SQL 参考資料

AWS Clean Rooms

AWS Clean Rooms: SQL 參考資料

Table of Contents

中的 SQL 概觀 AWS Clean Rooms

歡迎使用 AWS Clean Rooms SQL 參考資料。

AWS Clean Rooms 是以業界標準的結構化查詢語言 (SQL) 為基礎建置，這是一種查詢語言，其中包含用來處理資料庫和資料庫物件的命令和函數。SQL 還強制執行有關使用數據類型，表達式和文字的規則。

下列主題提供有關慣例、命名規則和資料類型的一般資訊：

主題

- SQL 參考慣例
- 命名規則
- 資料類型

若要瞭解您可以在中使用的 SQL 命令、SQL 函數類型以及 SQL 條件 AWS Clean Rooms，請檢閱下列主題：

- 中的 SQL 命令 AWS Clean Rooms
- 中的 SQL 函數 AWS Clean Rooms
- 中的 SQL 條件 AWS Clean Rooms

如需詳細資訊 AWS Clean Rooms，請參閱AWS Clean Rooms 使用者指南和 AWS Clean Rooms API 參考。

SQL 參考慣例

本節說明用來撰寫 SQL 運算式、命令和函數語法的慣例。

字元	描述
CAPS (大寫字母)	大寫字詞為關鍵字。
[ ]	方括號是用來表示選用的引數。方括號中的多個引數，代表您可以選擇任何數目的引數。此外，方括號中不同

字元	描述
	行的引數，代表 剖析器期望引數的排序，等於在語法中所列出的順序。
{ }	大括號表示您需要選擇大括號內的其中一個引數。
|	直立線符號會將您可選擇的引數隔開。
斜體	斜體的字表示預留位置。您必須插入適當的值來取代斜體字詞。
...	省略符號表示您可以重複前一個元素。
'	單引號中的文字表示您必須輸入引號。

命名規則

以下各節說明中的 SQL 命名規則AWS Clean Rooms。

已設定的資料表關聯名稱和欄

可以查詢的成員使用已設定的資料表關聯名稱作為查詢中的資料表名稱 配置的表關聯名稱和配置的表格列可以在查詢中別名。

下列命名規則適用於已設定的資料表關聯名稱、設定的表格資料欄名稱和別名：

- 它們只能使用英數字元、底線 (_) 或連字號 (-) 字元，但不能以連字號開頭或結尾。
  - (僅自訂分析規則) 他們可以使用美元符號 ($)，但不能使用跟隨美元引號字串常數的模式。

    一個以美元引號的字符串常量包括：
    - 美元符號 ($)
    - 零個或多個字符的可選「標籤」
    - 另一個美元符號
    - 構成字符串內容的任意字符序列
    - 美元符號 ($)
    - 開始美元報價的相同標籤
    - 一個美元符號

例如：$$invalid$$

- 它們不能包含連續的連字號 (-) 字元。
- 它們不能以下列任何前置詞開頭：

  padb_, pg_, stcs_, stl_, stll_, stv_, svcs_, svl_, svv_, sys_, systable_
- 它們不能包含反斜線字元 (\)、引號 (') 或非雙引號的空格。
- 如果它們以非字母字元開頭，則必須位於雙引號 (」「) 內。
- 如果它們包含連字號 (-) 字元，則必須位於雙引號 (」「) 內。
- 它們的長度必須介於 1 到 127 個字元之間。
- 保留字必須在雙引號內 (」「)。
- 保留下列列名稱不能用於AWS Clean Rooms（即使有引號）：
  - oid
  - 表情
  - X 分
  - 分鐘
  - X 最大
  - C 最大
  - 感染者

文字

常值或常數是固定的資料值，由字元序列或數字常數組成。

以下命名規則適用於AWS Clean Rooms:

- 支援數字、字元和日期、時間和時間戳記常值。
- 只有TAB,CARRIAGE RETURN(公司註冊處)，以及LINE FEED(LF) 支援來自 Unicode 一般類別 (Cc) 的控制字元。
- SELECT 陳述式不支援直接參照投影清單中的常值。

  例如：

```
SELECT 'test', consumer.first_purchase_day
FROM consumer
```

```
INNER JOIN provider2
ON consumer.hashed_email = provider2.hashedemail
```

保留字

以下是保留字的清單AWS Clean Rooms。

AES128	DELTA32KDESC	LEADING	PRIMARY
AES256ALL	DISTINCT	LEFTLIKE	RAW
ALLOWOVER WRITEANALYSE	DO	LIMIT	READRATIO
ANALYZE	DISABLE	LOCALTIME	RECOVERRE FERENCES
AND	ELSE	LOCALTIMESTAMP	REJECTLOG
ANY	EMPTYASNU LLENABLE	LUN	RESORT
ARRAY	ENCODE	LUNS	RESPECT
AS	ENCRYPT	LZO	RESTORE
ASC	ENCRYPTIONEND	LZOP	RIGHTSELECT
AUTHORIZATION	EXCEPT	MINUS	SESSION_USER
AZ64	EXPLICITFALSE	MOSTLY16	SIMILAR
BACKUPBETWEEN	FOR	MOSTLY32	SNAPSHOT
BINARY	FOREIGN	MOSTLY8NATURAL	SOME
BLANKSASN ULLBOTH	FREEZE	NEW	SYSDATESYSTEM
BYTEDICT	FROM	NOT	TABLE

BZIP2CASE	FULL	NOTNULL	TAG
CAST	GLOBALDICT256	NULL	TDES
CHECK	GLOBALDIC T64KGRANT	NULLSOFF	TEXT255
COLLATE	GROUP	OFFLINEOFFSET	TEXT32KTHEN
COLUMN	GZIPHAVING	OID	TIMESTAMP
CONSTRAINT	IDENTITY	OLD	TO
CREATE	IGNOREILIKE	ON	TOPTRAILING
CREDENTIA LSCROSS	IN	ONLY	TRUE
CURRENT_DATE	INITIALLY	OPEN	TRUNCATEC OLUMNSUNION
CURRENT_TIME	INNER	OR	UNIQUE
CURRENT_T IMESTAMP	INTERSECT	ORDER	UNNEST
CURRENT_USER	INTERVAL	OUTER	USING
CURRENT_U SER_IDDEFAULT	INTO	OVERLAPS	VERBOSE
DEFERRABLE	IS	PARALLELP ARTITION	WALLETWHEN
DEFLATE	ISNULL	PERCENT	WHERE
DEFRAG	JOIN	PERMISSIONS	WITH
DELTA	LANGUAGE	PIVOTPLACING	WITHOUT

資料類型

AWS Clean Rooms 儲存或擷取的每個值都具有一組固定關聯性質的資料類型。資料類型會在資料表建立時宣告，用來限制欄或引數可包含的一組值。

下表列出了您可以在 AWS Clean Rooms 表中使用的數據類型。

資料類型	Aliases	描述
ARRAY	不適用	數組嵌套數據類型
BIGINT	不適用	帶正負號的 8 位元組整數
BOOLEAN	BOOL	邏輯布林值 (true/false)
CHAR	CHARACTER	固定長度的字元字串
DATE	不適用	日曆日期 (年、月、日)
DECIMAL	NUMERIC	可選擇精確度 (有效位數) 的精確數值
DOUBLE PRECISION	FLOAT8、FLOAT	雙精度浮點數
INTEGER	INT	帶正負號的 4 位元組整數
MAP	不適用	映射嵌套數據類型
REAL	FLOAT4	單精度浮點數
SMALLINT	不適用	帶正負號的 2 位元組整數
STRUCT	不適用	結構嵌套數據類型
SUPER	不適用	超集數據類型，包括所有標量類型， AWS Clean Rooms 包括複雜類型，如 ARRAY 和 STRUCT。
TIME	不適用	一天中的時間

資料類型	Aliases	描述
TIMETZ	不適用	一天中的時間，含時區
VARBYTE	VARBINARY、BINARY VARYING	可變長度二進位值
VARCHAR	字元變化	可變長度的字元字串 (使用者定義的限制)

> **Note**
>
> ARRAY、STRUCT 和 MAP 巢狀資料類型目前僅針對自訂分析規則啟用。如需詳細資訊，請參閱 嵌套類型。

多位元組字元

VARCHAR 資料類型支援最多 4 個位元組的 UTF-8 多位元組字元，不支援 5 個位元組或更長的字元。若要針對包含多位元組字元的 VARCHAR 資料欄，計算其大小，請將字元數乘以每個字元的位元組數。例如，如果字串包含 4 個中文字，而每個字的長度是 3 個位元組，那麼您將需要使用 VARCHAR(12) 資料欄來儲存這個字串。

VARCHAR 資料類型不支援下列無效的 UTF-8 碼位：

0xD800 – 0xDFFF (位元組序列：ED A0 80 - ED BF BF)

CHAR 資料類型不支援多位元組字元。

數值類型

主題

- 整數類型
- DECIMAL 或 NUMERIC 類型
- 關於使用 128 位元 DECIMAL 或 NUMERIC 資料欄的備註
- 浮點類型
- 數值的計算

數值資料類型包括整數、小數和符點數。

整數類型

使用 SMALLINT、INTEGER 和 BIGINT 資料類型來儲存各種範圍的整數。您無法將值儲存在每個類型的允許範圍之外。

名稱	儲存	範圍
SMALLINT	2 位元組	-32768 到 +32767
整數或整數	4 位元組	-2147483648 到 +2147483647
BIGINT	8 位元組	-9223372036854775808 到 9223372036854775807

DECIMAL 或 NUMERIC 類型

使用 DECIMAL 或 NUMERIC 資料類型，以使用者定義的精確度來儲存數值。DECIMAL 和 NUMERIC 關鍵字可互換使用。在本文件中，小數是此資料類型的首選用詞。數值一詞通常是用來指稱整數、小數和浮點資料類型。

儲存	範圍
變數，未壓縮的 DECIMAL 類型最多 128 位元。	128 位元帶正負號的整數，具備最高 38 個位數的精確度。

藉由指定 *precision* 和 *scale*，來定義資料表中的 DECIMAL 欄：

```
decimal(precision, scale)
```

precision

整個值中有效位數的總數：小數點兩邊的位數數量。例如，數字 48.2891 的精確度 (有效位數) 為 6，小數位數為 4。如果未指定，預設的精確度為 18，最高精確度為 38。

如果輸入值中小數點左側的位數超過欄的精確度減去其小數位數，則無法將該值複製到欄中 (或插入或更新)。此規則適用於超出資料欄定義範圍之外的任何值。例如，numeric(5,2) 欄的值，其允許的範圍為 -999.99 到 999.99。

scale

數值小數部分中，位於小數點右邊的小數位數數目。整數的小數位數為 0。在資料欄的規格中，小數位數的值必須小於或等於精確度的值。如果未指定，預設的小數位數為 0，最大的小數位數為 37。

如果載入資料表的輸入值，其小數位數大於資料欄的小數位數，則此值會四捨五入至指定的小數位數。例如，SALES 資料表中的 PRICEPAID 資料欄為 DECIMAL(8,2) 資料欄。如果將 DECIMAL(8,4) 值插入 PRICEPAID 資料欄，會將此值四捨五入為 2 個小數位數。

```
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)
```

不過，從資料表所選取值的明確轉換結果，不會四捨五入。

> **Note**
>
> 可以插入 DECIMAL(19,0) 資料欄的正數值上限為 9223372036854775807 (2^{63} -1)。負數值上限為 -9223372036854775807。例如，如果試圖插入數值 9999999999999999999 (19 個 9)，將會造成溢位錯誤。無論小數點的位置何在， AWS Clean Rooms 可以表示為 DECIMAL 數值的最大字串是 9223372036854775807。例如，可以載入 DECIMAL(19,18) 資料欄的最大值為 9.223372036854775807。
> 這些規則是因為以下原因：
>
> - 具有 19 個或更少有效位數的十進制值在內部存儲為 8 字節整數。
> - 具有 20 到 38 個有效位數的十進制值被存儲為 16 字節整數。

關於使用 128 位元 DECIMAL 或 NUMERIC 資料欄的備註

除非您確定應用程式需要該精確度，否則請勿任意指派最大有效位數給 DECIMAL 欄。128 位元值使用的磁碟空間是 64 位元值的兩倍，而且可能會減慢查詢執行時間。

浮點類型

使用 REAL 和 DOUBLE PRECISION 資料類型，以可變精確度來儲存數值。這些是不精確的類型，代表某些數值會以近似值儲存，因此在儲存和傳回特定值時，可能會造成些微的出入。如果您需要精確的儲存和計算 (例如貨幣金額)，請使用 DECIMAL 資料類型。

REAL 表示單精度浮點格式，根據 IEEE 標準 754 用於浮點運算。它具有大約 6 位數的精確度，並且範圍約為 1E-37 到 1E+37。您也可以將此資料類型指定為 FLOAT4。

DOUBLE PRECISION 代表遵循二進位浮點數運算之 IEEE 標準 754 的雙精確度浮點格式。它具有大約 15 位數的精確度，並且範圍約為 1E-307 到 1E+308。您也可以將此資料類型指定為 FLOAT 或 FLOAT8。

數值的計算

在中 AWS Clean Rooms，計算是指二進制數學運算：加法，減法，乘法和除法。本節說明這些運算預期的傳回類型，以及使用 DECIMAL 資料類型時，用來決定精確度與小數位數的特定公式。

在查詢處理作業期間計算數值時，可能會遇到無法進行計算的情況，而且查詢會傳回數值溢位錯誤。您也可能會遇到計算值的小數位數改變或出乎意料的情況。針對某些運算，您可以使用明確轉換 (類型提升) 或 AWS Clean Rooms 設定參數，來解決這些問題。

關於使用 SQL 函式進行類似計算的結果，詳細資訊請參閱 中的 SQL 函數 AWS Clean Rooms。

計算的傳回類型

鑑於中支援的一組數值資料類型 AWS Clean Rooms，下表顯示加法、減法、乘法和除法運算的預期傳回類型。表格左側的第一欄代表計算中的第一個運算元，最上面的列代表第二個運算元。

	木屬	整數	大整特	DECIMAL	FLOAT4	FLOAT8
木屬	SMALLINT	INTEGER	BIGINT	DECIMAL	FLOAT8	FLOAT8
整數	INTEGER	INTEGER	BIGINT	DECIMAL	FLOAT8	FLOAT8
大整特	BIGINT	BIGINT	BIGINT	DECIMAL	FLOAT8	FLOAT8

DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT4	FLOAT8
FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8

計算出 DECIMAL 結果的精確度和小數位數

下表顯示摘要，說明在數學運算傳回 DECIMAL 結果時，用來計算結果精確度和小數位數的規則。在這個表格中，p1 和 s1 代表計算式中第一個運算元的精確度和小數位數，p2 和 s2 代表第二個運算元的精確度和小數位數。(無論這些計算如何，結果的最高精確度為 38、結果的最大小數位數為 38。)

作業	結果的精確度與小數位數
+ 或 -	擴展 = max(s1,s2) 精確度 = max(p1-s1,p2-s2)+1+scale
*	擴展 = s1+s2 精確度 = p1+p2+1
/	擴展 = max(4,s1+p2-s2+1) 精確度 = p1-s1+ s2+scale

例如，SALES 資料表中的 PRICEPAID 和 COMMISSION 資料欄都是 DECIMAL(8,2) 資料欄。如果將 PRICEPAID 除以 COMMISSION (或反過來)，會如下套用公式：

```
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)
```

下列的計算是一般規則，適用於針對 DECIMAL 數值的運算 (使用 UNION、INTERSECT 和 EXCEPT 等集合運算子，或 COALESCE 和 DECODE 等函式)，計算出結果的精確度和小數位數：

```
Scale = max(s1,s2)
Precision = min(max(p1-s1,p2-s2)+scale,19)
```

例如，包含一個 DECIMAL(7,2) 資料欄的 DEC1 資料表，會與包含一個 DECIMAL(15,3) 資料欄的 DEC2 資料表聯結，以產生 DEC3 資料表。DEC3 的結構描述顯示，其資料欄會變成 NUMERIC(15,3) 資料欄。

```
select * from dec1 union select * from dec2;
```

在上述的範例中，會如下套用公式：

```
Precision = min(max(7-2,15-3) + max(2,3), 19)
= 12 + 3 = 15

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

關於除法運算的備註

對於除法作業， divide-by-zero 條件會傳回錯誤。

在計算出精確度和小數位數之後，會套用 100 個小數位數的限制。如果計算結果的小數位數大於 100，除的結果會如下設定小數位數：

- 精確度 = `precision - (scale - max_scale)`
- 擴展 = `max_scale`

如果計算出的精確度大於最高精確度 (38)，則精確度會降低為 38，而小數位數會變成下列算式的結果：`max(38 + scale - precision), min(4, 100))`

溢位狀況

會針對所有數值運算檢查溢位。具有 19 個 (含) 以下有效位數 (精確度) 的 DECIMAL 資料，會儲存為 64 位元整數。具有 19 個 (含) 以上有效位數 (精確度) 的 DECIMAL 資料，會儲存為 128 位元整數。所有 DECIMAL 數值的最高精確度為 38、最大小數位數為 37。當數值超出這些限值時，會出現溢位錯誤，這會同時發生於中間的和最終的結果集：

- 當特定資料值不符合轉換函式所要求的精確度或指定的小數位數時，明確轉換作業會產生執行時期的溢位錯誤。例如，您不能從銷售表中的「價格支付」列中轉換所有值（十進制（8,2）列）並返回十進制（7,3）結果：

