Amazon Redshift 查询最佳实践

AWS 规范性指导

AWS 规范性指导: Amazon Redshift 查询最佳实践

Table of Contents

Amazon Redshift 查询最佳实践

Ethan Stark，Amazon Web Services（AWS）

2024 年 6 月（文档历史记录）

概览

本指南提供在 Amazon Redshift 中优化查询和表性能的建议和最佳实践。您可以使用 Amazon Redshift，通过标准 SQL 跨数据仓库和数据湖查询 PB 级的结构化和半结构化数据。本指南还概述了 Amazon Redshift 数据仓库的核心架构组件。这些知识以及对表属性、集群配置和查询结构等查询性能因素的了解，可以帮助您为 Amazon Redshift 数据仓库设计高效且有效的表和查询。

目标受众

本指南适用于在 Amazon Redshift 中设计或使用表和查询的数据工程师、数据架构师和数据分析师。

目标

本指南可以帮助您和您的组织实现以下目标：

- 设计表以实现最佳数据存储和检索操作
- 设计查询以获得最佳性能并节省成本
- 优化 Amazon Redshift Spectrum 的性能，以直接从 Amazon Simple Storage Service（Amazon S3）上的文件查询数据

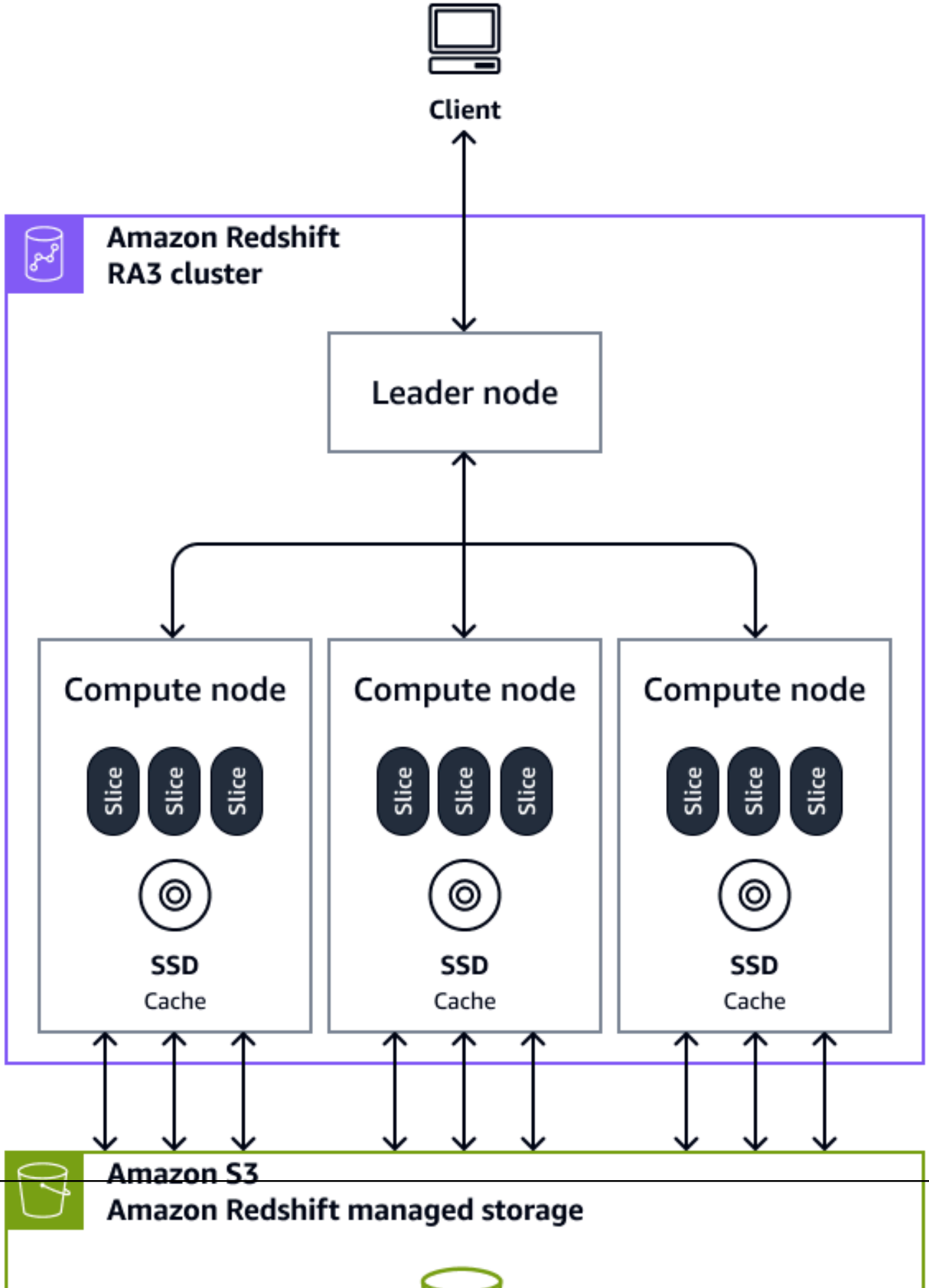
Amazon Redshift 数据仓库的架构组件

我们建议您对 Amazon Redshift 数据仓库中的核心架构组件有基本的了解。这些知识可以帮助您更好地了解如何设计查询和表以实现最佳性能。

Amazon Redshift 中的数据仓库由以下核心架构组件组成：

- 集群：集群由一个或多个计算节点组成，是 Amazon Redshift 数据仓库的核心基础设施组件。计算节点对外部应用程序是透明的，但您的客户端应用程序仅与领导节点直接交互。一个典型的集群包含两个或多个计算节点。计算节点通过领导节点进行协调。
- 领导节点：领导节点管理客户端程序和所有计算节点的通信。每当向集群提交查询时，领导节点还会准备运行查询的计划。计划准备好后，领导节点编译节点、将编译后的节点分发给计算节点，然后将数据切片分配给每个计算节点以处理查询结果。
- 计算节点：计算节点运行查询。领导节点为运行查询的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、内存和连接的磁盘存储。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和存储容量。
- 节点切片：一个计算节点会分成多个称为切片的单元。将为计算节点中的每个切片分配节点的内存和磁盘空间的一部分，用于处理分配给节点的工作负载的一部分。然后，切片将并行工作以完成操作。数据根据特定表的分配方式和分配键在切片之间分配。数据的均匀分布使 Amazon Redshift 能够将工作负载均匀分配给切片，并最大限度地发挥并行处理的优势。每个计算节点的切片数是根据节点的类型决定的。有关更多信息，请参阅 Amazon Redshift 文档中的 Amazon Redshift 中的集群和节点。
