACTIVE_CURSORS

SVV_ACTIVE_CURSORS 显示当前打开的游标的详细信息。有关更多信息，请参阅 DECLARE。

SVV_ACTIVE_CURSORS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。用户只能查看自己打开的光标。超级用户可以查看所有光标。

表列

列名称	数据类型	描述
user_id	整数	创建游标的用户的 ID。
cursor_name	varchar(128)	游标的名称。
transaction_id	bigint(128)	事务的 ID。
session_id	整数	游标处于活动状态的进程的 ID。
declare_time	时间戳	声明游标的时间。
total_bytes	bigint	游标结果集的大小，以字节为单位。
total_rows	bigint	游标结果集中的行数。
fetched_rows	bigint	当前从游标结果集提取的行数。
cursor_storage_limit_used_percent	整数	游标当前使用的磁盘空间的百分比。

SVV_ALL_COLUMNS

使用 SVV_ALL_COLUMNS 可以查看来自 Amazon Redshift 表的列的联合，如 SVV_REDSHIFT_COLUMNS 中所示，以及所有外部表中所有外部列的合并列表。有关 Amazon Redshift 列的信息，请参阅SVV_REDSHIFT_COLUMNS。

SVV_ALL_COLUMNS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	varchar(128)	数据库的名称。
schema_name	varchar(128)	架构的名称。
table_name	varchar(128)	表的名称。
column_name	varchar(128)	列的名称。
ordinal_position	integer	列在表中的位置。
column_default	varchar(4000)	列的默认值。
is_nullable	varchar(3)	指示列是否可为 null 的值。可能的值为 yes 和 no。
data_type	varchar(128)	列的数据类型。
character_maximum_length	integer	列中的最大字符数。
numeric_precision	integer	数值精度。
numeric_scale	integer	小数位数。
remarks	varchar(256)	备注。

示例查询

下面的示例返回 SVV_ALL_COLUMNS 的输出。

```
SELECT *
FROM svv_all_columns
WHERE database_name = 'tickit_db'
    AND TABLE_NAME = 'tickit_sales_redshift'
ORDER BY COLUMN_NAME,
    SCHEMA_NAME
LIMIT 5;
```

```
 database_name | schema_name |     table_name        | column_name | ordinal_position
 | column_default | is_nullable | data_type | character_maximum_length |
 numeric_precision | numeric_scale | remarks
 -------------+------------+----------------------+-------------+-----------------
+----------------+-------------+-----------+--------------------------
+------------------+--------------+---------
   tickit_db   |    public   | tickit_sales_redshift |    buyerid  |        4         |
               |      NO     |  integer  |                          |         32
  |       0       |
   tickit_db   |    public   | tickit_sales_redshift | commission  |        9         |
               |     YES     |  numeric  |                          |          8
  | 2       |
   tickit_db   |    public   | tickit_sales_redshift |    dateid   |        7         |
               |      NO     |  smallint |                          |         16
  |       0       |
   tickit_db   |    public   | tickit_sales_redshift |   eventid   |        5         |
               |      NO     |  integer  |                          |         32
  |       0       |
   tickit_db   |    public   | tickit_sales_redshift |    listid   |        2         |
               |      NO     |  integer  |                          |         32
  |       0       |
```

SVV_ALL_SCHEMAS

使用 SVV_ALL_SCHEMAS 可以查看 Amazon Redshift schema 的联合，如 SVV_REDSHIFT_SCHEMAS 所示，以及所有数据库中所有外部 schema 的综合列表。有关 Amazon Redshift schema 的更多信息，请参阅SVV_REDSHIFT_SCHEMAS。

SVV_ALL_SCHEMAS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	varchar(128)	schema 所存在的数据库的名称。
schema_name	varchar(128)	架构的名称。

列名称	数据类型	描述
schema_owner	integer	schema 拥有者的用户 ID。有关用户 ID 的信息，请参阅PG_USER_INFO。
schema_type	varchar(128)	schema 的类型。可能的值包括外部、本地和共享 schema。
schema_acl	varchar(128)	为 Schema 的指定用户或用户组定义权限的字符串。
source_database	varchar(128)	外部 schema 的源数据库的名称。
schema_option	varchar(256)	schema 的选项。这是一个外部 schema 属性。

示例查询

下面的示例返回 SVV_ALL_SCHEMAS 的输出。

```
SELECT *
FROM svv_all_schemas
WHERE database_name = 'tickit_db'
ORDER BY database_name,
    SCHEMA_NAME;


 database_name |    schema_name     | schema_owner | schema_type | schema_acl |
 source_database | schema_option
---------------+--------------------+--------------+-------------+------------
+-----------------+---------------
   tickit_db   |       public       |       1      |   shared    |            |
         |
```

SVV_ALL_TABLES

使用 SVV_ALL_TABLES 可以查看 Amazon Redshift 表的联合，如 SVV_REDSHIFT_TABLES 中所示，以及所有外部 schema 中所有外部表的综合列表。有关 Amazon Redshift 表的信息，请参阅SVV_REDSHIFT_TABLES。

SVV_ALL_TABLES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	varchar(128)	表所在数据库的名称。
schema_name	varchar(128)	表 schema 的名称。
table_name	varchar(128)	表的名称。
table_acl	varchar(128)	为表的指定用户或用户组定义权限的字符串。
table_type	varchar(128)	表的类型。可能的值包括视图、基表、外部表和共享表。
remarks	varchar(256)	备注。

示例查询

以下示例返回 SVV_ALL_TABLES 的输出。

```
SELECT *
FROM svv_all_tables
WHERE database_name = 'tickit_db'
ORDER BY TABLE_NAME,
    SCHEMA_NAME
LIMIT 5;

 database_name | schema_name |        table_name        | table_type | table_acl |
 remarks
```

```
--------------+------------+-------------------------+-----------+----------
+---------
   tickit_db   |    public   | tickit_category_redshift |    TABLE   |          |
   tickit_db   |    public   |   tickit_date_redshift   |    TABLE   |          |
   tickit_db   |    public   |   tickit_event_redshift  |    TABLE   |          |
   tickit_db   |    public   | tickit_listing_redshift  |    TABLE   |          |
   tickit_db   |    public   |   tickit_sales_redshift  |    TABLE   |          |
```

如果 table_acl 值为 null，则没有显式授予针对相应表的访问权限。

SVV_ALTER_TABLE_RECOMMENDATIONS

记录当前针对表的 Amazon Redshift Advisor 建议。此视图显示针对所有表的建议，无论它们是否为自动优化而定义。要查看某张表是否定义为自动优化，请参阅SVV_TABLE_INFO。条目仅针对当前会话数据库中可见的表显示。在（Amazon Redshift 或您）应用建议后，该建议将不再显示在视图中。

SVV_ALTER_TABLE_RECOMMENDATIONS 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
type	character (30)	建议的类型。可能的值包括 distkey 和 sortkey。
数据库	character (128)	数据库名称。
table_id	integer	表标识符。
group_id	integer	一组建议的组编号。应该应用一组中的所有建议，以查看最大收益。排序键建议的可能值为 -1，对于分配键建议，可能值为大于零的数字。
ddl	character (1024)	必须运行以应用建议的 SQL 语句。
auto_eligible	character (1)	该值指示建议是否符合 Amazon Redshift 自动运行的条件。如果此值为 t，则指示为 true，如果为 f，则指示为 false。

示例查询

在以下示例中，结果中的行显示分配键和排序键的建议。这些行还会显示这些建议是否有资格让 Amazon Redshift 自动应用它们。

```
select type, database, table_id, group_id, ddl, auto_eligible
from svv_alter_table_recommendations;
```

```
 type      | database | table_id | group_id | ddl

                   | auto_eligible
diststyle | db0      | 117884   | 2        | ALTER TABLE "sch"."dp21235_tbl_1" ALTER
DISTSTYLE KEY DISTKEY "c0"
                     | f
diststyle | db0      | 117892   | 2        | ALTER TABLE "sch"."dp21235_tbl_1" ALTER
DISTSTYLE KEY DISTKEY "c0"
                     | f
diststyle | db0      | 117885   | 1        | ALTER TABLE "sch"."catalog_returns"
ALTER DISTSTYLE KEY DISTKEY "cr_sold_date_sk", ALTER COMPOUND SORTKEY
("cr_sold_date_sk","cr_returned_time_sk") | t
sortkey   | db0      | 117890   | -1       | ALTER TABLE "sch"."customer_addresses"
ALTER COMPOUND SORTKEY ("ca_address_sk")
                      | t
```

SVV_ATTACHED_MASKING_POLICY

使用 SVV_ATTACHED_MASKING_POLICY 查看所有关系和已在当前连接的数据库上附加了策略的角色/用户。

只有拥有 sys:secadmin 角色的超级用户和用户才可以查看 SVV_ATTACHED_MASKING_POLICY。常规用户将看到 0 行。

表列

列名称	数据类型	描述
policy_name	文本	已附加到表的屏蔽策略的名称。
schema_name	文本	已附加策略的表的 Schema。

列名称	数据类型	描述
table_name	文本	已附加策略的表的名称。
table_type	文本	已附加策略的表的类型。
grantor	文本	已附加策略的用户的名称。
grantee	文本	已为其附加策略的用户/角色的名称。
grantee_type	文本	被授权者的类型。这可以是角色、用户或公有。
priority	int	附加策略的优先级。
input_columns	文本	附加策略的输入列属性。
output_columns	文本	附加策略的输出列属性。

内部函数

SVV_ATTACHED_MASKING_POLICY 支持以下内部函数：

mask_get_policy_for_role_on_column

获取适用于给定列/角色对的最高优先级策略。

语法

```
mask_get_policy_for_role_on_column
                        (relschema,
                        relname,
                        colname,
                        rolename);
```

参数

relschema

策略所在 Schema 的名称。

relname

策略所在表的名称。

colname

策略所附加到的列的名称。

rolename

策略所附加到的角色的名称。

mask_get_policy_for_user_on_column

获取适用于给定列/用户对的最高优先级策略。

语法

```
mask_get_policy_for_user_on_column
                        (relschema,
                        relname,
                        colname,
                        username);
```

参数

relschema

策略所在 Schema 的名称。

relname

策略所在表的名称。

colname

策略所附加到的列的名称。

rolename

策略所附加到的用户的名称。

SVV_COLUMNS

使用 SVV_COLUMNS 查看有关本地和外部表及视图的列的目录信息，包括后期绑定视图。

SVV_COLUMNS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

SVV_COLUMNS 视图合并来自 系统目录表（带 PG 前缀的表）和 SVV_EXTERNAL_COLUMNS 系统视图的表元数据。系统目录表描述 Amazon Redshift 数据库表。SVV_EXTERNAL_COLUMNS 描述与 Amazon Redshift Spectrum 结合使用的外部表。

所有用户均可查看系统目录表中的所有行。普通用户只能从其已被授予访问权限的外部表的“SVV_EXTERNAL_COLUMNS”视图中查看列定义。虽然普通用户可以在系统目录表中查看表元数据，但如果他们拥有表或已被授予访问权限，则他们只能从用户定义的表中选择数据。

表列

列名称	数据类型	描述
table_catalog	text	表所在目录的名称。
table_schema	text	表 schema 的名称。
table_name	text	表的名称。
column_name	text	列的名称。
ordinal_position	int	列在表中的位置。
column_default	text	列的默认值。
is_nullable	text	指示列是否可为 null 的值。
data_type	text	列的数据类型。
character_maximum_length	int	列中的最大字符数。
numeric_precision	int	数值精度。如果 data_type 列是数字，则此列返回整个值中的有效位数。
numeric_precision_radix	int	数值精度的基数。如果 data_type 列是数字，则此列返回 numeric_precision 和 numeric_scale 列的基数。

列名称	数据类型	描述
numeric_scale	int	小数位数。如果 data_type 列是数字，则此列返回十进制值中的有效位数。
datetime_precision	int	日期时间的精度。
interval_type	text	间隔类型。
interval_precision	text	间隔精度。
character_set_catalog	text	字符集目录。
character_set_schema	text	字符集模式。
character_set_name	text	字符集名称。
collation_catalog	text	排序规则目录。
collation_schema	text	排序规则模式。
collation_name	text	排序规则名称。
domain_name	text	域名。
remarks	text	备注。

SVV_COLUMN_PRIVILEGES

使用 SVV_COLUMN_PRIVILEGES 查看向当前数据库中的用户、角色和组显式授予的列权限。

SVV_COLUMN_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
namespace_name	文本	指定关系所在命名空间的名称。
relation_name	文本	关系的名称。
column_name	text	列的名称。
privilege_type	文本	权限类型。可能的值为 SELECT 或 UPDATE。
identity_id	整数	身份 ID。可能的值为用户 ID、角色 ID 或组 ID。
identity_name	文本	身份的名称。
identity_type	文本	身份的类型。可能的值为用户、角色、组或公有。

示例查询

以下示例显示了 SVV_COLUMN_PRIVILEGES 的结果。

```
SELECT
 namespace_name,relation_name,COLUMN_NAME,privilege_type,identity_name,identity_type
FROM svv_column_privileges WHERE relation_name = 'lineitem';

 namespace_name | relation_name | column_name | privilege_type | identity_name |
 identity_type
----------------+---------------+-------------+----------------+---------------
+----------------
    public      |   lineitem    | l_orderkey  |     SELECT     |    reguser    |
 user
    public      |   lineitem    | l_orderkey  |     SELECT     |     role1     |
 role
    public      |   lineitem    | l_partkey   |     SELECT     |    reguser    |
 user
```

```
   public      |   lineitem    | l_partkey   |     SELECT     |     role1     |
role
```

SVV_DATABASE_PRIVILEGES

使用 SVV_DATABASE_PRIVILEGES 查看向 Amazon Redshift 集群中的用户、角色和组显式授予的数据库权限。

SVV_DATABASE_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
database_name	文本	数据库的名称。
privilege_type	文本	权限类型。可能的值为 USAGE、CREATE 或 TEMP。
identity_id	整数	身份 ID。可能的值为用户 ID、角色 ID 或组 ID。
identity_name	文本	身份的名称。
identity_type	文本	身份的类型。可能的值为用户、角色、组或公有。
admin_option	布尔值	一个指示用户是否可以向其他用户和角色授予权限的值。对于角色和组身份类型，它始终为 false。

示例查询

以下示例显示了 SVV_DATABASE_PRIVILEGES 的结果。

```
SELECT database_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_database_privileges
WHERE database_name = 'test_db';

 database_name | privilege_type | identity_name | identity_type | admin_option
---------------+----------------+---------------+---------------+--------------
     test_db   |     CREATE     |     reguser   |      user     |     False
     test_db   |     CREATE     |      role1    |      role     |     False
     test_db   |     TEMP       |      public   |      public   |     False
     test_db   |     TEMP       |      role1    |      role     |     False
```

SVV_DATASHARE_PRIVILEGES

使用 SVV_DATASHARE_PRIVILEGES 查看向 Amazon Redshift 集群中的用户、角色和组显式授予的数据共享权限。

SVV_DATASHARE_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
datashare_name	文本	数据共享的名称。
privilege_type	文本	权限类型。可能的值为 ALTER 或 SHARE。
identity_id	整数	身份 ID。可能的值为用户 ID、角色 ID 或组 ID。
identity_name	文本	身份的名称。

列名称	数据类型	描述
identity_type	文本	身份的类型。可能的值为用户、角色、组或公有。
admin_option	布尔值	一个指示用户是否可以向其他用户和角色授予权限的值。对于角色和组身份类型，它始终为 false。

示例查询

以下示例显示 SVV_DATASHARE_PRIVILEGES 的结果。

```
SELECT datashare_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_datashare_privileges
WHERE datashare_name = 'demo_share';

 datashare_name | privilege_type |  identity_name | identity_type | admin_option
----------------+----------------+----------------+---------------+--------------
   demo_share   |     ALTER      |    superuser   |     user      |   False
   demo_share   |     ALTER      |    reguser     |     user      |   False
```

SVV_DATASHARES

使用 SVV_DATASHARES 可以查看在集群上创建的数据共享的列表，以及与集群共享的数据共享。

SVV_DATASHARES 对以下用户可见：

- 超级用户
- 数据共享拥有者
- 对数据共享拥有 ALTER 或 USAGE 权限的用户

其他用户无法看到任何行。有关 ALTER 和 USAGE 权限的信息，请参阅GRANT。

表列

列名称	数据类型	描述
share_name	varchar(128)	数据共享的名称。

列名称	数据类型	描述
share_id	integer	数据共享的 ID。
share_owner	integer	数据共享的拥有者。
source_database	varchar(128)	此数据共享的源数据库。
consumer_database	varchar(128)	从此数据共享创建的使用者数据库。
share_type	varchar(8)	数据共享的类型。可能的值包括 INBOUND 和 OUTBOUND。
createdate	不带时区的时间戳	创建数据共享的日期。
is_publicaccessible	布尔值	指定是否可以将数据共享共享给可公开访问的集群的属性。
share_acl	varchar(256)	为数据共享的指定用户或用户组定义权限的字符串。
producer_account	varchar(16)	数据共享创建者账户的 ID。
producer_namespace	varchar(64)	数据共享创建者集群的唯一集群标识符。
managed_by	varchar(64)	用于指定管理数据共享的 AWS 服务。

使用说明

检索其他元数据：使用 `share_owner` 列中返回的整数时，您可以与 SVL_USER_INFO 中的 `usesysid` 进行联接，以获取有关数据共享所有者的数据。该数据包括名称和其他属性。

示例查询

以下示例返回 SVV_DATASHARES 的输出。

```
SELECT share_owner, source_database, share_type, is_publicaccessible
FROM svv_datashares
WHERE share_name LIKE 'tickit_datashare%'
AND source_database = 'dev';

  share_owner | source_database | share_type  | is_publicaccessible
--------------+-----------------+-------------+---------------------
     100      |      dev        |   OUTBOUND  |        True
(1 rows)
```

以下示例返回出站数据共享的 SVV_DATASHARES 的输出。

```
SELECT share_name, share_owner, btrim(source_database), btrim(consumer_database),
 share_type, is_publicaccessible, share_acl, btrim(producer_account),
 btrim(producer_namespace), btrim(managed_by) FROM svv_datashares WHERE share_type =
 'OUTBOUND';

   share_name   | share_owner | source_database | consumer_database | share_type |
 is_publicaccessible | share_acl | producer_account|         producer_namespace
   | managed_by
----------------+-------------+-----------------+-------------------
+------------+---------------------+-----------+-----------------
+--------------------------------------+------------
    salesshare  |      1      |       dev       |                   |  OUTBOUND  |
   True         |           |   123456789012  | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |

 marketingshare |      1      |       dev       |                   |  OUTBOUND  |
   True         |           |   123456789012  | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
```

以下示例返回入站数据共享的 SVV_DATASHARES 的输出。

```
SELECT share_name, share_owner, btrim(source_database), btrim(consumer_database),
 share_type, is_publicaccessible, share_acl, btrim(producer_account),
 btrim(producer_namespace), btrim(managed_by) FROM svv_datashares WHERE share_type =
 'INBOUND';

  share_name    | share_owner | source_database | consumer_database | share_type |
 is_publicaccessible | share_acl | producer_account |         producer_namespace
    | managed_by
```

```
---------------+-------------+-----------------+-------------------
+------------+--------------------+-----------+-----------------
+--------------------------------------+------------
  salesshare    |             |                 |                    |  INBOUND   |
   False         |           |  123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
 |
 marketingshare |             |                 |                    |  INBOUND   |
  False         |           |  123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
 ADX
```

SVV_DATASHARE_CONSUMERS

使用 SVV_DATASHARE_CONSUMERS 可以查看在集群上创建的数据共享的使用者列表。

SVV_DATASHARE_CONSUMERS 对以下用户可见：

- 超级用户
- 数据共享拥有者
- 对数据共享拥有 ALTER 或 USAGE 权限的用户

其他用户无法看到任何行。有关 ALTER 和 USAGE 权限的信息，请参阅GRANT。

表列

列名称	数据类型	描述
share_name	varchar(128)	数据共享的名称。
consumer_account	varchar(16)	数据共享使用者的账户 ID。
consumer_namespace	varchar(64)	数据共享使用者集群的唯一集群标识符。
share_date	不带时区的时间戳	共享数据共享的日期。

示例查询

以下示例返回 SVV_DATASHARE_CONSUMERS 的输出。

```
SELECT count(*)
FROM svv_datashare_consumers
WHERE share_name LIKE 'tickit_datashare%';

1
```

SVV_DATASHARE_OBJECTS

使用 SVV_DATASHARE_OBJECTS 可以查看在集群上创建或与集群共享的所有数据共享中的对象列表。

SVV_DATASHARE_OBJECTS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

有关查看数据共享列表的信息，请参阅 SVV_DATASHARES。

表列

列名称	数据类型	描述
share_type	varchar(8)	指定数据共享的类型。可能的值包括 OUTBOUND 和 INBOUND。
share_name	varchar(128)	数据共享的名称。
object_type	varchar(64)	指定对象的类型。可能的值包括 schema、表、视图、后期绑定视图、实体化视图和函数。
object_name	varchar(512)	对象的名称。对象名称扩展为包含 schema 名称，如 schema1.t1。
producer_account	varchar(16)	数据共享创建者账户的 ID。
producer_namespace	varchar(64)	数据共享创建者集群的唯一集群标识符。

列名称	数据类型	描述
include_new	布尔值	指定是否将在指定 schema 中创建的任何未来表、视图或 SQL 用户定义函数 (UDF) 添加到数据共享中的属性。此参数仅与 OUTBOUND 数据共享相关，并且仅适用于数据共享中的 schema 类型。

示例查询

以下示例返回 SVV_DATASHARE_OBJECTS 的输出。

```
SELECT share_type,
    btrim(share_name)::varchar(16) AS share_name,
    object_type,
    object_name
FROM svv_datashare_objects
WHERE share_name LIKE 'tickit_datashare%'
AND object_name LIKE '%tickit%'
ORDER BY object_name
LIMIT 5;

 share_type |     share_name     | object_type |          object_name
------------+--------------------+-------------+--------------------------------
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_category_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_date_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_event_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_listing_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_sales_redshift
```

```
SELECT * FROM SVV_DATASHARE_OBJECTS WHERE share_name like 'sales%';

share_type | share_name | object_type | object_name  | producer_account |
 producer_namespace          | include_new
-----------+------------+-------------+--------------+------------------
+--------------------------------------+-------------
 OUTBOUND  | salesshare | schema      | public       | 123456789012     |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |      t
```

```
OUTBOUND  | salesshare | table       | public.sales | 123456789012     |
13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
```

SVV_DEFAULT_PRIVILEGES

使用 SVV_DEFAULT_PRIVILEGES 可查看用户在 Amazon Redshift 集群中有权访问的默认权限。

SVV_DEFAULT_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们获得的默认权限。

表列

列名称	数据类型	描述
schema_name	文本	架构的名称。
object_type	文本	对象类型。可能的值为 RELATION、FUNCTION 或 PROCEDURE。
owner_id	整数	所有者 ID。可能的值是用户 ID。
owner_name	文本	所有者的名称。
owner_type	文本	所有者类型。可能的值是用户。
privilege_type	文本	权限类型。可能的值为 INSERT、SELECT、UPDATE、DELETE、RULE、REFERENCES TRIGGER、DROP 和 EXECUTE。
grantee_id	整数	被授权者 ID。可能的值为用户 ID、角色 ID 或组 ID。
grantee_type	文本	被授权者类型。可能的值为用户、角色或公有。
admin_option	布尔值	指示用户是否可以向其他用户和角色授予权限的值。对于角色和组类型，它始终为 false。

示例查询

以下示例返回 SVV_DEFAULT_PRIVILEGES 的输出。

```
SELECT * from svv_default_privileges;

 schema_name |    object_type    | owner_id | owner_name | owner_type | privilege_type
 | grantee_id | grantee_name | grantee_type | admin_option
-------------+------------------+--------- +------------+------------+----------------
+------------+--------------+--------------+-------------+
   public    |     RELATION      |    106   |     u1     |    user    |     UPDATE
 |     107    |      u2      |     user     |      f      |
   public    |     RELATION      |    106   |     u1     |    user    |     SELECT
 |     107    |      u2      |     user     |      f      |
```

SVV_DISKUSAGE

Amazon Redshift 通过联接 STV_TBL_PERM 和 STV_BLOCKLIST 表来创建 SVV_DISKUSAGE 系统视图。SVV_DISKUSAGE 视图包含数据库中表的数据分配的相关信息。

针对 SVV_DISKUSAGE 使用聚合查询（如以下示例所示）可确定为每个数据库、表、切片或列分配的磁盘数据块的数目。每个数据块使用 1 MB。您还可以使用 STV_PARTITIONS 查看有关磁盘利用率的摘要信息。

SVV_DISKUSAGE 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> 此视图仅在查询预置集群时可用。

表列

列名称	数据类型	描述
db_id	integer	数据库 ID。

列名称	数据类型	描述
名称	character (72)	表名称。
slice	integer	分配到表的数据切片。
col	integer	列的零基索引。您创建的每个表都附加有以下三个隐藏列：INSERT_XID、DELETE_XID 和 ROW_ID (OID)。包含 3 个用户定义的列的表实际上包含 6 列，用户定义的列在内部编号为 0、1 和 2。在此示例中，INSERT_XID、DELETE_XID 和 ROW_ID 列分别编号为 3、4 和 5。
tbl	integer	表 ID。
blocknum	integer	数据块的 ID。
num_values	integer	数据块上包含的值的数目。
minvalue	bigint	数据块上包含的最小值。
maxvalue	bigint	数据块上包含的最大值。
sb_pos	integer	磁盘上超级数据块位置的内部标识符。
pinned	integer	是否在预加载时将数据块固定到内存中。0 = false；1 = true。默认设置为“假”。
on_disk	integer	是否自动在磁盘上存储数据块。0 = false；1 = true。默认设置为“假”。
modified	integer	是否已修改数据块。0 = false；1 = true。默认设置为“false”。
hdr_modified	integer	是否已修改数据块标头。0 = false；1 = true。默认设置为“假”。
unsorted	integer	数据块是否未排序。0 = false；1 = true。默认为“真”。
tombstone	integer	供内部使用。
preferred_diskno	integer	数据块应该位于的磁盘的编号（除非磁盘已出故障）。磁盘一旦修复，该数据块就将移回到该磁盘。

列名称	数据类型	描述
temporary	integer	数据块是否包含临时表或中间查询结果等位置的临时数据。0 = false；1 = true。默认设置为“假”。
newblock	integer	指示数据块是新数据块（真）还是以前从未提交到过磁盘（假）。0 = 假；1 = 真。

示例查询

分配的每个磁盘数据块在 SVV_DISKUSAGE 中对应一行，因此选择所有行的查询可能会返回非常多的行。建议仅对 SVV_DISKUSAGE 使用聚合查询。

返回曾经分配到 USERS 表中第 6 列（EMAIL 列）的最大数据块数。

```
select db_id, trim(name) as tablename, max(blocknum)
from svv_diskusage
where name='users' and col=6
group by db_id, name;

db_id  | tablename | max
--------+-----------+-----
175857 | users     |   2
(1 row)
```

对于名为 SALESNEW 的 10 列大型表中的所有列，下面的查询返回相似的结果。（列 10 到 12 的最后三行用于隐藏元数据列。）

```
select db_id, trim(name) as tablename, col, tbl, max(blocknum)
from svv_diskusage
where name='salesnew'
group by db_id, name, col, tbl
order by db_id, name, col, tbl;

db_id  | tablename  | col |  tbl   | max
--------+------------+-----+--------+-----
175857 | salesnew   |   0 | 187605 | 154
175857 | salesnew   |   1 | 187605 | 154
175857 | salesnew   |   2 | 187605 | 154
```

```
175857 | salesnew   |   3 | 187605 | 154
175857 | salesnew   |   4 | 187605 | 154
175857 | salesnew   |   5 | 187605 |  79
175857 | salesnew   |   6 | 187605 |  79
175857 | salesnew   |   7 | 187605 | 302
175857 | salesnew   |   8 | 187605 | 302
175857 | salesnew   |   9 | 187605 | 302
175857 | salesnew   |  10 | 187605 |   3
175857 | salesnew   |  11 | 187605 |   2
175857 | salesnew   |  12 | 187605 | 296
(13 rows)
```

SVV_EXTERNAL_COLUMNS

使用 SVV_EXTERNAL_COLUMNS 可查看外部表中列的详细信息。对于跨数据库查询，也可以使用 SVV_EXTERNAL_COLUMNS 来查看用户有权访问的未连接数据库上的表中所有列的详细信息。

SVV_EXTERNAL_COLUMNS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
redshift_database_name	文本	本地 Amazon Redshift 数据库的名称。
schemaname	text	外部表的 Amazon Redshift 外部 schema 的名称。
tablename	text	外部表的名称。
columnname	text	列的名称。
external_type	text	列的数据类型。
columnnum	integer	外部列编号，从 1 开始。
part_key	integer	如果列是分区键，则此值为键的顺序。如果列不是分区，则此值为 **0**。

列名称	数据类型	描述
is_nullable	text	定义列是否可为 null。某些值为 `true`、`false` 或不表示任何信息的 "" 空字符串。

SVV_EXTERNAL_DATABASES

使用 SVV_EXTERNAL_DATABASES 可查看外部数据库的详细信息。

SVV_EXTERNAL_DATABASES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
eskind	integer	数据库的外部目录的类型；**1** 表示数据目录，**2** 表示 Hive 元存储。
esoptions	text	数据库所在目录的详细信息。
databasename	text	外部目录中数据库的名称。
位置	text	数据库的位置。
parameters	text	数据库参数。

SVV_EXTERNAL_PARTITIONS

使用 SVV_EXTERNAL_PARTITIONS 可查看外部表中分区的详细信息。

SVV_EXTERNAL_PARTITIONS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
schemaname	text	具有指定分区的外部表的 Amazon Redshift 外部 schema 的名称。
tablename	text	外部表的名称。
values	text	分区的值。
位置	text	分区的位置。列大小限制为 128 个字符。较长的值将被截断。
input_format	text	输入格式。
output_format	text	输出格式。
serialization_lib	text	序列化库。
serde_parameters	text	SerDe 参数。
compressed	integer	用于指示分区是否已压缩的值；**1** 表示已压缩，**0** 表示未压缩。
parameters	text	分区属性。

SVV_EXTERNAL_SCHEMAS

使用 SVV_EXTERNAL_SCHEMAS 可查看有关外部 schema 的信息。有关更多信息，请参阅 CREATE EXTERNAL SCHEMA。

SVV_EXTERNAL_SCHEMAS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
esoid	oid	外部 schema ID。

列名称	数据类型	描述
eskind	smallint	外部模式的外部目录的类型：1 表示数据目录，2 表示 Hive 元存储，3 表示对 Aurora PostgreSQL 或 Amazon RDS PostgreSQL 的联合查询，4 表示本地 Amazon Redshift 数据库的模式，5 表示远程 Amazon Redshift 数据库的模式，6 表示系统表的模式，8 表示远程 MySQL 数据库的模式，9 表示 Amazon Kinesis 数据流的模式，10 表示 Amazon Managed Streaming for Apache Kafka 数据流。
schemaname	名称	外部 schema 名称。
esowner	integer	外部 schema 所有者的用户 ID。
databasename	text	外部数据库名称。
esoptions	text	外部 schema 选项。

示例

以下示例显示有关外部 schema 的详细信息。

```
select * from svv_external_schemas;

esoid  | eskind | schemaname | esowner | databasename | esoptions

-------+--------+------------+---------+--------------
+-------------------------------------------------------------
100133 |      1 | spectrum   |     100 | redshift     |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

SVV_EXTERNAL_TABLES

使用 SVV_EXTERNAL_TABLES 可查看外部表的详细信息；有关更多信息，请参阅CREATE EXTERNAL SCHEMA。对于跨数据库查询，也可以使用 SVV_EXTERNAL_TABLES 来查看用户有权访问的未连接数据库上的所有表的元数据。

SVV_EXTERNAL_TABLES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
redshift_database_name	文本	本地 Amazon Redshift 数据库的名称。
schemaname	text	外部表的 Amazon Redshift 外部 schema 的名称。
tablename	text	外部表的名称。
tabletype	text	表的类型。某些值为 TABLE、VIEW、MATERIALIZED VIEW 或不表示任何信息的 "" 空字符串。
位置	text	表的位置。
input_format	text	输入格式
output_format	text	输出格式。
serialization_lib	text	序列化库。
serde_parameters	text	SerDe 参数。
compressed	integer	用于指示表是否已压缩的值；**1** 表示已压缩，**0** 表示未压缩。
parameters	text	表属性。

示例

以下示例显示 svv_external_tables 详细信息以及联合查询使用的外部 schema 上的谓词。

```
select schemaname, tablename from svv_external_tables where schemaname = 'apg_tpch';
```

```
schemaname  | tablename
------------+-----------
apg_tpch    | customer
apg_tpch    | lineitem
apg_tpch    | nation
apg_tpch    | orders
apg_tpch    | part
apg_tpch    | partsupp
apg_tpch    | region
apg_tpch    | supplier
(8 rows)
```

SVV_FUNCTION_PRIVILEGES

使用 SVV_FUNCTION_PRIVILEGES 查看向当前数据库中的用户、角色和组显式授予的函数权限。

SVV_FUNCTION_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
namespace_name	文本	指定函数所在命名空间的名称。
function_name	文本	函数的名称。
argument_types	文本	表示函数输入参数类型的字符串。
privilege_type	文本	权限类型。可能的值为 EXECUTE。
identity_id	整数	身份 ID。可能的值为用户 ID、角色 ID 或组 ID。

列名称	数据类型	描述
identity_name	文本	身份的名称。
identity_type	文本	身份的类型。可能的值为用户、角色、组或公有。
admin_option	布尔值	一个指示用户是否可以向其他用户和角色授予权限的值。对于角色和组身份类型，它始终为 false。

示例查询

以下示例显示了 SVV_FUNCTION_PRIVILEGES 的结果。

```
SELECT
 namespace_name,function_name,argument_types,privilege_type,identity_name,identity_type,admin_o
 FROM svv_function_privileges
WHERE identity_name IN ('role1', 'reguser');

 namespace_name | function_name |       argument_types       | privilege_type |
 identity_name | identity_type | admin_option
----------------+---------------+----------------------------+----------------
+----------------+---------------+-------------
    public      | test_func1    | integer                    |    EXECUTE     |
 role1     |     role      |  False
    public      | test_func2    | integer, character varying |    EXECUTE     |
 reguser    |     user      |  False
```

SVV_GEOGRAPHY_COLUMNS

使用 SVV_GEOGRAPHY_COLUMNS 查看数据仓库中 GEOGRAPHY 列的列表。该列表包括来自数据共享的列。

SVV_GEOGRAPHY_COLUMNS 对所有用户均可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
f_table_catalog	varchar(128)	包含 GEOGRAPHY 列的表所在数据库的名称。
f_table_schema	varchar(128)	包含 GEOGRAPHY 列的表所在架构的名称。
f_table_name	varchar(128)	GEOGRAPHY 列所在的表的名称。
f_geography_column	varchar(128)	GEOGRAPHY 列的名称。
coord_dimension	整数	GEOGRAPHY 数据的维度数。
srid	整数	GEOGRAPHY 数据的空间参考系统标识符 (SRID)。
type	varchar(128)	空间地理数据类型名称。

示例查询

以下示例显示了 SVV_GEOGRAPHY_COLUMNS 的结果。

```
SELECT * FROM svv_geography_columns;

f_table_catalog  | f_table_schema  | f_table_name  | f_geography_column  |
 coord_dimension | srid |  type
-----------------+-----------------+---------------+---------------------
+-----------------+------+-------------
dev              | public          | spatial_test  | test_geography      | 2
    | 0    | GEOGRAPHY
```

SVV_GEOMETRY_COLUMNS

使用 SVV_GEOMETRY_COLUMNS 查看数据仓库中 GEOMETRY 列的列表。该列表包括来自数据共享的列。

SVV_GEOMETRY_COLUMNS 对所有用户均可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
f_table_catalog	varchar(128)	包含 GEOMETRY 列的表所在数据库的名称。
f_table_schema	varchar(128)	包含 GEOMETRY 列的表所在架构的名称。
f_table_name	varchar(128)	GEOMETRY 列所在的表的名称。
f_geometry_column	varchar(128)	GEOMETRY 列的名称。
coord_dimension	整数	GEOMETRY 数据的维度数。
srid	整数	GEOMETRY 列的空间参考系统标识符 (SRID)。
type	varchar(128)	空间几何类型名称。

示例查询

以下示例显示了 SVV_GEOMETRY_COLUMNS 的结果。

```
SELECT * FROM svv_geometry_columns;

f_table_catalog  | f_table_schema  | f_table_name  | f_geometry_column   |
 coord_dimension | srid |  type
```

```
-----------------+-----------------+--------------+--------------------
+-----------------+------+-------------
dev               | public          | accomodations | shape               | 2
    | 0    | GEOMETRY
dev               | public          | zipcode       | wkb_geometry        | 2
    | 0    | GEOMETRY
```

SVV_IAM_PRIVILEGES

使用 SVV_IAM_PRIVILEGES 查看向用户、角色和组显式授予的 IAM 权限。

SVV_IAM_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问的条目。

表列

列名称	数据类型	描述
iam_arn	文本	命名空间的名称。
commandtype	文本	权限类型。可能的值为 COPY、UNLOAD、CREATE MODEL 或 EXTERNAL FUNCTION。
identity_id	整数	身份 ID。可能的值为用户 ID、角色 ID 或组 ID。
identity_name	文本	身份名称。
identity_type	文本	身份类型。可能的值为用户、角色、组或公有。

示例查询

以下示例显示了 SVV_IAM_PRIVILEGES 的结果。

```
SELECT * from SVV_IAM_PRIVILEGES ORDER BY IDENTITY_ID;
       iam_arn        | command_type | identity_id | identity_name | identity_type
----------------------+--------------+-------------+---------------+---------------
 default-aws-iam-role | COPY         |           0 | public        | public
 default-aws-iam-role | UNLOAD       |           0 | public        | public
 default-aws-iam-role | CREATE MODEL |           0 | public        | public
 default-aws-iam-role | EXFUNC       |           0 | public        | public
 default-aws-iam-role | COPY         |         106 | u1            | user
 default-aws-iam-role | UNLOAD       |         106 | u1            | user
 default-aws-iam-role | CREATE MODEL |         106 | u1            | user
 default-aws-iam-role | EXFUNC       |         106 | u1            | user
 default-aws-iam-role | COPY         |      118413 | r1            | role
 default-aws-iam-role | UNLOAD       |      118413 | r1            | role
 default-aws-iam-role | CREATE MODEL |      118413 | r1            | role
 default-aws-iam-role | EXFUNC       |      118413 | r1            | role
(12 rows)
```

SVV_IDENTITY_PROVIDERS

SVV_IDENTITY_PROVIDERS 视图返回身份提供商的名称和其他属性。有关如何创建身份提供商角色的更多信息，请参阅 CREATE IDENTITY PROVIDER。

SVV_IDENTITY_PROVIDERS 只对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
uid	整数	已注册的身份提供商的唯一 ID。
name	文本	身份提供商名称。
type	文本	身份提供商类型。

列名称	数据类型	描述
instanceid	文本	相同类型的实例之间的唯一区分元素。
namespc	文本	身份提供商的命名空间前缀。
params	文本	带有身份提供商参数的 JSON 对象。
已启用	布尔	指示身份提供商是否已启用。

示例查询

要查看身份提供商属性，请在创建身份提供商之后，运行类似以下内容的查询。

```
SELECT name, type, instanceid, namespc, params, enabled
FROM svv_identity_providers
ORDER BY 1;
```

示例输出包括参数描述。

```
       name       | type  |              instanceid              | namespc |

                                     params

                     | enabled
------------------+-------+--------------------------------------+---------
+-----------------------------------------------------------------------------------------------------------------
+---------
 rs5517_azure_idp | azure | e40d4bb2-7670-44ae-bfb8-5db013221d73 | abc     |
 {"issuer":"https://login.microsoftonline.com/e40d4bb2-7670-44ae-bfb8-5db013221d73/
v2.0", "client_id":"871c010f-5e61-4fb1-83ac-98610a7e9110", "client_secret":,
 "audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift", "https://
analysis.windows.net/powerbi/connector/AWSRDS"]} | t
(1 row)
```

SVV_INTEGRATION

SVV_INTEGRATION 显示有关集成配置的详细信息。

SVV_INTEGRATION 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

有关零 ETL 集成的信息，请参阅使用零 ETL 集成。

表列

列名称	数据类型	描述
integration_id	character (128)	与集成关联的标识符。
target_database	character (128)	Amazon Redshift 中接收集成数据的数据库。
source	character (128)	集成的源数据。可能的类型包括 MySQL 和 PostgreSQL 。
state	character (128)	集成的状态。可能的值包括 PendingDbConnectState 、SchemaDiscoveryState 、CdcRefreshState 和 ErrorState 。
current_lag	bigint	集成的源和目标之间的当前延迟时间（毫秒）。
last_replicated_checkpoint	character (128)	复制的最后一个检查点。
total_tables_replicated	整数	当前处于已复制状态的表总数。
total_tables_failed	整数	当前处于失败状态的表总数。
creation_time	时间戳	创建集成的时间（UTC）。它定义为通过集成创建目标数据库的时间。

示例查询

以下 SQL 命令显示当前定义的集成。

```
select * from svv_integration;
```

```
           integration_id              | target_database | source |      state
   | current_lag |      last_replicated_checkpoint     | total_tables_replicated |
 total_tables_failed |       creation_time
-------------------------------------+-----------------+--------+----------------
+-------------+--------------------------------------+-------------------------
+---------------------+---------------------------
  99108e72-1cfd-414f-8cc0-0216acefac77 |     perfdb      |  MySQL | CdcRefreshState |
  56606106  | {"txn_seq":9834,"txn_id":126597515} |            152          |
 0         | 2023-09-19 21:05:27.520299
```

SVV_INTEGRATION_TABLE_STATE

SVV_INTEGRATION_TABLE_STATE 显示有关表级集成信息的详情。

SVV_INTEGRATION_TABLE_STATE 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

有关更多信息，请参阅使用零 ETL 集成。

表列

列名称	数据类型	描述
integration_id	character(128)	与集成关联的标识符。
target_database	character(128)	Amazon Redshift 数据库的名称。
schema_name	character(128)	Amazon Redshift 架构的名称。
table_name	character(128)	表的名称。
table_state	character(128)	表的状态。可能的值为 Synced、Failed、Deleted、ResyncRequired 和 ResyncInitiated 。
table_last_replicated_checkpoint	character(128)	当前同步的日志坐标。
reason	character(256)	最后一次状态转换的原因。常见原因可能是表中存在不支持的数据类型，表没有主键。要了解有关如何排查常见问题的更多信息，请参

列名称	数据类型	描述
		阅 Troubleshooting zero-ETL integrations in Amazon Redshift。
last_updated_timestamp	不带时区的时间戳	表最后一次更新的时间 (UTC)。

示例查询

以下 SQL 命令显示集成日志。

```
select * from svv_integration_table_state;

          integration_id              | target_database | schema_name |     table_name
   | Table_state  |table_last_replicated_checkpoint | reason | last_updated_timestamp
--------------------------------------+-----------------+-------------
+------------------+-------------+--------------------------------
+-------------------------------------
 4798e675-8f9f-4686-b05f-92c538e19629 |  sample_test2   |    sample   |
 SampleTestChannel |    Synced    |   {"txn_seq":3,"txn_id":3122}   |        |
 2023-05-12 12:40:30.656625
```

SVV_INTERLEAVED_COLUMNS

使用 SVV_INTERLEAVED_COLUMNS 视图可帮助确定使用交错排序键的表是否应使用 VACUUM REINDEX 重建索引。有关如何确定运行 VACUUM 的频度和运行 VACUUM REINDEX 的时机的更多信息，请参阅管理 vacuum 操作次数。

SVV_INTERLEAVED_COLUMNS 只对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
tbl	integer	表 ID。

列名称	数据类型	描述
col	integer	列的零基索引。
interleaved_skew	numeric 9,2)	表示表的交错排序键列中存在的偏斜量的比率。值 1.00 表示无偏斜，值越大表示偏斜越大。具有较大偏斜的表应通过 VACUUM REINDEX 命令重建索引。
last_reindex	timestar	上次为指定的表运行 VACUUM REINDEX 的时间。如果表从未重新编制索引，或者自上次重建索引后基础系统日志表 STL_VACUUM 已轮换，则此值为 NULL。

示例查询

要找出可能需要重建索引的表，请运行下面的查询。

```
select tbl as tbl_id, stv_tbl_perm.name as table_name,
col, interleaved_skew, last_reindex
from svv_interleaved_columns, stv_tbl_perm
where svv_interleaved_columns.tbl = stv_tbl_perm.id
and interleaved_skew is not null;

 tbl_id | table_name | col | interleaved_skew | last_reindex
--------+------------+-----+------------------+---------------------
 100068 | lineorder  |   0 |             3.65 | 2015-04-22 22:05:45
 100068 | lineorder  |   1 |             2.65 | 2015-04-22 22:05:45
 100072 | customer   |   0 |             1.65 | 2015-04-22 22:05:45
 100072 | lineorder  |   1 |             1.00 | 2015-04-22 22:05:45
(4 rows)
```

SVV_LANGUAGE_PRIVILEGES

使用 SVV_LANGUAGE_PRIVILEGES 查看向当前数据库中的用户、角色和组显式授予的语言权限。

SVV_LANGUAGE_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
language_name	文本	语言的名称。
privilege_type	文本	权限类型。可能的值是 USAGE。
identity_id	整数	身份 ID。可能的值为用户 ID、角色 ID 或组 ID。
identity_name	文本	身份的名称。
identity_type	文本	身份的类型。可能的值为用户、角色、组或公有。
admin_option	布尔值	一个指示用户是否可以向其他用户和角色授予权限的值。对于角色和组身份类型，它始终为 false。

示例查询

以下示例显示了 SVV_LANGUAGE_PRIVILEGES 的结果。

```
SELECT language_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_language_privileges
WHERE identity_name IN ('role1', 'reguser');

 language_name | privilege_type | identity_name | identity_type | admin_option
---------------+----------------+---------------+---------------+--------------
   exfunc      |     USAGE      |    reguser    |     user      |    False
   exfunc      |     USAGE      |     role1     |     role      |    False
   plpythonu   |     USAGE      |    reguser    |     user      |    False
```

SVV_MASKING_POLICY

使用 SVV_MASKING_POLICY 查看在集群上创建的所有屏蔽策略。

只有拥有 sys:secadmin 角色的超级用户和用户才可以查看 SVV_MASKING_POLICY。常规用户将看到 0 行。

表列

列名称	数据类型	描述
policy_database	文本	在其中创建了屏蔽策略的数据库的名称。
policy_name	文本	屏蔽策略的名称。
input_columns	文本	CREATE POLICY 语句的 WITH 子句中提供的属性。
policy_expression	文本	策略中使用的屏蔽表达式。
policy_modified_by	文本	上次修改策略的用户的名称。
policy_modified_time	时间戳	创建或上次修改策略时的时间戳。

SVV_ML_MODEL_INFO

有关机器学习模型当前状态的状态信息。

SVV_ML_MODEL_INFO 对所有用户均可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	char(128)	模型的数据库。
schema_name	char(128)	模型的架构。
user_name	char(128)	模型的拥有者。
model_name	char(128)	模型的名称。
life_cycle	char(20)	模型的生命周期状态。

列名称	数据类型	描述
is_refreshable	integer	如果训练查询中的原始表和列仍然存在，并且用户仍具有对它们的权限，则模型的状态为是否可刷新。可能的值为：1（可刷新）和 0（不可刷新）。
model_state	char(128)	模型的当前状态。

示例查询

以下查询显示机器学习模型的当前状态。

```
SELECT schema_name, model_name, model_state
FROM svv_ml_model_info;

 schema_name |        model_name             |             model_state
-------------+-------------------------------+-----------------------------------
 public      | customer_churn_auto_model     | Train Model On SageMaker In Progress
 public      | customer_churn_xgboost_model | Model is Ready
(2 row)
```

SVV_ML_MODEL_PRIVILEGES

使用 SVV_ML_MODEL_PRIVILEGES 查看向集群中的用户、角色和组显式授予的机器学习模型权限。

SVV_ML_MODEL_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
namespace_name	文本	指定机器学习模型所在命名空间的名称。
model_name	文本	机器学习模型的名称。
model_version	整数	模型的版本号。
privilege_type	文本	权限类型。可能的值为 EXECUTE。
identity_id	整数	身份 ID。可能的值为用户 ID、角色 ID 或组 ID。
identity_name	文本	身份的名称。
identity_type	文本	身份的类型。可能的值为用户、角色、组或公有。
admin_option	布尔值	一个指示用户是否可以向其他用户和角色授予权限的值。对于角色和组身份类型，它始终为 false。

示例查询

以下示例显示了 SVV_ML_MODEL_PRIVILEGES 的结果。

```
SELECT
 namespace_name,model_name,model_version,privilege_type,identity_name,identity_type,admin_optio
 FROM svv_ml_model_privileges
WHERE model_name = 'test_model';

 namespace_name | model_name | model_version | privilege_type |  identity_name |
 identity_type | admin_option
----------------+------------+---------------+----------------+----------------
+---------------+--------------
      public    | test_model |       1       |    EXECUTE     |     reguser    |
 user      |    False
```

```
      public    | test_model |       1       |    EXECUTE     |     role1      |
 role      |    False
```

SVV_MV_DEPENDENCY

SVV_MV_DEPENDENCY 表显示了实体化视图与 Amazon Redshift 内其他实体化视图的依赖关系。

有关实体化视图的更多信息，请参阅在 Amazon Redshift 中创建实体化视图。

SVV_MV_DEPENDENCY 对所有用户均可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	char(128)	包含特定具体化视图的数据库。
schema_name	char(128)	具体化视图的架构。
名称	char(128)	实体化视图的名称。
dependent_database_name	char(128)	此实体化视图所依赖的实体化视图数据库。
dependent_schema_name	char(128)	此具体化视图所依赖的具体化视图 schema。
dependent_name	char(128)	此具体化视图所依赖的具体化视图的名称。

示例查询

以下查询返回一个输出行，该行指示具体化视图 mv_over_foo 在其定义中使用具体化视图 mv_foo 作为二依赖项。

```
CREATE SCHEMA test_ivm_setup;
CREATE TABLE test_ivm_setup.foo(a INT);
CREATE MATERIALIZED VIEW test_ivm_setup.mv_foo AS SELECT * FROM test_ivm_setup.foo;
```

```
CREATE MATERIALIZED VIEW test_ivm_setup.mv_over_foo AS SELECT * FROM
 test_ivm_setup.mv_foo;

SELECT * FROM svv_mv_dependency;

 database_name | schema_name          | name        | dependent_database_name |
 dependent_schema_name     | dependent_name
---------------+----------------------+-------------+-------------------------
+---------------------------+----------
 dev           | test_ivm_setup       | mv_over_foo |                     dev |
 test_ivm_setup            | mv_foo
```

SVV_MV_INFO

SVV_MV_INFO 表针对每个实体化视图、数据是否陈旧以及状态信息包含一行。

有关实体化视图的更多信息，请参阅在 Amazon Redshift 中创建实体化视图。

SVV_MV_INFO 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	char(128)	包含实体化视图的数据库。
schema_name	char(128)	数据库的架构。
user_name	char(128)	拥有实体化视图的用户。
name	char(128)	实体化视图名称。
is_stale	char(1)	t 表示实体化视图已过时。过时的实体化视图是基表已更新但尚未刷新实体化视图的视图。如果自上次重新启动以来尚未运行刷新，则此信息可能不准确。
state	integer	实体化视图的状态如下： • 0 – 刷新时完全重新计算实体化视图。

列名称	数据类型	描述
		• 1 – 实体化视图是递增的。 • 101 – 实体化视图由于删除的列而无法刷新。即使在实体化视图中未使用列，此约束也会适用。 • 102 – 由于更改的列类型，实体化视图无法刷新。即使在实体化视图中未使用列，此约束也会适用。 • 103 – 由于重命名的表，实体化视图无法刷新。 • 104 – 由于重命名的列，实体化视图无法刷新。即使在实体化视图中未使用列，此约束也会适用。 • 105 – 由于重命名的 schema，实体化视图无法刷新。
自动重写	char(1)	t 表示实体化视图有资格自动重写查询。
自动刷新	char(1)	t 表示实体化视图可以自动刷新。

示例查询

要查看所有实体化视图的状态，请运行以下查询。

```
select * from svv_mv_info;
```

此查询返回以下示例输出。

```
database_name |       schema_name       | user_name |   name   |  is_stale | state |
 autorefresh | autorewrite
--------------+-------------------------+-----------+----------+-----------+-------
+-------------+----------------
 dev          | test_ivm_setup          | catch-22  | mv       |   f       |     1 |
       1 |            0
 dev          | test_ivm_setup          | lotr      | old_mv   |   t       |     1 |
       0 |            1
```

SVV_QUERY_INFLIGHT

使用 SVV_QUERY_INFLIGHT 视图可以确定当前正在对数据库运行的查询。此视图将 STV_INFLIGHT 联接到 STL_QUERYTEXT。SVV_QUERY_INFLIGHT 不会显示仅领导节点查询。有关更多信息，请参阅 仅领导节点函数。

SVV_QUERY_INFLIGHT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> 此视图仅在查询预置集群时可用。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。
slice	integer	正在运行查询的分片。
query	integer	查询 ID。可用于联接各种其他系统表和视图。
pid	integer	进程 ID。会话中的所有查询在同一进程中运行，因此，如果您在同一会话中运行一系列查询，则此值保持不变。您可以使用此列联接到 STL_ERROR 表。
starttime	timestamp	开始查询的时间。
suspended	integer	查询是否已暂停：**0** = false；**1** = true。
text	character(200)	查询文本，以 200 个字符递增。
sequence	integer	查询语句段的序号。

示例查询

下面的示例输出显示了两个当前正在运行的查询：SVV_QUERY_INFLIGHT 查询本身及查询 428（分成该表中的三行）。（在本示例输出中，starttime 和 statement 列被截断了。）

```
select slice, query, pid, starttime, suspended, trim(text) as statement, sequence
from svv_query_inflight
order by query, sequence;

slice|query| pid  |      starttime       |suspended| statement | sequence
-----+-----+------+----------------------+---------+-----------+---------
1012 | 428 | 1658 | 2012-04-10 13:53:... |       0 | select ...|    0
1012 | 428 | 1658 | 2012-04-10 13:53:... |       0 | enueid ...|    1
1012 | 428 | 1658 | 2012-04-10 13:53:... |       0 | atname,...|    2
1012 | 429 | 1608 | 2012-04-10 13:53:... |       0 | select ...|    0
(4 rows)
```

SVV_QUERY_STATE

使用 SVV_QUERY_STATE 查看有关当前正在运行的查询的运行时信息。

SVV_QUERY_STATE 视图包含 STV_EXEC_STATE 表的数据子集。

SVV_QUERY_STATE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

> **Note**
>
> 此视图仅在查询预置集群时可用。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。
query	integer	查询 ID。可用于联接各种其他系统表和视图。

列名称	数据类型	描述
seg	整数	正在运行的查询段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。查询段可并行运行。每个段在一个进程中运行。
step	整数	正在运行的查询步骤编号。步骤是最小的查询运行时单位。每个步骤代表独立的工作单位，如扫描表、返回结果或排序数据。
maxtime	interval	供此步骤运行的最大时长（单位为微秒）。
avgtime	interval	供此步骤运行的平均时长（单位为微秒）。
rows	bigint	正在运行的步骤产生的行数。
bytes	bigint	正在运行的步骤产生的字节数。
cpu	bigint	供内部使用。
memory	bigint	供内部使用。
rate_row	double precision	查询开始以来的每秒行数速率，计算方法为：汇总行数，然后除以从开始查询以来到当前时间的秒数。
rate_byte	double precision	查询开始以来的每秒字节速率，计算方法为：汇总字节数，然后除以从开始查询以来到当前时间的秒数。
label	character(25)	查询标签：步骤名称，如 `scan` 或 `sort`。
is_diskbased	character(1)	此查询步骤是否作为基于磁盘的操作运行：true (**t**) 或 false (**f**)。只有哈希、排序和聚合等特定步骤才能转到磁盘。许多类型的步骤始终在内存中执行。
workmem	bigint	分配到查询步骤的工作内存量（单位为字节）。
num_parts	integer	执行哈希步骤期间将哈希表划分成的分区数。此列中的正数并不表示哈希步骤是作为基于磁盘的操作运行的。请查看 IS_DISKBASED 列中的值以了解哈希步骤是否基于磁盘。

列名称	数据类型	描述
is_rrscan	character(1)	如果为 true (**t**)，则表示对步骤使用了限制范围的扫描。默认为 false (**f**)。
is_delayed_scan	character(1)	如果为 true (**t**)，则表示对步骤使用了延迟扫描。默认为 false (**f**)。

示例查询

按步骤确定查询的处理时间

下面的查询显示查询 ID 为 279 的查询每一步的运行时长以及 Amazon Redshift 处理的数据行数：

```
select query, seg, step, maxtime, avgtime, rows, label
from svv_query_state
where query = 279
order by query, seg, step;
```

此查询检索有关查询 279 的处理信息，如下面的示例输出所示：

```
query |   seg   | step | maxtime | avgtime |  rows   | label
------+---------+------+---------+---------+---------+-------------------
  279 |       3 |    0 | 1658054 | 1645711 | 1405360 | scan
  279 |       3 |    1 | 1658072 | 1645809 |       0 | project
  279 |       3 |    2 | 1658074 | 1645812 | 1405434 | insert
  279 |       3 |    3 | 1658080 | 1645816 | 1405437 | distribute
  279 |       4 |    0 | 1677443 | 1666189 | 1268431 | scan
  279 |       4 |    1 | 1677446 | 1666192 | 1268434 | insert
  279 |       4 |    2 | 1677451 | 1666195 |       0 | aggr
(7 rows)
```

确定当前是否在磁盘上运行任何事件查询

下面的查询显示磁盘上有没有任何当前正在运行的活动查询：

```
select query, label, is_diskbased from svv_query_state
where is_diskbased = 't';
```

此示例输出显示磁盘上当前正在运行的任何活动查询：

```
 query | label         | is_diskbased
-------+---------------+--------------
1025   | hash tbl=142 |      t
(1 row)
```

SVV_REDSHIFT_COLUMNS

使用 SVV_REDSHIFT_COLUMNS 可以查看用户有权访问的所有列的列表。这组列包括集群上的列和远程集群提供的数据共享中的列。

SVV_REDSHIFT_COLUMNS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	varchar(128)	包含列的表所在数据库的名称。
schema_name	varchar(128)	表 schema 的名称。
table_name	varchar(128)	表的名称。
column_name	varchar(128)	列的名称。
ordinal_position	integer	列在表中的位置。
data_type	varchar(32)	列的数据类型。
column_default	varchar(4000)	列的默认值。
is_nullable	varchar(3)	定义列是否可为 null 的值。可能的值包括 yes、no 或不表示任何信息的 "" 空字符串。
编码	varchar(128)	列的编码类型。
distkey	布尔值	一个值，如果此列为表的分配键，则为真；否则为假。

列名称	数据类型	描述
sortkey	integer	指定排序键中列的顺序的值。 如果表使用一个复合排序键，则排序键中的所有列将具有一个正值，该值指示列在排序键中的位置。 如果表使用交错排序键，则排序键中的所有列将具有一个正值或负值。其中，绝对值指示列在排序键中的位置。 如果 sortkey 为 0，则列不是排序键的一部分。
column_acl	varchar(128)	为列的指定用户或用户组定义权限的字符串。
remarks	varchar(256)	备注。

示例查询

下面的示例返回 SVV_REDSHIFT_COLUMNS 的输出。

```
SELECT *
FROM svv_redshift_columns
WHERE database_name = 'tickit_db'
    AND TABLE_NAME = 'tickit_sales_redshift'
ORDER BY COLUMN_NAME,
    TABLE_NAME,
    database_name
LIMIT 5;

database_name | schema_name |       table_name      | column_name | ordinal_position |
 data_type | column_default | is_nullable | encoding | distkey | sortkey | column_acl
 | remarks
```

```
-------------+------------+----------------------+------------+-----------------
+-----------+---------------+------------+----------+---------+---------
+-------------+--------
   tickit_db  |   public    | tickit_sales_redshift |   buyerid   |        4         |
 integer  |               |      NO     |   az64   |  False  |    0    |             |
   tickit_db  |   public    | tickit_sales_redshift |  commission |        9         |
 numeric  |      (8,2)    |     YES     |   az64   |  False  |    0    |             |
   tickit_db  |   public    | tickit_sales_redshift |    dateid   |        6         |
 smallint |               |      NO     |   none   |  False  |    1    |             |
   tickit_db  |   public    | tickit_sales_redshift |   eventid   |        5         |
 integer  |               |      NO     |   az64   |  False  |    0    |       |
   tickit_db  |   public    | tickit_sales_redshift |   listid    |        2         |
 integer  |               |      NO     |   az64   |  True   |    0    |             |
```

SVV_REDSHIFT_DATABASES

使用 SVV_ REDSHIFT_DATABASES 可以查看用户有权访问的所有数据库的列表。这包括集群上的数据库以及从远程集群提供的数据共享创建的数据库。

SVV_REDSHIFT_DATABASES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	varchar(128)	数据库的名称。
database_owner	integer	数据库拥有者用户 ID。
database_type	varchar(32)	数据库的类型。可能的类型包括本地数据库或共享数据库。
database_acl	varchar(128)	此信息仅供内部使用。
database_options	varchar(128)	数据库的属性。
database_isolation_level	varchar(128)	数据库的隔离级别。可能的值包括：Snapshot Isolation 和 Serializable 。

示例查询

以下示例返回 SVV_REDSHIFT_DATABASES 的输出。

```
select database_name, database_owner, database_type, database_options,
 database_isolation_level
from  svv_redshift_databases;


database_name | database_owner | database_type | database_options |
 database_isolation_level
--------------+----------------+---------------+------------------+------------------
   dev        |  1             | local         | NULL             | Serializable
```

SVV_REDSHIFT_FUNCTIONS

使用 SVV_REDSHIFT_FUNCTIONS 查看用户有权访问的所有函数的列表。这组函数包括集群上的函数和远程集群提供的数据共享中的函数。

SVV_REDSHIFT_FUNCTIONS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	varchar(128)	具有这些函数的集群所在的数据库的名称。
schema_name	varchar(128)	指定给定函数的 schema 的名称。
function_name	varchar(128)	指定函数的名称。
function_type	varchar(128)	函数的类型。可能的值包括常规函数、聚合函数和存储过程。
argument_type	varchar(512)	表示函数输入参数类型的字符串。

列名称	数据类型	描述
result_type	varchar(128)	函数返回值的数据类型。

示例查询

下面的示例返回 SVV_REDSHIFT_FUNCTIONS 的输出。

```
SELECT *
FROM svv_redshift_functions
WHERE database_name = 'tickit_db'
    AND SCHEMA_NAME = 'public'
ORDER BY function_name
LIMIT 5;

database_name | schema_name |      function_name    |  function_type   |
 argument_type  | result_type
--------------+-------------+-----------------------+------------------
+------------------+-------------
   tickit_db  |    public   |     shared_function   | REGULAR FUNCTION | integer,
 integer |   integer
```

SVV_REDSHIFT_SCHEMA_QUOTA

显示数据库中每个模式的限额和当前磁盘使用情况。

SVV_REDSHIFT_SCHEMA_QUOTA 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

当查询预调配集群或 Redshift Serverless 工作组时，此视图可用。

表列

列名称	数据类型	描述
database_name	character(128)	包含模式的数据库。
schema_name	character(128)	架构的名称。
schema_owner	integer	schema 拥有者的内部用户 ID。

列名称	数据类型	描述
配额	integer	schema 可以使用的磁盘空间量（以 MB 为单位）。
disk_usage	integer	schema 当前使用的磁盘空间（以 MB 为单位）。

示例查询

以下示例显示名为 sales_schema 的模式的限额和当前磁盘使用情况。

```
SELECT TRIM(SCHEMA_NAME) "schema_name", QUOTA, disk_usage FROM
 svv_redshift_schema_quota
WHERE SCHEMA_NAME = 'sales_schema';


schema_name   | quota | disk_usage
--------------+-------+-----------
sales_schema  | 2048  | 30
```

SVV_REDSHIFT_SCHEMAS

使用 SVV_REDSHIFT_SCHEMAS 可以查看用户有权访问的所有 schema 的列表。这组 schema 包括集群上的 schema 和远程集群提供的数据共享中的 schema。

SVV_REDSHIFT_SCHEMAS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	varchar(128)	指定 schema 所在的数据库的名称。
schema_name	varchar(128)	命名空间或 schema 名称。

列名称	数据类型	描述
schema_owner	integer	schema 拥有者的内部用户 ID。
schema_type	varchar(16)	schema 的类型。可能的值包括共享和本地 schema。
schema_acl	varchar(128)	为 Schema 的指定用户或用户组定义权限的字符串。
schema_option	varchar(128)	schema 的选项。

示例查询

下面的示例返回 SVV_REDSHIFT_SCHEMAS 的输出。

```
SELECT *
FROM svv_redshift_schemas
WHERE database_name = 'tickit_db'
ORDER BY database_name,
    SCHEMA_NAME;

database_name |    schema_name     | schema_owner | schema_type | schema_acl |
 schema_option
--------------+--------------------+--------------+-------------+------------
+---------------
   tickit_db  |       public       |       1      |    shared   |            |
```

SVV_REDSHIFT_TABLES

使用 SVV_REDSHIFT_TABLES 可以查看用户有权访问的所有表的列表。这组表包括集群上的表和远程集群提供的数据共享中的表。

SVV_REDSHIFT_TABLES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
database_name	varchar(128)	指定表所在数据库的名称。
schema_name	varchar(128)	表 schema 的名称。
table_name	varchar(128)	表的名称。
table_type	varchar(128)	表的类型。可能的值包括视图和表格。
table_acl	varchar(128)	为表的指定用户或用户组定义权限的字符串。
remarks	varchar(128)	备注。
table_owner	varchar(128)	表的所有者。

示例查询

下面的示例返回 SVV_REDSHIFT_TABLES 的输出。

```
SELECT *
FROM svv_redshift_tables
WHERE database_name = 'tickit_db' AND TABLE_NAME LIKE 'tickit_%'
ORDER BY database_name,
TABLE_NAME;

database_name | schema_name |         table_name       | table_type | table_acl |
 remarks | table_owner
--------------+-------------+--------------------------+------------+-----------
+---------+-----------
   tickit_db  |    public   | tickit_category_redshift |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_date_redshift   |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_event_redshift  |    TABLE   |           |
   +
```

```
tickit_db  |    public    |  tickit_listing_redshift |    TABLE    |           |
+
tickit_db  |    public    |   tickit_sales_redshift  |    TABLE    |           |
+
tickit_db  |    public    |   tickit_users_redshift  |    TABLE    |           |
+
tickit_db  |    public    |   tickit_venue_redshift  |    TABLE    |           |
```

如果 table_acl 值为 null，则没有显式授予针对相应表的访问权限。

SVV_RELATION_PRIVILEGES

使用 SVV_RELATION_PRIVILEGES 查看向当前数据库中的用户、角色和组显式授予的关系（表和视图）权限。

SVV_RELATION_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 SYSLOG ACCESS UNRESTRICTED 权限的用户

其他用户只能看到他们有权访问或拥有的身份。有关数据可见性的更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
namespace_name	文本	指定关系所在命名空间的名称。
relation_name	文本	关系的名称。
privilege_type	文本	权限类型。可能的值为 INSERT、SELECT、UPDATE、DELETE、REFERENCES 或 DROP。
identity_id	整数	身份 ID。可能的值为用户 ID、角色 ID 或组 ID。
identity_name	文本	身份的名称。
identity_type	文本	身份的类型。可能的值为用户、角色、组或公有。

列名称	数据类型	描述
admin_option	布尔值	一个指示用户是否可以向其他用户和角色授予权限的值。对于角色和组身份类型，它始终为 false。

示例查询

以下示例显示了 SVV_RELATION_PRIVILEGES 的结果。

```
SELECT
 namespace_name,relation_name,privilege_type,identity_name,identity_type,admin_option
 FROM svv_relation_privileges
WHERE relation_name = 'orders' AND privilege_type = 'SELECT';

 namespace_name | relation_name | privilege_type |  identity_name | identity_type |
 admin_option
----------------+---------------+----------------+----------------+---------------
+--------------
     public     |    orders     |     SELECT     |    reguser     |     user      |
 False
     public     |    orders     |     SELECT     |     role1      |     role      |
 False
```

SVV_RLS_APPLIED_POLICY

使用 SVV_RLS_APPLIED_POLICY 跟踪 RLS 策略在引用受 RLS 保护的关系的查询上的应用情况。

SVV_RLS_APPLIED_POLICY 对以下用户可见：

- 超级用户
- 拥有 sys:operator 角色的用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

请注意，sys:secadmin 未被授予此系统权限。

表列

列名称	数据类型	描述
username	文本	运行查询的用户的名称。
query	整数	查询的 ID。
xid	long	事务的上下文。
pid	整数	运行查询的领导进程。
recordtime	time	记录查询时的时间。
命令	char(1)	应用了 RLS 策略的命令。可能的值包括 k（用于 unknown）、s（用于 select）、u（用于 update）、i（用于 insert）、y（用于 utility），以及 d（用于 delete）。
datname	文本	行级别安全性策略附加到的关系的数据库的名称。
relschema	文本	行级别安全性策略附加到的关系的架构的名称。
relname	文本	行级别安全性策略附加到的关系的名称。
polname	文本	附加到关系的行级别安全性策略的名称。
poldefault	char(1)	附加到关系的行级别安全性策略的默认设置。可能的值包括用于 false 的 f（如果已应用默认的 false 策略）；以及用于 true 的 t（如果已应用默认的 true 策略）。

示例查询

以下示例显示了 SVV_RLS_APPLIED_POLICY 的结果。要查询 SVV_RLS_APPLIED_POLICY，您必须拥有 ACCESS SYSTEM TABLE 权限。

```
-- Check what RLS policies were applied to the run query.
SELECT username, command, datname, relschema, relname, polname, poldefault
FROM svv_rls_applied_policy
WHERE datname = CURRENT_DATABASE() AND query = PG_LAST_QUERY_ID();
```

```
 username | command |  datname  | relschema |          relname          |      polname
  | poldefault
----------+---------+-----------+-----------+---------------------------
+-----------------+------------
   molly  |    s    | tickit_db |   public  | tickit_category_redshift |
 policy_concerts |
```

SVV_RLS_ATTACHED_POLICY

使用 SVV_RLS_ATTACHED_POLICY 以查看已在当前连接的数据库上附加了一个或多个行级别安全性策略的所有关系和用户的列表。

只有拥有 sys:secadmin 角色的用户才能查询此视图。

表列

列名称	数据类型	描述
relschema	文本	行级别安全性策略附加到的关系的架构的名称。
relname	文本	行级别安全性策略附加到的关系的名称。
relkind	文本	对象的类型，如表。
polname	文本	附加到关系的行级别安全性策略的名称。
grantor	文本	已附加此策略的用户的名称。
grantee	文本	此策略已附加到的用户或角色的名称。
granteekind	文本	被授权者的类型。可能的值为用户或角色。
is_pol_on	布尔值	指示是否在表上打开或关闭行级别安全性策略的参数。可能的值包括 true 和 false。
is_rls_on	布尔值	指示是否在表上打开或关闭行级别安全性的参数。可能的值包括 true 和 false。
rls_conjunction_type	character (3)	指示关系是使用 and 还是 or 合并 RLS 策略的参数。

示例查询

以下示例显示了 SVV_RLS_ATTACHED_POLICY 的结果。

```
--Inspect the policy in SVV_RLS_ATTACHED_POLICY
SELECT * FROM svv_rls_attached_policy;

 relschema |        relname           | relkind |     polname     | grantor | grantee
 | granteekind | is_pol_on | is_rls_on | rls_conjuntion_type
-----------+--------------------------+---------+-----------------+---------+----------
+-------------+-----------+-----------+---------------------
 public    | tickit_category_redshift |  table  | policy_concerts |   bob   |  analyst
 |    role     |    True   |    True   |      and
 public    | tickit_category_redshift |  table  | policy_concerts |   bob   |  dbadmin
 |    role     |    True   |    True   |      and
```

SVV_RLS_POLICY

使用 SVV_RLS_POLICY 以查看在 Amazon Redshift 集群上创建的所有行级别安全性策略的列表。

SVV_RLS_POLICY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
poldb	文本	在其中创建行级别安全性策略的数据库的名称。
polname	文本	行级别安全性策略的名称。
polalias	文本	策略定义中使用的表别名。
polatts	文本	提供给策略定义的属性。
polqual	文本	在 CREATE POLICY 语句的 USING 子句中提供的策略条件。
polenabled	布尔值	是否在全局打开策略。
polmodifiedby	文本	最近创建或修改策略的用户的名称。

列名称	数据类型	描述
polmodifiedtime	时间戳	创建或上次修改策略时的时间戳。

示例查询

以下示例显示了 SVV_RLS_POLICY 的结果。

```
-- Create some policies.
CREATE RLS POLICY pol1 WITH (a int) AS t USING ( t.a IS NOT NULL );
CREATE RLS POLICY pol2 WITH (c varchar(10)) AS t USING ( c LIKE '%public%');

-- Inspect the policy in SVV_RLS_POLICY
SELECT * FROM svv_rls_policy;

 poldb | polname | polalias |                    polatts                     |
         polqual               | polenabled | polmodifiedby |   polmodifiedtime
-------+---------+----------+------------------------------------------------
+---------------------------------------+-----------+--------------
+---------------------
 my_db | pol1    | t        | [{"colname":"a","type":"integer"}]             |
 "t"."a" IS NOT NULL                   | t          | policy_admin  | 2022-02-11
 14:40:49
 my_db | pol2    | t        | [{"colname":"c","type":"character varying(10)"}] |
 "t"."c" LIKE CAST('%public%' AS TEXT) | t          | policy_admin  | 2022-02-11
 14:41:28
```

SVV_RLS_RELATION

使用 SVV_RLS_RELATION 以查看受 RLS 保护的所有关系的列表。

SVV_RLS_RELATION 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
datname	文本	包含关系的数据库的名称。

列名称	数据类型	描述
relschema	文本	包含关系的架构的名称。
relname	文本	关系的名称。
relkind	文本	关系的类型，如表或视图。
is_rls_on	布尔值	指示关系是否受 RLS 保护的参数。
is_rls_datashare_on	布尔值	指示关系是否受通过数据共享实施的 RLS 保护的参数。
rls_conjunction_type	character(3)	指示关系是使用 and 还是 or 合并 RLS 策略的参数。
rls_datashare_conjunction_type	character(3)	指示关系是使用 and 还是 or 通过数据共享合并 RLS 策略的参数。

示例查询

以下示例显示了 SVV_RLS_RELATION 的结果。

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON FOR DATASHARES;


--Inspect RLS state on the relations using SVV_RLS_RELATION.
SELECT datname, relschema, relname, relkind, is_rls_on, is_rls_datashare_on FROM
 svv_rls_relation ORDER BY relname;

  datname  | relschema |        relname           | relkind | is_rls_on |
 is_rls_datashare_on | rls_conjunction_type | rls_datashare_conjunction_type
-----------+-----------+--------------------------+---------+-----------
+---------------------+----------------------+--------------------------------
 tickit_db |   public  | tickit_category_redshift |  table  |      t    |           t
       |          and         |              and
(1 row)
```

SVV_ROLE_GRANTS

使用 SVV_ROLE_GRANTS 查看在集群中显式授予的角色的角色列表。

SVV_ROLE_GRANTS 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
role_id	整数	角色 ID。
role_name	文本	角色的名称。
granted_role_id	整数	授予的角色的 ID。
granted_role_name	文本	授予的角色的名称。

示例查询

以下示例将返回 SVV_ROLE_GRANTS 的输出。

```
GRANT ROLE role1 TO ROLE role2;
GRANT ROLE role2 TO ROLE role3;

SELECT role_name, granted_role_name FROM svv_role_grants;

 role_name |  granted_role_name
-----------+--------------------
   role2   |      role1
   role3   |      role2
(2 rows)
```

SVV_ROLES

使用 SVV_ROLES 查看用户有权访问的角色列表。

SVV_ROLES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
role_id	整数	角色 ID。
role_name	文本	角色的名称。
role_owner	文本	角色拥有者的名称。
external_id	文本	角色在第三方身份提供商中的唯一标识符。

示例查询

以下示例将返回 SVV_ROLES 的输出。

```
SELECT role_name,role_owner FROM svv_roles WHERE role_name IN ('role1', 'role2');

 role_name | role_owner
-----------+------------
   role1   | superuser
   role2   | superuser
```

SVV_SCHEMA_PRIVILEGES

使用 SVV_SCHEMA_PRIVILEGES 查看向当前数据库中的用户、角色和组显式授予的 Schema 权限。

SVV_SCHEMA_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
namespace_name	文本	指定 Schema 所在命名空间的名称。
privilege_type	文本	权限类型。可能的值为 USAGE 或 CREATE。
identity_id	整数	身份 ID。可能的值为用户 ID、角色 ID 或组 ID。
identity_name	文本	身份的名称。
identity_type	文本	身份的类型。可能的值为用户、角色、组或公有。
admin_option	布尔值	一个指示用户是否可以向其他用户和角色授予权限的值。对于角色和组身份类型，它始终为 false。

示例查询

以下示例显示了 SVV_SCHEMA_PRIVILEGES 的结果。

```
SELECT namespace_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_schema_privileges
WHERE namespace_name = 'test_schema1';

 namespace_name | privilege_type |  identity_name | identity_type | admin_option
----------------+----------------+----------------+---------------+--------------
 test_schema1   |    USAGE       |     reguser    |     user      |   False
 test_schema1   |    USAGE       |     role1      |     role      |   False
```

SVV_SCHEMA_QUOTA_STATE

显示每个 schema 的配额和当前磁盘使用情况。

普通用户可以查看其拥有 USAGE 权限的 schema 的信息。超级用户可以查看当前数据库中所有 schema 的信息。

SVV_SCHEMA_QUOTA_STATE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> 此视图仅在查询预置集群时可用。