  ```
  select pricepaid::decimal(7,3) from sales;
  ERROR:  Numeric data overflow (result precision)
  ```

  之所以發生這個錯誤，是因為 PRICEPAYED 資料行中有些較大的值無法轉換。

- 在乘法運算所產生的結果中，其小數位數是每個運算元小數位數的總和。例如，如果兩個運算元都有 4 個小數位數，結果的小數位數是 8 個，使得小數點的左邊只有 10 個位數。因此，在將兩個都擁有大量小數位數的大數值相乘時，就會相當容易產生溢位狀況。

INTEGER 和 DECIMAL 類型的數值計算

當計算中的其中一個運算元具有 INTEGER 資料類型，而另一個運算元為 DECIMAL 時，INTEGER 運算元會隱含轉換為 DECIMAL。

- 小麥林被轉換為十進制（5,0）
- 整數轉換為十進制（10,0）
- 大 INT 被轉換為十進制（19,0）

例如，如果將 DECIMAL(8,2) 資料欄 SALES.COMMISSION 乘以 SMALLINT 資料欄 SALES.QTYSOLD，此計算式會進行如下的轉換：

```
DECIMAL(8,2) * DECIMAL(5,0)
```

字元類型

字元資料類型包括 CHAR (字元) 和 VARCHAR (可變長度字元)。

儲存與範圍

CHAR 和 VARCHAR 資料類型是以字元組而非字元來定義。CHAR 資料欄只能包含單位元組字元，因此 CHAR(10) 資料欄可包含最大長度為 10 位元組的字串。VARCHAR 可包含多位元組字元，每個字元最多 4 個位元組。例如，VARCHAR(12) 資料欄可包含 12 個單位元組的字元、6 個 2 位元組的字元、4 個 3 位元組的字元，或是 3 個 4 位元組的字元。

名稱	儲存	範圍 (資料欄的寬度)
CHAR 或 CHARACTER	字串的長度，包括多餘的空格 (如果有的話)	4096 位元組
VARCHAR 或 CHARACTER VARYING	4 位元組 + 字元的總位元組數，其中每個字元都可以是 1 到 4 個位元組。	65535 位元組 (64K -1)

CHAR 或 CHARACTER

使用 CHAR 或 CHARACTER 資料欄來儲存固定長度的字串。這些字串會用空格填充，因此 CHAR(10) 資料欄一律會佔 10 個位元組的儲存空間。

```
char(10)
```

未指定長度規格的 CHAR 資料欄，會變成 CHAR(1) 資料欄。

VARCHAR 或 CHARACTER VARYING

使用 VARCHAR 或 CHARACTER VARYING 資料欄，來儲存具有固定限制的可變長度字串。這些字串並未使用空格填充，因此，VARCHAR(120) 資料欄最多可包含 120 個單位元組的字元、60 個 2 位元組的字元、40 個 3 位元組的字元，或是 30 個 4 位元組的字元。

```
varchar(120)
```

多餘空格的意義

CHAR 和 VARCHAR 資料類型都會儲存長度最多 n 個位元組的字串。嘗試將較長的字串儲存到這些類型的資料行中會導致錯誤。但是，如果多餘的字元都是空格 (空白)，則字串會被截斷為最大長度。如果字串短於最大長度，CHAR 值會以空格填充，但 VARCHAR 值則會儲存不含空格的字串。

CHAR 值中的多餘空格在語義上一律不具有意義。這些空格會在您比較兩個 CHAR 值時被忽略、不列入 LENGTH 的計算中，而且會在您將 CHAR 值轉換為另一種字串類型時移除。

在比較值時，VARCHAR 和 CHAR 值中的多餘空格，在語義上會視為不具意義。

長度的計算會傳回 VARCHAR 字元字串的長度，其中也包含多餘的空格。多餘的空格不會列入固定長度字元字串的長度計算。

日期時間 (Datetime) 類型

日期時間 (Datetime) 資料類型包含 DATE、TIME、TIMETZ、TIMESTAMP 與 TIMESTAMPTZ。

主題

儲存與範圍

名稱	儲存	範圍	解析度
DATE	4 位元組	4713 BC 到 294276 AD	1 天
TIME	8 位元組	00:00:00 至 24:00:00	1 毫秒
TIMETZ	8 位元組	00:00:00+1459 至 00:00:00+1459	1 毫秒
TIMESTAMP	8 位元組	4713 BC 到 294276 AD	1 毫秒
TIMESTAMPTZ	8 位元組	4713 BC 到 294276 AD	1 毫秒

DATE

使用 DATE 資料類型來儲存不含時間戳記的簡單日曆日期。

TIME

使用 TIME 資料類型來儲存一天中的時間。

TIME 欄可針對小數秒數，儲存精確度最高 6 位數的數值。

依預設，使用者資料表和 AWS Clean Rooms 系統資料表中的 TIME 值都是國際標準時間 (UTC)。

TIMETZ

使用 TIMETZ 資料類型來儲存具有時區的一天中的時間。

TIMETZ 欄可針對小數秒數，儲存精確度最高 6 位數的數值。

根據預設，TIMETZ 值在使用者資料表和 AWS Clean Rooms 系統資料表中都是 UTC。

TIMESTAMP

使用 TIMESTAMP 資料類型來儲存完整的時間戳記值，其中包含日期和當日的時間。

TIMESTAMP 欄可針對小數秒數，儲存精確度最高 6 位數的數值。

如果將日期插入 TIMESTAMP 欄，或具有部分時間戳記值的日期，則該值會隱含轉換為完整時間戳記值。此完整時間戳記值對於缺少的小時、分鐘和秒具有預設值 (00)。輸入字串中的時區值會遭到忽略。

根據預設，使用者資料表和 AWS Clean Rooms 系統資料表中的時間戳記值都是 UTC。

TIMESTAMPTZ

使用 TIMESTAMPTZ 資料類型來輸入完整的時間戳記值，其中包含日期、當日的時間和時區。當輸入值包含時區時， AWS Clean Rooms 會使用時區將值轉換為 UTC 並儲存 UTC 值。

若要查看受支援時區名稱的清單，請執行下列命令。

```
select my_timezone_names();
```

若要查看受支援時區縮寫的清單，請執行下列命令。

```
select my_timezone_abbrevs();
```

在 IANA 時區資料庫中，也提供了關於時區的最新資訊。

下表提供時區格式的範例。

格式	範例
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 美國時間/太平洋時區
yyyy-mm-dd H: 三分鐘:SS+/-TZ	1997-12-17 07:37:16-08
dd.mm.yyyy hh:mi:ss tz	17.12.1997 07:37:16.00 PST

TIMESTAMPTZ 欄可針對小數秒數，儲存精確度最高 6 位數的數值。

如果將日期插入 TIMESTAMPTZ 欄，或具有部分時間戳記的日期，則該值會隱含轉換為完整時間戳記值。此完整時間戳記值對於缺少的小時、分鐘和秒具有預設值 (00)。

TIMESTAMPTZ 值在使用者資料表中採用 UTC。

日期時間 (Datetime) 類型範例

下列範例說明如何使用支援的日期時間類型。 AWS Clean Rooms

日期範例

以下範例插入具有不同格式的日期並顯示輸出。

```
select * from datetable order by 1;

start_date |  end_date
-----------------------
2008-06-01 | 2008-12-31
2008-06-01 | 2008-12-31
```

如果將時間戳記值插入 DATE 資料欄，會略過時間的部分，只載入日期。

時間範例

以下範例插入具有不同格式的 TIME 和 TIMETZ 值並顯示輸出。

```
select * from timetable order by 1;
start_time |  end_time
```

```
------------------------
 19:11:19  | 20:41:19+00
 19:11:19  | 20:41:19+00
```

時間戳記範例

如果將日期插入 TIMESTAMP 或 TIMESTAMPTZ 資料欄，則時間會預設為午夜。例如，如果插入常值 20081231，則儲存的值為 2008-12-31 00:00:00。

下列範例會插入具有不同格式的時間戳記，並顯示輸出。

```
timeofday
---------------------
2008-06-01 09:59:59
2008-12-31 18:20:00
(2 rows)
```

日期、時間和時間戳記常值

以下是使用由 AWS Clean Rooms支持的日期，時間和時間戳文字的規則。

日期

下表顯示輸入日期，這些日期是您可以載入到 AWS Clean Rooms 資料表中之常值的有效範例。假設預設 MDY DateStyle 模式有效。此模式表示在字串中月份值位於日期值之前，例如 1999-01-08 和 01/02/00。

> **Note**
>
> 載入資料表時，日期或時間戳記常值必須用引號括住。

輸入的日期	完整日期
January 8, 1999	January 8, 1999
1999-01-08	January 8, 1999
1/8/1999	January 8, 1999
01/02/00	2000 年 1 月 2 日

輸入的日期	完整日期
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 (日期的 3 位數部分必須介於 001 到 366 之間)

Times

下表顯示輸入時間，這些時間值是您可以載入到 AWS Clean Rooms 資料表的常值時間值的有效範例。

輸入時間	說明 (時間的部分)
04:05:06.789	4:05 AM 又 6.789 秒
04:05:06	4:05 AM 又 6 秒
04:05	4:05 AM 整
040506	4:05 AM 又 6 秒
04:05 AM	4:05 AM 整；AM 為選用
04:05 PM	4:05 PM 整；小時值必須小於 12。
16:05	4:05 PM 整

時間戳記

下表顯示輸入時間戳記，這些時間戳記是您可以載入到 AWS Clean Rooms 資料表的常值時間值的有效範例。有效的日期常值全都可以和下列的時間常值合併。

輸入的時間戳記 (串接的日期和時間)	說明 (時間的部分)
20080215 04:05:06.789	4:05 AM 又 6.789 秒
20080215 04:05:06	4:05 AM 又 6 秒
20080215 04:05	4:05 AM 整
20080215 040506	4:05 AM 又 6 秒
20080215 04:05 AM	4:05 AM 整；AM 為選用
20080215 04:05 PM	4:05 PM 整；小時值必須小於 12。
20080215 16:05	4:05 PM 整
20080215	午夜 (預設)

特殊的日期時間 (Datetime) 值

下表顯示了可用作日期時間常值和作為日期函數引數的特殊值。這些值需使用單引號，而且會在查詢處理作業進行期間，轉換為一般的時間戳記值。

特殊值	描述
now	轉換為目前交易的開始時間，並傳回毫秒精確度的時間戳記。
today	轉換為適當的日期，並傳回時間戳記，其中時間的部分全部以 0 表示。
tomorrow	轉換為適當的日期，並傳回時間戳記，其中時間的部分全部以 0 表示。
yesterday	轉換為適當的日期，並傳回時間戳記，其中時間的部分全部以 0 表示。

下列範例顯示 now 和 today 如何與 DATEADD 函數搭配使用。

```
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)
```

間隔常值

以下是與所 AWS Clean Rooms支持的間隔文字工作的規則。

使用間隔常值來表示指定的時間期間，例如 `12 hours` 或 `6 weeks`。您可以在需要表示日期時間的條件和表達式中，使用這些間隔常值。

> **Note**
>
> 您無法針對資料 AWS Clean Rooms 表中的資料行使用 INTERVAL 資料類型。

間隔的表示方式，是結合 INTERVAL 關鍵字、數量和支援的日期部分，例如 `INTERVAL '7 days'` 或 `INTERVAL '59 minutes'`。您可以串連幾個數量和單位，來組成更精確的間隔時間，例如：`INTERVAL '7 days, 3 hours, 59 minutes'`。也支援每種單位的縮寫和複數，例如 `5 s`、`5 second` 和 `5 seconds` 是相同的間隔時間。

如果未指定日期部分，則間隔值代表秒。您可以指定小數格式的數量值 (例如：`0.5 days`)。

範例

下列範例顯示不同間隔值的一連串計算。

下面的例子增加了 1 秒到指定的日期。

```
select caldate + interval '1 second' as dateplus from date
```

```
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 00:00:01
(1 row)
```

下面的例子添加 1 分鐘到指定的日期。

```
select caldate + interval '1 minute' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 00:01:00
(1 row)
```

下列範例會將 3 小時 35 分鐘新增至指定日期。

```
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)
```

下列範例會將 52 週新增至指定的日期。

```
select caldate + interval '52 weeks' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2009-12-30 00:00:00
(1 row)
```

下列範例會將 1 週、1 小時、1 分鐘和 1 秒新增至指定日期。

```
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)
```

下面的例子增加了 12 小時（半天）到指定的日期。

```
select caldate + interval '0.5 days' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 12:00:00
(1 row)
```

下列範例會從 2023 年 2 月 15 日起減去 4 個月，結果為 2022 年 10 月 15 日。

```
select date '2023-02-15' - interval '4 months';

?column?
---------------------
2022-10-15 00:00:00
```

下列範例會從 2023 年 3 月 31 日起減去 4 個月，結果為 2022 年 11 月 30 日。計算有將一個月中的天數納入考量。

```
select date '2023-03-31' - interval '4 months';

?column?
---------------------
2022-11-30 00:00:00
```

布林值 (Boolean) 類型

使用 BOOLEAN 資料類型，在單位元組資料欄中儲存 true 和 false 值。下表說明 Boolean 值的三種可能狀態，以及導致狀態的字面值。無論輸入的字串為何，Boolean 資料欄都會分別將「t」和「f」儲存和輸出為 true 與 false。

State	有效的常值	儲存
True	`TRUE 't' 'true' 'y' 'yes' '1'`	1 位元組

State	有效的常值	儲存
False	FALSE 'f' 'false' 'n' 'no' '0'	1 位元組
不明	NULL	1 位元組

您只可以透過 WHERE 子句中述詞的形式使用 IS 比較來檢查布林值。您無法使用 IS 比較來搭配 SELECT 清單中的布林值。

範例

您可以使用 BOOLEAN 資料欄來儲存客戶資料表中每個客戶的「使用中/非作用中」狀態。

```
select * from customer;
custid | active_flag
-------+-------------
   100 | t
```

在此範例中，下列查詢會從 USERS 表格中選取喜歡運動但不喜歡劇院的使用者：

```
select firstname, lastname, likesports, liketheatre
from users
where likesports is true and liketheatre is false
order by userid limit 10;

firstname |  lastname  | likesports | liketheatre
----------+------------+------------+-------------
Alejandro | Rosalez    | t          | f
Akua      | Mansa      | t          | f
Arnav     | Desai      | t          | f
Carlos    | Salazar    | t          | f
Diego     | Ramirez    | t          | f
Efua      | Owusu      | t          | f
John      | Stiles     | t          | f
Jorge     | Souza      | t          | f
Kwaku     | Mensah     | t          | f
Kwesi     | Manu       | t          | f
(10 rows)
```

下列範例中的查詢會從 USERS 資料表中選取不確定是否喜歡搖滾樂的使用者。

```
select firstname, lastname, likerock
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
----------+----------+----------
Alejandro | Rosalez   |
Carlos    | Salazar   |
Diego     | Ramirez   |
John      | Stiles    |
Kwaku     | Mensah    |
Martha    | Rivera    |
Mateo     | Jackson   |
Paulo     | Santos    |
Richard   | Roe       |
Saanvi    | Sarkar    |
(10 rows)
```

下列範例會傳回錯誤，因為它在 SELECT 清單中使用 IS 比較。

```
select firstname, lastname, likerock is true as "check"
from users
order by userid limit 10;

[Amazon](500310) Invalid operation: Not implemented
```

下列範例會成功，因為它在 SELECT 清單中使用相等的比較 (=)，而非IS比較。

```
select firstname, lastname, likerock = true as "check"
from users
order by userid limit 10;

firstname | lastname  | check
----------+-----------+------
Alejandro | Rosalez   |
Carlos    | Salazar   |
Diego     | Ramirez   | true
John      | Stiles    |
Kwaku     | Mensah    | true
Martha    | Rivera    | true
```