- 大规模并行处理（MPP）：Amazon Redshift 使用 MPP 架构来快速处理数据，甚至处理复杂的查询和海量的数据。多个计算节点对数据的各部分运行相同的查询代码，以最大限度地提升并行处理效率。
- 客户端应用程序：Amazon Redshift 与各种数据加载、提取、转换、加载（ETL）、商业智能（BI）报告、数据挖掘和分析工具集成。所有客户端应用程序均仅通过领导节点与集群通信。

下图显示 Amazon Redshift 数据仓库的架构组件如何协同工作以加快查询速度。


Client
Amazon Redshift
RA3 cluster
Leader node
Compute node
Slice
Slice
Slice
SSD
Cache
Compute node
Slice
Slice
Slice
SSD
Cache
Compute node
Slice
Slice
Slice
SSD
Cache
Amazon S3
Amazon Redshift managed storage

查询生命周期分为七个阶段：

1. 查询接收和解析：
   - 领导节点接收查询并解析 SQL。
   - 解析器生成初始查询树，其表示原始查询的逻辑结构。
   - Amazon Redshift 将该查询树馈入查询优化器。
2. 查询优化：
   - 优化器评估查询，并在必要时重写查询以最大限度提升效率。
   - 此优化过程可能涉及创建多个相关查询来替换单个查询。
3. 查询计划生成：
   - 优化器会生成一个查询计划（或如果需要可生成多个计划）以供执行。
   - 查询计划指定执行选项，如联接类型、联接顺序、聚合方法和数据分配要求。
4. 执行引擎转换：
   - 执行引擎将查询计划转换为离散的步骤、分段和流：
     - 步骤：表示查询执行期间所需的单个操作。步骤可以组合起来，使计算节点能够执行查询、联接或其他数据库操作。
     - 分段：组合一个过程可以执行的多个步骤。这是计算节点切片可执行的最小编译单元。（切片是 Amazon Redshift 中并行处理的单元。）
     - 流：分布在可用计算节点切片上的分段集合。
   - 执行引擎基于这些步骤、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，并且消耗的计算容量更少。
   - 领导节点将编译的代码广播到计算节点。
5. 并行执行：
   - 此步骤对每个流执行一次。
   - 计算节点切片并行运行查询分段。
   - 在该过程中，Amazon Redshift 优化网络通信、内存使用和磁盘管理，以将中间结果从一个查询计划步骤传递到下一个。
   - 此优化有助于加快查询执行速度。
6. 流处理：
   - 此步骤对每个流执行一次。
   - 引擎会为每个流创建可执行分段，以实现高效的并行处理。
7. 最终排序与聚合：