表列

列名称	数据类型	描述
schema_id	integer	命名空间或 schema ID。
schema_name	character (128)	命名空间或 schema 名称。
schema_owner	integer	schema 拥有者的内部用户 ID。
配额	integer	schema 可以使用的磁盘空间量（以 MB 为单位）。
disk_usage	integer	schema 当前使用的磁盘空间（以 MB 为单位）。
disk_usage_pct	double precision	schema 当前使用的磁盘空间相对于为其配置的配额的百分比。

示例查询

以下示例显示 schema 的配额和当前磁盘使用情况。

```
SELECT TRIM(SCHEMA_NAME) "schema_name", QUOTA, disk_usage, disk_usage_pct FROM
 svv_schema_quota_state
```

```
WHERE SCHEMA_NAME = 'sales_schema';
schema_name   | quota | disk_usage | disk_usage_pct
--------------+-------+------------+----------------
sales_schema  | 2048  | 30         | 1.46
(1 row)
```

SVV_SYSTEM_PRIVILEGES

SVV_SYSTEM_PRIVILEGES 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到他们有权访问或拥有的身份。

表列

列名称	数据类型	描述
system_privilege	文本	系统权限的名称。
identity_id	整数	身份 ID。可能的值为用户 ID 或角色 ID。
identity_name	文本	身份的名称。
identity_type	文本	身份的类型。可能的值为用户或角色。

示例查询

以下示例显示了指定参数的结果。

```
SELECT system_privilege,identity_name,identity_type FROM svv_system_privileges
WHERE system_privilege = 'ALTER TABLE' AND identity_name = 'sys:superuser';

 system_privilege | identity_name | identity_type
```

```
------------------+---------------+---------------
   ALTER TABLE    | sys:superuser |     role
```

SVV_TABLE_INFO

显示数据库中表的摘要信息。该视图筛选系统表并且只显示用户定义的表。

您可以使用 SVV_TABLE_INFO 视图来诊断和解决可能影响查询性能的表设计问题。这包括压缩编码、分配键、排序方式、数据分配偏斜、表大小和统计数据等方面的问题。SVV_TABLE_INFO 视图对于空表不返回任何信息。

SVV_TABLE_INFO 视图汇总了来自 STV_BLOCKLIST、STV_NODE_STORAGE_CAPACITY、STV_TBL_PERM 和 STV_SLICES 系统表以及来自 PG_DATABASE、PG_ATTRIBUTE、PG_CLASS、PG_NAMESPACE 和 PG_TYPE 目录表的信息。

SVV_TABLE_INFO 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。要允许用户查询视图，请向该用户授予 SVV_TABLE_INFO 上的 SELECT 权限。

表列

列名称	数据类型	描述
database	text	数据库名称。
schema	text	Schema 名称。
table_id	oid	表 ID。
table	text	表名称。
encoded	text	一个指示是否有任何列定义了压缩编码的值。
diststyle	text	分配方式或分配键列（如果定义了键分配）。可能的值包括 EVEN、KEY(*column*)、ALL、AUTO(A 、AUTO(EVEN) 和 AUTO(KEY(*column*))。

列名称	数据类型	描述
sortkey1	text	排序键中的第一列（如果定义了排序键）。可能的值包括 *column*、AUTO(SORT KEY) 和 AUTO(SORT KEY(*column*))。
max_varchar	integer	使用 VARCHAR 数据类型的最大列的大小。
sortkey1_enc	character(32)	排序键中第一列的压缩编码（如果定义了排序键）。
sortkey_num	integer	定义为排序键的列数。
size	bigint	表的大小（单位为 1MB 数据块）。
pct_used	numeric(10,4)	表使用的可用空间的百分比。
empty	bigint	供内部使用。此列已不再使用，并将在未来的发行版中删除。
unsorted	numeric(5,2)	表中未排序行的百分比。
stats_off	numeric(5,2)	一个指示表统计数据过时程度的数字；0 表示最新，100 表示过时。
tbl_rows	numeric(38,0)	表中的总行数。此值包括标记为删除但尚未执行 vacuum 操作的行。
skew_sortkey1	numeric(19,2)	最大非排序键列的大小与排序键第一列的大小的比率（如果定义了排序键）。使用此值可以评估排序键的有效性。

列名称	数据类型	描述
skew_rows	numeric(19,2)	行数最多的切片中的行数与行数最少的切片中的行数的比率。
estimated_visible_rows	numeric(38,0)	表中的估计行数。此值不包括标记为删除的行。

risk_event	text	有关表的风险信息。该字段分为几部分：
		`risk_type |xid|timestamp`
		• `risk_type` ，其中 1 表示 `COPY command with the EXPLICIT_IDS option` 运行。Amazon Redshift 不再检查表中 IDENTITY 列的唯一性。有关更多信息，请参阅 EXPLICIT_IDS。
		• 引入风险的事务 ID `xid`。
		• COPY 命令运行时的 `timestamp` 。
		以下示例显示字段中的值。
		`1|1107|2019-06-22 07:16:11.292952`

vacuum_sort_benefit	numeric(12,2)	运行 vacuum 排序时，扫描查询性能的最大估计改进百分比。
create_time	不带时区的时间戳	表创建时间的时间戳。

示例查询

下面的示例显示数据库中所有用户定义的表的编码、分配方式、排序和数据偏斜。其中，“table”是一个保留字，因此必须用双引号括起来。

```
select "table", encoded, diststyle, sortkey1, skew_sortkey1, skew_rows
from svv_table_info
order by 1;

table          | encoded | diststyle       | sortkey1     | skew_sortkey1 | skew_rows
---------------+---------+-----------------+--------------+---------------+----------
category       | N       | EVEN            |              |               |
date           | N       | ALL             | dateid       |          1.00 |
event          | Y       | KEY(eventid)    | dateid       |          1.00 |      1.02
listing        | Y       | KEY(listid)     | dateid       |          1.00 |      1.01
sales          | Y       | KEY(listid)     | dateid       |          1.00 |      1.02
users          | Y       | KEY(userid)     | userid       |          1.00 |      1.01
venue          | N       | ALL             | venueid      |          1.00 |
(7 rows)
```

SVV_TABLES

使用 SVV_TABLES 查看本地和外部目录中的表。

SVV_TABLES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
table_catalog	text	表所在目录的名称。
table_schema	text	表 schema 的名称。
table_name	text	表的名称。
table_type	text	表的类型。可能的值包括视图、外部表和基表。
备注	text	备注。

SVV_TRANSACTIONS

记录当前锁定到数据库中的表的事务的相关信息。使用 SVV_TRANSACTIONS 视图可标识未结事务和锁定争用问题。有关锁定的更多信息，请参阅管理并发写入操作和LOCK。

SVV_TRANSACTIONS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
txn_owner	text	事务所有者的名称。
txn_db	text	与事务关联的数据库的名称。
xid	bigint	事务 ID。
pid	integer	与锁定关联的进程 ID。
txn_start	timestamp	事务的开始时间。
lock_mode	text	此进程保持或请求的锁定的名称。如果 lock_mode 为 ExclusiveLock 并且 granted 为 true (t)，则此事务 ID 是未结事务。
lockable_object_type	text	请求或保持锁定的对象的类型。对象是表时，类型为 relation，对象是事务时，类型为 transactionid 。
关系	integer	获取锁定的表（关系）的表 ID。如果 lockable_object_type 为 transactionid ，则此值为 NULL。

列名称	数据类型	描述
granted	布尔值	一个值，指示锁定是已获得许可 (t) 还是处于挂起状态 (f)。

示例查询

以下命令显示所有活动事务和每个事务请求的锁定。

```
select * from svv_transactions;

 txn_
 lockable_
 owner | txn_db |  xid   |  pid  |         txn_start          |      lock_mode      |
 object_type    | relation | granted
-------+--------+--------+-------+----------------------------+---------------------
+----------------+----------+---------
 root  | dev    | 438484 | 22223 | 2016-03-02 18:42:18.862254 | AccessShareLock     |
 relation       |   100068 | t
 root  | dev    | 438484 | 22223 | 2016-03-02 18:42:18.862254 | ExclusiveLock       |
 transactionid  |          | t
 root  | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | AccessShareLock     |
 relation       |    50860 | t
 root  | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | AccessShareLock     |
 relation       |    52310 | t
 root  | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | ExclusiveLock       |
 transactionid  |          | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100068 | f
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | RowExclusiveLock    |
 relation       |    16688 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessShareLock     |
 relation       |   100064 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100166 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100171 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100190 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | ExclusiveLock       |
 transactionid  |          | t
(12 rows)
```

```
(12 rows)
```

SVV_USER_GRANTS

使用 SVV_USER_GRANTS 查看在集群中被显式授予了角色的用户列表。

SVV_USER_GRANTS 对以下用户可见：

- 超级用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

其他用户只能看到明确授予给他们的角色。

表列

列名称	数据类型	描述
user_id	整数	用户的用户 ID。
user_name	文本	用户的名称。
role_id	整数	所授予角色的角色 ID。
role_name	文本	所授予角色的角色名称。
admin_option	布尔值	一个指示用户是否可以向其他用户和角色授予角色的值。

示例查询

以下查询将向用户授予角色并显示被显式授予了角色的用户列表。

```
GRANT ROLE role1 TO reguser;
GRANT ROLE role2 TO reguser;
GRANT ROLE role1 TO superuser;
GRANT ROLE role2 TO superuser;

SELECT user_name,role_name,admin_option FROM svv_user_grants;
```

```
 user_name | role_name | admin_option
-----------+-----------+--------------
 superuser |  role1    | False
 reguser   |  role1    | False
 superuser |  role2    | False
  reguser  |  role2    | False
```

SVV_USER_INFO

您可以使用 SVV_USER_INFO 视图检索有关 Amazon Redshift 数据库用户的数据。

SVV_USER_INFO 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_name	文本	角色的用户名称。
user_id	整数	用户的用户 ID。
createdb	布尔值	一个值，指示用户是否具有创建数据库的权限。
超级用户	布尔值	一个值，指示用户是否为超级用户。
catalog_update	布尔值	一个值，指示用户是否可以更新系统目录。
connection_limit	文本	用户可以打开的连接数。
syslog_access	文本	一个值，指示用户是否具有访问系统日志的权限。两个可能的值为 RESTRICTED 和 UNRESTRICTED 。RESTRICTED 表示非超级用户可以看到自己的记录。UNRESTRICTED 表示非超级用户可以看到他们拥有 SELECT 权限的系统视图和表中的所有记录。
last_ddl_timestamp	时间戳	用户运行的最后一条数据定义语言 (DDL) create 语句的时间戳。

列名称	数据类型	描述
session_timeout	整数	超时前会话保持非活动状态或空闲状态的最长时间（秒）。0 表示未设置超时。有关集群的闲置或非活动超时设置的信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 中的配额和限制。
external_user_id	文本	用户在第三方身份提供者中的唯一标识符。

示例查询

以下命令从 SVV_USER_INFO 检索用户信息。

```
SELECT * FROM SVV_USER_INFO;
```

SVV_VACUUM_PROGRESS

此视图返回完成当前正在执行的 vacuum 操作的估算时间。

SVV_VACUUM_PROGRESS 只对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_VACUUM_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

有关 SVV_VACUUM_SUMMARY 的信息，请参阅 SVV_VACUUM_SUMMARY。

有关 SVL_VACUUM_PERCENTAGE 的信息，请参阅 SVL_VACUUM_PERCENTAGE。

> **Note**
>
> 此视图仅在查询预置集群时可用。

表列

列名称	数据类型	描述
table_name	text	当前正在执行 vacuum 操作的表或最近一次执行 vacuum 操作的表（如果当前未执行任何操作）的名称。
status	text	作为 vacuum 操作的一部分完成的当前活动的描述： • Initialize • 排序 • 合并 • 删除 • Select • 失败 • 完成 • Skipped • 正在生成 INTERLEAVED SORTKEY 顺序
time_remaining_estimate	text	完成当前 vacuum 操作的估算剩余时间（单位为分和秒）：如 **5m 10s**。在 vacuum 完成其第一个排序操作前不会返回估算时间。如果没有正在执行的 vacuum 操作，则显示最近一次执行的 vacuum 操作（STATUS 列显示 **Completed** ，TIME_REMAINING_ESTIMATE 列为空）。随着 vacuum 的执行，估算通常会越来越准。

示例查询

以下查询（隔几分钟运行）显示正在对名为 SALESNEW 的大型表执行 vacuum 操作。

```
select * from svv_vacuum_progress;

table_name    |            status              | time_remaining_estimate
--------------+--------------------------------+-------------------------
salesnew      |  Vacuum: initialize salesnew  |
```

```
(1 row)
...
select * from svv_vacuum_progress;

table_name   |         status         | time_remaining_estimate
-------------+------------------------+-------------------------
salesnew     |  Vacuum salesnew sort  | 33m 21s
(1 row)
```

下面的查询显示当前没有正在执行的 vacuum 操作。最近一次执行 vacuum 操作的表是 SALES 表。

```
select * from svv_vacuum_progress;

table_name   |  status  | time_remaining_estimate
-------------+----------+-------------------------
  sales      | Complete |
(1 row)
```

SVV_VACUUM_SUMMARY

SVV_VACUUM_SUMMARY 视图联接 STL_VACUUM、STL_QUERY、STV_TBL_PERM 表，以汇总系统记录的 vacuum 操作的信息。该视图每个 vacuum 事务每张表返回一行。该视图记录操作执行前后的用时、所创建的排序分区的数量、需要的合并增量的数量以及行和数据块计数中的增量。

SVV_VACUUM_SUMMARY 只对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_VACUUM_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

有关 SVV_VACUUM_PROGRESS 的信息，请参阅 SVV_VACUUM_PROGRESS。

有关 SVL_VACUUM_PERCENTAGE 的信息，请参阅 SVL_VACUUM_PERCENTAGE。

> **Note**
>
> 此视图仅在查询预置集群时可用。

表列

列名称	数据类型	描述
table_name	text	执行 vacuum 操作的表的名称。
xid	bigint	VACUUM 操作的事务 ID。
sort_partitions	bigint	vacuum 操作排序阶段期间创建的排序分区的数量。
merge_increments	bigint	完成 vacuum 操作合并阶段所需的合并增量的数量。
elapsed_time	bigint	Vacuum 操作已经过的运行时（单位为微秒）。
row_delta	bigint	执行 vacuum 操作前后表总行数的差值。
sortedrow_delta	bigint	执行 vacuum 操作前后排序表行数的差值。
block_delta	integer	执行 vacuum 操作前后表数据块数的差值。
max_merge_partitions	integer	此列用于性能分析并表示 vacuum 可在每个合并阶段迭代中为表处理的分区的最大数量。（Vacuum 将未排序的区域分为一个或多个已排序的分区。根据表中的列数和当前的 Amazon Redshift 配置，合并阶段可在一个合并迭代中处理最大数量的分区。如果已排序分区的数量超出了合并分区的最大数量，合并阶段仍然有效，但将需要更多合并迭代。）

示例查询

下面的查询返回三个不同的表的 vacuum 操作的统计数据。SALES 表执行了两次 vacuum 操作。

```
select table_name, xid, sort_partitions as parts, merge_increments as merges,
elapsed_time, row_delta, sortedrow_delta as sorted_delta, block_delta
from svv_vacuum_summary
order by xid;

table_  | xid  |parts|merges| elapsed_ | row_    | sorted_ | block_
name    |      |     |      | time     | delta   | delta   | delta
```

```
--------+------+-----+------+----------+---------+---------+--------
users   | 2985 |   1 |    1 | 61919653 |       0 |   49990 |      20
category| 3982 |   1 |    1 | 24136484 |       0 |      11 |       0
sales   | 3992 |   2 |    1 | 71736163 |       0 | 1207192 |      32
sales   | 4000 |   1 |    1 | 15363010 | -851648 | -851648 |    -140
(4 rows)
```

SYS 监控视图

监控视图是 Amazon Redshift 中用于监控预置集群和无服务器工作组的查询和工作负载资源使用情况的系统视图。这些视图位于 pg_catalog 架构中。要显示这些视图提供的信息，请运行 SQL SELECT 语句。

除非另有说明，否则这些视图适用于 Amazon Redshift 集群和 Amazon Redshift Serverless 工作组。

SYS_SERVERLESS_USAGE 仅会收集 Amazon Redshift Serverless 的使用情况数据。

主题

- SYS_LOAD_ERROR_DETAIL
- SYS_LOAD_HISTORY
- SYS_MV_REFRESH_HISTORY
- SYS_MV_STATE
- SYS_PROCEDURE_CALL
- SYS_PROCEDURE_MESSAGES
- SYS_QUERY_DETAIL
- SYS_QUERY_HISTORY
- SYS_QUERY_TEXT
- SYS_RESTORE_LOG
- SYS_RESTORE_STATE
- SYS_SCHEMA_QUOTA_VIOLATIONS
- SYS_SERVERLESS_USAGE
- SYS_SESSION_HISTORY
- SYS_SPATIAL_SIMPLIFY
- SYS_STREAM_SCAN_ERRORS
- SYS_STREAM_SCAN_STATES
- SYS_TRANSACTION_HISTORY
- SYS_UDF_LOG
- SYS_UNLOAD_DETAIL
- SYS_UNLOAD_HISTORY
- SYS_USERLOG
- SYS_VACUUM_HISTORY

SYS_ANALYZE_COMPRESSION_HISTORY

记录在 COPY 或 ANALYZE COMPRESSION 命令期间压缩分析操作的详细信息。

SYS_ANALYZE_COMPRESSION_HISTORY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	生成该条目的用户的 ID。
start_time	timestamp	开始压缩分析操作的时间。
transaction_id	bigint	压缩分析操作的事务 ID。
table_id	整数	所分析的表的表 ID。
table_name	character(128)	所分析的表的名称。
column_position	整数	表中进行分析以确定压缩编码的列的索引。
old_encoding	character(15)	压缩分析之前的编码类型。
new_encoding	character(15)	压缩分析之后的编码类型。
mode	character(14)	可能的值包括： PRESET 指定 new_encoding 由 Amazon Redshift COPY 命令根据列数据类型决定。不会对数据进行采样。 开 指定 new_encoding 由 Amazon Redshift COPY 命令根据采样数据的分析决定。 ANALYZE ONLY 指定 new_encoding 由 Amazon Redshift ANALYZE COMPRESSION 命令根据采样数据的分析决定。但是，不会改变所分析的列的编码类型。

示例查询

以下示例检查在同一个会话中运行的最后一个 COPY 命令对 lineitem 表执行压缩分析的详细信息。

```
select transaction_id, table_id, btrim(table_name) as table_name, column_position,
 old_encoding, new_encoding, mode
from sys_analyze_compression_history
where transaction_id = (select transaction_id from sys_query_history where query_id =
 pg_last_copy_id()) order by column_position;

 transaction_id  |  table_id   | table_name | column_position |  old_encoding   |
 new_encoding   |      mode
-----------------+-------------+------------+-----------------+-----------------
+-----------------+-------------
      8196       |   248126    | lineitem   |        0        | mostly32        |
 mostly32        | ON
      8196       |   248126    | lineitem   |        1        | mostly32        | lzo
           | ON
      8196       |   248126    | lineitem   |        2        | lzo             |
 delta32k        | ON
      8196       |   248126    | lineitem   |        3        | delta           | delta
           | ON
      8196       |   248126    | lineitem   |        4        | bytedict        |
 bytedict        | ON
      8196       |   248126    | lineitem   |        5        | mostly32        |
 mostly32        | ON
      8196       |   248126    | lineitem   |        6        | delta           | delta
           | ON
      8196       |   248126    | lineitem   |        7        | delta           | delta
           | ON
      8196       |   248126    | lineitem   |        8        | lzo             | zstd
           | ON
      8196       |   248126    | lineitem   |        9        | runlength       | zstd
           | ON
      8196       |   248126    | lineitem   |       10        | delta           | lzo
           | ON
      8196       |   248126    | lineitem   |       11        | delta           | delta
           | ON
      8196       |   248126    | lineitem   |       12        | delta           | delta
           | ON
      8196       |   248126    | lineitem   |       13        | bytedict        | zstd
           | ON
      8196       |   248126    | lineitem   |       14        | bytedict        | zstd
           | ON
```

```
      8196      |   248126    | lineitem   |       15        | text255         | zstd
          | ON
(16 rows)
```

SYS_ANALYZE_HISTORY

记录 ANALYZE 操作的详细信息。

SYS_ANALYZE_HISTORY 只对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	生成该条目的用户的 ID。
transaction_id	long	事务 ID。
query_id	long	SYS_QUERY_HISTORY 中的查询标识符。
database_name	char(30)	数据库的名称。
table_name	char(30)	表的名称。
table_id	整数	表的 ID。
is_automatic	char(1)	如果操作默认情况下包含 Amazon Redshift ANALYZE 操作，则值为 true (t)。如果显式运行了 ANALYZE 命令，则值为 false (f)。
status	char(15)	分析命令的结果。可能的值为 Full、Skipped 和 Predicate Column。
start_time	时间戳	ANALYZE 操作开始运行的时间（采用 UTC 表示）。
end_time	时间戳	ANALYZE 操作完成运行的时间（采用 UTC 表示）。
rows	double	表中的总行数

列名称	数据类型	描述
modified_rows	double	自上次执行 ANALYZE 操作以来修改的行的总数。
analyze_threshold_percent	整数	analyze_threshold_percent 参数的值。
last_analyze_time	时间戳	上次分析表的时间（采用 UTC 表示）。

示例查询

```
 user_id | transaction_id | database_name | schema_name |      table_name     |
 table_id | is_automatic | Status |         start_time         |          end_time
     | rows | modified_rows | analyze_threshold_percent |  last_analyze_time
---------+----------------+---------------+------------+---------------------
+----------+--------------+--------+---------------------------+----------
+-----------------+------+--------------+--------------------------
+---------------------
     101 |           8006 |           dev |      public | test_table_562bf8dc
 |   110427 |            f |   Full | 2023-09-21 18:33:08.504646 | 2023-09-21
 18:33:24.296498 |    5 |             5 |                          0 | 2000-01-01
 00:00:00
```

SYS_APPLIED_MASKING_POLICY_LOG

使用 SYS_APPLIED_MASKING_POLICY_LOG 跟踪动态数据掩蔽策略在引用受 RLS 保护关系的查询上的应用情况。

SYS_APPLIED_MASKING_POLICY_LOG 对以下用户可见：

- 超级用户
- 拥有 `sys:operator` 角色的用户
- 拥有 ACCESS SYSTEM TABLE 权限的用户

常规用户将看到 0 行。

请注意，拥有 sys:secadmin 角色的用户看不到 SYS_APPLIED_MASKING_POLICY_LOG。

有关动态数据屏蔽的更多信息，请参阅动态数据掩蔽。

表列

列名称	数据类型	描述
policy_name	文本	屏蔽策略的名称。
user_id	文本	发起查询的用户的 ID。
record_time	时间戳	记录系统视图条目的时间。
session_id	int	进程 ID。
transaction_id	long	事务 ID。
query_id	int	查询 ID。
database_name	文本	在其上运行了查询的数据库的名称。
relation_name	文本	要对其应用屏蔽策略的表的名称。
schema_name	文本	表所在架构的名称。
attachment_id	long	所附加屏蔽策略的 ID。
relation_kind	文本	对其应用屏蔽策略的关系的类型。可能的值为 TABLE、VIEW、LATE BINDING VIEW 和 MATERIALIZED VIEW。

示例查询

以下示例显示 mask_credit_card_full 屏蔽策略已附加到 credit_db.public.credit_cards 表中。

```
select policy_name, database_name, relation_name, schema_name, relation_kind
from sys_applied_masking_policy_log;

policy_name           | database_name | relation_name | schema_name | relation_kind
----------------------+---------------+---------------+-------------+--------------
mask_credit_card_full | credit_db     | credit_cards  | public      | table

(1 row)
```

SYS_AUTO_TABLE_OPTIMIZATION

记录 Amazon Redshift 对自动优化定义的表执行的自动操作。

SYS_AUTO_TABLE_OPTIMIZATION 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
transaction_id	long	事务标识符。
session_id	int	执行 Alter 命令的进程的标识符。
table_id	int	表标识符。
alter_table_type	character (32)	建议的类型。可能的值为 distkey、sortkey 和 encode。
status	character (128)	建议的完成状态。可能的值包括 Start、Complete、Skipped、Abort、Checkpoint 和 Failed。

列名称	数据类型	描述
event_time	时间戳	状态列的时间戳。
alter_from	character (200)	应用建议之前表的上一个分配方式和排序键。该值被截断为 200 个字符的增量。
alter_to	character (200)	应用建议后表的当前分配方式和排序键。该值被截断为 200 个字符的增量。

示例查询

在以下示例中，结果中的行显示了由 Amazon Redshift 执行的操作。

```
SELECT table_id, alter_table_type, status, event_time, alter_from
FROM SYS_AUTO_TABLE_OPTIMIZATION;

 table_id |  alter_table_type   |                        status
 |         event_time          |  alter_from
----------+---------------------+-----------------------------------------------------
+-----------------------------+-----------------
   118082 | sortkey             | Start
 | 2020-08-22 19:42:20.727049  |
   118078 | sortkey             | Start
 | 2020-08-22 19:43:54.728819  |
   118082 | sortkey             | Start
 | 2020-08-22 19:42:52.690264  |
   118072 | sortkey             | Start
 | 2020-08-22 19:44:14.793572  |
   118082 | sortkey             | Failed
 | 2020-08-22 19:42:20.728917  |
   118078 | sortkey             | Complete
 | 2020-08-22 19:43:54.792705  |  SORTKEY: None;
   118086 | sortkey             | Complete
 | 2020-08-22 19:42:00.72635   |  SORTKEY: None;
   118082 | sortkey             | Complete
 | 2020-08-22 19:43:34.728144  |  SORTKEY: None;
   118072 | sortkey             | Skipped:Retry exceeds the maximum limit for a table.
 | 2020-08-22 19:44:46.706155  |
```

```
  118086 | sortkey             | Start
| 2020-08-22 19:42:00.685255  |
  118082 | sortkey             | Start
| 2020-08-22 19:43:34.69531   |
  118072 | sortkey             | Start
| 2020-08-22 19:44:46.703331  |
  118082 | sortkey             | Checkpoint: progress 14.755079%
| 2020-08-22 19:42:52.692828  |
  118072 | sortkey             | Failed
| 2020-08-22 19:44:14.796071  |
  116723 | sortkey             | Abort:This table is not AUTO.
| 2020-10-28 05:12:58.479233  |
  110203 | distkey             | Abort:This table is not AUTO.
| 2020-10-28 05:45:54.67259   |
```

SYS_CONNECTION_LOG

记录身份验证尝试以及连接与断开连接。

SYS_CONNECTION_LOG 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
event	character(50)	连接或身份验证事件。
record_time	时间戳	事件发生的时间。
remote_host	character(45)	远程主机的名称或 IP 地址。
remote_port	character(32)	远程主机的端口号。
session_id	整数	与语句关联的进程 ID。
database_name	character(50)	数据库名称。
user_name	character(50)	用户名。

列名称	数据类型	描述
auth_method	character(32)	身份验证方法。
duration	integer	连接的持续时间（单位为微秒）。
ssl_version	character(50)	安全套接字层 (SSL) 版本。
ssl_cipher	character(128)	SSL 密码。
mtu	integer	最大传输单元 (MTU)。
ssl_compression	character(64)	SSL 压缩类型。
ssl_expansion	character(64)	SSL 扩展类型。
iam_auth_guid	character(36)	CloudTrail 请求的 IAM 身份验证 ID。
application_name	character(250)	会话应用程序的初始名称或更新名称。
driver_version	character(64)	ODBC 或 JDBC 驱动程序的版本，该版本可以从您的第三方 SQL 客户端工具连接到您的 Amazon Redshift 集群。
os_version	character(64)	连接到 Amazon Redshift 集群的客户端计算机上的操作系统版本。
plugin_name	character(32)	用于连接到您的 Amazon Redshift 集群的插件名称。

列名称	数据类型	描述
protocol_version	integer	Amazon Redshift 驱动程序在与服务器建立连接时使用的内部协议版本。协议版本是在驱动程序和服务器之间协商的。版本介绍了可用的功能。有效值包括： • 0 (BASE_SERVER_PROTOCOL_VERSION) • 1 (EXTENDED_RESULT_METADATA_SERVER_PROTOCOL_VERSION) – 为了保存每个查询的往返行程，服务器会发送额外的结果集元数据信息。 • 2 (BINARY_PROTOCOL_VERSION) – 根据结果集的数据类型，服务器以二进制格式发送数据。 • 3 (EXTENDED2_RESULT_METADATA_SERVER_PROTOCOL_VERSION) – 服务器发送的列信息区分大小写（排序）。
global_session_id	character(36)	当前会话的全局唯一标识符。节点故障重新启动时，会话 ID 仍然存在。

示例查询

要查看打开连接的详细信息，请运行以下查询。

```
select record_time, user_name, database_name, remote_host, remote_port
from sys_connection_log
where event = 'initiating session'
and session_id not in
(select session_id from sys_connection_log
where event = 'disconnecting session')
order by 1 desc;

record_time         | user_name   | database_name   | remote_host   | remote_port

--------------------+-------------+-----------------+---------------
+---------------------------------
2014-11-06 20:30:06 | rdsdb       | dev             | [local]       |

2014-11-06 20:29:37 | test001     | test            | 10.49.42.138  | 11111
```

```
2014-11-05 20:30:29 | rdsdb       | dev             | 10.49.42.138  | 33333

2014-11-05 20:28:35 | rdsdb       | dev             | [local]       |
(4 rows)
```

以下示例反映了一次失败的身份验证尝试和一次成功的连接与断开连接。

```
select event, record_time, remote_host, user_name
from sys_connection_log order by record_time;

            event      |         record_time        |  remote_host  | user_name

-----------------------+----------------------------+---------------+---------
authentication failure | 2012-10-25 14:41:56.96391  | 10.49.42.138  | john

authenticated          | 2012-10-25 14:42:10.87613  | 10.49.42.138  | john

initiating session     | 2012-10-25 14:42:10.87638  | 10.49.42.138  | john

disconnecting session  | 2012-10-25 14:42:19.95992  | 10.49.42.138  | john

(4 rows)
```

以下示例显示了 ODBC 驱动程序的版本、客户端计算机上的操作系统以及用于连接到 Amazon Redshift 集群的插件。在此示例中，使用的插件用于使用登录名和密码进行标准 ODBC 驱动程序身份验证。

```
select driver_version, os_version, plugin_name from sys_connection_log;

driver_version                          |  os_version                       |
 plugin_name
-----------------------------------------+-----------------------------------
+--------------------
Amazon Redshift ODBC Driver 1.4.15.0001 | Darwin 18.7.0 x86_64              | none
Amazon Redshift ODBC Driver 1.4.15.0001 | Linux 4.15.0-101-generic x86_64   | none
```

以下示例显示了客户端计算机上的操作系统版本、驱动程序版本和协议版本。

```
select os_version, driver_version, protocol_version from sys_connection_log;

os_version                      |  driver_version              | protocol_version
```

```
--------------------------------+------------------------------+--------------------
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
```

SYS_COPY_JOB（预览版）

这是面向预览版中的自动复制（SQL COPY JOB）的预发行文档。文档和特征都可能会更改。我们建议您仅在测试环境中使用此特征，不要在生产环境中使用。公开预览将于 2024 年 7 月 31 日结束。在预览结束两周后，将自动删除预览集群。有关预览条款和条件，请参阅 AWS 服务条款中的测试版和预览。

使用 SYS_COPY_JOB 查看 COPY JOB 命令的详细信息。

此视图包含已创建的 COPY JOB 命令。

SYS_COPY_JOB 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
job_id	bigint	复制作业标识符。
job_name	character(128)	复制作业的名称。
iam_role	character(128)	COPY 语句中指定的 IAM 角色。
job_text	character(256)	COPY 语句的参数。
is_auto	整数	指明 Amazon Redshift 是否自动运行 COPY JOB。1 表示 true，0 表示 false。
on_error_suspend	整数	此信息仅供内部使用。

SYS_COPY_REPLACEMENTS

显示当无效的 UTF-8 字符由带有 ACCEPTINVCHARS 选项的 COPY 命令替换时所记录的日志。对于每个至少需要一次替换的节点切片上的前 100 行，将针对每行向 SYS_COPY_REPLACEMENTS 添加一个日志条目。

您可以使用此视图查看有关无服务器工作组和预置集群的信息。

SYS_COPY_REPLACEMENTS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	生成该查询的用户的 ID。
query_id	bigint	查询 ID。用于联接其他系统表和视图的列。
table_id	integer	表 ID。
file_name	character (256)	COPY 命令的输入文件的完整路径。
column_name	character (127)	包含无效 UTF-8 字符的首个字段。
line_number	bigint	输入数据文件中包含无效 UTF-8 字符的行号。-1 表示行号不可用，例如从列式数据文件复制时。
raw_line	character (1024)	包含无效 UTF-8 字符的原始加载数据。

示例查询

以下示例返回最近的 COPY 操作中的替换。

```
select query_idp, table_id, file_name, line_number, colname
from sys_copy_replacements
```

```
where query = pg_last_copy_id();

 query_id | table_id |   file_name                          | line_number | column_name
 ---------+----------+--------------------------------------+-------------+--------
    96    |    26    | s3://mybucket/allusers_pipe.txt      |         123 | city
    96    |    26    | s3://mybucket/allusers_pipe.txt      |         456 | city
    96    |    26    | s3://mybucket/allusers_pipe.txt      |         789 | city
    96    |    26    | s3://mybucket/allusers_pipe.txt      |         012 | city
    96    |    26    | s3://mybucket/allusers_pipe.txt      |         119 | city
...
```

SYS_DATASHARE_CHANGE_LOG

记录用于跟踪创建器和使用者集群上的数据共享更改的综合视图。

SYS_DATASHARE_CHANGE_LOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	执行操作的用户的 ID。
user_name	varchar(128)	执行操作的用户的名称。
session_id	整数	会话 ID。
transaction_id	bigint	事务的 ID。
share_id	integer	受影响的数据共享的 ID。
share_name	varchar(128)	数据共享的名称。
source_database_id	integer	数据共享所属的数据库的 ID。

列名称	数据类型	描述
source_database_name	varchar(128)	数据共享所属的数据库的名称。
consumer_database_id	integer	从数据共享中导入的数据库的 ID。
consumer_database_name	varchar(128)	从数据共享导入的数据库的名称。
arn	varchar(192)	支持所导入数据库的资源的 ARN。
record_time	时间戳	操作的时间戳。
action	varchar(128)	正在运行的操作。可能的值有：CREATE DATASHARE、DROP DATASHARE、GRANT ALTER、REVOKE ALTER、GRANT SHARE、REVOKE SHARE、ALTER ADD、ALTER REMOVE、ALTER SET、GRANT USAGE、REVOKE USAGE、CREATE DATABASE、GRANT 或共享数据库上的 REVOKE USAGE、DROP SHARED DATABASE、ALTER SHARED DATABASE。
status	integer	操作的状态。可能的值为 SUCCESS 和 ERROR-ERROR CODE。
share_object_type	varchar(64)	在数据共享中添加或删除的数据库对象的类型。可能的值包括 schema、表、列、函数和视图。这是创建器集群的字段。
share_object_id	integer	在数据共享中添加或删除的数据库对象的 ID。这是创建器集群的字段。
share_object_name	varchar(128)	在数据共享中添加或删除的数据库对象的名称。这是创建器集群的字段。
target_user_type	varchar(16)	被授予权限的用户或组的类型。这是创建器集群和使用者集群的字段。

列名称	数据类型	描述
target_user_ID	整数	被授予权限的用户或组的 ID。这是创建器集群和使用者集群的字段。
target_user_name	varchar(128)	被授予权限的用户或组的名称。这是创建器集群和使用者集群的字段。
consumer_account	varchar(16)	数据使用者的账户 ID。这是创建器集群的字段。
consumer_namespace	varchar(64)	数据使用者账户的命名空间。这是创建器集群的字段。
producer_account	varchar(16)	数据共享所属的创建者账户的账户 ID。这是使用者集群的字段。
producer_namespace	varchar(64)	数据共享所属的创建者账户的命名空间。这是使用者集群的字段。
attribute_name	varchar(64)	数据共享或共享数据库的属性的名称。
attribute_value	varchar(128)	数据共享或共享数据库的属性值。
message	varchar(512)	操作失败时出现的错误消息。

示例查询

以下示例显示了 SYS_DATASHARE_CHANGE_LOG 视图。

```
SELECT DISTINCT action
FROM sys_datashare_change_log
WHERE share_object_name LIKE 'tickit%';

         action
 ----------------------
```

```
  "ALTER DATASHARE ADD"
```

SYS_DATASHARE_CROSS_REGION_USAGE

使用 SYS_DATASHARE_CROSS_REGION_USAGE 视图，可以获取跨区域数据共享查询所导致的跨区域数据传输使用情况的摘要。SYS_DATASHARE_CROSS_REGION_USAGE 汇总了段级别的详细信息。

SYS_DATASHARE_CROSS_REGION_USAGE 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
query_id	整数	查询的 ID。使用此值可以联接其他系统表和视图。
child_query_sequence	整数	重写的用户查询的顺序，从 1 开始。
segment_id	bigint	分段的编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。
start_time	time	数据传输开始的时间，使用 UTC。
end_time	time	数据传输结束的时间，使用 UTC。
transferred_data	bigint	从生产者区域向使用者区域传输的数据字节数。
source_region	char(25)	查询从中传输数据的生产者区域。

示例查询

以下示例显示了 SYS_DATASHARE_CROSS_REGION_USAGE 视图。

```
SELECT query, segment, transferred_data, source_region
```

```
from sys_datashare_cross_region_usage
where query = pg_last_query_id()
order by query,segment;

  query | segment | transferred_data | source_region
--------+---------+------------------+---------------
 200048 |       2 |          4194304 |    us-west-1
 200048 |       2 |          4194304 |    us-east-2
```

SYS_DATASHARE_USAGE_CONSUMER

记录数据共享的活动和使用情况。此视图仅与使用者集群相关。

SYS_DATASHARE_USAGE_CONSUMER 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	发出请求的用户的 ID。
session_id	整数	运行查询的领导节点进程的 ID。
transaction_id	bigint	当前事务的上下文。
request_id	varchar(50)	请求的 API 调用的唯一 ID。
request_type	varchar(25)	向创建器集群发出的请求的类型。
transaction_uid	varchar(50)	事务的唯一 ID。
record_time	时间戳	记录操作的时间。
status	integer	请求的 API 调用的状态。

列名称	数据类型	描述
error_message	varchar(512)	错误消息。

示例查询

以下示例显示了 SYS_DATASHARE_USAGE_CONSUMER 视图。

```
SELECT request_type, status, trim(error) AS error
FROM sys_datashare_usage_consumer

  request_type  | status | error_message
----------------+--------+---------------
 "GET RELATION" |   0    |
```

SYS_DATASHARE_USAGE_PRODUCER

记录数据共享的活动和使用情况。此视图仅与创建器集群有关。

SYS_DATASHARE_USAGE_PRODUCER 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
share_id	integer	数据共享的对象 ID (OID)。
share_name	varchar(128)	数据共享的名称。
request_id	varchar(50)	请求的 API 调用的唯一 ID。
request_type	varchar(25)	向创建器集群发出的请求的类型。

列名称	数据类型	描述
object_type	varchar(64)	从数据共享中共享的对象的类型。可能的值包括 schema、表、列、函数和视图。
object_oid	integer	从数据共享中共享的对象的 ID。
object_name	varchar(128)	从数据共享中共享的对象的名称。
consumer_account	varchar(16)	数据共享所共享到的使用者账户的账户。
consumer_namespace	varchar(64)	数据共享所共享到的使用者账户的命名空间。
consumer_transaction_uid	varchar(50)	使用者集群上语句的唯一事务 ID。
record_time	时间戳	记录操作的时间。
status	integer	数据共享的状态。
error_message	varchar(512)	错误消息。
consumer_region	char(64)	使用者集群所在的区域。

示例查询

以下示例显示了 SYS_DATASHARE_USAGE_PRODUCER 视图。

```
SELECT DISTINCT
FROM sys_datashare_usage_producer
WHERE object_name LIKE 'tickit%';

   request_type
```

```
------------------
  "GET RELATION"
```

SYS_EXTERNAL_QUERY_DETAIL

使用 SYS_EXTERNAL_QUERY_DETAIL 查看段级别查询的详细信息。每行代表来自特定 WLM 查询的一段，其中包含 Amazon S3 中处理的行数、处理的字节数以及外部表的分区信息等详细信息。此视图除了具有与外部查询处理相关的更多详细信息之外，每一行还将在 SYS_QUERY_DETAIL 视图中有一个相应条目。

SYS_EXTERNAL_QUERY_DETAIL 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	integer	提交查询的用户标识符。
query_id	bigint	外部查询的查询标识符。
transaction_id	bigint	事务标识符。
child_query_sequence	integer	重写的用户查询的顺序。从 0 开始，类似于 segment_id。
segment_id	integer	查询分段的分段标识符。
source_type	character(32)	查询的数据源类型，可能是针对 Redshift Spectrum 的 S3、针对联合查询的 PG。
start_time	timestamp	查询开始的时间。
end_time	timestamp	查询完成的时间。
duration	bigint	在查询上花费的时间（微秒）。

列名称	数据类型	描述
total_partitions	integer	Amazon S3 查询所需的分区数。
qualified_partitions	integer	Amazon S3 查询扫描的分区数。
scanned_file	bigint	扫描的 Amazon S3 文件数。
returned_rows	bigint	Amazon S3 查询的扫描行数，或联合查询返回的行数。
returned_bytes	bigint	Amazon S3 查询的扫描字节数或联合查询返回的字节数。
file_format	文本	Amazon S3 文件的文件格式。
file_location	文本	外部表的 Amazon S3 位置。
external_query_text	文本	联合查询的段级查询文本。
warning_message	character(4000)	查询运行的时候显示的警告消息。
table_name	character(136)	正在操作的步骤的表名。
is_recursive	character(1)	表示是否对子文件夹进行递归扫描。
is_nested	character(1)	表示是否访问嵌套列数据类型。
s3list_time	bigint	文件列出的持续时间（以毫秒为单位）。
get_partition_time	long	从 AWS Glue Data Catalog 和 Apache Hive 中为给定的外部对象列出和限定分区所花费的时间。

示例查询

以下查询显示了外部查询详细信息。

```
SELECT query_id,
       segment_id,
       start_time,
       end_time,
       total_partitions,
       qualified_partitions,
       scanned_files,
       returned_rows,
       returned_bytes,
       trim(external_query_text) query_text,
       trim(file_location) file_location
FROM sys_external_query_detail
ORDER BY query_id, start_time DESC
LIMIT 2;
```

示例输出。

```
 query_id | segment_id |         start_time         |          end_time
   | total_partitions | qualified_partitions | scanned_files | returned_rows |
 returned_bytes | query_text | file_location
----------+------------+----------------------------+----------------------------
+------------------+----------------------+---------------+---------------
+----------------+------------+---------------
   763251 |          0 | 2022-02-15 22:32:23.312448 | 2022-02-15 22:32:24.036023 |
          3 |                    3 |             3 |         38203 |        2683414 |
          |
   763254 |          0 | 2022-02-15 22:32:40.17103  | 2022-02-15 22:32:40.839313 |
          3 |                    3 |             3 |         38203 |        2683414 |
          |
```

SYS_EXTERNAL_QUERY_ERROR

您可以查询 SYS_EXTERNAL_QUERY_ERROR 系统视图，以获取有关 Redshift Spectrum 扫描错误的信息。SYS_EXTERNAL_QUERY_ERROR 显示所记录的错误的示例。每个查询默认显示 10 个条目。

SYS_EXTERNAL_QUERY_ERROR 对所有用户均可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	生成此行的用户的标识符。
query_id	bigint	生成此行的查询的查询标识符。
file_location	char(256)	查询的数据位置。
rowid	char(2100)	文件中错误的位置。各个 rowid 部分用 :（冒号）分隔，将来可能会添加其他部分。 *row_offset :row_group :row_id* row_offset 是文件中行的偏移量（以字节为单位），对于不支持的文件格式，设置为 -1。表划分为 row_group，每个组都有带不同 row_id 的行。
column_name	char(127)	查询返回的列的名称。
original_value	char(1024)	查询的原始值。
modified_value	char(1024)	根据查询中指定的数据处理配置选项返回的修改值。
trigger	char(128)	查询中指定的数据处理选项。
action	char(128)	与查询中指定的数据处理选项相关的操作。
action_value	char(128)	与查询中指定的数据处理选项相关的操作参数值。
error_code	整数	查询中指定的数据处理选项的结果代码。

示例查询

以下查询将返回所执行数据处理操作的行列表。

```
SELECT * FROM sys_external_query_error;
```

该查询返回的结果类似于以下内容。

```
  user_id   query_id  file_location                                  rowid
column_name           original_value             modified_value       trigger
action                action_value                error_code
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0
league_name           Barclays Premier League    Barclays Premier Lea UNSPECIFIED
TRUNCATE                                          156
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0
league_nspi           34595                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:1
league_nspi           34151                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2
league_name           Barclays Premier League    Barclays Premier Lea UNSPECIFIED
TRUNCATE                                          156
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2
league_nspi           33223                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3
league_name           Barclays Premier League    Barclays Premier Lea UNSPECIFIED
TRUNCATE                                          156
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3
league_nspi           32808                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:4
league_nspi           32790                      32767                UNSPECIFIED
OVERFLOW_VALUE                                    199
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:5
league_name           Spanish Primera Division   Spanish Primera Divi UNSPECIFIED
TRUNCATE                                          156
    100     1574007   s3://spectrum-uddh/league/spi_global_rankings.0:6
league_name           Spanish Primera Division   Spanish Primera Divi UNSPECIFIED
TRUNCATE                                          156
```

SYS_INTEGRATION_ACTIVITY

SYS_INTEGRATION_ACTIVITY 显示有关已完成集成运行的详细信息。

SYS_INTEGRATION_ACTIVITY 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

有关零 ETL 集成的信息，请参阅《Amazon Redshift 管理指南》中的使用零 ETL 集成。

表列

列名称	数据类型	描述
integration_id	character (128)	与集成关联的标识符。
target_database	character (128)	Amazon Redshift 中接收集成数据的数据库。
source	character (128)	集成的源数据。可能的类型包括 MySQL 和 PostgreSQL 。
checkpoint_name	character (128)	复制二进制日志坐标的检查点的名称。
checkpoint_type	character (16)	检查点的类型。可能的值包括：snapshot、cdc。
checkpoint_bytes	bigint	此检查点中的字节数。
last_commit_timestamp	时间戳	上次在此检查点中提交的时间戳。
modified_tables	整数	在检查点中修改的表的数量。
integration_start_time	time	此检查点的集成开始时间 (UTC)。
integration_end_time	time	此检查点的集成结束时间 (UTC)。

示例查询

以下 SQL 命令显示集成日志。

```
select * from sys_integration_activity;

          integration_id              | target_database | source |
   checkpoint_name                   | checkpoint_type  | checkpoint_bytes |
 last_commit_timestamp   | modified_tables |   integration_start_time   |
 integration_end_time
-------------------------------------+-----------------+--------
+---------------------------------------------+------------------+------------------
+-------------------------+-----------------+---------------------------
+----------------------------
 76b15917-afae-4447-b7fd-08e2a5acce7b |   demo1         | MySQL   | checkpoints/
checkpoint_3_241_3_510.json     |        cdc       |        762       | 2023-05-10
 23:00:14.201 |         1       | 2023-05-10 23:00:45.054265 | 2023-05-10
 23:00:46.339826
 76b15917-afae-4447-b7fd-08e2a5acce7b |   demo1         | MySQL   | checkpoints/
checkpoint_3_16329_3_17839.json |        cdc       |       13488      | 2023-05-11
 01:33:57.411 |         2       | 2023-05-11 02:19:09.440121 | 2023-05-11
 02:19:16.090492
 76b15917-afae-4447-b7fd-08e2a5acce7b |   demo1         | MySQL   | checkpoints/
checkpoint_3_5103_3_5532.json   |        cdc       |        1657      | 2023-05-10
 23:13:14.205 |         2       | 2023-05-10 23:13:23.545487 | 2023-05-10
 23:13:25.652144
```

SYS_INTEGRATION_TABLE_STATE_CHANGE

SYS_INTEGRATION_TABLE_STATE_CHANGE 显示有关集成的表状态更改日志的详细信息。

超级用户可以查看此表中的所有行。

有关更多信息，请参阅使用零 ETL 集成。

表列

列名称	数据类型	描述
integration_id	character (128)	与集成关联的标识符。

列名称	数据类型	描述
database_name	character (128)	Amazon Redshift 数据库的名称。
schema_name	character (128)	Amazon Redshift 架构的名称。
table_name	character (128)	表的名称。
new_state	character (128)	表的状态。可能的值包括 Synced、ResyncRequired 、ResyncInitiated 、Deleted、Failed 和 ResyncDeleted 。
table_last_replicated_checkpoint	character (128)	当前同步的日志坐标。
state_change_reason	character (256)	最后一次状态转换的原因。
record_time	时间戳	更新此记录的时间（UTC）。

示例查询

以下 SQL 命令显示集成日志。

```
select * from sys_integration_table_state_change;

            integration_id            | database_name | schema_name | table_name
 | new_state |  table_last_replicated_checkpoint   | state_change_reason |
 record_time
--------------------------------------+---------------+-------------+------------
+-----------+--------------------------------------+---------------------
+----------------------------
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest79   |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 19:39:50.087868
```

```
99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest56   |
Synced    | {"txn_seq":9834,"txn_id":126597515} |                    | 2023-09-20
19:39:45.54005
99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest50   |
Synced    | {"txn_seq":9834,"txn_id":126597515} |                    | 2023-09-20
19:40:20.362504
99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest18   |
Synced    | {"txn_seq":9834,"txn_id":126597515} |                    | 2023-09-20
19:40:32.544084
99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest40t3s | sbtest23   |
Synced    | {"txn_seq":9834,"txn_id":126597515} |                    | 2023-09-20
15:49:05.186209
```

SYS_LOAD_DETAIL

返回用于跟踪或排查数据加载的信息。

此视图记录了在每个数据文件加载到数据库表中时的进度。

此视图对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	生成该条目的用户 ID。
query_id	整数	查询 ID。
file_name	character(256)	要加载的文件名。
bytes_scanned	整数	从 Amazon S3 内的文件中扫描到的字节数。
lines_scanned	integer	从加载文件中扫描的行数。此数字可能与实际加载的行数不匹配。例如，根据 COPY 命令中的 MAXERROR 选项，加载操作可能会扫描但会容忍大量的错误记录。
record_time	时间戳	上次更新此条目的时间。

列名称	数据类型	描述
splits_scanned	此文件的拆分次数。	此文件的拆分次数。
start_time	时间戳	开始此文件处理的时间。
end_time	时间戳	完成此文件处理的时间。

示例查询

以下示例返回上次 COPY 操作的详细信息。

```
select query_id, trim(file_name) as file, record_time
from sys_load_detail
where query_id = pg_last_copy_id();

 query_id |               file                |          record_time
----------+-----------------------------------+----------------------------
 28554    | s3://dw-tickit/category_pipe.txt | 2013-11-01 17:14:52.648486
(1 row)
```

以下查询包含 TICKIT 数据库中初次加载表时的条目：

```
select query_id, trim(file_name), record_time
from sys_load_detail
where file_name like '%tickit%' order by query_id;

 query_id |           btrim           |          record_time
----------+---------------------------+----------------------------
 22475    | tickit/allusers_pipe.txt | 2013-02-08 20:58:23.274186
 22478    | tickit/venue_pipe.txt    | 2013-02-08 20:58:25.070604
 22480    | tickit/category_pipe.txt | 2013-02-08 20:58:27.333472
 22482    | tickit/date2008_pipe.txt | 2013-02-08 20:58:28.608305
 22485    | tickit/allevents_pipe.txt| 2013-02-08 20:58:29.99489
 22487    | tickit/listings_pipe.txt | 2013-02-08 20:58:37.632939
 22593    | tickit/allusers_pipe.txt | 2013-02-08 21:04:08.400491
 22596    | tickit/venue_pipe.txt    | 2013-02-08 21:04:10.056055
 22598    | tickit/category_pipe.txt | 2013-02-08 21:04:11.465049
 22600    | tickit/date2008_pipe.txt | 2013-02-08 21:04:12.461502
 22603    | tickit/allevents_pipe.txt| 2013-02-08 21:04:14.785124
```

```
 22605    | tickit/listings_pipe.txt | 2013-02-08 21:04:20.170594

(12 rows)
```

向此系统视图的日志文件写入一个记录的事实并不表示加载操作已作为其所属的事务的一部分成功提交。要验证加载提交，请查询 STL_UTILITYTEXT 视图并查找与 COPY 事务对应的 COMMIT 记录。例如，此查询基于针对 STL_UTILITYTEXT 的子查询联接 SYS_LOAD_DETAIL 和 STL_QUERY：

```
select l.query_id,rtrim(l.file_name),q.xid
from sys_load_detail l, stl_query q
where l.query_id=q.query
and exists
(select xid from stl_utilitytext where xid=q.xid and rtrim("text")='COMMIT');

 query_id |           rtrim           |  xid
----------+---------------------------+-------
 22600    | tickit/date2008_pipe.txt  | 68311
 22480    | tickit/category_pipe.txt  | 68066
  7508    | allusers_pipe.txt         | 23365
  7552    | category_pipe.txt         | 23415
  7576    | allevents_pipe.txt        | 23429
  7516    | venue_pipe.txt            | 23390
  7604    | listings_pipe.txt         | 23445
 22596    | tickit/venue_pipe.txt     | 68309
 22605    | tickit/listings_pipe.txt  | 68316
 22593    | tickit/allusers_pipe.txt  | 68305
 22485    | tickit/allevents_pipe.txt | 68071
  7561    | allevents_pipe.txt        | 23429
  7541    | category_pipe.txt         | 23415
  7558    | date2008_pipe.txt         | 23428
 22478    | tickit/venue_pipe.txt     | 68065
   526    | date2008_pipe.txt         |  2572
  7466    | allusers_pipe.txt         | 23365
 22482    | tickit/date2008_pipe.txt  | 68067
 22598    | tickit/category_pipe.txt  | 68310
 22603    | tickit/allevents_pipe.txt | 68315
 22475    | tickit/allusers_pipe.txt  | 68061
   547    | date2008_pipe.txt         |  2572
 22487    | tickit/listings_pipe.txt  | 68072
  7531    | venue_pipe.txt            | 23390
  7583    | listings_pipe.txt         | 23445
(25 rows)
```

SYS_LOAD_ERROR_DETAIL

使用 SYS_LOAD_ERROR_DETAIL 查看 COPY 命令错误的详细信息。每行代表一个 COPY 命令。它包含正在运行和已完成的 COPY 命令。

SYS_LOAD_ERROR_DETAIL 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	integer	提交副本用户的标识符。
query_id	bigint	副本的查询标识符。
transaction_id	bigint	事务标识符。
session_id	integer	运行副本的进程的进程标识符。
database_name	character(64)	发起复制时用户连接到其中的数据库的名称。
table_id	integer	表标识符。
start_time	timestamp	复制开始的时间 (UTC)。
file_name	character(256)	要加载的输入文件的完整路径。
line_number	bigint	加载文件中出现错误的行号。加载 JSON 文件时，包含错误的 JSON 对象的最后一行的行号。
column_name	character(127)	存在错误的字段。
column_type	character(10)	存在错误的字段的数据类型。

列名称	数据类型	描述
column_length	character(10)	列长度（如果适用）。当数据类型具有限制长度时填充此字段。例如，对于数据类型为“character(3)”的列，此列将包含值“3”。
position	integer	字段中错误的位置。
error_code	integer	错误代码。
error_message	character(512)	错误的说明。

示例查询

以下查询显示了特定查询的复制命令的加载错误详情。

```
SELECT query_id,
       table_id,
       start_time,
       trim(file_name) AS file_name,
       trim(column_name) AS column_name,
       trim(column_type) AS column_type,
       trim(error_message) AS error_message
FROM sys_load_error_detail
WHERE query_id = 762949
ORDER BY start_time
LIMIT 10;
```

示例输出。

```
 query_id | table_id |         start_time         |               file_name
      | column_name | column_type |                 error_message
----------+----------+----------------------------
+-----------------------------------------+-------------+-------------
+-----------------------------------------------
   762949 |   137885 | 2022-02-15 22:14:46.759151 | s3://load-test/copyfail/
wrong_format_000 | id          | int4        | Invalid digit, Value 'a', Pos 0, Type:
 Integer
```

```
   762949 |   137885 | 2022-02-15 22:14:46.759151 | s3://load-test/copyfail/
wrong_format_001 | id          | int4        | Invalid digit, Value 'a', Pos 0, Type:
 Integer
```

SYS_LOAD_HISTORY

使用 SYS_LOAD_HISTORY 查看 COPY 命令的详细信息。每行代表一个 COPY 命令，其中包含某些字段的累积统计数据。它包含正在运行和已完成的 COPY 命令。

SYS_LOAD_HISTORY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	integer	提交副本用户的标识符。
query_id	bigint	副本的查询标识符。
transaction_id	bigint	事务标识符。
session_id	integer	运行副本的进程的进程标识符。
database_name	文本	在发起操作时用户连接到其中的数据库的名称。
status	文本	副本的状态。有效值为 `running`、`completed` 、`aborted`。
table_name	文本	复制到其中的表名称。
start_time	timestamp	复制开始的时间。
end_time	timestamp	复制完成的时间。
duration	bigint	在 COPY 命令中花费的时间（微秒）。

列名称	数据类型	描述
data_source	文本	要复制的文件输入的 Amazon S3 位置。
file_format	文本	源文件格式。格式包括 csv、txt、json、avro、orc 或 parquet。
Loaded_rows	bigint	复制到表的行数。
Loaded_bytes	bigint	复制到表的字节数。
source_file_count	integer	源文件中的文件数目。
source_file_bytes	bigint	源文件中的字节数。
file_count_scanned	整数	从 Amazon S3 中扫描到的文件的数量。
file_bytes_scanned	bigint	从 Amazon S3 内的文件中扫描到的字节数。
error_count	bigint	错误计数。
copy_job_id	bigint	复制作业标识符。0 表示没有作业标识符。

示例查询

以下查询显示了特定复制命令的加载行、字节、表和数据源。

```
SELECT query_id,
       table_name,
       data_source,
       loaded_rows,
       loaded_bytes
FROM sys_load_history
WHERE query_id IN (6389,490791,441663,74374,72297)
ORDER BY query_id,
```

```
         data_source DESC;
```

示例输出。

```
 query_id |    table_name    |                                data_source
              | loaded_rows | loaded_bytes
----------+------------------
+-------------------------------------------------------------------------+-------------
+---------------
     6389 | store_returns    | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
store_returns/    |   287999764 | 1196240296158
    72297 | web_site         | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
web_site/         |          54 |         43808
    74374 | ship_mode        | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
ship_mode/        |          20 |          1320
   441663 | income_band      | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
income_band/      |          20 |          2152
   490791 | customer_address | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
customer_address/ |     6000000 |     722924305
```

以下查询显示了复制命令的加载行、字节、表和数据源。

```
SELECT query_id,
       table_name,
       data_source,
       loaded_rows,
       loaded_bytes
FROM sys_load_history
ORDER BY query_id DESC
LIMIT 10;
```

示例输出。

```
 query_id |       table_name       |                                  data_source
                          | loaded_rows |  loaded_bytes
----------+------------------------
+-------------------------------------------------------------------------
+-------------+-----------------
   491058 | web_site               | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_site/               |          54 |           43808
   490947 | web_sales              | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_sales/              |   720000376 |  22971988122819
```

```
   490923 | web_returns            | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_returns/           |    71997522 |     96597496325
   490918 | web_page               | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_page/              |        3000 |            1320
   490907 | warehouse              | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/warehouse/             |          20 |            1320
   490902 | time_dim               | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/time_dim/              |       86400 |            1320
   490876 | store_sales            | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/store_sales/           |  2879987999 | 151666241887933
   490870 | store_returns          | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/store_returns/         |   287999764 |   1196405607941
   490865 | store                  | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/store/                 |        1002 |          365507
```

以下查询显示了复制命令的每日加载行和字节。

```
SELECT date_trunc('day',start_time) AS exec_day,
       SUM(loaded_rows) AS loaded_rows,
       SUM(loaded_bytes) AS loaded_bytes
FROM sys_load_history
GROUP BY exec_day
ORDER BY exec_day DESC;
```

示例输出。

```
      exec_day       | loaded_rows |   loaded_bytes
---------------------+-------------+------------------
 2022-01-20 00:00:00 |  6347386005 |  258329473070606
 2022-01-19 00:00:00 | 19042158015 |  775198502204572
 2022-01-18 00:00:00 | 38084316030 | 1550294469446883
 2022-01-17 00:00:00 | 25389544020 | 1033271084791724
 2022-01-16 00:00:00 | 19042158015 |  775222736252792
 2022-01-15 00:00:00 | 19834245387 |  798122849155598
 2022-01-14 00:00:00 | 75376544688 | 3077040926571384
```

SYS_MV_REFRESH_HISTORY

结果包括有关所有实体化视图的刷新历史记录的信息。结果包括刷新类型（例如手动或自动）以及最近刷新的状态。

SYS_MV_REFRESH_HISTORY 对所有用户均可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	提交刷新的用户的标识符。
session_id	整数	运行实体化视图刷新的进程的进程标识符。
transaction_id	bigint	事务标识符。
database_name	char(128)	包含实体化视图的数据库。
schema_name	char(128)	实体化视图的架构。
mv_id	bigint	实体化视图的对象 ID。
mv_name	char(128)	实体化视图名称。
refresh_type	char(32)	刷新的类型，例如手动或自动。
status	text	刷新的状态。有关状态的详细信息，请参阅 SVL_MV_REFRESH_STATUS 的状态列。
start_time	时间戳	刷新的开始时间。
end_time	时间戳	刷新的结束时间。
duration	bigint	刷新实体化视图所用的时间（以微秒为单位）。

示例查询

以下查询显示了实体化视图的刷新历史记录。

```
SELECT user_id,
     session_id,
     transaction_id,
     database_name,
     schema_name,
     mv_id,
     mv_name,
     refresh_type,
     status,
     start_time,
     end_time,
     duration
     from sys_mv_refresh_history;
```

查询返回以下示例输出：

```
 user_id | session_id | transaction_id | database_name | schema_name                |
 mv_id  |  mv_name           |  refresh_type  |  status
         |  start_time                |  end_time                  |  duration
---------+------------+----------------+---------------
+----------------------------+--------+--------------------+----------------
+-----------------------------------------------+---------------------------
+----------------------------+-----------
       1 | 1073815659 |          15066 | dev           | test_stl_mv_refresh_schema |
 203762 | mv_incremental     | Manual         | MV was already updated
         | 2023-10-26 15:59:20.952179 | 2023-10-26 15:59:20.952866 |      687
       1 | 1073815659 |          15068 | dev           | test_stl_mv_refresh_schema |
 203771 | mv_nonincremental  | Manual         | MV was already updated
         | 2023-10-26 15:59:21.008049 | 2023-10-26 15:59:21.008658 |      609
       1 | 1073815659 |          15070 | dev           | test_stl_mv_refresh_schema |
 203779 | mv_refresh_error   | Manual         | MV was already updated
         | 2023-10-26 15:59:21.064252 | 2023-10-26 15:59:21.064885 |      633
       1 | 1073815659 |          15074 | dev           | test_stl_mv_refresh_schema
 | 203762 | mv_incremental     | Manual         | Refresh successfully updated MV
 incrementally   | 2023-10-26 15:59:29.693329 | 2023-10-26 15:59:43.482842 | 13789513
       1 | 1073815659 |          15076 | dev           | test_stl_mv_refresh_schema |
 203771 | mv_nonincremental  | Manual         | Refresh successfully recomputed MV from
 scratch | 2023-10-26 15:59:43.550184 | 2023-10-26 15:59:47.880833 |  4330649
       1 | 1073815659 |          15078 | dev           | test_stl_mv_refresh_schema |
 203779 | mv_refresh_error   | Manual         | Refresh failed due to an internal error
         | 2023-10-26 15:59:47.949052 | 2023-10-26 15:59:52.494681 |  4545629
(6 rows)
```

SYS_MV_STATE

结果包括有关所有实体化视图的状态的信息。这包括基表信息、架构属性和有关最近事件（例如删除列）的信息。

SYS_MV_STATE 对所有用户均可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	bigint	创建事件的用户的 ID。
transaction_id	bigint	事件的事务 ID。
database_name	char(128)	包含实体化视图的数据库。
event_desc	char(500)	提示状态更改的事件。示例值包括： • 列类型已更改 • 列已被删除 • 列已重命名 • 架构名称已更改 • 小型表转换 • TRUNCATE • Vacuum 请注意，此列有其它可能的值。
start_time	时间戳	事件的开始时间。
base_table_database_name	char(128)	基表的数据库名称。
base_table_schema	char(128)	基表的架构。

列名称	数据类型	描述
base_table_name	char(128)	基表的名称。
mv_schema	char(128)	实体化视图的架构。
mv_name	char(128)	实体化视图的名称。
state	character(32)	实体化视图的已更改状态，如下所示： • 重新计算 • 无法刷新

示例查询

以下查询显示了实体化视图状态。

```
select * from sys_mv_state;
```

查询返回以下示例输出：

```
 user_id | transaction_id | database_name | event_desc                  | start_time
              | base_table_database_name | base_table_schema | base_table_name     |
 mv_schema  | mv_name                    | state
---------+----------------+---------------+-----------------------------
+-----------------------------+-------------------------+------------------
+---------------------+-------------+---------------------------+--------------
 106     | 12720          | tickit_db     | TRUNCATE                    | 2023-07-26
 14:59:12.788268 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_a1f3f862 | Recompute
 106     | 12724          | tickit_db     | ALTER TABLE ALTER DISTSTYLE | 2023-07-26
 14:59:51.409014 | tickit_db                | mv_schema         | test_table_58102435 |
 mv_schema   | materialized_view_ca746631 | Recompute
 106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
 14:59:12.822928 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_5750a8d4 | Unrefreshable
 106     | 12727          | tickit_db     | Table was renamed           | 2023-07-26
 15:00:08.051244 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema   | materialized_view_5750a8d4 | Unrefreshable
```

```
106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
14:59:12.857755 | tickit_db                 | mv_schema         | test_table_95d6d861 |
mv_schema   | materialized_view_5750a8d4 | Unrefreshable
106     | 12727          | tickit_db     | Table was renamed           | 2023-07-26
15:00:08.051358 | tickit_db                 | mv_schema         | test_table_95d6d861 |
mv_schema   | materialized_view_5ef0d754 | Unrefreshable
106     | 12720          | tickit_db     | TRUNCATE                    | 2023-07-26
14:59:12.788159 | tickit_db                 | mv_schema         | test_table_95d6d861 |
mv_schema   | materialized_view_5750a8d4 | Recompute
106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
14:59:12.857799 | tickit_db                 | mv_schema         | test_table_95d6d861 |
mv_schema   | materialized_view_a1f3f862 | Unrefreshable
106     | 12720          | tickit_db     | TRUNCATE                    | 2023-07-26
14:59:12.788327 | tickit_db                 | mv_schema         | test_table_95d6d861 |
mv_schema   | materialized_view_5ef0d754 | Recompute
106     | 12727          | tickit_db     | ALTER TABLE ALTER SORTKEY   | 2023-07-26
15:00:08.006235 | tickit_db                 | mv_schema         | test_table_58102435 |
mv_schema   | materialized_view_ca746631 | Recompute
106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
14:59:12.82297  | tickit_db                 | mv_schema         | test_table_95d6d861 |
mv_schema   | materialized_view_a1f3f862 | Unrefreshable
106     | 12727          | tickit_db     | Table was renamed           | 2023-07-26
15:00:08.051321 | tickit_db                 | mv_schema         | test_table_95d6d861 |
mv_schema   | materialized_view_a1f3f862 | Unrefreshable
```

SYS_PROCEDURE_CALL

使用 SYS_PROCEDURE_CALL 视图以获取有关存储过程调用的信息，包括开始时间、结束时间、存储过程调用的状态以及嵌套存储过程调用的调用层次结构。每个存储过程调用接受一个查询 ID。

SYS_PROCEDURE_CALL 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
session_user_id	整数	用户的标识符，该用户创建了会话并且是顶级存储过程调用的调用方。

列名称	数据类型	描述
security_user_id	整数	用户的标识符，该用户的权限用于在存储过程中运行语句。如果存储过程是 DEFINER，则这将是存储过程的拥有者 user_id。
query_id	整数	存储过程调用的查询标识符。
query_text	char(4000)	存储过程调用查询的文本。
start_time	时间戳	查询开始运行的时间（采用 UTC 表示）。时间戳使用六位数精度表示小数秒，例如：2009-06-12 11:29:19.131358。
end_time	时间戳	查询完成运行的时间（采用 UTC 表示）。时间戳使用六位数精度表示小数秒，例如：2009-06-12 11:29:19.131358。
status	char(10)	存储过程调用的状态。当系统停止存储过程或用户取消存储过程时，该值将为已取消。如果存储过程调用运行至完成，则值为成功。
caller_procedure_query_id	整数	如果对存储过程调用的调用是由其它存储过程调用执行的，则此列包含外部调用的查询 ID。否则该字段为 NULL。

示例查询

以下查询返回嵌套存储过程调用的层次结构。

```
select query_id, datediff(seconds, start_time, end_time) as elapsed_time, status,
 trim(query_text) as call, caller_procedure_query_id from sys_procedure_call;
```

示例输出。

```
 query_id | elapsed_time | status  |                      call                      |
 caller_procedure_query_id
----------+--------------+---------+------------------------------------------------
+---------------------------
     3087 |           18 | success | CALL proc_bd906c98c45443ffa165e9552056902d(1)    |
          3085
     3085 |           18 | success | CALL proc_bd906c98c45443ffa165e9552056902d_2(1); |

(2 rows)
```

SYS_PROCEDURE_MESSAGES

SYS_PROCEDURE_MESSAGES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
transaction_id	bigint	事务标识符。
query_id	整数	存储过程调用的查询标识符。
record_time	时间戳	生成消息时的时间（采用 UTC 表示）。
log_level	char(10)	所生成的消息的日志级别。可能的值为 LOG、INFO、NOTICE、WARNING 和 EXCEPTION。
消息	char(1024)	生成的消息的文本。
line_number	整数	生成的消息的行号。

示例查询

以下查询显示了 SYS_PROCEDURE_MESSAGES 的示例输出。

```
select transaction_id, query_id, record_time, log_level, trim(message), line_number
 from sys_procedure_messages;
```

```
transaction_id | query_id |        record_time         | log_level |           btrim
        | line_number
---------------+----------+----------------------------+-----------
+---------------------------+-------------
     25267     |   80562  | 2023-07-17 14:38:31.910136 |   NOTICE  |
 test_notice_msg_b9f1e749  |     8
     25267     |   80562  | 2023-07-17 14:38:31.910002 |    LOG    |
 test_log_msg_833c7420     |     6
     25267     |   80562  | 2023-07-17 14:38:31.910111 |    INFO   |
 test_info_msg_651373d9    |     7
     25267     |   80562  | 2023-07-17 14:38:31.910154 |   WARNING |
 test_warning_msg_831c5747 |     9
(4 rows)
```

SYS_QUERY_DETAIL

使用 SYS_QUERY_DETAILS 在步骤级别查看查询的详细信息。每一行代表特定 WLM 查询的一个步骤，其中包含详细信息。此视图包含许多类型的查询，例如 DDL、DML 和实用程序命令（例如：复制和卸载）。根据查询类型，某些列可能不相关。例如，external_scanned_bytes 与内部表无关。

SYS_QUERY_DETAIL 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	integer	提交查询的用户标识符。
query_id	bigint	查询标识符。
child_query_sequence	整数	重写的用户查询的顺序，从 1 开始。

列名称	数据类型	描述
stream_id	整数	查询流的流标识符。
segment_id	整数	查询运行分段的分段标识符。
step_id	integer	分段中的步骤标识符。
step_name	文本	分段中的步骤名称。可能的值为 aggregate 、broadcast 、delete、distribut e 、hash、hashjoin、insert、li 、unique 和 window。
table_id	integer	永久性表扫描的表标识符。
table_name	character(136)	正在操作的步骤的表名。
is_rrscan	字符	指示步骤是否为扫描步骤的值。True (t) 则表示使用了限制范围的扫描。
start_time	时间戳	查询步骤开始的时间。
end_time	时间戳	查询步骤完成的时间。
duration	bigint	在步骤上花费的时间（微秒）。
警报	文本	提示事件的描述。
input_bytes	bigint	当前步骤的输入字节。
input_rows	bigint	当前步骤的输入行。
output_bytes	bigint	当前步骤的输出字节。
output_rows	bigint	当前步骤的输出行。
blocks_read	bigint	步骤读取的数据块数。

列名称	数据类型	描述
blocks_write	bigint	步骤写入的数据块数。
local_read_IO	bigint	从本地磁盘缓存中读取的数据块的数量。
remote_read_IO	bigint	从远程读取的数据块数。
源	文本	扫描的数据库对象的类型。只有当行的 step_name 值为 `scan` 时，此列才有值。
data_skewness	整数	所有步骤间输出行分布的偏度。这是一个介于 0% 到 100% 之间的数字。该数字越大，分布越不平衡。
time_skewness	整数	所有步骤间执行时间分布的偏度。这是一个介于 0% 到 100% 之间的数字。该数字越大，分布越不平衡。
is_active	字符	步骤级别的查询状态。可能的值为“t”（表示步骤正在运行）或“f”（表示步骤已完成运行）。
spilled_block_local_disk	bigint	溢出到本地磁盘的块数。
spilled_block_remote_disk	bigint	溢出到 Amazon Simple Storage Service 的块数。
step_attribute	character(64)	包含相关步骤的信息。扫描步骤的可能值：`multi-dimensional` 。

示例查询

以下示例返回了 SYS_QUERY_DETAIL 的输出。

以下查询显示了步骤级别的查询元数据详细信息，包括步骤名称、input_bytes、output_bytes、input_rows、output_rows。

```
SELECT query_id,
       child_query_sequence,
       stream_id,
       segment_id,
       step_id,
       trim(step_name) AS step_name,
       duration,
       input_bytes,
       output_bytes,
       input_rows,
       output_rows
FROM sys_query_detail
WHERE query_id IN (193929)
ORDER BY query_id,
         stream_id,
         segment_id,
         step_id DESC;
```

示例输出。

```
 query_id | child_query_sequence | stream_id | segment_id | step_id | step_name  |
 duration     | input_bytes | output_bytes | input_rows | output_rows
----------+----------------------+-----------+------------+---------+------------
+-----------------+-------------+--------------+------------+-------------
   193929 |                    2 |         0 |          0 |       3 | hash       |
      37144 |           0 |      9350272 |          0 |      292196
   193929 |                    5 |         0 |          0 |       3 | hash       |
       9492 |           0 |        23360 |          0 |        1460
   193929 |                    1 |         0 |          0 |       3 | hash       |
      46809 |           0 |      9350272 |          0 |      292196
   193929 |                    4 |         0 |          0 |       2 | return     |
       7685 |           0 |          896 |          0 |         112
   193929 |                    1 |         0 |          0 |       2 | project    |
      46809 |           0 |            0 |          0 |      292196
   193929 |                    2 |         0 |          0 |       2 | project    |
      37144 |           0 |            0 |          0 |      292196
```

```
193929 |                   5 |         0 |          0 |       2 | project    |
    9492 |           0 |           0 |          0 |        1460
193929 |                   3 |         0 |          0 |       2 | return     |
   11033 |           0 |       14336 |          0 |         112
193929 |                   2 |         0 |          0 |       1 | project    |
   37144 |           0 |           0 |          0 |      292196
193929 |                   1 |         0 |          0 |       1 | project    |
   46809 |           0 |           0 |          0 |      292196
193929 |                   5 |         0 |          0 |       1 | project    |
    9492 |           0 |           0 |          0 |        1460
193929 |                   3 |         0 |          0 |       1 | aggregate  |
   11033 |           0 |      201488 |          0 |          14
193929 |                   4 |         0 |          0 |       1 | aggregate  |
    7685 |           0 |       28784 |          0 |          14
193929 |                   5 |         0 |          0 |       0 | scan       |
    9492 |           0 |       23360 |     292196 |        1460
193929 |                   4 |         0 |          0 |       0 | scan       |
    7685 |           0 |        1344 |        112 |         112
193929 |                   2 |         0 |          0 |       0 | scan       |
   37144 |           0 |     7304900 |     292196 |      292196
193929 |                   3 |         0 |          0 |       0 | scan       |
   11033 |           0 |       13440 |        112 |         112
193929 |                   1 |         0 |          0 |       0 | scan       |
   46809 |           0 |     7304900 |     292196 |      292196
193929 |                   5 |         0 |          0 |      -1 |            |
    9492 |       12288 |           0 |          0 |         0
193929 |                   1 |         0 |          0 |      -1 |            |
   46809 |       16384 |           0 |          0 |         0
193929 |                   2 |         0 |          0 |      -1 |            |
   37144 |       16384 |           0 |          0 |         0
193929 |                   4 |         0 |          0 |      -1 |            |
    7685 |       28672 |           0 |          0 |         0
193929 |                   3 |         0 |          0 |      -1 |            |
   11033 |      114688 |           0 |          0 |         0
```

要按从最常用到最不常用的顺序查看数据库中的表，请使用以下示例。将 *sample_data_dev* 替换为您自己的数据库。请注意，此查询将计算从创建集群时开始的查询数量，但当您的数据仓库空间不足时，不会保存系统视图数据。

```
SELECT table_name, COUNT (DISTINCT query_id)
FROM SYS_QUERY_DETAIL
WHERE table_name LIKE 'sample_data_dev%'
GROUP BY table_name
```

```
ORDER BY COUNT(*) DESC;

+---------------------------------+-------+
|           table_name            | count |
+---------------------------------+-------+
| sample_data_dev.tickit.venue    |     4 |
| sample_data_dev.myunload1.venue |     3 |
| sample_data_dev.tickit.listing  |     1 |
| sample_data_dev.tickit.category |     1 |
| sample_data_dev.tickit.users    |     1 |
| sample_data_dev.tickit.date     |     1 |
| sample_data_dev.tickit.sales    |     1 |
| sample_data_dev.tickit.event    |     1 |
+---------------------------------+-------+
```