```
Mateo     | Jackson   |
Paulo     | Santos    | false
Richard   | Roe       |
Saanvi    | Sarkar    |
```

SUPER 類型

使用 SUPER 資料類型將半結構化資料或文件儲存為值。

半結構化資料不符合 SQL 資料庫中使用的關聯式資料模型的剛性和表格結構。SUPER 資料類型包含參照資料中不同實體的標籤。SUPER 資料類型可以包含複雜的值，例如陣列、巢狀結構，以及其他與序列化格式相關聯的複雜結構，例如 JSON。SUPER 數據類型是一組包含所有其他標量類型的無模式數組和結構值。 AWS Clean Rooms

SUPER 資料類型最多支援 1 MB 的個別 SUPER 欄位或物件的資料。

SUPER 資料類型具有下列屬性：

- 一個 AWS Clean Rooms 標量值：
  - Null
  - 布林值
  - 數字，例如 smallint、integer、bigint、decimal 或浮點數 (例如 float4 或 float8)
  - 字串值，如 varchar 或 char
- 複雜值：
  - 值的陣列，包括純量或複雜
  - 一種結構，也稱為元組或物件，它是屬性名稱和值 (純量或複雜) 的映射

這兩種類型的複雜值中的任何一種都包含它們自己的純量或複雜值，而沒有任何正規性限制。

SUPER 資料類型支援在無結構描述形式的半結構化資料的持久性。儘管階層式資料模型可能會發生變化，但舊版本的資料可以共存於同一 SUPER 欄中。

嵌套類型

AWS Clean Rooms 支援包含巢狀資料類型的資料查詢，特別是 AWS Glue 結構、陣列和對映資料行類型。只有自訂分析規則支援巢狀資料類型。

值得注意的是，巢狀資料類型不符合 SQL 資料庫關聯式資料模型的剛性、表格結構。

嵌套數據類型包含引用數據中不同實體的標籤。它們可以包含複雜值，例如陣列、巢狀結構，以及與序列化格式相關聯的其他複雜結構，例如 JSON。巢狀資料類型針對個別巢狀資料類型欄位或物件最多支援 1 MB 的資料。

嵌套數據類型的例子

對於`struct<given:varchar, family:varchar>`類型，有兩個屬性名稱：`given`、和`family`，每個屬性名稱對應於一個`varchar`值。

對於`array<varchar>`類型，陣列會指定為的清單`varchar`。

該`array<struct<shipdate:timestamp, price:double>>`類型是指具有類`struct<shipdate:timestamp, price:double>`型的元素列表。

數`map`據類型`array`的行為類似`structs`，其中數組中每個元素的屬性名稱由表示，`key`並映射到. `value`

Example

例如，`map<varchar(20), varchar(20)>`類型被視為`array<struct<key:varchar(20), value:varchar(20)>>`，其中`key`並`value`引用基礎數據中映射的屬性。

如需如何 AWS Clean Rooms 啟用導覽至陣列和結構的資訊，請參閱Navigation (導覽)。

如需如何透過使用查詢的 FROM 子句導覽陣列來 AWS Clean Rooms 啟用陣列反覆運算的資訊，請參閱取消巢狀查詢。

VARBYTE 類型

使用 VARBYTE、VARBINARY 或 BINARY VARYING 欄來儲存具有固定限制的可變長度二進位值。

```
varbyte [ (n) ]
```

最大位元組數 (n) 的範圍可以介於 1 - 1,024,000 之間。預設值為 64,000。

可以使用 VARBYTE 資料類型的一些範例如下：

- 在 VARBYTE 欄上聯結資料表。
- 建立包含 VARBYTE 欄的具體化視觀表。支援包含 VARBYTE 欄的具體化視觀表累加式重新整理。但是，VARBYTE 欄上的 COUNT、MIN 和 MAX 和 GROUP BY 以外的彙總函數不支援累加式重新整理。

若要確保所有位元組都是可列印的字元，請 AWS Clean Rooms 使用十六進位格式來列印 VARBYTE 值。例如，下列 SQL 會將十六進位字串 6162 轉換成二進位值。即使傳回的值是二進位值，結果仍會列印為十六進位 6162。

```
select from_hex('6162');

 from_hex
----------
 6162
```

AWS Clean Rooms 支持在 VARBYTE 和以下數據類型之間進行轉換：

- CHAR
- VARCHAR
- SMALLINT
- INTEGER
- BIGINT

下列 SQL 陳述式將 VARCHAR 字串轉換為 VARBYTE。即使傳回的值是二進位值，結果仍會列印為十六進位 616263。

```
select 'abc'::varbyte;

 varbyte
---------
 616263
```

下列 SQL 陳述式將欄中的 CHAR 值轉換為 VARBYTE。這個範例會建立一個包含 CHAR(10) 欄 (c) 的資料表，插入長度小於 10 的字元值。產生的轉換將結果以空格字元 (hex'20') 填入定義的欄大小。即使傳回的值是二進位值，結果仍會列印為十六進位。

```
create table t (c char(10));
insert into t values ('aa'), ('abc');
select c::varbyte from t;
          c
----------------------
 6161202020202020202020
 6162632020202020202020
```

下列 SQL 陳述式將 SMALLINT 字串轉換為 VARBYTE。即使傳回的值是二進位值，結果仍會列印為十六進位 0005，即兩個位元組或四個十六進位字元。

```
select 5::smallint::varbyte;

 varbyte
---------
 0005
```

下列 SQL 陳述式將 INTEGER 轉換為 VARBYTE。即使傳回的值是二進位值，結果仍會列印為十六進位 00000005，即四個位元組或八個十六進位字元。

```
select 5::int::varbyte;

 varbyte
----------
 00000005
```

下列 SQL 陳述式將 BIGINT 轉換為 VARBYTE。即使傳回的值是二進位值，結果仍會列印為十六進位 0000000000000005，即八個位元組或 16 個十六進位字元。

```
select 5::bigint::varbyte;

     varbyte
------------------
 0000000000000005
```

搭配使用 VARBYTE 資料類型時的限制 AWS Clean Rooms

以下是搭 AWS Clean Rooms配使用 VARBYTE 資料類型時的限制：

- AWS Clean Rooms 僅支援「實木地板」和「ORC」檔案的 VARBYTE 資料類型。
- AWS Clean Rooms 查詢編輯器尚未完全支持 VARBYTE 數據類型。因此，在使用 VARBYTE 運算式時，請使用不同的 SQL 用戶端。

  作為使用查詢編輯器的解決方法，如果資料長度低於 64 KB 且內容是有效的 UTF-8，則可以將 VARBYTE 值轉換為 VARCHAR，例如：

```
select to_varbyte('6162', 'hex')::varchar;
```

- 您不能將 VARBYTE 資料類別搭配 Python 或 Lambda 使用者定義函數 (UDF) 使用。
- 您無法從 VARBYTE 欄建立 HLLSKETCH 欄，也無法在 VARBYTE 欄上使用 APPROXIMATE COUNT DISTINCT。

類型相容性與轉換

下列討論說明類型轉換規則和資料類型相容性在中的運作方式 AWS Clean Rooms。

相容性

在資料庫各種操作的作業期間，會進行資料類型的比對，以及字面值與常數和資料類型的比對，包括下列的操作：

- 對資料表進行的資料處理語言 (DML) 操作
- UNION、INTERSECT 和 EXCEPT 查詢
- CASE 表達式
- 述詞的評估，例如 LIKE 和 IN
- 針對進行資料比較或擷取的 SQL 函式，進行評估
- 數學運算子的比較

這些操作的結果，取決於類型轉換規則和資料類型的相容性。兼容性意味著並不總是需要 one-to-one 匹配某個值和特定數據類型。由於某些資料類型相容，因此可能會隱含轉換或強制轉換。如需詳細資訊，請參閱 隱含轉換類型。當資料類型不相容時，有時您可以使用明確的轉換函式，來將值從一種資料類型轉換為另一種。

一般相容性與轉換規則

請注意下列的相容性與轉換規則：

- 一般而言，屬於相同類型類別的資料類型 (例如不同的數值資料類型)，彼此可以相容和隱含轉換。

  例如，進行隱含轉換時，您可以將小數值插入整數資料欄。小數會經過四捨五入而變成整數。或者，您可以從日期中擷取 2008 等數值，然後將該數值插入整數資料欄。

- 數值資料類型會強制執行嘗試插入 out-of-range 值時發生的溢位情況。例如，精確度為 5 的小數值，不符合精確度定義為 4 的小數資料欄。整數或小數的整數部分永遠不會被截斷。但是，小數的小數部分可以適當地向上或向下四捨五入。不過，從資料表所選取值的明確轉換結果，不會四捨五入。
- 不同類型的字符串是兼容的。包含單字節數據和 CHAR 列字符串的 VARCHAR 列字符串是可比較的和隱式轉換的。包含多位元組資料的 VARCHAR 字串並不相容。此外，如果字串是適當的常值，您可以將字元字串轉換為日期、時間戳記或數值。任何前導或尾隨空格都會被忽略。相反地，您也可以將日期、時間、時間戳記或數值，轉換為固定長度或可變長度的字元字串。

> **Note**
>
> 您想要轉換為數值類型的字元字串，必須包含表示數字的字元。例如，您可以將字串`'1.0'`轉換`'5.9'`為十進位值，但無法`'ABC'`將字串轉換為任何數值類型。

- 如果您比較 DECIMAL 值與字元字串，會 AWS Clean Rooms 嘗試將字元字串轉換為 DECIMAL 值。比較所有其他數值和字元字串時，數值會轉換為字元字串。若要強制執行相反的轉換 (例如，將字元字串轉換為整數，或將 DECIMAL 值轉換為字元字串)，請使用明確函數，例如 CAST 函數。
- 若要將 64 位元的 DECIMAL 或 NUMERIC 值轉換為較高的精確度，您必須使用明確轉換函式，例如 CAST 或 CONVERT 函式。
- 將 DATE 或 TIMESTAMP 轉換為 TIMESTAMPTZ，或將 TIME 轉換為 TIMETZ 時，時區會設定為目前的工作階段時區。工作階段預設的時區為 UTC。
- 同樣地，TIMESTAMPTZ 轉換為 DATE、TIME 或 TIMESTAMP 時，也會使用目前工作階段的時區。工作階段預設的時區為 UTC。在轉換之後，會去掉時區的資訊。
- 用來表示指定時區之時間戳記的字元字串，會使用目前工作階段時區 (預設為 UTC) 轉換為 TIMESTAMPTZ。同樣地，代表指定時區之時間的字元字串，也會使用目前的工作階段時區 (預設為 UTC) 轉換為 TIMETZ。

隱含轉換類型

隱含轉換有兩種：

- 指派中的隱含轉換，例如在 INSERT 或 UPDATE 命令中設定值
- 運算式中的隱含轉換，例如在 WHERE 子句中執行比較

下表列出可在指派或運算式中隱含轉換的資料類型。您也可以使用明確轉換函式來進行這些轉換。

轉換前的類型	轉換後的類型
BIGINT	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
CHAR	VARCHAR
DATE	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT
	CHAR
	DOUBLE PRECISION (FLOAT8)
	整數)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR

轉換前的類型	轉換後的類型
DOUBLE PRECISION (FLOAT8)	BIGINT
	CHAR
	DECIMAL (NUMERIC)
	整數（整數）
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
整數（整數）	BIGINT
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
REAL (FLOAT4)	BIGINT
	CHAR
	DECIMAL (NUMERIC)
	整數（整數）
	SMALLINT

轉換前的類型	轉換後的類型
	VARCHAR
SMALLINT	BIGINT
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	整數（整數）
	REAL (FLOAT4)
	VARCHAR
TIMESTAMP	CHAR
	DATE
	VARCHAR
	TIMESTAMPTZ
	TIME
TIMESTAMPTZ	CHAR
	DATE
	VARCHAR
	TIMESTAMP
	TIMETZ
TIME	VARCHAR

轉換前的類型	轉換後的類型
	TIMETZ
TIMETZ	VARCHAR
	TIME

> **Note**
>
> TIMESTAMPTZ、TIMESTAMP、DATE、TIME、TIMETZ 或字元字串之間的隱含轉換，會使用目前工作階段的時區。
> VARBYTE 資料類型無法隱含轉換至任何其他資料類型。如需更多詳細資訊，請參閱 CAST 函數。

中的 SQL 命令 AWS Clean Rooms

中支援下列 SQL 指令 AWS Clean Rooms：

主題

- SELECT

SELECT

SELECT 命令從表和用戶定義函數返回行。

以下是中支援的「選取 SQL」指令 AWS Clean Rooms：

主題

- SELECT list
- WITH 子句
- FROM 子句
- WHERE 子句
- GROUP BY 子句
- HAVING 子句
- 設定運算子
- ORDER BY 子句
- 子查詢範例
- 相互關聯子查詢

SELECT list

您希望查詢傳回的資料行、函數和運算式的SELECT list名稱。清單查詢的輸出。

語法

```
SELECT
[ TOP number ]
[ DISTINCT ] | expression [ AS column_alias ] [, ...]
```

參數

TOP *##*

TOP需要一個正整數作為它的參數，它定義了返回給客戶端的行數。TOP子句的行為與子LIMIT句的行為相同。傳回的資料列數目是固定的，但是資料列集合不是固定的。若要傳回一致的資料列集合，請使用TOP或LIMIT搭配子ORDERBY句。

DISTINCT

此選項會根據一個或多個資料欄中相符的值，從結果集中消除重複的資料列。

###

表達式是由查詢所參考資料表中的一個或多個資料欄構成。表達式可包含 SQL 函數。例如：

```
coalesce(dimension, 'stringifnull') AS column_alias
```

AS column_alias

資料欄的暫時名稱，會在最終結果集中使用。該AS關鍵字是可選的。例如：

```
coalesce(dimension, 'stringifnull') AS dimensioncomplete
```

若您沒有為表達式指定非簡單資料欄名稱的別名，結果集將會套用預設名稱至該資料欄。

> **Note**
>
> 別名在目標清單中定義之後立即直接辨識。您無法在相同目標清單之後定義的其他運算式中使用別名。

使用須知

TOP是一個 SQL 擴展。 TOP提供了LIMIT行為的替代方法。你不能在同一個查詢LIMIT中使用TOP和。

WITH 子句

WITH 子句是選用的子句，位於查詢中的 SELECT 前面。WITH 子句會定義一個或多個 common_table_expressions。每個通用資料表運算式 (CTE) 都會定義一個暫存資料表，與檢視定

義類似。您可以在 FROM 子句中參考這些暫存資料表。這些資料表僅會在其所屬的查詢執行時使用。WITH 子句中的每個 CTE 都會指定資料表名稱、選用的資料欄名稱清單，以及判斷值為資料表的查詢表達式 (SELECT 陳述式)。

WITH 子句子查詢是定義資料表時較有效率的方式，可在執行單一查詢的過程中使用。在所有任何情況下，於 SELECT 陳述式的本體中使用子查詢都可產生相同的結果，但 WITH 子句子查詢對於寫入和讀取來說可能較為簡單。參考多次的 WITH 子句子查詢會盡可能最佳化為通用子表達式；也就是說，或許可以評估 WITH 子查詢一次並重複使用其結果 (請注意，通用子表達式不限於 WITH 子句中所定義者)。

語法

```
[ WITH  common_table_expression [, common_table_expression , ...] ]
```

其中通用表達式可以是非遞歸的。以下是非遞迴形式：

```
CTE_table_name AS ( query )
```

參數

common_table_expression

定義可在 FROM 子句 中參照的暫存資料表，且僅在執行該資料表所屬的查詢期間使用此暫存資料表。

CTE_table_name

此臨時資料表的唯一名稱會定義 WITH 子句子查詢的結果。您無法在單一 WITH 子句內使用重複的名稱。每個子查詢都必須有可在 FROM 子句 中參考的資料表名稱。

query

任何 AWS Clean Rooms 支持的 SELECT 查詢。請參閱 SELECT。

使用須知

您可以在下列 SQL 陳述式中使用 WITH 子句：

- 選取、使用、聯集、交集及除外

如果查詢的 FROM 子句包含 WITH 子句，但未參考 WITH 子句定義的任何資料表，則會忽略 WITH 子句，而查詢會照常執行。

WITH 子句子查詢定義的資料表只能在 WITH 子句開始的 SELECT 查詢範圍內參考。例如，您可以在 SELECT 清單、WHERE 子句或 HAVING 子句中，子查詢的 FROM 子句內參考這類資料表。您無法在子查詢中使用 WITH 子句，並於主查詢或其他子查詢的 FROM 子句內參考其資料表。此查詢模式會針對 WITH 子句資料表產生 `relation table_name doesn't exist` 形式的錯誤訊息。

您無法在 WITH 子句子查詢內指定另一個 WITH 子句。

您無法對 WITH 子句子查詢定義的資料表進行向前參考。例如，以下查詢會傳回錯誤訊息，因為資料表 W1 的定義中有對資料表 W2 的向前參考：

```
with w1 as (select * from w2), w2 as (select * from w1)
select * from sales;
ERROR:  relation "w2" does not exist
```

範例

下列範例顯示包含 WITH 子句的最簡單查詢案例。名為 VENUECOPY 的 WITH 查詢會從 VENUE 資料表選取所有資料列。主查詢會接著從 VENUECOPY 選取所有資料列。VENUECOPY 資料表僅在此查詢期間存在。

```
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)
```

下列範例顯示 WITH 子句，它會產生兩個資料表，分別名為 VENUE_SALES 和 TOP_VENUES。第二個 WITH 查詢資料表會從第一個資料表選取。接著主查詢區塊的 WHERE 子句會包含限制 TOP_VENUES 資料表的子查詢。

```
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)
```

以下兩個範例將示範根據 WITH 子句子查詢的資料表參考範圍規則。第一個查詢會執行，但第二個會失敗，並產生預期的錯誤。第一個查詢會在主查詢的 SELECT 清單內包含 WITH 子句子查詢。WITH 子句定義的資料表 (HOLIDAYS) 會在 SELECT 清單中子查詢的 FROM 子句中參考：

```
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)
```

第二個查詢會失敗，因為它會嘗試參考主查詢以及 SELECT 清單子查詢中的 HOLIDAYS 資料表。而主查詢參考超出範圍。

```
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
```

FROM 子句

查詢中的 FROM 子句列出資料表參考 (資料表、檢視和子查詢)，此為選取資料的來源位置。若列出多個資料表參考，則必須在 FROM 子句或 WHERE 子句中使用適當的語法聯結資料表。若未指定聯結條件，則系統會將查詢當做交叉聯結 (笛卡兒乘積) 處理。

主題

語法

```
FROM table_reference [, ...]
```

其中 table_reference 是下列其中一項：

```
with_subquery_table_name | table_name | ( subquery ) [ [ AS ] alias ]
table_reference [ NATURAL ] join_type table_reference [ USING ( join_column [, ...] ) ]
table_reference [ INNER ] join_type table_reference ON expr
```

參數

with_subquery_table_name

WITH 子句 中子查詢所定義的資料表。

table_name

資料表或檢視的名稱。

alias

資料表或檢視的暫時替代名稱。必須為衍生自子查詢的資料表提供別名。在其他資料表參考中，別名是選用的。AS關鍵字始終是可選的。資料表別名提供了方便在查詢的其他部分中識別資料表的捷徑，例如 WHERE 子句。

例如：