- 领导节点处理查询所需的任何最终排序或聚合。
- 完成后，领导节点将结果返回至客户端。

有关架构组件的信息，请参阅 Amazon Redshift 文档中的数据仓库系统架构。

Amazon Redshift 的查询性能因素

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：

- 表属性
  - 排序键（Amazon Redshift Advisor）
  - 数据压缩（自动）
  - 数据分布（自动）
  - 表维护（自动）
- 集群配置
  - 节点类型
  - 节点大小、节点数量和切片
  - 工作负载管理（自动）
  - 短查询加速（自动）
- SQL 查询
  - 查询结构
  - 代码编译

表属性

Amazon Redshift 表是在 Amazon Redshift 中存储数据的基本单元，每个表都有一组决定其行为和可访问性的属性。这些属性包括排序、分发方式、压缩编码等。了解这些属性对于优化 Amazon Redshift 表的性能、安全性和成本效益至关重要。

排序键

Amazon Redshift 根据表的排序键将数据按照排序顺序存储在磁盘中。查询优化器和查询处理器使用有关数据在计算节点内位置的信息来减少必须扫描的数据块数。这样可以通过减少待处理的数据量显著提升查询速度。我们建议您使用排序键来简化 `WHERE` 子句中的筛选器。有关更多信息，请参阅 Amazon Redshift 文档中的使用排序键。

数据压缩

数据压缩降低了存储需求，从而减少了磁盘 I/O 并提高了查询性能。在运行查询时，压缩的数据将读入内存，然后在查询运行时解压缩。通过将少量数据加载到内存中，Amazon Redshift 可以分配更多内存

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使用 Amazon Redshift 中的 AUTO 选项，以在您将表与数据一起加载时应用最佳压缩编码。要了解有关使用自动数据压缩的更多信息，请参阅 Amazon Redshift 文档中的使用自动压缩加载表。

数据分布

Amazon Redshift 根据表的分配方式在计算节点上存储数据。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。我们建议您使用分配键来简化最常见的联接。有关更多信息，请参阅 Amazon Redshift 文档中的使用数据分配方式。

表维护

尽管 Amazon Redshift 为大多数工作负载提供了开箱即用的行业领先性能，但要保持 Amazon Redshift 集群的良好运行仍需要维护。更新和删除数据会产生必须进行清理的无效行；如果追加顺序与排序键不一致，则即使是仅追加表也必须重新排序。

Vacuum

Amazon Redshift 中的 vacuum 操作过程对于 Amazon Redshift 集群的正常运行和维护至关重要。它还会影响查询的性能。由于删除和更新都会标记旧数据，但实际上并未将其删除，因此您必须使用 vacuum 操作来回收被之前的 UPDATE 和 DELETE 操作标记为删除的表行所占用的磁盘空间。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。如果表具有排序键，并且表加载未优化为在其插入时进行排序，则必须使用 vacuum 对数据进行重新排序（这对性能至关重要）。有关更多信息，请参阅 Amazon Redshift 文档中的对表执行 vacuum 操作。

分析

ANALYZE 操作会更新 Amazon Redshift 数据库中表的统计元数据。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 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，ANALYZE 操作将在工作负载较轻的时段自动运行。如果您选择显式运行 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。

- 对您创建的新表和正在进行重大更改的现有表或列运行 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 命令时请使用 PREDICATE COLUMNS 子句。

集群配置

集群是多个节点组的集合，这些节点执行实际的数据存储和处理。如果您想实现以下目标，正确设置 Amazon Redshift 集群至关重要：

- 高可扩展性和并发性
- 高效使用 Amazon Redshift
- 提升性能
- 降低成本

节点类型

Amazon Redshift 集群可以使用多种节点类型之一（RA3 DC2、和 DS2）。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。成本和性能优化始于选择正确的节点类型和大小。有关节点类型的更多信息，请参阅 Amazon Redshift 文档中的 Amazon Redshift 集群概览。