SYS_QUERY_HISTORY

使用 SYS_QUERY_HISTORY 查看用户查询的详细信息。每行代表一个用户查询，其中包含某些字段的累积统计数据。此视图包含许多类型的查询，例如数据定义语言 (DDL)、数据操作语言 (DML)、复制、卸载和 Amazon Redshift Spectrum。它包含正在运行和已完成的查询。

SYS_QUERY_HISTORY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	integer	提交查询的用户标识符。
query_id	bigint	查询标识符。
query_label	character(320)	查询的短名称。
transaction_id	bigint	事务标识符。
session_id	integer	运行查询进程的进程标识符。
database_name	character(128)	在发起查询时用户连接到的数据库的名称。

列名称	数据类型	描述
query_type	character(32)	查询类型，例如，SELECT、INSERT、UPDATE、UNLOAD、COPY、COMMAND、DDL、UTILITY、CTAS 和 OTHER。
status	character(10)	查询的状态。有效值：planning、queued、running、returning、failed、canceled 和 success。
result_cache_hit	布尔值	指示查询是否匹配结果缓存。
start_time	timestamp	查询开始的时间。
end_time	timestamp	查询完成的时间。
elapsed_time	bigint	在查询上花费的总时间（微秒）。
queue_time	bigint	在服务类查询队列上花费的总时间（微秒）。
execution_time	bigint	在服务类中运行的总时间（微秒）。
error_message	character(512)	查询失败的原因。
returned_rows	bigint	返回到客户端的行数。
returned_bytes	bigint	返回到客户端的字节数。
query_text	character(4000)	查询字符串。此字符串可能会被截断。
redshift_version	character(256)	查询运行时的 Amazon Redshift 版本。

列名称	数据类型	描述
usage_limit	character(150)	查询达到的使用限制 ID 的列表。
compute_type	varchar(32)	指示查询运行在主集群还是并发扩展集群上。可能的值为 primary（查询在主集群上运行）、secondary （查询在辅助集群上运行）或 primary-scale （查询在并发集群上运行）。这仅适用于预调配的集群。
compile_time	bigint	在编译上花费的总时间（微秒）。
planning_time	bigint	在规划查询时花费的总时间（微秒）。
lock_wait_time	bigint	等待关系锁定所花费的总时间（微秒）。

示例查询

以下查询返回了正在运行的查询和已排队的查询。

```
SELECT user_id,
       query_id,
       transaction_id,
       session_id,
       status,
       trim(database_name) AS database_name,
       start_time,
       end_time,
       result_cache_hit,
       elapsed_time,
       queue_time,
       execution_time
FROM sys_query_history
```

```
WHERE status IN ('running','queued')
ORDER BY start_time;
```

示例输出。

```
 user_id | query_id | transaction_id | session_id | status  | database_name |
 start_time         |          end_time          | result_cache_hit | elapsed_time |
 queue_time | execution_time
---------+----------+----------------+------------+---------+---------------
+--------------------------+--------------------------+-----------------
+--------------+------------+----------------
     101 |   760705 |         852337 | 1073832321 | running | tpcds_1t      |
 2022-02-15 19:03:19.67849 | 2022-02-15 19:03:19.739811 | f                |
 61321 |          0 |              0
```

以下查询返回了特定查询的查询开始时间、结束时间、排队时间、已用时间、计划时间和其他元数据。

```
SELECT user_id,
       query_id,
       transaction_id,
       session_id,
       status,
       trim(database_name) AS database_name,
       start_time,
       end_time,
       result_cache_hit,
       elapsed_time,
       queue_time,
       execution_time,
       planning_time,
       trim(query_text) as query_text
FROM sys_query_history
WHERE query_id = 3093;
```

示例输出。

```
user_id | query_id | transaction_id | session_id |   status   | database_name |
   start_time         |          end_time          | result_cache_hit | elapsed_time |
 queue_time | execution_time | planning_time | query_text
--------+----------+----------------+------------+------------
+---------------+----------------------------+----------------------------
```

```
+-----------------+-------------+-----------+---------------+--------------
+-------------------------------------
    106 |     3093 |          11759 | 1073750146 | success    | dev           |
 2023-03-16 16:53:17.840214 | 2023-03-16 16:53:18.106588 | f               |
 266374 |          0 |         105725 |        136589 | select count(*) from item;
```

以下示例列出了 10 个最近的 SELECT 查询。

```
SELECT query_id,
       transaction_id,
       session_id,
       start_time,
       elapsed_time,
       queue_time,
       execution_time,
       returned_rows,
       returned_bytes
FROM sys_query_history
WHERE query_type = 'SELECT'
ORDER BY start_time DESC limit 10;
```

示例输出。

```
 query_id | transaction_id | session_id |         start_time         | elapsed_time |
 queue_time | execution_time | returned_rows | returned_bytes
----------+----------------+------------+----------------------------+--------------
+------------+----------------+---------------+----------------
   526532 |          61093 | 1073840313 | 2022-02-09 04:43:24.149603 |       520571 |
        0 |         481293 |              1 |           3794
   526520 |          60850 | 1073840313 | 2022-02-09 04:38:27.24875  |       635957 |
        0 |         596601 |              1 |           3679
   526508 |          60803 | 1073840313 | 2022-02-09 04:37:51.118835 |       563882 |
        0 |         503135 |              5 |          17216
   526505 |          60763 | 1073840313 | 2022-02-09 04:36:48.636224 |       649337 |
        0 |         589823 |              1 |            652
   526478 |          60730 | 1073840313 | 2022-02-09 04:36:11.741471 |     14611321 |
        0 |       14544058 |              0 |              0
   526467 |          60636 | 1073840313 | 2022-02-09 04:34:11.91463  |     16711367 |
        0 |       16633767 |              1 |            575
   511617 |         617946 | 1074009948 | 2022-01-20 06:21:54.44481  |      9937090 |
        0 |        9899271 |            100 |          12500
```

```
   511603 |         617941 | 1074259415 | 2022-01-20 06:21:45.71744  |      8065081 |
        0 |        7582500 |          100 |           8889
   511595 |         617935 | 1074128320 | 2022-01-20 06:21:44.030876 |      1051270 |
        0 |        1014879 |            1 |             72
   511584 |         617931 | 1074030019 | 2022-01-20 06:21:42.764088 |       609033 |
        0 |         485887 |          100 |           8438
```

以下查询显示了每日 Select 查询的计数和平均查询用时。

```
SELECT date_trunc('day',start_time) AS exec_day,
       status,
       COUNT(*) AS query_cnt,
       AVG(datediff (microsecond,start_time,end_time)) AS elapsed_avg
FROM sys_query_history
WHERE query_type = 'SELECT'
AND start_time >= '2022-01-14'
AND start_time <= '2022-01-18'
GROUP BY exec_day,
         status
ORDER BY exec_day,
         status;
```

示例输出。

```
      exec_day       | status  | query_cnt | elapsed_avg
---------------------+---------+-----------+------------
 2022-01-14 00:00:00 | success |      5253 |  56608048
 2022-01-15 00:00:00 | success |      7004 |  56995017
 2022-01-16 00:00:00 | success |      5253 |  57016363
 2022-01-17 00:00:00 | success |      5309 |  55236784
 2022-01-18 00:00:00 | success |      8092 |  54355124
```

以下查询显示了每日查询所用时间性能。

```
SELECT distinct date_trunc('day',start_time) AS exec_day,
       query_count.cnt AS query_count,
       Percentile_cont(0.5) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P50_runtime,
       Percentile_cont(0.8) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P80_runtime,
       Percentile_cont(0.9) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P90_runtime,
```

```
       Percentile_cont(0.99) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P99_runtime,
       Percentile_cont(1.0) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS max_runtime
FROM sys_query_history
LEFT JOIN (SELECT  date_trunc('day',start_time) AS day, count(*) cnt
           FROM sys_query_history
           WHERE query_type = 'SELECT'
           GROUP by 1) query_count
ON date_trunc('day',start_time) = query_count.day
WHERE query_type = 'SELECT'
ORDER BY exec_day;
```

示例输出。

```
      exec_day       | query_count | p50_runtime | p80_runtime | p90_runtime |
 p99_runtime  | max_runtime
--------------------+-------------+-------------+-------------+-------------
+--------------+--------------
 2022-01-14 00:00:00 |        5253 |  16816922.0 |  69525096.0 | 158524917.8 |
 486322477.52 | 1582078873.0
 2022-01-15 00:00:00 |        7004 |  15896130.5 |  71058707.0 | 164314568.9 |
 500331542.07 | 1696344792.0
 2022-01-16 00:00:00 |        5253 |  15750451.0 |  72037082.2 | 159513733.4 |
 480372059.24 | 1594793766.0
 2022-01-17 00:00:00 |        5309 |  15394513.0 |  68881393.2 | 160254700.0 |
 493372245.84 | 1521758640.0
 2022-01-18 00:00:00 |        8092 |  15575286.5 |  68485955.4 | 154559572.5 |
 463552685.39 | 1542783444.0
 2022-01-19 00:00:00 |        5860 |  16648747.0 |  72470482.6 | 166485138.2 |
 492038228.67 | 1693483241.0
 2022-01-20 00:00:00 |        1751 |  15422072.0 |  69686381.0 | 162315385.0 |
 497066615.00 | 1439319739.0
 2022-02-09 00:00:00 |          13 |   6382812.0 |  17616161.6 |  21197988.4 |
 23021343.84 |   23168439.0
```

以下查询显示了查询类型分布。

```
SELECT query_type,
       COUNT(*) AS query_count
FROM sys_query_history
GROUP BY query_type
```

```
ORDER BY query_count DESC;
```

示例输出。

```
 query_type | query_count
------------+-------------
 UTILITY    |      134486
 SELECT     |       38537
 DDL        |        4832
 OTHER      |         768
 LOAD       |         768
 CTAS       |         748
 COMMAND    |          92
```

SYS_QUERY_TEXT

使用 SYS_QUERY_TEXT 查看所有查询的查询文本。每行代表最多 4000 个字符的查询的查询文本（从序列号 0 开始）。当查询语句包含超过 4000 个字符时，通过增加每行的序列号来为该语句记录额外的行。此视图记录所有用户查询文本，例如 DDL、实用程序、Amazon Redshift 查询和仅限领导节点的查询。

SYS_QUERY_TEXT 对所有用户均可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	integer	提交查询的用户标识符。
query_id	bigint	查询标识符。
transaction_id	bigint	与此语句关联的事务的标识符。
session_id	整数	运行查询的会话的进程标识符。
start_time	时间戳	查询开始的时间。

列名称	数据类型	描述
sequence	整数	当单个语句包含超过 4000 个字符时，将为该语句记录额外的行。序列 0 是第一行，1 是第二行，依此类推。
文本	character (4000)	以 4000 个字符为增量的 SQL 查询文本。此字段可能包含反斜杠 (\) 和换行符 (\n) 等特殊字符。

示例查询

以下查询返回了正在运行的查询和已排队的查询。

```
SELECT user_id,
 query_id,
 transaction_id,
 session_id, start_time,
 sequence, trim(text) as text from sys_query_text
 ORDER BY sequence;
```

示例输出。

```
 user_id | query_id | transaction_id | session_id |        start_time          |
 sequence |                                                        text
--------+----------+----------------+------------+----------------------------
+----------
+-----------------------------------------------------------------------------------------------------------
   100  |     4    |       1396     | 1073750220 | 2023-04-28 16:44:55.887184 |
   0    | SELECT trim(text) as text, sequence FROM sys_query_text WHERE query_id =
 pg_last_query_id() AND user_id > 1 AND start
_time > '2023-04-28 16:44:55.922705+00:00'::timestamp order by sequence;
```

以下查询返回已在数据库的组中授予或撤消的权限。

```
SELECT
    SPLIT_PART(text, ' ', 1) as grantrevoke,
    SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), 'GROUP'))), ' ', 2) as group,
```

```
    SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), ' '))), 'ON', 1) as type,
    SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), 'ON'))), ' ', 2) || ' ' ||
 SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), 'ON'))), ' ', 3) as entity
FROM SYS_QUERY_TEXT
WHERE (text LIKE 'GRANT%' OR text LIKE 'REVOKE%') AND text LIKE '%GROUP%';

+-------------+----------+--------+----------+
| grantrevoke |  group   |  type  |  entity  |
+-------------+----------+--------+----------+
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | USAGE  | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
+-------------+----------+--------+----------+
```

SYS_RESTORE_LOG

在经典大小调整期间，使用 SYS_RESTORE_LOG 监控集群中每个表迁入 RA3 节点的进度。它捕获调整大小操作期间数据迁移的历史吞吐量。有关经典调整 RA3 节点大小的更多信息，请参阅经典调整大小。

SYS_RESTORE_LOG 仅对超级用户可见。

表列

列名称	数据类型	描述
event_time	时间戳	一个时间戳，用于指示何时记录日志条目。
database_name	char(128)	数据库的名称。
schema_name	char(128)	架构的名称。
table_name	char(128)	表的名称。
table_id	整数	表的 ID。
action	char(128)	输入时采取的操作。值可以包括：迁移已开始、检查点、

列名称	数据类型	描述
		已恢复、已完成、已取消或重置。
table_size	long	表的大小。
total_data_processed	long	到此为止已处理的表数据大小 (MB)。
delta_data_processed	long	上次 event_time 更新后处理的数据大小 (MB)。这有助于您确定自上次记录的时间间隔以来已处理了多少数据。您可以将其与 table_size 进行比较，以了解数据处理的速度。
消息	char(512)	对操作列中值的详细说明。
redistribution_type	char(32)	表的再分配类型。KEY 转换或 EVEN 再平衡任务。有关分配方式的更多信息，请参阅分配方式。

示例查询

以下查询使用 SYS_RESTORE_LOG 计算数据处理的吞吐量。

```
SELECT
    ROUND(sum(delta_data_processed) / 1024.0, 2) as data_processed_gb,
    ROUND(datediff(sec, min(event_time), max(event_time)) / 3600.0, 2) as duration_hr,
    ROUND(data_processed_gb/duration_hr, 2) as throughput_gb_per_hr
from sys_restore_log;
```

示例输出。

```
 data_processed_gb | duration_hr | throughput_gb_per_hr
-------------------+-------------+----------------------
              0.91 |        8.37 |                 0.11
```

```
(1 row)
```

以下查询显示所有再分配类型。

```
SELECT * from sys_restore_log ORDER BY event_time;
```

```
 database_name |     schema_name      |      table_name      | table_id |
 action             | total_data_processed | delta_data_processed |         event_time
        | table_size | message |   redistribution_type
---------------+----------------------+----------------------+----------
+----------------------------+----------------------+----------------------
+----------------------------+------------+---------+--------------------------
 dev           | schemaaaa877096d844d | customer_key         |   106424 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:18:00.744977 |        325 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | dp30907_t2_autokey   |   106430 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:18:02.756675 |         90 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | dp30907_t2_autokey   |   106430 |
 Redistribution completed    |                   90 |                   90 | 2024-01-05
 02:23:30.643718 |         90 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | customer_key         |   106424 |
 Redistribution completed    |                  325 |                  325 | 2024-01-05
 02:23:45.998249 |        325 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | dp30907_t1_even      |   106428 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:23:46.083849 |         30 |         | Rebalance Disteven Table
 dev           | schemaaaa877096d844d | dp30907_t5_auto_even |   106436 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:23:46.855728 |         45 |         | Rebalance Disteven Table
 dev           | schemaaaa877096d844d | dp30907_t5_auto_even |   106436 |
 Redistribution completed    |                   45 |                   45 | 2024-01-05
 02:24:16.343029 |         45 |         | Rebalance Disteven Table
 dev           | schemaaaa877096d844d | dp30907_t1_even      |   106428 |
 Redistribution completed    |                   30 |                   30 | 2024-01-05
 02:24:20.584703 |         30 |         | Rebalance Disteven Table
 dev           | schemaefd028a2a48a4c | customer_even        |   130512 |
 Redistribution started      |                    0 |                      | 2024-01-05
 04:54:55.641741 |        190 |         | Restore Disteven Table
 dev           | schemaefd028a2a48a4c | customer_even        |   130512 |
 Redistribution checkpointed |     29.4342113157737 |     29.4342113157737 | 2024-01-05
 04:55:04.770696 |        190 |         | Restore Disteven Table
```

```
(8 rows)
```

SYS_RESTORE_STATE

在经典大小调整期间，使用 SYS_RESTORE_STATE 监控每个表的迁移进度。当目标节点类型为 RA3 时，这特别适用。有关经典调整 RA3 节点大小的更多信息，请参阅经典调整大小。

SYS_RESTORE_STATE 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	integer	提交查询的用户标识符。
database_name	char(64)	表的数据库的名称。
schema_id	整数	表的架构 ID。
table_id	整数	表的 ID。
table_name	char(128)	表的名称。
redistribution_status	char(128)	表的再分配进度的状态。可能的值为 `Completed` 、`In progress` 和 `Pending`。
percentage_redistributed	float	表的再分配进度的百分比。可能的值介于 0 到 100% 之间。例如，值为 25 表示 25% 的数据再分配。
redistribution_type	char(32)	表的再分配类型。KEY 转换或 EVEN 再平衡任务。有关分配方式的更多信息，请参阅分配方式。

示例查询

以下查询返回正在运行的查询和已排队的查询的记录。

```
SELECT * FROM sys_restore_state;
```

示例输出。

```
 userid | database_name | schema_id | table_id |   table_name   | redistribution_status
 | precentage_redistributed |   redistribution_type
--------+---------------+-----------+----------+----------------
+----------------------+--------------------------+-------------------------
    1   |     test1     |   124865  |  124878  | customer_key_4 |         Pending
 |      0                   |  Rebalance Disteven Table
    1   |      dev      |   124865  |  124874  | customer_key_3 |         Pending
 |      0                   |  Rebalance Disteven Table
    1   |      dev      |   124865  |  124870  | customer_key_2 |        Completed
 |     100                  |  Rebalance Disteven Table
    1   |      dev      |   124865  |  124866  | customer_key_1 |       In progress
 |     13.52                |  Restore Distkey Table
```

以下是数据处理状态。

```
SELECT
    redistribution_status, ROUND(SUM(block_count) / 1024.0, 2) AS total_size_gb
FROM sys_restore_state sys inner join stv_tbl_perm stv
    on sys.table_id = stv.id
GROUP BY sys.redistribution_status;
```

示例输出。

```
 redistribution_status | total_size_gb
-----------------------+---------------
 Completed             |          0.07
 Pending               |          0.71
 In progress           |          0.20
(3 rows)
```

SYS_SCHEMA_QUOTA_VIOLATIONS

记录超出架构限额时的匹配项、事务 ID 和其他有用信息。此系统表是 STL_SCHEMA_QUOTA_VIOLATIONS 的转换。

r_SYS_SCHEMA_QUOTA_VIOLATIONS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
owner_id	整数	schema 拥有者的 ID。
user_id	整数	生成该条目的用户的 ID。
transaction_id	bigint	与语句关联的事务 ID。
session_id	整数	与语句关联的进程 ID。
schema_id	integer	命名空间或 schema ID。
schema_name	character (128)	命名空间或 schema 名称。
配额	integer	schema 可以使用的磁盘空间量（以 MB 为单位）。
disk_usage	integer	schema 当前使用的磁盘空间（以 MB 为单位）。
record_time	不带时区的时间戳	违规情况发生的时间。

示例查询

以下查询显示违反限额的结果：

```
SELECT user_id, TRIM(schema_name) "schema_name", quota, disk_usage, record_time FROM
sys_schema_quota_violations WHERE SCHEMA_NAME = 'sales_schema' ORDER BY timestamp DESC;
```

此查询返回指定 schema 的以下示例输出：

```
user_id| schema_name  | quota | disk_usage | record_time
-------+--------------+-------+------------+----------------------------
104    | sales_schema | 2048  | 2798       | 2020-04-20 20:09:25.494723
(1 row)
```

SYS_SERVERLESS_USAGE

使用 SYS_SERVERLESS_USAGE 查看资源的 Amazon Redshift Serverless 使用情况的详细信息。此系统视图不适用于预置 Amazon Redshift 集群。

此视图包含无服务器使用情况摘要，包括用于处理查询的计算容量数量以及以 1 分钟粒度使用的 Amazon Redshift 托管式存储量。计算容量以 Redshift 处理单元 (RPU) 为单位衡量，并按每秒计量以 RPU 秒为单位运行的工作负载。RPU 用于处理对数据仓库中加载数据的查询、从 Amazon S3 数据湖查询或使用联合查询从操作数据库访问。Amazon Redshift Serverless 可在 SYS_SERVERLESS_USAGE 中将信息保留 7 天时间。

有关计算成本计费的示例，请参阅 Amazon Redshift Serverless 的计费。

SYS_SERVERLESS_USAGE 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
start_time	timestamp	时间间隔开始的时间。
end_time	timestamp	时间间隔完成的时间。
compute_seconds	double precision	在此时间间隔内消耗的累计计算单位 (RPU) 秒数。此值考虑了为账户分配的基本 RPU 容量。
compute_capacity	double precision	在此时间间隔内分配的计算单位（Redshift 处理单元或 RPU）的平均数。

列名称	数据类型	描述
		compute_capacity 值可动态更改。
data_storage	整数	在此时间间隔内使用的平均数据存储空间（以 MB 为单位）。 随着数据库加载或删除数据，使用的数据存储可能会动态更改。
cross_region_transferred_data	整数	在此时间间隔内为跨区域数据共享而传输的累积数据（以字节为单位）。
charged_seconds	整数	在此时间间隔内收费的累计计算单位 (RPU) 秒数。这在事务结束后计算，因此在事务运行时此值可以是 0。使用 charged_seconds 计算 Amazon Redshift Serverless 工作组的成本。此值考虑了分配给 Amazon Redshift Serverless 工作组的 RPU 容量。

使用说明

- 在某些情况下，compute_seconds 为 0，但 charged_seconds 大于 0，反之亦然。这是由于在系统视图中记录数据的方式而导致的正常行为。为了更准确地表示无服务器使用情况的详细信息，我们建议聚合数据。

示例

要通过查询 charged_seconds 获取一段时间间隔内使用的 RPU 小时数的总费用，请运行以下查询：

```
select trunc(start_time) "Day",
(sum(charged_seconds)/3600::double precision) * <Price for 1 RPU> as cost_incurred
from sys_serverless_usage
group by 1
order by 1
```

请注意，在此时间间隔内可能会有空闲时间。空闲时间不会增加使用的 RPU。

SYS_SESSION_HISTORY

使用 SYS_SESSION_HISTORY 查看有关当前活动的会话和会话历史记录的信息。

SYS_SESSION_HISTORY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	char(50)	生成该条目的用户的标识符。
session_id	整数	与语句关联的会话的 ID。
database_name	char(50)	数据库的名称。
status	char	会话的状态。可能的值为 `active`、`timed out` 和 `closed`。
session_timeout	整数	超时前会话保持非活动状态或空闲状态的最长时间（秒）。0 表示未设置超时。
start_time	时间戳	建立连接的时间戳。
end_time	时间戳	停止连接的时间戳。

示例

以下是 SYS_SESSION_HISTORY 的示例输出。

```
select * from sys_session_history;
 user_id | session_id |   database_name  | status | session_timeout |
 start_time         |          end_time
---------+------------+------------------+--------+-----------------
+----------------------------+----------------------------
       1 | 1073971370 | dev              | closed |        0        | 2023-07-17
 15:50:12.030104 | 2023-07-17 15:50:12.123218
       1 | 1073979694 | dev              | closed |        0        | 2023-07-17
 15:50:24.117947 | 2023-07-17 15:50:24.131859
       1 | 1073873049 | dev              | closed |        0        | 2023-07-17
 15:49:29.067398 | 2023-07-17 15:49:29.070294
       1 | 1073873086 | database18127a4a | closed |        0        | 2023-07-17
 15:49:29.119018 | 2023-07-17 15:49:29.125925
       1 | 1073832112 | dev              | closed |        0        | 2023-07-17
 15:49:29.164688 | 2023-07-17 15:49:29.179631
       1 | 1073987697 | dev              | closed |        0        | 2023-07-17
 15:49:29.26749  | 2023-07-17 15:49:29.273034
       1 | 1073922429 | dev              | closed |        0        | 2023-07-17
 15:49:33.35315  | 2023-07-17 15:49:33.367499
       1 | 1073766783 | dev              | closed |        0        | 2023-07-17
 15:49:45.38237  | 2023-07-17 15:49:45.396902
       1 | 1073807506 | dev              | active |        0        | 2023-07-17
 15:51:48        |
```

SYS_SPATIAL_SIMPLIFY

您可以查询系统视图 SYS_SPATIAL_SIMPLIFY，以使用 COPY 命令获取有关简化空间几何对象的信息。在 shapefile 上使用 COPY 时，您可以指定 SIMPLIFY `tolerance`、SIMPLIFY AUTO 和 SIMPLIFY AUTO `max_tolerance` 摄入选项。简化的结果在 SYS_SPATIAL_SIMPLIFY 系统视图中进行总结。

当设置 SIMPLIFY AUTO `max_tolerance` 时，此视图为每个超出最大大小的几何体都包含一行。当设置 SIMPLIFY `tolerance` 时，则存储整个 COPY 操作的一行。此行引用 COPY 查询 ID 和指定的简化容差。

有关加载 shapefile 的更多信息，请参阅将 shapefile 加载到 Amazon Redshift。

SYS_SPATIAL_SIMPLIFY 对所有用户均可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
query_id	bigint	生成此行的查询（COPY 命令）的 ID。
line_number	bigint	指定 COPY SIMPLIFY AUTO 选项时，此值是 shapefile 中简化记录的记录编号。
maximum_tolerance	double precision	COPY 命令中指定的距离容差值。它是使用 SIMPLIFY AUTO 选项的最大容差值，或使用 SIMPLIFY 选项的固定容差值。
initial_size	bigint	以字节为单位的 GEOMETRY 数据值简化前的大小。
simplified	char(1)	当指定 COPY SIMPLIFY AUTO 选项时，如果几何体已成功简化，则为 t，否则为 f。如果在使用给定的最大容差进行简化后，其大小仍然大于最大几何体大小，则可能无法成功简化几何体。
final_size	bigint	当指定 COPY SIMPLIFY AUTO 选项时，它是以字节数为单位的几何体简化后的大小。
final_tolerance	double precision	为简化而选择的最终容差。

示例查询

以下查询返回 COPY 简化的记录的列表。

```
SELECT * FROM sys_spatial_simplify;


 query_id | line_number | maximum_tolerance | initial_size | simplified | final_size |
  final_tolerance
----------+-------------+-------------------+--------------+------------+------------
+----------------------
    20    |     1184704 |                -1 |      1513736 | t          |    1008808 |
  1.276386653895e-05
```

```
    20    |     1664115 |               -1 |      1233456 | t           |    1023584 |
6.11707814796635e-06
```

SYS_STREAM_SCAN_ERRORS

记录通过串流摄取加载的记录的错误。

SYS_STREAM_SCAN_ERRORS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
external_schema_name	character (128)	Kinesis 流或 Amazon MSK 主题的 Schema 的名称。区分大小写。
stream_name	character (255)	流或主题的名称。区分大小写。
mv_name	character (128)	关联的实体化视图的名称。如果无关联视图，则为空。区分大小写。
transaction_id	bigint	事务 ID。
query_id	bigint	查询 ID。
stream_timestamp_type	character (1)	流时间戳的类型。区分大小写。
stream_timestamp	不带时区的时间戳	记录抵达的时间。

列名称	数据类型	描述
record_time	不带时区的时间戳	记录错误消息的时间。
partition_id	character (128)	分区/分片 ID。区分大小写。
position	character (128)	记录的位置。这对应于 Kinesis 中的序列号或 Amazon MSK 中的偏移。区分大小写。
error_code	integer	错误代码。
error_reason	character (128)	错误原因。区分大小写。

SYS_STREAM_SCAN_STATES

记录通过串流摄取加载的记录的扫描状态。

SYS_STREAM_SCAN_STATES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
external_schema_name	character (128)	外部模式名称。区分大小写。
stream_name	character (255)	流名称。区分大小写。

列名称	数据类型	描述
mv_name	character (128)	关联的实体化视图的名称。如果无关联视图，则为空。区分大小写。
transaction_id	bigint	事务 ID。
query_id	bigint	查询 ID。
record_time	不带时区的时间戳	记录数据的时间。
partition_id	character (128)	分区或分片 ID。区分大小写。
latest_position	character (128)	批次中读取的最后一条记录的位置。这对应于 Kinesis 中的序列号或 Amazon MSK 中的偏移。区分大小写。
scanned_rows	bigint	批次中扫描的记录数。
skipped_rows	bigint	批次中跳过的记录数。
scanned_bytes	bigint	批次中扫描的字节数。
stream_record_time_min	不带时区的时间戳	批次中最早记录的 Kinesis 或 Amazon MSK 抵达时间。
stream_record_time_max	不带时区的时间戳	批次中最晚记录的 Kinesis 或 Amazon MSK 抵达时间。

以下查询显示了特定查询的串流和主题数据。

```
select
 query_id,mv_name::varchar,external_schema_name::varchar,stream_name::varchar,sum(scanned_rows)
 total_records,
sum(scanned_bytes) total_bytes from sys_stream_scan_states where query in
 (5401180,8601939) group by 1,2,3,4;

  query_id  |    mv_name     | external_schema_name |   stream_name   | total_records |
  total_bytes
------------+----------------+----------------------+-----------------+---------------
+----------------
 5401180    | kinesistest    | kinesis              | kinesisstream   |    1493255696 |
 3209006490704
 8601939    | msktest        | msk                  | mskstream       |      14677023 |
   31056580668
(2 rows)
```

SYS_TRANSACTION_HISTORY

跟踪查询时，使用 SYS_TRANSACTION_HISTORY 查看事务的详细信息。

SYS_TRANSACTION_HISTORY 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	生成该条目的用户 ID。
transaction_id	bigint	事务的 ID。
isolation_level	文本	事务的隔离级别。可能的值为 `Serializable` 和 `Snapshot Isolation` 。
status	文本	事务的状态。可能的状态为 `committed` 和 `rolledback` 。
transaction_start_time	时间戳	事务的开始时间。

列名称	数据类型	描述
commit_start_time	时间戳	提交的开始时间。
commit_end_time	时间戳	提交的结束时间。
blocks_committed	bigint	必须作为此提交的一部分写入的数据块的数量。
undo_transaction_id	bigint	撤消事务的 ID（如果有任何事务已被撤消）。否则，此值为 -1。

示例查询

```
select * from sys_transaction_history order by transaction_start_time desc;

 user_id | transaction_id | isolation_level |   status   |   transaction_start_time
   |     commit_start_time      |      commit_end_time       | blocks_committed |
 undo_transaction_id
---------+----------------+-----------------+------------+--------------------------
+---------------------------+---------------------------+------------------
+---------------------
     100 |           1310 | Serializable    | committed  | 2023-08-27 21:03:11.822205 |
 2023-08-28 21:03:11.825287 | 2023-08-28 21:03:11.854883 |               17 |
        -1
     101 |           1345 | Serializable    | committed  | 2023-08-27 21:03:12.000278 |
 2023-08-28 21:03:12.003736 | 2023-08-28 21:03:12.030061 |               17 |
        -1
     102 |           1367 | Serializable    | committed  | 2023-08-27 21:03:12.1532   |
 2023-08-28 21:03:12.156124 | 2023-08-28 21:03:12.186468 |               17 |
        -1
     100 |           1370 | Serializable    | committed  | 2023-08-27 21:03:12.199316 |
 2023-08-28 21:03:12.204854 | 2023-08-28 21:03:12.238186 |               24 |
        -1
     100 |           1408 | Serializable    | committed  | 2023-08-27 21:03:53.891107 |
 2023-08-28 21:03:53.894825 | 2023-08-28 21:03:53.927465 |               17 |
        -1
     100 |           1409 | Serializable    | rolledback | 2023-08-27 21:03:53.936431 |
 2000-01-01 00:00:00        | 2023-08-28 21:04:08.712532 |                0 |
      1409
```

```
    101 |           1415 | Serializable    | committed  | 2023-08-27 21:04:24.283188 |
2023-08-28 21:04:24.289196 | 2023-08-28 21:04:24.374318 |               25 |
       -1
    101 |           1416 | Serializable    | committed  | 2023-08-27 21:04:24.38818  |
2023-08-28 21:04:24.391688 | 2023-08-28 21:04:24.415135 |               17 |
       -1
    100 |           1417 | Serializable    | rolledback | 2023-08-27 21:04:24.424252 |
2000-01-01 00:00:00        | 2023-08-28 21:04:28.354826 |                0 |
     1417
    101 |           1418 | Serializable    | rolledback | 2023-08-27 21:04:24.425195 |
2000-01-01 00:00:00        | 2023-08-28 21:04:28.680355 |                0 |
     1418
    100 |           1420 | Serializable    | committed  | 2023-08-27 21:04:28.697607 |
2023-08-28 21:04:28.702374 | 2023-08-28 21:04:28.735541 |               23 |
       -1
    101 |           1421 | Serializable    | committed  | 2023-08-27 21:04:28.744854 |
2023-08-28 21:04:28.749344 | 2023-08-28 21:04:28.779029 |               23 |
       -1
    101 |           1423 | Serializable    | committed  | 2023-08-27 21:04:28.78942  |
2023-08-28 21:04:28.791556 | 2023-08-28 21:04:28.817485 |               16 |
       -1
    100 |           1430 | Serializable    | committed  | 2023-08-27 21:04:28.917788 |
2023-08-28 21:04:28.919993 | 2023-08-28 21:04:28.944812 |               16 |
       -1
    102 |           1494 | Serializable    | committed  | 2023-08-27 21:04:37.029058 |
2023-08-28 21:04:37.033137 | 2023-08-28 21:04:37.062001 |               16 |
       -1
```

SYS_UDF_LOG

记录在执行用户定义的函数 (UDF) 期间生成的系统定义的错误和警告消息。

SYS_UDF_LOG 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
query_id	bigint	查询标识符。

列名称	数据类型	描述
function_name	文本	用户定义函数的名称。
record_time	时间戳	创建记录的时间。
sequence	整数	单条日志消息的序列。
message	文本	日志消息文本。

示例查询

以下示例说明 UDF 如何处理系统定义的错误。第一个块显示了返回参数的逆参数的 UDF 函数的定义。在运行函数并提供 0 作为参数时，函数返回错误。最后一条语句返回在 SYS_UDF_LOG 中记录的错误消息。

```
-- Create a function to find the inverse of a number.
CREATE OR REPLACE FUNCTION f_udf_inv(a int)

RETURNS float

IMMUTABLE AS $$return 1/a

$$ LANGUAGE plpythonu;

-- Run the function with 0 to create an error.
Select f_udf_inv(0);

-- Query SYS_UDF_LOG to view the message.
Select query_id, record_time, message::varchar from sys_udf_log;


query_id    |    record_time              |                message
----------+----------------------------
+-------------------------------------------------------
2211        | 2023-08-23 15:53:11.360538 |  ZeroDivisionError: integer division or
 modulo by zero line 2, in f_udf_inv\n return 1/a\n
```

以下示例将日志记录和警告消息添加到 UDF 中，以便被零除运算生成警告消息而不是停止并显示错误消息。

```
-- Create a function to find the inverse of a number and log a warning if you input 0.
CREATE OR REPLACE FUNCTION f_udf_inv_log(a int)
  RETURNS float IMMUTABLE
 AS $$
  import logging
  logger = logging.getLogger() #get root logger
  if a==0:
    logger.warning('You attempted to divide by zero.\nReturning zero instead of error.
\n')
    return 0
  else:
     return 1/a
$$ LANGUAGE plpythonu;

-- Run the function with 0 to trigger the warning.
Select f_udf_inv_log(0);

-- Query SYS_UDF_LOG to view the message.
Select query_id, record_time, message::varchar from sys_udf_log;

 query_id |        record_time         |                                      message
----------+----------------------------
+-------------------------------------------------------------------------------
     0   | 2023-08-23 16:10:48.833503 | WARNING: You attempted to divide by zero.
\nReturning zero instead of error.\n
```

SYS_UNLOAD_DETAIL

使用 SYS_UNLOAD_DETAIL 查看 UNLOAD 操作的详细信息。这会为由 UNLOAD 语句创建的每个文件记录一行。例如，如果 UNLOAD 创建 12 个文件，则 SYS_UNLOAD_DETAIL 将包含 12 个对应的行。

SYS_UNLOAD_DETAIL 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	生成该条目的用户的标识符。

列名称	数据类型	描述
query_id	整数	UNLOAD 命令的查询标识符。
session_id	整数	与查询语句关联的进程的 ID。
transaction_id	bigint	与查询语句关联的事务 ID。
file_name	character (1280)	文件的完整 Amazon S3 对象路径。
start_time	时间戳	事务开始的时间。
end_time	时间戳	事务结束的时间。
line_count	bigint	卸载到文件的行数。
transfer_size	bigint	传输的字节数。
file_format	character (10)	已卸载文件的文件格式。

示例查询

以下查询显示了卸载查询详细信息，包括卸载命令的格式、行和文件计数。

```
select query_id, substring(file_name, 0, 50), transfer_size, file_format from
 sys_unload_detail;
```

示例输出。

```
 query_id |                    substring                     | transfer_size |
 file_format
----------+--------------------------------------------------+---------------
+-------------
     9272 | s3://my-bucket/my_unload_doc_venue0000_part_00.gz |        395886 | Text

     9272 | s3://my-bucket/my_unload_doc_venue0001_part_00.gz |        406444 | Text

     9272 | s3://my-bucket/my_unload_doc_venue0002_part_00.gz |        409431 | Text
```

```
     9272 | s3://my-bucket/my_unload_doc_venue0003_part_00.gz |        403051 | Text

     9272 | s3://my-bucket/my_unload_doc_venue0004_part_00.gz |        413592 | Text

     9272 | s3://my-bucket/my_unload_doc_venue0005_part_00.gz |        395689 | Text

(6 rows)
```

SYS_UNLOAD_HISTORY

使用 SYS_UNLOAD_HISTORY 查看 UNLOAD 命令的详细信息。每行代表一个 UNLOAD 命令，其中包含某些字段的累积统计数据。它包含正在运行和已完成的 UNLOAD 命令。

SYS_UNLOAD_HORITY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	integer	提交卸载的用户标识符。
query_id	bigint	UNLOAD 命令的查询标识符。
transaction_id	bigint	事务标识符。
session_id	integer	运行卸载的进程的进程标识符。
database_name	文本	在发起操作时用户连接到其中的数据库的名称。
status	文本	UNLOAD 命令的状态。有效值包括 `running`、`completed`、`aborted` 和 `unknown`。
start_time	timestamp	卸载开始的时间。
end_time	timestamp	卸载完成的时间。

列名称	数据类型	描述
duration	bigint	在 UNLOAD 命令中花费的时间（微秒）。
file_format	文本	输出文件的文件格式。
compression_type	文本	压缩的类型。
unloaded_location	文本	已卸载文件的 Amazon S3 位置。
unloadd_rows	bigint	行数。
unloaded_files_count	bigint	输出文件的文件计数。
unloaded_files_size	bigint	输出文件的文件大小。
error_message	文本	UNLOAD 命令的错误消息。

示例查询

以下查询显示了卸载查询详细信息，包括卸载命令的格式、行和文件计数。

```
SELECT query_id,
       file_format,
       start_time,
       duration,
       unloaded_rows,
       unloaded_files_count
FROM sys_unload_history
ORDER BY query_id,
file_format limit 100;
```

示例输出。

```
 query_id | file_format |         start_time         | duration | unloaded_rows |
 unloaded_files_count
----------+-------------+----------------------------+----------+---------------
+----------------------
```

```
  527067 | Text        | 2022-02-09 05:18:35.844452 |  5932478 |           10 |
           1
```

SYS_USERLOG

记录数据库用户的以下更改的详细信息。

- 创建用户
- 删除用户
- 更改用户（重命名）
- 更改用户（更改属性）

您可以查询此视图，以查看有关无服务器工作组和预置集群的信息。

SYS_USERLOG 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	integer	提交卸载的用户标识符。
user_name	character(50)	受更改影响的用户的用户名。
original_user_name	character(50)	重命名操作中的原始用户名。对于任何其他操作，此字段为空。
操作	character(10)	发生的操作。有效值包括 alter、create、drop 和 rename。
has_create_db_privs	整数	如果为 true（值为 1），则表示用户具有创建数据库的权限。
is_superuser	整数	如果为 true（值为 1），则用户可以更新系统目录。

列名称	数据类型	描述
has_update_catalog_privs	整数	如果为 true（值为 1），则用户可以更新系统目录。
password_expiration	时间戳	密码到期日期。
session_id	整数	进程 ID。
transaction_id	bigint	事务 ID。
record_time	时间戳	查询开始的时间（采用 UTC 时间）。

示例查询

以下示例执行四个用户操作，然后查询 SYS_USERLOG 视图。

```
CREATE USER userlog1 password 'Userlog1';
ALTER USER userlog1 createdb createuser;
ALTER USER userlog1 rename  to userlog2;
DROP user userlog2;

SELECT user_id, user_name, original_user_name, action, has_create_db_privs,
 is_superuser from SYS_USERLOG order by record_time desc;
```

```
user_id |  user_name | original_user_name |  action | has_create_db_privs |
 is_superuser
--------+------------+--------------------+---------+---------------------+------------
    108 |  userlog2  |                    | drop    |                   1 |   1
    108 |  userlog2  |     userlog1       | rename  |                   1 |   1
    108 |  userlog1  |                    | alter   |                   1 |   1
    108 |  userlog1  |                    | create  |                   0 |   0
 (4 rows)
```

SYS_VACUUM_HISTORY

使用 SYS_VACUUM_HISTORY 查看 vacuum 查询的详细信息。有关 VACUUM 命令的信息，请参阅VACUUM。

SYS_VACUUM_HISTORY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
user_id	整数	发起查询的用户的 ID。
transaction_id	long	VACUUM 语句的事务 ID。
query_id	long	VACUUM 语句的查询标识符。您可以将此表联接到 SYS_QUERY_DETAIL 视图，以查看为某一给定 VACUUM 事务运行的各个 SQL 语句。如果您将整个数据库真空化，则将在单独的事务中将每个表真空化。对于自动 VACUUM 操作，此值为 Null。
database_name	文本	数据库的名称。
schema_name	文本	架构的名称。
table_name	text	表的名称。
table_id	整数	表的 ID。
vacuum_type	字符	VACUUM 操作的类型。可能值如下所示： • **Delete** • **Sort** • **Reindex**

列名称	数据类型	描述
		• **Recluster** • **Full** 有关 vacuum 类型的更多信息，请参阅VACUUM。
is_automatic	布尔值	如果操作是自动 vacuum，则为 `true`。否则为 `false`。
status	字符	作为 vacuum 操作的一部分完成的当前活动的描述： • Initialize • 排序 • 合并 • 删除 • Select • 失败 • 完成 • Skipped • 正在生成 INTERLEAVED SORTKEY 顺序
start_time	时间戳	vacuum 操作开始的时间。
end_time	时间戳	vacuum 操作结束的时间。如果操作正在进行中，则此字段为空。
record_time	时间戳	在 SYS_VACUUM_HISTORY 中记录 vacuum 操作的时间。

列名称	数据类型	描述
duration	整数	vacuum 操作开始和结束之间的微秒数。如果 vacuum 操作正在进行中，则此字段为空。
rows_before_vacuum	bigint	表中的行加上仍存储在磁盘上的所有已删除的行（正在等待执行 vacuum 操作）的实际数量。
size_before_vacuum	整数	vacuum 操作开始前表的大小，以 MB 为单位。
reclaimable_rows	bigint	vacuum 操作在开始之前估计将回收的行数。
reclaimed_rows	bigint	vacuum 操作回收的行数。
reclaimed_blocks	bigint	vacuum 操作回收的块数。
sortedrows_before_vacuum	整数	vacuum 操作开始之前表中已排序行的数量。
sortedrows_after_vacuum	整数	vacuum 操作完成后表中增加的已排序行数量。这不包括计入 `sortedrows_before_vacuum` 的行数。

用于迁移到 SYS 监控视图的系统视图映射

当您将 Amazon Redshift 预调配集群迁移到 Amazon Redshift Serverless 时，您的监控或诊断查询可能会引用仅在预调配集群上可用的系统视图。您可以更新查询以使用 SYS 监控视图。本页提供了仅预调配视图到 SYS 视图的映射，供您在更新查询时参考。

主题

- SYS_QUERY_HISTORY
- SYS_QUERY_DETAIL

- SYS_RESTORE_LOG
- SYS_RESTORE_STATE
- SYS_TRANSACTION_HISTORY
- SYS_QUERY_TEXT
- SYS_CONNECTION_LOG
- SYS_SESSION_HISTORY
- SYS_LOAD_DETAIL
- SYS_LOAD_HISTORY
- SYS_LOAD_ERROR_DETAIL
- SYS_UNLOAD_HISTORY
- SYS_UNLOAD_DETAIL
- SYS_COPY_REPLACEMENTS
- SYS_DATASHARE_USAGE_CONSUMER
- SYS_DATASHARE_USAGE_PRODUCER
- SYS_DATASHARE_CROSS_REGION_USAGE
- SYS_DATASHARE_CHANGE_LOG
- SYS_EXTERNAL_QUERY_DETAIL
- SYS_EXTERNAL_QUERY_ERROR
- SYS_VACUUM_HISTORY
- SYS_ANALYZE_HISTORY
- SYS_ANALYZE_COMPRESSION_HISTORY
- SYS_MV_REFRESH_HISTORY
- SYS_MV_STATE
- SYS_PROCEDURE_CALL
- SYS_PROCEDURE_MESSAGES
- SYS_UDF_LOG
- SYS_USERLOG
- SYS_SCHEMA_QUOTA_VIOLATIONS
- SYS_SPATIAL_SIMPLIFY

SYS_QUERY_HISTORY

以下各表中的部分或全部列也在 SYS_QUERY_HISTORY 中定义。

- STL_DDLTEXT
- STL_ERROR
- STL_QUERY
- STL_UTILITYTEXT
- STL_WLM_QUERY
- STV_INFLIGHT
- STV_RECENTS
- STV_WLM_QUERY_STATE
- SVL_COMPILE
- SVL_MULTI_STATEMENT_VIOLATIONS
- SVL_QLOG
- SVL_QUERY_QUEUE_INFO
- SVL_STATEMENTTEXT
- SVL_TERMINATE

SYS_QUERY_DETAIL

以下各表中的部分或全部列也在 SYS_QUERY_DETAIL 中定义。

- STL_AGGR
- STL_ALERT_EVENT_LOG
- STL_BCAST
- STL_DELETE
- STL_DIST
- STL_EXPLAIN
- STL_HASH
- STL_HASHJOIN
- STL_INSERT
- STL_LIMIT

- STL_MERGE
- STL_MERGEJOIN
- STL_NESTLOOP
- STL_PARSE
- STL_PLAN_INFO
- STL_PROJECT
- STL_QUERY_METRICS
- STL_RETURN
- STL_SAVE
- STL_SCAN
- STL_SORT
- STL_STREAM_SEGS
- STL_UNIQUE
- STL_WINDOW
- STV_EXEC_STATE
- STV_QUERY_METRICS
- SVCS_QUERY_SUMMARY
- SVL_QUERY_METRICS
- SVL_QUERY_METRICS_SUMMARY
- SVL_QUERY_REPORT
- SVL_QUERY_SUMMARY
- SVV_QUERY_STATE

SYS_RESTORE_LOG

下表中的部分或全部列也在 SYS_RESTORE_LOG 中定义。

- SVL_RESTORE_ALTER_TABLE_PROGRESS

SYS_RESTORE_STATE

下表中的部分或全部列也在 SYS_RESTORE_STATE 中定义。

- STV_XRESTORE_ALTER_QUEUE_STATE

SYS_TRANSACTION_HISTORY

以下各表中的部分或全部列也在 SYS_TRANSACTION_HISTORY 中定义。

- STL_COMMIT_STATS
- STL_TR_CONFLICT
- STL_UNDONE

SYS_QUERY_TEXT

下表中的部分或全部列也在 SYS_QUERY_TEXT 中定义。

- STL_QUERYTEXT

SYS_CONNECTION_LOG

下表中的部分或全部列也在 SYS_CONNECTION_LOG 中定义。

- STL_CONNECTION_LOG

SYS_SESSION_HISTORY

以下各表中的部分或全部列也在 SYS_SESSION_HISTORY 中定义。

- STL_SESSIONS
- STL_RESTARTED_SESSIONS
- STV_SESSIONS

SYS_LOAD_DETAIL

下表中的部分或全部列也在 SYS_LOAD_DETAIL 中定义。

- STL_LOAD_COMMITS

SYS_LOAD_HISTORY

下表中的部分或全部列也在 SYS_LOAD_HISTORY 中定义。

- STL_LOAD_COMMITS

SYS_LOAD_ERROR_DETAIL

以下各表中的部分或全部列也在 SYS_LOAD_ERROR_DETAIL 中定义。

- STL_LOADERROR_DETAIL
- STL_LOAD_ERRORS

SYS_UNLOAD_HISTORY

下表中的部分或全部列也在 SYS_UNLOAD_HISTORY 中定义。

- STL_UNLOAD_LOG

SYS_UNLOAD_DETAIL

下表中的部分或全部列也在 SYS_UNLOAD_DETAIL 中定义。

- STL_UNLOAD_LOG

SYS_COPY_REPLACEMENTS

下表中的部分或全部列也在 SYS_COPY_REPLACEMENTS 中定义。

- STL_REPLACEMENTS

SYS_DATASHARE_USAGE_CONSUMER

下表中的部分或全部列也在 SYS_DATASHARE_USAGE_CONSUMER 中定义。

- SVL_DATASHARE_USAGE_CONSUMER

SYS_DATASHARE_USAGE_PRODUCER

下表中的部分或全部列也在 SYS_DATASHARE_USAGE_PRODUCER 中定义。

- SVL_DATASHARE_USAGE_PRODUCER

SYS_DATASHARE_CROSS_REGION_USAGE

下表中的部分或全部列也在 SYS_DATASHARE_CROSS_REGION_USAGE 中定义。

- SVL_DATASHARE_CROSS_REGION_USAGE

SYS_DATASHARE_CHANGE_LOG

下表中的部分或全部列也在 SYS_DATASHARE_CHANGE_LOG 中定义。

- SVL_DATASHARE_CHANGE_LOG

SYS_EXTERNAL_QUERY_DETAIL

以下各表中的部分或全部列也在 SYS_EXTERNAL_QUERY_DETAIL 中定义。

- SVL_FEDERATED_QUERY
- SVL_S3LIST
- SVL_S3QUERY
- SVL_S3QUERY_SUMMARY

SYS_EXTERNAL_QUERY_ERROR

以下各表中的部分或全部列也在 SYS_EXTERNAL_QUERY_ERROR 中定义。

- SVL_SPECTRUM_SCAN_ERROR

SYS_VACUUM_HISTORY

以下各表中的部分或全部列也在 SYS_VACUUM_HISTORY 中定义。

- STL_VACUUM
- SVL_VACUUM_PERCENTAGE
- SVV_VACUUM_PROGRESS
- SVV_VACUUM_SUMMARY

SYS_ANALYZE_HISTORY

以下各表中的部分或全部列也在 SYS_ANALYZE_HISTORY 中定义。

- STL_ANALYZE

SYS_ANALYZE_COMPRESSION_HISTORY

以下各表中的部分或全部列也在 SYS_ANALYZE_COMPRESSION_HISTORY 中定义。

- STL_ANALYZE_COMPRESSION

SYS_MV_REFRESH_HISTORY

以下各表中的部分或全部列也在 SYS_MV_REFRESH_HISTORY 中定义。

- SVL_MV_REFRESH_STATUS

SYS_MV_STATE

以下各表中的部分或全部列也在 SYS_MV_STATE 中定义。

- STL_MV_STATE

SYS_PROCEDURE_CALL

以下各表中的部分或全部列也在 SYS_PROCEDURE_CALL 中定义。

- SVL_STORED_PROC_CALL

SYS_PROCEDURE_MESSAGES

以下各表中的部分或全部列也在 SYS_PROCEDURE_MESSAGES 中定义。

- SVL_STORED_PROC_MESSAGES

SYS_UDF_LOG

以下各表中的部分或全部列也在 SYS_UDF_LOG 中定义。

- SVL_UDF_LOG

SYS_USERLOG

以下各表中的部分或全部列也在 SYS_USERLOG 中定义。

- STL_USERLOG

SYS_SCHEMA_QUOTA_VIOLATIONS

以下各表中的部分或全部列也在 SYS_SCHEMA_QUOTA_VIOLATIONS 中定义。

- STL_SCHEMA_QUOTA_VIOLATIONS

SYS_SPATIAL_SIMPLIFY

以下各表中的部分或全部列也在 SYS_SPATIAL_SIMPLIFY 中定义。

- SVL_SPATIAL_SIMPLIFY

系统监控（仅已预置）

可以在已预置集群上查询以下系统表和视图。STL 和 STV 表和视图包含一些系统表中的数据的子集。它们提供对在这些表中找到的常用查询数据的更快速、更便捷的访问。

SVCS 视图提供了有关主集群和并发扩展集群上的查询的详细信息。SVL 视图仅提供在主集群上运行的查询的信息，但 SVL_STATEMENTTEXT 除外。SVL_STATEMENTTEXT 可以包含在并发扩展集群和主集群上运行的查询的信息。

主题

用于日志记录的 STL 视图

STL 系统视图从 Amazon Redshift 日志文件中生成，以提供该系统的历史记录。

这些文件位于数据仓库集群中的每个节点上。STL 视图提取日志中的信息并将信息的格式设置为可供系统管理员使用的视图。

日志保留 – STL 系统视图保留七天的日志历史记录。所有集群大小和节点类型的日志保留均得到保证，并且不受集群工作负载变化的影响。日志保留也不会受到集群状态（例如集群暂停）的影响。仅当集群是新集群时，您的日志历史记录才会少于七天。您不必执行任何操作即可保留日志，但必须定期将日志数据复制到其他表，或将其上传到 Amazon S3，以保留超过 7 天的日志数据。

主题

- STL_HASHJOIN
- STL_INSERT
- STL_LIMIT
- STL_LOAD_COMMITS
- STL_LOAD_ERRORS
- STL_LOADERROR_DETAIL
- STL_MERGE
- STL_MERGEJOIN
- STL_MV_STATE
- STL_NESTLOOP
- STL_PARSE
- STL_PLAN_INFO
- STL_PROJECT
- STL_QUERY
- STL_QUERY_METRICS
- STL_QUERYTEXT
- STL_REPLACEMENTS
- STL_RESTARTED_SESSIONS
- STL_RETURN
- STL_S3CLIENT
- STL_S3CLIENT_ERROR
- STL_SAVE
- STL_SCAN
- STL_SCHEMA_QUOTA_VIOLATIONS
- STL_SESSIONS
- STL_SORT
- STL_SSHCLIENT_ERROR
- STL_STREAM_SEGS
- STL_TR_CONFLICT
- STL_UNDONE

- STL_UNIQUE
- STL_UNLOAD_LOG
- STL_USAGE_CONTROL
- STL_USERLOG
- STL_UTILITYTEXT
- STL_VACUUM
- STL_WINDOW
- STL_WLM_ERROR
- STL_WLM_RULE_ACTION
- STL_WLM_QUERY

STL_AGGR

分析查询的聚合执行步骤。这些步骤发生于执行聚合函数和 GROUP BY 语句期间。

STL_AGGR 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_AGGR 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。

列名称	数据类型	描述
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
bytes	bigint	该步骤中所有输出行的大小（以字节为单位）。
slots	integer	哈希桶的数量。
occupied	integer	包含记录的槽位的数量。
maxlength	integer	最大槽位的大小。
tbl	integer	表 ID。
is_diskbased	character(1)	如果为 true (t)，则查询是作为基于磁盘的操作运行的。如果为 false (f)，则查询是在内存中运行。
workmem	bigint	分配给步骤的工作内存的字节数。
type	character(6)	步骤的类型。有效值为： • HASHED。表示步骤使用了已分组但未排序的聚合。 • PLAIN。表示步骤使用了未分组的标量聚合。 • SORTED。表示步骤使用了已分组且已排序的聚合。
resizes	integer	此信息仅供内部使用。
flushable	integer	此信息仅供内部使用。

示例查询

返回有关 SLICE 1 和 TBL 239 的聚合执行步骤的信息。

```
select query, segment, bytes, slots, occupied, maxlength, is_diskbased, workmem, type
from stl_aggr where slice=1 and tbl=239
order by rows
limit 10;
```

```
 query | segment | bytes |  slots  | occupied | maxlength | is_diskbased |  workmem  |
 type
-------+---------+-------+---------+----------+-----------+--------------+-----------
+--------
   562 |       1 |     0 | 4194304 |        0 |         0 | f            | 383385600 |
 HASHED
   616 |       1 |     0 | 4194304 |        0 |         0 | f            | 383385600 |
 HASHED
   546 |       1 |     0 | 4194304 |        0 |         0 | f            | 383385600 |
 HASHED
   547 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   685 |       1 |    32 | 4194304 |        1 |         0 | f            | 383385600 |
 HASHED
   652 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   680 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   658 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   686 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   695 |       1 |    32 | 4194304 |        1 |         0 | f            | 383385600 |
 HASHED
(10 rows)
```

STL_ALERT_EVENT_LOG

当查询优化程序发现可能指示性能问题的条件时记录警报。使用 STL_ALERT_EVENT_LOG 视图标识用于改进查询性能的机会。

一个查询包含多个区段，而且每个区段包含一个或多个步骤。有关更多信息，请参阅 查询处理。

STL_ALERT_EVENT_LOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_ALERT_EVENT_LOG 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
pid	integer	与语句和切片关联的进程 ID。如果同一查询在多个切片上运行，则该查询可能有多个 PID。
xid	bigint	与语句关联的事务 ID。
event	character (1024)	警报事件的描述。
solution	character (1024)	建议的解决方案。
event_time	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。

使用说明

您可以使用 STL_ALERT_EVENT_LOG 来标识查询中的潜在问题，然后按照优化查询性能中的做法来优化数据库设计并重新编写查询。STL_ALERT_EVENT_LOG 将记录以下警报：

- 缺失统计数据

  统计数据缺失。在加载数据或进行大量更改之后运行 ANALYZE 并结合使用 STATUPDATE 和 COPY 操作。有关更多信息，请参阅 设计查询的 Amazon Redshift 最佳实践。
- 嵌套循环

  一个嵌套循环通常是一个笛卡尔积。评估您的查询以确保所有参与表均已高效地联接。
- 选择性非常强的筛选条件

  返回的行与扫描的行的比率小于 0.05。扫描的行数是 `rows_pre_user_filter` 的值，而返回的行数是 STL_SCAN 系统视图中的行数值。表示查询正在扫描数量极其大的行来确定结果集。这可能是由于排序键缺失或不正确导致的。有关更多信息，请参阅 使用排序键。
- 过多的虚影行

  扫描跳过了相对大量的标记为已删除但未抽空的行或已插入但未提交的行。有关更多信息，请参阅 对表执行 vacuum 操作。
- 大型分配

  为进行哈希联接或聚合重新分配了超过 1000000 的行。有关更多信息，请参阅 使用数据分配样式。
- 大型广播

  为进行哈希联接广播了超过了 1000000 的行。有关更多信息，请参阅 使用数据分配样式。
- 顺序执行

  DS_DIST_ALL_INNER 重新分配方式已在查询计划中指明，此方式强制实施序列执行，因为整个内部表已重新分配到单个节点。有关更多信息，请参阅 使用数据分配样式。

示例查询

以下查询显示了四种查询的警报事件。

```
SELECT query, substring(event,0,25) as event,
substring(solution,0,25) as solution,
trim(event_time) as event_time from stl_alert_event_log order by query;
```

```
 query |             event             |          solution             |     event_time

-------+-------------------------------+-------------------------------
+---------------------
  6567 | Missing query planner statist | Run the ANALYZE command       | 2014-01-03
 18:20:58
  7450 | Scanned a large number of del | Run the VACUUM command to rec| 2014-01-03
 21:19:31
  8406 | Nested Loop Join in the query | Review the join predicates to| 2014-01-04
 00:34:22
 29512 | Very selective query filter:r | Review the choice of sort key| 2014-01-06
 22:00:00

(4 rows)
```

STL_ANALYZE

记录 ANALYZE 操作的详细信息。

STL_ANALYZE 只对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_ANALYZE_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户的 ID。
xid	long	事务 ID。
数据库	char(30)	数据库名称。
table_id	integer	表 ID。
status	char(15)	分析命令的结果。可能的值为 `Full`、`Skipped` 和 `PredicateColumn` 。
rows	double	表中的总行数。

列名称	数据类型	描述
modified_rows	double	自上次执行 ANALYZE 操作以来修改的行的总数。
threshold_percent	integer	analyze_threshold_percent 参数的值。
is_auto	char(1)	如果操作预设情况下包含 Amazon Redshift 分析，则值为 true (t)。如果明确运行了 ANALYZE 命令，则值为 false (f)。
starttime	timestamp	分析操作开始运行的时间（采用 UTC 表示）。
endtime	timestamp	分析操作完成运行的时间（采用 UTC 表示）。
prevtime	timestamp	上次分析表的时间（采用 UTC 表示）。
num_predicate_cols	integer	表中现有的谓词列的数量。
num_new_predicate_cols	integer	自上次分析操作以来表中新谓词列的数量。
is_background	character(1)	如果分析由自动分析操作运行，则值为 true (t)。否则，该值将设置为 false (f)。
auto_analyze_phase	character(100)	保留供内部使用。
schema_name	char(128)	表 schema 的名称。
table_name	char(136)	表的名称。

示例查询

以下示例联接 STV_TBL_PERM 以显示表名称和执行详细信息。

```
select distinct a.xid, trim(t.name) as name, a.status, a.rows, a.modified_rows,
 a.starttime, a.endtime
from stl_analyze a
join stv_tbl_perm t  on t.id=a.table_id
where name = 'users'
order by starttime;

xid    | name  | status          | rows  | modified_rows | starttime           |
 endtime
-------+-------+-----------------+-------+---------------+---------------------
+---------------------
  1582 | users | Full            | 49990 |         49990 | 2016-09-22 22:02:23 |
 2016-09-22 22:02:28
244287 | users | Full            | 24992 |         74988 | 2016-10-04 22:50:58 |
 2016-10-04 22:51:01
244712 | users | Full            | 49984 |         24992 | 2016-10-04 22:56:07 |
 2016-10-04 22:56:07
245071 | users | Skipped         | 49984 |             0 | 2016-10-04 22:58:17 |
 2016-10-04 22:58:17
245439 | users | Skipped         | 49984 |          1982 | 2016-10-04 23:00:13 |
 2016-10-04 23:00:13
(5 rows)
```

STL_ANALYZE_COMPRESSION

记录在 COPY 或 ANALYZE COMPRESSION 命令期间压缩分析操作的详细信息。

STL_ANALYZE_COMPRESSION 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_ANALYZE_COMPRESSION_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户的 ID。
start_time	timestamp	开始压缩分析操作的时间。

列名称	数据类型	描述
xid	bigint	压缩分析操作的事务 ID。
tbl	integer	所分析的表的表 ID。
tablename	character(128)	所分析的表的名称。
col	integer	表中进行分析以确定压缩编码的列的索引。
old_encoding	character(15)	压缩分析之前的编码类型。
new_encoding	character(15)	压缩分析之后的编码类型。
mode	character(14)	可能的值包括： PRESET 指定 new_encoding 由 Amazon Redshift COPY 命令根据列数据类型决定。不会对数据进行采样。 开 指定 new_encoding 由 Amazon Redshift COPY 命令根据采样数据的分析决定。 ANALYZE ONLY 指定 new_encoding 由 Amazon Redshift ANALYZE COMPRESSION 命令根据采样数据的分析决定。但是，不会改变所分析的列的编码类型。
best_compression_encoding	character(15)	提供最佳压缩比的编码类型。
recommended_bytes	character(15)	采用新编码所使用的字节。

列名称	数据类型	描述
best_compression_bytes	character(15)	采用最佳压缩编码所使用的字节。
ndv	bigint	采样行中独特值的数量。

示例查询

以下示例检查在同一个会话中运行的最后一个 COPY 命令对 `lineitem` 表执行压缩分析的详细信息。

```
select xid, tbl, btrim(tablename) as tablename, col, old_encoding, new_encoding,
 best_compression_encoding, mode
from stl_analyze_compression
where xid = (select xid from stl_query where query = pg_last_copy_id()) order by col;

 xid  |  tbl   | tablename | col |  old_encoding   |  new_encoding   |
 best_compression_encoding |      mode
------+--------+-----------+-----+-----------------+-----------------
+---------------------------+----------------
 5308 | 158961 | $lineitem |   0 | mostly32        | az64            | delta
          | ON
 5308 | 158961 | $lineitem |   1 | mostly32        | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   2 | lzo             | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   3 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   4 | bytedict        | az64            | bytedict
          | ON
 5308 | 158961 | $lineitem |   5 | mostly32        | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   6 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   7 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   8 | lzo             | lzo             | lzo
          | ON
 5308 | 158961 | $lineitem |   9 | runlength       | runlength       | runlength
          | ON
```

```
 5308 | 158961 | $lineitem |  10 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |  11 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |  12 | delta           | az64            | az64
          | ON
 5308 | 158961 | $lineitem |  13 | bytedict        | bytedict        | bytedict
          | ON
 5308 | 158961 | $lineitem |  14 | bytedict        | bytedict        | bytedict
          | ON
 5308 | 158961 | $lineitem |  15 | text255         | text255         | text255
          | ON
(16 rows)
```

STL_BCAST

记录有关在执行广播数据的查询步骤期间的网络活动的信息。将按照在对给定切片执行给定步骤期间通过网络发送的行、字节和数据包的数量来捕获网络流量。步骤的持续时间是记录的开始时间与结束时间之差。

要在查询中标识广播步骤，请在 SVL_QUERY_SUMMARY 视图中查找 bcast 标签或者运行 EXPLAIN 命令，然后查找包括 bcast 的步骤属性。

STL_BCAST 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_BCAST 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。

列名称	数据类型	描述
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
bytes	bigint	该步骤中所有输出行的大小（以字节为单位）。
packets	integer	通过网络发送的数据包的总数。

示例查询

以下示例返回查询的广播信息，其中有一个或多个数据包而且查询的开始时间与结束时间之差为一秒或一秒以上。

```
select query, slice, step, rows, bytes, packets, datediff(seconds, starttime, endtime)
from stl_bcast
where packets>0 and datediff(seconds, starttime, endtime)>0;
```

```
 query | slice | step | rows | bytes | packets | date_diff
-------+-------+------+------+-------+---------+-----------
   453 |     2 |    5 |    1 |   264 |       1 |         1
   798 |     2 |    5 |    1 |   264 |       1 |         1
  1408 |     2 |    5 |    1 |   264 |       1 |         1
  2993 |     0 |    5 |    1 |   264 |       1 |         1
  5045 |     3 |    5 |    1 |   264 |       1 |         1
```

```
  8073 |     3 |    5 |    1 |   264 |       1 |         1
  8163 |     3 |    5 |    1 |   264 |       1 |         1
  9212 |     1 |    5 |    1 |   264 |       1 |         1
  9873 |     1 |    5 |    1 |   264 |       1 |         1
(9 rows)
```

STL_COMMIT_STATS

提供与提交性能相关的指标，包括提交的各个阶段的时间和提交的数据块的数量。查询 STL_COMMIT_STATS 以确定提交时使用的是事务的哪个部分以及出现了多少队列。

STL_COMMIT_STATS 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_TRANSACTION_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
xid	bigint	所提交的事务 ID。
node	integer	节点编号。-1 是领导节点。
startqueue	timestamp	提交的队列的开始时间。
startwork	timestamp	提交的开始时间。
endflush	timestamp	脏数据块刷新阶段的结束时间。
endstage	timestamp	元数据暂存阶段的结束时间。
endlocal	timestamp	本地提交阶段的结束时间。
startglobal	timestamp	全局阶段的开始时间。
endtime	timestamp	提交的结束时间。
queuelen	bigint	提交队列中位于此事务之前的事务数。
permblocks	bigint	此次提交时现有永久性数据块的数量。

列名称	数据类型	描述
newblocks	bigint	此次提交时新的永久性数据块的数量。
dirtyblocks	bigint	必须作为此提交的一部分写入的数据块的数量。
headers	bigint	必须作为此提交的一部分写入的数据块标头的数量。
numxids	integer	活动 DML 事务的数量。
oldestxid	bigint	最旧的活动 DML 事务的 XID。
extwritelatency	bigint	此信息仅供内部使用。
metadatawritten	int	此信息仅供内部使用。
tombstonedblocks	bigint	此信息仅供内部使用。
tossedblocks	bigint	此信息仅供内部使用。
batched_by	bigint	此信息仅供内部使用。

示例查询

```
select node, datediff(ms,startqueue,startwork) as queue_time,
datediff(ms, startwork, endtime) as commit_time, queuelen
from stl_commit_stats
where xid = 2574
order by node;

node | queue_time   | commit_time | queuelen
-----+--------------+-------------+---------
  -1 |            0 |         617 |        0
   0 | 444950725641 |         616 |        0
   1 | 444950725636 |         616 |        0
```

STL_CONNECTION_LOG

记录身份验证尝试以及连接与断开连接。

STL_CONNECTION_LOG 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_CONNECTION_LOG 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
event	character(50)	连接或身份验证事件。
recordtime	timestamp	事件发生的时间。
remotehost	character(45)	远程主机的名称或 IP 地址。
remoteport	character(32)	远程主机的端口号。
pid	integer	与语句关联的进程 ID。
dbname	character(50)	数据库名称。
username	character(50)	用户名。
authmethod	character(32)	身份验证方法。
duration	integer	连接的持续时间（单位为微秒）。
sslversion	character(50)	安全套接字层 (SSL) 版本。
sslcipher	character(128)	SSL 密码。
mtu	integer	最大传输单元 (MTU)。
sslcompression	character(64)	SSL 压缩类型。

列名称	数据类型	描述
sslexpansion	character(64)	SSL 扩展类型。
iamauthguid	character(36)	CloudTrail 请求的 IAM 身份验证 ID。
application_name	character(250)	会话应用程序的初始名称或更新名称。
os_version	character(64)	连接到 Amazon Redshift 集群的客户端计算机上的操作系统版本。
driver_version	character(64)	ODBC 或 JDBC 驱动程序的版本，该版本可以从您的第三方 SQL 客户端工具连接到您的 Amazon Redshift 集群。
plugin_name	character(32)	用于连接到您的 Amazon Redshift 集群的插件名称。
protocol_version	integer	Amazon Redshift 驱动程序在与服务器建立连接时使用的内部协议版本。协议版本是在驱动程序和服务器之间协商的。版本介绍了可用的功能。有效值包括： • 0 (BASE_SERVER_PROTOCOL_VERSION) • 1 (EXTENDED_RESULT_METADATA_SERVER_PROTOCOL_VERSION) – 为了保存每个查询的往返行程，服务器会发送额外的结果集元数据信息。 • 2 (BINARY_PROTOCOL_VERSION) – 根据结果集的数据类型，服务器以二进制格式发送数据。 • 3 (EXTENDED2_RESULT_METADATA_SERVER_PROTOCOL_VERSION) – 服务器发送的列信息区分大小写（排序）。
sessionid	character(36)	当前会话的全局唯一标识符。节点故障重新启动时，会话 ID 仍然存在。
compression	character(16)	连接正在使用的压缩算法。

示例查询

要查看打开连接的详细信息，请运行以下查询。

```
select recordtime, username, dbname, remotehost, remoteport
from stl_connection_log
where event = 'initiating session'
and pid not in
(select pid from stl_connection_log
where event = 'disconnecting session')
order by 1 desc;

recordtime          | username    | dbname     | remotehost    | remoteport

--------------------+-------------+------------+---------------+------------
2014-11-06 20:30:06 | rdsdb       | dev        | [local]       |

2014-11-06 20:29:37 | test001     | test       | 10.49.42.138  | 11111

2014-11-05 20:30:29 | rdsdb       | dev        | 10.49.42.138  | 33333

2014-11-05 20:28:35 | rdsdb       | dev        | [local]       |
(4 rows)
```

以下示例反映了一次失败的身份验证尝试和一次成功的连接与断开连接。

```
select event, recordtime, remotehost, username
from stl_connection_log order by recordtime;

            event      |         recordtime        |  remotehost   | username

-----------------------+---------------------------+---------------+---------
authentication failure | 2012-10-25 14:41:56.96391 | 10.49.42.138 | john

authenticated          | 2012-10-25 14:42:10.87613 | 10.49.42.138 | john

initiating session     | 2012-10-25 14:42:10.87638 | 10.49.42.138 | john

disconnecting session  | 2012-10-25 14:42:19.95992 | 10.49.42.138 | john

(4 rows)
```

以下示例显示了 ODBC 驱动程序的版本、客户端计算机上的操作系统以及用于连接到 Amazon Redshift 集群的插件。在此示例中，使用的插件用于使用登录名和密码进行标准 ODBC 驱动程序身份验证。

```
select driver_version, os_version, plugin_name from stl_connection_log;

driver_version                          |  os_version                       |
 plugin_name
----------------------------------------+-----------------------------------
+--------------------
Amazon Redshift ODBC Driver 1.4.15.0001 | Darwin 18.7.0 x86_64              | none
Amazon Redshift ODBC Driver 1.4.15.0001 | Linux 4.15.0-101-generic x86_64   | none
```

以下示例显示了客户端计算机上的操作系统版本、驱动程序版本和协议版本。

```
select os_version, driver_version, protocol_version from stl_connection_log;

os_version                      |  driver_version              | protocol_version
--------------------------------+------------------------------+--------------------
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
```

STL_DDLTEXT

捕获在系统上运行的以下 DDL 语句。

这些 DDL 语句包括以下查询和对象：

- CREATE SCHEMA、TABLE、VIEW
- DROP SCHEMA、TABLE、VIEW
- ALTER SCHEMA、TABLE

另请参阅 STL_QUERYTEXT、STL_UTILITYTEXT 和 SVL_STATEMENTTEXT。这些视图提供了在系统上运行的 SQL 命令的时间表，此历史记录非常适合用于排查问题以及创建所有系统活动的审计跟踪记录。

使用 STARTTIME 和 ENDTIME 列了解在某个给定时间段内记录了哪些语句。SQL 文本的长数据块已分为 200 个字符长的行；SEQUENCE 列标识了属于一个语句的文本片段。

STL_DDLTEXT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
xid	bigint	与语句关联的事务 ID。
pid	integer	与语句关联的进程 ID。
label	character(320)	用于运行查询的文件的名称或使用 SET QUERY_GROUP 命令定义的标签。如果查询并非基于文件或未设置 QUERY_GROUP 参数，则此字段为空。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
sequence	integer	当一个语句包含 200 多个字符时，将为该语句记录额外的行。序列 0 是第一行，1 是第二行，依此类推。
text	character(200)	SQL 文本，以 200 个字符递增。此字段可能包含反斜杠 (\\) 和换行符 (\n) 等特殊字符。

示例查询

以下查询将返回包含以前运行的 DDL 语句的记录。

```
select xid, starttime, sequence, substring(text,1,40) as text
```

```
from stl_ddltext order by xid desc, sequence;
```

以下是显示了四个 CREATE TABLE 语句的示例输出。DDL 语句显示在 text 列，为方便阅读，该列已被截断。

```
 xid  |         starttime          | sequence |                  text
------+----------------------------+----------
+----------------------------------------
 1806 | 2013-10-23 00:11:14.709851 |        0 | CREATE TABLE supplier ( s_suppkey int4
 N
 1806 | 2013-10-23 00:11:14.709851 |        1 |  s_comment varchar(101) NOT NULL )
 1805 | 2013-10-23 00:11:14.496153 |        0 | CREATE TABLE region ( r_regionkey int4
 N
 1804 | 2013-10-23 00:11:14.285986 |        0 | CREATE TABLE partsupp ( ps_partkey int8
 1803 | 2013-10-23 00:11:14.056901 |        0 | CREATE TABLE part ( p_partkey int8 NOT
 N
 1803 | 2013-10-23 00:11:14.056901 |        1 | ner char(10) NOT NULL , p_retailprice
 nu
(6 rows)
```

重新构造存储的 SQL

以下 SQL 列出了存储在 STL_DDLTEXT 的 text 列中的行。这些行按 xid 和 sequence 排序。如果原始 SQL 的长度超过 200 个字符并有多个行，则 STL_DDLTEXT 可以按 sequence 包含多个行。

```
SELECT xid, sequence, LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text)
 END, '') WITHIN GROUP (ORDER BY sequence) as query_statement
FROM stl_ddltext GROUP BY xid, sequence ORDER BY xid, sequence;
```

```
xid     |  sequence | query_statement
--------+-----------+-----------------
7886671    0          create external schema schema_spectrum_uddh\nfrom data catalog
\ndatabase 'spectrum_db_uddh'\niam_role ''\ncreate external database if not exists;
7886752    0          CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league\n(\n
  league_rank smallint,\n  prev_rank   smallint,\n  club_name   varchar(15),\n
 league_name varchar(20),\n  league_off  decimal(6,2),\n  le
7886752    1          ague_def  decimal(6,2),\n  league_spi  decimal(6,2),\n
 league_nspi smallint\n)\nROW FORMAT DELIMITED \n    FIELDS TERMINATED BY ',' \n
 LINES TERMINATED BY '\\n\\l'\nstored as textfile\nLOCATION 's
7886752    2          3://mybucket-spectrum-uddh/'\ntable properties
 ('skip.header.line.count'='1');
```

```
...
```

要重新构造存储在 STL_DDLTEXT 的 text 列中的 SQL，请运行以下 SQL 语句。它聚合了 text 列中的一个或多个段的 DDL 语句。在运行重新构造的 SQL 之前，将任何 (\n) 特殊字符替换为 SQL 客户端中的新行。以下 SELECT 语句的结果按顺序将三个行放在一起，以便在 query_statement 字段中重新构建 SQL。

```
SELECT LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END) WITHIN
 GROUP (ORDER BY sequence) as query_statement
FROM stl_ddltext GROUP BY xid, endtime order by xid, endtime;
```

```
query_statement
--------------
create external schema schema_spectrum_uddh\nfrom data catalog\ndatabase
 'spectrum_db_uddh'\niam_role ''\ncreate external database if not exists;
CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league\n(\n  league_rank smallint,
\n  prev_rank   smallint,\n  club_name   varchar(15),\n  league_name varchar(20),\n
 league_off  decimal(6,2),\n  league_def  decimal(6,2),\n  league_spi  decimal(6,2),
\n  league_nspi smallint\n)\nROW FORMAT DELIMITED \n    FIELDS TERMINATED BY ',' \n
  LINES TERMINATED BY '\\n\\l'\nstored as textfile\nLOCATION 's3://mybucket-spectrum-
uddh/'\ntable properties ('skip.header.line.count'='1');
```