```
select * from sales s, listing l
where s.listid=l.listid
```

如果定義了一個表別名定義，那麼別名必須用於在查詢中引用該表。

例如，如果查詢是SELECT "tbl"."col" FROM "tbl" AS "t"，則查詢將失敗，因為現在基本上已覆寫資料表名稱。在這種情況下，一個有效的查詢將是SELECT "t"."col" FROM "tbl" AS "t"。

column_alias

資料表或檢視中資料欄的暫時替代名稱。

subquery

判斷值為資料表的查詢表達式。資料表只會在查詢期間存在，通常會為其命名或提供別名。不過，不需要別名。您也可以為衍生自子查詢的資料表定義資料欄名稱。當您想要將子查詢的結果與其他資料表聯結時，以及您想要在查詢中的其他位置選取或限制這些資料欄時，為資料欄指定別名就很重要。

子查詢可包含 ORDER BY 子句，但是，若未指定 LIMIT 或 OFFSET 子句，則此子句不一定有作用。

NATURAL

定義聯結，此聯結會自動使用兩個資料表中所有同名資料欄的配對做為聯結資料欄。不需要明確的聯結條件。例如，若 CATEGORY 和 EVENT 資料表都有名為 CATID 的資料欄，則這兩個資料表的 natural 聯結會是透過其 CATID 資料欄的聯結。

> **Note**
>
> 若已指定 NATURAL 聯結，但是要聯結的資料表中並沒有同名的資料欄配對，則查詢會預設為交叉聯結。

join_type

指定下列其中一種聯結類型：

- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- CROSS JOIN

交叉聯結是沒有限定的聯結，會傳回兩個資料表的笛卡兒乘積。

內部和外部聯結為限定聯結。它們會以隱含方式 (在 natural 聯結中)、在 FROM 子句中使用 ON 或 USING 語法，或使用 WHERE 子句條件限定。

內部聯結只會根據聯結條件或聯結資料欄清單傳回相符的資料列。外部聯結會傳回對等內部聯結傳回的所有資料列，加上「左側」資料表和/或「右側」資料表的不相符資料列。左側資料表是最先列出的資料表，右側資料表是其次列出的資料表。不相符的資料列包含要填入輸出資料欄中空處的 NULL 值。

ON join_condition

聯結規格的類型，其中聯結資料欄會做為條件陳述，後面接著 ON 關鍵字。例如：

```
sales join listing
on sales.listid=listing.listid and sales.eventid=listing.eventid
```

USING (join_column [, ...])

聯結規格的類型，其中聯結資料欄會在括號內列出。若指定了多個聯結資料欄，則會以逗號分隔。USING 關鍵字必須放在清單前面。例如：

```
sales join listing
using (listid,eventid)
```

使用須知

聯結資料欄必須採用可比較的資料類型。

NATURAL 或 USING 聯結只會針對中繼結果集內每個聯結資料欄配對保留一個。

使用 ON 語法的聯結則會保留其中繼結果集內的兩個聯結資料欄。

另請參閱 WITH 子句。

JOIN 範例

SQL JOIN 子句用於根據通用欄位，結合兩個或多個資料表中的資料。結果可能會或可能不會改變，具體取決於指定的聯結方法。如需 JOIN 子句語法的相關資訊，請參閱 參數。

下列查詢是 LISTING 資料表和 SALES 資料表之間的內部聯結 (沒有 JOIN 關鍵字)，其中 LISTING 資料表中的 LISTID 是介於 1 和 5 之間。此查詢會比對 LISTING 資料表 (左側資料表) 和 SALES 資料表 (右側資料表) 中 LISTID 資料欄的值。結果顯示 LISTID 1、4 和 5 符合條件。

```
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
```

下列查詢是左側外部聯結。未在另一個資料表中找到相符項目時，左和右外部聯結會保留來自其中一個聯結資料表的值。左側和右側資料表分別是語法中最先和其次列出的資料表。NULL 值會用來填入結果集中的「空處」。此查詢會比對 LISTING 資料表 (左側資料表) 和 SALES 資料表 (右側資料表) 中 LISTID 資料欄的值。結果顯示，LISTID 2 和 3 並未產生任何銷售。

```
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
```

下列查詢是右側外部聯結。此查詢會比對 LISTING 資料表 (左側資料表) 和 SALES 資料表 (右側資料表) 中 LISTID 資料欄的值。結果顯示 LISTID 1、4 和 5 符合條件。

```
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
```

下列查詢是完全聯結。未在另一個資料表中找到相符項目時，完全聯結會保留來自聯結資料表的值。左側和右側資料表分別是語法中最先和其次列出的資料表。NULL 值會用來填入結果集中的「空處」。此查詢會比對 LISTING 資料表 (左側資料表) 和 SALES 資料表 (右側資料表) 中 LISTID 資料欄的值。結果顯示，LISTID 2 和 3 並未產生任何銷售。

```
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
```

下列查詢是完全聯結。此查詢會比對 LISTING 資料表 (左側資料表) 和 SALES 資料表 (右側資料表) 中 LISTID 資料欄的值。只有不會產生任何銷售 (ListID 2 和 3) 的資料列才會顯示在結果中。

```
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
```

下列範例是一個內部聯結搭配 ON 子句。在此情況下，不會傳回 NULL 資料列。

```
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
```

下列查詢是 LISTING 資料表和 SALES 資料表的交叉聯結或笛卡爾聯結，並且使用述詞來限制結果。此查詢會在 SALES 資料表和 LISTING 資料表中比對 LISTID 資料欄值，以找出兩個資料表中的 LISTID 1、2、3、4 和 5。結果顯示 20 個符合條件的資料列。

```
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
```

下列範例是兩個資料表之間的自然聯結。在這種情況下，兩個資料表中的 listid、sellerid、eventid 和 dateid 資料欄會具有相同的名稱和資料類型，因此會被用來作為聯結資料欄。結果限制為 5 個資料列。

```
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
```

下列範例是兩個資料表之間的聯結和 USING 子句。在這種情況下，資料欄 listid 和 eventid 會被用來作為聯結資料欄。結果限制為 5 個資料列。

```
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
```

以下查詢為 FROM 子句中兩個子查詢的內部聯結。查詢會尋找不同類別活動 (演奏會和表演) 的已售出和未售出票券數目。FROM 子句子查詢是資料表子查詢，可傳回多個資料欄和資料列。

```
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 子句包含聯結資料表或套用述詞至資料表中資料欄的條件。資料表可以藉由在 WHERE 子句或 FROM 子句中使用適當的語法進行內部聯結。外部連結條件必須在 FROM 子句中指定。

語法

```
[ WHERE condition ]
```

條件

任何產生布林值結果的搜尋條件，例如，資料表資料欄的聯結條件或述詞。以下範例為有效的聯結條件：

```
sales.listid=listing.listid
sales.listid<>listing.listid
```

以下範例對於資料表中的資料欄是有效的條件：

```
catgroup like 'S%'
venueseats between 20000 and 50000
eventname in('Jersey Boys','Spamalot')
```

```
year=2008
length(catdesc)>25
date_part(month, caldate)=6
```

條件可分成簡單和複雜；若是複雜條件，您可以使用括號來隔離邏輯單位。在下列範例中，聯結條件會以括號包圍。

```
where (category.catid=event.catid) and category.catid in(6,7,8)
```

使用須知

您無法在 WHERE 子句中使用別名來參考選取清單表達式。

您無法限制 WHERE 子句中彙整函數的結果；請使用 HAVING 子句達成此目的。

WHERE 子句中限制的資料欄必須衍生自 FROM 子句中的資料表參考。

範例

以下查詢使用不同 WHERE 子句限制的組合，包括 SALES 和 EVENT 資料表的聯結條件、EVENTNAME 資料欄上的述詞，以及 STARTTIME 資料欄上的兩個述詞。

```
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)
```

GROUP BY 子句

GROUP BY 子句會識別查詢的分組資料欄。分組資料欄必須在查詢使用標準函數運算彙整時宣告，像是 SUM、AVG 和 COUNT。如果 SELECT 運算式中存在彙總函數，則 SELECT 運算式中不在彙總函數中的任何資料行都必須位於 GROUP BY 子句中。

如需詳細資訊，請參閱 中的 SQL 函數 AWS Clean Rooms。

語法

```
GROUP BY group_by_clause [, ...]

group_by_clause := {
    expr |
        ROLLUP ( expr [, ...] ) |
        }
```

參數

expr

在查詢的選取清單中，資料欄或表達式的清單必須符合非彙整表達式的清單。例如，請考量以下簡單查詢。

```
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)
```

在此查詢中，選取清單是由兩個彙整表達式所構成。第一個使用 SUM 函數，第二個使用 COUNT 函數。其餘兩個資料欄 LISTID 和 EVENTID 必須宣告為分組資料欄。

GROUP BY 子句中的表達式也可以使用序數來參考選取清單。例如，前一個範例可縮減如下。

```
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)
```

ROLLUP

您可以使用彙總擴充功能 ROLUP，在單一陳述式中執行多個 GROUP BY 作業的工作。如需彙總延伸項目及相關函數的相關資訊，請參閱 彙總延伸項目。

彙總延伸項目

AWS Clean Rooms 支援彙總擴充功能，以在單一陳述式中執行多個 GROUP BY 作業的工作。

GROUPING SETS

在單一陳述式中計算一或多個群組集。群組集是單一 GROUP BY 子句的集合，也就是一組 0 個或多個資料行，您可以以此將查詢的結果集分組。GROUP BY GROUPING SETS 等同於在由不同資料欄分組的一個結果集上執行 UNION ALL 查詢。例如，GROUP BY GROUPING SETS((a), (b)) 等同於 GROUP BY a UNION ALL GROUP BY b。

下列範例會傳回訂單資料表的產品根據產品類別和銷售產品種類進行分組的成本。

```
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

假設有一個階層，其中之前的資料欄被視為後續資料欄的父項。ROLLUP 會依提供的資料欄將資料分組，並傳回額外的小計資料列 (代表所有分組資料欄層級的總計)，以及已分組的資料列。例如，您可以使用 GROUP BY ROLLUP((a), (b)) 傳回一個先按 a 分組，然後按 b 分組的結果集 (假設 b 是 a 的子區段)。ROLLUP 也會傳回包含整個結果集的資料列，而不會將資料欄分組。

GROUP BY ROLLUP((a), (b)) 等於 GROUP BY GROUPING SETS((a,b), (a), ())。

下列範例會傳回訂單資料表產品先依類別分組，然後依產品分組的成本，其中以產品做為類別的細項。

```
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

依提供的資料欄將資料分組，並傳回額外的小計資料列 (代表所有分組資料欄層級的總計)，以及已分組的資料列。CUBE 會傳回與 ROLLUP 相同的資料列，同時針對 ROLUP 未涵蓋的每個分組資料欄組合

新增額外的小計資料列。例如，您可以使用 GROUP BY CUBE ((a), (b)) 傳回一個先按 a 分組，然後按 b 分組的結果集 (假設 b 是 a 的子區段)，然後再獨自按 b 分組。CUBE 也會傳回包含整個結果集的資料列，而不會將資料欄分組。

GROUP BY CUBE((a), (b)) 等於 GROUP BY GROUPING SETS((a, b), (a), (b), ())。

下列範例會傳回訂單資料表產品先依類別分組，然後依產品分組的成本，其中以產品做為類別的細項。與前面的 ROLLUP 範例不同，陳述式會傳回每個分組資料欄組合的結果。

```
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)
```

HAVING 子句

HAVING 子句會將條件套用至查詢傳回的中繼分組結果集。

語法

```
[ HAVING condition ]
```

例如，您可以限制 SUM 函數的結果：

```
having sum(pricepaid) >10000
```

HAVING 會在套用所有 WHERE 子句條件且完成 GROUP BY 操作之後套用。

條件本身會採用與任何 WHERE 子句條件相同的形式。

使用須知

- HAVING 子句條件中參考的任何資料欄必須是分組資料欄，或是參考彙整函數結果的資料欄。
- 在 HAVING 子句中，您無法指定：
  - 參考選取清單項目的序數。只有 GROUP BY 和 ORDER BY 子句接受序數。

範例

以下查詢會依名稱計算售票總金額，然後消除總金額低於 800,000 USD 的活動。HAVING 條件會套用至選取清單中彙整函數的結果：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
(6 rows)
```

以下查詢會計算類似的結果集。不過，在此情況下，HAVING 條件會套用至選取清單中未指定的彙整：sum(qtysold)。未銷售超過 2,000 張票的活動將從最終結果中消除。

```
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
(8 rows)
```

設定運算子

UNION、INTERSECT 和 EXCEPT 集合運算子可用來比較和合併兩種不同查詢表達式的結果。例如，如果您想知道哪些網站使用者同時是買方和賣家，但其使用者名稱儲存在不同的資料欄或資料表中，您可以找出這兩種類型使用者的交集。如果您想知道哪些網站使用者是買方，但不是賣家，您可以使用 EXCEPT 運算子找出兩份使用者清單之間的差異。如果您想要建構所有使用者的清單，但不考慮角色，您可以使用 UNION 運算子。

> **Note**
>
> 順序 BY，限制，選擇頂部和偏移子句不能在由聯集，聯集所有，交集和除外集合運算符合併的查詢表達式中使用。

主題

語法

```
query
{ UNION [ ALL ] | INTERSECT | EXCEPT | MINUS }
query
```

參數

query

此查詢表達式會以其選取清單形式，對應至接在 UNION、INTERSECT 或 EXCEPT 運算子後面的另一個查詢表達式。兩個表達式必須包含採用相容資料類型的相同輸出資料欄數，否則就無法比較和合併這兩個結果集。集合操作不允許在不同類別的資料類型之間進行隱含轉換；如需詳細資訊，請參閱 類型相容性與轉換。

您可以建構包含無限查詢表達式數目的查詢，並將它們與 UNION、INTERSECT 和 EXCEPT 運算子的任意組合連結。例如，假設資料表 T1、T2 和 T3 包含相容的資料欄集，則以下查詢結構有效：

```
select * from t1
union
select * from t2
except
select * from t3
```

UNION

此集合操作會從兩個查詢表達式傳回資料列，無論資料列衍生自其中一個或兩個表達式。

INTERSECT

此集合操作會傳回衍生自兩個查詢表達式的資料列。未由兩個表達式傳回的資料列則會遭到捨棄。

EXCEPT | MINUS

此集合操作會傳回衍生自兩個查詢表達式之一的資料列。若要限定結果，資料列必須存在第一個結果資料表中，但不能存在第二個資料表中。MINUS 和 EXCEPT 是一模一樣的同義詞。

ALL

ALL 關鍵字會保留 UNION 所產生的任何重複資料列。未使用 ALL 關鍵字時的預設行為是捨棄這些重複項目。不支援 INTERSECT ALL、EXCEPT ALL 和 MINUS ALL。

集合運算子的評估順序

UNION 和 EXCEPT 集合運算子為左關聯。若未指定括號來影響優先順序，則會從左到右評估這些集合運算子的組合。例如，在下列查詢中，T1 和 T2 的 UNION 會先評估，然後在 UNION 結果上執行 EXCEPT 操作：

```
select * from t1
union
select * from t2
except
select * from t3
```

在相同查詢中使用運算子組合時，INTERSECT 運算子的優先順序高於 UNION 和 EXCEPT 運算子。例如，下列查詢會先評估 T2 和 T3 的交集，再將結果與 T1 進行聯集：

```
select * from t1
union
select * from t2
intersect
select * from t3
```

加入括號就可以強制執行不同的評估順序。在下列案例中，T1 和 T2 的聯集結果會與 T3 交集，而查詢可能會產生不同的結果。

```
(select * from t1
union
select * from t2)
intersect
(select * from t3)
```

使用須知

- 集合操作查詢的結果中傳回的資料欄名稱，是來自第一個查詢表達式的資料表中的資料欄名稱 (或別名)。這些資料欄名稱可能會造成誤導，因為資料欄中的值是從任一邊集合運算子的資料表衍生，所以建議您為結果集提供有意義的別名。
- 當集合運算子查詢傳回小數結果時，對應的結果資料欄就會提升，以傳回相同的精確度和小數位數。例如，在以下查詢中，T1.REVENUE 是 DECIMAL(10,2) 資料欄，而 T2.REVENUE 是 DECIMAL(8,4) 資料欄，小數結果會提升為 DECIMAL(12,4)：

  ```
  select t1.revenue union select t2.revenue;
  ```

  小數位數為 4，因為這是兩個資料欄的小數位數上限。精確度為 12，因為 T1.REVENUE 要求小數點左邊有 8 位數 (12 - 4 = 8)。此類型提升可確保 UNION 兩邊的所有值都能納入結果中。若是 64 位元值，最高結果精確度為 19，而結果小數位數上限為 18。若是 128 位元值，最高結果精確度為 38，而結果小數位數上限為 37。

如果產生的資料類型超過有效位 AWS Clean Rooms 數和比例限制，則查詢會傳回錯誤。

- 在集合操作中，若每個對應資料欄配對的這兩個資料值為等於或兩者皆為 NULL，則這兩個資料列會視為相同。例如，若資料表 T1 和 T2 都包含一個資料欄和一個資料列，而該資料列在兩個資料表中都是 NULL，則對這些資料表執行 INTERSECT 操作就會傳回該資料列。

範例 UNION 查詢

在下列 UNION 查詢中，SALES 資料表中的資料列會與 LISTING 資料表中的資料列合併。會從每個資料表選取三個相容的資料欄；在此情況下，對應的資料欄會有相同的名稱和資料類型。