节点大小、节点数量和切片

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。不过，更多的节点也意味着更高的开支。这意味着您必须找到适合系统的成本与性能之间的平衡。有关 Amazon Redshift 集群架构的更多信息，请参阅 Amazon Redshift 文档中的数据仓库系统架构。

工作负载管理

利用 Amazon Redshift 工作负载管理（WLM），用户能够通过优先级灵活地管理工作负载队列，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。自动 WLM 使用机器学习（ML）算法来分析查询，并将其放入具有适当资源的相应队列中，同时管理查询并发性和内存分配。有关 WLM 的更多信息，请参阅 Amazon Redshift 文档中的实施工作负载管理。

短查询加速

短查询加速（SQA）可让短时查询优先于长时查询。SQA 在专用空间中运行查询，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。如果使用 SQA，短时查询会更快地开始运行，您可以更快地看到结果。如果您启用 SQA，则可以减少或消除专用于运行短查询的 WLM 队列。此外，长时查询无需争用 WLM 队列中的槽位。这意味着您可以将 WLM 队列配置为使用较少的查询槽位。如果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 SQA 的更多信息，请参阅 Amazon Redshift 文档中的使用短查询加速。

SQL 查询

数据库查询是对数据库中数据发出的请求。请求应使用 SQL 发送到 Amazon Redshift 集群。Amazon Redshift 支持通过 Java 数据库连接（JDBC）和开放式数据库连接（ODBC）来连接 SQL 客户端工具。您可以使用支持 JDBC 或 ODBC 驱动程序的大多数 SQL 客户端工具。

查询结构

查询的编写方式会显著影响其性能。我们建议您编写查询时，仅处理和返回满足需求必需的最少数据。有关如何构建查询的更多信息，请参阅本指南的设计 Amazon Redshift 查询的最佳实践部分。

代码编译

Amazon Redshift 会为每个查询执行计划生成和编译经过优化的代码。编译代码执行更快，因为它消除了使用解释器的开销。为了最大限度地减少新查询的延迟，同时保留编译代码的性能优势，Amazon Redshift 使用了一种称为组合的技术。组合生成预先存在的逻辑的轻量级排列，以便立即处理新查询，同时在后台编译高度优化的查询专用代码。这会将编译从查询执行的关键路径中移除。这意味着新查询可以更快地启动并提供与后续运行一致的性能。

Amazon Redshift 还使用无服务器编译服务将查询编译扩展到亚马逊 Redshift 集群的计算资源之外。编译后的代码段既可以在集群上本地缓存，也可以在集群重启后保留的几乎无限的远程缓存中。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。通过使用可扩展的编译服务，Amazon Redshift 可以并行编译代码，从而提供始终如一的快速性能。

设计 Amazon Redshift 表的最佳实践

本节概述设计数据库表的最佳实践。我们建议您遵循以下最佳实践，以实现最佳查询性能和效率。

了解排序键的工作原理

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。为了有效使用排序键，我们建议您执行以下操作：

- 尽可能保持表的排序状态。
- 使用 VACUUM 排序以恢复最佳性能。
- 避免压缩排序键列。
- 如果排序键已压缩，且 sortkey1_skew 比率非常高，则在不对排序键启用压缩功能的情况下重新创建表。
- 避免对排序键列应用函数。例如，在以下查询中，如果将 trans_dt : TIMESTAMPTZ 排序键列转换为 DATE，则不使用该列：

```
select order_id, order_amt
from sales
where trans_dt::date = '2021-01-08'::date
```

- 按排序键顺序执行 INSERT 操作。
- 尽可能在 GROUP BY 子句中使用排序键。

查询调优提示

我们建议您执行以下操作以调优查询：

- 始终按从最低基数到最高基数的顺序对复合排序键进行排序以获得最佳效果。
- 如果复合排序键中的前导键相对唯一（即具有较高的基数），则应避免向排序键中添加额外的列。添加额外的列对查询性能影响不大，但会增加维护成本。

评估排序键有效性

要优化查询，您必须能够评估查询的有效性。我们建议您使用 SVL_QUERY_SUMMARY 视图查找有关查询执行的一般信息。在此视图中，您可以使用属性 IS_RRSCAN 来确定 EXPLAIN 计划步骤是否使用范围受限的扫描。您也可以使用属性 rows_pre_filter 来确定排序键的选择性。