STL_DELETE

分析查询的删除执行步骤。

STL_DELETE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_DELETE 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
tbl	integer	表 ID。

示例查询

为了在 STL_DELETE 中创建行，以下示例将一行插入到 EVENT 表中，然后删除该行。

首先，将一行插入到 EVENT 表中并确认已插入该行。

```
insert into event(eventid,venueid,catid,dateid,eventname)
values ((select max(eventid)+1 from event),95,9,1857,'Lollapalooza');
```

```
select * from event
where eventname='Lollapalooza'
order by eventid;
```

```
 eventid | venueid | catid | dateid |  eventname   |      starttime
---------+---------+-------+--------+--------------+---------------------
    4274 |     102 |     9 |   1965 | Lollapalooza | 2008-05-01 19:00:00
    4684 |     114 |     9 |   2105 | Lollapalooza | 2008-10-06 14:00:00
    5673 |     128 |     9 |   1973 | Lollapalooza | 2008-05-01 15:00:00
    5740 |      51 |     9 |   1933 | Lollapalooza | 2008-04-17 15:00:00
    5856 |     119 |     9 |   1831 | Lollapalooza | 2008-01-05 14:00:00
    6040 |     126 |     9 |   2145 | Lollapalooza | 2008-11-15 15:00:00
    7972 |      92 |     9 |   2026 | Lollapalooza | 2008-07-19 19:30:00
    8046 |      65 |     9 |   1840 | Lollapalooza | 2008-01-14 15:00:00
    8518 |      48 |     9 |   1904 | Lollapalooza | 2008-03-19 15:00:00
    8799 |      95 |     9 |   1857 | Lollapalooza |
(10 rows)
```

现在，删除您添加到 EVENT 表中的行并确认已删除该行。

```
delete from event
where eventname='Lollapalooza' and eventid=(select max(eventid) from event);
```

```
select * from event
where eventname='Lollapalooza'
order by eventid;
```

```
 eventid | venueid | catid | dateid |  eventname   |      starttime
---------+---------+-------+--------+--------------+---------------------
    4274 |     102 |     9 |   1965 | Lollapalooza | 2008-05-01 19:00:00
    4684 |     114 |     9 |   2105 | Lollapalooza | 2008-10-06 14:00:00
    5673 |     128 |     9 |   1973 | Lollapalooza | 2008-05-01 15:00:00
    5740 |      51 |     9 |   1933 | Lollapalooza | 2008-04-17 15:00:00
    5856 |     119 |     9 |   1831 | Lollapalooza | 2008-01-05 14:00:00
    6040 |     126 |     9 |   2145 | Lollapalooza | 2008-11-15 15:00:00
    7972 |      92 |     9 |   2026 | Lollapalooza | 2008-07-19 19:30:00
    8046 |      65 |     9 |   1840 | Lollapalooza | 2008-01-14 15:00:00
    8518 |      48 |     9 |   1904 | Lollapalooza | 2008-03-19 15:00:00
(9 rows)
```

然后，查询 stl_delete 以查看删除的执行步骤。在本示例中，查询返回 300 多行，因此已缩短下面的输出以便于显示。

```
select query, slice, segment, step, tasknum, rows, tbl from stl_delete order by query;
```

```
 query | slice | segment | step | tasknum | rows |  tbl
-------+-------+---------+------+---------+------+--------
     7 |     0 |       0 |    1 |       0 |    0 | 100000
     7 |     1 |       0 |    1 |       0 |    0 | 100000
     8 |     0 |       0 |    1 |       2 |    0 | 100001
     8 |     1 |       0 |    1 |       2 |    0 | 100001
     9 |     0 |       0 |    1 |       4 |    0 | 100002
     9 |     1 |       0 |    1 |       4 |    0 | 100002
    10 |     0 |       0 |    1 |       6 |    0 | 100003
    10 |     1 |       0 |    1 |       6 |    0 | 100003
    11 |     0 |       0 |    1 |       8 |    0 | 100253
    11 |     1 |       0 |    1 |       8 |    0 | 100253
    12 |     0 |       0 |    1 |       0 |    0 | 100255
    12 |     1 |       0 |    1 |       0 |    0 | 100255
    13 |     0 |       0 |    1 |       2 |    0 | 100257
    13 |     1 |       0 |    1 |       2 |    0 | 100257
    14 |     0 |       0 |    1 |       4 |    0 | 100259
    14 |     1 |       0 |    1 |       4 |    0 | 100259
    ...
```

STL_DISK_FULL_DIAG

有关磁盘已满时记录的错误的日志信息。

STL_DISK_FULL_DIAG 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述		
currenttime	bigint	自 2000 年 1 月 1 日以来以微秒为单位生成错误的日期和时间。		
node_num	bigint	节点的标识符。		
query_id	bigint	导致错误的查询的标识符。		
temp_blocks	bigint	查询创建的临时块数。		

示例查询

以下示例返回有磁盘已满错误时存储的数据的详细信息。

```
select * from stl_disk_full_diag
```

以下示例将 currenttime 转换为时间戳。

```
select '2000-01-01'::timestamp + (currenttime/1000000.0)* interval '1 second' as
 currenttime,node_num,query_id,temp_blocks from pg_catalog.stl_disk_full_diag;
```

```
        currenttime         | node_num | query_id | temp_blocks
----------------------------+----------+----------+-------------
 2019-05-18 19:19:18.609338 |        0 |   569399 |       70982
 2019-05-18 19:37:44.755548 |        0 |   569580 |       70982
 2019-05-20 13:37:20.566916 |        0 |   597424 |       70869
```

STL_DIST

记录有关在执行分配数据的查询步骤期间的网络活动的信息。将按照在对给定切片执行给定步骤期间通过网络发送的行、字节和数据包的数量来捕获网络流量。步骤的持续时间是记录的开始时间与结束时间之差。

要标识查询中的分配步骤，请在 QUERY_SUMMARY 视图中查找 dist 标签或者运行 EXPLAIN 命令，然后查找包含 dist 的步骤属性。

STL_DIST 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_DIST 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
bytes	bigint	该步骤中所有输出行的大小（以字节为单位）。
packets	integer	通过网络发送的数据包的总数。

示例查询

以下示例返回具有一个或多个数据包且持续时间大于零的查询的分配信息。

```
select query, slice, step, rows, bytes, packets,
datediff(seconds, starttime, endtime) as duration
from stl_dist
where packets>0 and datediff(seconds, starttime, endtime)>0
order by query
limit 10;
```

```
 query  | slice | step |  rows  |  bytes   | packets | duration
--------+-------+------+--------+----------+---------+----------
    567 |     1 |    4 |  49990 | 6249564 |     707 |         1
    630 |     0 |    5 |   8798 |  408404 |      46 |         2
    645 |     1 |    4 |   8798 |  408404 |      46 |         1
    651 |     1 |    5 | 192497 | 9226320 |    1039 |         6
    669 |     1 |    4 | 192497 | 9226320 |    1039 |         4
    675 |     1 |    5 |   3766 |  194656 |      22 |         1
    696 |     0 |    4 |   3766 |  194656 |      22 |         1
    705 |     0 |    4 |    930 |   44400 |       5 |         1
 111525 |     0 |    3 |     68 |   17408 |       2 |         1
(9 rows)
```

STL_ERROR

记录由 Amazon Redshift 数据库引擎生成的内部处理错误。STL_ERROR 不记录 SQL 错误或消息。STL_ERROR 中的信息对排除某些错误非常有用。AWS Support 工程师可能要求您提供此信息来作为故障排除过程的一部分。

STL_ERROR 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

有关在使用 Copy 命令加载数据时可生成的错误代码的列表，请参阅加载错误参考。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
process	character(12)	引发异常的进程。
recordtime	timestamp	错误发生的时间。
pid	integer	进程 ID。STL_QUERY 表包含进程 ID 和已完成查询的唯一查询 ID。

列名称	数据类型	描述
errcode	integer	对应于错误类型的错误代码。
file	character(90)	已发生错误的源文件的名称。
linenum	integer	已发生错误的源文件中的行号。
context	character(100)	错误的原因。
error	character(512)	错误消息。

示例查询

以下示例检索了 STL_ERROR 中的错误信息。

```
select process, errcode, linenum as line,
trim(error) as err
from stl_error;

   process    | errcode | line |                                 err
--------------+---------+------
+-------------------------------------------------------------------
 padbmaster   |    8001 |  194 | Path prefix: s3://redshift-downloads/testnulls/
venue.txt*
 padbmaster   |    8001 |  529 | Listing bucket=redshift-downloads prefix=tests/
category-csv-quotes
 padbmaster   |       2 |  190 | database "template0" is not currently accepting
 connections
 padbmaster   |      32 | 1956 | pq_flush: could not send data to client: Broken pipe
(4 rows)
```

STL_EXPLAIN

显示已提交供执行的查询的 EXPLAIN 计划。

STL_EXPLAIN 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_EXPLAIN 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
nodeid	integer	计划节点标识符，在执行查询时一个节点将映射到一个或多个步骤。
parentid	integer	父节点的计划节点标识符。父节点具有一些数量的子节点。例如，合并联接是针对联接表的扫描的父级。
plannode	character(400)	EXPLAIN 输出中的节点文本。表示在计算节点上执行的计划节点在 EXPLAIN 输出中使用 **XN** 作为前缀。
info	character(400)	计划节点的限定词和筛选条件信息。例如，联接条件和 WHERE 子句限制包括在此列中。

示例查询

考虑聚合联接查询的以下 EXPLAIN 输出：

```
explain select avg(datediff(day, listtime, saletime)) as avgwait
from sales, listing where sales.listid = listing.listid;
                                QUERY PLAN

------------------------------------------------------------------------------
 XN Aggregate  (cost=6350.30..6350.31 rows=1 width=16)
  ->  XN Hash Join DS_DIST_NONE  (cost=47.08..6340.89 rows=3766 width=16)
        Hash Cond: ("outer".listid = "inner".listid)
```

```
        -> XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=12)
        -> XN Hash  (cost=37.66..37.66 rows=3766 width=12)
              -> XN Seq Scan on sales  (cost=0.00..37.66 rows=3766 width=12)
(6 rows)
```

如果您运行此查询且其查询 ID 为 10，则您可以使用 STL_EXPLAIN 表来查看 EXPLAIN 命令返回的同类信息：

```
select query,nodeid,parentid,substring(plannode from 1 for 30),
substring(info from 1 for 20) from stl_explain
where query=10 order by 1,2;

query| nodeid |parentid|           substring            |    substring
-----+--------+--------+--------------------------------+-------------------
10   |      1 |      0 |XN Aggregate  (cost=6717.61..6  |
10   |      2 |      1 |  -> XN Merge Join DS_DIST_NO   | Merge Cond:("outer"
10   |      3 |      2 |       -> XN Seq Scan on lis    |
10   |      4 |      2 |       -> XN Seq Scan on sal    |
(4 rows)
```

请考虑以下查询：

```
select event.eventid, sum(pricepaid)
from event, sales
where event.eventid=sales.eventid
group by event.eventid order by 2 desc;

eventid |   sum
--------+----------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
...
```

如果此查询的 ID 为 15，则以下系统视图查询返回已完成的计划节点。在这种情况下，将反转节点的顺序以显示执行的实际顺序：

```
select query,nodeid,parentid,substring(plannode from 1 for 56)
from stl_explain where query=15 order by 1, 2 desc;
```

```
query|nodeid|parentid|                              substring
-----+------+--------+-------------------------------------------------------------
15   |    8 |      7 |                                    -> XN Seq Scan on eve
15   |    7 |      5 |                               -> XN Hash(cost=87.98..87.9
15   |    6 |      5 |                               -> XN Seq Scan on sales(cos
15   |    5 |      4 |                         -> XN Hash Join DS_DIST_OUTER(cos
15   |    4 |      3 |                  -> XN HashAggregate(cost=862286577.07..
15   |    3 |      2 |           -> XN Sort(cost=1000862287175.47..10008622871
15   |    2 |      1 |  -> XN Network(cost=1000862287175.47..1000862287197.
15   |    1 |      0 |XN Merge(cost=1000862287175.47..1000862287197.46 rows=87
(8 rows)
```

以下查询检索包含一个窗口函数的任何查询计划的查询 ID：

```
select query, trim(plannode) from stl_explain
where plannode like '%Window%';

query|                                       btrim
-----+-------------------------------------------------------------------------
26   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
27   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
(2 rows)
```

STL_FILE_SCAN

返回通过使用 COPY 命令加载数据时 Amazon Redshift 读取的文件。

查询此视图可帮助排查数据加载错误。STL_FILE_SCAN 可对于精确查找并行数据加载中的问题特别有用，因为并行数据加载通常只需一个 COPY 命令即可加载许多文件。

STL_FILE_SCAN 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_FILE_SCAN 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_LOAD_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
名称	character(90)	已加载的文件的完整路径和名称。
lines	bigint	从文件中读取的行数。
bytes	bigint	从文件中读取的字节数。
loadtime	bigint	加载文件消耗的时间量（单位为微秒）。
curtime	时间戳	表示 Amazon Redshift 开始处理文件的时间的时间戳。
is_partial	integer	值，如果为真 (1) 表示在 COPY 操作期间输入文件被拆分为范围。如果此值为假 (0)，则不会拆分输入文件。
start_offset	bigint	值，如果在 COPY 操作期间拆分输入文件，则表示拆分的偏移值（以字节为单位）。如果文件未拆分，则此值为 0。

示例查询

以下查询检索花费了超过 1,000,000 微秒来让 Amazon Redshift 读取任何文件的名称和加载时间：

```
select trim(name)as name, loadtime from stl_file_scan
where loadtime > 1000000;
```

此查询返回以下示例输出：

```
           name             | loadtime
----------------------------+----------
 listings_pipe.txt          |  9458354
 allusers_pipe.txt          |  2963761
 allevents_pipe.txt         |  1409135
 tickit/listings_pipe.txt   |  7071087
```

```
 tickit/allevents_pipe.txt |  1237364
 tickit/allusers_pipe.txt  |  2535138
 listings_pipe.txt         |  6706370
 allusers_pipe.txt         |  3579461
 allevents_pipe.txt        |  1313195
 tickit/allusers_pipe.txt  |  3236060
 tickit/listings_pipe.txt  |  4980108
(11 rows)
```

STL_HASH

分析查询的哈希执行步骤。

STL_HASH 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_HASH 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。

列名称	数据类型	描述
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
bytes	bigint	该步骤中所有输出行的大小（以字节为单位）。
slots	integer	哈希桶的总数。
occupied	integer	包含记录的槽位的总数。
maxlength	integer	最大槽位的大小。
tbl	integer	表 ID。
is_diskbased	character(1)	如果为 true (t)，则查询是作为基于磁盘的操作执行的。如果为 false (f)，则查询是在内存中执行。
workmem	bigint	分配给步骤的工作内存的字节总数。
num_parts	integer	哈希表在一个哈希步骤期间被分为的分区的总数。
est_rows	bigint	要进行哈希处理的行的估计数量。
num_blocks_permitted	integer	此信息仅供内部使用。
resizes	integer	此信息仅供内部使用。
checksum	bigint	此信息仅供内部使用。
runtime_filter_size	integer	运行时筛选器的大小（以字节为单位）。

列名称	数据类型	描述
max_runtime_filter_size	integer	运行时筛选器的最大大小（以字节为单位）。

示例查询

以下示例返回有关在查询 720 的哈希中使用的分区数量的信息，并指示在磁盘上未运行任何步骤。

```
select slice, rows, bytes, occupied, workmem, num_parts, est_rows,
 num_blocks_permitted, is_diskbased
from stl_hash
where query=720 and segment=5
order by slice;
```

```
 slice | rows | bytes  | occupied | workmem  | num_parts | est_rows |
 num_blocks_permitted | is_diskbased
-------+------+--------+----------+----------+-----------+----------
+----------------------+--------------
     0 |  145 | 585800 |        1 | 88866816 |        16 |        1 |
 52                   f
     1 |    0 |      0 |        0 |        0 |        16 |        1 |
 52                   f
(2 rows)
```

STL_HASHJOIN

分析查询的哈希联接执行步骤。

STL_HASHJOIN 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_HASHJOIN 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
tbl	integer	表 ID。
num_parts	integer	哈希表在一个哈希步骤期间被分为的分区的总数。
join_type	integer	步骤的联接类型： • 0. 查询使用了内联接。 • 1. 查询使用了左外联接。 • 2. 查询使用了全外联接。 • 3. 查询使用了右外联接。 • 4. 查询使用了 UNION 运算符。 • 5. 查询使用了 IN 条件。 • 6. 此信息仅供内部使用。

列名称	数据类型	描述
		• 7. 此信息仅供内部使用。 • 8. 此信息仅供内部使用。 • 9. 此信息仅供内部使用。 • 10. 此信息仅供内部使用。 • 11. 此信息仅供内部使用。 • 12. 此信息仅供内部使用。
hash_looped	character(1)	此信息仅供内部使用。
switched_parts	character(1)	指示构建（或外部）和探测（内部）端是否已切换。
used_prefetching	character(1)	此信息仅供内部使用。
hash_segment	integer	相应哈希步骤的分段。
hash_step	integer	相应哈希步骤的步骤数。
checksum	bigint	此信息仅供内部使用。
分配	integer	此信息仅供内部使用。

示例查询

以下示例返回在查询 720 的哈希联接中使用的分区的数量。

```
select query, slice, tbl, num_parts
from stl_hashjoin
where query=720 limit 10;
```

```
 query | slice | tbl | num_parts
-------+-------+-----+-----------
   720 |     0 | 243 |         1
   720 |     1 | 243 |         1
```

```
(2 rows)
```

STL_INSERT

分析查询的插入执行步骤。

STL_INSERT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_INSERT 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。

列名称	数据类型	描述
rows	bigint	处理的总行数。
tbl	integer	表 ID。
inserted_mega_value	character(1)	此信息仅供内部使用。此信息显示给定的插入步骤是否插入了大值。大值将存储在多个块中。原定设置情况下，块大小为 1 MB，大值大于原定设置中的 1 MB。

示例查询

以下示例返回最近查询的插入执行步骤。

```
select slice, segment, step, tasknum, rows, tbl
from stl_insert
where query=pg_last_query_id();
```

```
 slice | segment | step | tasknum | rows  |  tbl
-------+---------+------+---------+-------+--------
     0 |       2 |    2 |      15 | 24958 | 100548
     1 |       2 |    2 |      15 | 25032 | 100548
(2 rows)
```

STL_LIMIT

分析在 SELECT 查询中使用 LIMIT 子句时发生的执行步骤。

STL_LIMIT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_LIMIT 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
checksum	bigint	此信息仅供内部使用。

示例查询

为了在 STL_LIMIT 中生成行，此示例首先使用 LIMIT 子句对 VENUE 表运行了以下查询。

```
select * from venue
order by 1
limit 10;
```

```
 venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
       1 | Toyota Park                | Bridgeview      | IL         |          0
       2 | Columbus Crew Stadium      | Columbus        | OH         |          0
```

```
       3 | RFK Stadium                | Washington      | DC         |          0
       4 | CommunityAmerica Ballpark  | Kansas City     | KS         |          0
       5 | Gillette Stadium           | Foxborough      | MA         |      68756
       6 | New York Giants Stadium    | East Rutherford | NJ         |      80242
       7 | BMO Field                  | Toronto         | ON         |          0
       8 | The Home Depot Center      | Carson          | CA         |          0
       9 | Dick's Sporting Goods Park | Commerce City   | CO         |          0
      10 | Pizza Hut Park             | Frisco          | TX         |          0
(10 rows)
```

然后，运行以下查询来查找您已对 VENUE 表运行的最后一个查询的查询 ID。

```
select max(query)
from stl_query;
```

```
  max
--------
 127128
(1 row)
```

或者，您可以运行以下查询来验证查询 ID 是否对应于之前运行的 LIMIT 查询。

```
select query, trim(querytxt)
from stl_query
where query=127128;
```

```
 query  |                  btrim
--------+-----------------------------------------
 127128 | select * from venue order by 1 limit 10;
(1 row)
```

最后，运行以下查询来返回有关 STL_LIMIT 表中的 LIMIT 查询的信息。

```
select slice, segment, step, starttime, endtime, tasknum
from stl_limit
where query=127128
order by starttime, endtime;
```

```
  slice | segment | step |         starttime          |          endtime           |
 tasknum
```

```
 -------+---------+------+---------------------------+---------------------------
+---------
      1 |       1 |    3 | 2013-09-06 22:56:43.608114 | 2013-09-06 22:56:43.609383 |
   15
      0 |       1 |    3 | 2013-09-06 22:56:43.608708 | 2013-09-06 22:56:43.609521 |
   15
  10000 |       2 |    2 | 2013-09-06 22:56:43.612506 | 2013-09-06 22:56:43.612668 |
    0
(3 rows)
```

STL_LOAD_COMMITS

返回用于跟踪或排查数据加载的信息。

此视图记录了在每个数据文件加载到数据库表中时的进度。

STL_LOAD_COMMITS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_LOAD_COMMITS 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_LOAD_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	为此条目加载的切片。
名称	character(256)	System-defined value。
filename	character(256)	所跟踪的文件的名称。
byte_offset	integer	此信息仅供内部使用。

列名称	数据类型	描述
lines_scanned	integer	从加载文件中扫描的行数。此数字可能与实际加载的行数不匹配。例如，根据 COPY 命令中的 MAXERROR 选项，加载操作可能会扫描但会容忍大量的错误记录。
errors	integer	此信息仅供内部使用。
curtime	timestamp	上次更新此条目的时间。
status	integer	此信息仅供内部使用。
file_format	character(16)	加载文件的格式。可能值如下所示： • Avro • JSON • ORC • Parquet • 文本
is_partial	integer	值，如果为真 (1) 表示在 COPY 操作期间输入文件被拆分为范围。如果此值为假 (0)，则不会拆分输入文件。
start_offset	bigint	值，如果在 COPY 操作期间拆分输入文件，则表示拆分的偏移值（以字节为单位）。每个文件拆分都记录为具有相应 start_offset 值的单独记录。如果文件未拆分，则此值为 0。
copy_job_id	bigint	复制作业标识符。0 表示没有作业标识符。

示例查询

以下示例返回上次 COPY 操作的详细信息。

```
select query, trim(filename) as file, curtime as updated
from stl_load_commits
where query = pg_last_copy_id();

 query |               file                |          updated
-------+-----------------------------------+----------------------------
```

```
 28554 | s3://dw-tickit/category_pipe.txt | 2013-11-01 17:14:52.648486
(1 row)
```

以下查询包含 TICKIT 数据库中初次加载表时的条目：

```
select query, trim(filename), curtime
from stl_load_commits
where filename like '%tickit%' order by query;
```

```
 query |           btrim           |          curtime
-------+---------------------------+---------------------------
 22475 | tickit/allusers_pipe.txt  | 2013-02-08 20:58:23.274186
 22478 | tickit/venue_pipe.txt     | 2013-02-08 20:58:25.070604
 22480 | tickit/category_pipe.txt  | 2013-02-08 20:58:27.333472
 22482 | tickit/date2008_pipe.txt  | 2013-02-08 20:58:28.608305
 22485 | tickit/allevents_pipe.txt | 2013-02-08 20:58:29.99489
 22487 | tickit/listings_pipe.txt  | 2013-02-08 20:58:37.632939
 22593 | tickit/allusers_pipe.txt  | 2013-02-08 21:04:08.400491
 22596 | tickit/venue_pipe.txt     | 2013-02-08 21:04:10.056055
 22598 | tickit/category_pipe.txt  | 2013-02-08 21:04:11.465049
 22600 | tickit/date2008_pipe.txt  | 2013-02-08 21:04:12.461502
 22603 | tickit/allevents_pipe.txt | 2013-02-08 21:04:14.785124
 22605 | tickit/listings_pipe.txt  | 2013-02-08 21:04:20.170594

(12 rows)
```

向此系统视图的日志文件写入一个记录的事实并不表示加载操作已作为其所属的事务的一部分成功提交。要验证加载提交，请查询 STL_UTILITYTEXT 视图并查找与 COPY 事务对应的 COMMIT 记录。例如，此查询基于针对 STL_UTILITYTEXT 的子查询联接 STL_LOAD_COMMITS 和 STL_QUERY：

```
select l.query,rtrim(l.filename),q.xid
from stl_load_commits l, stl_query q
where l.query=q.query
and exists
(select xid from stl_utilitytext where xid=q.xid and rtrim("text")='COMMIT');

 query |           rtrim           |  xid
-------+---------------------------+-------
 22600 | tickit/date2008_pipe.txt  | 68311
 22480 | tickit/category_pipe.txt  | 68066
  7508 | allusers_pipe.txt         | 23365
  7552 | category_pipe.txt         | 23415
```

```
  7576 | allevents_pipe.txt        | 23429
  7516 | venue_pipe.txt            | 23390
  7604 | listings_pipe.txt         | 23445
 22596 | tickit/venue_pipe.txt     | 68309
 22605 | tickit/listings_pipe.txt  | 68316
 22593 | tickit/allusers_pipe.txt  | 68305
 22485 | tickit/allevents_pipe.txt | 68071
  7561 | allevents_pipe.txt        | 23429
  7541 | category_pipe.txt         | 23415
  7558 | date2008_pipe.txt         | 23428
 22478 | tickit/venue_pipe.txt     | 68065
   526 | date2008_pipe.txt         |  2572
  7466 | allusers_pipe.txt         | 23365
 22482 | tickit/date2008_pipe.txt  | 68067
 22598 | tickit/category_pipe.txt  | 68310
 22603 | tickit/allevents_pipe.txt | 68315
 22475 | tickit/allusers_pipe.txt  | 68061
   547 | date2008_pipe.txt         |  2572
 22487 | tickit/listings_pipe.txt  | 68072
  7531 | venue_pipe.txt            | 23390
  7583 | listings_pipe.txt         | 23445
(25 rows)
```

以下示例突出显示 is_partial 和 start_offset 列值。

```
-- Single large file copy without scan range
SELECT count(*) FROM stl_load_commits WHERE query = pg_last_copy_id();
1

-- Single large uncompressed, delimited file copy with scan range
SELECT count(*) FROM stl_load_commits WHERE query = pg_last_copy_id();
16

-- Scan range offset logging in the file at 64MB boundary.
SELECT start_offset FROM stl_load_commits
WHERE query = pg_last_copy_id() ORDER BY start_offset;
0
67108864
134217728
201326592
268435456
335544320
402653184
```

```
469762048
536870912
603979776
671088640
738197504
805306368
872415232
939524096
1006632960
```

STL_LOAD_ERRORS

显示所有 Amazon Redshift 加载错误的记录。

STL_LOAD_ERRORS 包含所有 Amazon Redshift 加载错误的历史记录。有关可能的加载错误和说明的全面列表，请参阅加载错误参考。

在您查询 STL_LOAD_ERRORS 以了解有关错误的一般信息后，可查询 STL_LOADERROR_DETAIL 以获取其他详细信息，如发生解析错误的准确的数据行和列。

STL_LOAD_ERRORS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_LOAD_ERRORS 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_LOAD_ERROR_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
slice	integer	发生错误的切片。
tbl	integer	表 ID。
starttime	timestamp	加载的开始时间（采用 UTC 表示）。

列名称	数据类型	描述
session	integer	执行加载的会话的会话 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
filename	character(256)	加载的输入文件的完整路径。
line_number	bigint	加载文件中的错误所在的行号。对于 JSON 中的 COPY，这是存在错误的 JSON 对象的最后一行的行号。
colname	character(127)	存在错误的字段。
type	character(10)	字段的数据类型。
col_length	character(10)	列长度（如果适用）。当数据类型具有限制长度时填充此字段。例如，对于数据类型为“character(3)”的列，此列将包含值“3”。
position	integer	字段中的错误的位置。
raw_line	character(1024)	包含错误的原始加载数据。加载数据中的多字节字符替换为句点。
raw_field_value	char(1024)	字段“colname”的预解析值，可导致解析错误。
err_code	integer	错误代码。
err_reason	character(100)	有关错误的说明。
is_partial	integer	值，如果为真 (1) 表示在 COPY 操作期间输入文件被拆分为范围。如果此值为假 (0)，则不会拆分输入文件。
start_offset	bigint	值，如果在 COPY 操作期间拆分输入文件，则表示拆分的偏移值（以字节为单位）。如果文件中的行号未知，则行号为 -1。如果文件未拆分，则此值为 0。
copy_job_id	bigint	复制作业标识符。0 表示没有作业标识符。

示例查询

以下查询将 STL_LOAD_ERRORS 联接到 STL_LOADERROR_DETAIL 以查看最近加载期间发生的错误的详细信息。

```
select d.query, substring(d.filename,14,20),
d.line_number as line,
substring(d.value,1,16) as value,
substring(le.err_reason,1,48) as err_reason
from stl_loaderror_detail d, stl_load_errors le
where d.query = le.query
and d.query = pg_last_copy_id();

 query |    substring      | line |  value   |             err_reason
-------+-------------------+------+----------+----------------------------
    558| allusers_pipe.txt |  251 | 251      | String contains invalid or
                                              unsupported UTF8 code
    558| allusers_pipe.txt |  251 | ZRU29FGR | String contains invalid or
                                              unsupported UTF8 code
    558| allusers_pipe.txt |  251 | Kaitlin  | String contains invalid or
                                              unsupported UTF8 code
    558| allusers_pipe.txt |  251 | Walter   | String contains invalid or
                                              unsupported UTF8 code
```

以下示例结合使用 STL_LOAD_ERRORS 和 STV_TBL_PERM 来创建新视图，然后使用该视图来确定在将数据加载到 EVENT 表中时发生的错误：

```
create view loadview as
(select distinct tbl, trim(name) as table_name, query, starttime,
trim(filename) as input, line_number, colname, err_code,
trim(err_reason) as reason
from stl_load_errors sl, stv_tbl_perm sp
where sl.tbl = sp.id);
```

接下来，以下查询实际返回在加载 EVENT 表时发生的最后一个错误：

```
select table_name, query, line_number, colname, starttime,
trim(reason) as error
from loadview
where table_name ='event'
order by line_number limit 1;
```

查询返回 EVENT 表发生的最后一个加载错误。如果未发生任何加载错误，则查询返回零行。在此示例中，查询返回一个错误：

```
 table_name | query | line_number | colname | error | starttime
------+-----+----+----+-----------------------------------------------------------
+----------------------
event | 309 |  0 |  5 | Error in Timestamp value or format [%Y-%m-%d %H:%M:%S] |
 2014-04-22 15:12:44

(1 row)
```

在 COPY 命令自动拆分大的、未压缩的、以文本分隔的文件数据以促进并行化的情况下，line_number、is_partial 和 start_offset 列显示与拆分有关的信息。（在原始文件中行号不可用的情况下，行号可能是未知的。）

```
--scan ranges information
SELECT line_number, POSITION, btrim(raw_line), btrim(raw_field_value),
btrim(err_reason), is_partial, start_offset FROM stl_load_errors
WHERE query = pg_last_copy_id();

--result
-1,51,"1008771|13463413|463414|2|28.00|38520.72|0.06|0.07|NO|1998-08-30|1998-09-25|
1998-09-04|TAKE BACK RETURN|RAIL|ans cajole sly","NO","Char length exceeds DDL
 length",1,67108864
```

STL_LOADERROR_DETAIL

显示使用 COPY 命令加载表时发生的数据解析错误的日志记录。为了节省磁盘空间，对于每个加载操作，每个节点切片最多记录 20 个错误。

在将数据行中的某个字段加载到表时，如果 Amazon Redshift 无法解析该字段，则会发生解析错误。例如，如果表列应为整数数据类型，而且数据文件包含该字段中的字母字符串，则它会导致解析错误。

在您查询 STL_LOAD_ERRORS 以了解有关错误的一般信息之后，请查询 STL_LOADERROR_DETAIL 以获取其他详细信息，如发生解析错误的准确的数据行和列。

STL_LOADERROR_DETAIL 视图包含所有数据列，其中包括解析错误发生的列及其之前的列。使用 VALUE 字段来查看实际已在此列（包括在发生此错误之前已正确解析的列）中解析的数据值。

此视图对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_LOADERROR_DETAIL 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_LOAD_ERROR_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
slice	integer	发生错误的切片。
session	integer	执行加载的会话的会话 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
filename	character(256)	加载的输入文件的完整路径。
line_number	bigint	加载文件中的错误所在的行号。
field	integer	存在错误的字段。
colname	character(1024)	列名称。
值	character(1024)	字段的已解析数据值。（可能已被截断。）加载数据中的多字节字符替换为句点。
is_null	integer	解析的值是否为 null。
type	character(10)	字段的数据类型。
col_length	character(10)	列长度（如果适用）。当数据类型具有限制长度时填充此字段。例如，对于数据类型为“character(3)”的列，此列将包含值“3”。

示例查询

以下查询将 STL_LOAD_ERRORS 联接到 STL_LOADERROR_DETAIL 以查看加载 EVENT 表（表 ID 为 100133）时发生的解析错误的详细信息：

```
select d.query, d.line_number, d.value,
le.raw_line, le.err_reason
from stl_loaderror_detail d, stl_load_errors le
where
d.query = le.query
and tbl = 100133;
```

以下示例输出显示已加载成功的列，包括存在错误的列。在此示例中，在第三个列中发生解析错误之前已成功加载两个列，第三个列中应为整数的字段不正确地解析为了字符串。由于该字段应为整数，因此它将字符串“aaa”（未初始化的数据）解析为了 null 并生成了解析错误。输出显示原始值、解析的值和错误原因：

```
query  | line_number | value | raw_line | err_reason
-------+-------------+-------+----------+----------------
4      |      3      |  1201 |  1201    | Invalid digit
4      |      3      |   126 |   126    | Invalid digit
4      |      3      |       |   aaa    | Invalid digit
(3 rows)
```

当查询联接 STL_LOAD_ERRORS 和 STL_LOADERROR_DETAIL 时，它将显示数据行中每列的错误原因，这只是意味着该行中发生了错误。结果中的最后一行是发生解析错误的实际列。

STL_MERGE

分析查询的合并执行步骤。在合并并行操作（如排序和联接）的结果以进行后续处理时，将会发生这些步骤。

STL_MERGE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_MERGE 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。

示例查询

以下示例返回 10 个合并执行结果。

```
select query, step, starttime, endtime, tasknum, rows
from stl_merge
limit 10;
```

```
 query | step |       starttime     |        endtime      | tasknum | rows
-------+------+---------------------+---------------------+---------+------
     9 |    0 | 2013-08-12 20:08:14 | 2013-08-12 20:08:14 |       0 |    0
    12 |    0 | 2013-08-12 20:09:10 | 2013-08-12 20:09:10 |       0 |    0
    15 |    0 | 2013-08-12 20:10:24 | 2013-08-12 20:10:24 |       0 |    0
    20 |    0 | 2013-08-12 20:11:27 | 2013-08-12 20:11:27 |       0 |    0
```

```
    26 |    0 | 2013-08-12 20:12:28 | 2013-08-12 20:12:28 |       0 |    0
    32 |    0 | 2013-08-12 20:14:33 | 2013-08-12 20:14:33 |       0 |    0
    38 |    0 | 2013-08-12 20:16:43 | 2013-08-12 20:16:43 |       0 |    0
    44 |    0 | 2013-08-12 20:17:05 | 2013-08-12 20:17:05 |       0 |    0
    50 |    0 | 2013-08-12 20:18:48 | 2013-08-12 20:18:48 |       0 |    0
    56 |    0 | 2013-08-12 20:20:48 | 2013-08-12 20:20:48 |       0 |    0
(10 rows)
```

STL_MERGEJOIN

分析查询的合并联接步骤。

STL_MERGEJOIN 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_MERGEJOIN 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。

列名称	数据类型	描述
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
tbl	integer	表 ID。这是合并联接中使用的内部表的 ID。
checksum	bigint	此信息仅供内部使用。

示例查询

以下示例返回最近查询的合并联接结果。

```
select sum(s.qtysold), e.eventname
from event e, listing l, sales s
where e.eventid=l.eventid
and l.listid= s.listid
group by e.eventname;

select * from stl_mergejoin where query=pg_last_query_id();
```

```
 userid | query | slice | segment | step |         starttime   |          endtime    |
 tasknum | rows | tbl
--------+-------+-------+---------+------+---------------------+---------------------
+---------+------+-----
    100 | 27399 |     3 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |43428 | 240
    100 | 27399 |     0 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |43159 | 240
    100 | 27399 |     2 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |42778 | 240
    100 | 27399 |     1 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |43091 | 240
```

STL_MV_STATE

STL_MV_STATE 视图对于实体化视图的每次状态转换包含一行。

有关实体化视图的更多信息，请参阅在 Amazon Redshift 中创建实体化视图。

STL_MV_STATE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_MV_STATE 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	bigint	创建事件的用户的 ID。
starttime	timestamp	事件的开始时间。
xid	bigint	事件的事务 ID。
event_desc	char(500)	提示状态更改的事件。一些示例值包括： • 列类型已更改 • 列已被删除 • 列已重命名 • 架构名称已更改 • 小型表转换 • TRUNCATE • Vacuum 请注意，此列有其它可能的值。
db_name	char(128)	包含实体化视图的数据库。
base_table_schema	char(128)	基表的架构。

列名称	数据类型	描述
base_table_name	char(128)	基表的名称。
mv_schema	char(128)	实体化视图的架构。
mv_name	char(128)	实体化视图的名称。
state	character(32)	实体化视图的已更改状态如下所示： • 重新计算 • 无法刷新

下表显示了 event_desc 和 state 的示例组合。

```
          event_desc     |     state
-------------------------+---------------
 TRUNCATE                | Recompute
 TRUNCATE                | Recompute
 Small table conversion  | Recompute
 Vacuum                  | Recompute
 Column was renamed      | Unrefreshable
 Column was dropped      | Unrefreshable
 Table was renamed       | Unrefreshable
 Column type was changed | Unrefreshable
 Schema name was changed | Unrefreshable
```

示例查询

要查看实体化视图的状态转换的日志，请运行以下查询。

```
select * from stl_mv_state;
```

此查询返回以下示例输出：

```
 userid |         starttime          | xid  |            event_desc       | db_name |
 base_table_schema   |   base_table_name    |      mv_schema       | mv_name       |
  state
--------+----------------------------+------+-----------------------------+---------
+---------------------+---------------------+---------------------+--------------
+---------------
    138 | 2020-02-14 02:21:25.578885 | 5180 | TRUNCATE                    | dev     |
 public               | mv_base_table        | public               | mv_test       |
 Recompute
    138 | 2020-02-14 02:21:56.846774 | 5275 | Column was dropped          | dev     |
                      | mv_base_table        | public               | mv_test       |
 Unrefreshable
    100 | 2020-02-13 22:09:53.041228 | 1794 | Column was renamed          | dev     |
                      | mv_base_table        | public               | mv_test       |
 Unrefreshable
      1 | 2020-02-13 22:10:23.630914 | 1893 | ALTER TABLE ALTER SORTKEY   | dev     |
 public               | mv_base_table_sorted | public               | mv_test       |
 Recompute
      1 | 2020-02-17 22:57:22.497989 | 8455 | ALTER TABLE ALTER DISTSTYLE | dev     |
 public               | mv_base_table        | public               | mv_test       |
 Recompute
    173 | 2020-02-17 22:57:23.591434 | 8504 | Table was renamed           | dev     |
                      | mv_base_table        | public               | mv_test       |
 Unrefreshable
    173 | 2020-02-17 22:57:27.229423 | 8592 | Column type was changed     | dev     |
                      | mv_base_table        | public               | mv_test       |
 Unrefreshable
    197 | 2020-02-17 22:59:06.212569 | 9668 | TRUNCATE                    | dev     |
 schemaf796e415850f4f | mv_base_table        | schemaf796e415850f4f | mv_test       |
 Recompute
    138 | 2020-02-14 02:21:55.705655 | 5226 | Column was renamed          | dev     |
                      | mv_base_table        | public               | mv_test       |
 Unrefreshable
      1 | 2020-02-14 02:22:26.292434 | 5325 | ALTER TABLE ALTER SORTKEY   | dev     |
 public               | mv_base_table_sorted | public               | mv_test       |
 Recompute
```

STL_NESTLOOP

分析查询的嵌套循环的联接执行步骤。

STL_NESTLOOP 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_NESTLOOP 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
tbl	integer	表 ID。
checksum	bigint	此信息仅供内部使用。

示例查询

由于以下查询忽略了联接 CATEGORY 表，因此它会生成一个部分笛卡尔积（不推荐这样做）。这里只是为了说明嵌套循环。

```
select count(event.eventname), event.eventname, category.catname, date.caldate
from event, category, date
where event.dateid = date.dateid
group by event.eventname, category.catname, date.caldate;
```

以下查询显示来自 STL_NESTLOOP 视图中的上个查询的结果。

```
select query, slice, segment as seg, step,
datediff(msec, starttime, endtime) as duration, tasknum, rows, tbl
from stl_nestloop
where query = pg_last_query_id();
```

```
 query | slice | seg | step | duration | tasknum | rows  | tbl
-------+-------+-----+------+----------+---------+-------+-----
  6028 |     0 |   4 |    5 |       41 |      22 | 24277 | 240
  6028 |     1 |   4 |    5 |       26 |      23 | 24189 | 240
  6028 |     3 |   4 |    5 |       25 |      23 | 24376 | 240
  6028 |     2 |   4 |    5 |       54 |      22 | 23936 | 240
```

STL_PARSE

分析将字符串解析为二进制值以进行加载的查询步骤。

STL_PARSE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_PARSE 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。

示例查询

以下示例返回切片 1 和分段 0 的所有查询步骤结果，其中的字符串已解析为二进制值。

```
select query, step, starttime, endtime, tasknum, rows
from stl_parse
where slice=1 and segment=0;
```

```
 query | step |     starttime       |        endtime      | tasknum |  rows
-------+------+---------------------+---------------------+---------+--------
   669 |    1 | 2013-08-12 22:35:13 | 2013-08-12 22:35:17 |      32 | 192497
   696 |    1 | 2013-08-12 22:35:49 | 2013-08-12 22:35:49 |      32 |      0
   525 |    1 | 2013-08-12 22:32:03 | 2013-08-12 22:32:03 |      13 |  49990
   585 |    1 | 2013-08-12 22:33:18 | 2013-08-12 22:33:19 |      13 |    202
```

```
   621 |    1 | 2013-08-12 22:34:03 | 2013-08-12 22:34:03 |      27 |    365
   651 |    1 | 2013-08-12 22:34:47 | 2013-08-12 22:34:53 |      35 | 192497
   590 |    1 | 2013-08-12 22:33:28 | 2013-08-12 22:33:28 |      19 |      0
   599 |    1 | 2013-08-12 22:33:39 | 2013-08-12 22:33:39 |      31 |     11
   675 |    1 | 2013-08-12 22:35:26 | 2013-08-12 22:35:27 |      38 |   3766
   567 |    1 | 2013-08-12 22:32:47 | 2013-08-12 22:32:48 |      23 |  49990
   630 |    1 | 2013-08-12 22:34:17 | 2013-08-12 22:34:17 |      36 |      0
   572 |    1 | 2013-08-12 22:33:04 | 2013-08-12 22:33:04 |      29 |      0
   645 |    1 | 2013-08-12 22:34:37 | 2013-08-12 22:34:38 |      29 |   8798
   604 |    1 | 2013-08-12 22:33:47 | 2013-08-12 22:33:47 |      37 |      0
(14 rows)
```

STL_PLAN_INFO

使用 STL_PLAN_INFO 视图以一组行的形式查看一个查询的 EXPLAIN 输出。这是查看查询计划的一种替代方法。

STL_PLAN_INFO 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_PLAN_INFO 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
nodeid	integer	计划节点标识符，在执行查询时一个节点将映射到一个或多个步骤。
segment	integer	标识查询区段的数字。
step	integer	标识查询步骤的数字。

列名称	数据类型	描述
locus	integer	步骤运行的位置。如果在计算节点上，则为 0；如果在领导节点上，则为 1。
plannode	integer	计划节点的枚举值。查看以下表可了解 plannode 的枚举。（STL_EXPLAIN 中的 PLANNODE 列包含计划节点文本。）
startupcost	double precision	此步骤中返回第一行的估计相对成本。
totalcost	double precision	执行此步骤的估计相对成本。
rows	bigint	将在此步骤中生成的估计行数。
bytes	bigint	将在此步骤中生成的估计字节数。

示例查询

以下示例比较了通过使用 EXPLAIN 命令和通过查询 STL_PLAN_INFO 视图返回的简单 SELECT 查询的查询计划。

```
explain select * from category;
QUERY PLAN
------------------------------------------------------------
XN Seq Scan on category (cost=0.00..0.11 rows=11 width=49)
(1 row)

select * from category;
catid | catgroup | catname | catdesc
-------+----------+-----------+--------------------------------------------
1 | Sports | MLB | Major League Baseball
3 | Sports | NFL | National Football League
5 | Sports | MLS | Major League Soccer
...

select * from stl_plan_info where query=256;

query | nodeid | segment | step | locus | plannode | startupcost | totalcost
| rows | bytes
-------+--------+---------+------+-------+----------+-------------+-----------+------
+-------
```

```
256 | 1 | 0 | 1 | 0 | 104 | 0 | 0.11 | 11 | 539
256 | 1 | 0 | 0 | 0 | 104 | 0 | 0.11 | 11 | 539
(2 rows)
```

在本示例中，PLANNODE 104 表示 CATEGORY 表的顺序扫描。

```
select distinct eventname from event order by 1;

eventname
----------------------------------------------------------------------
.38 Special
3 Doors Down
70s Soul Jam
A Bronx Tale
...

explain select distinct eventname from event order by 1;

QUERY PLAN
------------------------------------------------------------------------------------------
XN Merge (cost=1000000000136.38..1000000000137.82 rows=576 width=17)
Merge Key: eventname
-> XN Network (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Send to leader
-> XN Sort (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Sort Key: eventname
-> XN Unique (cost=0.00..109.98 rows=576 width=17)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=17)
(8 rows)

select * from stl_plan_info where query=240 order by nodeid desc;

query | nodeid | segment | step | locus | plannode | startupcost |
totalcost | rows | bytes
-------+--------+---------+------+-------+----------+-----------------
+------------------+------+--------
240 | 5 | 0 | 0 | 0 | 104 | 0               | 87.98   | 8798 | 149566
240 | 5 | 0 | 1 | 0 | 104 | 0               | 87.98   | 8798 | 149566
240 | 4 | 0 | 2 | 0 | 117 | 0               | 109.975 | 576  | 9792
240 | 4 | 0 | 3 | 0 | 117 | 0               | 109.975 | 576  | 9792
```

```
240 | 4 | 1 | 0 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 1 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 3 | 1 | 2 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 3 | 2 | 0 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 2 | 2 | 1 | 0 | 123 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 1 | 3 | 0 | 0 | 122 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
(10 rows)
```

STL_PROJECT

包含用于计算表达式的查询步骤的行。

STL_PROJECT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_PROJECT 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。

列名称	数据类型	描述
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
checksum	bigint	此信息仅供内部使用。

示例查询

以下示例返回已用于计算切片 0 和分段 1 的表达式的查询步骤的所有行。

```
select query, step, starttime, endtime, tasknum, rows
from stl_project
where slice=0 and segment=1;
```

```
 query  | step |         starttime   |          endtime    | tasknum | rows
--------+------+---------------------+---------------------+---------+------
  86399 |    2 | 2013-08-29 22:01:21 | 2013-08-29 22:01:21 |      25 |   -1
  86399 |    3 | 2013-08-29 22:01:21 | 2013-08-29 22:01:21 |      25 |   -1
    719 |    1 | 2013-08-12 22:38:33 | 2013-08-12 22:38:33 |       7 |   -1
  86383 |    1 | 2013-08-29 21:58:35 | 2013-08-29 21:58:35 |       7 |   -1
    714 |    1 | 2013-08-12 22:38:17 | 2013-08-12 22:38:17 |       2 |   -1
  86375 |    1 | 2013-08-29 21:57:59 | 2013-08-29 21:57:59 |       2 |   -1
  86397 |    2 | 2013-08-29 22:01:20 | 2013-08-29 22:01:20 |      19 |   -1
    627 |    1 | 2013-08-12 22:34:13 | 2013-08-12 22:34:13 |      34 |   -1
  86326 |    2 | 2013-08-29 21:45:28 | 2013-08-29 21:45:28 |      34 |   -1
  86326 |    3 | 2013-08-29 21:45:28 | 2013-08-29 21:45:28 |      34 |   -1
  86325 |    2 | 2013-08-29 21:45:27 | 2013-08-29 21:45:27 |      28 |   -1
  86371 |    1 | 2013-08-29 21:57:42 | 2013-08-29 21:57:42 |       4 |   -1
 111100 |    2 | 2013-09-03 19:04:45 | 2013-09-03 19:04:45 |      12 |   -1
    704 |    2 | 2013-08-12 22:36:34 | 2013-08-12 22:36:34 |      37 |   -1
    649 |    2 | 2013-08-12 22:34:47 | 2013-08-12 22:34:47 |      38 |   -1
    649 |    3 | 2013-08-12 22:34:47 | 2013-08-12 22:34:47 |      38 |   -1
    632 |    2 | 2013-08-12 22:34:22 | 2013-08-12 22:34:22 |      13 |   -1
    705 |    2 | 2013-08-12 22:36:48 | 2013-08-12 22:36:49 |      13 |   -1
    705 |    3 | 2013-08-12 22:36:48 | 2013-08-12 22:36:49 |      13 |   -1
```

```
      3 |    1 | 2013-08-12 20:07:40 | 2013-08-12 20:07:40 |       3 |   -1
  86373 |    1 | 2013-08-29 21:57:58 | 2013-08-29 21:57:58 |       3 |   -1
 107976 |    1 | 2013-09-03 04:05:12 | 2013-09-03 04:05:12 |       3 |   -1
  86381 |    1 | 2013-08-29 21:58:35 | 2013-08-29 21:58:35 |       8 |   -1
  86396 |    1 | 2013-08-29 22:01:20 | 2013-08-29 22:01:20 |      15 |   -1
    711 |    1 | 2013-08-12 22:37:10 | 2013-08-12 22:37:10 |      20 |   -1
  86324 |    1 | 2013-08-29 21:45:27 | 2013-08-29 21:45:27 |      24 |   -1
(26 rows)
```

STL_QUERY

返回有关数据库查询的执行信息。

> **Note**
>
> STL_QUERY 和 STL_QUERYTEXT 视图仅包含有关查询的信息，不包含有关其他实用工具和 DDL 命令的信息。对于有关 Amazon Redshift 运行的所有语句的列表和信息，您还可以查询 STL_DDLTEXT 和 STL_UTILITYTEXT 视图。有关 Amazon Redshift 运行的所有语句的完整列表，您可以查询 SVL_STATEMENTTEXT 视图。

STL_QUERY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
label	character(320)	用于运行查询的文件的名称或使用 SET QUERY_GROUP 命令定义的标签。如果查询并非基于文件或未设置 QUERY_GROUP 参数，则此字段值为 `default`。
xid	bigint	事务 ID。

列名称	数据类型	描述
pid	integer	进程 ID。一般情况下，会话中的所有查询在同一进程中运行，因此，如果您在同一会话中运行一系列查询，则此值通常保持不变。在特定的内部事件之后，Amazon Redshift 可能会重新启动一个活动会话并分配新的 PID。有关更多信息，请参阅 STL_RESTARTED_SESSIONS。
数据库	character(32)	在发起查询时用户连接到的数据库的名称。
querytxt	character(4000)	查询的实际查询文本。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
aborted	integer	如果查询已由系统停止或已由用户取消，则此列包含 **1**。如果查询已完成（包括将结果返回到客户端），则此列包含 **0**。如果客户端在接收结果之前断开连接，则查询将被标记为已取消 (**1**)，即使查询已在后端成功完成也是如此。
insert_pristine	integer	当前查询正在运行时，是否可以运行写查询。1 = 不允许写查询。0 = 允许写查询。此列专用在调试中。
concurrency_scaling_status	integer	指示查询运行在主集群还是并发扩展集群上。可能值如下所示： 0 - 运行在主集群上 1 - 运行在并发扩展集群上 > 1 - 运行在主集群上

示例查询

以下示例列出最近的 5 个查询。

```
select query, trim(querytxt) as sqlquery
from stl_query
order by query desc limit 5;

query |                                  sqlquery
------+--------------------------------------------------
129 | select query, trim(querytxt) from stl_query order by query;
128 | select node from stv_disk_read_speeds;
127 | select system_status from stv_gui_status
126 | select * from systable_topology order by slice
125 | load global dict registry
(5 rows)
```

以下查询按降序返回在 2013 年 2 月 15 日运行的查询所消耗的时间。

```
select query, datediff(seconds, starttime, endtime),
trim(querytxt) as sqlquery
from stl_query
where starttime >= '2013-02-15 00:00' and endtime < '2013-02-16 00:00'
order by date_diff desc;

 query | date_diff |  sqlquery
-------+-----------+-------------------------------------------
 55    |       119 | padb_fetch_sample: select count(*) from category
121    |         9 | select * from svl_query_summary;
181    |         6 | select * from svl_query_summary where query in(179,178);
172    |         5 | select * from svl_query_summary where query=148;
...
(189 rows)
```

以下查询显示查询的队列时间和执行时间。concurrency_scaling_status = 1 的查询运行在并发扩展集群上。所有其他查询都运行在主集群上。

```
SELECT w.service_class AS queue
     , q.concurrency_scaling_status
     , COUNT( * ) AS queries
     , SUM( q.aborted )  AS aborted
     , SUM( ROUND( total_queue_time::NUMERIC / 1000000,2 ) ) AS queue_secs
```

```
     , SUM( ROUND( total_exec_time::NUMERIC / 1000000,2 ) )  AS exec_secs
FROM stl_query q
     JOIN stl_wlm_query w
          USING (userid,query)
WHERE q.userid > 1
  AND service_class > 5
  AND q.starttime > '2019-03-01 16:38:00'
  AND q.endtime   < '2019-03-01 17:40:00'
GROUP BY 1,2
ORDER BY 1,2;
```

STL_QUERY_METRICS

包含已完成在用户定义的查询队列中运行的查询的指标信息（如处理的行数、CPU 利用率、输入/输出和磁盘利用率）。要查看当前运行的活动查询的指标，请参阅 STV_QUERY_METRICS 系统视图。

查询指标按一秒的间隔采样。因此，同一查询的不同运行可能返回稍微不同的时间。此外，运行不到 1 秒的查询段可能不会记录。

STL_QUERY_METRICS 可跟踪和聚合查询、段和步骤级别的指标。有关查询段和步骤的信息，请参阅查询计划和执行工作流程。很多指标（如 max_rows、cpu_time 等）是跨节点切片进行合计的。有关节点切片的更多信息，请参阅数据仓库系统架构。

要确定行在哪个级别报告指标，请检查 segment 和 step_type 列。

- 如果 segment 和 step_type 均为 -1，则行在查询级别报告指标。
- 如果 segment 不为 -1，而 step_type 为 -1，则行在段级别报告指标。
- 如果 segment 和 step_type 均不为 -1，则行在步骤级别报告指标。

SVL_QUERY_METRICS 视图和 SVL_QUERY_METRICS_SUMMARY 视图将聚合此视图中的数据并以更容易访问的形式呈现信息。

STL_QUERY_METRICS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	运行生成条目的查询的用户的 ID。
service_class	integer	服务类的 ID。查询队列在 WLM 配置中定义。仅对用户定义的队列报告的指标。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
segment	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。查询段可并行运行。每个段在一个进程中运行。如果段值为 -1，则指标段值将汇总到查询级别。
step_type	integer	运行的步骤的类型。有关步骤类型的说明，请参阅步骤类型。
starttime	timestamp	开始执行查询的时间（用 UTC 表示），有 6 位数字精度，可精确到小数秒。例如：2009-06-12 11:29:19.131358 。
切片	integer	集群的切片数。
max_rows	bigint	某个步骤的最大行输出数（跨所有切片进行聚合）。
rows	bigint	步骤处理的行数。
max_cpu_time	bigint	使用的最大 CPU 时间（以微秒为单位）。在段级别，段的跨所有切片使用的最大 CPU 时间。在查询级别，查询段使用的最大 CPU 时间。
cpu_time	bigint	使用的 CPU 时间（单位为微秒）。在段级别，段的跨所有切片的总 CPU 时间。在查询级别，查询的跨所有切片和段的 CPU 时间合计。
max_blocks_read	bigint	段读取的 1 MB 数据块的最大数量（跨所有切片进行聚合）。在段级别，段的跨所有切片读取的 1 MB 数据块的最大数量。在查询级别，任何查询段读取的 1 MB 数据块的最大数量。
blocks_read	bigint	查询或段读取的 1 MB 数据块的数量。

列名称	数据类型	描述
max_run_time	bigint	段的最大已用时间（以微秒为单位）。在段级别，段的跨所有切片的最大运行时间。在查询级别，任何查询段的最大运行时间。
run_time	bigint	跨所有切片合计的总运行时间。运行时间不包括等待时间。 在段级别，跨所有切片合计的段的运行时间。在查询级别，跨所有切片和段合计的查询的运行时间。由于此值是一个合计，因此运行时间与查询执行时间无关。
max_blocks_to_disk	bigint	用于写入中间结果的最大磁盘空间量（以 MB 数据块为单位）。在段级别，段跨所有切片使用的最大磁盘空间量。在查询级别，任何查询段使用的最大磁盘空间量。
blocks_to_disk	bigint	查询和段用于写入中间结果使用的磁盘空间量（以 MB 数据块为单位）。
step	integer	运行的查询步骤。
max_query_scan_size	bigint	查询扫描的数据的最大大小（以 MB 为单位）。在段级别，段跨所有切片扫描的数据的最大大小。在查询级别，任何查询段扫描的数据的最大大小。
query_scan_size	bigint	查询扫描的数据的大小（以 MB 为单位）。
query_priority	integer	查询的优先级。可能的值为 -1、0、1、2、3 和 4，其中 -1 表示不支持查询优先级。
query_queue_time	bigint	查询排队的时间（以微秒为单位）。
service_class_name	character (64)	服务类的名称。

示例查询

要查找具有较长的 CPU 时间（1000 秒以上）的查询，请运行以下查询。

```
Select query, cpu_time / 1000000 as cpu_seconds
from stl_query_metrics where segment = -1 and cpu_time > 1000000000
order by cpu_time;

query | cpu_seconds
------+------------
25775 |        9540
```

要查找具有返回超过一百万个行的嵌套循环联接的活动查询，请运行以下查询。

```
select query, rows
from stl_query_metrics
where step_type = 15 and rows > 1000000
order by rows;

query | rows
------+-----------
25775 | 2621562702
```

要查找已运行超过 60 秒且使用的 CPU 时间不到 10 秒的活动查询，请运行以下查询。

```
select query, run_time/1000000 as run_time_seconds
from stl_query_metrics
where segment = -1 and run_time > 60000000 and cpu_time < 10000000;

query | run_time_seconds
------+-----------------
25775 |              114
```

STL_QUERYTEXT

捕获 SQL 命令的查询文本。

查询 STL_QUERYTEXT 视图以捕获为以下语句记录的 SQL：

- SELECT、SELECT INTO
- INSERT、UPDATE、DELETE
- COPY
- UNLOAD
- 运行 VACUUM 和 ANALYZE 生成的查询

- CREATE TABLE AS (CTAS)

要查询某个给定时间段中针对这些语句的活动，请联接 STL_QUERYTEXT 和 STL_QUERY 视图。

> **Note**
>
> STL_QUERY 和 STL_QUERYTEXT 视图仅包含有关查询的信息，不包含有关其他实用工具和 DDL 命令的信息。对于有关 Amazon Redshift 运行的所有语句的列表和信息，您还可以查询 STL_DDLTEXT 和 STL_UTILITYTEXT 视图。有关 Amazon Redshift 运行的所有语句的完整列表，您可以查询 SVL_STATEMENTTEXT 视图。

另请参阅 STL_DDLTEXT、STL_UTILITYTEXT 和 SVL_STATEMENTTEXT。

STL_QUERYTEXT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_TEXT 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
xid	bigint	事务 ID。
pid	integer	进程 ID。一般情况下，会话中的所有查询在同一进程中运行，因此，如果您在同一会话中运行一系列查询，则此值通常保持不变。在特定的内部事件之后，Amazon Redshift 可能会重新启动一个活动会话并分配新的 PID。有关更多信息，请参阅 STL_RESTARTED_SESSIONS。您可以使用此列联接到 STL_ERROR 视图。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
sequence	integer	当一个语句包含 200 多个字符时，将为该语句记录额外的行。序列 0 是第一行，1 是第二行，依此类推。

列名称	数据类型	描述
text	character(200)	SQL 文本，以 200 个字符递增。此字段可能包含反斜杠 (\\) 和换行符 (\n) 等特殊字符。

示例查询

您可以使用 PG_BACKEND_PID() 函数检索当前会话的信息。例如，以下查询返回当前会话中完成的查询的查询 ID 和一部分查询文本。

```
select query, substring(text,1,60)
from stl_querytext
where pid = pg_backend_pid()
order by query desc;

 query |                          substring
-------+------------------------------------------------------------
 28262 | select query, substring(text,1,80) from stl_querytext where
 28252 | select query, substring(path,0,80) as path from stl_unload_l
 28248 | copy category from 's3://dw-tickit/manifest/category/1030_ma
 28247 | Count rows in target table
 28245 | unload ('select * from category') to 's3://dw-tickit/manifes
 28240 | select query, substring(text,1,40) from stl_querytext where
(6 rows)
```

重新构造存储的 SQL

要重新构造存储在 STL_QUERYTEXT 的text 列中的 SQL，请运行 SELECT 语句，以从 text 列中的一个或多个部分创建 SQL。在运行重新构造的 SQL 之前，将任何 (\n) 特殊字符替换为新行。以下 SELECT 语句的结果是 query_statement 字段中重新构造的 SQL 的行。

```
SELECT query, LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END)
 WITHIN GROUP (ORDER BY sequence) as query_statement, COUNT(*) as row_count
FROM stl_querytext GROUP BY query ORDER BY query desc;
```

例如，以下查询选择 3 列。查询本身超过 200 个字符，并存储在 STL_QUERYTEXT 中的几个部分内。

```
select
```

```
1 AS a012345678901234567890123456789012345678901234567890123456789012345678901234567890,
2 AS b012345678901234567890123456789012345678901234567890123456789012345678901234567890,
3 AS b012345678901234567890123456789012345678901234
FROM stl_querytext;
```

在此示例中，查询存储在 STL_QUERYTEXT 的 text 列的 2 个部分（行）中。

```
select query, sequence, text
from stl_querytext where query=pg_last_query_id() order by query desc, sequence limit
 10;
```

```
query  | sequence |
                          text

-------+----------
+-----------------------------------------------------------------------------------------------------------------
    45 |        0 | select\n1 AS
 a012345678901234567890123456789012345678901234567890123456789012345678901234567890,\n2 AS
 b012345678901234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b012345678901234567890123456789012345678901234
    45 |        1 | \nFROM stl_querytext;
```

要重新构造存储在 STL_QUERYTEXT 中的 SQL，请运行以下 SQL。

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS text
from stl_querytext where query=pg_last_query_id();
```

要在客户端中使用生成的重新构造的 SQL，需将任何 (\n) 特殊字符替换为新行。

```
                      text

-----------------------------------------------------------------------------------------------------------------
 select\n1 AS a012345678901234567890123456789012345678901234567890123456789012345678901234567890,
\n2 AS b012345678901234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b012345678901234567890123456789012345678901234\nFROM stl_querytext;
```

STL_REPLACEMENTS

显示当无效的 UTF-8 字符由带有 ACCEPTINVCHARS 选项的 COPY 命令替换时所记录的日志。对于每个至少需要一次替换的节点切片上的前 100 行，将为每个行向 STL_REPLACEMENTS 添加一个日志条目。

STL_REPLACEMENTS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_NESTLOOP 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_COPY_REPLACEMENTS。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	发生替换的节点切片的编号。
tbl	integer	表 ID。
starttime	timestamp	COPY 命令的开始时间（采用 UTC 表示）。
session	integer	执行 COPY 命令的会话的会话 ID。
filename	character (256)	COPY 命令的输入文件的完整路径。
line_number	bigint	包含了无效的 UTF-8 字符的输入数据文件中的行号。-1 表示行号不可用，例如，在从列式数据文件中复制时。
colname	character (127)	包含了无效的 UTF-8 字符的首个字段。

列名称	数据类型	描述
raw_line	character (1024)	包含了无效的 UTF-8 字符的原始加载数据。

示例查询

以下示例返回最近的 COPY 操作中的替换。

```
select query, session, filename, line_number, colname
from stl_replacements
where query = pg_last_copy_id();

 query | session |   filename                          | line_number | colname
 ------+---------+-------------------------------------+-------------+--------
    96 |    6314 | s3://mybucket/allusers_pipe.txt   |         251 | city
    96 |    6314 | s3://mybucket/allusers_pipe.txt   |         317 | city
    96 |    6314 | s3://mybucket/allusers_pipe.txt   |         569 | city
    96 |    6314 | s3://mybucket/allusers_pipe.txt   |         623 | city
    96 |    6314 | s3://mybucket/allusers_pipe.txt   |         694 | city
...
```

STL_RESTARTED_SESSIONS

为了在发生特定内部事件后保持连续可用性，Amazon Redshift 可能会重新启动具有新进程 ID (PID) 的活动会话。在 Amazon Redshift 重新启动会话时，STL_RESTARTED_SESSIONS 将记录新 PID 和旧 PID。

有关更多信息，请参阅本部分中后面的示例。

STL_RESTARTED_SESSIONS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_SESSION_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
currenttime	timestamp	事件的时间。
dbname	character (50)	与会话关联的数据库的名称。
newpid	integer	重新启动的会话的进程 ID。
oldpid	integer	原始会话的进程 ID。
username	character (50)	与会话关联的用户的名称。
remotehost	character (45)	远程主机的名称或 IP 地址。
remoteport	character (32)	远程主机的端口号。
parkedtime	timestamp	此信息仅供内部使用。
session_vars	character (2000)	此信息仅供内部使用。

示例查询

以下示例将 STL_RESTARTED_SESSIONS 与 STL_SESSIONS 联接以显示已重新启动的会话的用户名。

```
select process, stl_restarted_sessions.newpid, user_name
from stl_sessions
inner join stl_restarted_sessions on stl_sessions.process =
 stl_restarted_sessions.oldpid
order by process;

...
```

STL_RETURN

包含查询中的返回 步骤的详细信息。返回步骤将在计算节点上完成的查询的结果返回到领导节点。然后，领导节点合并数据并将结果返回至请求客户端。对于在领导节点上完成的查询，返回步骤将结果返回至客户端。

一个查询包含多个区段，而且每个区段包含一个或多个步骤。有关更多信息，请参阅 查询处理。

STL_RETURN 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_RETURN 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。

列名称	数据类型	描述
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
bytes	bigint	该步骤中所有输出行的大小（以字节为单位）。
packets	integer	通过网络发送的数据包的总数。
checksum	bigint	此信息仅供内部使用。

示例查询

以下示例查询显示最近查询中在每个切片上执行的步骤。

```
SELECT query, slice, segment, step, endtime, rows, packets
from stl_return where query = pg_last_query_id();

 query |  slice | segment | step |          endtime           | rows | packets
-------+--------+---------+------+----------------------------+------+---------
     4 |      2 |       3 |    2 | 2013-12-27 01:43:21.469043 |    3 |       0
     4 |      3 |       3 |    2 | 2013-12-27 01:43:21.473321 |    0 |       0
     4 |      0 |       3 |    2 | 2013-12-27 01:43:21.469118 |    2 |       0
     4 |      1 |       3 |    2 | 2013-12-27 01:43:21.474196 |    0 |       0
     4 |      4 |       3 |    2 | 2013-12-27 01:43:21.47704  |    2 |       0
     4 |      5 |       3 |    2 | 2013-12-27 01:43:21.478593 |    0 |       0
     4 |   12811|       4 |    1 | 2013-12-27 01:43:21.480755 |    0 |       0
(7 rows)
```

STL_S3CLIENT

记录传输时间和其他性能指标。

使用 STL_S3CLIENT 表可查找传输 Amazon S3 中的数据所花费的时间。

STL_S3CLIENT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
recordtime	timestamp	记录的记录时间。
pid	integer	进程 ID。会话中的所有查询在同一进程中运行，因此，如果您在同一会话中运行一系列查询，则此值保持不变。
http_method	character (64)	与 Amazon S3 请求对应的 HTTP 方法名称。
桶	character (64)	S3 桶名称。
key	character (256)	与 Amazon S3 对象对应的键。
transfer_size	bigint	传输的字节数。
data_size	bigint	数据的字节数。此值与未压缩的数据的 transfer_size 相同。如果使用了压缩，则这是未压缩的数据的大小。
start_time	bigint	传输开始时的时间（自 2000 年 1 月 1 日起以微秒为单位）。
end_time	bigint	传输结束时的时间（自 2000 年 1 月 1 日起以微秒为单位）。
transfer_time	bigint	传输花费的时间（单位为微秒）。
compression_time	bigint	用来解压缩数据花费的传输时间的一部分（单位为微秒）。
connect_time	bigint	从开始直至连接到远程服务器完成的时间（单位为微秒）。
app_connect_time	bigint	从开始直至与远程主机的 SSL 连接/握手完成的时间（单位为微秒）。

列名称	数据类型	描述
retries	bigint	重试传输的次数。
request_id	char(32)	Amazon S3 HTTP 响应标头中的请求 ID
extended_request_id	char(128)	Amazon S3 HTTP 标头响应中的扩展请求 ID（x-amz-id-2）。
ip_address	char(64)	服务器的 IP 地址（ip V4 或 V6）。
is_partial	integer	值，如果为真 (1) 表示在 COPY 操作期间输入文件被拆分为范围。如果此值为假 (0)，则不会拆分输入文件。
start_offset	bigint	值，如果在 COPY 操作期间拆分输入文件，则表示拆分的偏移值（以字节为单位）。如果文件未拆分，则此值为 0。

示例查询

以下查询返回使用 COPY 命令加载文件花费的时间。

```
select slice, key, transfer_time
from stl_s3client
where query = pg_last_copy_id();
```

结果

```
 slice |   key                          | transfer_time
------+--------------------------------+---------------
     0 | listing10M0003_part_00         |    16626716
     1 | listing10M0001_part_00         |    12894494
     2 | listing10M0002_part_00         |    14320978
     3 | listing10M0000_part_00         |    11293439
  3371 | prefix=listing10M;marker=      |       99395
```

以下示例将 start_time 和 end_time 转换为时间戳。

```
select userid,query,slice,pid,recordtime,start_time,end_time,
'2000-01-01'::timestamp + (start_time/1000000.0)* interval '1 second' as start_ts,
'2000-01-01'::timestamp + (end_time/1000000.0)* interval '1 second' as end_ts
```

```
from stl_s3client where query> -1 limit 5;
```

```
 userid | query | slice |  pid  |         recordtime         |   start_time    |
 end_time     |          start_ts          |           end_ts
--------+-------+-------+-------+----------------------------+-----------------
+-----------------+----------------------------+----------------------------
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.207839 | 616436837154256 |
 616436837207838 | 2019-07-14 16:27:17.154256 | 2019-07-14 16:27:17.207838
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.252521 | 616436837208208 |
 616436837252520 | 2019-07-14 16:27:17.208208 | 2019-07-14 16:27:17.25252
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.284376 | 616436837208460 |
 616436837284374 | 2019-07-14 16:27:17.20846  | 2019-07-14 16:27:17.284374
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.285307 | 616436837208980 |
 616436837285306 | 2019-07-14 16:27:17.20898  | 2019-07-14 16:27:17.285306
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.353853 | 616436837302216 |
 616436837353851 | 2019-07-14 16:27:17.302216 | 2019-07-14 16:27:17.353851
```

STL_S3CLIENT_ERROR

记录从 Amazon S3 加载文件时切片遇到的错误。

使用 STL_S3CLIENT_ERROR 来查找在从 Amazon S3 传输数据（作为 COPY 命令的一部分）时遇到的错误的详细信息。

STL_S3CLIENT_ERROR 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。查询 ID -1 供内部使用。
sliceid	integer	标识运行查询所在切片的标识符。

列名称	数据类型	描述
recordtime	timestamp	记录的记录时间。
pid	integer	进程 ID。会话中的所有查询在同一进程中运行，因此，如果您在同一会话中运行一系列查询，则此值保持不变。
http_method	character (64)	与 Amazon S3 请求对应的 HTTP 方法名称。
桶	character (64)	Amazon S3 桶名称。
key	character (256)	与 Amazon S3 对象对应的键。
error	character (1024)	错误消息。
is_partial	integer	值，如果为真 (1)，则表示在 COPY 操作期间将输入文件拆分为范围。如果此值为假 (0)，则不会拆分输入文件。
start_offset	bigint	值，如果在 COPY 操作期间拆分输入文件，则表示拆分的偏移值（以字节为单位）。如果文件未拆分，则此值为 0。

使用说明

如果您看到多个包含“Connection timed out”的错误，您可能遇到了联网问题。如果您使用的是增强型 VPC 路由，请确认您在集群的 VPC 与数据资源之间具有有效的网络路径。有关更多信息，请参阅 Amazon Redshift 增强型 VPC 路由。

示例查询

以下查询返回当前会话期间完成的 COPY 命令中的错误。

```
select query, sliceid, substring(key from 1 for 20) as file,
substring(error from 1 for 35) as error
from stl_s3client_error
where pid = pg_backend_pid()
order by query desc;
```

结果

```
 query  | sliceid |     file           |              error

--------+---------+--------------------+-----------------------------------
 362228 |      12 | part.tbl.25.159.gz | transfer closed with 1947655 bytes
 362228 |      24 | part.tbl.15.577.gz | transfer closed with 1881910 bytes
 362228 |       7 | part.tbl.22.600.gz | transfer closed with 700143 bytes r
 362228 |      22 | part.tbl.3.34.gz   | transfer closed with 2334528 bytes
 362228 |      11 | part.tbl.30.274.gz | transfer closed with 699031 bytes r
 362228 |      30 | part.tbl.5.509.gz  | Unknown SSL protocol error in conne
 361999 |      10 | part.tbl.23.305.gz | transfer closed with 698959 bytes r
 361999 |      19 | part.tbl.26.582.gz | transfer closed with 1881458 bytes
 361999 |       4 | part.tbl.15.629.gz | transfer closed with 2275907 bytes
 361999 |      20 | part.tbl.6.456.gz  | transfer closed with 692162 bytes r
(10 rows)
```

STL_SAVE

包含查询中的保存 步骤的详细信息。保存步骤将输入流保存到一个临时表。临时表是查询执行期间存储中间结果的临时表。

一个查询包含多个区段，而且每个区段包含一个或多个步骤。有关更多信息，请参阅 查询处理。

STL_SAVE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_SAVE 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。

列名称	数据类型	描述
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
bytes	bigint	该步骤中所有输出行的大小（以字节为单位）。
tbl	integer	具体化临时表的 ID。
is_diskbased	character(1)	此查询步骤是否已作为基于磁盘的操作执行：true (t) 或 false (f)。
workmem	bigint	分配给步骤的工作内存的字节数。

示例查询

以下查询显示最近查询中在每个切片上执行的保存步骤。

```
select query, slice, segment, step, tasknum, rows,  tbl
from stl_save where query = pg_last_query_id();

 query | slice | segment | step | tasknum | rows | tbl
-------+-------+---------+------+---------+------+-----
 52236 |     3 |       0 |    2 |      21 |    0 | 239
```

```
 52236 |     2 |       0 |    2 |      20 |    0 | 239
 52236 |     2 |       2 |    2 |      20 |    0 | 239
 52236 |     3 |       2 |    2 |      21 |    0 | 239
 52236 |     1 |       0 |    2 |      21 |    0 | 239
 52236 |     0 |       0 |    2 |      20 |    0 | 239
 52236 |     0 |       2 |    2 |      20 |    0 | 239
 52236 |     1 |       2 |    2 |      21 |    0 | 239
(8 rows)
```

STL_SCAN

分析查询的表扫描步骤。此表中行的步骤数始终为 0，因为扫描是分段中的第一步。

STL_SCAN 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_SCAN 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。

列名称	数据类型	描述
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
bytes	bigint	该步骤中所有输出行的大小（以字节为单位）。
fetches	bigint	此信息仅供内部使用。
type	integer	扫描类型的 ID。有关有效值的列表，请参阅下表。
tbl	integer	表 ID。
is_rrscan	character(1)	如果为 true (t)，则表示对步骤使用了限制范围的扫描。
is_delayed_scan	character(1)	此信息仅供内部使用。
rows_pre_filter	bigint	对于永久表的扫描，这是在筛选标记为删除的行（虚影行）之前和应用用户定义的查询筛选条件之前发出的行的总数。
rows_pre_user_filter	bigint	对于永久表的扫描，这是在筛选标记为删除的行（虚影行）之后但在应用用户定义的查询筛选条件之前处理的行的数量。
perm_table_name	character(136)	对于永久表的扫描，这是扫描的表的名称。
is_rlf_scan	character(1)	如果为 true (t)，则表示对步骤使用了行级别筛选。
is_rlf_scan_reason	integer	此信息仅供内部使用。
num_em_blocks	integer	此信息仅供内部使用。

列名称	数据类型	描述
checksum	bigint	此信息仅供内部使用。
runtime_filtering	character(1)	如果为 true (t)，则表示应用了运行时筛选器。
scan_region	整数	此信息仅供内部使用。
num_sortkey_as_predicate	整数	此信息仅供内部使用。
row_fetcher_state	整数	此信息仅供内部使用。
consumed_scan_ranges	bigint	此信息仅供内部使用。
work_stealing_reason	bigint	此信息仅供内部使用。
is_vectorized_scan	character(1)	此信息仅供内部使用。
is_vectorized_scan_reason	整数	此信息仅供内部使用。
row_fetcher_reason	bigint	此信息仅供内部使用。
topology_signature	bigint	此信息仅供内部使用。

列名称	数据类型	描述
use_tpm_partition	character(1)	此信息仅供内部使用。
is_rrscan_expr	character(1)	此信息仅供内部使用。
scanned_mega_value	character(1)	此信息仅供内部使用。此信息显示给定的扫描步骤是否扫描了大值。大值将存储在多个块中。原定设置情况下，块大小为 1 MB，大值大于原定设置中的 1 MB。

扫描类型

类型 ID	描述
1	网络中的数据。
2	压缩共享内存中的永久用户表。
3	临时行式表。
21	从 Amazon S3 加载文件。
22	从 Amazon DynamoDB 中加载表。
23	从远程 SSH 连接中加载数据。
24	从远程集群（已排序区域）中加载数据。这可用于调整大小。
25	从远程集群（未排序区域）中加载数据。这可用于调整大小。
28	在多个表上使用 UNION ALL 从时间序列视图读取数据。
29	从 Amazon S3 外部表中读取数据。
30	读取 Amazon S3 外部表的分区信息。
33	从远程 Postgres 表中读取数据。

类型 ID	描述
36	从远程 MySQL 表中读取数据。
37	从远程 Kinesis 流中读取数据。

使用说明

理想情况下，rows 应相对接近于 rows_pre_filter。rows 与 rows_pre_filter 之间的一个很大的区别是，它意味着执行引擎将要扫描稍后将被丢弃的行（这样做的效率较低）。rows_pre_filter 与 rows_pre_user_filter 之间的区别是扫描中虚影行的数量。运行 VACUUM 可移除标记为删除的行。rows 与 rows_pre_user_filter 之间的区别是查询筛选的行的数量。如果用户筛选器丢弃了大量行，则查看您选择的排序列，或者，如果这是由于大量未排序的区域造成的，请运行 vacuum。

示例查询

以下示例显示 rows_pre_filter 大于 rows_pre_user_filter，因为该表已删除尚未真空化的行（虚影行）。