```
select listid, sellerid, eventid from listing
union select listid, sellerid, eventid from sales


listid | sellerid | eventid
--------+----------+---------
1 |    36861 |    7872
2 |    16002 |    4806
3 |    21461 |    4256
4 |     8117 |    4337
5 |     1616 |    8647
```

以下範例說明如何將常值新增至 UNION 查詢的輸出，以便查看結果集中的每個資料列是由哪個查詢表達式所產生。查詢會將來自第一個查詢表達式的資料列識別為 "B" (表示買方)，來自第二個查詢表達式的資料列識別為 "S" (表示賣家)。

查詢會識別價值 10,000 USD 以上的票券交易買方和賣家。UNION 運算子任一邊的兩個查詢表達式之間唯一的差異，就是 SALES 資料表的聯結資料欄。

```
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

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
```

以下範例使用 UNION ALL 運算子，因為重複的資料列 (若找到) 需保留在結果中。若是特定活動 ID 系列，查詢會針對與各個活動相關聯的每筆銷售傳回 0 或更多資料列，並針對該活動的每份清單傳回 0 或 1。LISTING 和 EVENT 資料表中每個資料列的活動 ID 都是唯一的，但 SALES 資料表中相同的活動和清單 ID 組合可能會有多筆銷售。

結果集中的第三個資料欄會識別資料列的來源。若是來自 SALES 資料表，則會在 SALESROW 資料欄中標示為「Yes (是)」 (SALESROW 是 SALES.LISTID 的別名)。若資料列來自 LISTING 資料表，則會在 SALESROW 資料欄中標示為「No (否)」。

在此情況下，結果集會包含活動 7787、清單 500 的三個銷售資料列。換句話說，此清單與活動組合發生了三筆不同的交易。另外兩份清單 501 和 502 並未產生任何銷售，因此查詢針對這些清單 ID 產生的唯一資料列是來自 LISTING 資料表 (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)

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
```

若您執行相同查詢，但未使用 ALL 關鍵字，結果只會保留其中一項銷售交易。

```
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)

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
```

範例 UNION ALL 查詢

以下範例使用 UNION ALL 運算子，因為重複的資料列 (若找到) 需保留在結果中。若是特定活動 ID 系列，查詢會針對與各個活動相關聯的每筆銷售傳回 0 或更多資料列，並針對該活動的每份清單傳回 0 或 1。LISTING 和 EVENT 資料表中每個資料列的活動 ID 都是唯一的，但 SALES 資料表中相同的活動和清單 ID 組合可能會有多筆銷售。

結果集中的第三個資料欄會識別資料列的來源。若是來自 SALES 資料表，則會在 SALESROW 資料欄中標示為「Yes (是)」(SALESROW 是 SALES.LISTID 的別名)。若資料列來自 LISTING 資料表，則會在 SALESROW 資料欄中標示為「No (否)」。

在此情況下，結果集會包含活動 7787、清單 500 的三個銷售資料列。換句話說，此清單與活動組合發生了三筆不同的交易。另外兩份清單 501 和 502 並未產生任何銷售，因此查詢針對這些清單 ID 產生的唯一資料列是來自 LISTING 資料表 (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)

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
```

```
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
```

若您執行相同查詢，但未使用 ALL 關鍵字，結果只會保留其中一項銷售交易。

```
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)
eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
```

範例 INTERSECT 查詢

比較下列範例與第一個 UNION 範例。這兩個範例的唯一差異在於使用的集合運算子，但結果非常不同。只有其中一個資料列相同：

```
235494 |    23875 |    8771
```

這是在限制的 5 個資料列結果中，唯一同時存在兩個資料表中的資料列。

```
select listid, sellerid, eventid from listing
intersect
select listid, sellerid, eventid from sales

listid | sellerid | eventid
--------+----------+---------
235494 |    23875 |    8771
235482 |     1067 |    2667
235479 |     1589 |    7303
235476 |    15550 |     793
```

```
235475 |    22306 |    7848
```

以下查詢會尋找三月份同時於紐約市和洛杉磯的場館舉行的活動 (有售票)。兩個查詢表達式之間唯一的差異就是 VENUECITY 資料欄的限制條件。

```
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';

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
```

範例 EXCEPT 查詢

在數據庫中的類別表包含以下 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)
```

假設 CATEGORY_STAGE 資料表 (臨時資料表) 包含一個額外的資料列：

```
 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)
```

傳回兩個資料表之間的差異。換句話說，會傳回 CATEGORY_STAGE 資料表而非 CATEGORY 資料表中的資料列：

```
select * from category_stage
except
select * from category;

catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
  12  | Concerts | Comedy  | All stand up comedy performances
(1 row)
```

以下對等查詢使用同義詞 MINUS。

```
select * from category_stage
```

```
minus
select * from category;

catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
  12  | Concerts | Comedy  | All stand up comedy performances
(1 row)
```

若您將 SELECT 表達式的順序反轉，則查詢不會傳回任何資料列。

ORDER BY 子句

ORDER BY 子句會排序查詢的結果集。

> **Note**
>
> 最外層的 ORDER BY 運算式必須只有位於選取清單中的欄。

主題

- 語法
- 參數
- 使用須知
- ORDER BY 的範例

語法

```
[ ORDER BY expression [ ASC | DESC ] ]
[ NULLS FIRST | NULLS LAST ]
[ LIMIT { count | ALL } ]
[ OFFSET start ]
```

參數

運算式

定義查詢結果排序順序的運算式。它由選擇列表中的一個或多個列組成。結果會根據二進位 UTF-8 順序傳回。您還可以指定下列項目：

- 代表選取清單項目位置的序數 (或是，若沒有選取清單的話，則為資料欄在資料表中的位置)
- 定義選取清單項目的別名

當 ORDER BY 子句包含多個表達式時，結果集會根據第一個表達式排序，然後對擁有與第一個表達式相符之值的資料列套用第二個表達式，以此類推。

ASC | DESC

此選項會定義表達式的排序順序，如下所示：

- ASC：遞增 (例如，數值從低到高，字元字串 'A' 到 'Z')。若未指定選項，資料會預設為遞增排序。
- DESC：遞減 (數值從高到低，字串 'Z' 到 'A')。

NULLS FIRST | NULLS LAST

這些選項指定 NULL 值應該排序在最前 (在非 null 值之前) 或排序在最後 (在非 null 值之後)。根據預設，依 ASC 順序排序時，NULL 值排在最後面，而依 DESC 順序排序時，則排在最前面。

LIMIT number | ALL

此選項會控制查詢傳回的排序資料列數。LIMIT 數字必須是正整數；最大值為 2147483647。

LIMIT 0 不會傳回任何資料列。您可以使用此語法進行測試：查看查詢執行情形 (不顯示任何資料列)，或從資料表傳回資料欄清單。若您使用 LIMIT 0 傳回資料欄清單，則 ORDER BY 子句是多餘的。預設值為 LIMIT ALL。

OFFSET start

此選項會指定先略過 start 之前的資料列數，再開始傳回資料列。OFFSET 數字必須是正整數；最大值為 2147483647。搭配 LIMIT 選項使用時，會先略過 OFFSET 資料列，再開始計算傳回的 LIMIT 資料列。如果未使用 LIMIT 選項，則結果集中的資料列數會減掉略過的資料列數。OFFSET 子句略過的資料列仍須經過掃描，因此使用較大的 OFFSET 值可能會導致效率不佳。

使用須知

請注意以下使用 ORDER BY 子句的預期行為：

- NULL 值會視為「高於」所有其他值。使用預設的遞增排序順序時，NULL 值會排列在最後面。若要變更此行為，請使用 NULLS FIRST 選項。

- 若查詢未包含 ORDER BY 子句，系統傳回的結果集當中就不會有可預測的資料列排列順序。執行相同的查詢兩次，可能會傳回依不同順序排列的結果集。
- LIMIT 和 OFFSET 選項可在沒有 ORDER BY 子句的情況下使用；不過，若要傳回一致的資料列集，請使用這些選項搭配 ORDER BY。
- 在任何 parallel 系統中 AWS Clean Rooms，當 ORDER BY 不產生唯一的排序時，行的順序是不確定的。也就是說，如果 ORDER BY 運算式產生重複的值，這些資料列的傳回順序可能會因其他系統而異，或從一個執行 AWS Clean Rooms 到下一個執行。
- AWS Clean Rooms ORDER BY 子句中不支持字符串文字。

ORDER BY 的範例

從 CATEGORY 資料表傳回全部 11 列，並依第二個資料欄 CATGROUP 排序。若結果擁有相同的 CATGROUP 值，則依字元字串的長度排列 CATDESC 資料欄的值。然後，依 CATID 和 CATNAME 欄排序。

```
select * from category order by 2, 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)
```

從 SALES 資料表傳回選取的欄，並依最高 QTYSOLD 值排序。將結果限制為前 10 個資料列：

```
select salesid, qtysold, pricepaid, commission, saletime from sales
order by qtysold, pricepaid, commission, salesid, saletime desc

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
```

使用 LIMIT 0 語法會傳回一份資料欄清單，但沒有資料列：

```
select * from venue limit 0;
venueid | venuename | venuecity | venuestate | venueseats
---------+-----------+-----------+------------+------------
(0 rows)
```

子查詢範例

下列範例顯示將子查詢納入 SELECT 查詢的不同方式。請參閱 JOIN 範例，了解另一個使用子查詢的範例。

SELECT 清單子查詢

以下範例包含 SELECT 清單中的子查詢。此子查詢為純量：它只會傳回一個資料欄和一個值，該值會在從外部查詢傳回的每個資料列的結果中重複出現。查詢會比較子查詢運算的 Q1SALES 值與 2008 年另兩季 (2 和 3) 的銷售數字，如外部查詢所定義。

```
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 子句子查詢

以下範例包含 WHERE 子句中的資料表子查詢。此子查詢會產生多個資料列。在此情況下，資料列只會包含一個資料欄，但資料表子查詢可包含多個資料欄和資料列，就像任何其他資料表一樣。

查詢會尋找票券銷售量最高的前 10 名賣家。前 10 名清單受到子查詢的限制，會移除居住在有售票場地之城市的使用者。此查詢可透過不同的方式撰寫；例如，子查詢可重寫為主查詢內的聯結。

```
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 子句子查詢

請參閱 WITH 子句。

相互關聯子查詢

以下範例包含 WHERE 子句中的相互關聯子查詢；這類子查詢的資料欄與外部查詢產生的資料欄之間包含一項或多項相互關聯。在此情況下，相互關聯為 `where s.listid=l.listid`。針對外部查詢產生的每個資料列，子查詢會執行以限定或取消限定資料列。

```
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)
```

不支援的相互關聯子查詢模式

查詢規劃器會使用一種查詢重寫方法，稱為子查詢解除相互關聯，在 MPP 環境中最佳化數種相互關聯子查詢模式以供執行。幾種類型的相關子查詢遵循 AWS Clean Rooms 無法裝飾關聯且不支持的模式。包含下列相互關聯參考的查詢會傳回錯誤：

- 略過查詢區塊的相互關聯參考，也稱為「略過層級相互關聯參考」。例如，在下列查詢中，包含相互關聯參考的區塊和略過的區塊會以 NOT EXISTS 述詞連接：

  ```
  select event.eventname from event
  where not exists
  (select * from listing
  where not exists
  (select * from sales where event.eventid=sales.eventid));
  ```

  在此案例中略過的區塊是對 LISTING 資料表的子查詢。相互關聯參考會將 EVENT 和 SALES 資料表相互關聯。
- 來自子查詢的相互關聯參考，它是外部查詢中 ON 子句的一部分：

  ```
  select * from category
  left join event
  on category.catid=event.catid and eventid =
  (select max(eventid) from sales where sales.eventid=event.eventid);
  ```

  ON 子句包含從子查詢中 SALES 對外部查詢中 EVENT 的相互關聯參考。

- 系統 AWS Clean Rooms 資料表的空值敏感關聯參照。例如：

```
select attrelid
from my_locks sl, my_attribute
where sl.table_id=my_attribute.attrelid and 1 not in
(select 1 from my_opclass where sl.lock_owner = opcowner);
```

- 來自子查詢內的相互關聯參考，當中包含視窗函數。

```
select listid, qtysold
from sales s
where qtysold not in
(select sum(numtickets) over() from listing l where s.listid=l.listid);
```

- GROUP BY 資料欄中對相互關聯子查詢結果的參考。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;
```

- 子查詢中使用彙整函數和 GROUP BY 子句的相互關聯參考，會透過 IN 述詞連接至外部查詢。(此限制不會套用至 MIN 和 MAX 彙整函數)。例如：

```
select * from listing where listid in
(select sum(qtysold)
from sales
where numtickets>4
group by salesid);
```

中的 SQL 函數 AWS Clean Rooms

AWS Clean Rooms 支援下列 SQL 函數：

主題

彙總函數

AWS Clean Rooms 支援下列彙總函式：

主題

- MEDIAN 函數
- MIN 函數
- PERCENTILE_CONT 函數
- STDDEV_SAMP 和 STDDEV_POP 函數
- SUM和SUM DISTINCT功能
- VAR_SAMP 和 VAR_POP 函數

ANY_VALUE 函數

ANY_VALUE 函數從輸入運算式值非確定性傳回任何值。如果輸入運算式不會傳回任何資料列，此函數可以傳回 NULL。

語法

```
ANY_VALUE ( [ DISTINCT | ALL ] expression )
```

引數

DISTINCT | ALL

指定 DISTINCT 或 ALL 可從輸入運算式值傳回任何值。DISTINCT 引數沒有任何作用，而且會被忽略。

expression

函數運算的目標欄或運算式。expression 是下列其中一種資料類型：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISION
- BOOLEAN
- CHAR
- VARCHAR

- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

傳回值

傳回與 expression 相同的資料類型。

使用須知

如果指定欄 ANY_VALUE 函數的陳述式也包含第二個欄參考，則第二個欄必須出現在 GROUP BY 子句中，或包含在彙總函數中。

範例

下列範例會傳回 eventname is 任何dateid位置的執行個體Eagles。

```
select any_value(dateid) as dateid, eventname from event where eventname ='Eagles'
 group by eventname;
```

以下是結果。

```
dateid | eventname
-------+---------------
 1878  | Eagles
```

下列範例會傳回 is 或任何dateid位置的執行eventname個Eagles體Cold War Kids。

```
select any_value(dateid) as dateid, eventname from event where eventname in('Eagles',
 'Cold War Kids') group by eventname;
```

以下是結果。

```
dateid | eventname
```

```
-------+---------------
 1922  | Cold War Kids
 1878  | Eagles
```

APPROXIMATE PERCENTILE_DISC 函數

APPROXIMATE PERCENTILE_DISC 是採用離散分佈模型的反向分佈函數。它採用百分位數值和排序規格，且會傳回給定集裡的一個元素。近似法可讓函數執行較快，其相對錯誤率低到約 0.5%。

對於給定的百分位數值，APPROXIMATE PERCENTILE_DISC 使用分位數摘要演算法，大致估計 ORDER BY 子句中的表達式的離散百分位數。APPROXIMATE PERCENTILE_DISC 傳回的值具有大於或等於百分位數的最小累積分佈值 (根據相同的排序規格)。

APPROXIMATE PERCENTILE_DISC 是僅限於運算節點的函數。如果查詢未參考使用者定義的資料表或 AWS Clean Rooms 系統資料表，函式會傳回錯誤。

語法

```
APPROXIMATE  PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
```

引數

percentile

介於 0 和 1 之間的數值常數。計算時會忽略 Null。

WITHIN GROUP (ORDER BY expr)

此子句指定要排序和計算百分位數的數值或日期/時間值。

傳回值

資料類型與 WITHIN GROUP 子句中的 ORDER BY 表達式相同。

使用須知

如果 APPROXIMATE PERCENTILE_DISC 陳述式包含 GROUP BY 子句，則會限制結果集。限制會根據節點類型和節點數目而不同。如果超出限制，函數會失敗並傳回下列錯誤。

```
GROUP BY limit for approximate percentile_disc exceeded.
```

如果您需要評估的組數超過限制所允許，請考慮使用 PERCENTILE_CONT 函數。

範例

下列範例傳回排名前 10 個日期的銷售數量、銷售總計及第五個百分位數值。

```
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 函數

該AVG函數返回輸入表達式值的平均值（算術平均值）。該AVG函數與數值一起使用，並忽略 NULL 值。

語法

```
AVG (column)
```

引數

##

該函數運行的目標列。資料行為下列其中一種資料類型：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- DOUBLE

資料類型

AVG函數支援的引數類型為SMALLINTINTEGER、BIGINT、DECIMAL、和DOUBLE。

該AVG函數支持的返回類型如下：

- BIGINT對於任何整數類型參數
- DOUBLE對於浮點引數
- 返回與任何其他參數類型的表達式相同的數據類型

帶有引數的AVG函數結果的默認有效位DECIMAL數為 38。結果的小數位數和引數的小數位數相同。例如，一個AVGDEC(5,2)列返回一個DEC(38,2)數據類型。