你也可以使用 GitHub 名为 v_my_last_query_su mmary 的管理员视图。该视图显示上次运行的查询的信息。

以下语句显示如何查找关于查询执行的一般信息。

```
select lpad(' ',stm+seg+step) || label as label,
       rows,
       bytes,
       is_diskbased,
       is_rrscan,
       rows_pre_filter
from svl_query_summary
where query = pg_last_query_id()
order by stm, seg, step;
```

上述查询返回以下示例输出。

label	rows	bytes	is_diskbased	is_rrscan	rows_pre_filter
scan tbl=163860 name=orders	1500000	24000000	f	f	1500000
project	1500000	0	f	f	0
project	1500000	0	f	f	0
hash tbl=968	1500000	24000000	f	f	0
scan tbl=163852 name=lineitem	6001215	144029160	f	t	6001215
project	6001215	0	f	f	0
project	6001215	0	f	f	0
hjoin tbl=968	6001215	0	f	f	0
project	6001215	0	f	f	0
project	6001215	0	f	f	0

了解您的表

了解表的关键属性很重要。要了解关于表的更多信息，请执行以下操作：

- 使用 PG_TABLE_DEF 查看关于表列的信息。

- 使用 SVV_TABLE_INFO 查看关于表的更全面的信息，包括数据分配偏斜、密钥分配偏斜、表大小和统计数据。

选择正确的表分配方式

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以最大程度地减小重新分配步骤的影响。

我们建议使用以下方法来选择正确的表分配方式：

- 通过将行并置在同一个节点内，避免在查询执行计划中进行广播和重新分配。例如，通过选择 DISTKEY，可在其共用列上分配事实数据表和一维表。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。
- 确保创建分配键的列没有偏度。否则，某个计算节点可能会比其他计算节点执行更多繁重的工作。如果发现偏度，请考虑更改分配键列。如果列值分布均匀或基数值较高，则可以将其视为分配键的候选列。
- 如果联接条件中使用的表较小（小于 1 GB），则考虑分配方式 ALL。
- 您可以压缩分配键，但必须避免压缩排序键列（尤其是排序键的第一列）。

> **Note**
>
> 如果使用自动表优化，则不需要选择表的分配方式。有关更多信息，请参阅 Amazon Redshift 文档中的使用自动表优化。要让 Amazon Redshift 选择适当的分配方式，请指定 AUTO 作为分配方式。

设计 Amazon Redshift 查询的最佳实践

本节概述设计查询的最佳实践。我们建议您遵循本节中的最佳实践，以实现最佳查询性能和效率。

避免使用 SELECT * FROM 语句

我们建议您避免使用 SELECT * FROM 语句。相反，请始终列出要分析的列。这样可以减少查询执行时间，并降低 Amazon Redshift Spectrum 查询的扫描成本。

应避免的情况示例

```
select *
from sales;
```

最佳实践示例

```
select sales_date, sales_amt
from sales;
```

识别查询问题

我们建议您检查 STL_ALERT_EVENT_LOG 视图，以识别并纠正查询中可能存在的问题。

获取有关查询的摘要信息

我们建议您使用 SVL_QUERY_SUMMARY 和 SVL_QUERY_REPORT 视图来获取有关查询的摘要信息。您可以使用此信息来优化查询。

避免交叉联接

除非绝对有必要，否则建议避免使用交叉联接。如果没有联接条件，交叉联接会生成两个表的笛卡尔乘积。交叉联接通常作为嵌套循环联接（可能最慢的联接类型）运行。

应避免的情况示例

```
select c.c_name,
          n.n_name
```

```
from tpch.customer c,
        tpch.nation n;
```

最佳实践示例

```
select c.c_name,
           n.n_name
from tpch.customer c,
join tpch.nation n
  on n.n_nationkey = c.c_nationkey;
```

避免在查询谓词中使用函数

我们建议您避免在查询谓词中使用函数。在查询谓词中使用函数可能会对性能产生负面影响，因为函数通常会给每行增加额外的处理开销，并减慢查询的整体执行速度。

应避免的情况示例

```
select sum(o_totalprice)
from tpch.orders
where datepart(year, o_orderdate) = 1992;
```

最佳实践示例

```
select sum(o_totalprice)
from tpch.orders
where o_orderdate between '1992-01-01' and '1992-12-31';
```