```
SELECT query, slice, segment,step,rows, rows_pre_filter, rows_pre_user_filter
from stl_scan where query = pg_last_query_id();

 query |  slice | segment | step | rows  | rows_pre_filter | rows_pre_user_filter
-------+--------+---------+------+-------+-----------------+----------------------
 42915 |      0 |       0 |    0 | 43159 |           86318 |                43159
 42915 |      0 |       1 |    0 |     1 |               0 |                    0
 42915 |      1 |       0 |    0 | 43091 |           86182 |                43091
 42915 |      1 |       1 |    0 |     1 |               0 |                    0
 42915 |      2 |       0 |    0 | 42778 |           85556 |                42778
 42915 |      2 |       1 |    0 |     1 |               0 |                    0
 42915 |      3 |       0 |    0 | 43428 |           86856 |                43428
 42915 |      3 |       1 |    0 |     1 |               0 |                    0
 42915 |  10000 |       2 |    0 |     4 |               0 |                    0
(9 rows)
```

STL_SCHEMA_QUOTA_VIOLATIONS

记录超出 schema 配额时的匹配项、时间戳、XID 和其他有用信息。

STL_SCHEMA_QUOTA_VIOLATIONS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_SCHEMA_QUOTA_VIOLATIONS 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
ownerid	integer	schema 拥有者的 ID。
xid	bigint	与语句关联的事务 ID。
pid	integer	与语句关联的进程 ID。
userid	integer	生成该条目的用户的 ID。
schema_id	integer	命名空间或 schema ID。
schema_name	character (128)	命名空间或 schema 名称。
配额	integer	schema 可以使用的磁盘空间量（以 MB 为单位）。
disk_usage	integer	schema 当前使用的磁盘空间（以 MB 为单位）。
disk_usage_pct	double precision	schema 当前使用的磁盘空间相对于为其配置的配额的百分比。
timestamp	不带时区的时间戳	违规情况发生的时间。

示例查询

以下查询显示配额冲突的结果：

```
SELECT userid, TRIM(SCHEMA_NAME) "schema_name", quota, disk_usage, disk_usage_pct,
 timestamp FROM
```

```
stl_schema_quota_violations WHERE SCHEMA_NAME = 'sales_schema' ORDER BY timestamp DESC;
```

此查询返回指定 schema 的以下示例输出：

```
userid | schema_name  | quota | disk_usage | disk_usage_pct |timestamp
-------+--------------+-------+------------+----------------
+----------------------------
104    | sales_schema | 2048  | 2798       |  136.62        | 2020-04-20
 20:09:25.494723
(1 row)
```

STL_SESSIONS

返回有关用户会话历史记录的信息。

STL_SESSIONS 与 STV_SESSIONS 的不同之处在于，STL_SESSIONS 包含会话历史记录，而 STV_SESSIONS 包含当前活动会话。

STL_SESSIONS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_SESSION_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
starttime	timestamp	会话启动的时间（采用 UTC 表示）。
endtime	timestamp	会话结束的时间（采用 UTC 表示）。
process	integer	会话的进程 ID。
user_name	character(50)	与会话关联的用户名称。
db_name	character(50)	与会话关联的数据库的名称。

列名称	数据类型	描述
timeout_sec	int	超时前会话保持非活动状态或空闲状态的最长时间（秒）。0 表示未设置超时。
timed_out	int	指示会话是否超时的值：1（如果会话超时），否则为 0。

示例查询

要查看 TICKIT 数据库的会话历史记录，请键入以下查询：

```
select starttime, process, user_name, timeout_sec, timed_out
from stl_sessions
where db_name='tickit' order by starttime;
```

此查询返回以下示例输出：

```
    starttime              | process |  user_name              | timeout_sec | timed_out
---------------------------+---------+-------------------------+-------------
+-------------
2008-09-15 09:54:06.746705 |   32358 | dwuser                  | 120         | 1
2008-09-15 09:56:34.30275  |   32744 | dwuser                  | 60          | 1
2008-09-15 11:20:34.694837 |   14906 | dwuser                  | 0           | 0
2008-09-15 11:22:16.749818 |   15148 | dwuser                  | 0           | 0
2008-09-15 14:32:44.66112  |   14031 | dwuser                  | 0           | 0
2008-09-15 14:56:30.22161  |   18380 | dwuser                  | 0           | 0
2008-09-15 15:28:32.509354 |   24344 | dwuser                  | 0           | 0
2008-09-15 16:01:00.557326 |   30153 | dwuser                  | 120         | 1
2008-09-15 17:28:21.419858 |   12805 | dwuser                  | 0           | 0
2008-09-15 20:58:37.601937 |   14951 | dwuser                  | 60          | 1
2008-09-16 11:12:30.960564 |   27437 | dwuser                  | 60          | 1
2008-09-16 14:11:37.639092 |   23790 | dwuser                  | 3600        | 1
2008-09-16 15:13:46.02195  |    1355 | dwuser                  | 120         | 1
2008-09-16 15:22:36.515106 |    2878 | dwuser                  | 120         | 1
2008-09-16 15:44:39.194579 |    6470 | dwuser                  | 120         | 1
2008-09-16 16:50:27.02138  |   17254 | dwuser                  | 120         | 1
2008-09-17 12:05:02.157208 |    8439 | dwuser                  | 3600        | 0
(17 rows)
```

STL_SORT

显示查询的排序执行步骤，如使用 ORDER BY 处理的步骤。

STL_SORT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_SORT 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。

列名称	数据类型	描述
bytes	bigint	该步骤中所有输出行的大小（以字节为单位）。
tbl	integer	表 ID。
is_diskbased	character(1)	如果为 true (t)，则查询是作为基于磁盘的操作执行的。如果为 false (f)，则查询是在内存中执行。
workmem	bigint	已分配给步骤的工作内存中的字节总数。
checksum	bigint	此信息仅供内部使用。

示例查询

以下示例返回切片 0 和分段 1 的排序结果。

```
select query, bytes, tbl, is_diskbased, workmem
from stl_sort
where slice=0 and segment=1;
```

```
 query |  bytes  | tbl | is_diskbased |  workmem
-------+---------+-----+--------------+-----------
   567 | 3126968 | 241 | f            | 383385600
   604 |    5292 | 242 | f            | 383385600
   675 |  104776 | 251 | f            | 383385600
   525 | 3126968 | 251 | f            | 383385600
   585 |    5068 | 241 | f            | 383385600
   630 |  204808 | 266 | f            | 383385600
   704 |       0 | 242 | f            |         0
   669 | 4606416 | 241 | f            | 383385600
   696 |  104776 | 241 | f            | 383385600
   651 | 4606416 | 254 | f            | 383385600
   632 |       0 | 256 | f            |         0
   599 |     396 | 241 | f            | 383385600
 86397 |       0 | 242 | f            |         0
   621 |    5292 | 241 | f            | 383385600
 86325 |       0 | 242 | f            |         0
   572 |    5068 | 242 | f            | 383385600
   645 |  204808 | 241 | f            | 383385600
   590 |     396 | 242 | f            | 383385600
```

```
(18 rows)
```

STL_SSHCLIENT_ERROR

记录由 SSH 客户端发现的所有错误。

STL_SSHCLIENT_ERROR 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
recordtime	timestamp	记录错误的时间。
pid	integer	记录错误的进程。
ssh_username	character (1024)	SSH 用户名称。
endpoint	character (1024)	SSH 终端节点。
命令	character (4096)	完整的 SSH 命令。
error	character (1024)	错误消息。

STL_STREAM_SEGS

列出流与并发分段之间的关系。

在这种情况下，流是 Amazon Redshift 流。此系统视图不适用于串流摄取。

STL_STREAM_SEGS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_STREAM_SEGS 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
stream	integer	查询的并发分段集。
segment	integer	标识查询区段的数字。

示例查询

要查看最近查询的流与并发分段之间的关系，请键入以下查询：

```
select *
from stl_stream_segs
where query = pg_last_query_id();

 query | stream | segment
-------+--------+---------
    10 |      1 |       2
    10 |      0 |       0
    10 |      2 |       4
    10 |      1 |       3
    10 |      0 |       1
(5 rows)
```

STL_TR_CONFLICT

显示用于确定并解决与数据库表的事务冲突的信息。

当两个或更多用户正在查询和修改表中的数据行（以使其事务无法序列化）时，会发生事务冲突。如果事务运行一个将会破坏可序列性的语句，将会被停止并回滚。每次发生事务冲突时，Amazon Redshift 都会在包含有关被取消事务详细信息的 STL_TR_CONFLICT 系统表中写入数据行。有关更多信息，请参阅 可序列化的隔离。

STL_TR_CONFLICT 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_TRANSACTION_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
xact_id	bigint	已回滚事务的事务 ID。
process_id	bigint	与回滚的事务关联的进程。
xact_start_ts	时间戳	事务开始的时间戳 (UTC)。
abort_time	时间戳	事务停止的时间戳 (UTC)。
table_id	bigint	发生冲突的表的表 ID。

示例查询

要返回有关涉及某一特定表的冲突的信息，请运行指定了表 ID 的查询：

```
select * from stl_tr_conflict where table_id=100234
order by xact_start_ts;

xact_id|process_|      xact_start_ts       |        abort_time        |table_
       |id      |                          |                          |id
-------+--------+--------------------------+--------------------------+------
  1876 |  8551  |2010-03-30 09:19:15.852326|2010-03-30 09:20:17.582499|100234
  1928 | 15034  |2010-03-30 13:20:00.636045|2010-03-30 13:20:47.766817|100234
  1991 | 23753  |2010-04-01 13:05:01.220059|2010-04-01 13:06:06.94098 |100234
```

```
  2002 | 23679  |2010-04-01 13:17:05.173473|2010-04-01 13:18:27.898655|100234
(4 rows)
```

您可以从针对可串行性冲突的错误消息的 DETAIL 部分中获取表 ID（错误 1023）。

STL_UNDONE

显示有关已撤消的事务的信息。

STL_UNDONE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_TRANSACTION_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
xact_id	bigint	已撤消事务的 ID。
xact_id_undone	bigint	已撤消的事务的 ID。
undo_start_ts	timestamp	已撤消事务的开始时间。
undo_end_ts	timestamp	已撤消事务的结束时间。
table_id	bigint	受已撤消事务影响的表的 ID。

示例查询

要查看所有已撤消事务的简明日志，请键入以下命令：

```
select xact_id, xact_id_undone, table_id from stl_undone;
```

此命令返回以下示例输出：

```
 xact_id | xact_id_undone | table_id
---------+----------------+----------
1344 |           1344 |   100192
1326 |           1326 |   100192
1551 |           1551 |   100192
(3 rows)
```

STL_UNIQUE

分析在 SELECT 列表中使用 DISTINCT 函数时或者在 UNION 或 INTERSECT 查询中删除重复项时发生的执行步骤。

STL_UNIQUE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_UNIQUE 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。

列名称	数据类型	描述
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
type	character(6)	步骤的类型。有效值为： • HASHED。表示步骤使用了已分组但未排序的聚合。 • PLAIN。表示步骤使用了未分组的标量聚合。 • SORTED。表示步骤使用了已分组且已排序的聚合。
is_diskbased	character(1)	如果为 true (t)，则查询是作为基于磁盘的操作执行的。如果为 false (f)，则查询是在内存中执行。
slots	integer	哈希桶的总数。
workmem	bigint	已分配给步骤的工作内存中的字节总数。
max_buffers_used	bigint	转到磁盘之前在哈希表中使用的缓冲区的最大数量。
resizes	integer	此信息仅供内部使用。
occupied	整数	此信息仅供内部使用。
flushable	integer	此信息仅供内部使用。
used_unique_prefetching	character(1)	此信息仅供内部使用。
bytes	biginit	该步骤中所有输出行的字节数。

示例查询

假设您运行以下查询：

```
select distinct eventname
from event order by 1;
```

假定上个查询的 ID 是 6313，以下示例显示由分段 0 和 1 中每个切片的唯一步骤生成的行的数量。

```
select query, slice, segment, step, datediff(msec, starttime, endtime) as msec,
 tasknum, rows
from stl_unique where query = 6313
order by query desc, slice, segment, step;
```

```
 query | slice | segment | step | msec | tasknum | rows
-------+-------+---------+------+------+---------+------
  6313 |     0 |       0 |    2 |    0 |      22 |  550
  6313 |     0 |       1 |    1 |  256 |      20 |  145
  6313 |     1 |       0 |    2 |    1 |      23 |  540
  6313 |     1 |       1 |    1 |   42 |      21 |  127
  6313 |     2 |       0 |    2 |    1 |      22 |  540
  6313 |     2 |       1 |    1 |  255 |      20 |  158
  6313 |     3 |       0 |    2 |    1 |      23 |  542
  6313 |     3 |       1 |    1 |   38 |      21 |  146
(8 rows)
```

STL_UNLOAD_LOG

记录卸载操作的详细信息。

STL_UNLOAD_LOG 为由 UNLOAD 语句创建的每个文件记录一行。例如，如果 UNLOAD 创建 12 个文件，则 STL_UNLOAD_LOG 将包含 12 个对应的行。

STL_UNLOAD_LOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_UNLOAD_LOG 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图

SYS_UNLOAD_HISTORY 和 SYS_UNLOAD_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。
slice	integer	标识运行查询所在切片的标识符。
pid	integer	与查询语句关联的进程 ID。
path	character(1280)	文件的完整 Amazon S3 对象路径。
start_time	timestamp	事务的开始时间。
end_time	timestamp	事务的结束时间。
line_count	bigint	已卸载到文件的行的数量。
transfer_size	bigint	传输的字节数。
file_format	character(10)	卸载的文件格式。

示例查询

要获得已由 UNLOAD 命令写入到 Amazon S3 的文件的列表，您可以在 UNLOAD 完成后调用 Amazon S3 列表操作。您还可以查询 STL_UNLOAD_LOG。

以下查询返回上次完成的查询的 UNLOAD 创建的文件的路径名：

```
select query, substring(path,0,40) as path
from stl_unload_log
where query = pg_last_query_id()
order by path;
```

此命令返回以下示例输出：

```
 query |             path
-------+---------------------------------------
  2320 | s3://my-bucket/venue0000_part_00
  2320 | s3://my-bucket/venue0001_part_00
  2320 | s3://my-bucket/venue0002_part_00
  2320 | s3://my-bucket/venue0003_part_00
(4 rows)
```

STL_USAGE_CONTROL

STL_USAGE_CONTROL 视图包含在达到使用限制时记录的信息。有关使用限制的更多信息，请参阅《Amazon Redshift 管理指南》中的管理使用限制。

STL_USAGE_CONTROL 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
eventtime	timestamp	查询超出使用限制的时间 (UTC)。
query	integer	查询标识符。您可以使用此 ID 联接各种其他系统表和视图。
xid	bigint	事务标识符。
pid	integer	与查询关联的进程标识符。
usage_limit_id	character(40)	Amazon Redshift 生成的通用唯一标识符 (UUID)，例如 `25d9297e-3e7b-41c8-9f4d-c4b6eb731c09` 。
feature_type	character(30)	已超出其使用限制的功能。可能的值包括 `CONCURRENCY_SCALING` 和 `SPECTRUM`。

示例查询

以下 SQL 示例返回达到使用限制时记录的部分信息。

```
select query, pid, eventtime, feature_type
```

```
from stl_usage_control
order by eventtime desc
limit 5;
```

STL_USERLOG

记录数据库用户的以下更改的详细信息。

- 创建用户
- 删除用户
- 更改用户（重命名）
- 更改用户（更改属性）

STL_USERLOG 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_USERLOG 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	受更改影响的用户的 ID。
username	character(50)	受更改影响的用户的用户名称。
oldusername	character(50)	对于重命名操作，这是原始用户名称。对于任何其他操作，此字段为空。
操作	character(10)	发生的操作。有效值： • 更改 • 创建 • Drop • 重命名
usecreatedb	integer	如果为 true (1)，则表示用户具有创建数据库的权限。

列名称	数据类型	描述
usesuper	integer	如果为 true (1)，则表示用户为超级用户。
usecatupd	integer	如果为 true (1)，则表示用户可更新系统目录。
valuntil	timestamp	密码到期日期。
pid	integer	进程 ID。
xid	bigint	事务 ID。
recordtime	timestamp	查询开始的时间（采用 UTC 表示）。

示例查询

以下示例执行四种用户操作，然后查询 STL_USERLOG 视图。

```
create user userlog1 password 'Userlog1';
alter user userlog1 createdb createuser;
alter user userlog1 rename  to userlog2;
drop user userlog2;

select userid, username, oldusername, action, usecreatedb, usesuper from stl_userlog
 order by recordtime desc;
```

```
 userid |  username | oldusername |  action | usecreatedb | usesuper
--------+-----------+-------------+---------+-------------+----------
    108 | userlog2  |             | drop    |           1 |   1
    108 | userlog2  | userlog1    | rename  |           1 |   1
    108 | userlog1  |             | alter   |           1 |   1
    108 | userlog1  |             | create  |           0 |   0
 (4 rows)
```

STL_UTILITYTEXT

捕获在数据库上运行的非 SELECT SQL 命令的文本。

查询 STL_UTILITYTEXT 视图以捕获在系统上运行的 SQL 语句的下列子集：

- ABORT、BEGIN、COMMIT、END、ROLLBACK
- ANALYZE
- CALL
- CANCEL
- COMMENT
- CREATE、ALTER、DROP DATABASE
- CREATE、ALTER、DROP USER
- EXPLAIN
- GRANT、REVOKE
- LOCK
- RESET
- SET
- SHOW
- TRUNCATE

另请参阅 STL_DDLTEXT、STL_QUERYTEXT 和 SVL_STATEMENTTEXT。

使用 STARTTIME 和 ENDTIME 列了解在某个给定时间段内记录了哪些语句。SQL 文本的长数据块已分为 200 个字符长的行；SEQUENCE 列标识了属于一个语句的文本片段。

STL_UTILITYTEXT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
xid	bigint	事务 ID。

列名称	数据类型	描述
pid	integer	与查询语句关联的进程 ID。
label	character(320)	用于运行查询的文件的名称或使用 SET QUERY_GROUP 命令定义的标签。如果查询并非基于文件或未设置 QUERY_GROUP 参数，则此字段为空。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
sequence	integer	当一个语句包含 200 多个字符时，将为该语句记录额外的行。序列 0 是第一行，1 是第二行，依此类推。
text	character(200)	SQL 文本，以 200 个字符递增。此字段可能包含反斜杠 (\\) 和换行符 (\n) 等特殊字符。

示例查询

以下查询返回 2012 年 1 月 26 日运行的“utility”命令的文本。在这种情况下，运行的是一些 SET 命令和一个 SHOW ALL 命令：

```
select starttime, sequence, rtrim(text)
from stl_utilitytext
where starttime like '2012-01-26%'
order by starttime, sequence;

starttime          | sequence |              rtrim
---------------------------+-----+---------------------------------
2012-01-26 13:05:52.529235 |   0 | show all;
2012-01-26 13:20:31.660255 |   0 | SET query_group to ''
2012-01-26 13:20:54.956131 |   0 | SET query_group to 'soldunsold.sql'
...
```

重新构造存储的 SQL

要重新构造存储在 STL_UTILITYTEXT 的 text 列中的 SQL，请运行 SELECT 语句，以从 text 列中的一个或多个部分创建 SQL。在运行重新构造的 SQL 之前，将任何 (\n) 特殊字符替换为新行。以下 SELECT 语句的结果是 query_statement 字段中重新构造的 SQL 的行。

```
SELECT LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END) WITHIN
 GROUP (ORDER BY sequence) as query_statement
FROM stl_utilitytext GROUP BY xid order by xid;
```

例如，以下查询将 query_group 设置为零的字符串。查询本身超过 200 个字符，并存储在 STL_UTILITYTEXT 中的几个部分内。

```
set query_group to
 '00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
             000000';
```

在此示例中，查询存储在 STL_UTILITYTEXT 的 text 列的 2 个部分（行）中。

```
select query, sequence, text
from stl_utilitytext where query=pg_last_query_id() order by query desc, sequence limit
 10;
```

```
         starttime          | sequence |
                                                text

----------------------------+----------
+-----------------------------------------------------------------------------------------------
 2019-07-23 22:55:34.926198 |        0 | set query_group to
 '00000000000000000000000000000000000000000000000000000000000000000000000000000000000\n00000000000
 2019-07-23 22:55:34.926198 |        1 |                   000000';
```

要重新构造存储在 STL_UTILITYTEXT 中的 SQL，请运行以下 SQL。

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS query_statement
from stl_utilitytext where query=pg_last_query_id();
```

要在客户端中使用生成的重新构造的 SQL，需将任何 (\n) 特殊字符替换为新行。

```
                                                query_statement


-------------------------------------------------------------------------------------------------------------
 set query_group to
 '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000\n0000000000000
                  000000';
```

STL_VACUUM

显示已真空化的表的行和数据块统计数据。

该视图显示了特定于每个 vacuum 操作开始和完成时的信息，并演示了运行此操作的好处。有关运行该命令的要求的信息，请参阅 VACUUM 命令描述。

STL_VACUUM 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_VACUUM_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户的 ID。
xid	bigint	VACUUM 语句的事务 ID。您可以将此表联接到 STL_QUERY 视图以查看为某一给定 VACUUM 事务运行的各个 SQL 语句。如果您将整个数据库真空化，则将在单独的事务中将每个表真空化。
table_id	integer	表 ID。
status	character(30)	每个表的 VACUUM 操作的状态。可能的值包括： • **Started**

列名称	数据类型	描述
		• **Started Delete Only** • **Started Delete Only (Sorted >= nn%)** 只为 VACUUM FULL 启动了删除阶段。表的排序程度已达到或超出排序阈值，因此跳过了排序阶段。 • **Started Sort Only** • **Started Ranged Partition** • **Started Reindex** • **Finished** 表完成操作的时间。要了解某一特定表执行 vacuum 操作的时长，可以用特定事务 ID 和表 ID 的“Finished”时间减去“Started”时间。 • **Skipped** 表已完全有序且没有带删除标记的行，因此跳过该表。 • **Skipped (delete only)** 指定了 DELETE ONLY，但没有带删除标记的行，因此跳过该表。 • **Skipped (sort only)** 指定了 SORT ONLY，但表已完全有序，因此跳过该表。 • **Skipped (sort only, sorted>=xx%)** 指定了 SORT ONLY，但表的排序程度已达到或超出排序阈值，因此跳过该表。 • **Skipped (0 rows)** 因为表是空的，因此已跳过表。 • **VacuumBG** 在后台执行了自动 vacuum 操作。自动执行这些操作时，此状态先于其他状态。例如，自动执行的

列名称	数据类型	描述
		delete only vacuum 操作将具有状态为 [VacuumBG] Started Delete Only 的起始行。 有关 VACUUM 排序阈值设置的更多信息，请参阅 VACUUM。
rows	bigint	表中的行加上仍存储在磁盘上的所有已删除的行（正在等待执行 vacuum 操作）的实际数量。此列显示带有 **Started** 状态的行开始 vacuum 操作之前的计数以及带有 **Finished** 状态的行完成 vacuum 操作之后的计数。
sortedrows	integer	表中已排序的行的数量。此列显示在“状态”列中具有 **Started** 状态的行开始 vacuum 操作之前的计数，以及在“状态”列中具有 **Finished** 状态的行完成 vacuum 操作之后的计数。
blocks	integer	用于存储 vacuum 操作之前的表数据（带有 **Started** 状态的行）和 vacuum 操作之后的表数据（**Finished** 列）的数据块的总数。每个数据块使用 1 MB。
max_merge_partitions	integer	此列用于性能分析并表示 vacuum 可在每个合并阶段迭代中为表处理的分区的最大数量。（Vacuum 将未排序的区域分为一个或多个已排序的分区。根据表中的列数和当前的 Amazon Redshift 配置，合并阶段可在一个合并迭代中处理最大数量的分区。如果已排序分区的数量超出了合并分区的最大数量，合并阶段仍然有效，但将需要更多合并迭代。）
eventtime	timestamp	当 vacuum 操作开始或完成时。
reclaimable_rows	bigint	当前 cutoff_xid 的可回收行数。此列显示了 Redshift 在对状态为 **Started** 的行开始 vacuum 前的估计可回收行数，以及在对状态为 **Finished** 的行完成 vacuum 后剩余的可回收行的实际数量。

列名称	数据类型	描述
reclaimable_space_mb	bigint	当前 cutoff_xid 的可回收空间（以 MB 为单位）。此列显示了 Redshift 在对状态为 **Started** 的行开始 vacuum 前的估计可回收空间量，以及在对状态为 **Finished** 的行完成 vacuum 后剩余的可回收空间的实际量。
cutoff_xid	bigint	VACUUM 操作的截止事务 ID。截止后的任何事务均不包含在 VACUUM 操作中。
is_recluster	整数	如果为 1 (true)，则 VACUUM 操作执行重新聚类算法，如果为 0 (false)，则不执行此算法。

示例查询

以下查询报告表 108313 的 vacuum 统计数据。此表在一系列插入和删除后已执行 vacuum 操作。

```
select xid, table_id, status, rows, sortedrows, blocks, eventtime,
                reclaimable_rows, reclaimable_space_mb
from stl_vacuum where table_id=108313 order by eventtime;

 xid   | table_id | status                  | rows | sortedrows | blocks |  eventtime
         | reclaimable_rows | reclaimable_space_mb
-------+----------+-------------------------+------+------------+--------
+----------------------+------------------+----------------------
 14294 |   108313 | Started                 | 1950 |        408 |     28 |  2016-05-19
 17:36:01 |              984 |                   17
 14294 |   108313 | Finished                |  966 |        966 |     11 |  2016-05-19
 18:26:13 |                0 |                    0
 15126 |   108313 | Skipped(sorted>=95%)    |  966 |        966 |     11 |  2016-05-19
 18:26:38 |                0 |                    0
```

在 VACUUM 启动时，表包含了存储在 28 个 1MB 数据块中的 1,950 行。Amazon Redshift 估计，它可以通过 vacuum 操作回收 984 行或 17 个数据块的磁盘空间。

在状态为“完成”的行中，ROWS 列显示值 966，而 BLOCKS 列值为 11，低于 28。vacuum 回收了估计的磁盘空间量，在 vacuum 操作完成后，没有剩余可回收的行或空间。

在排序阶段（事务 15126），vacuum 无法跳过此表，因为行是按排序键顺序插入的。

以下示例显示在大型 INSERT 操作之后，针对 SALES 表（在此示例中为表 110116）的 SORT ONLY vacuum 操作的统计数据：

```
vacuum sort only sales;

select xid, table_id, status, rows, sortedrows, blocks, eventtime
from stl_vacuum order by xid, table_id, eventtime;

xid |table_id|     status      | rows  |sortedrows|blocks|     eventtime
----+--------+-----------------+-------+----------+------+--------------------
...
2925| 110116 |Started Sort Only|1379648|   172456 |  132 | 2011-02-24 16:25:21...
2925| 110116 |Finished         |1379648|  1379648 |  132 | 2011-02-24 16:26:28...
```

STL_WINDOW

分析执行窗口函数的查询步骤。

STL_WINDOW 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> STL_WINDOW 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
slice	integer	标识运行查询所在切片的标识符。
segment	integer	标识查询区段的数字。

列名称	数据类型	描述
step	integer	运行的查询步骤。
starttime	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
endtime	时间戳	查询完成的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。
tasknum	整数	分配用于运行步骤的查询任务进程的数量。
rows	bigint	处理的总行数。
is_diskbased	character(1)	如果为 true (t)，则查询是作为基于磁盘的操作执行的。如果为 false (f)，则查询是在内存中执行。
workmem	bigint	已分配给步骤的工作内存中的字节总数。

示例查询

以下示例返回切片 0 和分段 3 的窗口函数结果。

```
select query, tasknum, rows, is_diskbased, workmem
from stl_window
where slice=0 and segment=3;
```

```
 query | tasknum | rows | is_diskbased | workmem
-------+---------+------+--------------+----------
 86326 |      36 | 1857 | f            | 95256616
   705 |      15 | 1857 | f            | 95256616
 86399 |      27 | 1857 | f            | 95256616
   649 |      10 |    0 | f            | 95256616
(4 rows)
```

STL_WLM_ERROR

随时记录发生的任何与 WLM 相关的错误。

STL_WLM_ERROR 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
recordtime	timestamp	错误发生的时间。
pid	integer	生成错误的进程的 ID。
error_string	character(256)	错误描述。

STL_WLM_RULE_ACTION

记录有关从 WLM 查询监控规则生成的与用户定义的队列关联的操作的详细信息。有关更多信息，请参阅 WLM 查询监控规则。

STL_WLM_RULE_ACTION 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	运行查询的用户。
query	integer	查询 ID。
service_class	integer	服务类的 ID。查询队列在 WLM 配置中定义。5 个以上的服务类为用户定义的队列。
规则	character(256)	查询监控规则的名称。
action	character(256)	生成的操作。可能值如下所示： • log • hop(reassign)

列名称	数据类型	描述
		• hop(restart) • abort • change_query_priority • 无 值 none 表示该规则的预测已应验，但操作被严重性更高的另一个规则取代。
recordtime	时间戳	以 UTC 格式记录操作的时间。
action_value	character(256)	如果 action 为 change_query_priority ，则可能的值为 highest、high、normal、low 和 lowest。 如果 action 为 log、hop 或 abort，则该值为空。
service_class_name	character(64)	服务类的名称。

示例查询

以下示例查找由查询监控规则停止的查询。

```
Select query, rule
from stl_wlm_rule_action
where action = 'abort'
order by query;
```

STL_WLM_QUERY

包含在由 WLM 处理的服务类中每次尝试执行查询的记录。

STL_WLM_QUERY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
xid	integer	查询或子查询的事务 ID。
task	integer	用于通过工作负荷管理器跟踪查询的 ID。可与多个查询 ID 关联。如果重新启动了某个查询，则会为该查询分配一个新的查询 ID 但不分配新的任务 ID。
query	integer	查询 ID。如果重新启动了某个查询，则会为该查询分配一个新的查询 ID 但不分配新的任务 ID。
service_class	integer	服务类的 ID。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。
slot_count	整数	根据为队列设置的并发级别，查询使用的 WLM 查询插槽的数量。默认值为 1。有关更多信息，请参阅 wlm_query_slot_count。
service_class_start_time	timestamp	将查询分配给服务类的时间。这个时间在 UTC 时区。
queue_start_time	timestamp	查询进入服务类的队列的时间。这个时间在 UTC 时区。
queue_end_time	timestamp	查询离开服务类的队列的时间。这个时间在 UTC 时区。
total_queue_time	bigint	查询已在队列中消耗的总微秒数
exec_start_time	timestamp	查询开始在服务类中执行的时间。这个时间在 UTC 时区。
exec_end_time	timestamp	查询在服务类中完成执行的时间。这个时间在 UTC 时区。
total_exec_time	bigint	查询进行执行所消耗的微秒数。

列名称	数据类型	描述
service_class_end_time	timestamp	查询离开服务类的时间。这个时间在 UTC 时区。
final_state	character(16)	保留供系统使用。
est_peak_mem	bigint	保留供系统使用。
query_priority	char(20)	查询的优先级。可能的值为 n/a、lowest、low、normal、high 和 highest，其中 n/a 表示不支持查询优先级。
service_class_name	character(64)	服务类名称。有关服务类的更多信息，请参阅 WLM 系统表和视图。

示例查询

查看在队列中等待和执行的平均查询时间

以下查询显示 4 以上的服务类的当前配置。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。

以下查询返回针对每个服务类的每个查询在查询队列和执行过程中消耗的平均时间（单位为微秒）。

```
select service_class as svc_class, count(*),
avg(datediff(microseconds, queue_start_time, queue_end_time)) as avg_queue_time,
avg(datediff(microseconds, exec_start_time, exec_end_time )) as avg_exec_time
from stl_wlm_query
where service_class > 4
group by service_class
order by service_class;
```

此查询返回以下示例输出：

```
 svc_class | count | avg_queue_time | avg_exec_time
-----------+-------+----------------+---------------
         5 | 20103 |              0 |         80415
         5 |  3421 |          34015 |        234015
         6 |    42 |              0 |        944266
         7 |   196 |           6439 |       1364399
```

```
(4 rows)
```

查看在队列中等待和执行的最大查询时间

以下查询返回针对每个服务类的每个查询在任何查询队列和执行过程中消耗的总时间量（单位为微秒）。

```
select service_class as svc_class, count(*),
max(datediff(microseconds, queue_start_time, queue_end_time)) as max_queue_time,
max(datediff(microseconds, exec_start_time, exec_end_time )) as max_exec_time
from stl_wlm_query
where svc_class > 5
group by service_class
order by service_class;
```

```
 svc_class | count | max_queue_time | max_exec_time
-----------+-------+----------------+---------------
         6 |    42 |              0 |       3775896
         7 |   197 |          37947 |      16379473
(4 rows)
```

快照数据的 STV 表

STV 表是虚拟系统表，包含当前系统数据的快照。

主题

- STV_ACTIVE_CURSORS
- STV_BLOCKLIST
- STV_CURSOR_CONFIGURATION
- STV_DB_ISOLATION_LEVEL
- STV_EXEC_STATE
- STV_INFLIGHT
- STV_LOAD_STATE
- STV_LOCKS
- STV_ML_MODEL_INFO
- STV_MV_DEPS

- STV_MV_INFO
- STV_NODE_STORAGE_CAPACITY
- STV_PARTITIONS
- STV_QUERY_METRICS
- STV_RECENTS
- STV_SESSIONS
- STV_SLICES
- STV_STARTUP_RECOVERY_STATE
- STV_TBL_PERM
- STV_TBL_TRANS
- STV_WLM_CLASSIFICATION_CONFIG
- STV_WLM_QMR_CONFIG
- STV_WLM_QUERY_QUEUE_STATE
- STV_WLM_QUERY_STATE
- STV_WLM_QUERY_TASK_STATE
- STV_WLM_SERVICE_CLASS_CONFIG
- STV_WLM_SERVICE_CLASS_STATE
- STV_XRESTORE_ALTER_QUEUE_STATE

STV_ACTIVE_CURSORS

STV_ACTIVE_CURSORS 显示当前打开的光标的详细信息。有关更多信息，请参阅 DECLARE。

STV_ACTIVE_CURSORS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。用户只能查看自己打开的光标。超级用户可以查看所有光标。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。

列名称	数据类型	描述
名称	characte (256)	光标名称。
xid	bigint	事务上下文。
pid	integer	运行查询的领导节点进程。
starttime	timestam	声明光标的时间。
row_count	bigint	光标结果集中的行数。
byte_count	bigint	光标结果集中的字节数。
fetched_r ows	bigint	当前从光标结果集提取的行数。

STV_BLOCKLIST

STV_BLOCKLIST 包含数据库中每个分片、表或列所使用的 1 MB 磁盘块的数目。

将聚合查询与 STV_BLOCKLIST 结合使用（如以下示例所示）可确定为每个数据库、表、分片或列分配的 1 MB 磁盘块的数目。您还可以使用 STV_PARTITIONS 查看有关磁盘利用率的摘要信息。

STV_BLOCKLIST 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
slice	integer	节点分片。
col	integer	列的零基索引。您创建的每个表都附加有以下三个隐藏列：INSERT_XID、DELETE_XID 和 ROW_ID (OID)。包含 3 个用户定义的列的表实际上包含 6 列，用户定义的列在内部编号为 0、1 和 2。在此示例中，INSERT_XID、DELETE_XID 和 ROW_ID 列分别编号为 3、4 和 5。

列名称	数据类型	描述
tbl	integer	数据库表的表 ID。
blocknum	integer	数据块的 ID。
num_values	integer	数据块上包含的值的数目。
extended_limits	integer	供内部使用。
minvalue	bigint	块的最小数据值。将非数字数据的前八个字符存储为 64 位整数。用于磁盘扫描。
maxvalue	bigint	块的最大数据值。将非数字数据的前八个字符存储为 64 位整数。用于磁盘扫描。
sb_pos	integer	磁盘上的超级数据块位置的内部 Amazon Redshift 标识符。
pinned	integer	是否在预加载时将数据块固定到内存中。0 = false；1 = true。默认设置为“假”。
on_disk	integer	是否自动在磁盘上存储数据块。0 = false；1 = true。默认设置为“假”。
modified	integer	是否已修改数据块。0 = false；1 = true。默认设置为“false”。
hdr_modified	integer	是否已修改数据块标头。0 = false；1 = true。默认设置为“假”。
unsorted	integer	数据块是否未排序。0 = false；1 = true。默认为“真”。
tombstone	integer	供内部使用。
preferred_diskno	integer	数据块应该位于的磁盘的编号（除非磁盘已出故障）。磁盘一旦修复，该数据块就将移回到该磁盘。
temporary	integer	数据块是否包含临时表或中间查询结果等位置的临时数据。0 = false；1 = true。默认设置为“假”。

列名称	数据类型	描述
newblock	integer	指示数据块是新数据块 (true) 还是以前从未提交到过磁盘 (false)。0 = false；1 = true。
num_readers	integer	每个块上的引用数。
flags	integer	块标头的内部 Amazon Redshift 标志。

示例查询

分配的每个磁盘块在 STV_BLOCKLIST 中对应一行，因此选择所有行的查询可能会返回非常多的行。建议仅将聚合查询与 STV_BLOCKLIST 结合使用。

SVV_DISKUSAGE 视图以对用户更友好的格式提供相似信息；而以下示例演示 STV_BLOCKLIST 表的一个用例。

要确定 VENUE 表中每一列所使用的 1 MB 块数，请键入以下查询：

```
select col, count(*)
from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl = stv_tbl_perm.id
and stv_blocklist.slice = stv_tbl_perm.slice
and stv_tbl_perm.name = 'venue'
group by col
order by col;
```

此查询返回分配到 VENUE 表中每一列的 1 MB 块的数目，如以下示例数据所示：

```
 col | count
-----+-------
   0 |  4
   1 |  4
   2 |  4
   3 |  4
   4 |  4
   5 |  4
   7 |  4
   8 |  4
```

```
(8 rows)
```

以下查询显示表数据是否实际分布于所有分片间：

```
select trim(name) as table, stv_blocklist.slice, stv_tbl_perm.rows
from stv_blocklist,stv_tbl_perm
where stv_blocklist.tbl=stv_tbl_perm.id
and stv_tbl_perm.slice=stv_blocklist.slice
and stv_blocklist.id > 10000 and name not like '%#m%'
and name not like 'systable%'
group by name, stv_blocklist.slice, stv_tbl_perm.rows
order by 3 desc;
```

此查询生成以下示例输出，其中显示行数最多的表的平均数据分布：

```
table   | slice | rows
----------+-------+-------
listing  |    13 | 10527
listing  |    14 | 10526
listing  |     8 | 10526
listing  |     9 | 10526
listing  |     7 | 10525
listing  |     4 | 10525
listing  |    17 | 10525
listing  |    11 | 10525
listing  |     5 | 10525
listing  |    18 | 10525
listing  |    12 | 10525
listing  |     3 | 10525
listing  |    10 | 10525
listing  |     2 | 10524
listing  |    15 | 10524
listing  |    16 | 10524
listing  |     6 | 10524
listing  |    19 | 10524
listing  |     1 | 10523
listing  |     0 | 10521
...
(180 rows)
```

以下查询确定是否向磁盘提交了任何已逻辑删除的块：

```
select slice, col, tbl, blocknum, newblock
from stv_blocklist
where  tombstone > 0;

slice | col |   tbl   | blocknum | newblock
-------+-----+--------+----------+----------
4     |  0  | 101285 |    0     |   1
4     |  2  | 101285 |    0     |   1
4     |  4  | 101285 |    1     |   1
5     |  2  | 101285 |    0     |   1
5     |  0  | 101285 |    0     |   1
5     |  1  | 101285 |    0     |   1
5     |  4  | 101285 |    1     |   1
...
(24 rows)
```

STV_CURSOR_CONFIGURATION

STV_CURSOR_CONFIGURATION 显示光标配置约束。有关更多信息，请参阅 游标约束。

STV_CURSOR_CONFIGURATION 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
current_cursor_count	integer	当前打开的光标数。
max_diskspace_usable	integer	可用于光标的磁盘空间量（以 MB 为单位）。此约束基于光标的最大光标结果集大小。
current_diskspace_used	integer	光标当前使用的磁盘空间量（以 MB 为单位）。

STV_DB_ISOLATION_LEVEL

STV_DB_ISOLATION_LEVEL 显示数据库的当前隔离级别。有关隔离级别的更多信息，请参阅 CREATE DATABASE。

STV_DB_ISOLATION_LEVEL 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
db_name	character (128)	数据库名称。
isolation_level	character (20)	数据库的隔离级别。可能的值包括 `Serializable` 和 `Snapshot Isolation` 。

STV_EXEC_STATE

使用 STV_EXEC_STATE 表可查看有关正在计算节点上主动运行的查询和查询步骤的信息。

此信息通常仅用于排查工程问题。视图 SVV_QUERY_STATE 和 SVL_QUERY_SUMMARY 从 STV_EXEC_STATE 提取其信息。

STV_EXEC_STATE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。
query	integer	查询 ID。可用于联接各种其他系统表和视图。

列名称	数据类型	描述
slice	integer	步骤完成所在的节点切片。
segment	integer	运行的查询段。查询段是一系列步骤。
step	integer	完成的查询段步骤。步骤是查询执行的最小单元。
starttime	timestamp	步骤运行的时间。
currenttime	timestamp	当前时间。
tasknum	integer	分配到完成步骤的查询任务进程。
rows	bigint	已处理的行数。
bytes	bigint	已处理的字节数。
label	char(256)	由查询步骤名称和适用的表 ID 和表名组成的步骤标签（例如 `scan tbl=100448 name =user` ）。三位表 ID 通常是指临时表的扫描。如果显示 `tbl=0` ，则通常是指扫描常量值。
is_diskbased	char(1)	此查询步骤是否已作为基于磁盘的操作完成：true（**t**）或 false（**f**）。只有哈希、排序和聚合等特定步骤才能转到磁盘。许多类型的步骤始终在内存中完成。
workmem	bigint	分配给步骤的工作内存的字节数。
num_parts	integer	执行哈希步骤期间将哈希表划分成的分区数。此列中的正数并不表示哈希步骤是作为基于磁盘的操作完成的。请查看 IS_DISKBASED 列中的值以了解哈希步骤是否基于磁盘。
is_rrscan	char(1)	如果为 true (**t**)，则表示对步骤使用了限制范围的扫描。默认为 false (**f**)。
is_delayed_scan	char(1)	如果为 true (**t**)，则表示对步骤使用了延迟扫描。默认为 false (**f**)。

示例查询

Amazon Redshift 建议查询 SVL_QUERY_SUMMARY 或 SVV_QUERY_STATE 以对用户更友好的格式获取 STV_EXEC_STATE 中的信息，而不是直接查询 STV_EXEC_STATE。有关更多详细信息，请参阅 SVL_QUERY_SUMMARY 或 SVV_QUERY_STATE 表文档。

STV_INFLIGHT

使用 STV_INFLIGHT 表可确定当前正在对集群运行的查询。如果您正在进行故障排除，这有助于检查长时间运行的查询的状态。

STV_INFLIGHT 不会显示仅领导节点查询。有关更多信息，请参阅 仅领导节点函数。STV_INFLIGHT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

使用 STV_INFLIGHT 进行故障排除

如果您使用 STV_INFLIGHT 对一个查询或一组查询的性能进行故障排除，请注意以下几点：

- 长时间运行的未结事务通常会增加负载。这些未结事务可能会导致其他查询的运行时间更长。
- 如果长时间运行的 COPY 和 ETL 任务占用了大量计算资源，则可能会影响集群上运行的其他查询。在大多数情况下，将这些长时间运行的任务移至使用率较低的时间，可以提高报告或分析工作负载的性能。
- 有些视图为 STV_INFLIGHT 提供了相关信息。其中包括 STL_QUERYTEXT（用于捕获 SQL 命令的查询文本）和 SVV_QUERY_INFLIGHT（用于将 STV_INFLIGHT 联接到 STL_QUERYTEXT）。您也可以将 STV_RECENTS 与 STV_INFLIGHT 结合使用来进行故障排除。例如，STV_RECENTS 可以指示特定查询是处于正在运行 还是已完成 状态。将这些信息与来自 STV_INFLIGHT 的结果相结合，可以为您提供有关查询的属性和计算资源影响的更多信息。

您还可以使用 Amazon Redshift 控制台监控正在运行的查询。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。

列名称	数据类型	描述
slice	integer	正在运行查询的分片。
query	integer	查询 ID。可用于联接各种其他系统表和视图。
label	character(320)	用于运行查询的文件的名称或使用 SET QUERY_GROUP 命令定义的标签。如果查询并非基于文件或未设置 QUERY_GROUP 参数，则此字段为空。
xid	bigint	事务 ID。
pid	integer	进程 ID。会话中的所有查询在同一进程中运行，因此，如果您在同一会话中运行一系列查询，则此值保持不变。您可以使用此列联接到 STL_ERROR 表。
starttime	timestamp	开始查询的时间。
text	character(100)	查询文本，截断为 100 个字符（如果语句超过此限制）。
suspended	integer	查询是否已暂停。0 = false；1 = true。
insert_pristine	integer	当前查询正在运行时，是否可以运行写查询。1 = 不允许写查询。0 = 允许写查询。此列专用在调试中。
concurrency_scaling_status	integer	指示查询运行在主集群还是并发扩展集群上，可能值如下所示： 0 - 运行在主集群上 1 - 运行在并发扩展集群上

示例查询

要查看当前正在对数据库运行的所有活动查询，请键入以下查询：

```
select * from stv_inflight;
```

下面的示例输出显示当前正在运行两个查询，其中包括 STV_INFLIGHT 查询自身和已从名为 `avgwait.sql` 的脚本运行的查询：

```
select slice, query, trim(label) querylabel, pid,
starttime, substring(text,1,20) querytext
from stv_inflight;

slice|query|querylabel | pid |        starttime         |      querytext
-----+-----+-----------+-----+--------------------------+--------------------
1011 |  21 |           | 646 |2012-01-26 13:23:15.645503|select slice, query,
1011 |  20 |avgwait.sql| 499 |2012-01-26 13:23:14.159912|select avg(datediff(
(2 rows)
```

以下查询选择了几列，包括 concurrency_scaling_status。此列表示是否正在向并发扩展集群发送查询。如果对于某些结果，该值为 1，则表明正在使用并发扩展计算资源。有关更多信息，请参阅 使用并发扩展。

```
select userid,
query,
pid,
starttime,
text,
suspended,
concurrency_scaling_status
 from STV_INFLIGHT;
```

示例输出显示了正在将一个查询发送到并发扩展集群。

```
 query  | pid     |        starttime           |   text                 | suspended
 |  concurrency_scaling_status
--------+---------
+----------------------------|------------------------|--------------|-----------------------
1234567 | 123456  | 2012-01-26 13:23:15.645503 | select userid, query...  0
    1
2345678 | 234567  | 2012-01-26 13:23:14.159912 | select avg(datediff(...  0
    0
(2 rows)
```

有关排除查询性能故障的更多提示，请参阅查询故障排除。

STV_LOAD_STATE

使用 STV_LOAD_STATE 表可以查找有关正在进行的 COPY 语句当前状态的信息。

每加载百万条记录后，COPY 命令就会更新此表。

STV_LOAD_STATE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。
session	integer	执行加载的进程的会话 PID。
query	integer	查询 ID。可用于联接各种其他系统表和视图。
slice	integer	节点分片编号。
pid	integer	进程 ID。会话中的所有查询在同一进程中运行，因此，如果您在同一会话中运行一系列查询，则此值保持不变。
recordtime	timestamp	记录的记录时间。
bytes_to_load	bigint	此分片要加载的总字节数。如果要加载的数据已压缩，则为 0。
bytes_loaded	bigint	此分片加载的字节数。如果要加载的数据已压缩，则这是指解压缩数据后加载的字节数。
bytes_to_load_compressed	bigint	此分片要加载的已压缩数据的总字节数。如果要加载的数据未压缩，则为 0。
bytes_loaded_compressed	bigint	此分片加载的已压缩数据的字节数。如果要加载的数据未压缩，则为 0。
lines	integer	此分片加载的行数。
num_files	integer	此分片要加载的文件数。
num_files_complete	integer	此分片加载的文件数。
current_file	character (256)	此分片要加载的文件的名称。

列名称	数据类型	描述
pct_complete	integer	此分片完成的数据加载百分比。

示例查询

要查看 COPY 命令的每个分片进度，请键入以下查询。此示例使用 PG_LAST_COPY_ID() 函数检索最后一条 COPY 命令的信息。

```
select slice , bytes_loaded, bytes_to_load , pct_complete from stv_load_state where
 query = pg_last_copy_id();

 slice | bytes_loaded | bytes_to_load | pct_complete
-------+--------------+---------------+--------------
     2 |            0 |             0 |            0
     3 |     12840898 |      39104640 |           32
(2 rows)
```

STV_LOCKS

使用 STV_LOCKS 表可以查看数据库中各个表的所有当前更新。

Amazon Redshift 锁定表以防止两个用户同时更新同一个表。STV_LOCKS 表显示所有当前表更新，而查询 STL_TR_CONFLICT 表则可查看锁定冲突的日志。使用 SVV_TRANSACTIONS 视图可标识未结事务和锁定争用问题。

STV_LOCKS 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
table_id	bigint	获取锁定的表的表 ID。
last_commit	timestamp	表中上次提交的时间戳。
last_update	timestamp	表上次更新的时间戳。
lock_owner	bigint	与锁定关联的事务 ID。

列名称	数据类型	描述
lock_owner_pid	bigint	与锁定关联的进程 ID。
lock_owner_start_ts	timestamp	事务开始时间的时间戳。
lock_owner_end_ts	timestamp	事务结束时间的时间戳。
lock_status	character (22)	进程状态（等待锁或持有锁）。

示例查询

要查看当前事务中发生的所有锁定，请键入以下命令：

```
select table_id, last_update, lock_owner, lock_owner_pid from stv_locks;
```

此查询返回以下示例输出，其中显示三个当前生效的锁定：

```
 table_id |        last_update         | lock_owner | lock_owner_pid
----------+----------------------------+------------+----------------
100004   | 2008-12-23 10:08:48.882319 |       1043 |           5656
100003   | 2008-12-23 10:08:48.779543 |       1043 |           5656
100140   | 2008-12-23 10:08:48.021576 |       1043 |           5656
(3 rows)
```

STV_ML_MODEL_INFO

有关机器学习模型当前状态的状态信息。

STV_ML_MODEL_INFO 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
schema_name	char(128)	新模型的命名空间。

列名称	数据类型	描述
user_name	char(128)	模型的拥有者。
model_name	char(128)	模型的名称。
life_cycle	char(20)	模型的生命周期状态。
is_refreshable	integer	如果训练查询中的原始表和列仍然存在，并且用户仍具有对它们的权限，则模型的状态为是否可刷新。可能的值为：1（可刷新）和 0（不可刷新）。
model_state	char(128)	模型的当前状态。

示例查询

以下查询显示机器学习模型的当前状态。

```
SELECT schema_name, model_name, model_state
FROM stv_ml_model_info;

 schema_name |        model_name            |             model_state
-------------+------------------------------+-------------------------------------
 public      | customer_churn_auto_model    | Train Model On SageMaker In Progress
 public      | customer_churn_xgboost_model | Model is Ready
(2 row)
```

STV_MV_DEPS

STV_MV_DEPS 表显示了具体化视图与 Amazon Redshift 内其他具体化视图的依赖关系。

有关具体化视图的更多信息，请参阅在 Amazon Redshift 中创建实体化视图。

STV_MV_DEPS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
db_name	char(128)	包含特定具体化视图的数据库。
schema	char(128)	具体化视图的架构。
名称	char(128)	具体化视图的名称。
ref_schema	char(128)	此具体化视图所依赖的具体化视图 schema。
ref_name	char(128)	此具体化视图所依赖的具体化视图的名称。
ref_database_name	char(128)	此实体化视图所依赖的数据库的名称。

示例查询

以下查询返回一个输出行，该行指示具体化视图 mv_over_foo 在其定义中使用具体化视图 mv_foo 作为二依赖项。

```
CREATE SCHEMA test_ivm_setup;
CREATE TABLE test_ivm_setup.foo(a INT);
CREATE MATERIALIZED VIEW test_ivm_setup.mv_foo AS SELECT * FROM test_ivm_setup.foo;
CREATE MATERIALIZED VIEW test_ivm_setup.mv_over_foo AS SELECT * FROM
 test_ivm_setup.mv_foo;

SELECT * FROM stv_mv_deps;

 db_name | schema          | name        |   ref_schema   | ref_name |
 ref_database_name
---------+-----------------+-------------+----------------+----------
+------------------
 dev     | test_ivm_setup  | mv_over_foo | test_ivm_setup | mv_foo   | dev
```

STV_MV_INFO

STV_MV_INFO 表对于每个具体化视图、数据是否陈旧以及状态信息包含一行。

有关具体化视图的更多信息，请参阅[在 Amazon Redshift 中创建实体化视图]。

STV_MV_INFO 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
db_name	char(128)	包含具体化视图的数据库。
schema	char(128)	数据库的架构。
名称	char(128)	具体化视图名称。
updated_upto_xid	bigint	保留供内部使用。
is_stale	char(1)	t 表示实体化视图已过时。过时的实体化视图是基表已更新但尚未刷新实体化视图的视图。如果自上次重新启动以来尚未运行刷新，则信息可能不准确。 如果实体化视图依赖于可变函数，则 is_stale 列始终设置为 t。当给定相同的一个或多个参数时，可变函数会返回不同的结果。例如，大多数返回日期或时间戳的函数都是可变函数。
owner_user_name	char(128)	拥有具体化视图的用户。
state	integer	实体化视图的状态如下： • 0 – 刷新时完全重新计算实体化视图。 • 1 – 实体化视图是递增的。 • 101 – 实体化视图由于删除的列而无法刷新。即使在实体化视图中未使用列，此约束也会适用。 • 102 – 由于更改的列类型，实体化视图无法刷新。即使在实体化视图中未使用列，此约束也会适用。 • 103 – 由于重命名的表，实体化视图无法刷新。

列名称	数据类型	描述
		• 104 – 由于重命名的列，实体化视图无法刷新。即使在实体化视图中未使用列，此约束也会适用。 • 105 – 由于重命名的 schema，实体化视图无法刷新。
自动重写	char(1)	t 表示实体化视图有资格自动重写查询。
自动刷新	char(1)	t 表示实体化视图可以自动刷新。

示例查询

要查看所有实体化视图的状态，请运行以下查询。

```
select * from stv_mv_info;
```

此查询返回以下示例输出。

```
 db_name |       schema       |   name   | updated_upto_xid | is_stale | owner_user_name
 | state | autorefresh | autorewrite
---------+--------------------+---------+------------------+----------
+-----------------+-------+-------------+------------
 dev     | test_ivm_setup     | mv       |             1031 | f        | catch-22
 |     1 |           1 |           0
 dev     | test_ivm_setup     | old_mv   |              988 | t        | lotr
 |     1 |           0 |           1
```

STV_NODE_STORAGE_CAPACITY

STV_NODE_STORAGE_CAPACITY 表显示了集群中每个节点的总存储容量和总已用容量的详细信息。它为每个节点包含一行。

STV_NODE_STORAGE_CAPACITY 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
node	integer	节点编号。
used	integer	节点上当前使用的 1 MB 磁盘块的数目。对于 RA3 节点类型，使用的块包括本地缓存的块和 Amazon S3 中保留的块。
capacity	integer	1 MB 块中为节点预配置的总存储容量。容量包括 Amazon Redshift 在 DC2 节点类型预留供内部使用的空间。容量大于名义节点容量，即可用于用户数据的节点空间量。对于 RA3 节点类型，此容量与集群的总托管存储配额相同。有关按节点类型分类的容量的更多信息，请参阅《Amazon Redshift 管理指南》中的节点类型详细信息。

示例查询

> **Note**
>
> 以下示例的结果因集群的节点规范而异。将列 `capacity` 添加到 SQL SELECT 以检索集群的容量。

以下查询返回 1 MB 磁盘块中的已用空间和总容量。此示例在双节点 dc2.8xlarge 集群上运行。

```
select node, used from stv_node_storage_capacity order by node;
```

此查询返回以下示例输出。

```
 node | used
------+-------
    0 | 30597
    1 | 27089
```

以下查询返回 1 MB 磁盘块中的已用空间和总容量。此示例在一个双节点 ra3.16xlarge 集群上运行。

```
select node, used from stv_node_storage_capacity order by node;
```

此查询返回以下示例输出。

```
 node | used
------+-------
    0 | 30591
    1 | 27103
```

STV_PARTITIONS

使用 STV_PARTITIONS 表可了解 Amazon Redshift 的磁盘速度性能和磁盘利用率。

STV_PARTITIONS 中的一行对应每个节点的一个逻辑磁盘卷。

STV_PARTITIONS 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
owner	integer	拥有分区的磁盘节点。
host	integer	物理上附加到分区的节点。
diskno	integer	包含分区的磁盘。
part_begin	bigint	分区的偏移量。以逻辑方式将原始设备分区到镜像块的开放空间。
part_end	bigint	分区末尾。
used	integer	分区上当前使用的 1 MB 磁盘块的数目。
tossed	integer	已准备好可删除，但由于释放块的磁盘地址的做法不安全而尚未移除的块的数目。如果立即释放了这些地址，挂起的事务便可写入磁盘上的同一位置。因此，这些被丢弃的块会在下

列名称	数据类型	描述
		次提交时释放。磁盘块可能会标记为“被丢弃”，例如，在执行 INSERT 操作或基于磁盘的查询操作期间删除表列时。
capacity	integer	分区的总容量（1 MB 磁盘块数）。
reads	bigint	自上次集群重新启动以来发生的读取数。
writes	bigint	自上次集群重新启动以来发生的写入数。
seek_forward	integer	不是请求下一地址的次数（前一请求地址给定）。
seek_back	integer	不是请求前一地址的次数（下一地址给定）。
is_san	integer	分区是否属于 SAN。有效值为 **0** (false) 或 **1** (true)。
failed	integer	此列已弃用。
mbps	integer	磁盘速度（MB/秒）。
mount	character (256)	设备的目录路径。

示例查询

以下查询返回已用磁盘空间和容量（1 MB 磁盘块数），并计算磁盘利用率（占原始磁盘空间的百分比）。原始磁盘空间包括 Amazon Redshift 保留供内部使用的空间，因此它大于名义磁盘容量（可供用户使用的磁盘空间量）。Amazon Redshift 管理控制台的性能选项卡上的已用磁盘空间百分比指标报告集群使用的名义磁盘容量的百分比。建议您监控 Percentage of Disk Space Used 指标以维持集群的名义磁盘容量的使用量。

> **Important**
>
> 我们强烈建议您不要超过集群的名义磁盘容量。尽管这在某些情况下在技术上可能是可行的，但超过名义磁盘容量会降低集群的容错能力并增加丢失数据的风险。

此示例是在两个节点组成的集群上运行的（每个节点有六个逻辑磁盘分区）。各磁盘的空间使用非常平均，每个磁盘大约使用 25%。

```
select owner, host, diskno, used, capacity,
(used-tossed)/capacity::numeric *100 as pctused
from stv_partitions order by owner;

 owner | host | diskno |  used  | capacity | pctused
-------+------+--------+--------+----------+---------
   0   |  0   |    0   | 236480 |  949954  | 24.9
   0   |  0   |    1   | 236420 |  949954  | 24.9
   0   |  0   |    2   | 236440 |  949954  | 24.9
   0   |  1   |    2   | 235150 |  949954  | 24.8
   0   |  1   |    1   | 237100 |  949954  | 25.0
   0   |  1   |    0   | 237090 |  949954  | 25.0
   1   |  1   |    0   | 236310 |  949954  | 24.9
   1   |  1   |    1   | 236300 |  949954  | 24.9
   1   |  1   |    2   | 236320 |  949954  | 24.9
   1   |  0   |    2   | 237910 |  949954  | 25.0
   1   |  0   |    1   | 235640 |  949954  | 24.8
   1   |  0   |    0   | 235380 |  949954  | 24.8

(12 rows)
```

STV_QUERY_METRICS

包含正在用户定义的查询队列中运行的活动查询的指标信息（如处理的行数、CPU 利用率、输入/输出和磁盘利用率）。要查看已完成的查询的指标，请参阅 STL_QUERY_METRICS 系统表。

查询指标按一秒的间隔采样。因此，同一查询的不同运行可能返回稍微不同的时间。此外，运行不到 1 秒的查询段可能不会记录。

STV_QUERY_METRICS 可跟踪和聚合查询、段和步骤级别的指标。有关查询段和步骤的信息，请参阅查询计划和执行工作流程。很多指标（如 max_rows、cpu_time 等）是跨节点切片进行合计的。有关节点切片的更多信息，请参阅数据仓库系统架构。

要确定行在哪个级别报告指标，请检查 segment 和 step_type 列：

- 如果 segment 和 step_type 均为 -1，则行在查询级别报告指标。
- 如果 segment 不为 -1，而 step_type 为 -1，则行在段级别报告指标。
- 如果 segment 和 step_type 均不为 -1，则行在步骤级别报告指标。

STV_QUERY_METRICS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	运行生成条目的查询的用户的 ID。
service_class	integer	WLM 查询队列（服务类）的 ID。查询队列在 WLM 配置中定义。仅对用户定义的队列报告的指标。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
starttime	timestamp	开始执行查询的时间（用 UTC 表示），有 6 位数字精度，可精确到小数秒。例如：2009-06-12 11:29:19.131358 。
切片	integer	集群的切片数。
segment	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。查询段可并行运行。每个段在一个进程中运行。如果段值为 -1，则指标段值将汇总到查询级别。
step_type	integer	运行的步骤的类型。有关步骤类型的说明，请参阅步骤类型。
rows	bigint	步骤处理的行数。
max_rows	bigint	某个步骤的最大行输出数（跨所有切片进行聚合）。
cpu_time	bigint	使用的 CPU 时间（单位为微秒）。在段级别，段的跨所有切片的总 CPU 时间。在查询级别，查询的跨所有切片和段的 CPU 时间合计。
max_cpu_time	bigint	使用的最大 CPU 时间（以微秒为单位）。在段级别，段的跨所有切片使用的最大 CPU 时间。在查询级别，查询段使用的最大 CPU 时间。
blocks_read	bigint	查询或段读取的 1 MB 数据块的数量。

列名称	数据类型	描述
max_blocks_read	bigint	段读取的 1 MB 数据块的最大数量（跨所有切片进行聚合）。在段级别，段的跨所有切片读取的 1 MB 数据块的最大数量。在查询级别，任何查询段读取的 1 MB 数据块的最大数量。
run_time	bigint	跨所有切片合计的总运行时间。运行时间不包括等待时间。 在段级别，跨所有切片合计的段的运行时间。在查询级别，跨所有切片和段合计的查询的运行时间。由于此值是一个合计，因此运行时间与查询执行时间无关。
max_run_time	bigint	段的最大已用时间（以微秒为单位）。在段级别，段的跨所有切片的最大运行时间。在查询级别，任何查询段的最大运行时间。
max_blocks_to_disk	bigint	用于写入中间结果的最大磁盘空间量（以 1 MB 数据块为单位）。在段级别，段跨所有切片使用的最大磁盘空间量。在查询级别，任何查询段使用的最大磁盘空间量。
blocks_to_disk	bigint	查询和段用于写入中间结果使用的磁盘空间量（以 1 MB 数据块为单位）。
step	integer	运行的查询步骤。
max_query_scan_size	bigint	查询扫描的数据的最大大小（以 MB 为单位）。在段级别，段跨所有切片扫描的数据的最大大小。在查询级别，任何查询段扫描的数据的最大大小。
query_scan_size	bigint	查询扫描的数据的大小（以 MB 为单位）。
query_priority	integer	查询的优先级。可能的值为 -1、0、1、2、3 和 4，其中 -1 表示不支持查询优先级。
query_queue_time	bigint	查询排队的时间（以微秒为单位）。

步骤类型

下表列出了与数据库用户相关的步骤类型。该表未列出仅供内部使用的步骤类型。如果步骤类型为 -1，则不会在步骤级别报告指标。

步骤类型	描述
1	扫描表
2	插入行
3	聚合行
6	为步骤排序
7	合并步骤
8	分配步骤
9	广播分配步骤
10	哈希联接
11	合并联接
12	保存步骤
14	哈希
15	嵌套循环联接
16	项目字段和表达式
17	限制返回的行数
18	唯一
20	删除行
26	限制返回的已排序行的数量
29	计算窗口函数

步骤类型	描述
32	UDF
33	唯一
37	将行从领导节点返回到客户端
38	返回从计算节点到领导节点的行
40	Spectrum 扫描。

示例查询

要查找具有较长的 CPU 时间（1000 秒以上）的活动查询，请运行以下查询。

```
select query, cpu_time / 1000000 as cpu_seconds
from stv_query_metrics where segment = -1 and cpu_time > 1000000000
order by cpu_time;

query | cpu_seconds
------+------------
25775 |        9540
```

要查找具有返回超过一百万个行的嵌套循环联接的活动查询，请运行以下查询。

```
select query, rows
from stv_query_metrics
where step_type = 15 and rows > 1000000
order by rows;

query | rows
------+-----------
25775 | 1580225854
```

要查找已运行超过 60 秒且使用的 CPU 时间不到 10 秒的活动查询，请运行以下查询。

```
select query, run_time/1000000 as run_time_seconds
from stv_query_metrics
where segment = -1 and run_time > 60000000 and cpu_time < 10000000;
```

```
query | run_time_seconds
------+-----------------
25775 |              114
```

STV_RECENTS

使用 STV_RECENTS 表可以了解有关当前活动的和最近运行的数据库查询的信息。

STV_RECENTS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

使用 STV_RECENTS 进行故障排除

STV_RECENTS 对于确定一个查询或一组查询当前是正在运行还是已完成特别有用。它还显示查询运行的持续时间。这有助于了解哪些查询是长时间运行的。

您可以将 STV_RECENTS 联接到其他系统视图（例如 STV_INFLIGHT），以收集有关正在运行的查询的其他元数据。（示例查询部分中有一个示例显示了如何执行此操作。）您还可以使用从该视图返回的记录以及 Amazon Redshift 控制台中的监控功能来实时排除故障。

对 STV_RECENTS 形成补充的系统视图包括 STL_QUERYTEXT 和 SVV_QUERY_INFLIGHT，前者用于检索 SQL 命令的查询文本，后者用于将 STV_INFLIGHT 联接到 STL_QUERYTEXT。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。
status	character(20)	查询状态。有效值为 **Running**、**Done**。
starttime	timestamp	开始查询的时间。
duration	integer	自会话启动以来经过的微秒数。
user_name	character(50)	运行了进程的用户名。

列名称	数据类型	描述
db_name	character(50)	数据库的名称。
query	character(600)	查询文本，最多 600 个字符。截断所有的额外字符。
pid	integer	与查询关联的会话的进程 ID，对于已完成的查询，该值始终为 -1。

示例查询

要确定当前正在对数据库运行哪些查询，请运行以下查询：

```
select user_name, db_name, pid, query
from stv_recents
where status = 'Running';
```

下面的示例输出显示对 TICKIT 数据库运行的单个查询：

```
user_name | db_name |   pid   | query
----------+---------+---------+-------------
dwuser    | tickit  |  19996  |select venuename, venueseats from
venue where venueseats > 50000 order by venueseats desc;
```

以下示例返回正在运行或者在队列中等待运行的查询（如果有）的列表：

```
select * from stv_recents where status<>'Done';

status |    starttime        | duration |user_name|db_name| query     | pid
-------+---------------------+----------+---------+-------+-----------+------
Running| 2010-04-21 16:11... | 281566454| dwuser  |tickit | select ...| 23347
```

此查询不返回结果，除非正在运行一些并发查询，而其中一些查询位于队列中。

以下示例扩展前一示例。在此示例中，从结果中排除真正“正在进行”（正在运行，未在等待）的查询：

```
select * from stv_recents where status<>'Done'
and pid not in (select pid from stv_inflight);
```

```
...
```

有关排除查询性能故障的更多提示，请参阅查询故障排除。

STV_SESSIONS

使用 STV_SESSIONS 表可以查看有关 Amazon Redshift 的活动用户会话的信息。

要查看会话历史记录，请使用 STL_SESSIONS 表而非 STV_SESSIONS。

STV_SESSIONS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_SESSION_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
starttime	timestamp	会话的开始时间。
process	integer	会话的进程 ID。
user_name	character(50)	与会话关联的用户。
db_name	character(50)	与会话关联的数据库的名称。
timeout_sec	int	超时前会话保持非活动状态或空闲状态的最长时间（秒）。0 表示未设置超时。

示例查询

要执行快速检查以了解是否有任何其他用户当前登录到 Amazon Redshift，请键入以下查询：

```
select count(*)
from stv_sessions;
```

如果结果大于 1，则表示至少有一个其他用户当前登录到该数据库。

要查看 Amazon Redshift 的所有活动会话，请键入以下查询：

```
select *
from stv_sessions;
```

下面的结果显示当前在 Amazon Redshift 上运行的四个活动会话：

```
       starttime         | process |user_name                   | db_name
      | timeout_sec
--------------------------+---------+----------------------------
+-----------------------------+-------------
  2018-08-06 08:44:07.50 |   13779 | IAMA:aws_admin:admin_grp   | dev
      | 0
  2008-08-06 08:54:20.50 |   19829 | dwuser                     | dev
      | 120
  2008-08-06 08:56:34.50 |   20279 | dwuser                     | dev
      | 120
  2008-08-06 08:55:00.50 |   19996 | dwuser                     | tickit
      | 0
(3 rows)
```

前缀为 IAMA 的用户名表示此用户是使用联合单点登录进行登录的。有关更多信息，请参阅使用 IAM 身份验证生成数据库用户凭证。

STV_SLICES

使用 STV_SLICES 表可以查看分片到节点的当前映射。

STV_SLICES 中的信息主要用于调查目的。

STV_SLICES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
node	integer	分片所在的集群节点。

列名称	数据类型	描述
slice	integer	节点分片。
localslice	integer	此信息仅供内部使用。
type	character (1)	此信息仅供内部使用。

示例查询

要查看哪些集群节点管理哪些分片，请键入以下查询：

```
select node, slice from stv_slices;
```

此查询返回以下示例输出：

```
 node | slice
------+-------
    0 |     2
    0 |     3
    0 |     1
    0 |     0
(4 rows)
```

STV_STARTUP_RECOVERY_STATE

记录在执行集群重新启动操作期间临时锁定的表的状态。Amazon Redshift 对于正在处理以解决集群重新启动后过时的事务的表，会临时锁定这些表。

STV_STARTUP_RECOVERY_STATE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
db_id	integer	数据库 ID。

列名称	数据类型	描述
table_id	integer	表 ID。
table_name	character(137)	表名称。

示例查询

要监控临时锁定了哪些表，请在集群重新启动之后运行以下查询。

```
select * from STV_STARTUP_RECOVERY_STATE;

  db_id | tbl_id | table_name
--------+--------+------------
 100044 | 100058 | lineorder
 100044 | 100068 | part
 100044 | 100072 | customer
 100044 | 100192 | supplier
(4 rows)
```

STV_TBL_PERM

STV_TBL_PERM 表包含有关 Amazon Redshift 中永久表的信息，其中包括用户为当前会话创建的临时表。STV_TBL_PERM 包含所有数据库中的所有表的信息。

此表不同于 STV_TBL_TRANS，后者包含有关系统在查询处理期间创建的临时数据库表的信息。

STV_TBL_PERM 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
slice	integer	分配到表的节点分片。
id	integer	表 ID。
名称	character (72)	表名称。

列名称	数据类型	描述
rows	bigint	分片中的数据行数。
sorted_rows	bigint	已在磁盘上排序的分片中行的数目。如果此数目与 ROWS 数目不符，则真空化表以重新排序行。
temp	integer	该表是否为临时表。0 = false；1 = true。
db_id	integer	表创建于的数据库的 ID。
insert_pristine	integer	供内部使用。
delete_pristine	integer	供内部使用。
backup	integer	指示表是否包含在集群快照中的值。0 = no；1 = yes。有关更多信息，请参阅 CREATE TABLE 命令的 BACKUP 参数。
dist_style	整数	切片所属表的分布方式。有关值的更多信息，请参阅查看分配方式。有关分布方式的更多信息，请参阅分配方式。
block_count	整数	切片使用的数据块数。在无法计算数据块数时，该值为 -1。

示例查询

以下查询返回不同表 ID 和名称的列表：

```
select distinct id, name
from stv_tbl_perm order by name;

   id   |          name
--------+------------------------
 100571 | category
 100575 | date
 100580 | event
 100596 | listing
 100003 | padb_config_harvest
 100612 | sales
```

```
...
```

其他系统表使用表 ID，因此知道特定表对应于哪个表 ID 会非常有用。在此示例中，SELECT DISTINCT 用于删除重复项（表分布在多个分片中）。

要确定 VENUE 表中每一列所使用的块数，请键入以下查询：

```
select col, count(*)
from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl = stv_tbl_perm.id
and stv_blocklist.slice = stv_tbl_perm.slice
and stv_tbl_perm.name = 'venue'
group by col
order by col;

 col | count
-----+-------
   0 |     8
   1 |     8
   2 |     8
   3 |     8
   4 |     8
   5 |     8
   6 |     8
   7 |     8
(8 rows)
```

使用说明

ROWS 列包括尚未真空化（或者已真空化但使用 SORT ONLY 选项）的已删除行的计数。因此，当您直接查询给定表时，STV_TBL_PERM 表中的 ROWS 列的 SUM 可能与 COUNT(*) 结果不符。例如，如果从 VENUE 中删除了两行，则 COUNT(*) 结果为 200，但 SUM(ROWS) 结果仍为 202：

```
delete from venue
where venueid in (1,2);

select count(*) from venue;
count
-------
200
(1 row)
```

```
select trim(name) tablename, sum(rows)
from stv_tbl_perm where name='venue' group by name;

tablename | sum
-----------+-----
venue     | 202
(1 row)
```

要同步 STV_TBL_PERM 中的数据，请对 VENUE 表运行完全真空化。

```
vacuum venue;

select trim(name) tablename, sum(rows)
from stv_tbl_perm
where name='venue'
group by name;

tablename | sum
-----------+-----
venue     | 200
(1 row)
```

STV_TBL_TRANS

使用 STV_TBL_TRANS 表可以找到有关当前位于内存中的临时数据库表的信息。

临时表通常是在运行查询时用作中间结果的临时行集。STV_TBL_TRANS 不同于 STV_TBL_PERM，因为 STV_TBL_PERM 包含有关永久数据库表的信息。

STV_TBL_TRANS 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
slice	integer	分配到表的节点分片。
id	integer	表 ID。
rows	bigint	表中的数据行数。
size	bigint	分配到表的字节数。

列名称	数据类型	描述
query_id	bigint	查询 ID。
ref_cnt	integer	引用数。
from_suspended	integer	此表是否已在现已暂停的查询期间创建。
prep_swap	integer	此临时表是否准备交换到磁盘（如果需要）。（仅在内存低的情况下发生交换。）

示例查询

要查看查询 ID 为 90 的查询的临时表信息，请键入以下命令：

```
select slice, id, rows, size, query_id, ref_cnt
from stv_tbl_trans
where query_id = 90;
```

此查询返回查询 90 的临时表信息，如以下示例输出中所示：

```
slice | id | rows | size | query_ | ref_ | from_     | prep_
      |    |      |      | id     | cnt  | suspended | swap
------+----+------+------+--------+------+-----------+-------
 1013 | 95 |    0 |    0 |     90 |    4 |         0 |   0
    7 | 96 |    0 |    0 |     90 |    4 |         0 |   0
   10 | 96 |    0 |    0 |     90 |    4 |         0 |   0
   17 | 96 |    0 |    0 |     90 |    4 |         0 |   0
   14 | 96 |    0 |    0 |     90 |    4 |         0 |   0
    3 | 96 |    0 |    0 |     90 |    4 |         0 |   0
 1013 | 99 |    0 |    0 |     90 |    4 |         0 |   0
    9 | 96 |    0 |    0 |     90 |    4 |         0 |   0
    5 | 96 |    0 |    0 |     90 |    4 |         0 |   0
   19 | 96 |    0 |    0 |     90 |    4 |         0 |   0
    2 | 96 |    0 |    0 |     90 |    4 |         0 |   0
 1013 | 98 |    0 |    0 |     90 |    4 |         0 |   0
   13 | 96 |    0 |    0 |     90 |    4 |         0 |   0
    1 | 96 |    0 |    0 |     90 |    4 |         0 |   0
 1013 | 96 |    0 |    0 |     90 |    4 |         0 |   0
    6 | 96 |    0 |    0 |     90 |    4 |         0 |   0
```

```
   11 | 96 |    0 |    0 |     90 |    4 |         0 |   0
   15 | 96 |    0 |    0 |     90 |    4 |         0 |   0
   18 | 96 |    0 |    0 |     90 |    4 |         0 |   0
```