範例

從SALES表格中找出每筆交易的平均售出數量。

```
select avg(qtysold)from sales;
```

BOOL_AND 函數

BOOL_AND 函數會對單一布林值或整數欄或表達式執行操作。此函數會將類似邏輯套用至 BIT_AND 和 BIT_OR 函數。此函數的傳回類型為布林值 (`true` 或 `false`)。

如果一組值全部為 true，BOOL_AND 函數會傳回 `true` (`t`)。如果任何值為 false，此函數會傳回 `false` (`f`)。

語法

```
BOOL_AND ( [DISTINCT | ALL] expression )
```

引數

expression

函數運算的目標欄或表達式。此表達式必須為 BOOLEAN 或整數資料類型。函數的傳回類型為 BOOLEAN。

DISTINCT | ALL

如果指定引數 DISTINCT，則函數在計算結果之前，將消除指定之表達式的所有重複值。如果指定引數 ALL，則函數會保留所有重複值。ALL 為預設值。

範例

您可以對布林值表達式或整數表達式使用布林值函數。

例如，下列查詢從 TICKET 資料庫中的標準 USERS 資料表 (其中有幾個布林值欄) 傳回結果。

BOOL_AND 函數在全部五列中傳回 `false`。其中每個州並非所有使用者都喜歡運動。

```
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 函數

BOOL_OR 函數會對單一布林值或整數欄或表達式執行操作。此函數會將類似邏輯套用至 BIT_AND 和 BIT_OR 函數。此函數的傳回類型為布林值 (`true`、`false` 或 NULL)。

如果一組值之中的值為 `true`，BOOL_OR 函數會傳回 `true` (t)。如果集合中的值是 `false`，此函數會傳回 `false` (f)。如果該值未知，則可以傳回 NULL。

語法

```
BOOL_OR ( [DISTINCT | ALL] expression )
```

引數

expression

函數運算的目標欄或表達式。此表達式必須為 BOOLEAN 或整數資料類型。函數的傳回類型為 BOOLEAN。

DISTINCT | ALL

如果指定引數 DISTINCT，則函數在計算結果之前，將消除指定之表達式的所有重複值。如果指定引數 ALL，則函數會保留所有重複值。ALL 為預設值。

範例

您可以對布林值運算式或整數運算式使用布林值函數。例如，下列查詢從 TICKET 資料庫中的標準 USERS 資料表 (其中有幾個布林值欄) 傳回結果。

BOOL_OR 函數在全部五列中傳回 `true`。其中每個州至少有一個使用者喜歡運動。

```
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)
```

以下範例傳回 NULL。

```
SELECT BOOL_OR(NULL = '123')
               bool_or
------
```

```
NULL
```

COUNT和COUNT DISTINCT功能

該COUNT函數計算由表達式定義的行。此COUNT DISTINCT函數會計算資料行或運算式中不同非 NULL 值的數目。在執行計數之前，它會消除指定表達式中的所有重複值。

語法

```
COUNT (column)
```

```
COUNT (DISTINCT column)
```

引數

##

該函數運行的目標列。

資料類型

該COUNT函數和函數COUNT DISTINCT支持所有參數數據類型。

該COUNT DISTINCT函數返回BIGINT。

範例

統計佛羅里達州的所有用戶。

```
select count (identifier) from users where state='FL';
```

從表中計算所有唯一的場地 EVENT ID。

```
select count (distinct (venueid)) as venues from event;
```

COUNT 函數

COUNT 函數計算表達式所定義的列數。

COUNT 函數有下列版本。

- COUNT (*) 計算目標資料表中的所有列數，而不論是否包含 Null。
- COUNT (expression) 計算特定欄或表達式中不含 NULL 值的列數。
- COUNT (DISTINCT expression) 計算某欄或表達式中相異非 NULL 值的個數。
- APPROXIMATE COUNT DISTINCT 會大致估計某欄或運算式中相異非 NULL 值的個數。

語法

```
COUNT( * | expression )
```

```
COUNT ( [ DISTINCT | ALL ] expression )
```

```
APPROXIMATE COUNT ( DISTINCT expression )
```

引數

expression

函數運算的目標欄或表達式。COUNT 函數支援所有引數資料類型。

DISTINCT | ALL

如果指定引數 DISTINCT，則函數在計數之前會從指定的表達式中消除所有重複值。如果指定引數 ALL，則函數在計數時會保留表達式中的所有重複值。ALL 為預設值。

APPROXIMATE

當與近似使用時，COUNT DISTINCT 函數會使用 HyperLogLog 演算法來近似資料行或運算式中不同非空值的數目。使用 APPROXIMATE 關鍵字的查詢執行較快，其相對錯誤率低到約 2%。如果每個查詢或每一組 (若有 group by 子句) 傳回數百萬個以上的大量相異值，則查詢一定要採用近似法。如果相異值較少 (數千個)，則近似法可能比精確計數更慢。APPROXIMATE 只能與 COUNT DISTINCT 一起使用。

傳回類型

COUNT 函數傳回 BIGINT。

範例

計算佛羅里達州的所有使用者人數：

```
select count(*) from users where state='FL';

count
-------
510
```

計算 EVENT 表中的所有事件名稱：

```
select count(eventname) from event;

count
-------
8798
```

計算 EVENT 表中的所有事件名稱：

```
select count(all eventname) from event;

count
-------
8798
```

從 EVENT 資料表計算所有唯一會場 ID 的數目：

```
select count(distinct venueid) as venues from event;

venues
--------
204
```

計算每個賣方列出整批銷售門票超過四張的次數。結果依賣方 ID 分組：

```
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
...
```

下列範例比較 COUNT 和 APPROXIMATE COUNT 的傳回值和執行時間。

```
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 函數

對於查詢中的每一組，LISTAGG 彙整函數依據 ORDER BY 表達式來排序該組的列，然後將這些值串連成單一字串。

LISTAGG 是僅限於運算節點的函數。如果查詢未參考使用者定義的資料表或 AWS Clean Rooms 系統資料表，函式會傳回錯誤。

語法

```
LISTAGG( [DISTINCT] aggregate_expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
```

引數

DISTINCT

(選用) 此子句在串連值之前會從指定的表達式中消除重複值。結尾空格會忽略，所以字串 'a' 和 'a ' 視為重複值。LISTAGG 會使用第一個遇到的值。如需詳細資訊，請參閱 多餘空格的意義。

aggregate_expression

任何有效表達式 (例如欄名)，用於提供要彙總的值。忽略 NULL 值和空字串。

delimiter

(選用) 用來區隔串連值的字串常數。預設值為 NULL。

AWS Clean Rooms 支援可選逗號或冒號周圍任何數量的前導或尾隨空格，以及空字串或任意數量的空格。

有效值的範例如下：

', '

': '

' '

WITHIN GROUP (ORDER BY order_list)

(選用) 此子句指定彙總值的排序順序。

傳回值

VARCHAR(MAX)。如果結果集大於 VARCHAR 大小上限 (64K – 1，或 65535)，則 LISTAGG 會傳回下列錯誤：

```
Invalid operation: Result size exceeds LISTAGG limit
```

使用須知

如果陳述式包含多個使用 WITHIN GROUP 子句的 LISTAGG 函數，則每一個 WITHIN GROUP 子句必須使用相同的 ORDER BY 值。

例如，下列陳述式會傳回錯誤。

```
select listagg(sellerid)
within group (order by dateid) as sellers,
listagg(dateid)
within group (order by sellerid) as dates
from winsales;
```

下列陳述式將會成功執行。

```
select listagg(sellerid)
within group (order by dateid) as sellers,
listagg(dateid)
within group (order by dateid) as dates
from winsales;

select listagg(sellerid)
within group (order by dateid) as sellers,
listagg(dateid) as dates
from winsales;
```

範例

下列範例彙總賣方 ID，依賣方 ID 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
```

下列範例使用 DISTINCT 傳回唯一賣方 ID 的清單。

```
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
```

下列範例彙總賣方 ID，依日期順序排序。

```
select listagg(sellerid)
within group (order by dateid)
from winsales;

   listagg
-------------
 31141242333
```

下列範例以縱線分隔清單傳回買方 B 的銷售日期。

```
select listagg(dateid,'|')
within group (order by sellerid desc,salesid asc)
from winsales
where buyerid  = 'b';

             listagg
-------------------------------------
2003-08-02|2004-04-18|2004-04-18|2004-02-12
```

下列範例以逗號分隔清單傳回每個買方 ID 的銷售 ID。

```
select buyerid,
listagg(salesid,',')
within group (order by salesid) as sales_id
from winsales
group by buyerid
order by buyerid;

   buyerid | sales_id
-----------+------------------------
        a  |10005,40001,40005
        b  |20001,30001,30004,30003
        c  |10001,20002,30007,10006
```

MAX 函數

MAX 函數傳回一個列集的最大值。可使用 DISTINCT 或 ALL，但不影響結果。

語法

```
MAX ( [ DISTINCT | ALL ] expression )
```

引數

expression

函數運算的目標欄或表達式。expression 是下列其中一種資料類型：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISION
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

如果指定引數 DISTINCT，則函數在計算最大值之前，將從指定的表達式中消除所有重複值。如果指定引數 ALL，則函數在計算最大值時會保留表達式中的所有重複值。ALL 為預設值。

資料類型

傳回與 expression 相同的資料類型。

範例

從所有銷售中尋找最高支付價格：

```
select max(pricepaid) from sales;

max
----------
12624.00
(1 row)
```

從所有銷售中尋找每張門票的最高支付價格：

```
select max(pricepaid/qtysold) as max_ticket_price
from sales;

max_ticket_price
-----------------
2500.00000000
(1 row)
```

MEDIAN 函數

計算值範圍的中位數值。忽略範圍中的 NULL 值。

MEDIAN 是採用連續分佈模型的反向分佈函數。

MEDIAN 是 PERCENTILE_CONT(.5) 的特殊情況。

MEDIAN 是僅限於運算節點的函數。如果查詢未參考使用者定義的資料表或 AWS Clean Rooms 系統資料表，函式會傳回錯誤。

語法

```
MEDIAN ( median_expression )
```

引數

median_expression

函數運算的目標欄或表達式。

資料類型

傳回類型取決於 median_expression 的資料類型。下表顯示每一個 median_expression 資料類型的傳回類型。

輸入類型	傳回類型
數字，十進制	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

使用須知

如果 median_expression 引數是以最大精確度 38 位數定義的 DECIMAL 資料類型，MEDIAN 可能會傳回不準確的結果或錯誤。如果 MEDIAN 函數的傳回值超過 38 位數，會將結果截斷為適合長度，導致精確度降低。在插補期間，如果中間結果超過最大精確度，則會發生數值溢位，且函數會傳回錯誤。為了避免這些情況，建議使用精確度較低的資料類型，或將 median_expression 引數轉換為較低精確度。

如果陳述式中多次呼叫會排序的彙總函數 (LISTAGG、PERCENTILE_CONT 或 MEDIAN)，則所有呼叫必須使用相同的 ORDER BY 值。請注意，MEDIAN 會對表達式值套用隱含的 order by。

例如，下列陳述式會傳回錯誤。

```
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, pricepai...
```

```
ERROR: within group ORDER BY clauses for aggregate functions must be the same
```

下列陳述式會成功執行。

```
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (salesid)
from sales group by salesid, pricepaid;
```

範例

下列範例顯示 MEDIAN 產生與 PERCENTILE_CONT(0.5) 相同的結果。

```
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
---------+---------+-----------------+-------
       1 |       1 |             1.0 |    1.0
       2 |       3 |             3.0 |    3.0
       5 |       2 |             2.0 |    2.0
       9 |       4 |             4.0 |    4.0
      12 |       1 |             1.0 |    1.0
      16 |       1 |             1.0 |    1.0
      19 |       2 |             2.0 |    2.0
      19 |       3 |             3.0 |    3.0
      22 |       2 |             2.0 |    2.0
      25 |       2 |             2.0 |    2.0
```

MIN 函數

MIN 函數傳回一個列集的最小值。可使用 DISTINCT 或 ALL，但不影響結果。

語法

```
MIN ( [ DISTINCT | ALL ] expression )
```

引數

expression

函數運算的目標欄或表達式。expression 是下列其中一種資料類型：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISION
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

如果指定引數 DISTINCT，則函數在計算最小值之前，將從指定的表達式中消除所有重複值。如果指定引數 ALL，則函數在計算最小值時會保留表達式中的所有重複值。ALL 為預設值。

資料類型

傳回與 expression 相同的資料類型。

範例

從所有銷售中尋找最低支付價格：

```
select min(pricepaid) from sales;
```

```
min
-------
20.00
(1 row)
```

從所有銷售中尋找每張門票的最低支付價格：

```
select min(pricepaid/qtysold)as min_ticket_price
from sales;

min_ticket_price
------------------
20.00000000
(1 row)
```

PERCENTILE_CONT 函數

PERCENTILE_CONT 是採用連續分佈模型的反向分佈函數。它採用百分位數值和排序規格，且會傳回插入值，該值將根據排序規格落入給定的百分位數值。

PERCENTILE_CONT 在值排序後計算值之間的線性插值。此函數在列根據排序規格來排序後，使用彙總群組中的百分位數值 (P) 和非 Null 列數 (N) 來計算列號。此列號 (RN) 是根據公式 RN = (1+(P*(N-1)) 來計算。彙總函數的最終結果是以列號 CRN = CEILING(RN) 到 FRN = FLOOR(RN) 各列的值之間的線性插值來計算。

最終結果如下。

如果 (CRN = FRN = RN)，則結果為 (value of expression from row at RN)

否則結果如下：

(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN).

PERCENTILE_CONT 是僅限於運算節點的函數。如果查詢未參考使用者定義的資料表或 AWS Clean Rooms 系統資料表，函式會傳回錯誤。

語法

```
PERCENTILE_CONT ( percentile )
```

```
WITHIN GROUP (ORDER BY expr)
```

引數

percentile

介於 0 和 1 之間的數值常數。計算時會忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定要排序和計算百分位數的數值或日期/時間值。

傳回值

傳回類型取決於 WITHIN GROUP 子句中 ORDER BY 表達式的資料類型。下表顯示每一個 ORDER BY 表達式資料類型的傳回類型。

輸入類型	傳回類型
小型, 整數, 大整數, 數字, 十進制	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

使用須知

如果 ORDER BY 表達式是以最大精確度 38 位數定義的 DECIMAL 資料類型，PERCENTILE_CONT 可能會傳回不準確的結果或錯誤。如果 PERCENTILE_CONT 函數的傳回值超過 38 位數，結果會截斷為適合長度，導致精確度降低。在插補期間，如果中間結果超過最大精確度，則會發生數值溢位，且函數會傳回錯誤。為了避免這些情況，建議使用精確度較低的資料類型，或將 ORDER BY 表達式轉換為較低精確度。

如果陳述式中多次呼叫會排序的彙總函數 (LISTAGG、PERCENTILE_CONT 或 MEDIAN)，則所有呼叫必須使用相同的 ORDER BY 值。請注意，MEDIAN 會對表達式值套用隱含的 order by。

例如，下列陳述式會傳回錯誤。

```
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, pricepai...