避免不必要的类型转换

我们建议您避免对查询使用不必要的类型转换，因为转换数据类型需要时间和资源，并且会减慢查询的执行速度。

应避免的情况示例

```
select sum(o_totalprice)
from tpch.orders
where o_ordertime::date = '1992-01-01';
```

最佳实践示例

```
select sum(o_totalprice)
from tpch.orders
where o_ordertime between '1992-01-01 00:00:00' and '1992-12-31 23:59:59';
```

使用 CASE 表达式进行复杂聚合

我们建议您使用 CASE 表达式执行复杂聚合，而不是从同一个表多次选择。

应避免的情况示例

```
select sum(sales_amt) as us_sales
from sales
where country = 'US';

select sum(sales_amt) as ca_sales
from sales
where country = 'CA';
```

最佳实践示例

```
select sum(case when country = 'US' then sales_amt end) as us_sales,
           sum(case when country = 'CA' then sales_amt end) as ca_sales
from sales;
```

使用子查询

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（少于 200 行），则我们建议您使用子查询。

应避免的情况示例

如果子查询返回的行数少于 200 行：

```
select sum(order_amt) as total_sales
from sales
where region_key IN
        (select region_key
```

```
        from regions
        where state = 'CA');
```

最佳实践示例

如果子查询返回的行数大于或等于 200 行：

```
select sum(o.order_amt) as total_sales
from sales o
join regions r
  on r.region_key = o.region_key
  and r.state = 'CA';
```

使用谓词

我们建议您使用谓词尽可能限制数据集。在 SQL 中使用谓词以筛选和限制查询中返回的数据。通过使用谓词指定条件，您可以根据指定的条件指定查询结果中必须包含哪些行。这样便可仅检索自己感兴趣的数据，并提高查询的效率和准确性。有关更多信息，请参阅 Amazon Redshift 文档中的条件。

添加谓词以筛选包含联接的表

我们建议您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）。在 SQL 中使用谓词筛选包含联接的表，可以通过减少必须处理的数据量并减小中间结果集的大小来提高查询性能。通过在 WHERE 子句中指定联接操作的条件，查询执行引擎可以在联接前剔除不符合条件的行。这会生成更小的结果集并加快查询执行速度。

应避免的情况示例

```
select p.product_name, sum(o.order_amt)
from sales o
join product p
   on r.product_key = o.product_key
where o.order_date > '2022-01-01';
```

最佳实践示例

```
select p.product_name, sum(o.order_amt)
from sales o
join product p
```

```
  on p.product_key = o.product_key
  and p.added_date > '2022-01-01'
where o.order_date > '2022-01-01';
```

对谓词使用成本最低的运算符

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，而 LIKE 运算符仍优先于 SIMILAR TO 或 POSIX 运算符。

在 GROUP BY 子句中使用排序键

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，接下来按排序键顺序包含其他需要使用的排序键。

利用实体化视图

如果可能，请通过将复杂的代码替换为实体化视图来重写查询，这将显著提升查询的性能。有关更多信息，请参阅 Amazon Redshift 文档中的在 Amazon Redshift 中创建实体化视图。

注意 GROUP BY 和 ORDER BY 子句中的列

如果同时使用 GROUP BY 和 ORDER BY 子句，请确保 GROUP BY 和 ORDER BY 子句中各列的顺序保持一致。GROUP BY 隐式要求数据已排序。如果您的 ORDER BY 子句不同，则必须对数据进行两次排序。

应避免的情况示例

```
select a, b, c, sum(d)
from a_table
group by b, c, a
order by a, b, c
```

最佳实践示例

```
select a, b, c, sum(d)
from a_table
group by a, b, c
```

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

使用 Amazon Redshift Spectrum 的最佳实践

本节概述使用 Amazon Redshift Spectrum 的最佳实践。我们建议您遵循以下最佳实践，以在使用 Redshift Spectrum 时获得最佳性能：