在此示例中，您可以看到查询数据涉及表 95、96 和 98。由于未向此表分配任何字节数，因此可在内存中运行此查询。

STV_WLM_CLASSIFICATION_CONFIG

包含 WLM 的当前分类规则。

STV_WLM_CLASSIFICATION_CONFIG 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
id	integer	服务类 ID。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。
condition	character(128)	查询条件。
action_seq	integer	保留供系统使用。
操作	character(64)	保留供系统使用。
action_service_class	integer	操作发生于的服务类。

示例查询

```
select * from STV_WLM_CLASSIFICATION_CONFIG;

id | condition                                    | action_seq | action |
 action_service_class
---+----------------------------------------------+------------+--------
+---------------------
 1 | (system user) and (query group: health)      |          0 | assign |
     1
```

```
 2 | (system user) and (query group: metrics)    |          0 | assign |
     2
 3 | (system user) and (query group: cmstats)    |          0 | assign |
     3
 4 | (system user)                               |          0 | assign |
     4
 5 | (super user) and (query group: superuser)   |          0 | assign |
     5
 6 | (query group: querygroup1)                  |          0 | assign |
     6
 7 | (user group: usergroup1)                    |          0 | assign |
     6
 8 | (user group: usergroup2)                    |          0 | assign |
     7
 9 | (query group: querygroup3)                  |          0 | assign |
     8
10 | (query group: querygroup4)                  |          0 | assign |
     9
11 | (user group: usergroup4)                    |          0 | assign |
     9
12 | (query group: querygroup*)                  |          0 | assign |
    10
13 | (user group: usergroup*)                    |          0 | assign |
    10
14 | (querytype: any)                            |          0 | assign |
    11
(4 rows)
```

STV_WLM_QMR_CONFIG

记录 WLM 查询监控规则 (QMR) 的配置。有关更多信息，请参阅 WLM 查询监控规则。

STV_WLM_QMR_CONFIG 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
service_class	integer	WLM 查询队列（服务类）的 ID。查询队列在 WLM 配置中定义。仅为用户定义的队列定义规则。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。

列名称	数据类型	描述
rule_name	character(256)	查询监控规则的名称。
action	character(256)	规则操作。可能的值为 log、hop、abort 和 change_query_priority 。
metric_name	character(256)	指标的名称。
metric_operator	character(256)	指标运算符。可能的值为 >、=、<。
metric_value	double	指定指标的可触发操作的阈值。
action_value	character(256)	如果 action 为 change_query_priority ，则可能的值为 highest、high、normal、low 和 lowest。 如果 action 为 log、hop 或 abort，则该值为空。

示例查询

要查看 5 以上的所有服务类（包括用户定义的队列）的 QMR 规则定义，请运行以下查询。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。

```
Select *
from stv_wlm_qmr_config
where service_class > 5
order by service_class;
```

STV_WLM_QUERY_QUEUE_STATE

记录服务类的查询队列的当前状态。

STV_WLM_QUERY_QUEUE_STATE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
service_class	integer	服务类的 ID。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。
position	integer	查询在队列中的位置。具有最小 **position** 值的查询下一个运行。
task	integer	用于通过工作负荷管理器跟踪查询的 ID。可与多个查询 ID 关联。如果重新启动了某个查询，则会为该查询分配一个新的查询 ID 但不分配新的任务 ID。
query	integer	查询 ID。如果重新启动了某个查询，则会为该查询分配一个新的查询 ID 但不分配新的任务 ID。
slot_count	integer	WLM 查询槽位数。
start_time	timestamp	查询进入队列的时间。
queue_time	bigint	查询处于队列中的微秒数。

示例查询

以下查询显示 4 个以上服务类的队列中的查询。

```
select * from stv_wlm_query_queue_state
where service_class > 4
order by service_class;
```

此查询返回以下示例输出。

```
 service_class | position | task | query | slot_count |        start_time          |
 queue_time
---------------+----------+------+-------+------------+----------------------------
+------------
```

```
             5 |        0 |  455 |   476 |          5 | 2010-10-06 13:18:24.065838 |
 20937257
             6 |        1 |  456 |   478 |          5 | 2010-10-06 13:18:26.652906 |
 18350191
(2 rows)
```

STV_WLM_QUERY_STATE

记录 WLM 正在跟踪的查询的当前状态。

STV_WLM_QUERY_STATE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
xid	integer	查询或子查询的事务 ID。
task	integer	用于通过工作负荷管理器跟踪查询的 ID。可与多个查询 ID 关联。如果重新启动了某个查询，则会为该查询分配一个新的查询 ID 但不分配新的任务 ID。
query	integer	查询 ID。如果重新启动了某个查询，则会为该查询分配一个新的查询 ID 但不分配新的任务 ID。
service_class	integer	服务类的 ID。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。
slot_count	integer	WLM 查询槽位数。
wlm_start_time	timestamp	查询进入系统表队列或短查询队列的时间。
state	character(16)	查询或子查询的当前状态。 可能的值包括： • `Classified` – 查询已分配给服务类。

列名称	数据类型	描述
		• Completed – 查询已完成运行。查询要么成功运行，要么已被取消。对于最终状态，请检查 STL_QUERY 的结果。 • Dequeued – 仅限内部使用。 • Evicted – 查询已从服务类中移出以重新启动。 • Evicting – 正在从服务类中移出查询以重新启动。 • Initialized – 仅限内部使用。 • Invalid – 仅限内部使用。 • Queued – 查询被发送到查询队列，因为没有可用于运行查询队列的插槽。 • QueuedWaiting – 查询在查询队列中等待。 • Rejected – 仅限内部使用。 • Returning – 查询将结果返回到客户端。 • Running – 查询正在运行。 • TaskAssigned – 仅限内部使用。
queue_time	bigint	查询在队列中已耗费的微秒数。
exec_time	bigint	查询已运行的微秒数。
query_priority	char(20)	查询的优先级。可能的值为 n/a、lowest、low、normal、high 和 highest，其中 n/a 表示不支持查询优先级。

示例查询

以下查询显示当前正在 4 以上服务类中执行的所有查询。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。

```
select xid, query, trim(state) as state, queue_time, exec_time
from stv_wlm_query_state
where service_class > 4;
```

此查询返回以下示例输出：

```
xid    | query | state   | queue_time | exec_time
-------+-------+---------+------------+-----------
100813 | 25942 | Running |          0 |    1369029
100074 | 25775 | Running |          0 | 2221589242
```

STV_WLM_QUERY_TASK_STATE

包含服务类查询任务的当前状态。

STV_WLM_QUERY_TASK_STATE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
service_class	integer	服务类的 ID。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。
task	integer	用于通过工作负荷管理器跟踪查询的 ID。可与多个查询 ID 关联。如果重新启动了某个查询，则会为该查询分配一个新的查询 ID 但不分配新的任务 ID。
query	integer	查询 ID。如果重新启动了某个查询，则会为该查询分配一个新的查询 ID 但不分配新的任务 ID。
slot_count	integer	WLM 查询槽位数。
start_time	timestamp	查询开始执行的时间。
exec_time	bigint	查询一直在执行的微秒数。

示例查询

以下查询显示当前 4 以上服务类中查询的当前状态。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。

```
select * from stv_wlm_query_task_state
where service_class > 4;
```

此查询返回以下示例输出：

```
service_class | task | query |         start_time         | exec_time
--------------+------+-------+----------------------------+-----------
    5         |  466 |   491 | 2010-10-06 13:29:23.063787 | 357618748
(1 row)
```

STV_WLM_SERVICE_CLASS_CONFIG

记录 WLM 的服务类配置。

STV_WLM_SERVICE_CLASS_CONFIG 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
service_class	integer	服务类的 ID。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。
queueing_strategy	character(32)	保留供系统使用。
num_query_tasks	integer	服务类当前的实际并发级别。如果 num_query_tasks 和 target_num_query_tasks 不相同，则表示正在进行动态 WLM 转换。值为 -1 指示已配置自动 WLM。
target_num_query_tasks	integer	通过最新的 WLM 配置更改设置的并发级别。
evictable	character(8)	保留供系统使用。
eviction_threshold	bigint	保留供系统使用。
query_working_mem	integer	当前分配给服务类的实际工作内存量（以每槽、每节点 MB 数为单位）。如果 query_working_mem 和 target_query_working_mem 不相同，则表示正在进行动态 WLM 转换。值为 -1 指示已配置自动 WLM。

列名称	数据类型	描述
target_query_working_mem	integer	通过最新的 WLM 配置更改设置的工作内存量（以每槽、每节点 MB 数为单位）。
min_step_mem	integer	保留供系统使用。
名称	character(64)	服务类的名称。
max_execution_time	bigint	查询在终止前可运行的毫秒数。
user_group_wild_card	Boolean	如果为 TRUE，则 WLM 队列将星号 (*) 视为 WLM 配置的用户组字符串中的通配符。
query_group_wild_card	Boolean	如果为 TRUE，则 WLM 队列将星号 (*) 视为 WLM 配置的查询组字符串中的通配符。
concurrency_scaling	character(20)	描述并发扩展是为 on 还是 off。
query_priority	character(20)	查询优先级的值。
user_role_wild_card	布尔值	如果为 TRUE，则 WLM 队列将星号 (*) 视为 WLM 配置的用户字符串中的通配符。

示例查询

第一个用户定义的服务类为服务类 6（名为 Service class #1）。以下查询显示 4 以上的服务类的当前配置。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。

```
select rtrim(name) as name,
num_query_tasks as slots,
query_working_mem as mem,
max_execution_time as max_time,
user_group_wild_card as user_wildcard,
query_group_wild_card as query_wildcard
from stv_wlm_service_class_config
where service_class > 4;
```

```
name                          | slots | mem | max_time | user_wildcard | query_wildcard
------------------------------+-------+-----+----------+---------------+---------------
Service class for super user |     1 | 535 |        0 | false         | false
Queue 1                       |     5 | 125 |        0 | false         | false
Queue 2                       |     5 | 125 |        0 | false         | false
Queue 3                       |     5 | 125 |        0 | false         | false
Queue 4                       |     5 | 627 |        0 | false         | false
Queue 5                       |     5 | 125 |        0 | true          | true
Default queue                 |     5 | 125 |        0 | false         | false
```

以下查询显示动态 WLM 转换的状态。转换过程中，num_query_tasks 和 target_query_working_mem 会进行更新，直至等于目标值。有关更多信息，请参阅 WLM 动态和静态配置属性。

```
select rtrim(name) as name,
num_query_tasks as slots,
target_num_query_tasks as target_slots,
query_working_mem as memory,
target_query_working_mem as target_memory
from stv_wlm_service_class_config
where num_query_tasks > target_num_query_tasks
or query_working_mem > target_query_working_mem
and service_class > 5;

 name             | slots | target_slots | memory | target_mem
------------------+-------+--------------+--------+------------
 Queue 3          |     5 |           15 |    125 |       375
 Queue 5          |    10 |            5 |    250 |       125
 (2 rows)
```

STV_WLM_SERVICE_CLASS_STATE

包含服务类的当前状态。

STV_WLM_SERVICE_CLASS_STATE 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
service_class	integer	服务类的 ID。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。
num_queued_queries	integer	当前位于队列中的查询数。
num_executing_queries	integer	当前正在执行的查询数。
num_serviced_queries	integer	曾在服务类中的查询数。
num_executed_queries	integer	自 Amazon Redshift 重启以来已运行的查询数。
num_evicted_queries	integer	自 Amazon Redshift 重启以来已移出的查询数。移出查询的原因包括 WLM 超时、QMR 跃点操作、并发扩展集群上的查询失败等。
num_concurrency_scaling_queries	integer	自 Amazon Redshift 重启以来在并发扩展集群上运行的查询数。

示例查询

以下查询显示 5 以上服务类的状态。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。

```
select service_class, num_executing_queries,
num_executed_queries
from stv_wlm_service_class_state
where service_class > 5
order by service_class;
```

```
 service_class | num_executing_queries | num_executed_queries
---------------+-----------------------+----------------------
             6 |                     1 |                  222
             7 |                     0 |                  135
             8 |                     1 |                   39
```

```
(3 rows)
```

STV_XRESTORE_ALTER_QUEUE_STATE

在经典大小调整期间，使用 STV_XRESTORE_ALTER_QUEUE_STATE 监控每个表的迁移进度。当目标节点类型为 RA3 时，这特别适用。有关 RA3 节点经典大小调整的更多信息，请转到经典大小调整。

STV_XRESTORE_ALTER_QUEUE_STATE 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_RESTORE_STATE 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	整数	发起大小调整的用户的 ID。
db_id	整数	数据库的 ID。
schema	char(128)	架构的名称。
table_name	char(128)	表的名称。
tbl	整数	表的 ID。
status	char(64)	表迁移进度的状态。可能值如下所示。 • Waiting：正在等待再分发开始 • Applying：当前正在进行再分发 • Finished：已完成再分发
task_type	整数	表的再分配类型。可能值如下所示。 • 1：KEY • 2：EVEN 有关分发方式的更多信息，请参阅分配方式。

示例查询

以下查询显示了数据库中正在等待大小调整、当前正进行大小调整和已完成大小调整的表的数量。

```
select db_id, status, count(*)
from stv_xrestore_alter_queue_state
group by 1,2 order by 3 desc

db_id  |   status    | count
-------+-------------+------
694325 | Waiting     |   323
694325 | Finished    |    60
694325 | Applying    |     1
```

主集群和并发扩展集群的 SVCS 视图

带有前缀 SVCS 的 SVCS 系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 STL 的表类似，但 STL 表仅提供在主集群上运行的查询的信息。

主题

- SVCS_ALERT_EVENT_LOG
- SVCS_COMPILE
- SVCS_CONCURRENCY_SCALING_USAGE
- SVCS_EXPLAIN
- SVCS_PLAN_INFO
- SVCS_QUERY_SUMMARY
- SVCS_S3LIST
- SVCS_S3LOG
- SVCS_S3PARTITION_SUMMARY
- SVCS_S3QUERY_SUMMARY
- SVCS_STREAM_SEGS
- SVCS_UNLOAD_LOG

SVCS_ALERT_EVENT_LOG

当查询优化程序发现可能指示性能问题的条件时记录警报。该视图派生自 STL_ALERT_EVENT_LOG 系统表，但不显示在并发扩展集群上运行的查询的切片级别。使用 SVCS_ALERT_EVENT_LOG 表标识用于改进查询性能的机会。

一个查询包含多个区段，而且每个区段包含一个或多个步骤。有关更多信息，请参阅查询处理。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 STL 的表类似，但 STL 表仅提供在主集群上运行的查询的信息。

SVCS_ALERT_EVENT_LOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
segment	integer	标识查询区段的数字。
step	integer	运行的查询步骤。
pid	integer	与语句和切片关联的进程 ID。如果同一查询在多个切片上运行，则该查询可能有多个 PID。
xid	bigint	与语句关联的事务 ID。
event	character (1024)	警报事件的描述。
solution	character (1024)	建议的解决方案。

列名称	数据类型	描述
event_time	时间戳	查询开始的时间（采用 UTC 表示）。总时间包括排队和执行时间。秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358** 。

使用说明

您可以使用 SVCS_ALERT_EVENT_LOG 来标识查询中的潜在问题，然后按照优化查询性能中的做法来优化数据库设计并重新编写查询。SVCS_ALERT_EVENT_LOG 将记录以下警报：

- 缺失统计数据

  统计数据缺失。在加载数据或进行大量更改之后运行 ANALYZE 并结合使用 STATUPDATE 和 COPY 操作。有关更多信息，请参阅设计查询的 Amazon Redshift 最佳实践。

- 嵌套循环

  一个嵌套循环通常是一个笛卡尔积。评估您的查询以确保所有参与表均已高效地联接。

- 选择性非常强的筛选条件

  返回的行与扫描的行的比率小于 0.05。扫描的行是 `rows_pre_user_filter` 的值，而返回的行是 STL_SCAN 系统表中的行的值。表示查询正在扫描数量极其大的行来确定结果集。这可能是由于排序键缺失或不正确导致的。有关更多信息，请参阅使用排序键。

- 过多的虚影行

  扫描跳过了相对大量的标记为已删除但未抽空的行或已插入但未提交的行。有关更多信息，请参阅对表执行 vacuum 操作。

- 大型分配

  为进行哈希联接或聚合重新分配了超过 1000000 的行。有关更多信息，请参阅使用数据分配样式。

- 大型广播

  为进行哈希联接广播了超过了 1000000 的行。有关更多信息，请参阅使用数据分配样式。

- 顺序执行

  DS_DIST_ALL_INNER 重新分配方式已在查询计划中指明，此方式强制实施序列执行，因为整个内部表已重新分配到单个节点。有关更多信息，请参阅使用数据分配样式。

示例查询

以下查询显示了四种查询的警报事件。

```
SELECT query, substring(event,0,25) as event,
substring(solution,0,25) as solution,
trim(event_time) as event_time from svcs_alert_event_log order by query;

 query |             event             |          solution            |     event_time

-------+-------------------------------+------------------------------
+---------------------
  6567 | Missing query planner statist | Run the ANALYZE command      | 2014-01-03
 18:20:58
  7450 | Scanned a large number of del | Run the VACUUM command to rec| 2014-01-03
 21:19:31
  8406 | Nested Loop Join in the query | Review the join predicates to| 2014-01-04
 00:34:22
 29512 | Very selective query filter:r | Review the choice of sort key| 2014-01-06
 22:00:00

(4 rows)
```

SVCS_COMPILE

记录查询（包括在扩展集群上运行的查询以及在主集群上运行的查询）的每个查询段的编译时间和位置。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 SVL 的视图类似，但 SVL 视图仅提供在主集群上运行的查询的信息。

SVCS_COMPILE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

有关 SCL_COMPILE 的信息，请参阅 SVL_COMPILE。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户的 ID。
xid	bigint	与语句关联的事务 ID。
pid	integer	与语句关联的进程 ID。
query	integer	查询 ID。您可以使用此 ID 联接各种其他系统表和视图。
segment	integer	要编译的查询段。
locus	integer	运行段的位置。如果在计算节点上，则为 **1**；如果在领导节点上，则为 **2**。
starttime	timestamp	开始编译的时间，采用通用协调时 (UTC) 格式。
endtime	timestamp	编译结束的时间（UTC 时间）。
compile	integer	如果编译得到复用，则值为 **0**，如果编译了段，则值为 **1**。

示例查询

在本示例中，查询 35878 和 35879 运行相同的 SQL 语句。查询 35878 的编译列针对四个查询段均显示 1，表明这些段得到编译。查询 35879 的编译列针对每个段均显示 0，表明这些段无需再次编译。

```
select userid, xid,  pid, query, segment, locus,
datediff(ms, starttime, endtime) as duration, compile
from svcs_compile
where query = 35878 or query = 35879
order by query, segment;

 userid |  xid   |  pid  | query | segment | locus | duration | compile
--------+--------+-------+-------+---------+-------+----------+---------
    100 | 112780 | 23028 | 35878 |       0 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       1 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       2 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       3 |     1 |        0 |       0
```

```
    100 | 112780 | 23028 | 35878 |       4 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       5 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       6 |     1 |     1380 |       1
    100 | 112780 | 23028 | 35878 |       7 |     1 |     1085 |       1
    100 | 112780 | 23028 | 35878 |       8 |     1 |     1197 |       1
    100 | 112780 | 23028 | 35878 |       9 |     2 |      905 |       1
    100 | 112782 | 23028 | 35879 |       0 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       1 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       2 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       3 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       4 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       5 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       6 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       7 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       8 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       9 |     2 |        0 |       0
(20 rows)
```

SVCS_CONCURRENCY_SCALING_USAGE

记录并发扩展的使用期。每个使用期是单个并发扩展集群主动处理查询的连续的持续时间。

SVCS_CONCURRENCY_SCALING_USAGE 对超级用户可见。数据库超级用户可以选择为所有用户打开该视图。

表列

列名称	数据类型	描述
start_time	不带时区的时间戳	使用期开始时间。
end_time	不带时区的时间戳	使用期结束时间。
queries	bigint	在该使用期内运行的查询数。
usage_in_seconds	numeric(27,0)	该使用期的总秒数。

示例查询

要查看特定时间段的使用持续时间（以秒为单位），请输入以下查询：

```
select * from svcs_concurrency_scaling_usage order by start_time;

start_time | end_time | queries | usage_in_seconds
---------------------------+---------------------------+---------+-----------------
2019-02-14 18:43:53.01063 | 2019-02-14 19:16:49.781649 | 48 | 1977
```

SVCS_EXPLAIN

显示已提交供执行的查询的 EXPLAIN 计划。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 STL 的表类似，但 STL 表仅提供在主集群上运行的查询的信息。

SVCS_EXPLAIN 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
nodeid	integer	计划节点标识符，在执行查询时一个节点将映射到一个或多个步骤。
parentid	integer	父节点的计划节点标识符。父节点具有一些数量的子节点。例如，合并联接是针对联接表的扫描的父级。
plannode	character(400)	EXPLAIN 输出中的节点文本。表示在计算节点上执行的计划节点在 EXPLAIN 输出中使用 **XN** 作为前缀。

列名称	数据类型	描述
info	character(400)	计划节点的限定词和筛选条件信息。例如，联接条件和 WHERE 子句限制包括在此列中。

示例查询

考虑聚合联接查询的以下 EXPLAIN 输出：

```
explain select avg(datediff(day, listtime, saletime)) as avgwait
from sales, listing where sales.listid = listing.listid;
                                QUERY PLAN

------------------------------------------------------------------------------
 XN Aggregate  (cost=6350.30..6350.31 rows=1 width=16)
  ->  XN Hash Join DS_DIST_NONE  (cost=47.08..6340.89 rows=3766 width=16)
        Hash Cond: ("outer".listid = "inner".listid)
        -> XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=12)
        -> XN Hash  (cost=37.66..37.66 rows=3766 width=12)
              -> XN Seq Scan on sales  (cost=0.00..37.66 rows=3766 width=12)
(6 rows)
```

如果您运行此查询且其查询 ID 为 10，则您可以使用 SVCS_EXPLAIN 表来查看 EXPLAIN 命令返回的同类信息：

```
select query,nodeid,parentid,substring(plannode from 1 for 30),
substring(info from 1 for 20) from svcs_explain
where query=10 order by 1,2;

query| nodeid |parentid|           substring            |    substring
-----+--------+--------+--------------------------------+------------------
10   |      1 |      0 |XN Aggregate  (cost=6717.61..6  |
10   |      2 |      1 |  -> XN Merge Join DS_DIST_NO| Merge Cond:("outer"
10   |      3 |      2 |       -> XN Seq Scan on lis |
10   |      4 |      2 |       -> XN Seq Scan on sal |
(4 rows)
```

请考虑以下查询：

```
select event.eventid, sum(pricepaid)
```

```
from event, sales
where event.eventid=sales.eventid
group by event.eventid order by 2 desc;

eventid |   sum
--------+----------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
...
```

如果此查询的 ID 为 15，则以下系统表查询返回已完成的计划节点。在这种情况下，将反转节点的顺序以显示执行的实际顺序：

```
select query,nodeid,parentid,substring(plannode from 1 for 56)
from svcs_explain where query=15 order by 1, 2 desc;

query|nodeid|parentid|                              substring
-----+------+--------+-----------------------------------------------------------
15   |    8 |      7 |                                    -> XN Seq Scan on eve
15   |    7 |      5 |                              -> XN Hash(cost=87.98..87.9
15   |    6 |      5 |                              -> XN Seq Scan on sales(cos
15   |    5 |      4 |                         -> XN Hash Join DS_DIST_OUTER(cos
15   |    4 |      3 |                 -> XN HashAggregate(cost=862286577.07..
15   |    3 |      2 |          -> XN Sort(cost=1000862287175.47..10008622871
15   |    2 |      1 |  -> XN Network(cost=1000862287175.47..1000862287197.
15   |    1 |      0 |XN Merge(cost=1000862287175.47..1000862287197.46 rows=87
(8 rows)
```

以下查询检索包含一个窗口函数的任何查询计划的查询 ID：

```
select query, trim(plannode) from svcs_explain
where plannode like '%Window%';

query|                                      btrim
-----+-------------------------------------------------------------------------
26   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
27   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
(2 rows)
```

SVCS_PLAN_INFO

使用 SVCS_PLAN_INFO 表以一组行的形式查看一个查询的 EXPLAIN 输出。这是查看查询计划的一种替代方法。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 STL 的表类似，但 STL 表仅提供在主集群上运行的查询的信息。

SVCS_PLAN_INFO 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
nodeid	integer	计划节点标识符，在执行查询时一个节点将映射到一个或多个步骤。
segment	integer	标识查询区段的数字。
step	integer	标识查询步骤的数字。
locus	integer	步骤运行的位置。如果在计算节点上，则为 0；如果在领导节点上，则为 1。
plannode	integer	计划节点的枚举值。查看以下表可了解 plannode 的枚举。（SVCS_EXPLAIN 中的 PLANNODE 列包含计划节点文本。）
startupcost	double precision	此步骤中返回第一行的估计相对成本。
totalcost	double precision	执行此步骤的估计相对成本。

列名称	数据类型	描述
rows	bigint	将在此步骤中生成的估计行数。
bytes	bigint	将在此步骤中生成的估计字节数。

示例查询

以下示例比较了通过使用 EXPLAIN 命令和通过查询 SVCS_PLAN_INFO 表返回的简单 SELECT 查询的查询计划。

```
explain select * from category;
QUERY PLAN
------------------------------------------------------------
XN Seq Scan on category (cost=0.00..0.11 rows=11 width=49)
(1 row)

select * from category;
catid | catgroup | catname | catdesc
-------+----------+-----------+--------------------------------------------
1 | Sports | MLB | Major League Baseball
3 | Sports | NFL | National Football League
5 | Sports | MLS | Major League Soccer
...

select * from svcs_plan_info where query=256;

query | nodeid | segment | step | locus | plannode | startupcost | totalcost
| rows | bytes
-------+--------+---------+------+-------+----------+-------------+-----------+------
+-------
256 | 1 | 0 | 1 | 0 | 104 | 0 | 0.11 | 11 | 539
256 | 1 | 0 | 0 | 0 | 104 | 0 | 0.11 | 11 | 539
(2 rows)
```

在本示例中，PLANNODE 104 表示 CATEGORY 表的顺序扫描。

```
select distinct eventname from event order by 1;

eventname
----------------------------------------------------------------------
```

```
.38 Special
3 Doors Down
70s Soul Jam
A Bronx Tale
...

explain select distinct eventname from event order by 1;

QUERY PLAN
------------------------------------------------------------------------------
XN Merge (cost=1000000000136.38..1000000000137.82 rows=576 width=17)
Merge Key: eventname
-> XN Network (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Send to leader
-> XN Sort (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Sort Key: eventname
-> XN Unique (cost=0.00..109.98 rows=576 width=17)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=17)
(8 rows)

select * from svcs_plan_info where query=240 order by nodeid desc;

query | nodeid | segment | step | locus | plannode | startupcost |
totalcost | rows | bytes
-------+--------+---------+------+-------+----------+-----------------
+------------------+------+--------
240 | 5 | 0 | 0 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 5 | 0 | 1 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 4 | 0 | 2 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 0 | 3 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 0 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 1 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 3 | 1 | 2 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 3 | 2 | 0 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 2 | 2 | 1 | 0 | 123 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 1 | 3 | 0 | 0 | 122 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
(10 rows)
```

SVCS_QUERY_SUMMARY

使用 SVCS_QUERY_SUMMARY 视图查找有关查询执行的一般信息。

注意，SVCS_QUERY_SUMMARY 中的信息来自对所有节点的汇总。

> **Note**
>
> SVCS_QUERY_SUMMARY 视图只包含有关 Amazon Redshift 完成的查询的信息，不包含其它实用工具和 DDL 命令执行的查询的相关信息。有关 Amazon Redshift（包括 DDL 和实用工具命令）完成的所有语句的完整列表和信息，请查询 SVL_STATEMENTTEXT 视图。
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 SVL 的视图类似，但 SVL 视图仅提供在主集群上运行的查询的信息。

SVCS_QUERY_SUMMARY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

有关 SVL_QUERY_SUMMARY 的信息，请参阅 SVL_QUERY_SUMMARY。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。
query	integer	查询 ID。可用于联接各种其他系统表和视图。
stm	integer	流：查询中的一组并发段。查询拥有一个或多个流。
seg	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。查询段可并行运行。每个段在一个进程中运行。
step	integer	运行的查询步骤。
maxtime	bigint	供步骤运行的最大时长（单位为微秒）。

列名称	数据类型	描述
avgtime	bigint	供步骤运行的平均时长（单位为微秒）。
rows	bigint	查询步骤涉及的数据行数。
bytes	bigint	查询步骤涉及的数据字节数。
rate_row	double precision	每行的查询执行率。
rate_byte	double precision	每字节的查询执行率。
label	text	由查询步骤名称及（如适用）表 ID、表名称组成的步骤标签（例如 scan tbl=100448 name =user）。三位表 ID 通常是指临时表的扫描。如果显示 `tbl=0`，则通常是指常量值的扫描。
is_diskbased	character(1)	此查询步骤是否作为基于磁盘的操作在集群中的任意节点上执行：true（**t**）或 false（**f**）。只有哈希、排序和聚合等特定步骤才能转到磁盘。许多类型的步骤始终在内存中运行。
workmem	bigint	分配到查询步骤的工作内存量（单位为字节）。
is_rrscan	character(1)	如果为 true (**t**)，则表示对步骤使用了限制范围的扫描。默认为 false (**f**)。
is_delayed_scan	character(1)	如果为 true (**t**)，则表示对步骤使用了延迟扫描。默认为 false (**f**)。
rows_pre_filter	bigint	对于永久表的扫描，这是在筛选标记为删除的行（虚影行）之前发出的行的总数。

示例查询

查看查询步骤的处理信息

下面的查询显示查询 87 的每个步骤的基本处理信息：

```
select query, stm, seg, step, rows, bytes
```

```
from svcs_query_summary
where query = 87
order by query, seg, step;
```

此查询检索有关查询 87 的处理信息，如下面的示例输出所示：

```
 query | stm | seg | step |  rows  |  bytes
-------+-----+-----+------+--------+---------
87     |   0 |   0 |    0 |     90 |    1890
87     |   0 |   0 |    2 |     90 |     360
87     |   0 |   1 |    0 |     90 |     360
87     |   0 |   1 |    2 |     90 |    1440
87     |   1 |   2 |    0 | 210494 | 4209880
87     |   1 |   2 |    3 |  89500 |       0
87     |   1 |   2 |    6 |      4 |      96
87     |   2 |   3 |    0 |      4 |      96
87     |   2 |   3 |    1 |      4 |      96
87     |   2 |   4 |    0 |      4 |      96
87     |   2 |   4 |    1 |      1 |      24
87     |   3 |   5 |    0 |      1 |      24
87     |   3 |   5 |    4 |      0 |       0
(13 rows)
```

确定查询步骤是否溢出到磁盘

下面的查询显示查询 ID 为 1025 的查询是否有任何步骤（有关如何获取查询 ID 的信息，请参阅 SVL_QLOG 视图）溢出到磁盘，还是查询完全在内存中运行：

```
select query, step, rows, workmem, label, is_diskbased
from svcs_query_summary
where query = 1025
order by workmem desc;
```

此查询返回以下示例输出：

```
query| step|  rows  |  workmem   |  label        | is_diskbased
-----+-----+--------+------------+---------------+--------------
1025 |  0  |16000000|  141557760 |scan tbl=9     | f
1025 |  2  |16000000|  135266304 |hash tbl=142   | t
1025 |  0  |16000000|  128974848 |scan tbl=116536| f
1025 |  2  |16000000|  122683392 |dist           | f
(4 rows)
```

通过扫描 IS_DISKBASED 的值，您能够了解哪些查询步骤溢出到磁盘。对于查询 1025，哈希步骤在磁盘上运行。可能在磁盘上执行的步骤包括哈希、聚合和排序步骤。要仅查看基于磁盘的查询步骤，请在上面的示例中向 SQL 语句添加 **`and is_diskbased = 't'`** 子句。

SVCS_S3LIST

可以使用 SVCS_S3LIST 视图获取有关段级别的 Amazon Redshift Spectrum 查询的详细信息。一个段可以执行一个外部表扫描。该视图派生自 SVL_S3LIST 系统视图，但不显示在并发扩展集群上运行的查询的切片级别。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 SVL 的视图类似，但 SVL 视图仅提供在主集群上运行的查询的信息。

SVCS_S3LIST 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

有关 SVL_S3LIST 的信息，请参阅 SVL_S3LIST。

表列

列名称	数据类型	描述
query	integer	查询 ID。
segment	integer	段编号。查询由多个段组成。
node	integer	节点编号。
eventtime	timestamp	记录事件的 UTC 时间。
桶	char(256)	Amazon S3 桶名称。
前缀	char(256)	Amazon S3 桶位置的前缀。
recursive	char(1)	是否具有子文件夹的递归扫描。
retrieved_files	integer	列出的文件数。

列名称	数据类型	描述
max_file_size	bigint	列出的文件中的最大文件大小。
avg_file_size	double precision	列出的文件中的平均文件大小。
generated_splits	integer	文件拆分数。
avg_split_length	double precision	文件拆分的平均长度，以字节为单位。
duration	bigint	文件列表的持续时间，以微秒为单位。

示例查询

以下示例查询上次执行的查询的 SVCS_S3LIST。

```
select *
from svcs_s3list
where query = pg_last_query_id()
order by query,segment;
```

SVCS_S3LOG

可以使用 SVCS_S3LOG 视图获取有关段级别的 Redshift Spectrum 查询的故障排除详细信息。一个段可以执行一个外部表扫描。该视图派生自 SVL_S3LOG 系统视图，但不显示在并发扩展集群上运行的查询的切片级别。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 SVL 的视图类似，但 SVL 视图仅提供在主集群上运行的查询的信息。

SVCS_S3LOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

有关 SVL_S3LOG 的信息，请参阅 SVL_S3LOG。

表列

列名称	数据类型	描述
pid	integer	进程 ID。
query	integer	查询 ID。
segment	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。
step	integer	运行的查询步骤。
node	integer	节点编号。
eventtime	timestamp	记录事件的 UTC 时间。
message	char(512)	日志条目的消息。

示例查询

以下示例查询上次运行的查询的 SVCS_S3LOG。

```
select *
from svcs_s3log
where query = pg_last_query_id()
order by query,segment;
```

SVCS_S3PARTITION_SUMMARY

可以使用 SVCS_S3PARTITION_SUMMARY 视图获取段级别的 Redshift Spectrum 查询分区处理摘要。一个段可以执行一个外部表扫描。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 SVL 的视图类似，但 SVL 视图仅提供在主集群上运行的查询的信息。

SVCS_S3PARTITION_SUMMARY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

有关 SVL_S3PARTITION 的信息，请参阅 SVL_S3PARTITION。

表列

列名称	数据类型	描述
query	integer	查询 ID。您可以使用此值联接各种其他系统表和视图。
segment	integer	段编号。查询由多个段组成。
分配	char(1)	节点中的分区分配类型。
min_starttime	timestamp	开始执行分区处理的 UTC 时间。
max_endtime	timestamp	完成分区处理的 UTC 时间。
min_duration	bigint	节点在该查询中使用的最小分区处理时间（以微秒为单位）。
max_duration	bigint	节点在该查询中使用的最大分区处理时间（以微秒为单位）。
avg_duration	bigint	节点在该查询中使用的平均分区处理时间（以微秒为单位）。
total_partitions	integer	外部表中的总分区数。
qualified_partitions	integer	符合条件的总分区数。
min_assigned_partitions	integer	在一个节点上分配的最小分区数。

列名称	数据类型	描述
max_assigned_partitions	integer	在一个节点上分配的最大分区数。
avg_assigned_partitions	bigint	在一个节点上分配的平均分区数。

示例查询

以下示例获取上次执行的查询的分区扫描详细信息。

```
select query, segment, assignment, min_starttime, max_endtime, min_duration,
 avg_duration
from svcs_s3partition_summary
where query = pg_last_query_id()
order by query,segment;
```

SVCS_S3QUERY_SUMMARY

可以使用 SVCS_S3QUERY_SUMMARY 视图获取已在系统上运行的所有 Redshift Spectrum 查询（S3 查询）的摘要。一个段可以执行一个外部表扫描。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 SVL 的视图类似，但 SVL 视图仅提供在主集群上运行的查询的信息。

SVCS_S3QUERY_SUMMARY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

有关 SVL_S3QUERY 的信息，请参阅 SVL_S3QUERY。

表列

列名称	数据类型	描述
userid	integer	已生成给定条目的用户的 ID。
query	integer	查询 ID。您可以使用此值联接各种其他系统表和视图。
xid	bigint	事务 ID。
pid	integer	进程 ID。
segment	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。
step	integer	运行的查询步骤。
starttime	timestamp	开始运行该段中的 Redshift Spectrum 查询的 UTC 时间。一个段可以具有一个外部表扫描。
endtime	timestamp	完成该段中的 Redshift Spectrum 查询的 UTC 时间。一个段可以具有一个外部表扫描。
elapsed	integer	运行该段中的 Redshift Spectrum 查询所需的时间长度（以微秒为单位）。
aborted	integer	如果查询已由系统停止或已由用户取消，则此列包含 **1**。如果查询运行完成，则此列包含 **0**。
external_table_name	char(136)	外部表扫描的表的外部名称的名称内部格式。
file_format	character(16)	外部表数据的文件格式。
is_partitioned	char(1)	如果为 true (**t**)，则此列值表示外部表已进行分区。
is_rrscan	char(1)	如果为 true (**t**)，则此列值表示已应用限制范围的扫描。
is_nested	varchar(1)	如果为 true (**t**)，该列值表示访问嵌套的列数据类型。

列名称	数据类型	描述
s3_scanned_rows	bigint	已从 Amazon S3 扫描并发送到 Redshift Spectrum 层的行数量。
s3_scanned_bytes	bigint	已从 Amazon S3 扫描并发送到 Redshift Spectrum 层的基于压缩数据的字节的数量。
s3query_returned_rows	bigint	已从 Redshift Spectrum 层返回到集群的行的数量。
s3query_returned_bytes	bigint	已从 Redshift Spectrum 层返回到集群的字节的数量。返回到 Amazon Redshift 的大量数据可能会影响系统性能。
files	integer	针对此 Redshift Spectrum 查询已处理的文件的数量。文件数量少会限制并行处理的优势。
files_max	integer	一个切片上处理的最大文件数。
files_avg	integer	一个切片上处理的平均文件数。
splits	bigint	为此分段处理的拆分数。在此切片上处理的拆分数。对于大型可拆分数据文件（例如，大于 512 MB 左右的数据文件），Redshift Spectrum 会尝试将文件拆分为多个 S3 请求以便进行并行处理。
splits_max	integer	在此切片上处理的最大拆分数。
splits_avg	bigint	在此切片上处理的平均拆分数。
total_split_size	bigint	处理的所有拆分的总大小。
max_split_size	bigint	处理的最大拆分大小（以字节为单位）。
avg_split_size	bigint	处理的平均拆分大小（以字节为单位）。

列名称	数据类型	描述
total_retries	bigint	该段中的 Redshift Spectrum 查询的总重试次数。
max_retries	integer	一个处理的文件的最大重试次数。
max_request_duration	bigint	单个文件请求的最长持续时间（以微秒为单位）。长时间运行的查询可能表示瓶颈。
avg_request_duration	bigint	文件请求的平均持续时间（以微秒为单位）。
max_request_parallelism	integer	该 Redshift Spectrum 查询的一个切片中的最大并行请求数。
avg_request_parallelism	double precision	该 Redshift Spectrum 查询的一个切片中的平均并行请求数。
total_slowdown_count	bigint	在外部表扫描期间出现减速错误的总 Amazon S3 请求数。
max_slowdown_count	integer	在一个切片上的外部表扫描期间出现减速错误的最大 Amazon S3 请求数。

示例查询

以下示例获取上次运行的查询的扫描步骤详细信息。

```
select query, segment, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
from svcs_s3query_summary
where query = pg_last_query_id()
order by query,segment;
```

```
query | segment | elapsed | s3_scanned_rows | s3_scanned_bytes | s3query_returned_rows
 | s3query_returned_bytes | files
------+---------+---------+-----------------+------------------+----------------------
+-------------------------+------
 4587 |       2 |   67811 |               0 |                0 |                    0
 |                       0 |     0
 4587 |       2 |  591568 |          172462 |         11260097 |                 8513
 |                  170260 |     1
 4587 |       2 |  216849 |               0 |                0 |                    0
 |                       0 |     0
 4587 |       2 |  216671 |               0 |                0 |                    0
 |                       0 |     0
```

SVCS_STREAM_SEGS

列出流与并发分段之间的关系。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 STL 的表类似，但 STL 表仅提供在主集群上运行的查询的信息。

SVCS_STREAM_SEGS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
stream	integer	查询的并发分段集。
segment	integer	标识查询区段的数字。

示例查询

要查看最近查询的流与并发分段之间的关系，请键入以下查询：

```
select *
from svcs_stream_segs
where query = pg_last_query_id();

 query | stream | segment
-------+--------+---------
    10 |      1 |       2
    10 |      0 |       0
    10 |      2 |       4
    10 |      1 |       3
    10 |      0 |       1
(5 rows)
```

SVCS_UNLOAD_LOG

可以使用 SVCS_UNLOAD_LOG 获取 UNLOAD 操作的详细信息。

SVCS_UNLOAD_LOG 为 UNLOAD 语句创建的每个文件记录一行。例如，如果 UNLOAD 创建 12 个文件，则 SVCS_UNLOAD_LOG 包含 12 个相应的行。该视图派生自 STL_UNLOAD_LOG 系统表，但不显示在并发扩展集群上运行的查询的切片级别。

> **Note**
>
> 带有前缀 SVCS 的系统视图提供了有关主集群和并发扩展集群上的查询的详细信息。这些视图与带有前缀 STL 的表类似，但 STL 表仅提供在主集群上运行的查询的信息。

SVCS_UNLOAD_LOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户的 ID。

列名称	数据类型	描述
query	integer	查询 ID。
pid	integer	与查询语句关联的进程 ID。
path	character(1280)	文件的完整 Amazon S3 对象路径。
start_time	timestamp	UNLOAD 操作的开始时间。
end_time	timestamp	UNLOAD 操作的结束时间。
line_count	bigint	卸载到文件的行数。
transfer_size	bigint	传输的字节数。
file_format	character(10)	卸载的文件格式。

示例查询

要获得已由 UNLOAD 命令写入到 Amazon S3 的文件的列表，您可以在 UNLOAD 完成后调用 Amazon S3 列表操作；但是，根据您发出调用的速度，列表可能不完整，因为 Amazon S3 列表操作最终是一致的。要立即获得完整的权威列表，请查询 SVCS_UNLOAD_LOG。

以下查询返回上次完成的查询的 UNLOAD 创建的文件的路径名：

```
select query, substring(path,0,40) as path
from svcs_unload_log
where query = pg_last_query_id()
order by path;
```

此命令返回以下示例输出：

```
 query |             path
------+---------------------------------
  2320 | s3://my-bucket/venue0000_part_00
  2320 | s3://my-bucket/venue0001_part_00
  2320 | s3://my-bucket/venue0002_part_00
  2320 | s3://my-bucket/venue0003_part_00
```

```
(4 rows)
```

主集群的 SVL 视图

SVL 视图是 Amazon Redshift 中的系统视图，其中包含对 STL 表和日志的引用以提供更多详细信息。

这些视图提供对在这些表中找到的常用查询数据的更快速、更便捷的访问。

> **Note**
>
> SVL_QUERY_SUMMARY 视图只包含有关 Amazon Redshift 运行的查询的信息，不包含其它实用工具和 DDL 命令执行的查询的相关信息。有关 Amazon Redshift（包括 DDL 和实用工具命令）运行的所有语句的完整列表和信息，请查询 SVL_STATEMENTTEXT 视图。

主题

- SVL_AUTO_WORKER_ACTION
- SVL_COMPILE
- SVL_DATASHARE_CHANGE_LOG
- SVL_DATASHARE_CROSS_REGION_USAGE
- SVL_DATASHARE_USAGE_CONSUMER
- SVL_DATASHARE_USAGE_PRODUCER
- SVL_FEDERATED_QUERY
- SVL_MULTI_STATEMENT_VIOLATIONS
- SVL_MV_REFRESH_STATUS
- SVL_QERROR
- SVL_QLOG
- SVL_QUERY_METRICS
- SVL_QUERY_METRICS_SUMMARY
- SVL_QUERY_QUEUE_INFO
- SVL_QUERY_REPORT
- SVL_QUERY_SUMMARY
- SVL_RESTORE_ALTER_TABLE_PROGRESS

- SVL_S3LIST
- SVL_S3LOG
- SVL_S3PARTITION
- SVL_S3PARTITION_SUMMARY
- SVL_S3QUERY
- SVL_S3QUERY_SUMMARY
- SVL_S3RETRIES
- SVL_SPATIAL_SIMPLIFY
- SVL_SPECTRUM_SCAN_ERROR
- SVL_STATEMENTTEXT
- SVL_STORED_PROC_CALL
- SVL_STORED_PROC_MESSAGES
- SVL_TERMINATE
- SVL_UDF_LOG
- SVL_USER_INFO
- SVL_VACUUM_PERCENTAGE

SVL_AUTO_WORKER_ACTION

记录 Amazon Redshift 对自动优化定义的表执行的自动操作。

SVL_AUTO_WORKER_ACTION 对所有用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
table_id	integer	表标识符。
type	character (32)	建议的类型。可能的值包括 distkey 和 sortkey。

列名称	数据类型	描述
status	character (128)	建议的完成状态。可能的值包括 Start、Complete、Skipped、Abort、Checkpoint 和 Failed。
eventtime	timestamp	`status` 列的时间戳。
sequence	integer	截断的 `previous_state` 值的序列号。当一个 `previous_state` 包含 200 多个字符时，将为该值记录额外的行。序列 0 是第一行，1 是第二行，依此类推。
previous_state	character (200)	应用建议之前表的上一个分配方式和排序键。该值被截断为 200 个字符的增量。

`status` 列的一些值示例如下所示：

- 已跳过：找不到表。
- 已跳过：建议为空。
- 已跳过：应用排序键建议已禁用。
- 已跳过：重试次数超出表的最大限制。
- 已跳过：表列已更改。
- 中止：此表不是 AUTO。
- 中止：此表最近已转换。
- 中止：此表超出表大小阈值。
- 中止：此表已经是推荐的样式。
- 检查点：进度 *21.9963*%。

示例查询

在以下示例中，结果中的行显示了由 Amazon Redshift 执行的操作。

```
select table_id, type, status, eventtime, sequence, previous_state
from SVL_AUTO_WORKER_ACTION;
```

```
 table_id |  type   |                        status                        |
 eventtime          | sequence | previous_state
----------+---------+------------------------------------------------------
+---------------------------+----------+---------------
   118082 | sortkey | Start                                                | 2020-08-22
 19:42:20.727049 | 0        |
   118078 | sortkey | Start                                                | 2020-08-22
 19:43:54.728819 | 0        |
   118082 | sortkey | Start                                                | 2020-08-22
 19:42:52.690264 | 0        |
   118072 | sortkey | Start                                                | 2020-08-22
 19:44:14.793572 | 0        |
   118082 | sortkey | Failed                                               | 2020-08-22
 19:42:20.728917 | 0        |
   118078 | sortkey | Complete                                             | 2020-08-22
 19:43:54.792705 | 0        | SORTKEY: None;
   118086 | sortkey | Complete                                             | 2020-08-22
 19:42:00.72635  | 0        | SORTKEY: None;
   118082 | sortkey | Complete                                             | 2020-08-22
 19:43:34.728144 | 0        | SORTKEY: None;
   118072 | sortkey | Skipped:Retry exceeds the maximum limit for a table. | 2020-08-22
 19:44:46.706155 | 0        |
   118086 | sortkey | Start                                                | 2020-08-22
 19:42:00.685255 | 0        |
   118082 | sortkey | Start                                                | 2020-08-22
 19:43:34.69531  | 0        |
   118072 | sortkey | Start                                                | 2020-08-22
 19:44:46.703331 | 0        |
   118082 | sortkey | Checkpoint: progress 14.755079%                      | 2020-08-22
 19:42:52.692828 | 0        |
   118072 | sortkey | Failed                                               | 2020-08-22
 19:44:14.796071 | 0        |
   116723 | sortkey | Abort:This table is not AUTO.                        | 2020-10-28
 05:12:58.479233 | 0        |
   110203 | distkey | Abort:This table is not AUTO.                        | 2020-10-28
 05:45:54.67259  | 0        |
```

SVL_COMPILE

记录查询的每个查询段的编译时间和位置。

SVL_COMPILE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> SVL_COMPILE 仅包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_QUERY_HISTORY。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

有关 SVCS_COMPILE 的信息，请参阅 SVCS_COMPILE。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
xid	bigint	与语句关联的事务 ID。
pid	integer	与语句关联的进程 ID。
query	integer	查询 ID。可用于联接各种其他系统表和视图。
segment	integer	要编译的查询段。
locus	integer	运行段的位置。如果在计算节点上，则为 **1**；如果在领导节点上，则为 **2**。
starttime	timestamp	开始编译的时间（UTC 时间）。
endtime	timestamp	编译结束的时间（UTC 时间）。
compile	integer	如果编译得到复用，则为 **0**；如果编译了段，则为 **1**。

示例查询

在本示例中，查询 35878 和 35879 运行相同的 SQL 语句。查询 35878 的编译列针对四个查询段均显示 1，表明这些段得到编译。查询 35879 的编译列针对每个段均显示 0，表明这些段无需再次编译。

```
select userid, xid,  pid, query, segment, locus,
```

```
datediff(ms, starttime, endtime) as duration, compile
from svl_compile
where query = 35878 or query = 35879
order by query, segment;

 userid |  xid   |  pid  | query | segment | locus | duration | compile
--------+--------+-------+-------+---------+-------+----------+---------
    100 | 112780 | 23028 | 35878 |       0 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       1 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       2 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       3 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       4 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       5 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       6 |     1 |     1380 |       1
    100 | 112780 | 23028 | 35878 |       7 |     1 |     1085 |       1
    100 | 112780 | 23028 | 35878 |       8 |     1 |     1197 |       1
    100 | 112780 | 23028 | 35878 |       9 |     2 |      905 |       1
    100 | 112782 | 23028 | 35879 |       0 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       1 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       2 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       3 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       4 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       5 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       6 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       7 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       8 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       9 |     2 |        0 |       0
(20 rows)
```

SVL_DATASHARE_CHANGE_LOG

记录用于跟踪创建器和使用者集群上的数据共享更改的综合视图。

SVL_DATASHARE_CHANGE_LOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_DATASHARE_CHANGE_LOG 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	执行操作的用户的 ID。
username	varchar(128)	执行操作的用户的名称。
pid	integer	进程的 ID。
xid	bigint	事务的 ID。
share_id	integer	受影响的数据共享的 ID。
share_name	varchar(128)	数据共享的名称。
source_database_id	integer	数据共享所属的数据库的 ID。
source_database_name	varchar(128)	数据共享所属的数据库的名称。
consumer_database_id	integer	从数据共享中导入的数据库的 ID。
consumer_database_name	varchar(128)	从数据共享导入的数据库的名称。
arn	varchar(192)	支持所导入数据库的资源的 ARN。
recordtime	timestamp	操作的时间戳。
action	varchar(128)	正在运行的操作。可能的值有：CREATE DATASHARE、DROP DATASHARE、GRANT ALTER、REVOKE ALTER、GRANT

列名称	数据类型	描述
		SHARE、REVOKE SHARE、ALTER ADD、ALTER REMOVE、ALTER SET、GRANT USAGE、REVOKE USAGE、CREATE DATABASE、GRANT 或共享数据库上的 REVOKE USAGE、DROP SHARED DATABASE、ALTER SHARED DATABASE。
status	integer	操作的状态。可能的值为 SUCCESS 和 ERROR-ERROR CODE。
share_object_type	varchar(64)	在数据共享中添加或删除的数据库对象的类型。可能的值包括 schema、表、列、函数和视图。这是创建器集群的字段。
share_object_id	integer	在数据共享中添加或删除的数据库对象的 ID。这是创建器集群的字段。
share_object_name	varchar(128)	在数据共享中添加或删除的数据库对象的名称。这是创建器集群的字段。
target_user_type	varchar(16)	被授予权限的用户或组的类型。这是创建器集群和使用者集群的字段。
target_userid	integer	被授予权限的用户或组的 ID。这是创建器集群和使用者集群的字段。
target_username	varchar(128)	被授予权限的用户或组的名称。这是创建器集群和使用者集群的字段。
consumer_account	varchar(16)	数据使用者的账户 ID。这是创建器集群的字段。
consumer_namespace	varchar(64)	数据使用者账户的命名空间。这是创建器集群的字段。
producer_account	varchar(16)	数据共享所属的创建者账户的账户 ID。这是使用者集群的字段。
producer_namespace	varchar(64)	数据共享所属的创建者账户的命名空间。这是使用者集群的字段。

列名称	数据类型	描述
attribute_name	varchar(64)	数据共享或共享数据库的属性的名称。
attribute_value	varchar(128)	数据共享或共享数据库的属性值。
message	varchar(512)	操作失败时出现的错误消息。

示例查询

以下示例显示了 SVL_DATASHARE_CHANGE_LOG 视图。

```
SELECT DISTINCT action
FROM svl_datashare_change_log
WHERE share_object_name LIKE 'tickit%';

         action
 ------------------------
  "ALTER DATASHARE ADD"
```

SVL_DATASHARE_CROSS_REGION_USAGE

使用 SVL_DATASHARE_CROSS_REGION_USAGE 视图，可以获取跨区域数据共享查询所导致的跨区域数据传输使用情况的摘要。SVL_DATASHARE_CROSS_REGION_USAGE 汇总了段级别的详细信息。

SVL_DATASHARE_CROSS_REGION_USAGE 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_DATASHARE_CROSS_REGION_USAGE 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
query	整数	查询的 ID。使用此值可以联接其他系统表和视图。
segment	bigint	分段的编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。
start_time	time	数据传输开始的时间，使用 UTC。
end_time	time	数据传输结束的时间，使用 UTC。
transferred_data	bigint	从生产者区域向使用者区域传输的数据字节数。
source_region	char(25)	查询从中传输数据的生产者区域。
recordtime	timestamp	记录操作的时间。

示例查询

以下示例显示了 SVL_DATASHARE_CROSS_REGION_USAGE 视图。

```
SELECT query, segment, transferred_data, source_region
from svl_datashare_cross_region_usage
where query = pg_last_query_id()
order by query,segment;

  query | segment | transferred_data | source_region
--------+---------+------------------+---------------
 200048 |       2 |          4194304 |    us-west-1
 200048 |       2 |          4194304 |    us-east-2
```

SVL_DATASHARE_USAGE_CONSUMER

记录数据共享的活动和使用情况。此视图仅与使用者集群相关。

SVL_DATASHARE_USAGE_CONSUMER 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_DATASHARE_USAGE_CONSUMER 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	发出请求的用户的 ID。
pid	integer	运行查询的领导节点进程的 ID。
xid	bigint	当前事务的上下文。
request_id	varchar(50)	请求的 API 调用的唯一 ID。
request_type	varchar(25)	向创建器集群发出的请求的类型。
transaction_uid	varchar(50)	事务的唯一 ID。
recordtime	timestamp	记录操作的时间。
status	integer	请求的 API 调用的状态。
error	varchar(512)	错误消息。

示例查询

以下示例显示了 SVL_DATASHARE_USAGE_CONSUMER 视图。

```
SELECT request_type, status, trim(error) AS error
FROM svl_datashare_usage_consumer
```

```
  request_type   | status | error
-----------------+--------+--------
 "GET RELATION" |   0    |
```

SVL_DATASHARE_USAGE_PRODUCER

记录数据共享的活动和使用情况。此视图仅与创建器集群有关。

SVL_DATASHARE_USAGE_PRODUCER 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_DATASHARE_USAGE_PRODUCER 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
share_id	integer	数据共享的对象 ID (OID)。
share_name	varchar(128)	数据共享的名称。
request_id	varchar(50)	请求的 API 调用的唯一 ID。
request_type	varchar(25)	向创建器集群发出的请求的类型。
object_type	varchar(64)	从数据共享中共享的对象的类型。可能的值包括 schema、表、列、函数和视图。
object_oid	integer	从数据共享中共享的对象的 ID。
object_name	varchar(128)	从数据共享中共享的对象的名称。

列名称	数据类型	描述
consumer_account	varchar(16)	数据共享所共享到的使用者账户的账户。
consumer_namespace	varchar(64)	数据共享所共享到的使用者账户的命名空间。
consumer_transaction_uid	varchar(50)	使用者集群上语句的唯一事务 ID。
recordtime	timestamp	记录操作的时间。
status	integer	数据共享的状态。
error	varchar(512)	错误消息。
consumer_region	char(64)	使用者集群所在的区域。

示例查询

以下示例显示了 SVL_DATASHARE_USAGE_PRODUCER 视图。

```
SELECT DISTINCT request_type
FROM svl_datashare_usage_producer
WHERE object_name LIKE 'tickit%';

   request_type
 ------------------
   "GET RELATION"
```

SVL_FEDERATED_QUERY

使用 SVL_FEDERATED_QUERY 视图查看有关联合查询调用的信息。

SVL_FEDERATED_QUERY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_EXTERNAL_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	运行查询的用户的 ID。
xid	bigint	事务 ID。
pid	integer	运行查询的领导节点进程的 ID。
query	integer	联合调用的查询 ID。
sourcetype	character (32)	联合调用源类型，例如“PG”。
recordtime	timestamp	发送查询以进行联合调用的时间。使用 UTC。
querytext	character (4000)	发送到远程 PostgreSQL 引擎以执行的查询字符串。
num_rows	bigint	联合查询返回的行数。
num_bytes	bigint	联合查询返回的字节数。
duration	bigint	从游标调用中获取行所花费的时间（微秒）。这是运行联合查询以及获取结果所花费的时间。

示例查询

要显示有关联合查询调用的信息，请运行以下查询。

```
select query, trim(sourcetype) as type, recordtime, trim(querytext) as "PG Subquery"
 from svl_federated_query where query = 4292;

 query | type |         recordtime         |                          pg subquery
-------+------+----------------------------
+-------------------------------------------------------------
  4292 | PG   | 2020-03-27 04:29:58.485126 | SELECT "level" FROM functional.employees
 WHERE ("level" >= 6)
(1 row)
```

SVL_MULTI_STATEMENT_VIOLATIONS

使用 SVL_MULTI_STATEMENT_VIOLATIONS 视图获取已在系统上运行的违反了事务数据块限制的所有 SQL 命令的完整记录。

当您在事务数据块或多语句请求内运行 Amazon Redshift 限制的以下 SQL 命令时，会发生违规情况：

- CREATE DATABASE
- DROP DATABASE
- ALTER TABLE APPEND
- CREATE EXTERNAL TABLE
- DROP EXTERNAL TABLE
- RENAME EXTERNAL TABLE
- ALTER EXTERNAL TABLE
- CREATE TABLESPACE。
- DROP TABLESPACE
- CREATE LIBRARY
- DROP LIBRARY
- REBUILDCAT
- INDEXCAT
- REINDEX DATABASE
- VACUUM
- GRANT
- COPY

> **Note**
>
> 如果此视图中有任何条目，请更改相应的应用程序和 SQL 脚本。我们建议更改应用程序代码，以便将这些受限 SQL 命令的使用移到事务数据块之外。如果您需要进一步帮助，请联系 AWS Support。

SVL_MULTI_STATEMENT_VIOLATIONS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	导致违规的用户的 ID。
数据库	character(32)	用户连接到的数据库的名称。
cmdname	character(20)	无法在事务数据块或多语句请求内运行的命令的名称。例如，CREATE DATABASE、DROP DATABASE、ALTER TABLE APPEND、CREATE EXTERNAL TABLE、DROP EXTERNAL TABLE、RENAME EXTERNAL TABLE、ALTER EXTERNAL TABLE、CREATE LIBRARY、DROP LIBRARY、REBUILDCAT、INDEXCAT、REINDEX DATABASE、VACUUM、外部资源上的 GRANT、CLUSTER、COPY、CREATE TABLESPACE 和 DROP TABLESPACE。
xid	bigint	与语句关联的事务 ID。
pid	integer	语句的进程 ID。
label	character(320)	用于运行查询的文件的名称或使用 SET QUERY_GROUP 命令定义的标签。如果查询并非基于文件或未设置 QUERY_GROUP 参数，则此字段为空。

列名称	数据类型	描述
starttime	timestamp	开始执行语句的确切时间，秒的小数部分以 6 位精度表示 – 例如：**2009-06-12 11:29:19.131358**
endtime	timestamp	执行完语句的确切时间，秒的小数部分以 6 位精度表示 – 例如：**2009-06-12 11:29:19.193640**
sequence	integer	当一个语句包含 200 多个字符时，将为该语句记录额外的行。序列 0 是第一行，1 是第二行，依此类推。
type	varchar(10)	SQL 语句的类型：**QUERY**、**DDL** 或 **UTILITY**。
text	character(200)	SQL 文本，以 200 个字符递增。此字段可能包含反斜杠 (\\) 和换行符 (\n) 等特殊字符。

示例查询

以下查询返回多个具有违规情况的语句。

```
select * from svl_multi_statement_violations order by starttime asc;

userid | database | cmdname | xid | pid | label | starttime | endtime | sequence | type
 | text
================================================================================================
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ******** | ******* | 0 | DDL |
 create table c(b int);
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ******** | ******* | 0 | UTILITY |
 create database b;
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ******** | ******* | 0 | UTILITY |
 COMMIT
...
```

SVL_MV_REFRESH_STATUS

SVL_MV_REFRESH_STATUS 视图包含与具体化视图的刷新活动相对应的一行。

有关具体化视图的更多信息，请参阅在 Amazon Redshift 中创建实体化视图。

SVL_MV_REFRESH_STATUS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_MV_REFRESH_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
db_name	char(128)	包含具体化视图的数据库。
userid	bigint	执行刷新的用户的 ID。
schema_name	char(128)	实体化视图的架构。
mv_name	char(128)	具体化视图名称。
xid	bigint	刷新的事务 ID。
starttime	timestamp	刷新的开始时间。
endtime	timestamp	刷新的结束时间。
status	text	刷新的状态。示例值包括： • 刷新以增量方式成功更新了 MV 如果它是一个用于流式传输的实体化视图，那么消息可能具有有关记录数量的附加限定词。这些功能包括： • 串流未返回任何新数据 - 未检索到任何记录。 • 从流中收到的所有记录都被跳过 – 已检索记录，但由于出现错误，所有记录都被跳过。 • 一些流记录被跳过 – 已检索记录，但由于出现错误，一些记录被跳过。 如果没有任何限定条件，则至少检索到一条记录，所有记录在实体化视图中都可用。还有一个可能的限定条件： • 流可能包含更多数据 – 刷新在 Amazon Redshift 确定没有更多记录可供使用之前结

列名称	数据类型	描述
		束。流可能是最新的，但尚未得到 Amazon Redshift 确认。 • 刷新成功地从头开始重新计算了 MV • 刷新以增量方式部分更新了 MV，直至活动的事务 • MV 已更新 • 刷新失败。基表列已重命名 • 刷新失败。基表列类型已更改 • 刷新失败。基表已重命名 • 刷新由于内部错误而失败 • 刷新失败。基表列已被删除 • 刷新失败。MV 的模式已重命名 • 刷新失败。未找到 MV • 由于用户工作负载过多而导致自动刷新中止 • 刷新失败。可序列化的隔离违规
refresh_type	char(32)	刷新类型的定义。示例值包括 Manual 和 Auto。

示例查询

要查看具体化视图的刷新状态，请运行以下查询。

```
select * from svl_mv_refresh_status;
```

此查询返回以下示例输出：

```
 db_name | userid |  schema   |  name   |  xid  |         starttime          |
      endtime           |                        status                        |
 refresh_type
---------+--------+-----------+---------+-------+----------------------------
+----------------------------+-----------------------------------------------------
+-----------------
```

```
dev     |    169 | mv_schema | mv_test |  6640 | 2020-02-14 02:26:53.497935 |
2020-02-14 02:26:53.556156 | Refresh successfully recomputed MV from scratch     |
Manual
dev     |    166 | mv_schema | mv_test |  6517 | 2020-02-14 02:26:39.287438 |
2020-02-14 02:26:39.349539 | Refresh successfully updated MV incrementally       |
Auto
dev     |    162 | mv_schema | mv_test |  6388 | 2020-02-14 02:26:27.863426 |
2020-02-14 02:26:27.918307 | Refresh successfully recomputed MV from scratch     |
Manual
dev     |    161 | mv_schema | mv_test |  6323 | 2020-02-14 02:26:20.020717 |
2020-02-14 02:26:20.080002 | Refresh successfully updated MV incrementally       |
Auto
dev     |    161 | mv_schema | mv_test |  6301 | 2020-02-14 02:26:05.796146 |
2020-02-14 02:26:07.853986 | Refresh successfully recomputed MV from scratch     |
Manual
dev     |    153 | mv_schema | mv_test |  6024 | 2020-02-14 02:25:18.762335 |
2020-02-14 02:25:20.043462 | MV was already updated                              |
Manual
dev     |    143 | mv_schema | mv_test |  5557 | 2020-02-14 02:24:23.100601 |
2020-02-14 02:24:23.100633 | MV was already updated                              |
Manual
dev     |    141 | mv_schema | mv_test |  5447 | 2020-02-14 02:23:54.102837 |
2020-02-14 02:24:00.310166 | Refresh successfully updated MV incrementally       |
Auto
dev     |      1 | mv_schema | mv_test |  5329 | 2020-02-14 02:22:26.328481 |
2020-02-14 02:22:28.369217 | Refresh successfully recomputed MV from scratch     |
Auto
dev     |    138 | mv_schema | mv_test |  5290 | 2020-02-14 02:21:56.885093 |
2020-02-14 02:21:56.885098 | Refresh failed. MV was not found                    |
Manual
```

SVL_QERROR

SVL_QERROR 视图已弃用。

SVL_QLOG

SVL_QLOG 视图包含针对数据库运行的所有查询的日志。

Amazon Redshift 创建 SVL_QLOG 视图作为 STL_QUERY 表中信息的可读子集。使用该表可找出最近运行的查询的查询 ID，或查看完成某项查询用了多长时间。

SVL_QLOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成该条目的用户 ID。
query	integer	查询 ID。您可以使用此 ID 联接各种其他系统表和视图。
xid	bigint	事务 ID。
pid	integer	与查询关联的进程 ID。
starttime	timestamp	开始执行语句的确切时间，秒的小数部分以 6 位精度表示 – 例如：**2009-06-12 11:29:19.131358**
endtime	timestamp	执行完语句的确切时间，秒的小数部分以 6 位精度表示 – 例如：**2009-06-12 11:29:19.193640**
elapsed	bigint	运行查询花费了多长时间（单位为微秒）。
aborted	integer	如果查询已由系统停止或已由用户取消，则此列包含 **1**。如果查询运行完成，则此列包含 **0**。因工作负载管理目的取消（随后重新启动）的查询在此列中的值也为 **1**。
label	character(320)	用于运行查询的文件的名称或使用 SET QUERY_GROUP 命令定义的标签。如果查询并非基于文件或未设置 QUERY_GROUP 参数，则此字段值为 `default`。
substring	character(60)	截断的查询文本。
source_query	integer	如果查询使用了结果缓存，则是缓存结果源自的查询的查询 ID。如果未使用结果缓存，则此字段值为 NULL。
concurrency_scalin	text	关于查询运行在主集群还是并发扩展集群上的描述。

列名称	数据类型	描述
g_status_txt		
from_sp_call	integer	如果从存储过程调用了查询，则为过程调用的查询 ID。如果没有在存储过程中运行查询，则此字段为 NULL。

示例查询

下面的示例返回 `userid = 100` 的用户最近运行的五个数据库查询的查询 ID、执行时间和截断的查询文本。

```
select query, pid, elapsed, substring from svl_qlog
where userid = 100
order by starttime desc
limit 5;

 query  |  pid  | elapsed  |           substring
--------+-------+----------+-----------------------------------------------
 187752 | 18921 | 18465685 | select query, elapsed, substring from svl_...
 204168 |  5117 |    59603 | insert into testtable values (100);
 187561 | 17046 |  1003052 | select * from pg_table_def where tablename...
 187549 | 17046 |  1108584 | select * from STV_WLM_SERVICE_CLASS_CONFIG
 187468 | 17046 |  5670661 | select * from pg_table_def where schemaname...
(5 rows)
```

下面的示例返回已取消的查询 (**aborted=1**) 的 SQL 脚本名称（LABEL 列）和已用时间：

```
select query, elapsed, trim(label) querylabel
from svl_qlog where aborted=1;

 query | elapsed  |       querylabel
-------+----------+-------------------------
    16 |  6935292 | alltickittablesjoin.sql
(1 row)
```

SVL_QUERY_METRICS

SVL_QUERY_METRICS 视图显示已完成的查询的指标。此视图派生自 STL_QUERY_METRICS 系统表。使用此视图中的值以帮助确定用于定义查询监控规则的阈值。有关更多信息，请参阅 WLM 查询监控规则。

SVL_QUERY_METRICS 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	运行生成条目的查询的用户的 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
service_class	integer	WLM 查询队列（服务类）的 ID。查询队列在 WLM 配置中定义。仅对用户定义的队列报告的指标。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。
dimension	varchar(24)	报告的指标所针对的维度。可能的值为 query、segment 和 step。
segment	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。查询段可并行运行。每个段在一个进程中运行。如果段值为 0，则指标段值将汇总到查询级别。
step	integer	已执行步骤的类型的 ID。该步骤类型的描述显示在 `step_label` 列中。
step_label	varchar(30)	已执行的步骤类型。
query_cpu_time	bigint	查询使用的 CPU 时间（以秒为单位）。CPU 时间与查询运行时间不同。

列名称	数据类型	描述
query_blocks_read	bigint	查询读取的 1 MB 数据块的数量。
query_execution_time	bigint	执行查询所用的时间（以秒为单位）。执行时间不包括在队列中等待的时间。请参阅 `query_queue_time` 了解排队时间。
query_cpu_usage_percent	bigint	查询使用的 CPU 容量的百分比。
query_temp_blocks_to_disk	bigint	查询用于写入中间结果的磁盘空间量 (MB)。
segment_execution_time	bigint	执行单个段所用的时间（以秒为单位）。
cpu_skew	numeric(38,2)	任何切片的最大 CPU 使用率与所有切片的平均 CPU 使用率的比率。此指标在段级别进行定义。
io_skew	numeric(38,2)	任何切片的最大数据块读取 (I/O) 与所有切片的平均数据块读取的比率。
scan_row_count	bigint	扫描步骤中行的数量。行计数是在筛选标记为删除的行（虚影行）之前和应用用户定义的查询筛选之前发出的行的总数。
join_row_count	bigint	联接步骤中处理的行数。
nested_loop_join_row_count	bigint	嵌套循环联接中行的数量。
return_row_count	bigint	查询返回的行数。
spectrum_scan_row_count	bigint	Amazon S3 中由 Amazon Redshift Spectrum 查询扫描的行数。

列名称	数据类型	描述
spectrum_scan_size_mb	bigint	Amazon S3 中由 Amazon Redshift Spectrum 查询扫描的数据量 (MB)。
query_queue_time	bigint	查询排队的时间（以秒为单位）。

SVL_QUERY_METRICS_SUMMARY

SVL_QUERY_METRICS_SUMMARY 视图显示已完成的查询的指标的最大值。此视图派生自 STL_QUERY_METRICS 系统表。使用此视图中的值以帮助确定用于定义查询监控规则的阈值。有关 Amazon Redshift 查询监控的规则和指标的更多信息，请参阅WLM 查询监控规则。

SVL_QUERY_METRICS_SUMMARY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	运行生成条目的查询的用户的 ID。
query	integer	查询 ID。查询列可用于连接其他系统表和视图。
service_class	integer	WLM 查询队列（服务类）的 ID。查询队列在 WLM 配置中定义。仅对用户定义的队列报告的指标。有关服务类 ID 的列表，请参阅 WLM 服务类 ID。
query_cpu_time	bigint	查询使用的 CPU 时间（以秒为单位）。CPU 时间与查询运行时间不同。
query_blocks_read	bigint	查询读取的 1 MB 数据块的数量。

列名称	数据类型	描述
query_execution_time	bigint	执行查询所用的时间（以秒为单位）。执行时间不包括在队列中等待的时间。
query_cpu_usage_percent	numeric(38,2)	查询使用的 CPU 容量的百分比。
query_temp_blocks_to_disk	bigint	查询用于写入中间结果的磁盘空间量 (MB)。
segment_execution_time	bigint	执行单个段所用的时间（以秒为单位）。
cpu_skew	numeric(38,2)	任何切片的最大 CPU 使用率与所有切片的平均 CPU 使用率的比率。此指标在段级别进行定义。
io_skew	numeric(38,2)	任何切片的最大数据块读取 (I/O) 与所有切片的平均数据块读取的比率。
scan_row_count	bigint	扫描步骤中行的数量。行计数是在筛选标记为删除的行（虚影行）之前和应用用户定义的查询筛选之前发出的行的总数。
join_row_count	bigint	联接步骤中处理的行数。
nested_loop_join_row_count	bigint	嵌套循环联接中行的数量。
return_row_count	bigint	查询返回的行数。
spectrum_scan_row_count	bigint	Amazon S3 中由 Amazon Redshift Spectrum 查询扫描的行数。
spectrum_scan_size_mb	bigint	Amazon S3 中由 Amazon Redshift Spectrum 查询扫描的数据量 (MB)。
query_queue_time	bigint	查询排队的时间（以秒为单位）。

SVL_QUERY_QUEUE_INFO

总结查询在工作负载管理 (WLM) 查询队列或提交队列中花费时间的详细信息。

SVL_QUERY_QUEUE_INFO 视图筛选系统执行的查询并且只显示用户执行的查询。

SVL_QUERY_QUEUE_INFO 视图总结 STL_QUERY、STL_WLM_QUERY 和 STL_COMMIT_STATS 系统表的信息。

SVL_QUERY_QUEUE_INFO 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
数据库	text	在发起查询时用户连接到的数据库的名称。
query	integer	查询 ID。
xid	bigint	事务 ID。
userid	integer	生成该查询的用户的 ID。
querytxt	text	查询文本的前 100 个字符。
queue_start_time	timestamp	查询进入 WLM 队列的时间（UTC 时间）。
exec_start_time	timestamp	开始执行查询的时间（UTC 时间）。
service_class	integer	服务类的 ID。服务类在 WLM 配置文件中进行定义。
slots	integer	WLM 查询槽位数。
queue_elapsed	bigint	查询在 WLM 队列中等待所花费的时间（单位为秒）。
exec_elapsed	bigint	执行查询所花费的时间（单位为秒）。

列名称	数据类型	描述
wlm_total_elapsed	bigint	查询在 WLM 队列中花费的时间 (queue_elapsed) 加上执行该查询所花费的时间 (exec_elapsed)。
commit_queue_elapsed	bigint	查询在提交队列中等待所花费的时间（单位为秒）。
commit_exec_time	bigint	查询在提交操作中所花费的时间（单位为秒）。
service_class_name	character(64)	服务类的名称。

示例查询

下面的示例显示查询在 WLM 队列中所花费的时间。

```
select query, service_class, queue_elapsed, exec_elapsed, wlm_total_elapsed
from svl_query_queue_info
where wlm_total_elapsed > 0;

  query  | service_class | queue_elapsed | exec_elapsed | wlm_total_elapsed
---------+---------------+---------------+--------------+-------------------
 2742669 |             6 |             2 |          916 |               918
 2742668 |             6 |             4 |          197 |               201
(2 rows)
```

SVL_QUERY_REPORT

Amazon Redshift 通过对一系列 Amazon Redshift STL 系统表执行 UNION 来创建 SVL_QUERY_REPORT 视图，以提供有关所完成的查询步骤的信息。

此视图按切片和按步骤细分所完成的查询的相关信息，这有助于诊断 Amazon Redshift 集群中的节点和切片问题。

SVL_QUERY_REPORT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。
query	integer	查询 ID。可用于联接各种其他系统表和视图。
slice	integer	运行该步骤的数据切片。
segment	integer	段编号。 一个查询包含多个区段，而且每个区段包含一个或多个步骤。查询段可并行运行。每个段在一个进程中运行。
step	integer	已完成的查询步骤。
start_time	timestamp	开始执行段的确切时间（采用 UTC 表示），秒的小数部分以 6 位精度表示。例如：**2012-12-12 11:29:19.131358**
end_time	timestamp	段执行完成的确切时间（采用 UTC 表示），秒的小数部分以 6 位精度表示。例如：**2012-12-12 11:29:19.131467**
elapsed_time	bigint	运行段所花费的时间（微秒）。
rows	bigint	步骤产生的（每切片）行数。该数字代表执行步骤所产生的切片行数，不是步骤接收或处理的行数。换句话说，这是在该步骤中存留并传递到下一个步骤的行数。
bytes	bigint	步骤（每个切片）产生的字节数。
label	char(256)	由查询步骤名称和适用的表 ID 和表名组成的步骤标签（例如 `scan tbl=100448 name =user` ）。三位表 ID 通常是指临时表的扫描。如果显示 `tbl=0`，则通常是指常量值的扫描。

列名称	数据类型	描述
is_diskbased	character (1)	此查询步骤是否已作为基于磁盘的操作执行：true（**t**）或 false（**f**）。只有哈希、排序和聚合等特定步骤才能转到磁盘。许多类型的步骤始终在内存中执行。
workmem	bigint	分配到查询步骤的工作内存量（单位为字节）。此值是分配以供在执行期间使用的 query_working_mem 阈值，不是实际使用的内存量
is_rrscan	character (1)	如果为 true (**t**)，则表示对步骤使用了限制范围的扫描。
is_delayed_scan	character (1)	如果为 true (**t**)，则表示对步骤使用了延迟扫描。
rows_pre_filter	bigint	对于永久表的扫描，这是在筛选标记为删除的行（虚影行）之前和应用用户定义的查询筛选条件之前发出的行的总数。