ERROR: within group ORDER BY clauses for aggregate functions must be the same
```

下列陳述式會成功執行。

```
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (salesid)
from sales group by salesid, pricepaid;
```

範例

下列範例顯示 MEDIAN 產生與 PERCENTILE_CONT(0.5) 相同的結果。

```
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
---------+---------+-----------------+-------
       1 |       1 |             1.0 |    1.0
       2 |       3 |             3.0 |    3.0
       5 |       2 |             2.0 |    2.0
       9 |       4 |             4.0 |    4.0
      12 |       1 |             1.0 |    1.0
      16 |       1 |             1.0 |    1.0
      19 |       2 |             2.0 |    2.0
      19 |       3 |             3.0 |    3.0
```

```
      22 |       2 |             2.0 |    2.0
      25 |       2 |             2.0 |    2.0
```

STDDEV_SAMP 和 STDDEV_POP 函數

STDDEV_SAMP 和 STDDEV_POP 函數傳回一組數值 (整數、小數或浮點數) 的樣本標準差和母體標準差。STDDEV_SAMP 函數的結果相當於同一組值的樣本變異數平方根。

STDDEV_SAMP 和 STDDEV 是同一個函數的同義詞。

語法

```
STDDEV_SAMP | STDDEV ( [ DISTINCT | ALL ] expression)
STDDEV_POP ( [ DISTINCT | ALL ] expression)
```

表達式必須為整數、小數或浮點數資料類型。不論表達式的資料類型為何，此函數的傳回類型都是雙精確度數字。

> **Note**
>
> 標準差是採用浮點運算來計算，所得結果可能稍不精確。

使用須知

對包含單一值的表達式計算樣本標準差 (STDDEV 或 STDDEV_SAMP) 時，函數的結果為 NULL，不是 0。

範例

下列查詢傳回 VENUE 資料表的 VENUESEATS 欄中各值的平均值，接著傳回同一組值的樣本標準差和母體標準差。VENUESEATS 是 INTEGER 欄。結果的小數位數簡化到 2 位數。

```
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)
```

下列查詢傳回 SALES 資料表中的 COMMISSION 欄的樣本標準差。COMMISSION 是 DECIMAL 欄。結果的小數位數簡化到 10 位數。

```
select cast(stddev(commission) as dec(18,10))
from sales;

stddev
----------------
130.3912659086
(1 row)
```

下列查詢將 COMMISSION 欄的樣本標準差轉換為整數。

```
select cast(stddev(commission) as integer)
from sales;

stddev
--------
130
(1 row)
```

下列查詢傳回 COMMISSION 欄的樣本標準差和樣本變異數平方根。這些計算的結果相同。

```
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和SUM DISTINCT功能

該SUM函數返回輸入列或表達式值的總和。該SUM函數與數值一起使用，並忽略NULL值。

SUM DISTINCT函數會在計算總和之前，從指定的運算式中排除所有重複的值。

語法

```
SUM (column)
```

```
SUM (DISTINCT column )
```

引數

##

該函數運行的目標列。資料行為下列其中一種資料類型：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- DOUBLE

資料類型

SUM函數支援的引數類型為SMALLINTINTEGER、BIGINT、DECIMAL、和DOUBLE。

該SUM函數支持以下返回類型：

- BIGINT針對BIGINTSMALLINT、和INTEGER引數
- DOUBLE對於浮點引數
- 返回與任何其他參數類型的表達式相同的數據類型

帶有引數的SUM函數結果的默認有效位DECIMAL數為 38。結果的小數位數和引數的小數位數相同。例如，DEC(5,2)列SUM的 a 返回一個DEC(38,2)數據類型。

範例

查找從SALES表中支付的所有佣金的總和。

```
select sum(commission) from sales
```

查找從SALES表中支付的所有不同佣金的總和。

```
select sum (distinct (commission)) from sales
```

VAR_SAMP 和 VAR_POP 函數

VAR_SAMP 和 VAR_POP 函數傳回一組數值 (整數、小數或浮點數) 的樣本變異數和母體變異數。VAR_SAMP 函數的結果相當於同一組值的平方樣本標準差。

VAR_SAMP 和 VARIANCE 是同一個函數的同義詞。

語法

```
VAR_SAMP | VARIANCE ( [ DISTINCT | ALL ] expression)
VAR_POP ( [ DISTINCT | ALL ] expression)
```

表達式必須為整數、小數或浮點數資料類型。不論表達式的資料類型為何，此函數的傳回類型都是雙精確度數字。

> **Note**
>
> 這些函數的結果可能隨著資料倉儲叢集而有所不同，視每個案例中的叢集組態而定。

使用須知

對包含單一值的表達式計算樣本變異數 (VARIANCE 或 VAR_SAMP) 時，函數的結果為 NULL，不是 0。

範例

下列查詢傳回 LISTING 資料表中的 NUMTICKETS 欄的四捨五入樣本變異數和母體變異數。

```
select avg(numtickets),
round(var_samp(numtickets)) varsamp,
round(var_pop(numtickets)) varpop
from listing;

avg | varsamp | varpop
-----+---------+--------
10 |      54 |     54
(1 row)
```

下列查詢執行同樣的計算，但將結果轉換為小數值。

```
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)
```

陣列函數

本節說明中支援的 SQL 陣列函數AWS Clean Rooms。

主題

- 陣列函數
- 數組連接函數
- 數組平坦函數
- 獲取數組長度函數
- 分割到陣列函數
- 子陣列函數

陣列函數

創建 SUPER 數據類型的數組。

語法

```
ARRAY( [ expr1 ] [ , expr2 [ , ... ] ] )
```

引數

表示式 1，運算式 2

除了日期和時間類型之外的任何資料類型的運算式。參數不需要是相同的數據類型。

傳回類型

數組函數返回 SUPER 數據類型。

範例

下面的例子顯示了數值的數組和不同的數據類型的數組。

```
--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)
```

數組連接函數

該 array_concat 函數連接兩個數組以創建一個數組，該數組包含第一個數組中的所有元素，後跟第二個數組中的所有元素。這兩個參數必須是有效的數組。

語法

```
array_concat( super_expr1,  super_expr2 )
```

引數

超表達式 1

指定要串連的兩個陣列中第一個的值。

超表示式 2

指定要連接的兩個陣列中的第二個值。

傳回類型

該陣列連接函數返回一個超級數據值。

範例

下面的例子顯示了相同類型的兩個陣列和不同類型的兩個數組的連接的連接。

```
-- 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)
```

數組平坦函數

將多個陣列合併成 SUPER 類型的單個陣列。

語法

```
array_flatten( super_expr1,super_expr2,.. )
```

引數

超表示式 1，超級 _ 運算式 2

數組形式的有效 SUPER 表達式。

傳回類型

該陣列平坦函數返回一個超級數據值。

範例

下面的例子顯示了一個數組 _ 平坦函數。

```
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)
```

獲取數組長度函數

返回指定數組的長度。該 GET_ARRAY_LENGTH 函數返回給定的對象或數組路徑的超級數組的長度。

語法

```
get_array_length( super_expr )
```

引數

超級

數組形式的有效 SUPER 表達式。

傳回類型

該獲取數組長度函數返回一個大整數。

範例

下面的例子顯示了一個 get_array_length 函數。

```
SELECT GET_ARRAY_LENGTH(ARRAY(1,2,3,4,5,6,7,8,9,10));
 get_array_length
----------------------
            10
(1 row)
```

分割到陣列函數

使用分隔符號作為可選參數。如果沒有分隔符，則預設值為逗號。

語法

```
split_to_array( string,delimiter )
```

引數

string

要拆分的輸入字符串。

delimiter

在其上輸入字符串將被分割的可選值。預設為逗號。

傳回類型

分裂數組函數返回一個超級數據值。

範例

下面的例子顯示了一個分割數組函數。

```
SELECT SPLIT_TO_ARRAY('12|345|6789', '|');
     split_to_array
-------------------------
 ["12","345","6789"]
(1 row)
```

子陣列函數

操縱數組以返回輸入數組的子集。

語法

```
SUBARRAY( super_expr, start_position, length )
```

引數

超級

數組形式的有效 SUPER 表達式。

start_position

陣列中開始擷取的位置，從索引位置 0 開始。負位置從數組的末尾向後計數。

長度

要擷取的元素數目 (子字串的長度)。

傳回類型

子數組函數返回一個 SUPER 數據值。

範例

以下是子陣列函數的範例。

```
 SELECT SUBARRAY(ARRAY('a', 'b', 'c', 'd', 'e', 'f'), 2, 3);
   subarray
---------------
 ["c","d","e"]
(1 row)
```

條件式運算式

AWS Clean Rooms 支援下列條件運算式：

主題

- CASE 條件式運算式
- COALESCE表達
- GREATEST 和 LEAST 函數
- NVL 和 COALESCE 函數
- NVL2 函數
- NULLIF 函數

CASE 條件式運算式

CASE 表達式是條件式運算式，類似於其他語言中的 if/then/else 陳述式。有多個條件時會使用 CASE 來指定結果。在 SQL 運算式有效的情況下使用 CASE，例如在 SELECT 命令中。

CASE 表達式有兩種類型：簡單和搜尋。

- 在簡單 CASE 表達式中，表達式與值相比較。發現相符時，就套用 THEN 子句中指定的動作。未發現相符時，就套用 ELSE 子句中的動作。
- 在搜尋 CASE 表達式中，每一個 CASE 的評估根據為布林值表達式，而 CASE 陳述式會傳回第一個相符的 CASE。如果在 WHEN 子句之間找不到相符項目，就傳回 ELSE 子句中的動作。

語法

用來比對條件的簡單 CASE 陳述式：

```
CASE expression
  WHEN value THEN result
  [WHEN...]
  [ELSE result]
END
```

用來評估每一個條件的搜尋 CASE 陳述式：

```
CASE
  WHEN condition THEN result
  [WHEN ...]
  [ELSE result]
END
```

引數

運算式

欄名或任何有效表達式。

值

與表達式相比較的值，例如數值常數或字元字串。

result

評估表達式或布林值條件時傳回的目標值或表達式。所有結果運算式的資料類型必須轉換為單個輸出類型。

condition

評估 true 或 false 的布林值運算式。如果 condition 為真，CASE 運算式的值是遵循條件的結果，而 CASE 運算式的其餘部分則不會處理。如果 condition 為假，則評估任何後續 WHEN 子句。如果沒有 WHEN 條件結果為真，CASE 運算式的值是 ELSE 子句的結果。如果省略 ELSE 子句且沒有條件為 true，則結果為 Null。

範例

在針對 VENUE 資料表的查詢中，使用簡單 CASE 表達式以 `New York City` 取代 `Big Apple`。以 `other` 取代其他所有城市名稱。

```
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
...
```

使用搜尋 CASE 表達式以根據個別門票銷售的 PRICEPAID 值來指派群組號碼：

```
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
...
```

COALESCE表達

COALESCE運算式會傳回清單中第一個運算式非 null 的值。如果所有表達式都是 Null，則結果為 Null。找到非 Null 值時，就不會評估清單中剩餘的表達式。

如果您想在慣用值遺失或為 Null 時傳回備用值，這種表達式很有用。例如，查詢可能傳回三個電話號碼的其中之一 (依序為行動、住家或公司)，視資料表中最先找到何者而定 (不是 Null)。

語法

```
COALESCE (expression, expression, ... )
```

範例

將COALESCE表達式應用於兩列。

```
select coalesce(start_date, end_date)
from datetable
order by 1;
```

NVL 運算式的預設資料行名稱為。COALESCE下列查詢會傳回相同的結果。

```
select coalesce(start_date, end_date) from datetable order by 1;
```

GREATEST 和 LEAST 函數

從含有任何數量的表達式清單中傳回最大值或最小值。

語法

```
GREATEST (value [, ...])
```

```
LEAST (value [, ...])
```

參數

expression_list

逗號分隔的表達式清單，例如欄名。表達式必須全部可轉換為常見資料類型。忽略清單中的 NULL 值。如果所有表達式都評估為 NULL，則結果為 NULL。

傳回值

從提供的運算式清單中傳回最大值 (對於 GREATEST) 或最小值 (對於 LEAST)。

範例

下列範例按字母順序傳回 `firstname` 或 `lastname` 的最高值。

```
select firstname, lastname, greatest(firstname,lastname) from users
where userid < 10
order by 3;

 firstname | lastname  | greatest
-----------+-----------+-----------
 Alejandro | Rosalez   | Ratliff
 Carlos    | Salazar   | Carlos
 Jane      | Doe       | Doe
 John      | Doe       | Doe
 John      | Stiles    | John
 Shirley   | Rodriguez | Rodriguez
 Terry     | Whitlock  | Terry
 Richard   | Roe       | Richard
 Xiulan    | Wang      | Wang
(9 rows)
```

NVL 和 COALESCE 函數

傳回一系列運算式中不為 null 的第一個運算式的值。找到非 Null 值時，就不會評估清單中剩餘的運算式。

NVL 與 COALESCE 相同。它們是同義詞。本主題說明語法，並包含兩者的範例。

語法

```
NVL( expression, expression, ... )
```

COALESCE 的語法是相同的：

```
COALESCE( expression, expression, ... )
```

如果所有表達式都是 Null，則結果為 Null。

當您想要在主要值遺失或為 null 時傳回次要值，這些函數非常有用。例如，查詢可能會傳回三個可用電話號碼中的第一個：行動電話號碼、住家或公司。函數中運算式的順序決定評估的順序。

引數

運算式

要評估 Null 狀態的表達式，例如欄名。

傳回類型

AWS Clean Rooms 根據輸入運算式決定傳回值的資料類型。如果輸入運算式的資料類型沒有一般類型，則會傳回錯誤。

範例

如果清單包含整數運算式，該函數傳回一個整數。

```
SELECT COALESCE(NULL, 12, NULL);

coalesce
--------------
12
```

這個範例與前面的範例相同，不同之處在於它使用 NVL，會傳回相同的結果。

```
SELECT NVL(NULL, 12, NULL);

coalesce
```

```
--------------
12
```

下列範例會傳回字串類型。

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', NULL);

coalesce
--------------
AWS Clean Rooms
```

下列範例會導致錯誤，因為運算式清單中的資料類型不同。在這種情況下，清單中同時存在字串類型和數字類型。

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', 12);
ERROR: invalid input syntax for integer: "AWS Clean Rooms"
```

NVL2 函數

根據指定的表達式評估為 NULL 還是 NOT NULL，傳回兩個值中的一個。

語法

```
NVL2 ( expression, not_null_return_value, null_return_value )
```

引數

運算式

要評估 Null 狀態的表達式，例如欄名。

not_null_return_value

expression 評估為 NOT NULL 時所傳回的值。not_null_return_value 值的資料類型必須與 expression 相同，或可隱含地轉換為該資料類型。

null_return_value

expression 評估為 NULL 時所傳回的值。null_return_value 值的資料類型必須與 expression 相同，或可隱含地轉換為該資料類型。

傳回類型

NVL2 傳回類型的決定方式如下：

- 如果 not_null_return_value 或 null_return_value 為 Null，則傳回 not-null 表達式的資料類型。

如果 not_null_return_value 和 null_return_value 都不是 Null：

- 如果 not_null_return_value 和 null_return_value 有相同的資料類型，則傳回該資料類型。
- 如果 not_null_return_value 和 null_return_value 有不同的數值資料類型，則傳回最小可相容的數值資料類型。
- 如果 not_null_return_value 和 null_return_value 有不同的日期時間資料類型，則傳回時間戳記資料類型。
- 如果 not_null_return_value 和 null_return_value 有不同的字元資料類型，則傳回 not_null_return_value 的資料類型。
- 如果 not_null_return_value 和 null_return_value 有混合的數值和非數值資料類型，則傳回 not_null_return_value 的資料類型。

> **Important**
>
> 前兩個案例中傳回 not_null_return_value 的資料類型，而 null_return_value 會隱含地轉換為該資料類型。如果資料類型不相容，函數會失敗。

使用須知

如果是 NVL2，傳回的值會是不是 Not_null_return 值或空值參數，以函數選取為準，但會有資料類型為無。

例如，假設 column1 是 NULL，下列查詢會傳回相同的值。不過，DECODE 傳回值的資料類型為 INTEGER，而 NVL2 傳回值的資料類型為 VARCHAR。

```
select decode(column1, null, 1234, '2345');
select nvl2(column1, '2345', 1234);
```

範例

下列範例修改一些範例資料，然後評估兩個欄位來提供使用者的適當聯絡資訊：

```
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 (555) 555-0100
Caldwell Acevedo  Nunc.sollicitudin@example.ca
Quinn Adams     vel@example.com
Kamal Aguilar   quis@example.com
Samson Alexander  hendrerit.neque@example.com
Hall Alford     ac.mattis@example.com
Lane Allen      et.netus@example.com
Xander Allison    ac.facilisis.facilisis@example.com
Amaya Alvarado    dui.nec.tempus@example.com
Vera Alvarez    at.arcu.Vestibulum@example.com
Yetta Anthony   enim.sit@example.com
Violet Arnold   ad.litora@example.comm
August Ashley   consectetuer.euismod@example.com
Karyn Austin    ipsum.primis.in@example.com
Lucas Ayers     at@example.com
```

NULLIF 函數

語法

NULLIF 表達式比較兩個引數，如果引數相等，則傳回 Null。如果不相等，則傳回第一個引數。此表達式與 NVL 或 COALESCE 表達式相反。

```
NULLIF ( expression1, expression2 )
```

引數

expression1、expression2

比較的目標欄或表達式。傳回類型與第一個表達式的類型相同。NULLIF 結果的預設欄名為第一個表達式的欄名。

範例

在下列範例中，查詢會傳回字串 first，因為引數不相等。

```
SELECT NULLIF('first', 'second');

case
-------
first
```

在下列範例中，查詢會傳回 NULL，因為字串常值引數相等。

```
SELECT NULLIF('first', 'first');

case
-------
NULL
```

在下列範例中，查詢會傳回 1，因為整數引數不相等。

```
SELECT NULLIF(1, 2);

case
-------
1
```

在下列範例中，查詢會傳回 NULL，因為整數引數相等。

```
SELECT NULLIF(1, 1);

case
-------
NULL
```

在下列範例中，當 LISTID 和 SALESID 值相符時，查詢會傳回 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)
```

資料類型格式化函數

使用數據類型格式化函數，您可以將值從一種數據類型轉換為另一種數據類型。對於這些函數中的每一個，第一個參數始終是要格式化的值，第二個參數包含新格式的模板。 AWS Clean Rooms 支持多種數據類型格式化功能。

主題

- CAST 函數
- CONVERT 函數
- TO_CHAR
- TO_DATE 陣列
- TO_NUMBER
- 日期時間格式字串
- 數值格式字串
- Teradata樣式格式化數字資料的字元

CAST 函數

CAST 函數會將一種資料類型轉換為另一個相容的資料類型。例如，您可以將字串轉換為日期，或將數值類型轉換為字串。CAST 會執行執行期轉換，這表示轉換不會變更來源資料表中值的資料類型。它僅在查詢的上下文中進行更改。

CAST 函數非常類似 the section called “CONVERT”，因為它們都將一種資料類型轉換為另一種資料類型，但它們的呼叫方式不同。

某些資料類型需要使用 CAST 或 CONVERT 函數來明確轉換為其他資料類型。其他資料類型可在另一個命令中隱含地轉換，而不需要使用 CAST 或 CONVERT。請參閱類型相容性與轉換。

語法

使用兩種同等的語法格式中的任何一種，將運算式從一種資料類型轉換為另一種資料類型。