- 请注意，文件类型对 Redshift Spectrum 查询性能有显著影响。为了提高性能，请使用列式编码文件（例如 ORC 或 Parquet），并仅对非常小的维度表使用 CSV 格式。
- 使用基于前缀的分区以利用分区修剪功能。这意味着使用与数据湖中的分区键关联的筛选器。
- Redshift Spectrum 会自动扩展以处理大型请求，因此尽可能多地在 Redshift Spectrum 中完成操作（例如，谓词下推）。
- 注意高频筛选列的分区文件。如果数据按一个或多个筛选列进行分区，Redshift Spectrum 可以利用分区修剪功能，跳过对不需要的分区和文件的扫描。常见做法是根据时间对数据进行分区。
- 您可使用以下查询来检查分区的有效性以及 Redshift Spectrum 查询的效率。

```
Select query,
           segment,
           max(assigned_partitions) as total_partitions,
           max(qualified_partitions) as qualified_partitions
From svl_s3partition
Where query=pg_last_query_id()
Group by 1,2;
```

  上述查询将显示以下内容：

  - total_part itions — 识别的分区数量 AWS Glue Data Catalog
  - qualified_partitions：Redshift Spectrum 查询访问的 Amazon Simple Storage Service（Amazon S3）前缀数量
- 您还可以查看 SVL_S3QUERY_SUMMARY 系统表，以了解分区的有效性以及 Redshift Spectrum 查询的效率。为此，请使用以下语句。

```
Select *
From svl_s3query_summary
Where query=pg_last_query_id();
```

  除了显示分区修剪效率的文件外，上述查询还返回了更多信息，包括 is_partitioned、s3_scanned_rows/bytes 和 s3_returned_rows/bytes 值。

Redshift Spectrum 中的谓词下推

使用谓词下推可以避免消耗 Amazon Redshift 集群中的资源。您可以将许多 SQL 操作下推至 Redshift Spectrum 层。我们建议尽可能利用此功能。

记住以下内容：

- 您可以完全在 Redshift Spectrum 层评估某些类型的 SQL 操作，包括以下各项：
  - `GROUP BY` 子句
  - 比较和模式匹配条件（例如，`LIKE`）
  - 聚合函数（例如，`COUNT`、`SUM`、`AVG`、`MIN` 和 `MAX`）
  - `regex_replace`、`to_upper`、`date_trunc` 和其他函数
- 您无法将某些操作推送到 Redshift Spectrum 层，包括 `DISTINCT` 和 `ORDER BY`。如果可能，请仅在查询的顶层执行 `ORDER BY`，因为排序是在领导节点中完成的。
- 检查您的查询 `EXPLAIN` 计划以验证谓词下推是否有效。要在 `EXPLAIN` 命令中查找 Redshift Spectrum 部分，请查找以下步骤：
  - S3 Seq Scan
  - S3 HashAggregate
  - S3 Query Scan
  - Seq Scan PartitionInfo
  - Partition Loop
- 在查询中使用最少的列数。如果数据采用 Parquet 或 ORC 格式，Redshift Spectrum 可以排除一些要扫描的列。
- 充分利用分区进行并行处理和分区消除，并尽可能将文件大小保持在至少 64 MB。
- 如果您使用 `CREATE EXTERNAL TABLE` 或 `ALTER TABLE`，请设置 `TABLE PROPERTIES 'numRows'='nnn'`。Amazon Redshift 不分析外部表来生成表统计数据，查询优化器会使用这些统计数据来生成查询计划。如果未设置统计信息，则 Amazon Redshift 假设外部表较大，本地表较小。

Redshift Spectrum 的查询调优提示

我们建议您在查询调优时牢记以下几点：

- 您的 Amazon Redshift 集群可参与查询的 Redshift Spectrum 节点数量与集群中的切片数量相关。

- 扩大集群的规模可以提升集群的本地计算配置文件、存储配置文件以及 Amazon S3 数据湖查询的查询能力。
- 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。
- 由于 Redshift Spectrum 会自动扩展以处理大型请求，因此每当能够将处理推至 Redshift Spectrum 层时，整体性能都会提升。

资源

- Amazon Redshift 最佳实践（Amazon Redshift 文档）
- 设计查询的 Amazon Redshift 最佳实践（Amazon Redshift 文档）
- 查询性能优化（Amazon Redshift 文档）
- 查询计划（Amazon Redshift 文档）
- 提高 Amazon Redshift Spectrum 查询性能（Amazon Redshift 文档）
- Understanding the query lifecycle in Amazon Redshift（AWS 规范指引）

文档历史记录

下表介绍了本指南的一些重要更改。如果您希望收到有关未来更新的通知，可以订阅 RSS 源。

变更	说明	日期
已移除 AQUA	我们移除了关于 Advanced Query Accelerator（AQUA）的信息。	2024 年 6 月 14 日
初次发布	—	2023 年 2 月 3 日