示例查询

下面的查询演示了针对查询 ID 为 279 的查询返回的行的数据偏斜。使用此查询可以确定数据库数据是否在数据仓库集群中的切片上均匀分配：

```
select query, segment, step, max(rows), min(rows),
case when sum(rows) > 0
then ((cast(max(rows) -min(rows) as float)*count(rows))/sum(rows))
else 0 end
from svl_query_report
where query = 279
group by query, segment, step
order by segment, step;
```

此查询应返回类似以下示例输出的数据：

```
query | segment | step |   max    |   min    |         case
------+---------+------+----------+----------+----------------------
279 |       0 |    0 | 19721687 | 19721687 |                    0
279 |       0 |    1 | 19721687 | 19721687 |                    0
279 |       1 |    0 |   986085 |   986084 | 1.01411202804304e-06
279 |       1 |    1 |   986085 |   986084 | 1.01411202804304e-06
```

```
279 |       1 |    4 |   986085 |   986084 | 1.01411202804304e-06
279 |       2 |    0 |  1775517 |   788460 |     1.00098637606408
279 |       2 |    2 |  1775517 |   788460 |     1.00098637606408
279 |       3 |    0 |  1775517 |   788460 |     1.00098637606408
279 |       3 |    2 |  1775517 |   788460 |     1.00098637606408
279 |       3 |    3 |  1775517 |   788460 |     1.00098637606408
279 |       4 |    0 |  1775517 |   788460 |     1.00098637606408
279 |       4 |    1 |  1775517 |   788460 |     1.00098637606408
279 |       4 |    2 |        1 |        1 |                    0
279 |       5 |    0 |        1 |        1 |                    0
279 |       5 |    1 |        1 |        1 |                    0
279 |       6 |    0 |       20 |       20 |                    0
279 |       6 |    1 |        1 |        1 |                    0
279 |       7 |    0 |        1 |        1 |                    0
279 |       7 |    1 |        0 |        0 |                    0
(19 rows)
```

SVL_QUERY_SUMMARY

使用 SVL_QUERY_SUMMARY 视图查找有关查询执行的一般信息。

SVL_QUERY_SUMMARY 视图包含 SVL_QUERY_REPORT 视图的数据子集。注意，SVL_QUERY_SUMMARY 中的信息来自对所有节点的汇总。

> **Note**
>
> SVL_QUERY_SUMMARY 视图只包含有关 Amazon Redshift 执行的查询的信息，不包含其它实用工具和 DDL 命令执行的查询的相关信息。有关 Amazon Redshift（包括 DDL 和实用工具命令）执行的所有语句的完整列表和信息，请查询 SVL_STATEMENTTEXT 视图。

SVL_QUERY_SUMMARY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

有关 SVCS_QUERY_SUMMARY 的信息，请参阅 SVCS_QUERY_SUMMARY。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。
query	integer	查询 ID。可用于联接各种其他系统表和视图。
stm	integer	流：查询中的一组并发段。查询拥有一个或多个流。
seg	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。查询段可并行运行。每个段在一个进程中运行。
step	integer	运行的查询步骤。
maxtime	bigint	供步骤运行的最大时长（单位为微秒）。
avgtime	bigint	供步骤运行的平均时长（单位为微秒）。
rows	bigint	查询步骤涉及的数据行数。
bytes	bigint	查询步骤涉及的数据字节数。
rate_row	double precision	每行的查询执行率。
rate_byte	double precision	每字节的查询执行率。
label	text	由查询步骤名称及（如适用）表 ID、表名称组成的步骤标签（例如 scan tbl=100448 name =user）。三位表 ID 通常是指临时表的扫描。如果显示 `tbl=0`，则通常是指常量值的扫描。
is_diskbased	character(1)	此查询步骤是否作为基于磁盘的操作在集群中的任意节点上执行：true（**t**）或 false（**f**）。只有哈希、排序和聚合等特定步骤才能转到磁盘。许多类型的步骤始终在内存中执行。
workmem	bigint	分配到查询步骤的工作内存量（单位为字节）。
is_rrscan	character(1)	如果为 true (**t**)，则表示对步骤使用了限制范围的扫描。默认为 false (**f**)。

列名称	数据类型	描述
is_delayed_scan	character(1)	如果为 true (**t**)，则表示对步骤使用了延迟扫描。默认为 false (**f**)。
rows_pre_filter	bigint	对于永久表的扫描，这是在筛选标记为删除的行（虚影行）之前发出的行的总数。

示例查询

查看查询步骤的处理信息

下面的查询显示查询 87 的每个步骤的基本处理信息：

```
select query, stm, seg, step, rows, bytes
from svl_query_summary
where query = 87
order by query, seg, step;
```

此查询检索有关查询 87 的处理信息，如下面的示例输出所示：

```
 query | stm | seg | step |  rows  |  bytes
-------+-----+-----+------+--------+---------
87     |   0 |   0 |    0 |     90 |    1890
87     |   0 |   0 |    2 |     90 |     360
87     |   0 |   1 |    0 |     90 |     360
87     |   0 |   1 |    2 |     90 |    1440
87     |   1 |   2 |    0 | 210494 | 4209880
87     |   1 |   2 |    3 |  89500 |       0
87     |   1 |   2 |    6 |      4 |      96
87     |   2 |   3 |    0 |      4 |      96
87     |   2 |   3 |    1 |      4 |      96
87     |   2 |   4 |    0 |      4 |      96
87     |   2 |   4 |    1 |      1 |      24
87     |   3 |   5 |    0 |      1 |      24
87     |   3 |   5 |    4 |      0 |       0
(13 rows)
```

确定查询步骤是否溢出到磁盘

下面的查询显示查询 ID 为 1025 的查询是否有任何步骤（有关如何获取查询 ID 的信息，请参阅 SVL_QLOG 视图）溢出到磁盘，还是查询完全在内存中运行：

```
select query, step, rows, workmem, label, is_diskbased
from svl_query_summary
where query = 1025
order by workmem desc;
```

此查询返回以下示例输出：

```
query| step|  rows   |  workmem    |  label         | is_diskbased
-----+-----+--------+-----------+---------------+-------------
1025 |  0  |16000000|  141557760 |scan tbl=9     | f
1025 |  2  |16000000|  135266304 |hash tbl=142   | t
1025 |  0  |16000000|  128974848 |scan tbl=116536| f
1025 |  2  |16000000|  122683392 |dist           | f
(4 rows)
```

通过扫描 IS_DISKBASED 的值，您能够了解哪些查询步骤溢出到磁盘。对于查询 1025，哈希步骤在磁盘上运行。可能在磁盘上执行的步骤包括哈希、聚合和排序步骤。要仅查看基于磁盘的查询步骤，请在上面的示例中向 SQL 语句添加 **`and is_diskbased = 't'`** 子句。

SVL_RESTORE_ALTER_TABLE_PROGRESS

在经典大小调整期间，使用 SVL_RESTORE_ALTER_TABLE_PROGRESS 监控集群中每个表迁入 RA3 节点的进度。它捕获调整大小操作期间数据迁移的历史吞吐量。有关 RA3 节点经典大小调整的更多信息，请转到经典大小调整。

SVL_RESTORE_ALTER_TABLE_PROGRESS 只对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_RESTORE_LOG 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

> **Note**
>
> 进度为 `100.00%` 或 `ABORTED` 的行将在 7 天后删除。在经典大小调整期间或之后删除的表行仍可显示在 SVL_RESTORE_ALTER_TABLE_PROGRESS 中。

表列

列名称	数据类型	描述
tbl	整数	表的 ID。
进度	char(32)	表的再分配进度的状态。可能的值是 0.00% 到 100.00% 的百分比，并显示消息 ABORTED。ABORTED 表示再分发未完成就停止，原因将在 message 列中说明。
消息	char(256)	与表的再分发进度相关的消息。

示例查询

以下查询返回了正在运行的查询和已排队的查询。

```
select * from svl_restore_alter_table_progress;

tbl     | progress |                           message
--------+----------+-------------------------------------------------------------
105614  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105610  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105594  | 0.00%    | Table waiting for alter diststyle conversion.
105602  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105606  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105598  | 100.00%  | Restored to distkey successfully.
```

SVL_S3LIST

可以使用 SVL_S3LIST 视图获取有关段级别的 Amazon Redshift Spectrum 查询的详细信息。

SVL_S3LIST 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> SVL_S3LIST 只包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图

SYS_EXTERNAL_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
query	integer	查询 ID。
segment	integer	段编号。查询由多个段组成。
node	integer	节点编号。
slice	integer	运行的特定段所针对的数据切片。
eventtime	timestamp	记录事件的 UTC 时间。
桶	text	Amazon S3 桶名称。
前缀	text	Amazon S3 桶位置的前缀。
recursive	char(1)	是否具有子文件夹的递归扫描。
retrieved_files	integer	列出的文件数。
max_file_size	bigint	列出的文件中的最大文件大小。
avg_file_size	double precision	列出的文件中的平均文件大小。
generated_splits	integer	文件拆分数。
avg_split_length	double precision	文件拆分的平均长度，以字节为单位。
duration	bigint	文件列表的持续时间，以微秒为单位。

示例查询

以下示例查询上次运行的查询的 SVL_S3LIST。

```
select *
from svl_s3list
where query = pg_last_query_id()
order by query,segment;
```

SVL_S3LOG

使用 SVL_S3LOG 视图获取有关段和节点切片级别的 Amazon Redshift Spectrum 查询的详细信息。

SVL_S3LOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> SVL_S3LOG 只包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_EXTERNAL_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
pid	integer	进程 ID。
query	integer	查询 ID。
segment	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。
step	integer	运行的查询步骤。
node	integer	节点编号。
slice	integer	运行的特定段所针对的数据切片。

列名称	数据类型	描述
eventtime	timestamp	开始执行步骤的时间（采用 UTC 表示）。
message	text	日志条目的消息。

示例查询

以下示例查询上次运行的查询的 SVL_S3LOG。

```
select *
from svl_s3log
where query = pg_last_query_id()
order by query,segment,slice;
```

SVL_S3PARTITION

使用 SVL_S3PARTITION 视图可获取有关段和节点切片级别的 Amazon Redshift Spectrum 分区的详细信息。

SVL_S3PARTITION 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> SVL_S3PARTITION 只包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图 SYS_EXTERNAL_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
query	integer	查询 ID。
segment	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。

列名称	数据类型	描述
node	integer	节点编号。
slice	integer	运行的特定段所针对的数据切片。
starttime	不带时区的时间戳	分区修剪开始执行的时间（采用 UTC 表示）。
endtime	不带时区的时间戳	分区修剪完成的时间（采用 UTC 表示）。
duration	bigint	已用时间（以微秒为单位）
total_partitions	integer	分区总数。
qualified_partitions	integer	符合要求的分区数。
assigned_partitions	integer	切片上已分配的分区数。
分配	字符	分配类型。

示例查询

以下示例获取上次完成的查询的分区详细信息。

```
SELECT query, segment,
       MIN(starttime) AS starttime,
       MAX(endtime) AS endtime,
       datediff(ms,MIN(starttime),MAX(endtime)) AS dur_ms,
       MAX(total_partitions) AS total_partitions,
       MAX(qualified_partitions) AS qualified_partitions,
       MAX(assignment) as assignment_type
FROM svl_s3partition
WHERE query=pg_last_query_id()
GROUP BY query, segment
```

```
query | segment |           starttime           |           endtime           | dur_ms|
 total_partitions | qualified_partitions | assignment_type
```

```
------+---------+-------------------------------+----------------------------+-------
+-----------------+---------------------+---------------
99232 |       0 | 2018-04-17 22:43:50.201515    | 2018-04-17 22:43:54.674595  |  4473 |
       2526       |        334           | p
```

SVL_S3PARTITION_SUMMARY

可以使用 SVL_S3PARTITION_SUMMARY 视图获取段级别的 Redshift Spectrum 查询分区处理摘要。

SVL_S3PARTITION_SUMMARY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

有关 SVCS_S3PARTITION 的信息，请参阅 SVCS_S3PARTITION_SUMMARY。

表列

列名称	数据类型	描述
query	integer	查询 ID。您可以使用此值联接各种其他系统表和视图。
segment	integer	段编号。查询由多个段组成。
分配	char(1)	节点中的分区分配类型。
min_start time	timestamp	开始执行分区处理的 UTC 时间。
max_endti me	timestamp	完成分区处理的 UTC 时间。
min_durat ion	bigint	节点在该查询中使用的最小分区处理时间（以微秒为单位）。
max_durat ion	bigint	节点在该查询中使用的最大分区处理时间（以微秒为单位）。
avg_durat ion	bigint	节点在该查询中使用的平均分区处理时间（以微秒为单位）。
total_par titions	integer	外部表中的总分区数。

列名称	数据类型	描述
qualified_partitions	integer	符合条件的总分区数。
min_assigned_partitions	integer	在一个节点上分配的最小分区数。
max_assigned_partitions	integer	在一个节点上分配的最大分区数。
avg_assigned_partitions	bigint	在一个节点上分配的平均分区数。

示例查询

以下示例获取上次完成的查询的分区扫描详细信息。

```
select query, segment, assignment, min_starttime, max_endtime, min_duration,
 avg_duration
from svl_s3partition_summary
where query = pg_last_query_id()
order by query,segment;
```

SVL_S3QUERY

使用 SVL_S3QUERY 视图获取有关段和节点切片级别的 Amazon Redshift Spectrum 查询的详细信息。

SVL_S3QUERY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

> **Note**
>
> SVL_S3QUERY 只包含在主集群上运行的查询。它不包含在并发扩展集群上运行的查询。要访问在主集群和并发扩展集群上运行的查询，我们建议您使用 SYS 监控视图

SYS_EXTERNAL_QUERY_DETAIL。SYS 监控视图中的数据经过格式化处理，便于使用和理解。

表列

列名称	数据类型	描述
userid	integer	生成了给定条目的用户的 ID。
query	integer	查询 ID。
segment	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。
step	integer	运行的查询步骤。
node	integer	节点编号。
slice	integer	运行的特定段所针对的数据切片。
starttime	timestamp	开始执行查询的时间（采用 UTC 表示）。
endtime	timestamp	完成执行查询的时间（采用 UTC 表示）
elapsed	integer	已用时间（以微秒为单位）
external_table_name	char(136)	s3 扫描步骤的外部表名称的内部格式。
is_partitioned	char(1)	如果为 true (**t**)，则此列值表示外部表已进行分区。
is_rrscan	char(1)	如果为 true (**t**)，则此列值表示已应用限制范围的扫描。
s3_scanned_rows	bigint	已从 Amazon S3 扫描并发送到 Redshift Spectrum 层的行数量。

列名称	数据类型	描述
s3_scanned_bytes	bigint	已从 Amazon S3 扫描并发送到 Redshift Spectrum 层的字节的数量。
s3query_returned_rows	bigint	已从 Redshift Spectrum 层返回到集群的行的数量。
s3query_returned_bytes	bigint	已从 Redshift Spectrum 层返回到集群的字节的数量。
files	integer	此 S3 扫描步骤中在此切片上已处理的文件的数量。
splits	int	在此切片上处理的拆分数。对于大型可拆分数据文件（例如，大于 512 MB 左右的数据文件），Redshift Spectrum 会尝试将文件拆分为多个 S3 请求以便进行并行处理。
total_split_size	bigint	在此切片上处理的所有拆分的总大小（以字节为单位）。
max_split_size	bigint	为此切片处理的最大拆分大小（以字节为单位）。
total_retries	integer	已处理文件的总重试次数。
max_retries	integer	单个已处理文件的最大重试次数。
max_request_duration	integer	单个 Redshift Spectrum 请求的最长持续时间（以微秒为单位）。
avg_request_duration	double precision	Redshift Spectrum 请求的平均持续时间（以微秒为单位）。

列名称	数据类型	描述
max_request_parallelism	integer	此 S3 扫描步骤中此切片上的未完成的 Redshift Spectrum 的最大数量。
avg_request_parallelism	double precision	此 S3 扫描步骤中此切片上的并行 Redshift Spectrum 请求的平均数量。

示例查询

以下示例获取上次完成的查询的扫描步骤详细信息。

```
select query, segment, slice, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
from svl_s3query
where query = pg_last_query_id()
order by query,segment,slice;
```

```
query | segment | slice | elapsed | s3_scanned_rows | s3_scanned_bytes |
 s3query_returned_rows | s3query_returned_bytes | files
------+---------+-------+---------+-----------------+------------------
+----------------------+-------------------------+------
 4587 |       2 |     0 |   67811 |               0 |                0 |
      0 |                      0 |     0
 4587 |       2 |     1 |  591568 |          172462 |         11260097 |
   8513 |                  170260 |     1
 4587 |       2 |     2 |  216849 |               0 |                0 |
      0 |                      0 |     0
 4587 |       2 |     3 |  216671 |               0 |                0 |
      0 |                      0 |     0
```

SVL_S3QUERY_SUMMARY

使用 SVL_S3QUERY_SUMMARY 视图可获取已在系统上运行的所有 Amazon Redshift Spectrum 查询（S3 查询）的汇总。SVL_S3QUERY_SUMMARY 汇总段级别的 SVL_S3QUERY 的详细信息。

SVL_S3QUERY_SUMMARY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_EXTERNAL_QUERY_DETAIL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

对于 SVCS_S3QUERY_SUMMARY，请参阅 SVCS_S3QUERY_SUMMARY。

表列

列名称	数据类型	描述
userid	integer	已生成给定条目的用户的 ID。
query	integer	查询 ID。您可以使用此值联接各种其他系统表和视图。
xid	bigint	事务 ID。
pid	integer	进程 ID。
segment	integer	段编号。一个查询包含多个区段，而且每个区段包含一个或多个步骤。
step	integer	运行的查询步骤。
starttime	timestamp	开始执行查询的时间（采用 UTC 表示）。
endtime	timestamp	查询完成的时间（采用 UTC 表示）。
elapsed	integer	运行查询花费的时长（单位为微秒）。
aborted	integer	如果查询已由系统停止或已由用户取消，则此列包含 **1**。如果查询运行完成，则此列包含 **0**。
external_table_name	char(136)	外部表扫描的表的外部名称的名称内部格式。
file_format	character(16)	外部表数据的文件格式。
is_partitioned	char(1)	如果为 true (**t**)，则此列值表示外部表已进行分区。
is_rrscan	char(1)	如果为 true (**t**)，则此列值表示已应用限制范围的扫描。

列名称	数据类型	描述
is_nested	char(1)	如果为 true (**t**)，该列值表示访问嵌套的列数据类型。
s3_scanned_rows	bigint	已从 Amazon S3 扫描并发送到 Redshift Spectrum 层的行数量。
s3_scanned_bytes	bigint	已从 Amazon S3 扫描并发送到 Redshift Spectrum 层的基于压缩数据的字节的数量。
s3query_returned_rows	bigint	已从 Redshift Spectrum 层返回到集群的行的数量。
s3query_returned_bytes	bigint	已从 Redshift Spectrum 层返回到集群的字节的数量。返回到 Amazon Redshift 的大量数据可能会影响系统性能。
files	integer	针对此 Redshift Spectrum 查询已处理的文件的数量。文件数量少会限制并行处理的优势。
files_max	integer	在一个切片上处理的最大文件数。
files_avg	integer	在一个切片上处理的平均文件数。
splits	int	为此分段处理的拆分数。在此切片上处理的拆分数。对于大型可拆分数据文件（例如，大于 512 MB 左右的数据文件），Redshift Spectrum 会尝试将文件拆分为多个 S3 请求以便进行并行处理。
splits_max	int	在此切片上处理的最大拆分数。
splits_avg	int	在此切片上处理的平均拆分数。
total_split_size	bigint	处理的所有拆分的总大小。
max_split_size	bigint	处理的最大拆分大小（以字节为单位）。

列名称	数据类型	描述
avg_split_size	bigint	处理的平均拆分大小（以字节为单位）。
total_retries	integer	单个处理文件的最大重试次数。
max_retries	integer	任何已处理文件的最大重试次数。
max_request_duration	integer	单个文件请求的最长持续时间（以微秒为单位）。长时间运行的查询可能表示瓶颈。
avg_request_duration	double precision	文件请求的平均持续时间（以微秒为单位）。
max_request_parallelism	integer	该 Redshift Spectrum 查询的一个切片中的最大并行请求数。
avg_request_parallelism	double precision	该 Redshift Spectrum 查询的一个切片中的平均并行请求数。
total_slowdown_count	bigint	在外部表扫描期间出现减速错误的总 Amazon S3 请求数。
max_slowdown_count	integer	在一个切片上的外部表扫描期间出现减速错误的最大 Amazon S3 请求数。

示例查询

以下示例获取上次完成的查询的扫描步骤详细信息。

```
select query, segment, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
from svl_s3query_summary
```

```
where query = pg_last_query_id()
order by query,segment;
```

```
query | segment | elapsed | s3_scanned_rows | s3_scanned_bytes | s3query_returned_rows
 | s3query_returned_bytes | files
------+---------+---------+-----------------+------------------+----------------------
+-------------------------+------
 4587 |       2 |   67811 |               0 |                0 |                    0
 |                     0 |     0
 4587 |       2 |  591568 |          172462 |         11260097 |                 8513
 |                170260 |     1
 4587 |       2 |  216849 |               0 |                0 |                    0
 |                     0 |     0
 4587 |       2 |  216671 |               0 |                0 |                    0
 |                     0 |     0
```

SVL_S3RETRIES

使用 SVL_S3RETRIES 视图获取有关基于 Amazon S3 的 Amazon Redshift Spectrum 查询的失败原因的信息。

SVL_S3RETRIES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述		
query	integer	查询 ID。		
segment	integer	段编号。 一个查询包含多个区段，而且每个区段包含一个或多个步骤。查询段可并行运行。每个段在一个进程中运行。		
node	integer	节点编号。		

列名称	数据类型	描述		
slice	integer	运行的特定段所针对的数据切片。		
eventtime	不带时区的时间戳	开始执行步骤的时间（采用 UTC 表示）。		
retries	integer	查询重试次数。		
successful_fetches	integer	返回数据的次数。		
file_size	bigint	文件的大小（以字节为单位）。		
位置	text	表的位置。		
message	text	错误消息。		

示例查询

下面示例检索有关失败的 S3 查询的数据。

```
SELECT svl_s3retries.query, svl_s3retries.segment, svl_s3retries.node,
 svl_s3retries.slice, svl_s3retries.eventtime, svl_s3retries.retries,
svl_s3retries.successful_fetches, svl_s3retries.file_size,
 btrim((svl_s3retries."location")::text) AS "location",
 btrim((svl_s3retries.message)::text)
AS message FROM svl_s3retries;
```

SVL_SPATIAL_SIMPLIFY

您可以查询系统视图 SVL_SPATIAL_SIMPLIFY，以使用 COPY 命令获取有关简化空间几何对象的信息。在 shapefile 上使用 COPY 时，您可以指定 SIMPLIFY `tolerance`、SIMPLIFY AUTO 和 SIMPLIFY AUTO `max_tolerance` 摄入选项。简化的结果在 SVL_SPATIAL_SIMPLIFY 系统视图中进行总结。

当设置 SIMPLIFY AUTO `max_tolerance` 时，此视图为每个超出最大大小的几何体都包含一行。当设置 SIMPLIFY `tolerance` 时，则存储整个 COPY 操作的一行。此行引用 COPY 查询 ID 和指定的简化容差。

SVL_SPATIAL_SIMPLIFY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_SPATIAL_SIMPLIFY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
query	integer	生成此行的查询（COPY 命令）的 ID。
line_number	integer	指定 COPY `SIMPLIFY AUTO` 选项时，此值是 shapefile 中简化记录的记录编号。
maximum_tolerance	double	COPY 命令中指定的距离容差值。它是使用 `SIMPLIFY AUTO` 选项的最大容差值，或使用 `SIMPLIFY` 选项的固定容差值。
initial_size	integer	以字节为单位的 `GEOMETRY` 数据值简化前的大小。
simplified	char(1)	当指定 COPY `SIMPLIFY AUTO` 选项时，如果几何体已成功简化，则为 `t`，否则为 `f`。如果在使用给定的最大容差进行简化后，其大小仍然大于最大几何体大小，则可能无法成功简化几何体。
final_size	integer	当指定 COPY `SIMPLIFY AUTO` 选项时，它是以字节数为单位的几何体简化后的大小。
final_tolerance	double	

示例查询

以下查询返回 COPY 简化的记录的列表。

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
 query | line_number | maximum_tolerance | initial_size | simplified | final_size |
 final_tolerance
-------+-------------+-------------------+--------------+------------+------------
+----------------------
    20 |     1184704 |                -1 |      1513736 | t          |    1008808 |
 1.276386653895e-05
    20 |     1664115 |                -1 |      1233456 | t          |    1023584 |
 6.11707814796635e-06
```

SVL_SPECTRUM_SCAN_ERROR

您可以查询 SVL_SPECTRUM_SCAN_ERROR 系统视图，以获取有关 Redshift Spectrum 扫描错误的信息。

SVL_SPECTRUM_SCAN_ERROR 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_EXTERNAL_QUERY_ERROR 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

显示记录的错误示例。每个查询默认显示 10 个条目。

列名称	数据类型	描述
userid	整数	生成该行的用户的 ID。
query	整数	生成此行的查询的 ID。
location	character(128)	查询的数据位置。
rowid	character(128)	文件中错误的位置。各个 rowid 部分用 :（冒号）分隔，将来可能会添加其他部分。 *row_offset :row_group :row_id* row_offset 是文件中行的偏移量（以字节为单位），对于不支持的文件格式，设置为 -1。表划分为 row_group，每个组都有带不同 row_id 的行。

列名称	数据类型	描述
colname	character(128)	查询返回的列的名称。
original_value	character(128)	查询的原始值。
modified_value	character(128)	根据查询中指定的数据处理配置选项返回的修改值。
trigger	character(128)	查询中指定的数据处理选项。
action	character(128)	与查询中指定的数据处理选项相关的操作。
action_value	character(128)	与查询中指定的数据处理选项相关的操作参数值。
error_code	整数	查询中指定的数据处理选项的结果代码。

示例查询

以下查询将返回所执行数据处理操作的行列表。

```
SELECT * FROM svl_spectrum_scan_error;
```

该查询返回的结果类似于以下内容。

```
  userid  query     location                                        rowid   colname
         original_value             modified_value       trigger          action
                 action_valueerror_code
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0     league_name
         Barclays Premier League    Barclays Premier Lea UNSPECIFIED      TRUNCATE
                                     156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0     league_nspi
          34595                      32767                UNSPECIFIED
OVERFLOW_VALUE                                  199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:1     league_nspi
          34151                      32767                UNSPECIFIED
OVERFLOW_VALUE                                  199
```

```
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2     league_name
         Barclays Premier League    Barclays Premier Lea UNSPECIFIED      TRUNCATE
                                    156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2     league_nspi
          33223                     32767               UNSPECIFIED
OVERFLOW_VALUE                                  199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3     league_name
         Barclays Premier League    Barclays Premier Lea UNSPECIFIED      TRUNCATE
                                    156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3     league_nspi
          32808                     32767               UNSPECIFIED
OVERFLOW_VALUE                                  199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:4     league_nspi
          32790                     32767               UNSPECIFIED
OVERFLOW_VALUE                                  199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:5     league_name
         Spanish Primera Division   Spanish Primera Divi UNSPECIFIED      TRUNCATE
                                    156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:6     league_name
         Spanish Primera Division   Spanish Primera Divi UNSPECIFIED      TRUNCATE
                                    156
```

SVL_STATEMENTTEXT

使用 SVL_STATEMENTTEXT 视图获取已在系统上运行的所有 SQL 命令的完整记录。

SVL_STATEMENTTEXT 视图包含 STL_DDLTEXT、STL_QUERYTEXT、STL_UTILITYTEXT 表中所有行的联合。此外，此视图还包含到 STL_QUERY 表的联接。

SVL_STATEMENTTEXT 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	生成条目的用户的 ID。

列名称	数据类型	描述
xid	bigint	与语句关联的事务 ID。
pid	integer	语句的进程 ID。
label	character(320)	用于运行查询的文件的名称或使用 SET QUERY_GROUP 命令定义的标签。如果查询并非基于文件或未设置 QUERY_GROUP 参数，则此字段为空。
starttime	timestamp	开始执行语句的确切时间，秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.131358**
endtime	timestamp	语句执行完成的确切时间，秒的小数部分以 6 位精度表示。例如：**2009-06-12 11:29:19.193640**
sequence	integer	当一个语句包含 200 多个字符时，将为该语句记录额外的行。序列 0 是第一行，1 是第二行，依此类推。
type	varchar(10)	SQL 语句的类型：**QUERY**、**DDL** 或 **UTILITY**。
text	character(200)	SQL 文本，以 200 个字符递增。此字段可能包含反斜杠 (\\) 和换行符 (\n) 等特殊字符。

示例查询

下面的查询返回在 2009 年 6 月 16 日运行的 DDL 语句：

```
select starttime, type, rtrim(text) from svl_statementtext
where starttime like '2009-06-16%' and type='DDL' order by starttime asc;

starttime                  | type |              rtrim
---------------------------|------|--------------------------------
2009-06-16 10:36:50.625097 | DDL  | create table ddltest(c1 int);
2009-06-16 15:02:16.006341 | DDL  | drop view alltickitjoin;
2009-06-16 15:02:23.65285  | DDL  | drop table sales;
2009-06-16 15:02:24.548928 | DDL  | drop table listing;
2009-06-16 15:02:25.536655 | DDL  | drop table event;
...
```

重新构造存储的 SQL

要重新构造存储在 SVL_STATEMENTTEXT 的 text 列中的 SQL，请运行 SELECT 语句，以从 text 列中的一个或多个部分创建 SQL。在运行重新构造的 SQL 之前，将任何 (\n) 特殊字符替换为新行。以下 SELECT 语句的结果是 query_statement 字段中重新构造的 SQL 的行。

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS query_statement
from SVL_STATEMENTTEXT where pid=pg_backend_pid();
```

例如，以下查询选择 3 列。查询本身超过 200 个字符，并存储在 SVL_STATEMENTTEXT 中的多个部分内。

```
select
1 AS a01234567890123456789012345678901234567890123456789012345678901234567890,
2 AS b01234567890123456789012345678901234567890123456789012345678901234567890,
3 AS b01234567890123456789012345678901234567890123456789012345678901234
FROM stl_querytext;
```

在此示例中，查询存储在 SVL_STATEMENTTEXT 的 text 列的 2 个部分（行）中。

```
select sequence, text from SVL_STATEMENTTEXT where pid = pg_backend_pid() order by
 starttime, sequence;
```

```
 sequence |
                 text

----------
+-----------------------------------------------------------------------------------------------------------------
        0 | select\n1 AS
 a01234567890123456789012345678901234567890123456789012345678901234567890,\n2 AS
 b01234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b01234567890123456789012345678901234567890123456789012345678901234
        1 | \nFROM stl_querytext;
```

要重新构造存储在 STL_STATEMENTTEXT 中的 SQL，请运行以下 SQL。

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS text
```

```
from SVL_STATEMENTTEXT where pid=pg_backend_pid();
```

要在客户端中使用生成的重新构造的 SQL，需将任何 (\n) 特殊字符替换为新行。

```
                    text

----------------------------------------------------------------------------------------------------
 select\n1 AS a01234567890123456789012345678901234567890123456789012345678901234567890,
\n2 AS b01234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b012345678901234567890123456789012345678901234\nFROM stl_querytext;
```

SVL_STORED_PROC_CALL

您可以查询系统视图 SVL_STORED_PROC_CALL 以获取有关存储过程调用的信息，包括开始时间、结束时间以及是否取消调用。每个存储过程调用接受一个查询 ID。

SVL_STORED_PROC_CALL 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_PROCEDURE_CALL 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	使用其权限来运行语句的用户的 ID。如果此调用嵌套在 SECURITY DEFINER 存储过程中，则这是该存储过程的拥有者的 userid，
session_userid	integer	创建会话的用户的 ID，是顶级存储过程调用的调用方。
query	integer	过程调用的查询 ID。
label	character(320)	用于运行查询的文件的名称或使用 SET QUERY_GROUP 命令定义的标签。如果查询并非基于文件或未设置 QUERY_GROUP 参数，则此字段值为默认值。

列名称	数据类型	描述
xid	bigint	事务 ID。
pid	integer	进程 ID。通常情况下，会话中的所有查询在同一进程中运行，因此，如果您在同一会话中运行一系列查询，则此值通常保持不变。在特定的内部事件之后，Amazon Redshift 可能会重新启动一个活动会话并分配新的 PID 值。有关更多信息，请参阅 STL_RESTARTED_SESSIONS。
数据库	character(32)	在发起查询时用户连接到的数据库的名称。
querytxt	character(4000)	过程调用查询的实际文本。
starttime	timestamp	查询开始运行的时间（用 UTC 表示），有 6 位数字精度，可精确到小数秒，例如：**2009-06-12 11:29:19.131358.**
endtime	timestamp	查询完成运行的时间（用 UTC 表示），有 6 位数字精度，可精确到小数秒，例如：**2009-06-12 11:29:19.131358.**
aborted	integer	如果存储过程已由系统停止或已由用户取消，则此列包含 1。如果调用运行完成，则此列包含 0。
from_sp_call	integer	如果对过程调用的调用是由其他过程调用执行，则此列包含外部调用的查询 ID。否则该字段为 NULL。

示例查询

以下查询按降序顺序返回过去一天中，存储过程调用的用时以及完成状态。

```
select query, datediff(seconds, starttime, endtime) as elapsed_time, aborted,
 trim(querytxt) as call from svl_stored_proc_call where starttime >= getdate() -
 interval '1 day' order by 2 desc;

  query | elapsed_time | aborted |                                                call
--------+--------------+---------
+-------------------------------------------------------------------------------
   4166 |            7 |       0 | call search_batch_status(35,'succeeded');
```

```
2433 |            3 |       0 | call test_batch (123456)
1810 |            1 |       0 | call prod_benchmark (123456)
1836 |            1 |       0 | call prod_testing (123456)
1808 |            1 |       0 | call prod_portfolio ('N', 123456)
1816 |            1 |       1 | call prod_portfolio ('Y', 123456)
```

SVL_STORED_PROC_MESSAGES

您可以查询 SVL_STORED_PROC_MESSAGES 系统视图以获取有关存储过程消息的信息。即使存储过程调用已取消，也会记录引发的消息。每个存储过程调用接受一个查询 ID。有关如何设置记录的消息的最低级别的更多信息，请参阅 stored_proc_log_min_messages。

SVL_STORED_PROC_MESSAGES 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_PROCEDURE_MESSAGES 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
userid	integer	使用其权限来运行语句的用户的 ID。如果此调用嵌套在 SECURITY DEFINER 存储过程中，则这是该存储过程的拥有者的 userid，
session_userid	integer	创建会话的用户的 ID，是顶级存储过程调用的调用方。
pid	integer	进程 ID。
xid	bigint	过程调用查询的事务 ID。
query	integer	过程调用的查询 ID。
recordtime	timestamp	引发消息的时间 (UTC)。
loglevel	integer	引发的消息的日志级别数值。可能的值：20（对于 LOG）、30（对于 INFO）、40（对于 NOTICE）、50（对于 WARNING）、60（对于 EXCEPTION）

列名称	数据类型	描述
loglevel_text	character(10)	与 loglevel 中的数值对应的日志级别。可能的值：LOG、INFO、NOTICE、WARNING 和 EXCEPTION。
message	character(1024)	引发的消息的文本。
linenum	integer	引发的语句的行号。
querytext	character(500)	过程调用查询的实际文本。
label	character(320)	用于运行查询的文件的名称或使用 SET QUERY_GROUP 命令定义的标签。如果查询并非基于文件或未设置 QUERY_GROUP 参数，则此字段值为默认值。
aborted	integer	如果存储过程已由系统停止或已由用户取消，则此列包含 1。如果调用运行完成，则此列包含 0。
message_xid	bigint	引发的消息的事务 ID。

示例查询

以下 SQL 语句说明了如何使用 SVL_STORED_PROC_MESSAGES 查看引发的消息。

```
-- Create and run a stored procedure
CREATE OR REPLACE PROCEDURE test_proc1(f1 int) AS
$$
BEGIN
    RAISE INFO 'Log Level: Input f1 is %',f1;
    RAISE NOTICE 'Notice Level: Input f1 is %',f1;
    EXECUTE 'select invalid';
    RAISE NOTICE 'Should not print this';

EXCEPTION WHEN OTHERS THEN
     raise exception 'EXCEPTION level: Exception Handling';
END;
$$ LANGUAGE plpgsql;

-- Call this stored procedure
CALL test_proc1(2);
```

```
-- Show raised messages with level higher than INFO
SELECT query, recordtime, loglevel, loglevel_text, trim(message) as message, aborted
 FROM svl_stored_proc_messages
  WHERE loglevel > 30 AND query = 193 ORDER BY recordtime;

 query |         recordtime         | loglevel | loglevel_text |               message
              | aborted
-------+----------------------------+----------+---------------
+-------------------------------------+---------
   193 | 2020-03-17 23:57:18.277196 |       40 | NOTICE        | Notice Level: Input f1
 is 2         |       1
   193 | 2020-03-17 23:57:18.277987 |       60 | EXCEPTION     | EXCEPTION level:
 Exception Handling |       1
(2 rows)

-- Show raised messages at EXCEPTION level
SELECT query, recordtime, loglevel, loglevel_text, trim(message) as message, aborted
 FROM svl_stored_proc_messages
  WHERE loglevel_text = 'EXCEPTION' AND query = 193 ORDER BY recordtime;

 query |         recordtime         | loglevel | loglevel_text |               message
              | aborted
-------+----------------------------+----------+---------------
+-------------------------------------+---------
   193 | 2020-03-17 23:57:18.277987 |       60 | EXCEPTION     | EXCEPTION level:
 Exception Handling |       1
```

以下 SQL 语句说明了在创建存储过程时如何使用 SVL_STORED_PROC_MESSAGES 和 SET 选项查看引发的消息。由于 test_proc() 的最低日志级别为 NOTICE，因此，仅在 SVL_STORED_PROC_MESSAGES 中记录 NOTICE、WARNING 和 EXCEPTION 级别消息。

```
-- Create a stored procedure with minimum log level of NOTICE
CREATE OR REPLACE PROCEDURE test_proc() AS
$$
BEGIN
    RAISE LOG 'Raise LOG messages';
    RAISE INFO 'Raise INFO messages';
    RAISE NOTICE 'Raise NOTICE messages';
    RAISE WARNING 'Raise WARNING messages';
    RAISE EXCEPTION 'Raise EXCEPTION messages';
    RAISE WARNING 'Raise WARNING messages again'; -- not reachable
END;
```

```
$$ LANGUAGE plpgsql SET stored_proc_log_min_messages = NOTICE;

-- Call this stored procedure
CALL test_proc();

-- Show the raised messages
SELECT query, recordtime, loglevel_text, trim(message) as message, aborted FROM
 svl_stored_proc_messages
  WHERE query = 149 ORDER BY recordtime;

 query |         recordtime         | loglevel_text |          message         |
 aborted
-------+----------------------------+---------------+--------------------------
+---------
   149 | 2020-03-16 21:51:54.847627 | NOTICE        | Raise NOTICE messages    |
 1
   149 | 2020-03-16 21:51:54.84766  | WARNING       | Raise WARNING messages   |
 1
   149 | 2020-03-16 21:51:54.847668 | EXCEPTION     | Raise EXCEPTION messages |
 1
(3 rows)
```

SVL_TERMINATE

记录用户取消或终止进程的时间。

SELECT PG_TERMINATE_BACKEND(pid)、SELECT PG_CANCEL_BACKEND(pid) 和 CANCEL pid 在 SVL_TERMINATE 中创建日志条目。

SVL_TERMINATE 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_QUERY_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
pid	integer	已取消或已终止的进程的进程 ID。
eventtime	timestamp	取消或终止进程的时间。

列名称	数据类型	描述
userid	integer	运行命令的用户的用户 ID。
type	字符串	终止的类型。可以是 CANCEL 或 TERMINATE。

以下命令显示最新取消的查询。

```
select * from svl_terminate order by eventtime desc limit 1;
 pid  |         eventtime          | userid |  type
------+----------------------------+--------+--------
 8324 | 2020-03-24 09:42:07.298937 |      1 | CANCEL
(1 row)
```

SVL_UDF_LOG

记录在执行用户定义的函数 (UDF) 期间生成的系统定义的错误和警告消息。

SVL_UDF_LOG 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_UDF_LOG 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
query	bigint	查询 ID。您可以使用此 ID 联接各种其他系统表和视图。
message	char(4096)	由函数生成的消息。
created	timestamp	日志的创建时间。
traceback	char(4096)	此值提供 UDF 的堆栈回溯（如果可用）。有关更多信息，请参阅 Python 标准库中的回溯。

列名称	数据类型	描述
funcname	character(256)	正在执行的 UDF 的名称。
node	integer	生成消息的节点。
slice	integer	生成消息的切片。
seq	integer	消息在切片上的顺序。

示例查询

以下示例说明 UDF 如何处理系统定义的错误。第一个块显示了返回参数的逆参数的 UDF 函数的定义。在运行函数并提供参数 0 时，如第二个块所示，函数将返回错误。第三个语句将读取在 SVL_UDF_LOG 中记录的错误消息

```
-- Create a function to find the inverse of a number

CREATE OR REPLACE FUNCTION  f_udf_inv(a int)
  RETURNS float IMMUTABLE
AS $$
   return 1/a
$$ LANGUAGE plpythonu;

-- Run the function with a 0 argument to create an error
Select f_udf_inv(0) from sales;

-- Query SVL_UDF_LOG to view the message

Select query, created, message::varchar
from svl_udf_log;

 query |          created           | message
-------+----------------------------
+-----------------------------------------------------------
  2211 | 2015-08-22 00:11:12.04819  | ZeroDivisionError: long division or modulo by
 zero\nNone
```

以下示例将日志记录和警告消息添加到 UDF 中，以便被零除运算生成警告消息而不是停止并显示错误消息。

```
-- Create a function to find the inverse of a number and log a warning

CREATE OR REPLACE FUNCTION f_udf_inv_log(a int)
  RETURNS float IMMUTABLE
 AS $$
  import logging
  logger = logging.getLogger() #get root logger
  if a==0:
    logger.warning('You attempted to divide by zero.\nReturning zero instead of error.
\n')
    return 0
  else:
     return 1/a
$$ LANGUAGE plpythonu;
```

以下示例运行该函数，然后查询 SVL_UDF_LOG 以查看消息。

```
-- Run the function with a 0 argument to trigger the warning
Select f_udf_inv_log(0) from sales;

-- Query SVL_UDF_LOG to view the message

Select query, created, message::varchar
from svl_udf_log;

query |          created           | message
------+----------------------------+---------------------------------
    0 | 2015-08-22 00:11:12.04819  | You attempted to divide by zero.
                                     Returning zero instead of error.
```

SVL_USER_INFO

您可以使用 SVL_USER_INFO 视图检索有关 Amazon Redshift 数据库用户的数据。

SVL_USER_INFO 仅对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
usename	text	角色的用户名称。

列名称	数据类型	描述
usesysid	integer	用户的用户 ID。
usecreatedb	布尔值	一个值，指示用户是否具有创建数据库的权限。
usesuper	布尔值	一个值，指示用户是否为超级用户。
usecatupd	布尔值	一个值，指示用户是否可以更新系统目录。
useconnlimit	text	用户可以打开的连接数。
syslogaccess	text	一个值，指示用户是否具有访问系统日志的权限。两个可能的值为 RESTRICTED 和 UNRESTRICTED 。RESTRICTED 表示非超级用户可以看到自己的记录。UNRESTRICTED 表示非超级用户可以看到他们拥有 SELECT 权限的系统视图和表中的所有记录。
last_ddl_ts	timestamp	用户运行的最后一条数据定义语言 (DDL) create 语句的时间戳。
sessiontimeout	integer	超时前会话保持非活动状态或空闲状态的最长时间（秒）。0 表示未设置超时。有关集群的闲置或非活动超时设置的信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 中的配额和限制。
external_id	文本	用户在第三方身份提供者中的唯一标识符。

示例查询

以下命令从 SVL_USER_INFO 检索用户信息。

```
SELECT * FROM SVL_USER_INFO;
```

SVL_VACUUM_PERCENTAGE

SVL_VACUUM_PERCENTAGE 视图报告执行 vacuum 操作后为表分配的数据块的百分比。此百分比数字显示回收了多少磁盘空间。有关 vacuum 实用工具的更多信息，请参阅 VACUUM 命令。

SVL_VACUUM_PERCENTAGE 只对超级用户可见。有关更多信息，请参阅 系统表和视图中的数据可见性。

此表中的部分或全部数据也可以在 SYS 监控视图 SYS_VACUUM_HISTORY 中找到。SYS 监控视图中的数据经过格式化处理，便于使用和理解。我们建议您使用 SYS 监控视图进行查询。

表列

列名称	数据类型	描述
xid	bigint	vacuum 语句的事务 ID。
table_id	integer	已执行 vacuum 操作的表的 ID。
percentage	bigint	执行 vacuum 操作后数据块的百分比（相对于运行 vacuum 操作前表中的数据块数）。

示例查询

下面的查询显示表 100238 上特定操作的百分比：

```
select * from svl_vacuum_percentage
where table_id=100238 and xid=2200;

xid  | table_id | percentage
-----+----------+------------
1337 |   100238 |         60
(1 row)
```

执行此 vacuum 操作后，表包含原始数据块的 60%。

系统目录表

主题

- PG_ATTRIBUTE_INFO
- PG_CLASS_INFO
- PG_DATABASE_INFO
- PG_DEFAULT_ACL

- PG_EXTERNAL_SCHEMA
- PG_LIBRARY
- PG_PROC_INFO
- PG_STATISTIC_INDICATOR
- PG_TABLE_DEF
- PG_USER_INFO
- 查询目录表

系统目录存储 schema 元数据，例如，有关表和列的信息。系统目录表具有 PG 前缀。

Amazon Redshift 用户可访问标准 PostgreSQL 目录表。有关 PostgreSQL 系统目录的更多信息，请参阅 PostgreSQL 系统表。

PG_ATTRIBUTE_INFO

PG_ATTRIBUTE_INFO 是基于 PostgreSQL 目录表 PG_ATTRIBUTE 和内部目录表 PG_ATTRIBUTE_ACL 构建的 Amazon Redshift 系统视图。PG_ATTRIBUTE_INFO 包括有关表或视图的列的详细信息，包括列访问控制列表（如果有）。

表列

除了 PG_ATTRIBUTE 中的列之外，PG_ATTRIBUTE_INFO 还显示以下列。

列名称	数据类型	描述
attacl	aclitem[]	已经专门对此列授予的列级访问权限（如果有）。

PG_CLASS_INFO

PG_CLASS_INFO 是基于 PostgreSQL 目录表 PG_CLASS 和 PG_CLASS_EXTENDED 而构建的 Amazon Redshift 系统视图。PG_CLASS_INFO 包括有关表创建时间和当前分配方式的详细信息。有关更多信息，请参阅使用数据分配样式。

PG_CLASS_INFO 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

除了 PG_CLASS 中的列以外，PG_CLASS_INFO 还显示以下列。PG_CLASS 中的 `oid` 列在 PG_CLASS_INFO 表中称为 `relid`。

列名称	数据类型	描述
relcreationtime	timestamp	创建表的 UTC 时间。
releffectivediststyle	integer	表的分配方式，或者如果表使用自动分配，则为 Amazon Redshift 分配的当前分配方式。

PG_CLASS_INFO 中的 RELEFFECTIVEDISTSTYLE 列指示表的当前分配方式。如果表使用自动分配，则 RELEFFECTIVEDISTSTYLE 为 10、11 或 12，这指示有效分配方式是 AUTO（ALL）、AUTO（EVEN）或 AUTO（KEY）。如果表使用自动分配，则分配方式最初可能显示 AUTO（ALL）。然后，当表增长时更改为 AUTO（EVEN），或者，如果发现某个列可用作分配键，则更改为 AUTO（KEY）。

下表的 RELEFFECTIVEDISTSTYLE 列中提供了每个值的分配方式：

RELEFFECTIVEDISTSTYLE	当前分配方式
0	EVEN
1	KEY
8	ALL
10	AUTO (ALL)
11	AUTO (EVEN)
12	AUTO (KEY)

示例

以下查询返回目录中表的当前分配模式。

```
select reloid as tableid,trim(nspname) as schemaname,trim(relname) as
 tablename,reldiststyle,releffectivediststyle,
CASE WHEN "reldiststyle" = 0 THEN 'EVEN'::text
     WHEN "reldiststyle" = 1 THEN 'KEY'::text
     WHEN "reldiststyle" = 8 THEN 'ALL'::text
     WHEN "releffectivediststyle" = 10 THEN 'AUTO(ALL)'::text
     WHEN "releffectivediststyle" = 11 THEN 'AUTO(EVEN)'::text
     WHEN "releffectivediststyle" = 12 THEN 'AUTO(KEY)'::text ELSE '<<UNKNOWN>>'::text
 END as diststyle,relcreationtime
from pg_class_info a left join pg_namespace b on a.relnamespace=b.oid;
```

```
 tableid | schemaname | tablename | reldiststyle | releffectivediststyle | diststyle  |
      relcreationtime
---------+------------+-----------+--------------+-----------------------+------------
+----------------------------
 3638033 | public     | customer  |            0 |                     0 | EVEN       |
 2019-06-13 15:02:50.666718
 3638037 | public     | sales     |            1 |                     1 | KEY        |
 2019-06-13 15:03:29.595007
 3638035 | public     | lineitem  |            8 |                     8 | ALL        |
 2019-06-13 15:03:01.378538
 3638039 | public     | product   |            9 |                    10 | AUTO(ALL)  |
 2019-06-13 15:03:42.691611
 3638041 | public     | shipping  |            9 |                    11 | AUTO(EVEN) |
 2019-06-13 15:03:53.69192
 3638043 | public     | support   |            9 |                    12 | AUTO(KEY)  |
 2019-06-13 15:03:59.120695
(6 rows)
```

PG_DATABASE_INFO

PG_DATABASE_INFO 是一个扩展 PostgreSQL 目录表 PG_DATABASE 的 Amazon Redshift 系统视图。

PG_DATABASE_INFO 对所有用户可见。

表列

除了 PG_DATABASE 中的列之外，PG_DATABASE_INFO 还包含以下列。PG_DATABASE 中的 `oid` 列在 PG_DATABASE_INFO 表中称为 `datid`。有关更多信息，请参阅 PostgreSQL 文档。

列名称	数据类型	描述
datid	oid	系统表在内部使用的对象标识符 (OID)
datconnlimit	text	可以对此数据库进行的最大并行连接数。值 -1 表示无限制。

PG_DEFAULT_ACL

存储有关默认访问权限的信息。有关默认访问权限的更多信息，请参阅 ALTER DEFAULT PRIVILEGES。

PG_DEFAULT_ACL 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
defacluser	integer	列出的权限所应用于的用户的 ID。
defaclnamespace	oid	默认权限应用于的 schema 的对象 ID。如果未指定 schema，则默认值为 0。
defaclobjtype	character	默认权限应用于的对象的类型。有效值如下所示： • r–关系（表或视图） • f–函数 • p–存储过程
defaclacl	aclitem[]	一个字符串，该字符串定义指定用户或用户组及对象类型的默认权限。 如果向用户授予权限，则字符串采用以下格式： *{ username=privilegestring/grantor }* username

列名称	数据类型	描述
		向其授予权限的用户的名称。如果忽略 username，则将权限授予 PUBLIC。 如果向用户组授予权限，则字符串采用以下格式： *{ "group groupname=privilegestring/grantor" }* privilegestring 一个字符串，该字符串指定要授予的权限。 有效值为： • r–SELECT（读取） • a–INSERT（附加） • w–UPDATE（写入） • d–DELETE • x–授予用于创建外键约束的权限 (REFERENCES)。 • X–EXECUTE • *–指示接收上一权限的用户又可以将相同权限授予其他用户 (WITH GRANT OPTION)。 grantor 已授予权限的用户的名称。 以下示例指示用户 admin 已将所有权限（包括 WITH GRANT OPTION）授予用户 dbuser。 `dbuser=r*a*w*d*x*X*/admin`

示例

以下查询返回为数据库定义的所有默认权限。

```
select pg_get_userbyid(d.defacluser) as user,
n.nspname as schema,
case d.defaclobjtype when 'r' then 'tables' when 'f' then 'functions' end
as object_type,
array_to_string(d.defaclacl, ' + ')  as default_privileges
from pg_catalog.pg_default_acl d
left join pg_catalog.pg_namespace n on n.oid = d.defaclnamespace;

 user  | schema | object_type |              default_privileges
-------+--------+-------------+-----------------------------------------------------
 admin | tickit | tables      | user1=r/admin + "group group1=a/admin" + user2=w/admin
```

上述示例中的结果表明，对于用户 admin 在 tickit schema 中创建的所有新表，admin 将 SELECT 权限授予 user1，将 INSERT 权限授予 group1，并将 UPDATE 权限授予 user2。

PG_EXTERNAL_SCHEMA

存储有关外部 schema 的信息。

PG_EXTERNAL_SCHEMA 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其具有访问权限的元数据。有关更多信息，请参阅CREATE EXTERNAL SCHEMA。

表列

列名称	数据类型	描述
esoid	oid	外部 schema ID。
eskind	integer	外部 schema 的类型。
esdbname	text	外部数据库名称。
esoptions	text	外部 schema 选项。

示例

以下示例显示有关外部 schema 的详细信息。

```
select esoid, nspname as schemaname, nspowner, esdbname as external_db, esoptions
```

```
from pg_namespace a,pg_external_schema b where a.oid=b.esoid;

esoid  | schemaname      | nspowner | external_db | esoptions

-------+-----------------+----------+-------------
+-------------------------------------------------------------
100134 | spectrum_schema |      100 | spectrum_db |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
100135 | spectrum        |      100 | spectrumdb  |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
100149 | external        |      100 | external_db |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

PG_LIBRARY

存储有关用户定义的库的信息。

PG_LIBRARY 对所有用户可见。超级用户可以查看所有行；普通用户只能查看其自己的数据。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
名称	名称	库名称。
language_oid	oid	保留供系统使用。
file_store_id	integer	保留供系统使用。
owner	integer	库所有者的用户 ID。

示例

以下示例返回用户安装的库的信息。

```
select * from pg_library;
```

```
name       | language_oid | file_store_id | owner
-----------+--------------+---------------+------
f_urlparse |       108254 |          2000 |   100
```

PG_PROC_INFO

PG_PROC_INFO 是 Amazon Redshift 系统视图，构建在 PostgreSQL 目录表 PG_PROC 和内部目录表 PG_PROC_EXTENDED 之上。PG_PROC_INFO 包括有关存储过程和函数的详细信息，这包括与输出参数（如果有）相关的信息。

表列

除了 PG_PROC 中的列以外，PG_PROC_INFO 还显示以下列。PG_PROC 中的 `oid` 列在 PG_PROC_INFO 表中称为 `prooid`。

列名称	数据类型	描述
prooid	oid	函数或存储过程的对象 ID。
prokind	"char"	指示函数或存储过程的类型的值。对于常规函数，此值为“f”，对于存储过程为“p”，对于聚合函数为“a”。
proargmodes	"char"[ ]	带有过程参数的模式的数组，其编码方式为，“i”表示 IN 参数，“o”表示 OUT 参数，“b”表示 INOUT 参数。如果所有参数均为 IN 参数，则此字段为 NULL。下标对应于 proallargtypes 数组中的位置。
proallargtypes	oid[ ]	具有过程参数的数据类型的数组。此数组包括所有参数类型（包括 OUT 和 INOUT 参数）。但是，如果所有参数均为 IN 参数，则此字段为 NULL。订阅是从 1 开始的。与之对比，proargtypes 的下标从零开始。

PG_PROC_INFO 中的 proargnames 字段包含所有类型参数（包含 OUT 和 INOUT，如有）的名称。

PG_STATISTIC_INDICATOR

存储有关自上次执行 ANALYZE 以来插入或删除的行数的信息。由于 PG_STATISTIC_INDICATOR 表在 DML 操作后频繁更新，因此统计数据是近似值。

PG_STATISTIC_INDICATOR 只对超级用户可见。有关更多信息，请参阅系统表和视图中的数据可见性。

表列

列名称	数据类型	描述
stairelid	oid	表 ID
stairows	float	表中的总行数。
staiins	float	自上次执行 ANALYZE 以来插入的行数。
staidels	float	自上次执行 ANALYZE 以来删除或更新的行数。

示例

以下示例返回自上次执行 ANALYZE 以来发生更改的表的信息。

```
select * from pg_statistic_indicator;

stairelid | stairows | staiins | staidels
----------+----------+---------+---------
   108271 |       11 |       0 |        0
   108275 |      365 |       0 |        0
   108278 |     8798 |       0 |        0
   108280 |    91865 |       0 |   100632
   108267 |    89981 |   49990 |     9999
   108269 |      808 |     606 |      374
   108282 |   152220 |   76110 |   248566
```

PG_TABLE_DEF

存储有关表列的信息。

PG_TABLE_DEF 仅返回有关对用户可见的表的信息。如果 PG_TABLE_DEF 未返回预期结果，则验证 search_path 参数是否正确设置为包含相关 schemas。

可使用 SVV_TABLE_INFO 查看有关表的更多信息，包括数据分配偏斜、密钥分配偏斜、表大小和统计数据。

表列

列名称	数据类型	描述
schemaname	名称	schema 名称。
tablename	名称	表名称。
column	名称	列名称。
type	text	列的数据类型。
encoding	character(32)	列的编码。
distkey	布尔值	如果此列为表的分配键，则为 true。
sortkey	integer	排序键中的列的顺序。如果表使用一个复合排序键，则排序键中的所有列将具有一个正值，该值指示列在排序键中的位置。如果表使用交错排序键，则排序键中的每个列将具有一个正值或负值，其中，绝对值指示列在排序键中的位置。如果为 0，则列不是排序键的一部分。
notnull	布尔值	如果列具有 NOT NULL 约束，则为 true。

示例

以下示例显示 LINEORDER_COMPOUND 表的复合排序键列。

```
select "column", type, encoding, distkey, sortkey, "notnull"
from pg_table_def
where tablename = 'lineorder_compound'
and sortkey <> 0;

column       | type    | encoding | distkey | sortkey | notnull
-------------+---------+----------+---------+---------+--------
lo_orderkey  | integer | delta32k | false   |       1 | true
lo_custkey   | integer | none     | false   |       2 | true
lo_partkey   | integer | none     | true    |       3 | true
lo_suppkey   | integer | delta32k | false   |       4 | true
```

```
lo_orderdate | integer | delta    | false   |       5 | true
(5 rows)
```

以下示例显示 LINEORDER_INTERLEAVED 表的交错排序键列。

```
select "column", type, encoding, distkey, sortkey, "notnull"
from pg_table_def
where tablename = 'lineorder_interleaved'
and sortkey <> 0;

column       | type    | encoding | distkey | sortkey | notnull
-------------+---------+----------+---------+---------+--------
lo_orderkey  | integer | delta32k | false   |      -1 | true
lo_custkey   | integer | none     | false   |       2 | true
lo_partkey   | integer | none     | true    |      -3 | true
lo_suppkey   | integer | delta32k | false   |       4 | true
lo_orderdate | integer | delta    | false   |      -5 | true
(5 rows)
```

PG_TABLE_DEF 将仅返回搜索路径中包含的 schema 中的表的信息。有关更多信息，请参阅search_path。

例如，假定您创建一个新 schema 和一个新表，然后查询 PG_TABLE_DEF。

```
create schema demo;
create table demo.demotable (one int);
select * from pg_table_def where tablename = 'demotable';

schemaname|tablename|column| type | encoding | distkey | sortkey | notnull
----------+---------+------+------+----------+---------+---------+--------
```

此查询未返回新表的行。检查 search_path 的设置。

```
show search_path;

  search_path
---------------
 $user, public
(1 row)
```

将 demo 架构添加到搜索路径并重新运行查询。

```
set search_path to '$user', 'public', 'demo';

select * from pg_table_def where tablename = 'demotable';

schemaname| tablename |column|  type   | encoding |distkey|sortkey| notnull
----------+-----------+------+---------+----------+-------+-------+--------
demo      | demotable | one  | integer | none     | f     |     0 | f
(1 row)
```

PG_USER_INFO

PG_USER_INFO 是 Amazon Redshift 系统视图，此视图显示用户信息，例如用户 ID 和密码过期时间。

只有超级用户才能看到 PG_USER_INFO。

表列

PG_USER_INFO 包含以下列。有关更多信息，请参阅 PostgreSQL 文档。

列名称	数据类型	描述
usename	name	用户名。
usesysid	integer	用户 ID。
usecreatedb	布尔值	如果用户可以创建数据库，则为 true。
usesuper	布尔值	如果用户为超级用户，则为 true。
usecatupd	布尔值	如果用户可以更新系统目录，则为 true。
passwd	text	密码。
valuntil	abstime	密码的到期日期和时间。
使用 CONFIG	text[]	会话原定设置为运行时变量。

列名称	数据类型	描述
useconnlimit	text	用户可以打开的连接数。

查询目录表

主题

- 目录查询示例

通常，可将目录表和视图（其名称以 **PG_** 开头的关系）联接到 Amazon Redshift 表和视图。

目录表使用了 Amazon Redshift 不支持的大量数据类型。当查询将目录表联接到 Amazon Redshift 表时，支持以下数据类型：

- 布尔
- "char"
- float4
- int2
- int4
- int8
- 名称
- oid
- text
- varchar

如果编写一个显式或隐式引用具有不支持的数据类型的列的联接查询，则该查询将返回一个错误。某些目录表中使用的 SQL 函数也受支持，但 PG_SETTINGS 和 PG_LOCKS 表使用的 SQL 函数除外。

例如，无法在与 Amazon Redshift 表的联接中查询 PG_STATS 表，因为存在不受支持的函数。

以下目录表和视图提供了可联接到 Amazon Redshift 表中的信息的有用信息。由于数据类型和函数限制，其中的一些表仅允许部分访问。在查询可部分访问的表时，请仔细选择或引用其列。

以下表可完全访问且不包含任何不受支持的数据类型或函数：

- pg_attribute
- pg_cast
- pg_depend
- pg_description
- pg_locks
- pg_opclass

以下表可部分访问且包含一些不受支持的类型、函数和已截断的文本列。文本列中的值将截断为 varchar(256) 值。

- pg_class
- pg_constraint
- pg_database
- pg_group
- pg_language
- pg_namespace
- pg_operator
- pg_proc
- pg_settings
- pg_statistic
- pg_tables
- pg_type
- pg_user
- pg_views

对于 Amazon Redshift 管理员，此处未列出的目录表是不可访问的或可能没有用。不过，当查询未涉及与 Amazon Redshift 表的联接时，可以查询任何目录表或视图。

可以将 Postgres 目录表中的 OID 列用作联接列。例如，联接条件 `pg_database.oid = stv_tbl_perm.db_id` 将每个 PG_DATABASE 行的内部数据库对象 ID 与 STV_TBL_PERM 表中的可见 DB_ID 列匹配。OID 列是内部主键，当您从表中选择时，这些列不可见。目录视图没有 OID 列。

某些 Amazon Redshift 函数必须只在计算节点上运行。如果查询引用用户创建的表，则 SQL 在计算节点上运行。

仅引用目录表（具有 PG 前缀的表（如 PG_TABLE_DEF））或不引用任何表的查询在领导节点上以独占方式运行。

如果使用计算节点函数的查询不引用用户定义的表，否则 Amazon Redshift 系统表会返回以下错误。

```
[Amazon](500310) Invalid operation: One or more of the used functions must be applied
 on at least one user created table.
```

以下 Amazon Redshift 函数是仅计算节点函数：

系统信息函数

- LISTAGG
- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC 和 APPROXIMATE PERCENTILE_DISC

目录查询示例

以下查询说明了用来查询目录表以获取有关 Amazon Redshift 数据库的有用信息的几种方法。

查看表 ID、数据库、架构和表名称

以下视图定义将 STV_TBL_PERM 系统表与 PG_CLASS、PG_NAMESPACE 和 PG_DATABASE 系统目录表联接以返回表 ID、数据库名称、schema 名称和表名称。

```
create view tables_vw as
select distinct(id) table_id
,trim(datname)   db_name
,trim(nspname)   schema_name
,trim(relname)   table_name
from stv_tbl_perm
join pg_class on pg_class.oid = stv_tbl_perm.id
join pg_namespace on pg_namespace.oid = relnamespace
join pg_database on pg_database.oid = stv_tbl_perm.db_id;
```

以下示例返回 ID 为 117855 的表的信息。

```
select * from tables_vw where table_id = 117855;
```

```
table_id | db_name   | schema_name | table_name
---------+-----------+-------------+-----------
  117855 |       dev | public      | customer
```

列出每个 Amazon Redshift 表的列数

以下查询联接了一些目录表，以了解每个 Amazon Redshift 表包含多少列。Amazon Redshift 表名称存储在 PG_TABLES 和 STV_TBL_PERM 中；如果可能，请使用 PG_TABLES 返回 Amazon Redshift 表名称。

此查询不涉及任何 Amazon Redshift 表。

```
select nspname, relname, max(attnum) as num_cols
from pg_attribute a, pg_namespace n, pg_class c
where n.oid = c.relnamespace and  a.attrelid = c.oid
and c.relname not like '%pkey'
and n.nspname not like 'pg%'
and n.nspname not like 'information%'
group by 1, 2
order by 1, 2;

nspname | relname  | num_cols
--------+----------+----------
public  | category |        4
public  | date     |        8
public  | event    |        6
public  | listing  |        8
public  | sales    |       10
public  | users    |       18
public  | venue    |        5
(7 rows)
```

列出数据库中的架构和表

以下查询将 STV_TBL_PERM 联接到一些 PG 表以返回 TICKIT 数据库中的表及其 schema 名称（NSPNAME 列）的列表。该查询还返回每个表中的总行数。（在系统中的多个 schemas 具有相同的表名称时，此查询很有用。）

```
select datname, nspname, relname, sum(rows) as rows
from pg_class, pg_namespace, pg_database, stv_tbl_perm
where pg_namespace.oid = relnamespace
and pg_class.oid = stv_tbl_perm.id
and pg_database.oid = stv_tbl_perm.db_id
and datname ='tickit'
group by datname, nspname, relname
order by datname, nspname, relname;

datname | nspname | relname  |  rows
--------+---------+----------+--------
tickit  | public  | category |     11
tickit  | public  | date     |    365
tickit  | public  | event    |   8798
tickit  | public  | listing  | 192497
tickit  | public  | sales    | 172456
tickit  | public  | users    |  49990
tickit  | public  | venue    |    202
(7 rows)
```

列出表 ID、数据类型、列名称和表名称

以下查询列出有关每个用户表及其列的一些信息：表 ID、表名称、表列名称和每个列的数据类型：

```
select distinct attrelid, rtrim(name), attname, typname
from pg_attribute a, pg_type t, stv_tbl_perm p
where t.oid=a.atttypid and a.attrelid=p.id
and a.attrelid between 100100 and 110000
and typname not in('oid','xid','tid','cid')
order by a.attrelid asc, typname, attname;

attrelid |  rtrim   |    attname     |  typname
---------+----------+----------------+-----------
  100133 | users    | likebroadway   | bool
  100133 | users    | likeclassical  | bool
  100133 | users    | likeconcerts   | bool
...
  100137 | venue    | venuestate     | bpchar
  100137 | venue    | venueid        | int2
  100137 | venue    | venueseats     | int4
  100137 | venue    | venuecity      | varchar
...
```

计算表中的每个列的数据块数

以下查询将 STV_BLOCKLIST 表联接到 PG_CLASS 以返回 SALES 表中列的存储信息。

```
select col, count(*)
from stv_blocklist s, pg_class p
where s.tbl=p.oid and relname='sales'
group by col
order by col;

col | count
----+-------
  0 |     4
  1 |     4
  2 |     4
  3 |     4
  4 |     4
  5 |     4
  6 |     4
  7 |     4
  8 |     4
  9 |     8
 10 |     4
 12 |     4
 13 |     8
(13 rows)
```

配置参考

主题

- enable_spectrum_oid
- spectrum_query_maxerror
- statement_timeout
- stored_proc_log_min_messages
- timezone
- use_fips_ssl
- wlm_query_slot_count

修改服务器配置

您可以通过以下方式更改服务器配置：

- 使用 SET 命令仅在当前会话持续期间重写设置。

  例如：

  ```
  set extra_float_digits to 2;
  ```

- 修改集群的参数组设置。参数组设置包括可配置的其他参数。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 参数组。
- 通过使用 ALTER USER 命令，针对指定用户运行的所有会话将某个配置参数设置为新值。

  ```
  ALTER USER username SET parameter { TO | = } { value | DEFAULT }
  ```

使用 SHOW 命令查看当前参数设置。使用 SHOW ALL 查看可使用 SET 命令配置的所有设置。

```
SHOW ALL;

name                       | setting
---------------------------+--------------
analyze_threshold_percent | 10
datestyle                  | ISO, MDY
extra_float_digits         | 2
query_group                | default
search_path                | $user, public
statement_timeout          | 0
timezone                   | UTC
```

```
wlm_query_slot_count      | 1
```

> **Note**
>
> 请注意，配置参数适用于您在数据仓库中连接到的数据库。

analyze_threshold_percent

值（默认为粗体）

10，0至 100.0

描述

设置用于分析表的已更改行数百分比的阈值。为了减少处理时间并提高整体系统性能，Amazon Redshift 将跳过对更改行数百分比低于由 analyze_threshold_percent 指定的百分比的任何表的 ANALYZE。例如，如果表包含 100000000 个行，其中 9000000 个行自上次执行 ANALYZE 后发生了更改，则默认情况下，将跳过此表，因为已更改行数的百分比低于 10%。要在仅有少量行发生更改时对表进行分析，请将 analyze_threshold_percent 设置为任意较小的数字。例如，如果您将 analyze_threshold_percent 设置为 0.01，则在含有 100000000 个行的表中至少有 10000 个行发生更改时，将不会跳过该表。要在即使没有任何行发生更改时仍对所有表进行分析，请将 analyze_threshold_percent 设置为 0。

您可以使用 SET 命令仅修改当前会话的 analyze_threshold_percent 参数。无法在参数组中修改参数。

示例

```
set analyze_threshold_percent to 15;
set analyze_threshold_percent to 0.01;
set analyze_threshold_percent to 0;
```

cast_super_null_on_error

值（默认为粗体）

on、off

描述

指定当您尝试访问对象或数组元素的不存在成员时，如果您的查询在默认宽松模式下运行，Amazon Redshift 将返回 NULL 值。

datashare_break_glass_session_var

值（默认为粗体）

无默认设置。如下所述，当执行不推荐的操作时，该值可以是 Amazon Redshift 生成的任何字符串。

描述

应用允许某些操作的权限，这些操作通常不建议用于 AWS Data Exchange 数据共享。

一般情况下，建议您不要删除或更改使用 DROP DATASHARE 或 ALTER DATASHARE SET PUBLICACCESSIBLE 语句的 AWS Data Exchange 数据共享。如需允许删除或更改 AWS Data Exchange 数据共享以禁用可公开访问的设置，请将 `datashare_break_glass_session_var` 变量设为一次性值。此一次性值由 Amazon Redshift 生成，并在首次尝试相关操作后在错误消息中提供。

将变量设置为一次性生成值后，再次运行 DROP DATASHARE 或 ALTER DATASHARE 语句。

有关更多信息，请参阅 ALTER DATASHARE 使用说明 或 DROP DATASHARE 使用说明。

示例

```
set datashare_break_glass_session_var to '620c871f890c49';
```

datestyle

值（默认为粗体）

格式规范（ISO、Postgres、SQL 或 German），以及年/月/日排序（DMY、MDY、YMD）。

- ISO – 使用 YYYY-MM-DD HH:MM:SS 的日期风格。
- Postgres – 使用 MM-DD HH:MM:SS YYYY 的日期风格。
- SQL – 使用 MM-DD-YYYY HH:MM:SS 的日期风格。
- German – 使用 DD-MM-YYYY HH:MM:SS 的日期风格。

描述

设置日期和时间值的显示格式以及解释不确定的日期输入值的规则。字符串包含两个参数，这两个参数可单独或同时更改。

示例

```
show datestyle;
DateStyle
-----------
ISO, MDY
(1 row)

set datestyle to 'SQL,DMY';
```

default_geometry_encoding

值（默认为粗体）

1、2

描述

指定是否使用边界框对此会话期间创建的空间几何体进行编码的会话配置。如果 `default_geometry_encoding` 为 `1`，则不会使用边界框对几何体进行编码。如果 `default_geometry_encoding` 为 `2`，则会使用边界框对几何体进行编码。有关对边界框的支持的更多信息，请参阅边界框。

describe_field_name_in_uppercase

值（默认为粗体）

off (false)、on (true)

描述

指定 SELECT 语句返回的列名称采用大写还是小写形式。如果此参数为 on，则返回的列名称采用大写形式。如果此参数为 off，则返回的列名称采用小写形式。Amazon Redshift 采用小写形式存储列名称，无论 `describe_field_name_in_uppercase` 的设置如何。

示例

```
set describe_field_name_in_uppercase to on;

show describe_field_name_in_uppercase;

DESCRIBE_FIELD_NAME_IN_UPPERCASE
--------------------------------
on
```

downcase_delimited_identifier

值（默认为粗体）

on、off

描述

此配置正被停用。请改用 enable_case_sensitive_identifier。

使超级解析器能够读取大写或混合大小写的 JSON 字段。此外，对具有数据库、schema、表和列混合大小写名称的受支持 PostgreSQL 数据库启用联合查询支持。要使用区分大小写的标识符，请将此参数设置为 off。

使用说明

- 如果您使用的是行级别安全性或动态数据屏蔽功能，我们建议在集群或工作组的参数组中设置 downcase_delimited_identifier 值。这可确保在创建和附加策略，然后查询应用了策略的关系的整个过程中，downcase_delimited_identifier 保持不变。有关行级别安全性的信息，请参阅行级别安全性。有关动态数据屏蔽的信息，请参阅动态数据掩蔽。
- 当您将 downcase_delimited_identifier 设置为 off 并创建表时，可以设置区分大小写的列名称。当您将 downcase_delimited_identifier 设置为 on 并查询表时，列名称会采用小写。这可能会产生与 downcase_delimited_identifier 设置为 off 时不同的查询结果。考虑以下示例：

```
SET downcase_delimited_identifier TO off;
--Amazon Redshift preserves case for column names and other identifiers.
```

```
--Create a table with two columns that are identical except for the case.
CREATE TABLE t ("c" int, "C" int);

INSERT INTO t VALUES (1, 2);

SELECT * FROM t;

 c | C
---+---
 1 | 2
(1 row)

SET enable_downcase_delimited_identifier TO on;
--Amazon Redshift no longer preserves case for column names and other identifiers.

SELECT * FROM t;

 c | c
---+---
 1 | 1
(1 row)
```

- 我们建议，查询附加了动态数据掩蔽或行级安全策略的表的常规用户使用默认的 downcase_delimited_identifier 设置。有关行级安全性的更多信息，请参阅行级别安全性。有关动态数据掩蔽的信息，请参阅动态数据掩蔽。

enable_case_sensitive_identifier

值（默认为粗体）

true，false

描述

一个配置值，用于确定数据库、表和列的名称标识符是否区分大小写。当用双引号括起来时，名称标识符的大小写将被保留。当您将 enable_case_sensitive_identifier 设置为 true 时，将保留名称标识符的大小写。当您将 enable_case_sensitive_identifier 设置为 false 时，将不保留名称标识符的大小写。

无论 enable_case_sensitive_identifier 配置选项的设置如何，系统都会保留双引号括起来的 username（用户名）示例。

示例

以下示例说明了如何为表名及列名创建和使用区分大小写的标识符。

```
-- To create and use case sensitive identifiers
SET enable_case_sensitive_identifier TO true;

-- Create tables and columns with case sensitive identifiers
CREATE TABLE "MixedCasedTable" ("MixedCasedColumn" int);

CREATE TABLE MixedCasedTable (MixedCasedColumn int);

-- Now query with case sensitive identifiers
SELECT "MixedCasedColumn" FROM "MixedCasedTable";

MixedCasedColumn
------------------
(0 rows)


SELECT MixedCasedColumn FROM MixedCasedTable;

mixedcasedcolumn
------------------
(0 rows)
```

以下示例显示何时不保留标识符大小写。

```
-- To not use case sensitive identifiers
RESET enable_case_sensitive_identifier;

-- Mixed case identifiers are lowercased
CREATE TABLE "MixedCasedTable2" ("MixedCasedColumn" int);

CREATE TABLE MixedCasedTable2 (MixedCasedColumn int);

ERROR:  Relation "mixedcasedtable2" already exists


SELECT "MixedCasedColumn" FROM "MixedCasedTable2";

 mixedcasedcolumn
------------------
```

```
(0 rows)


SELECT MixedCasedColumn FROM MixedCasedTable2;

 mixedcasedcolumn
------------------
(0 rows)
```

使用说明

- 如果您为实体化视图使用自动刷新，我们建议您在集群或工作组的参数组中设置 enable_case_sensitive_identifier 值。这样可以确保在实体化视图刷新时 enable_case_sensitive_identifier 保持不变。有关实体化视图自动刷新的信息，请参阅刷新实体化视图。有关在参数组中设置配置值的信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 参数组。
- 如果您使用的是行级别安全性或动态数据屏蔽功能，我们建议在集群或工作组的参数组中设置 enable_case_sensitive_identifier 值。这可确保在创建和附加策略，然后查询应用了策略的关系的整个过程中，enable_case_sensitive_identifier 保持不变。有关行级别安全性的信息，请参阅行级别安全性。有关动态数据屏蔽的信息，请参阅动态数据掩蔽。
- 当您将 enable_case_sensitive_identifier 设置为 on 并创建表时，可以设置区分大小写的列名称。当您将 enable_case_sensitive_identifier 设置为 off 并查询表时，列名称会采用小写。这可能会产生与 enable_case_sensitive_identifier 设置为 on 时不同的查询结果。考虑以下示例：

```
SET enable_case_sensitive_identifier TO on;
--Amazon Redshift preserves case for column names and other identifiers.

--Create a table with two columns that are identical except for the case.
CREATE TABLE t ("c" int, "C" int);

INSERT INTO t VALUES (1, 2);

SELECT * FROM t;

 c | C
---+---
 1 | 2
(1 row)
```

```
SET enable_case_sensitive_identifier TO off;
--Amazon Redshift no longer preserves case for column names and other identifiers.

SELECT * FROM t;

 c | c
---+---
 1 | 1
(1 row)
```

- 我们建议，查询附加了动态数据掩蔽或行级安全策略的表的常规用户使用默认的 enable_case_sensitive_identifier 设置。有关行级别安全性的信息，请参阅行级别安全性。有关动态数据掩蔽的信息，请参阅动态数据掩蔽。

enable_case_sensitive_super_attribute

值（默认为粗体）

true，false

描述

一个配置值，用于确定导航具有非分隔属性名称的 SUPER 数据类型结构时是否区分大小写。将 enable_case_sensitive_super_attribute 设置为 true 时，导航具有非分隔属性名称的 SUPER 类型结构时区分大小写。将该值设置为 false 时，导航具有非分隔属性名称的 SUPER 类型结构时不区分大小写。

当您使用双引号括起属性名并将 enable_case_sensitive_identifier 设置为 true 时，将始终保留大小写，不论如何设置 enable_case_sensitive_super_attribute 配置选项。

enable_case_sensitive_super_attribute 仅适用于 SUPER 数据类型的列。对于所有其他列，请考虑改为使用 enable_case_sensitive_identifier。

有关 SUPER 数据类型的更多信息，请参阅 SUPER 类型和在 Amazon Redshift 中摄取和查询半结构化数据。

示例

以下示例显示了在启用和禁用 enable_case_sensitive_super_attribute 的情况下，选择 SUPER 值的结果。

```
--Create a table with a SUPER column.
CREATE TABLE tbl (col SUPER);

--Insert values.
INSERT INTO tbl VALUES (json_parse('{
 "A": "A", "a": "a"
}'));

SET enable_case_sensitive_super_attribute TO ON;

SELECT col.A FROM tbl;
  a
-----
 "A"
(1 row)

SELECT col.a FROM tbl;
  a
-----
 "a"
(1 row)

SET enable_case_sensitive_super_attribute TO OFF;

SELECT col.A FROM tbl;
  a
-----
 "a"
(1 row)

SELECT col.a FROM tbl;
  a
-----
 "a"
(1 row)
```

使用说明

- 视图和实体化视图遵循其创建时的 enable_case_sensitive_super_attribute 值。后期绑定视图、存储过程和用户定义的函数遵循查询时的 enable_case_sensitive_super_attribute 值。

- 如果您为实体化视图使用自动刷新，我们建议您在集群或工作组的参数组中设置 enable_case_sensitive_identifier value。这样可以确保在实体化视图刷新时 enable_case_sensitive_identifier 保持不变。有关实体化视图自动刷新的信息，请参阅刷新实体化视图。有关在参数组中设置配置值的信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 参数组。
- 无论 enable_case_sensitive_super_attribute 的值如何，语句结果中的列名始终为小写。要使列名也区分大小写，请启用 enable_case_sensitive_identifier。
- 我们建议，查询附加了行级安全策略的表的常规用户使用默认的 enable_case_sensitive_identifier 设置。有关行级安全性的更多信息，请参阅行级别安全性。

enable_numeric_rounding

值（默认为粗体）

on（true），off（false）

描述

指定是否使用数值四舍五入。如果 enable_numeric_rounding 为 on，则 Amazon Redshift 在将 NUMERIC 值强制转换为其他数值类型（例如 INTEGER 或 DECIMAL）时会四舍五入。如果 enable_numeric_rounding 为 off，则 Amazon Redshift 会在将 NUMERIC 值强制转换为其他数值类型时将其截断。有关数值类型的更多信息，请参阅数字类型。

示例

```
--Create a table and insert the numeric value 1.5 into it.
CREATE TABLE t (a numeric(10, 2));

INSERT INTO t VALUES (1.5);

SET enable_numeric_rounding to ON;
--Amazon Redshift now rounds NUMERIC values when casting to other numeric types.

SELECT a::int FROM t;

 a
---
 2
```

```
(1 row)


SELECT a::decimal(10, 0) FROM t;

 a
---
 2
(1 row)


 SELECT a::decimal(10, 5) FROM t;

    a
---------
 1.50000
(1 row)


SET enable_numeric_rounding to OFF;
--Amazon Redshift now truncates NUMERIC values when casting to other numeric types.

SELECT a::int FROM t;

 a
---
 1
(1 row)


SELECT a::decimal(10, 0) FROM t;

 a
---
 1
(1 row)


SELECT a::decimal(10, 5) FROM t;

    a
---------
 1.50000
```

```
(1 row)
```

enable_result_cache_for_session

值（默认为粗体）

on (true)、off (false)

描述

指定是否使用查询结果缓存。如果 enable_result_cache_for_session 是 on，则在提交查询时，Amazon Redshift 会检查有无查询结果的有效缓存副本。如果在结果缓存中找到匹配项，Amazon Redshift 会使用缓存的结果而不执行查询。如果 enable_result_cache_for_session 是 off，则 Amazon Redshift 会忽略结果缓存，并在所有查询提交时运行查询。

示例

```
SET enable_result_cache_for_session TO off;
--Amazon Redshift now ignores the results cache
```

enable_vacuum_boost

值（默认为粗体）

false, true

描述

指定是否为会话中运行的所有 VACUUM 命令启用 vacuum boost 选项。如果 enable_vacuum_boost 为 true，Amazon Redshift 会在会话中使用 BOOST 选项运行所有 VACUUM 命令。如果 enable_vacuum_boost 为 false，则预设情况下，Amazon Redshift 不会使用 BOOST 选项运行。有关 BOOST 选项的更多信息，请参阅VACUUM。

error_on_nondeterministic_update

值（默认为粗体）

false, true

描述

指定每行具有多个匹配项的 UPDATE 查询是否返回错误。

示例

```
SET error_on_nondeterministic_update TO true;

CREATE TABLE t1(x1 int, y1 int);

CREATE TABLE t2(x2 int, y2 int);

INSERT INTO t1 VALUES (1,10), (2,20), (3,30);

INSERT INTO t2 VALUES (2,40), (2,50);

UPDATE t1 SET y1=y2 FROM t2 WHERE x1=x2;

ERROR: Found multiple matches to update the same tuple.
```

extra_float_digits

值（默认为粗体）

0、-15到 2

描述

设置所显示的浮点值位数（包括 float4 和 float8）。该值将添加到标准位数（FLT_DIG 或 DBL_DIG，根据需要）。该值最高可设置为 2 以包括部分有效数字。这对于需要准确还原的输出浮点数据特别有用。也可以将它设置为负以隐藏不需要的位数。

示例

以下示例将 `extra_float_digits` 设置为 `-2`。首先，显示当前的参数设置。

```
show all;
      name                      | setting
--------------------------+----------------
```

```
analyze_threshold_percent | 10
datestyle                 | ISO, MDY
extra_float_digits        | 2
query_group               | default
search_path               | $user, public
statement_timeout         | 0
timezone                  | UTC
wlm_query_slot_count      | 1
```

然后，将新值设置为 -2。

```
set extra_float_digits to -2;
```

最后显示更新的参数设置。

```
show all;
      name                      | setting
--------------------------+----------------
analyze_threshold_percent | 10
datestyle                 | ISO, MDY
extra_float_digits        | -2
query_group               | default
search_path               | $user, public
statement_timeout         | 0
timezone                  | UTC
wlm_query_slot_count      | 1
```

interval_forbid_composite_literals

值（默认为粗体）

false、true

描述

一种会话配置，该配置可修改同时包含 YEAR TO MONTH 和 DAY TO SECOND 部分的间隔值。

如果 interval_forbid_composite_literals 为 true，则如果遇到同时包含 YEAR TO MONTH 和 DAY TO SECOND 部分的间隔，则会返回错误。例如，以下 SQL 包含 INTERVAL DAY TO SECOND，其中同时包含 YEAR TO MONTH 和 DAY TO SECOND 部分。

```
SELECT INTERVAL '1 year 1 day' DAY TO SECOND;
ERROR:  Interval Day To Second literal cannot contain year-month parts. Disable the GUC
 interval_forbid_composite_literals to suppress this error and silently discard the
 year-month part.
```

如果 interval_forbid_composite_literals 为 false，Amazon Redshift 会抑制错误，并从 INTERVAL DAY TO SECOND 值处截断 YEAR TO MONTH 部分。例如，以下 SQL 包含 INTERVAL DAY TO SECOND，其中同时包含 YEAR TO MONTH 和 DAY TO SECOND 部分。

```
SET interval_forbid_composite_literals to "false";
SELECT INTERVAL '1 year 1 day' DAY TO SECOND;


 intervald2s
------------------------------
1 days 0 hours 0 mins 0.0 secs
```

json_serialization_enable

值（默认为粗体）

false, true

描述

一种会话配置，用于修改 ORC、JSON、Ion 和 Parquet 格式化数据的 JSON 序列化行为。如果 json_serialization_enable 为 true，所有顶级集合都会自动序列化为 JSON 格式并以 VARCHAR(65535) 形式返回。非复杂列不会受到影响或被序列化。由于集合列被序列化为 VARCHAR(65535)，因此不能再将其嵌套子字段作为查询语法的一部分直接访问（即在筛选器子句中）。如果 json_serialization_enable 为 false，顶级集合不会序列化为 JSON。有关嵌套 JSON 序列化的更多信息，请参阅序列化复杂嵌套 JSON。

json_serialization_parse_nested_strings

值（默认为粗体）

false, true

描述

一种会话配置，用于修改 ORC、JSON、Ion 和 Parquet 格式化数据的 JSON 序列化行为。当 `json_serialization_parse_nested_strings` 和 `json_serialization_enable` 均为 true 时，存储在复杂类型（如映射、结构或数组）中的字符串值将被解析并直接将其写入结果中（如果它们是有效的 JSON）。如果 `json_serialization_parse_nested_strings` 为 false，则嵌套复杂类型中的字符串将被序列化为转义 JSON 字符串。有关更多信息，请参阅 序列化包含 JSON 字符串的复杂类型。

max_concurrency_scaling_clusters

值（默认为粗体）

1，0至 10

描述

设置启用并发扩展时允许的最大并发扩展集群数。如果需要更多的并发扩展，请增大该值。减小该值可减少并发扩展集群的使用率以及由此产生的账单费用。

并发扩展集群的最大数量是一个可调整的配额。有关更多信息，请参阅《Amazon Redshift 管理指南》中的 Amazon Redshift 配额。

max_cursor_result_set_size

值（默认为粗体）

0（默认为最大值）- 14400000 MB

描述

不再使用 `max_cursor_result_set_size` 参数。有关游标结果集大小的更多信息，请参阅游标约束。

mv_enable_aqmv_for_session

值（默认为粗体）

true，false

描述

指定 Amazon Redshift 是否可以在会话级别执行实体化视图的自动查询重写。

navigate_super_null_on_error

值（默认为粗体）

on、off

描述

指定当您尝试导航对象或数组元素的不存在成员时，如果您的查询在默认宽松模式下运行，Amazon Redshift 将返回 NULL 值。

parse_super_null_on_error

值（默认为粗体）

off, on

描述

指定当 Amazon Redshift 尝试解析对象或数组元素中不存在的成员时，如果您的查询在严格模式下运行，Amazon Redshift 将返回 NULL 值。

pg_federation_repeatable_read

值（默认为粗体）

true，false

描述

为来自 PostgreSQL 数据库的结果指定联合查询事务隔离级别。

- 如果 `pg_federation_repeatable_read` 为 true，则使用 REPEATABLE READ 隔离级别语义处理联合事务。这是默认模式。

- 如果 pg_federation_repeatable_read 为 false，则使用 READ COMMITTED 隔离级别语义处理联合事务。

有关更多信息，请参阅下列内容：

- 使用 Amazon Redshift 访问联合数据时的注意事项.
- 管理并发写入操作.

示例

以下命令将会话的 pg_federation_repeatable_read 设置为 on。show 命令显示设定值的值。

```
set pg_federation_repeatable_read to on;
```

```
show pg_federation_repeatable_read;

pg_federation_repeatable_read
-----------------------------
on
```

query_group

值（默认为粗体）

无默认值；该值可以是任何字符串。

描述

将用户定义的标签应用于在同一会话期间运行的一组查询。此标签在查询日志中捕获。您可以使用它来约束 STL_QUERY 和 STV_INFLIGHT 表以及 SVL_QLOG 视图中的结果。例如，您可以将一个单独的标签应用于您运行的每个查询以唯一标识查询，而不必查找其 ID。

此参数在服务器配置文件中不存在，必须使用 SET 命令在运行时设置此参数。虽然可使用长字符串作为标签，在 STL_QUERY 表和 SVL_QLOG 视图的 LABEL 列中，标签仍将被截断至 30 个字符（在 STV_INFLIGHT 中将被截断至 15 个字符）。

在以下示例中，query_group 将被设置为 **Monday**，然后使用该标签执行多个查询。

```
set query_group to 'Monday';
SET
select * from category limit 1;
...
...
select query, pid, substring, elapsed, label
from svl_qlog where label ='Monday'
order by query;

query | pid  |        substring                    |  elapsed  | label
------+------+-------------------------------------+-----------+--------
789   | 6084 | select * from category limit 1;     | 65468     | Monday
790   | 6084 | select query, trim(label) from ... | 1260327   | Monday
791   | 6084 | select * from svl_qlog where ..     | 2293547   | Monday
792   | 6084 | select count(*) from bigsales;      | 108235617 | Monday
...
```

search_path

值（默认为粗体）

'$user'、public、schema_names

现有 schema 名称的逗号分隔的列表。如果 '$user' 存在，则将替换与 SESSION_USER 同名的 schema，否则它将被忽略。

描述

指定不带 schema 组件的简单名称引用对象（例如表或函数）时搜索 schema 的顺序。

- 外部 schema 和外部表不支持搜索路径。外部表必须通过一个外部 schema 明确地进行限定。
- 如果在没有特定目标 schema 的情况下创建对象，则对象将被置于搜索路径中列出的第一个 schema 中。如果搜索路径为空，系统将返回错误。
- 如果不同的 schema 中存在具有相同名称的对象，则使用搜索路径中找到的第一个对象。
- 仅可通过使用合格的（用点分隔）名称指定其包含 schema 来引用未存在于搜索路径的任何 schema 中的对象。
- 将始终搜索系统目录 schema pg_catalog。如果路径中提到了该 schema，则将按指定顺序搜索该 schema。否则，将在任何路径项目之前搜索该 schema。

- 如果存在当前会话的临时表 schema pg_temp_nnn，则将始终搜索该 schema。可使用别名 pg_temp 在路径中明确列出该 schema。如果未在路径中列出该 schema，则将首先搜索该 schema（甚至在 pg_catalog 之前）。但是，仅在临时 schema 中搜索关系名称（表、视图）。不在临时 schema 中搜索函数名称。

示例

以下示例将创建 schema ENTERPRISE 并设置到新 schema 的 search_path。

```
create schema enterprise;
set search_path to enterprise;
show search_path;

 search_path
-------------
 enterprise
(1 row)
```

以下示例将 schema ENTERPRISE 添加到默认 search_path。

```
set search_path to '$user', public, enterprise;
show search_path;

         search_path
-----------------------------
 "$user", public, enterprise
(1 row)
```

以下示例将表 FRONTIER 添加到 schema ENTERPRISE。

```
create table enterprise.frontier (c1 int);
```

如果在相同的数据库中创建表 PUBLIC.FRONTIER，并且用户未在查询中指定 schema 名称，则 PUBLIC.FRONTIER 优先于 ENTERPRISE.FRONTIER。

```
create table public.frontier(c1 int);
insert into enterprise.frontier values(1);
select * from frontier;

frontier
```

```
----
(0 rows)

select * from enterprise.frontier;

c1
----
1
(1 row)
```

spectrum_enable_pseudo_columns

值（默认为粗体）

true，false

描述

您可以通过将 spectrum_enable_pseudo_columns 配置参数设置为 false 来禁用为会话创建伪列的功能。

示例

以下命令可禁用为会话创建伪列的功能。

```
set spectrum_enable_pseudo_columns to false;
```

enable_spectrum_oid

值（默认为粗体）

true，false

描述

您还可以通过将 enable_spectrum_oid 配置参数设置为 false 来仅禁用 $spectrum_oid 伪列。

示例

以下命令通过将 enable_spectrum_oid 配置参数设置为 false 来禁用 $spectrum_oid 伪列。

```
set enable_spectrum_oid to false;
```

spectrum_query_maxerror

值（默认为粗体）

–1，整数

描述

您可以在取消查询之前指定一个要接受的最大错误整数。负值会关闭最大错误数据处理。结果已记录在 SVL_SPECTRUM_SCAN_ERROR 中。

示例

以下示例代入了包含多余字符和无效字符的 ORC 数据。my_string 的列定义指定 3 个字符的长度。以下是此示例的示例数据：

```
my_string
---------------
abcdef
gh
ab
```

以下命令将最大错误数设置为 1 并执行查询。

```
set spectrum_query_maxerror to 1;
SELECT my_string FROM orc_data;
```

查询停止且结果已记录在 SVL_SPECTRUM_SCAN_ERROR 中。

statement_timeout

值（默认为粗体）

0（关闭限制）、x 毫秒

描述

停止接管指定毫秒数的任何语句。

`statement_timeout` 值表示查询在由 Amazon Redshift 终止前可运行的最长时间。此时间包含规划时间、工作负载管理 (WLM) 中的排队时间和执行时间。将此时间与 WLM 超时 (max_execution_time) 和 QMR (query_execution_time)（仅包含执行时间）进行比较。

如果也指定了 WLM 超时 (max_execution_time) 作为 WLM 配置的一部分，则使用 statement_timeout 和 max_execution_time 中较小者。有关更多信息，请参阅 WLM 超时。

示例

由于以下查询需要的时间超过 1 毫秒，因此它将超时且将被取消。

```
set statement_timeout = 1;

select * from listing where listid>5000;
ERROR:  Query (150) canceled on user's request
```

stored_proc_log_min_messages

值（默认为粗体）

LOG、INFO、NOTICE、WARNING、EXCEPTION

描述

指定引发的存储过程消息的最低日志记录级别。将记录指定级别或更高级别的消息。默认值为 LOG（记录所有消息）。日志级别从最高到最低的顺序如下：

1. EXCEPTION
2. WARNING
3. NOTICE
4. INFO
5. LOG

例如，如果您指定 NOTICE 值，则仅记录 NOTICE、WARNING 和 EXCEPTION 的消息。

timezone

值（默认为粗体）

UTC，时区

语法

```
SET timezone { TO | = } [ time_zone | DEFAULT ]

SET time zone [ time_zone | DEFAULT ]
```

描述

设置当前会话的时区。时区可以为协调世界时间 (UTC) 或时区名称的偏移。

> **Note**
>
> 您不能使用集群参数组设置 timezone 配置参数。使用 SET 命令只能为当前会话设置时区。要为某个特定数据库用户运行的所有会话设置时区，请使用 ALTER USER 命令。ALTER USER … SET TIMEZONE 将更改后续会话的时区，而不是更改当前会话的时区。

在使用带有 TO 或 = 的 SET timezone（一个词）命令时，您可以将 time_zone 指定为时区名称、POSIX 样式的格式偏移或 ISO-8601 格式偏移，如下所示。

```
SET timezone { TO | = } time_zone
```

在使用有关此行为的更多信息，请参阅没有 TO 或 = 的 SET time zone 命令时，您可以使用 INTERVAL 指定 time_zone，同时还能指定时区名称、POSIX 样式的格式偏移或 ISO-8601 格式偏移，如下所示。

```
SET time zone time_zone
```

时区格式

Amazon Redshift 支持以下时区格式：

- 时区名称
- INTERVAL
- POSIX 式时区规格
- ISO-8601 偏移

由于时区缩写，例如 PST 或 PDT，定义为从 UTC 的固定偏移，并且不包括夏令时时间规则，所以 SET 命令不支持时区缩写。

有关时区格式的详细信息，请参阅以下内容。

时区名称 – 完整时区名称，如 America/New_York。完整时区名称可以包含夏令时规则。

以下是完整时区名称的示例：

- Etc/Greenwich
- America/New_York
- CST6CDT
- GB

> **Note**
>
> 许多时区名称，例如 EST、MST、NZ 和 UCT，也是缩写。

要查看有效时区名称的列表，请运行以下命令。

```
select pg_timezone_names();
```

INTERVAL – UTC 的偏移。例如，PST 是 –8:00 或 –8 小时。

以下是 INTERVAL 时区偏移的示例：

- -8:00
- –8 小时
- 30 分钟

POSIX 样式的格式 – STDoffset 或 STDoffsetDST 形式的时区规范，其中 STD 是时区缩写，offset 是从 UTC 向西的小时数偏移，而 DST 是可选的夏令时区缩写。夏令时时间假定为比给定的偏移提前一个小时。

POSIX 样式的时区格式使用格林威治以西的正偏移，而 ISO-8601 约定则使用格林威治以东的正偏移量。

以下是 POSIX 样式时区的示例：

- PST8
- PST8PDT
- EST5
- EST5EDT

> **Note**
>
> Amazon Redshift 不验证 POSIX 样式时区规格，因此可能将时区设置为无效值。例如，即使将时区设置成了无效值，以下命令也没有返回错误。
>
> ```
> set timezone to ‘xxx36’;
> ```

ISO-8601 偏移 – ±[hh]:[mm] 形式的 UTC 偏移。

下面是 ISO-8601 偏移的示例：

- -8:00
- +7:30

示例

以下示例将当前会话的时区设置为 New York。

```
set timezone = 'America/New_York';
```

以下示例将当前会话的时区设置为 UTC–8 (PST)。

```
set timezone to '-8:00';
```

以下示例使用 INTERVAL 将时区设置为 PST。

```
set timezone interval '-8 hours'
```

以下示例将当前会话的时区重新设置为系统默认时区 (UTC)。

```
set timezone to default;
```

要为数据库用户设置时区，请使用 ALTER USER … SET 语句。以下示例将 dbuser 的时区设置为 New York。在所有后续会话中将为该用户保留新值。

```
ALTER USER dbuser SET timezone to 'America/New_York';
```

use_fips_ssl

值（默认为粗体）

true，false

描述

一个参数组值，用于指定是否使用符合 FIPS 的 SSL 模式。如果 `use_fips_ssl` 为 `true`，则使用符合 FIPS 的 SSL 模式。如果 `use_fips_ssl` 为 `false`，则不使用符合 FIPS 的 SSL 模式。有关更多信息，请参阅《Amazon Redshift 管理指南》中的配置连接的安全选项。

要为 Amazon Redshift 预调配集群配置参数，请参阅《Amazon Redshift 管理指南》中的关于参数组。要配置 Redshift Serverless 的参数，请参阅《Amazon Redshift 管理指南》中的配置与 Amazon Redshift Serverless 的符合 FIPS 的 SSL 连接以及《Redshift Serverless API 参考》中的 CreateWorkgroup 或 UpdateWorkgroup。

wlm_query_slot_count

值（默认为粗体）

1、1到 50（不能超过服务类的可用槽数（并发级别））

描述

设置查询使用的查询槽的数目。

工作负载管理 (WLM) 在根据为查询设置的并发级别在服务类中预留槽位。例如，如果并发级别设置为 5，则服务类具有 5 个槽位。WLM 会将服务类的可用内存均匀分配给每个槽。有关更多信息，请参阅实施工作负载管理。

> **Note**
>
> 如果 wlm_query_slot_count 的值大于服务类的可用槽数（并发级别），则查询将失败。如果您遇到错误，请将 wlm_query_slot_count 减少到允许的值。

对于性能受已分配的内存严重影响的操作（例如 Vacuuming），提高 wlm_query_slot_count 的值可以提高性能。尤其是对于较慢的 Vacuum 命令，应检查 SVV_VACUUM_SUMMARY 视图中的相应记录。如果您看到 SVV_VACUUM_SUMMARY 视图中 sort_partitions 和 merge_increments 的值较高（接近或高于 100），请考虑在您下次对该表运行 Vacuum 时提高 wlm_query_slot_count 的值。

提高 wlm_query_slot_count 的值将限制可运行的并发查询的数量。例如，假设服务类的并发级别为 5 且 wlm_query_slot_count 设置为 3。在 wlm_query_slot_count 设置为 3 的情况下，在会话中运行查询时，同一服务类中最多可运行 2 个并发查询。后续查询将排队等待，直至当前运行的查询完成且槽被释放。

示例

使用 SET 命令设置当前会话的持续期间的 wlm_query_slot_count 的值。

```
set wlm_query_slot_count to 3;
```

文档历史记录

> **Note**
>
> 有关 Amazon Redshift 中的新特征的描述，请参阅新增特征。

下表描述了在 2018 年 5 月后对《Amazon Redshift 数据库开发人员指南》进行的重要文档更改。如需对此文档更新的通知，您可以订阅 RSS 源。

API 版本：2012-12-01

有关对《Amazon Redshift 管理指南》进行的更改的列表，请参阅 Amazon Redshift 管理指南文档历史记录。

有关新功能的更多信息（包括每个版本的修复和关联的集群版本号的列表），请参阅集群版本历史记录。

变更	说明	日期
支持空间 3D 和 4D 几何体以及新的空间函数	现在，您可以使用其它空间函数，且某些函数已添加 3D 和 4D 几何体支持。	2021 年 8 月 19 日
支持列压缩编码以实现自动表优化	您可以为表指定 ENCODE AUTO 选项，以自动管理表中所有列的压缩编码。	2021 年 8 月 3 日
使用 Amazon Redshift Data API 支持多个 SQL 语句或带参数的 SQL 语句	现在，您可以使用 Amazon Redshift Data API 运行多个 SQL 语句或带参数的语句。	2021 年 7 月 28 日
支持使用列级别覆盖的不区分大小写的排序规则	现在，您可以在 CREATE DATABASE 语句中使用 COLLATE 子句来指定默认排序规则。	2021 年 6 月 24 日

支持跨账户数据共享	您现在可以跨 AWS 账户 共享数据。	2021 年 4 月 30 日
支持使用递归 CTE 的分层数据查询	现在，您可以在 SQL 中使用递归公用表表达式（CTE）。	2021 年 4 月 29 日
支持跨数据库查询	现在，您可以在集群数据库中查询数据。	2021 年 3 月 10 日
支持 COPY 和 UNLOAD 命令的访问权限精细控制	现在，您可以向 Amazon Redshift 集群中的特定用户和组授予运行 COPY 和 UNLOAD 命令的权限，以创建更精细的访问控制策略。	2021 年 1 月 12 日
支持本机 JSON 和半结构化数据	现在，您可以定义 SUPER 数据类型。	2020 年 12 月 9 日
支持对 MySQL 的联合查询	现在，您可以将联合查询写入受支持的 MySQL 引擎。	2020 年 12 月 9 日
支持数据共享	您现在可以跨 Amazon Redshift 集群共享数据。	2020 年 12 月 9 日
支持自动表优化	您现在可以定义自动分配和排序键。	2020 年 12 月 9 日
支持 Amazon Redshift ML	现在，您可以创建、训练和部署机器学习 (ML) 模型。	2020 年 12 月 8 日
支持具体化视图的自动刷新和查询重写	现在，您可以通过自动刷新使具体化视图保持最新状态，并且可以通过自动重写提高查询性能。	2020 年 11 月 11 日

支持 TIME 和 TIMETZ 数据类型	现在，您可以使用 TIME 和 TIMETZ 数据类型创建表。TIME 数据类型存储一天中的时间，不包含时区信息，TIMETZ 存储一天中的时间，包括时区信息	2020 年 11 月 11 日
支持 Lambda UDF 和令牌化	您现在可以编写 Lambda UDF 来启用数据的外部令牌化。	2020 年 10 月 26 日
支持更改表列编码	您现在可以更改表列编码。	2020 年 10 月 20 日
支持跨数据库查询	Amazon Redshift 可以跨集群中的数据库进行查询。	2020 年 10 月 15 日
支持S HyperLogLog 草图	Amazon Redshift 现在可以存储和处理 HyperLogLogSketches。	2020 年 10 月 2 日
支持 Apache Hudi 和 Delta Lake	为 Redshift Spectrum 创建外部表的增强功能。	2020 年 9 月 24 日
支持查询空间数据的增强功能	增强功能包括加载 shapefile 和多个新的空间 SQL 函数。	2020 年 9 月 15 日
具体化视图支持外部表	您可以在 Amazon Redshift 中创建引用外部数据源的具体化视图。	2020 年 6 月 19 日
支持写入外部表	可以通过运行 CREATE EXTERNAL TABLE AS SELECT 来写入新的外部表，或者运行 INSERT INTO 将数据插入到现有外部表中，从而写入外部表。	2020 年 6 月 8 日
支持 schema 的存储控制	管理 schema 存储控制的命令和视图的更新。	2020 年 6 月 2 日

支持联合查询通用版	更新了有关使用联合查询查询数据的信息。	2020 年 4 月 16 日
支持额外的空间函数	添加了其他空间函数的描述。	2020 年 4 月 2 日
支持具体化视图一般可用性	具体化视图通常从集群版本 1.0.13059 开始可用。	2020 年 2 月 19 日
支持列级权限	从集群版本 1.0.13059 开始，可以使用列级权限。	2020 年 2 月 19 日
ALTER TABLE	您可以使用 ALTER TABLE 命令与 ALTER DISTSTYLE ALL 子句来更改表的分配样式。	2020 年 2 月 11 日
支持联合查询	更新了指南，以描述使用更新的 CREATE EXTERNAL SCHEMA 进行的联合查询。	2019 年 12 月 3 日
支持数据湖导出	更新了指南以描述 UNLOAD 命令的新参数。	2019 年 12 月 3 日
对空间数据的支持	更新了指南以描述对空间数据的支持。	2019 年 11 月 21 日
对新控制台的支持	更新了指南以描述新的 Amazon Redshift 控制台。	2019 年 11 月 11 日
对自动表排序的支持	Amazon Redshift 可以自动对表数据进行排序。	2019 年 11 月 7 日
对 VACUUM BOOST 选项的支持	在对表进行 vacuum 操作时，可以使用 BOOST 选项。	2019 年 11 月 7 日
支持默认 IDENTITY 列	您可以创建包含默认 IDENTITY 列的表。	2019 年 9 月 19 日
支持 AZ64 压缩编码	您可以使用 AZ64 压缩编码为部分列进行编码。	2019 年 9 月 19 日

支持查询优先级	您可以设置自动 WLM 队列的查询优先级。	2019 年 8 月 22 日
对该项的支持AWS Lake Formation	您可以将 Lake Formation Data Catalog 与 Amazon Redshift Spectrum 结合使用。	2019 年 8 月 8 日
COMPUPDATE PRESET	您可以将 COPY 命令与 COMPUPDATE PRESET 结合使用，以使 Amazon Redshift 能够选择压缩编码。	2019 年 6 月 13 日
ALTER COLUMN	您可以将 ALTER TABLE 命令与 ALTER COLUMN 一起使用以增加 VARCHAR 列大小。	2019 年 5 月 22 日
对存储过程的支持	您可以在 Amazon Redshift 中定义 PL/pgSQL 存储过程。	2019 年 4 月 24 日
支持自动工作负载管理 (WLM) 配置	您可以允许 Amazon Redshift 在运行时具有自动 WLM。	2019 年 4 月 24 日
卸载到 Zstandard	您可以使用 UNLOAD 命令将 Zstandard 压缩应用于卸载到 Amazon S3 的文本和逗号分隔值 (CSV) 文件。	2019 年 4 月 3 日
并发扩展	启用并发扩展后，Amazon Redshift 会在需要时自动增加额外的集群容量来处理增多的并发读取查询。	2019 年 3 月 21 日
UNLOAD 到 CSV	您可以使用 UNLOAD 命令卸载到格式为 CSV 文本的文件。	2019 年 3 月 13 日

AUTO 分配方式	要启用自动分配，可以使用 CREATE TABLE 语句指定 AUTO 分配方式。当您启用自动分配时，Amazon Redshift 基于表数据分配最佳分配方式。几秒钟后，分配更改将在后台进行。	2019 年 1 月 23 日
Parquet 中的 COPY 支持 SMALLINT	COPY 现在支持从 Parquet 格式文件加载到使用 SMALLINT 数据类型的列。有关更多信息，请参阅列式数据格式中的 COPY。	2019 年 1 月 2 日
DROP EXTERNAL DATABASE	您可以通过将 DROP EXTERNAL DATABASE 子句与 DROP SCHEMA 命令包含在一起来删除外部数据库。	2018 年 12 月 3 日
跨区域 UNLOAD	可以通过指定 REGION 参数来 UNLOAD 到另一个 AWS 区域中的 Amazon S3 存储桶。	2018 年 10 月 31 日
自动 vacuum 删除	Amazon Redshift 在后台自动执行 VACUUM DELETE 操作，因此，您很少需要运行 DELETE ONLY vacuum。Amazon Redshift 安排 VACUUM DELETE 在负载减少期间运行，并在高负载期间暂停操作。	2018 年 10 月 31 日
自动分配	在未指定带 CREATE TABLE 语句的分配方式时，Amazon Redshift 会根据表数据分配最佳分配方式。几秒钟后，分配更改将在后台进行。	2018 年 10 月 31 日

中的精细访问控制AWS Glue Data Catalog	您现在可以指定存储在 AWS Glue Data Catalog 中的数据的访问级别。	2018 年 10 月 15 日
使用数据类型执行 UNLOAD	您可以使用 UNLOAD 命令指定 MANIFEST VERBOSE 选项来将元数据添加到清单文件，包括列的名称和数据类型、文件大小和行计数。	2018 年 10 月 10 日
使用单个 ALTER TABLE 语句添加多个分区	对于 Redshift Spectrum 外部表，您可以在单个 ALTER TABLE ADD 语句中组合多个 PARTITION 子句。有关更多信息，请参阅修改外部表示例。	2018 年 10 月 10 日
具有标题的 UNLOAD	您可以指定具有 UNLOAD 命令的 HEADER 选项添加标题行，其中在每个输出文件顶部包含列名称。	2018 年 9 月 19 日
新系统表和视图	已添加 SVL_S3Retries、SVL_USER_INFO 和 STL_DISK_FULL_DIAG 文档。	2018 年 8 月 31 日
Amazon Redshift Spectrum 中对于嵌套数据的支持	现在可以查询在 Amazon Redshift Spectrum 表中存储的嵌套数据。有关更多信息，请参阅教程：使用 Amazon Redshift Spectrum 查询嵌套数据。	2018 年 8 月 8 日

默认情况下启用 SQA	现在，默认情况下为所有新集群启用短查询加速 (SQA)。SQA 使用机器学习来提供更高的性能、更快的结果以及更好的查询执行时间可预测性。有关更多信息，请参阅短查询加速。	2018 年 8 月 8 日
Amazon Redshift Advisor	您现在可以从 Amazon Redshift Advisor 获得有关如何提高集群性能和降低运营成本的定制建议。有关更多信息，请参阅 Amazon Redshift Advisor。	2018 年 7 月 26 日
即时别名引用	在定义已指定别名的表达式后，您现在可以立即引用它。有关更多信息，请参阅 SELECT 列表。	2018 年 7 月 18 日
在创建外部表时指定压缩类型	现在，您可以在使用 Amazon Redshift Spectrum 创建外部表时指定压缩类型。有关更多信息，请参阅创建外部表。	2018 年 27 月 6 日
PG_LAST_UNLOAD_ID	添加了有关新系统信息功能的文档：PG_LAST_UNLOAD_ID。有关更多信息，请参阅 PG_LAST_UNLOAD_ID。	2018 年 27 月 6 日
ALTER TABLE RENAME COLUMN	ALTER TABLE 现在支持重命名外部表的列。有关更多信息，请参阅修改外部表示例。	2018 年 7 月 6 日

早期更新

下表描述 2018 年 6 月之前发布的每个 Amazon Redshift 数据库开发人员指南中的重要变化。

更改	描述	更改日期
Parquet 中的 COPY 包含 SMALLINT	COPY 现在支持从 Parquet 格式文件加载到使用 SMALLINT 数据类型的列。有关更多信息，请参阅从列式数据格式中执行 COPY 操作。	2019 年 1 月 2 日
列式格式中的 COPY	COPY 现支持在 Amazon S3 上从使用 Parquet 和 ORC 列式数据格式的文件中进行加载。有关更多信息，请参阅从列式数据格式中执行 COPY 操作。	2018 年 5 月 17 日
SQA 的动态最大运行时	默认情况下，工作负载管理 (WLM) 现在根据集群的工作负载分析为短查询加速 (SQA) 最大运行时动态分配值。有关更多信息，请参阅短查询的最大运行时间。	2018 年 5 月 17 日
STL_LOAD_COMMITS 中的新列	STL_LOAD_COMMITS 系统表有一个新列：`file_format` 。	2018 年 5 月 10 日
STL_HASHJOIN 和其他系统日志表中的新列	STL_HASHJOIN 系统表有三个新列：`hash_segment` 、`hash_step` 和 `checksum`。此外，`checksum` 已添加到 STL_MERGEJOIN、STL_NESTLOOP、STL_HASH、STL_SCAN、STL_SORT、STL_LIMIT 和 STL_PROJECT。	2018 年 5 月 17 日
STL_AGGR 中的新列	STL_AGGR 系统表有两个新列：`resizes` 和 `flushable` 。	2018 年 4 月 19 日
REGEX 函数的新选项	对于 REGEXP_INSTR 和 REGEXP_SUBSTR 函数，您现在可以指定要使用的匹配项的出现次数以及是否执行区分大小写的匹配。REGEXP_INSTR 还您让您指定是否返回匹配项的第一个字符的位置，或匹配项结尾后第一个字符的位置。	2018 年 3 月 22 日
系统表中的新列	STL_COMMIT_STATS 系统表中添加了 tombstone dblocks、tossedblocks 和 batched_by 列。向 STV_SLICES 系统视图中添加了 localslice 列。	2018 年 3 月 22 日

更改	描述	更改日期
在外部表中添加和删除列	ALTER TABLE 现在对 Amazon Redshift Spectrum 外部表支持 ADD COLUMN 和 DROP COLUMN。	2018 年 3 月 22 日
Redshift Spectrum 新 AWS 区域	Redshift Spectrum 现已在孟买和圣保罗区域推出。有关受支持的 区域的列表，请参阅Amazon Redshift Spectrum 区域。	2018 年 3 月 22 日
表限制增加为 20,000	最大表数现在对于 8xlarge 集群节点类型为 20000。对于 large 和 xlarge 节点类型，此限制为 9,900。有关更多信息，请参阅限制和配额。	2018 年 3 月 13 日
Redshift Spectrum 支持 JSON 和 Ion	使用 Redshift Spectrum，您可以引用包含采用 JSON 或 Ion 数据格式的标量数据。有关更多信息，请参阅CREATE EXTERNAL TABLE。	2018 年 2 月 26 日
适用于 Redshift Spectrum 的 IAM 角色串联	您可以串联 AWS Identity and Access Management (IAM) 角色，以便您的集群可以未附加到该集群的其他角色，包括属于其他 AWS 账户的角色。有关更多信息，请参阅在 Amazon Redshift Spectrum 中链接 IAM 角色。	2018 年 2 月 1 日
ADD PARTITION 支持 IF NOT EXISTS	ALTER TABLE 的 ADD PARTITION 子句现在支持 IF NOT EXISTS 选项。有关更多信息，请参阅ALTER TABLE。	2018 年 1 月 11 日
外部表的 DATE 数据	Redshift Spectrum 外部表现在支持 DATE 数据类型。有关更多信息，请参阅CREATE EXTERNAL TABLE。	2018 年 1 月 11 日
Redshift Spectrum 新 AWS 区域	Redshift Spectrum 现已在新加坡、悉尼、首尔和法兰克福区域可用。有关受支持的 AWS 区域的列表，请参阅Amazon Redshift Spectrum 区域。	2017 年 11 月 16 日
Amazon Redshift 工作负载管理 (WLM) 中的短查询加速	短查询加速 (SQA) 让选定的短时查询优先于长时查询。SQA 在专用空间中执行短时查询，因此 SQA 查询不会被迫排在队列中的长时查询后面等待。使用 SQA，短时查询会更快地开始执行，用户会更快地看到结果。有关更多信息，请参阅使用短查询加速。	2017 年 11 月 16 日

更改	描述	更改日期
WLM 重新分配跳过的查询	Amazon Redshift 工作负载管理 (WLM) 现在将有资格的被跳过查询重新分配到新队列，而不是取消并重新启动该查询。当 WLM 重新分配查询时，它会将查询移动到新队列并继续执行，这将节省时间和系统资源。没有重新分配资格的被跳过查询将会重新启动或取消。有关更多信息，请参阅WLM 查询队列跳过。	2017 年 11 月 16 日
用户的系统日志访问权限	在大多数对用户可见的系统日志表中，默认情况下，由其他用户生成的行对普通用户是不可见的。要允许普通用户查看用户可见表中的所有行 (包括由其他用户生成的行)，请运行 ALTER USER 或 CREATE USER 并将 SYSLOG ACCESS 参数设置为 UNRESTRICTED。	2017 年 11 月 16 日
结果缓存	借助 结果缓存 ，当您运行查询时，Amazon Redshift 会缓存结果。当您再次运行查询时，Amazon Redshift 会检查有无查询结果的有效缓存副本。如果在结果缓存中找到匹配项，Amazon Redshift 会使用缓存的结果而不运行查询。默认情况下，结果缓存处于打开状态。要禁用结果缓存，请将 enable_result_cache_for_session 配置参数设置为 `off`。	2017 年 11 月 16 日
列元数据函数	PG_GET_COLS 和 PG_GET_LATE_BINDING_VIEW_COLS 返回 Amazon Redshift 表、视图和后期绑定视图的列元数据。	2017 年 11 月 16 日
针对 CTAS 的 WLM 队列跳过	Amazon Redshift 工作负载管理 (WLM) 现在支持针对 CREATE TABLE AS (CTAS) 语句以及只读查询（例如 SELECT 语句）的查询队列跳过。有关更多信息，请参阅WLM 查询队列跳过。	2017 年 10 月 19 日
Amazon Redshift Spectrum 清单文件	当您创建 Redshift Spectrum 外部表时，可以指定一个清单文件，用以列出数据文件在 Amazon S3 上的位置。有关更多信息，请参阅CREATE EXTERNAL TABLE。	2017 年 10 月 19 日

更改	描述	更改日期
Amazon Redshift Spectrum 新 AWS 区域	Redshift Spectrum 现已在欧洲 (爱尔兰) 和亚太 (东京) 区域推出。有关受支持的 AWS 区域的列表，请参阅 Amazon Redshift Spectrum 注意事项。	2017 年 10 月 19 日
Amazon Redshift Spectrum 添加的文件格式	现在，您可以创建基于 Regex、OpenCSV 和 Avro 数据文件格式的 Redshift Spectrum 外部表。有关更多信息，请参阅CREATE EXTERNAL TABLE。	2017 年 10 月 5 日
Amazon Redshift Spectrum 外部表的 Pseudocolumns	您可以在 Redshift Spectrum 外部表中选择 `$path` 和 `$size` pseudocolumns 来查看被引用数据文件在 Amazon S3 中的位置和大小。有关更多信息，请参阅 Pseudocolumns。	2017 年 10 月 5 日
用于验证 JSON 的函数	您可以使用 IS_VALID_JSON 和 IS_VALID_JSON_ARRAY 函数来检查 JSON 格式设置是否有效。其他 JSON 函数现在有可选 `null_if_invalid` 参数。	2017 年 10 月 5 日
LISTAGG DISTINCT	您可以将 DISTINCT 子句与 LISTAGG 聚合函数以及 LISTAGG 开窗函数一起使用，以便在串联之前从指定的表达式中消除重复的值。	2017 年 10 月 5 日
以大写形式查看列名称	要以大写形式查看 SELECT 结果中的列名称，您可以将 describe_field_name_in_uppercase 配置参数设置为 `true`。	2017 年 10 月 5 日
跳过外部表中的标头行	您可以将 `skip.header.line.count` 命令中的 CREATE EXTERNAL TABLE 属性设置为在 Redshift Spectrum 数据文件的开头跳过标头行。	2017 年 10 月 5 日
扫描行数	WLM 查询监控规则使用 scan_row_count 指标返回扫描步骤中的行数。行计数是在筛选标记为删除的行 (虚影行) 之前和应用用户定义的查询筛选之前发出的行的总数。有关更多信息，请参阅预置的 Amazon Redshift 的查询监控指标。	2017 年 9 月 21 日

更改	描述	更改日期
SQL 用户定义的函数	标量 SQL 用户定义的函数 (UDF) 纳入了一个 SQL SELECT 子句，该子句在此函数被调用并返回单个值时执行。有关更多信息，请参阅创建标量 SQL UDF。	2017 年 8 月 31 日
后期绑定视图	后期绑定视图未绑定到基础数据库对象，例如表和用户定义的函数。因此，视图与其引用的对象之间不存在依赖关系。即使引用的对象不存在，您也可以创建视图。由于不存在依赖关系，删除或更改引用的对象不会影响视图。在查询视图之前，Amazon Redshift 不会检查依赖关系。要创建后期绑定视图，请使用 CREATE VIEW 语句指定 WITH NO SCHEMA BINDING 子句。有关更多信息，请参阅CREATE VIEW。	2017 年 8 月 31 日
OCTET_LENGTH 函数	OCTET_LENGTH 将以字节数形式返回指定字符串的长度。	2017 年 8 月 18 日
支持 ORC 和 Grok 文件类型	Amazon Redshift Spectrum 现在支持 Redshift Spectrum 数据文件的 ORC 和 Grok 数据格式。有关更多信息，请参阅为 Amazon Redshift Spectrum 中的查询创建数据文件。	2017 年 8 月 18 日
现在支持 RegexSerDe	Amazon Redshift Spectrum 现在支持 RegexSerDe 数据格式。有关更多信息，请参阅为 Amazon Redshift Spectrum 中的查询创建数据文件。	2017 年 7 月 19 日
向 SVV_TABLES 和 SVV_COLUMNS 中添加了新列	向 `domain_name` 中添加了 `remarks` 和 SVV_COLUMNS 列。向 SVV_TABLES 中添加了备注列。	2017 年 7 月 19 日
SVV_TABLES 和 SVV_COLUMNS 系统视图	SVV_TABLES 和 SVV_COLUMNS 系统视图提供有关本地和外部表及视图的列信息和其他详情。	2017 年 7 月 7 日

更改	描述	更改日期
Amazon Redshift Spectrum 与 Amazon EMR Hive 元数据存储配合使用时，不再需要 VPC	Redshift Spectrum 取消了以下要求：使用 Amazon EMR Hive 元数据存储时，Amazon Redshift 集群和 Amazon EMR 集群必须位于同一个 VPC 及同一个子网中。有关更多信息，请参阅在 Amazon Redshift Spectrum 中使用外部目录。	2017 年 7 月 7 日
用 UNLOAD 创建较小的文件	预设情况下，UNLOAD 会在 Amazon S3 中创建多个文件，每个文件的最大大小为 6.2 GB。要创建较小的文件，请在使用 UNLOAD 命令时指定 MAXFILESIZE。您可以将最大文件大小指定为 5 MB 到 6.2 GB。有关更多信息，请参阅UNLOAD。	2017 年 7 月 7 日
TABLE PROPERTIES	现在，您可以为 CREATE EXTERNAL TABLE 或 ALTER TABLE 设置 TABLE PROPERTIES numRows 参数以更新表统计数据，从而反映表中的行数。	2017 年 6 月 6 日
ANALYZE PREDICATE COLUMNS	为了节省时间和集群资源，您可以选择仅分析可能用作谓词的列。使用 PREDICATE COLUMNS 子句运行 ANALYZE 时，分析操作仅包括已在联接、筛选条件或 group by 子句中使用的列，或用作排序键或分配键的列。有关更多信息，请参阅分析表。	2017 年 5 月 25 日
适用于 Amazon Redshift Spectrum 的 IAM 策略	要仅使用 Redshift Spectrum 授予对 Amazon S3 存储桶的访问权限，您可以包括允许访问用户代理 `AWS Redshift/Spectrum` 的条件。有关更多信息，请参阅适用于 Amazon Redshift Spectrum 的 IAM 策略。	2017 年 5 月 25 日
Amazon Redshift Spectrum 递归扫描	Redshift Spectrum 现在扫描子文件夹以及 Amazon S3 中所指定文件夹内的文件。有关更多信息，请参阅为 Redshift Spectrum 创建外部表。	2017 年 5 月 25 日

更改	描述	更改日期
查询监控规则	使用 WLM 查询监控规则，您可以为 WLM 查询定义基于指标的性能边界，并指定查询超出这些边界时需要采取的操作—log、hop 或 abort。您将在工作负载管理 (WLM) 配置中定义查询监控规则。有关更多信息，请参阅WLM 查询监控规则。	2017 年 4 月 21 日
Amazon Redshift Spectrum	使用 Redshift Spectrum，您可以在 Amazon S3 中高效地查询和检索文件中的数据，而无需将数据加载到表中。Redshift Spectrum 查询针对大型数据集执行速度非常快，因为 Redshift Spectrum 直接在 Amazon S3 中扫描数据文件。很多处理发生在 Amazon Redshift Spectrum 层中，而大多数数据位于 Amazon S3 中。多个集群可同时查询 Amazon S3 上的同一数据集，而无需为每个集群复制数据。有关更多信息，请参阅使用 Amazon Redshift Spectrum 查询外部数据。	2017 年 4 月 19 日
支持 Redshift Spectrum 的新系统表	已添加以下新的系统视图来支持 Redshift Spectrum： • SVL_S3QUERY • SVL_S3QUERY_SUMMARY • SVV_EXTERNAL_COLUMNS • SVV_EXTERNAL_DATABASES • SVV_EXTERNAL_PARTITIONS • SVV_EXTERNAL_TABLES • PG_EXTERNAL_SCHEMA	2017 年 4 月 19 日
APPROXIMATE PERCENTILE_DISC 聚合函数	现在可以使用 APPROXIMATE PERCENTILE_DISC 聚合函数。	2017 年 4 月 4 日

更改	描述	更改日期
使用 KMS 的服务器端加密	您现在可以使用 AWS Key Management Service 密钥进行服务器端加密 (SSE-KMS)，将数据卸载到 Amazon S3。此外，COPY 现在可以从 Amazon S3 透明加载 KMS 加密的数据文件。有关更多信息，请参阅 UNLOAD。	2017 年 2 月 9 日
新的授权语法	您现在可以使用 IAM_ROLE、MASTER_SYMMETRIC_KEY、ACCESS_KEY_ID、SECRET_ACCESS_KEY 和 SESSION_TOKEN 参数为 COPY、UNLOAD 和 CREATE LIBRARY 命令提供授权和访问信息。新的授权语法为 CREDENTIALS 参数提供了单一字符串参数，是更具灵活性的方案。有关更多信息，请参阅授权参数。	2017 年 2 月 9 日
Schema 上限提高	每集群现在最多可以创建 9,900 个 schemas。有关更多信息，请参阅CREATE SCHEMA。	2017 年 2 月 9 日
默认表编码	CREATE TABLE 和 ALTER TABLE 现在向大多数新列分配 LZO 压缩编码。默认情况下，会向定义为排序键的列、定义为 BOOLEAN、REAL 或 DOUBLE PRECISION 数据类型的列以及临时表分配 RAW 编码。有关更多信息，请参阅ENCODE。	2017 年 2 月 6 日
ZSTD 压缩编码	Amazon Redshift 现在支持 ZSTD 列压缩编码。	2017 年 1 月 19 日
PERCENTILE_CONT 和 MEDIAN 聚合函数	PERCENTILE_CONT 和 MEDIAN 现在可作为聚合函数和窗口函数使用。	2017 年 1 月 19 日
用户定义的函数 (UDF) 用户日志记录	您可以使用 Python 日志记录模块在 UDF 中创建用户定义的错误和警告消息。执行查询后，您可以查询 SVL_UDF_LOG 系统视图以检索记录的消息。有关用户定义消息的更多信息，请参阅在 UDF 中记录错误和警告	2016 年 12 月 8 日

更改	描述	更改日期
ANALYZE COMPRESSION 估计的压缩量	ANALYZE COMPRESSION 命令现在可报告每一列的磁盘空间预估压缩百分比。有关更多信息，请参阅ANALYZE COMPRESSION。	2016 年 11 月 10 日
连接限制	您现在可以设置允许用户同时打开的数据库连接的数量限制。您也可以限制数据库并行连接数量。有关更多信息，请参阅CREATE USER和CREATE DATABASE。	2016 年 11 月 10 日
COPY 排序顺序增强	如果您以排序键顺序加载数据，COPY 现在可以自动将新行添加到表中已排序区域。有关启用此增强功能的具体要求，请参阅 按排序键顺序加载数据	2016 年 11 月 10 日
压缩 CTAS	CREATE TABLE AS (CTAS) 现在可以根据列的数据类型自动为新表分配压缩编码。有关更多信息，请参阅继承列和表属性。	2016 年 10 月 28 日
有时区数据类型的时间戳	Amazon Redshift 现在支持使用时区 (TIMESTAMPTZ) 数据类型的时间戳。此外，已添加几个新功能以支持新的数据类型。有关更多信息，请参阅日期和时间函数。	2016 年 9 月 29 日
分析阈值	为了减少 ANALYZE 操作的处理时间并提高整体系统性能，在自上次运行 ANALYZE 命令以来更改的行数百分比低于 analyze_threshold_percent 参数指定的分析阈值的情况下，Amazon Redshift 将跳过对表的分析。默认情况下，analyze_threshold_percent 为 10。	2016 年 8 月 9 日
新 STL_RESTARTED_SESSIONS 系统表	在 Amazon Redshift 重新启动会话时， STL_RESTARTED_SESSIONS 将记录新的进程 ID (PID) 和旧 PID。	2016 年 8 月 9 日
更新了“日期和时间函数”文档	添加了函数概要（其中包含指向 日期和时间函数 的链接），并更新了函数参考以确保一致性。	2016 年 6 月 24 日

更改	描述	更改日期
STL_CONNECTION_LOG 中新增了一些列	STL_CONNECTION_LOG 系统表有两个用于跟踪 SSL 连接的新列。如果您定期向 Amazon Redshift 表加载审核日志，则需要向目标表添加以下新列：sslcompression 和 sslexpansion。	2016 年 5 月 5 日
MD5 哈希密码	通过提供密码和用户名的 MD5 哈希字符串，您可为 CREATE USER 或 ALTER USER 命令指定密码。	2016 年 4 月 21 日
STV_TBL_PERM 中的新列	backup 系统视图中的 STV_TBL_PERM 列指示表是否包含在集群快照中。有关更多信息，请参阅BACKUP。	2016 年 4 月 21 日
无备份表	对于不会包含关键数据的表（如暂存表），您可在 CREATE TABLE 或 CREATE TABLE AS 语句中指定 BACKUP NO 来防止 Amazon Redshift 在自动或手动快照中包含表。使用无备份表可节省创建快照并从快照还原时的处理时间，并可减少在 Amazon S3 上占用的存储空间。	2016 年 4 月 7 日
VACUUM 删除阈值	默认情况下，VACUUM 命令现在将回收空间，以使至少 95% 的剩余行不会被标记为删除。因此，与回收 100% 的已删除行相比，VACUUM 在删除阶段所耗的时间通常少得多。您可以在运行 VACUUM 命令时包含 TO threshold PERCENT 参数，从而更改某个表的默认阈值。	2016 年 4 月 7 日
SVV_TRANSACTIONS 系统表	SVV_TRANSACTIONS 系统视图记录当前锁定到数据库中的表的事务的相关信息。	2016 年 4 月 7 日
使用 IAM 角色访问其他 AWS 资源	要在您的集群和其他 AWS 资源（如 Amazon S3、DynamoDB、Amazon EMR 或 Amazon EC2）之间移动数据，您的集群必须具有访问相应资源和执行所需操作的权限。作为为 COPY、UNLOAD 或 CREATE LIBRARY 命令提供访问密钥对的更安全的替代方法，您现在可以指定您的集群用于身份验证和授权的 IAM 角色。有关更多信息，请参阅基于角色的访问控制。	2016 年 3 月 29 日

更改	描述	更改日期
VACUUM 排序阈值	现在，当任意表中有 95% 的行已有序时，VACUUM 命令会为该表跳过排序阶段。您可以在运行 VACUUM 命令时包含 TO threshold PERCENT 参数，从而更改某个表的默认排序阈值。	2016 年 17 月 3 日
STL_CONNECTION_LOG 中新增了一些列	STL_CONNECTION_LOG 系统表新增了三列内容。如果您定期向 Amazon Redshift 表加载审核日志，则需要向目标表添加以下新列：sslversion、sslcipher 和 mtu。	2016 年 17 月 3 日
使用 bzip2 压缩执行 UNLOAD	您现在可以选择使用 bzip2 压缩执行 UNLOAD。	2016 年 2 月 8 日
ALTER TABLE APPEND	ALTER TABLE APPEND 通过从现有的源表移动数据，将行附加到目标表。由于是移动数据而不是复制数据，因此相比类似的 CREATE TABLE AS 或 INSERT INTO 操作，ALTER TABLE APPEND 通常要快得多。	2016 年 2 月 8 日
WLM 查询队列跳过	如果工作负载管理 (WLM) 由于 WLM 超时而取消一个只读查询（例如一个 SELECT 语句），WLM 会尝试将查询路由到下一个匹配的队列。有关更多信息，请参阅WLM 查询队列跳过。	2016 年 1 月 7 日
ALTER DEFAULT PRIVILEGES	您可以使用 ALTER DEFAULT PRIVILEGES 命令定义要应用到由指定用户在未来创建的对象的一组默认访问权限。	2015 年 12 月 10 日
bzip2 文件压缩	COPY 命令支持从使用 bzip2 压缩的文件加载数据。	2015 年 12 月 10 日
NULLS FIRST 和 NULLS LAST	您可以指定 ORDER BY 子句是将 NULLS 放在结果集的第一个还是最后一个。有关更多信息，请参阅ORDER BY 子句和窗口函数语法摘要。	2015 年 11 月 19 日

更改	描述	更改日期
CREATE LIBRARY 的 REGION 关键字	如果包含 UDF 库文件的 Amazon S3 存储桶与您的 Amazon Redshift 集群不在同一个 AWS 区域内，您可以使用 REGION 选项指定数据所在的区域。有关更多信息，请参阅CREATE LIBRARY。	2015 年 11 月 19 日
用户定义的标量函数 (UDF)	现在，您可以创建自定义的用户定义标量函数，以实施 Python 2.7 标准库中的 Amazon Redshift 支持的模块或您自己的基于 Python 编程语言的自定义 UDF 提供的非 SQL 处理功能。有关更多信息，请参阅创建用户定义的函数。	2015 年 9 月 11 日
WLM 配置中的动态属性	WLM 配置参数现在支持动态应用一些属性。其他属性保持静态更改，并需要重启关联的集群以便能应用配置更改。有关更多信息，请参阅WLM 动态和静态配置属性和实施工作负载管理。	2015 年 8 月 3 日
LISTAGG 函数	LISTAGG 函数 和 LISTAGG 窗口函数 返回通过串联一组列值而创建的字符串。	2015 年 7 月 30 日
淘汰的参数	max_cursor_result_set_size 配置参数已淘汰。光标结果集的大小受到集群的节点类型的限制。有关更多信息，请参阅游标约束。	2015 年 7 月 24 日
修订了 COPY 命令的文档	COPY 命令的参考内容进行了大范围的修订，以让材料更加友好、易读。	2015 年 7 月 15 日
从 Avro 格式 COPY	COPY 命令支持从 Amazon S3、Amazon EMR 上的数据文件以及从使用 SSH 的远程主机上加载 Avro 格式的数据。有关更多信息，请参阅AVRO和从 Avro 中复制的示例。	2015 年 7 月 8 日
STV_STARTUP_RECOVERY_STATE	STV_STARTUP_RECOVERY_STATE 系统表记录在执行集群重新启动操作期间临时锁定的表的状态。Amazon Redshift 对于正在处理以解决集群重新启动后过时的事务的表，会临时锁定这些表。	2015 年 5 月 25 日

更改	描述	更改日期
排名函数的可选 ORDER BY	对于部分开窗排名函数，现在可以使用可选的 ORDER BY 子句。有关更多信息，请参阅 CUME_DIST 开窗函数、DENSE_RANK 窗口函数、RANK 窗口函数、NTILE 窗口函数、PERCENT_RANK 开窗函数 和 ROW_NUMBER 窗口函数。	2015 年 5 月 25 日
交错排序键	交错排序键对于排序键中的每个列给予相同的权重。使用交错排序键而不是默认的复合键可以显著提升对辅助排序列使用限制性谓词的查询的性能，对于大型表尤其如此。当使用多个查询对同一个表中的不同列进行筛选时，交错排序还能提升整体性能。有关更多信息，请参阅使用排序键和CREATE TABLE。	2015 年 5 月 11 日
修订的优化查询性能主题	优化查询性能 已经扩展，加入了用于分析查询性能的新查询和更多示例。此外，此主题经过修订，内容更清晰、完整。设计查询的 Amazon Redshift 最佳实践 介绍了更多有关如何编写查询以改进性能的信息。	2015 年 3 月 23 日
SVL_QUERY_QUEUE_INFO	SVL_QUERY_QUEUE_INFO 视图详细总结了查询在 WLM 查询队列或提交队列中所用的时间。	2015 年 2 月 19 日
SVV_TABLE_INFO	您可以使用 SVV_TABLE_INFO 视图诊断和解决会影响查询性能的表设计问题，包括与压缩编码、分配键、排序方式、数据分配偏斜、表大小和统计数据相关的问题。	2015 年 2 月 19 日
UNLOAD 使用服务器端文件加密	UNLOAD 命令现在自动使用 Amazon S3 服务器端加密 (SSE) 来加密所有卸载数据文件。服务器端加密又增加了一层安全性，而且对性能只有少许或几乎没有影响。	2014 年 10 月 31 日
CUME_DIST 开窗函数	CUME_DIST 开窗函数 计算某个窗口或分区中某个值的累积分布。	2014 年 10 月 31 日
MONTHS_BETWEEN 函数	MONTHS_BETWEEN 函数 确定两个日期之间相隔的月数。	2014 年 10 月 31 日

更改	描述	更改日期
NEXT_DAY 函数	NEXT_DAY 函数 返回比给定日期晚的指定日期的第一个实例的日期。	2014 年 10 月 31 日
PERCENT_RANK 开窗函数	PERCENT_RANK 开窗函数 计算某个给定行的百分比排名。	2014 年 10 月 31 日
RATIO_TO_REPORT 开窗函数	RATIO_TO_REPORT 开窗函数 计算某个窗口或分区中一个值与值的总和的比率。	2014 年 10 月 31 日
TRANSLATE 函数	TRANSLATE 函数 使用指定的替换字符替换指定表达式中出现的所有指定字符。	2014 年 10 月 31 日
NVL2 函数	NVL2 函数 根据指定表达式的结果是 NULL 或 NOT NULL，返回这两个值中的一个。	2014 年 10 月 16 日
MEDIAN 开窗函数	MEDIAN 开窗函数 计算某个窗口或分区中值的范围的中间值。	2014 年 10 月 16 日
GRANT 和 REVOKE 命令的 ON ALL TABLES IN SCHEMA schema_name 子句	GRANT 和 REVOKE 命令已更新为可以使用 ON ALL TABLES IN SCHEMA schema_name 子句。使用此子句，您可以使用一个命令更改 schema 中所有表的权限。	2014 年 10 月 16 日
DROP SCHEMA、DROP TABLE、DROP USER 和 DROP VIEW 命令的 IF EXISTS 子句	DROP SCHEMA、DROP TABLE、DROP USER 和 DROP VIEW 命令已经更新为可以使用 IF EXISTS 子句。如果指定的对象不存在，使用此子句可让命令不执行任何更改并且返回消息，而不是以错误终止命名。	2014 年 10 月 16 日

更改	描述	更改日期
CREATE SCHEMA 和 CREATE TABLE 命令的 IF NOT EXISTS 子句	CREATE SCHEMA 和 CREATE TABLE 命令已经更新为可以使用 IF NOT EXISTS 子句。如果指定的对象已经存在，使用此子句可让命令不执行任何更改并且返回消息，而不是以错误终止命名。	2014 年 10 月 16 日
COPY 支持 UTF-16 编码	COPY 命令现在支持从使用 UTF-16 编码以及 UTF-8 编码的数据文件加载。有关更多信息，请参阅ENCODING。	2014 年 9 月 29 日
新的工作负载管理教程	教程：配置手动工作负载管理 (WLM) 队列 为您介绍了配置工作负载管理 (WLM) 队列以改进查询处理和分配查询的流程。	2014 年 9 月 25 日
AES 128 位加密	当使用 Amazon S3 客户端加密从数据文件加载数据时，COPY 命令现在支持 AES 128 位加密和 AES 256 位加密。有关更多信息，请参阅从 Amazon S3 中加载加密的数据文件。	2014 年 9 月 29 日
PG_LAST_UNLOAD_COUNT 函数	PG_LAST_UNLOAD_COUNT 函数返回在最近的 UNLOAD 操作中处理的行数。有关更多信息，请参阅PG_LAST_UNLOAD_COUNT。	2014 年 9 月 15 日
新的查询故障排除部分	查询故障排除 提供了快速参考，帮助您识别和解决一些在使用 Amazon Redshift 查询时可能会遇到的最常见问题和最严重问题。	2014 年 7 月 7 日
新的加载数据教程	教程：从 Amazon S3 加载数据 为您演练从 Amazon S3 存储桶中的数据文件将数据加载到您的 Amazon Redshift 数据库表中的完整过程。	2014 年 7 月 1 日
PERCENTILE_CONT 开窗函数	PERCENTILE_CONT 开窗函数 是一个假定连续分布模型的逆分布函数。该函数具有一个百分比值和一个排序规范，并返回一个在有关排序规范的给定百分比值范围内的内插值。	2014 年 6 月 30 日

更改	描述	更改日期
PERCENTILE_DISC 开窗函数	PERCENTILE_DISC 开窗函数 是一个假定离散分布模型的逆分布函数。该函数具有一个百分位数值和一个排序规范，并从集合中返回一个元素。	2014 年 6 月 30 日
GREATEST 和 LEAST 函数	GREATEST 和 LEAST 函数 函数从表达式列表返回最大或最小值。	2014 年 6 月 30 日
跨区域 COPY	COPY 命令支持从与 Amazon Redshift 集群位于不同区域的 Amazon S3 存储桶或 Amazon DynamoDB 表加载数据。有关更多信息，请参阅 COPY 命令参考中的REGION。	2014 年 6 月 30 日
“最佳实践”已扩充	Amazon Redshift 最佳实践已经扩充、重新组织，并移至导航层次结构中的顶级，以便读者更容易发现。	2014 年 5 月 28 日
UNLOAD 到单个文件	UNLOAD 命令通过添加 PARALLEL OFF 选项，可以依次将表数据卸载到 Amazon S3 上的一个文件。如果数据的大小超过文件大小上限 (6.2 GB)，UNLOAD 将创建更多文件。	2014 年 5 月 6 日
REGEXP 函数	REGEXP_COUNT、REGEXP_INSTR 和 REGEXP_REPLACE 函数根据正则表达式模式匹配处理字符串。	2014 年 5 月 6 日
从 Amazon EMR 执行 COPY 操作	COPY 命令支持直接从 Amazon EMR 集群加载数据。有关更多信息，请参阅从 Amazon EMR 中加载数据。	2014 年 4 月 18 日
WLM 并发限制增加	现在，您可以将工作负载管理 (WLM) 配置在用户定义的查询队列中并发运行最多 50 个查询。这一增加为用户提供了更大的灵活性，可以通过修改 WLM 配置来管理系统性能。有关更多信息，请参阅实施手动 WLM。	2014 年 4 月 18 日

更改	描述	更改日期
用于管理光标大小的新配置参数	`max_cursor_result_set_size` 配置参数定义了大型查询的每个光标结果集可以返回的数据大小（以兆字节为单位）。此参数值也会影响集群的并发光标数量，让您能够配置用于增加或减少集群光标数量的值。 有关更多信息，请参阅本指南中的DECLARE和《Amazon Redshift 管理指南》中的配置游标结果集的最大大小。	2014 年 3 月 28 日
从 JSON 格式数据执行的 COPY 操作	COPY 命令支持从 Amazon S3 上的数据文件以及从使用 SSH 的远程主机上加载 JSON 格式的数据。有关更多信息，请参阅 从 JSON 格式数据执行的 COPY 操作 使用说明。	2014 年 3 月 25 日
新系统表 STL_PLAN_INFO	STL_PLAN_INFO 表补充了 EXPLAIN 命令，作为另一种查看查询计划的方式。	2014 年 3 月 25 日
新函数 REGEXP_SUBSTR	通过搜索正则表达式模式，REGEXP_SUBSTR 函数 返回从字符串提取的字符。	2014 年 3 月 25 日
STL_COMMIT_STATS 的新列	STL_COMMIT_STATS 表添加了两个新的列：`numxids` 和 `oldestxid` 。	2014 年 3 月 6 日
从 SSH 执行 COPY 支持 gzip 和 lzop	当通过 SSH 连接加载数据时，COPY 命令支持 gzip 和 lzop 压缩。	2014 年 2 月 13 日
新函数	ROW_NUMBER 窗口函数 返回当前行的数字。STRTOL 函数 将指定基数的数字的字符串表达式转换为相当的整数值。PG_CANCEL_BACKEND 和 PG_TERMINATE_BACKEND 让用户可以取消查询和会话连接。已经添加 LAST_DAY 函数来实现对 Oracle 的兼容性。	2014 年 2 月 13 日

更改	描述	更改日期
新系统表	STL_COMMIT_STATS 系统表提供与提交性能相关的指标，包括提交的各个阶段的时间以及提交的数据块的数量。	2014 年 2 月 13 日
FETCH 单一节点集群	对单一节点集群使用光标时，可以使用 FETCH 命令提取的最大行数是 1000。单一节点集群不支持使用 FETCH FORWARD ALL。	2014 年 2 月 13 日
DS_DIST_ALL_INNER 重新分配策略	EXPLAIN 计划输出中的 DS_DIST_ALL_INNER 指明整个内部表已重新分配到一个切片，因为外部表使用 DISTSTYLE ALL。有关更多信息，请参阅联接类型示例和评估查询计划。	2014 年 1 月 13 日
查询的新系统表	Amazon Redshift 已添加新的系统表，客户可以使用它们来评估查询的执行情况，以进行优化和进行问题排查。有关更多信息，请参阅 SVL_COMPILE, STL_SCAN, STL_RETURN, STL_SAVESTL_ALERT_EVENT_LOG。	2014 年 1 月 13 日
单节点光标	单一节点集群现在支持光标。单一节点集群一次能打开两个光标，结果集最大为 32 GB。在单一节点集群上，我们建议将 ODBC 缓存大小参数设置为 1000。有关更多信息，请参阅DECLARE。	2013 年 12 月 13 日
ALL 分配方式	ALL 分配可以显著缩短特定类型查询的执行时间。当表使用 ALL 分配方式时，该表的一个副本将分配到每个节点。因为该表有效地与其他每个表并置，因此在执行查询期间无需重新分配。ALL 分配并不适用于所有表，因为它会增加存储要求和加载时间。有关更多信息，请参阅使用数据分配样式。	2013 年 11 月 11 日

更改	描述	更改日期
从远程主机执行 COPY	除了从 Amazon S3 上的数据文件和 Amazon DynamoDB 表加载表之外，COPY 命令还可以从 Amazon EMR 集群、Amazon EC2 实例和其他使用 SSH 连接的远程主机加载文本数据。Amazon Redshift 使用多个同时的 SSH 连接来并行读取和加载数据。有关更多信息，请参阅从远程主机中加载数据。	2013 年 11 月 11 日
使用的 WLM 内存百分比	您可以在工作负载管理 (WLM) 配置中为每个查询指定具体的内存百分比，来实现工作负载的平衡。有关更多信息，请参阅实施手动 WLM。	2013 年 11 月 11 日
APPROXIMATE COUNT(DISTINCT)	使用 APPROXIMATE COUNT(DISTINCT) 的查询执行速度更快，相对错误率大约为 2%。APPROXIMATE COUNT(DISTINCT) 行数使用 HyperLogLog 算法。有关更多信息，请参见 COUNT 函数。	2013 年 11 月 11 日
用于检索最近的查询详细信息的新 SQL 函数	四个新的 SQL 函数可以检索有关最近查询和 COPY 命令的详细信息。这四个新函数让查询系统日志表更简单易行，而且在很多时候，可以无需访问系统表即提供必要的详细信息。有关更多信息，请参阅 PG_BACKEND_PID, PG_LAST_COPY_ID, PG_LAST_COPY_COUNT, PG_LAST_QUERY_ID。	2013 年 11 月 1 日
UNLOAD 的 MANIFEST 选项	UNLOAD 命令的 MANIFEST 选项可以补充 COPY 命令的 MANIFEST 选项。使用带 MANIFEST 选项的 UNLOAD 命令可自动创建一个清单文件，该文件显式列出了卸载操作在 Amazon S3 上创建的数据文件。然后您可以在 COPY 命令中使用该清单文件来加载数据。有关更多信息，请参阅将数据卸载到 Amazon S3和UNLOAD 示例。	2013 年 11 月 1 日
COPY 的 MANIFEST 选项	您可以使用带 MANIFEST 选项的 COPY 命令来显式列出将要从 Amazon S3 加载的数据文件。	2013 年 10 月 18 日

更改	描述	更改日期
用于对查询进行问题排查的系统表	为用于对查询进行问题排查的系统表添加了文档。用于日志记录的 STL 视图 部分闲置包含以下系统表的文档：STL_AGGR、STL_BCAST、STL_DIST、STL_DELETE、STL_HASH、STL_HASHJOIN、STL_INSERT、STL_LIMIT、STL_MERGE、STL_MERGEJOIN、STL_NESTLOOP、STL_PARSE、STL_PROJECT、STL_SCAN、STL_SORT、STL_UNIQUE 和 STL_WINDOW。	2013 年 10 月 3 日
CONVERT_TIMEZONE 函数	CONVERT_TIMEZONE 函数 将一个时区的时间戳转换为另一个时区的时间戳，包含用于自动调整为夏令时的选项。	2013 年 10 月 3 日
SPLIT_PART 函数	SPLIT_PART 函数 用指定的分隔符拆分字符串，并返回指定位置的部分内容。	2013 年 10 月 3 日
STL_USERLOG 系统表	STL_USERLOG 记录创建、修改或删除数据库用户时发生的更改的详细信息。	2013 年 10 月 3 日
LZO 列编码和 LZOP 文件压缩。	LZO 列压缩编码既有非常高的压缩率，又有很好的性能。从 Amazon S3 进行 COPY 支持加载使用 LZOP 压缩进行压缩的文件。	2013 年 9 月 19 日
JSON、正则表达式和游标	添加了对分析 JSON 字符串的支持，使用正则表达式的模式匹配，以及使用光标通过 ODBC 连接检索大型数据集。有关更多信息，请参阅JSON 函数、模式匹配条件和DECLARE。	2013 年 9 月 10 日
COPY 的 ACCEPTINVCHAR 选项	通过指定带 ACCEPTINVCHAR 选项的 COPY 命令，您可以成功地加载包含无效的 UTF-8 字符的数据。	2013 年 8 月 29 日
COPY 的 CSV 选项	COPY 命令现在支持从 CSV 格式的输入文件加载。	2013 年 8 月 9 日
CRC32	CRC32 函数 执行循环冗余检验。	2013 年 8 月 9 日

更改	描述	更改日期
WLM 通配符	工作负载管理 (WLM) 支持在向队列添加用户组和查询组时使用通配符。有关更多信息，请参阅通配符。	2013 年 8 月 1 日
WLM 超时	要限制允许查询在给定 WLM 队列中停留的时间，您可以为每个队列设置 WLM 超时。有关更多信息，请参阅WLM 超时。	2013 年 8 月 1 日
新的 COPY 复制选项“auto”和“epochsecs”	COPY 命令可执行自动的日期和时间格式识别。新的时间格式“epochsecs”和“epochmillisecs”可让 COPY 加载纪元格式的数据。	2013 年 7 月 25 日
CONVERT_TIMEZONE 函数	CONVERT_TIMEZONE 函数 将一个时区的时间戳转换为另一个时区的时间戳。	2013 年 7 月 25 日
FUNC_SHA1 函数	FUNC_SHA1 函数 使用 SHA1 算法转换字符串。	2013 年 7 月 15 日
max_execution_time	要限制允许使用的队列时间，您可以在配置 WLM 期间设置 max_execution_time 参数。有关更多信息，请参阅修改 WLM 配置。	2013 年 7 月 22 日
四字节的 UTF-8 字符	VARCHAR 数据类型现在支持四字节的 UTF-8 字符。不支持 5 个字节或更长的 UTF-8 字符。有关更多信息，请参阅存储和范围。	2013 年 7 月 18 日
SVL_QERROR	SVL_QERROR 系统视图已淘汰。	2013 年 7 月 12 日
文件历史记录进行了修订	“文档历史记录”页面现在显示文档的更新日期。	2013 年 7 月 12 日
STL_UNLOAD_LOG	STL_UNLOAD_LOG 记录卸载操作的详细信息。	2013 年 7 月 5 日
JDBC 提取大小参数	为了避免在使用 JDBC 检索大型数据集时出现客户端内存不足错误，您可通过设置 JDBC 提取大小参数来使您的客户端能够成批提取数据。有关更多信息，请参阅设置 JDBC 提取大小参数。	2013 年 6 月 27 日

更改	描述	更改日期
UNLOAD 加密的文件	UNLOAD 现在支持向 Amazon S3 上的加密文件卸载表数据。	2013 年 5 月 22 日
临时凭证	COPY 和 UNLOAD 现在支持使用临时凭证。	2013 年 4 月 11 日
已添加说明	“设计表”和“加载数据”的讨论更加清晰并有所扩充。	2013 年 2 月 14 日
添加了最佳实践	添加了设计表的 Amazon Redshift 最佳实践和Amazon Redshift 加载数据的最佳实践。	2013 年 2 月 14 日
阐明了密码限制	阐明了针对 CREATE USER 和 ALTER USER 的密码限制，做出多处小修订。	2013 年 2 月 14 日
新指南	本指南是 Amazon Redshift 开发人员指南的第一个版本。	2013 年 2 月 14 日