```
CAST ( expression AS type )
expression :: type
```

引數

運算式

任何評估為一或多個值的表達式，例如欄名或常值。轉換 Null 值會傳回 Null。運算式不能包含空白或空白字串。

type

支援的其中一個資料類型，除了 VARBYTE、二進位和二進位變異資料類型。

傳回類型

CAST 傳回 type 引數指定的資料類型。

> **Note**
>
> AWS Clean Rooms 如果您嘗試執行有問題的轉換，例如失去精確度的 DECIMAL 轉換，就會傳回錯誤，如下所示：
>
> ```
> select 123.456::decimal(2,1);
> ```
>
> 或造成溢位的 INTEGER 轉換：
>
> ```
> select 12345678::smallint;
> ```

範例

下列兩個查詢相同。都是將小數值轉換為整數：

```
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)
```

以下產生類似的結果。它不需要執行範例資料：

```
select cast(162.00 as integer) as pricepaid;

pricepaid
-----------
162
(1 row)
```

在此範例中，時間戳記資料欄中的值會轉換為日期，因此會從每個結果中移除時間：

```
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)
```

如果您沒有按照上一個範例中所示使用 CAST，則結果將包括時間：2008-02-18 02:36:48。

下列查詢會將可變字元資料轉換為日期。它不需要執行範例資料。

```
select cast('2008-02-18 02:36:48' as date) as mysaletime;

mysaletime
--------------------
2008-02-18
(1 row)
```

在此範例中，日期欄中的值轉換為時間戳記：

```
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)
```

在與上一個示例類似的情況下，您可以通過使用來獲得對輸出格式的其他控制TO_CHAR。

在此範例中，整數轉換為字元字串：

```
select cast(2008 as char(4));

bpchar
--------
2008
```

在此範例中，DECIMAL(6,3) 值轉換為 DECIMAL(4,1) 值：

```
select cast(109.652 as decimal(4,1));

numeric
---------
109.7
```

此範例顯示更複雜的運算式。SALES 資料表中的 PRICEPAID 欄 (DECIMAL(8,2) 欄) 轉換為 DECIMAL(38,2) 欄，並將值乘以 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)
```

CONVERT 函數

像CAST 函數，CONVERT 函數將一種數據類型轉換為另一個兼容的數據類型。例如，您可以將字串轉換為日期，或將數字類型轉換為字串。CONVERT 會執行執行期轉換，這表示轉換不會變更來源資料表中值的資料類型。它僅在查詢的上下文中進行更改。

某些資料類型需要使用 CONVERT 函數來明確轉換為其他資料類型。其他資料類型可在另一個命令中隱含地轉換，而不需要使用 CAST 或 CONVERT。請參閱類型相容性與轉換。

語法

```
CONVERT ( type, expression )
```

引數

type

支援的其中一個資料類型，除了 VARBYTE、二進位和二進位變異資料類型。

運算式

任何評估為一或多個值的表達式，例如欄名或常值。轉換 Null 值會傳回 Null。運算式不能包含空白或空白字串。

傳回類型

CONVERT 傳回 type 引數指定的資料類型。

> **Note**
>
> AWS Clean Rooms 如果您嘗試執行有問題的轉換，例如失去精確度的 DECIMAL 轉換，就會傳回錯誤，如下所示：
>
> ```
> SELECT CONVERT(decimal(2,1), 123.456);
> ```
>
> 或造成溢位的 INTEGER 轉換：
>
> ```
> SELECT CONVERT(smallint, 12345678);
> ```

範例

下列查詢使用 CONVERT 函數將一欄小數轉換為整數

```
SELECT CONVERT(integer, pricepaid)
FROM sales WHERE salesid=100;
```

這個範例會將整數轉換為字元字串。

```
SELECT CONVERT(char(4), 2008);
```

此範例會將目前日期和時間轉換為可變字元資料類型：

```
SELECT CONVERT(VARCHAR(30), GETDATE());

getdate
---------
2023-02-02 04:31:16
```

這個範例會將 saletime 欄轉換成僅限時間，並從每一列移除日期。

```
SELECT CONVERT(time, saletime), salesid
FROM sales order by salesid limit 10;
```

下列範例會將可變字元資料轉換成日期時間物件。

```
SELECT CONVERT(datetime, '2008-02-18 02:36:48') as mysaletime;
```

TO_CHAR

TO_CHAR 將時間戳記或數值運算式轉換為字元字串資料格式。

語法

```
TO_CHAR (timestamp_expression | numeric_expression , 'format')
```

引數

timestamp_expression

此運算式產生 TIMESTAMP 或 TIMESTAMPTZ 類型值，或可隱含地強制轉換為時間戳記的值。

numeric_expression

此表達式產生數值資料類型的值，或可隱含地強制轉換為數值類型的值。如需詳細資訊，請參閱 數值類型。TO_CHAR 在數值字串左側插入空格。

> **Note**
>
> 字符不支持 128 位十進制值。

format

新值的格式。關於有效的格式，請參閱日期時間格式字串和 數值格式字串。

傳回類型

VARCHAR

範例

下列範例會將時間戳記轉換為日期和時間的值，其格式為月份名稱填滿九個字元、星期名稱以及月份的日期編號。

```
select to_char(timestamp '2009-12-31 23:15:59', 'MONTH-DY-DD-YYYY HH12:MIPM');
to_char
-------------------------
DECEMBER -THU-31-2009 11:15PM
```

下列範例會將時間戳記轉換為具有年份天數的值。

```
select to_char(timestamp '2009-12-31 23:15:59', 'DDD');

to_char
-------------------------
365
```

下列範例會將時間戳記轉換為一週的 ISO 天數。

```
select to_char(timestamp '2022-05-16 23:15:59', 'ID');

to_char
-------------------------
1
```

以下範例會從日期擷取月份名稱。

```
select to_char(date '2009-12-31', 'MONTH');

to_char
-------------------------
DECEMBER
```

下列範例將 EVENT 資料表中的每一個 STARTTIME 值，轉換為由時、分、秒組成的字串。

```
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
(5 rows)
```

下列範例將整個時間戳記值轉換成另一種格式。

```
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
(1 row)
```

下列範例將時間戳記常值轉換為字元字串。

```
select to_char(timestamp '2009-12-31 23:15:59','HH24:MI:SS');
to_char
----------
23:15:59
(1 row)
```

下列範例將數字轉換為結尾帶負號的字元字串。

```
select to_char(-125.8, '999D99S');
to_char
---------
125.80-
(1 row)
```

下列範例將數字轉換為帶有貨幣符號的字元字串。

```
select to_char(-125.88, '$S999D99');
to_char
---------
$-125.88
(1 row)
```

下列範例將數字轉換為字元字串，並使用角括號代表負數。

```
select to_char(-125.88, '$999D99PR');
to_char
---------
$<125.88>
(1 row)
```

下列範例將數字轉換為 Roman 數值字串。

```
select to_char(125, 'RN');
to_char
---------
CXXV
(1 row)
```

下列範例會顯示星期幾。

```
SELECT to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS');
               to_char
-----------------------
Wednesday, 31 09:34:26
```

下列範例顯示數字的序號字尾。

```
SELECT to_char(482, '999th');
               to_char
-----------------------
 482nd
```

下列範例將 sales 資料表中的支付價格減去佣金。然後將差異四捨五入並轉換為羅馬數字，顯示在to_char列中：

```
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
(10 rows)
```

下列範例會將貨幣符號新增至to_char欄中顯示的差異值：

```
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
(10 rows)
```

下列範例列出每一筆銷售完成的世紀。

```
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
(10 rows)
```

下列範例將 EVENT 資料表中的每一個 STARTTIME 值，轉換為由時、分、秒及時區組成的字串。

```
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
(5 rows)

(10 rows)
```

下列範例顯示秒、毫秒和微秒的格式。

```
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 陣列

TO_DATE 將字元字串所表示的日期轉換為 DATE 資料類型。

語法

```
TO_DATE(string, format)
```

```
TO_DATE(string, format, is_strict)
```

引數

string

要轉換的字串。

format

字串常值，定義輸出 string 的日期部分格式。如需有效日、月和年格式的清單，請參閱日期時間格式字串。

is_strict

選用的布林值，指定如果輸入日期值超出範圍，是否會傳回錯誤。當 is_strict 設定為 TRUE 時，如果有超出範圍的值，就會傳回錯誤。當 is_strict 設定為 FALSE (預設值) 時，就會接受溢位值。

傳回類型

TO_DATE 傳回 DATE，視 format 值而定。

如果轉換成 format 失敗，則會傳回錯誤。

範例

下列 SQL 陳述式會將日期 `02 Oct 2001` 轉換為日期資料類型。

```
select to_date('02 Oct 2001', 'DD Mon YYYY');

to_date
------------
2001-10-02
```

```
(1 row)
```

下列 SQL 陳述式會將字串 20010631 轉換為日期。

```
select to_date('20010631', 'YYYYMMDD', FALSE);
```

結果是 2001 年 7 月 1 日，因為 6 月只有 30 天。

```
to_date
------------
2001-07-01
```

下列 SQL 陳述式會將字串 20010631 轉換為日期：

```
to_date('20010631', 'YYYYMMDD', TRUE);
```

結果是錯誤，因為六月只有 30 天。

```
ERROR:  date/time field date value out of range: 2001-6-31
```

TO_NUMBER

TO_NUMBER 將字串轉換為數值 (十進位)。

語法

```
to_number(string, format)
```

引數

string

要轉換的字串。格式必須是文字值。

format

第二個引數是格式字串，指出如何剖析字元字串來建立數值。例如，格式 '99D999' 指定要轉換的字串包含五位數，且第三個位置是小數點。例如，to_number('12.345','99D999') 會將以數值傳回 12.345。如需有效格式的清單，請參閱 數值格式字串。

傳回類型

TO_NUMBER 傳回 DECIMAL 數字。

如果轉換成 format 失敗，則會傳回錯誤。

範例

下列範例將字串 12,454.8- 轉換為數字：

```
select to_number('12,454.8-', '99G999D9S');

to_number
-----------
-12454.8
```

下列範例將字串 $ 12,454.88 轉換為數字：

```
select to_number('$ 12,454.88', 'L 99G999D99');

to_number
-----------
12454.88
```

下列範例將字串 $ 2,012,454.88 轉換為數字：

```
select to_number('$ 2,012,454.88', 'L 9,999,999.99');

to_number
-----------
2012454.88
```

日期時間格式字串

下列日期時間格式字串適用於諸如 TO_CHAR 之類的函數。這些字串可以包含日期時間分隔符號 (例如 '-'、'/' 或 ':') 及下列「日期部分」和「時間部分」。

如需將日期格式化為字串的範例，請參閱TO_CHAR。

日期部分或時間部分	意義
BC 或 B.C.、AD 或 A.D.、b.c. 或 bc、ad 或 a.d。	大寫和小寫的紀元標記
CC	兩位數世紀編號
YYYY、YYY、YY、Y	4 位數、3 位數、2 位數、1 位數的年編號
Y、YYY	含逗號的 4 位數年編號
IYYY、IYY、IY、I	4 位數、3 位數、2 位數、1 位數的國際標準組織 (ISO) 年編號
Q	季編號 (1 到 4)
MONTH、Month、month	月名稱 (大寫、大小寫混合、小寫、以空格填補為 9 個字元)
MON、Mon、mon	縮寫月名稱 (大寫、大小寫混合、小寫、以空格填補為 3 個字元)
MM	月編號 (01-12)
RM、rm	羅馬數字的月編號 (I-XII，其中 I 代表一月，大寫或小寫)
W	月的第幾週 (1-5；第一週以月初第一天起算。)
WW	年的週編號 (1-53；第一週以年初第一天起算。)
IW	年的 ISO 週編號 (新年第一個星期四在第 1 週。)
DAY、Day、day	日名稱 (大寫、大小寫混合、小寫、以空格填補為 9 個字元)
DY、Dy、dy	縮寫日名稱 (大寫、大小寫混合、小寫、以空格填補為 3 個字元)

日期部分或時間部分	意義
DDD	年的第幾日 (001-366)
IDDD	ISO 8601 週編號年的第幾日 (001-371；一年的第 1 日是第一個 ISO 週的星期一)
DD	月的第幾日，以數字表示 (01-31)
D	星期幾 (1-7；星期日是 1) **Note** D 日期部分的行為不同於日期時間函數 DATE_PART 和 EXTRACT 使用的星期幾 (DOW) 日期部分。DOW 是基於整數 0-6，其中星期日為 0。如需詳細資訊，請參閱 日期或時間戳記函數的日期部分。
ID	ISO 8601 一週的星期幾，星期一 (1) 至星期日 (7)
J	羅馬曆日 (自紀元前 4712 年 1 月 1 日起算的天數)
HH24	小時 (24 小時制，00-23)
HH 或 HH12	小時 (12 小時制，01-12)
MI	分鐘 (00–59)
SS	秒 (00–59)
MS	毫秒 (.000)
US	微秒 (.000000)

日期部分或時間部分	意義
AM 或 PM、A.M. 或 P.M.、a.m. 或 p.m.、am 或 pm	大寫和小寫正午指標 (用於 12 小時制)
TZ、tz	大寫和小寫時區縮寫；僅適用於 TIMESTAMP TZ
OF	UTC 時差；僅適用於 TIMESTAMPTZ

> **Note**
>
> 您必須以單引號括住日期時間分隔符號 (例如 '-'、'/' 或 ':')，但必須以雙引號括住上表所列的「日期部分」和「時間部分」。

數值格式字串

下列數值格式字串適用於諸如 TO_NUMBER 和 TO_CHAR 之類的函數。

- 如需將字串格式化為數字的範例，請參閱TO_NUMBER。
- 如需將數字格式化為字串的範例，請參閱TO_CHAR。

格式	描述
9	含指定位數的數值。
0	開頭為零的數值。
. (點)、D	小數點。
, (逗號)	千位分隔符號。
CC	世紀代碼。例如，第 21 世紀從 2001-01-01 開始 (僅適用於 TO_CHAR)。
FM	填補模式。禁止填補空格和零。

格式	描述
PR	角括號中的負值。
S	緊貼於數字的正負號。
L	指定位置中的貨幣符號。
G	群組分隔符號。
MI	在小於 0 的數字中，位於指定位置的減號。
PL	在大於 0 的數字中，位於指定位置的加號。
SG	指定位置中的加號或減號。
RN	介於 1 和 3999 之間的羅馬數字 (僅適用於 TO_CHAR)。
TH 或 th	序號字尾。不轉換小於零的小數或值。

Teradata樣式格式化數字資料的字元

本主題向您展示文字 _ 到 _ 英特 _ 替代和文字至 _ 數值 _ ALT 函數如何解譯輸入運算式字串中的字元。在下表中，您還可以找到可以在格式短語中指定的字符列表。此外，您還可以找到 Teradata 樣式格式與 AWS Clean Rooms 格式選項之間差異的說明。

格式	描述
G	輸入 expression 字串中不支援做為群組分隔符號。您無法在 format 詞語中指定此字元。
D	基數符號。您可以在 format 詞語中指定此字元。此字元相當於 .(句號)。 基數符號不能以包含下列任何字元的格式片語出現： • .(句點)

格式	描述
	• S (大寫 's') • V (大寫 'v')
/ , : %	插入字元 / (正斜線)、逗號 (,)、: (冒號) 和 % (百分比符號)。 您不能在 format 詞語中包含這些字元。 AWS Clean Rooms 會忽略輸入運算式字串中的這些字元。
.	句點作為基數字符，即小數點。 此字元不能出現在以包含下列任何字元的 format 詞語中： • D (大寫 'd') • S (大寫 's') • V (大寫 'v')
B	您不能在 format 詞語中包含空格字元 (B)。在輸入 expression 字串中，開頭和結尾空格被忽略，並且不允許數字之間的空格。
+ -	您不能在 format 詞語中包含加號 (+) 或減號 (-)。但是，如果加號 (+) 和減號 (-) 出現在輸入 expression 字串中，則會隱含地剖析為數值的一部分。
V	小數點位置指示器。 此字元不能出現在以包含下列任何字元的 format 詞語中： • D (大寫 'd') • .(句點)

格式	描述
Z	零抑制的十進位數字。 AWS Clean Rooms 修剪前導零。Z 字元不能跟在 9 字元後面。如果小數部分包含 9 字元，則 Z 字元必須位於基數字元的左側。
9	小數位。
CHAR(n)	對於此格式，您可以指定下列選項： • 字符由 Z 或 9 個字符組成。 AWS Clean Rooms CHAR 值中不支援 + (加號) 或-(減號)。 • n 是整數常數 I 或 F。對於 I，這是顯示數字或整數資料之整數部分所需的字元數。對於 F，這是顯示數值資料的小數部分所需的字元數。
-	連字號 (-) 字元。 您不能在 format 詞語中包含此字元。 AWS Clean Rooms 會忽略輸入運算式字串中的這個字元。

格式	描述
S	簽名分區十進制。S 字元必須位於 format 詞語中最後一個十進位數字之後。輸入 expression 字串的最後一個字元和對應的數字轉換在 已簽署區域十進位、Teradata 樣式數值資料格式的資料格式化字元 中列出。 S 字元不能出現在包含下列任何字元的 format 詞語中： • + (加號) • .(句點) • D (大寫 'd') • Z (大寫 'z') • F (大寫 'f') • E (大寫 'e')
E	指數表示法。輸入 expression 字串可以包含指數字元。您不能在 format 詞語中指定 E 作為指數字元。
FN9	在中不支援 AWS Clean Rooms。
FNE	在中不支援 AWS Clean Rooms。
$、USD、美元	貨幣符號 ($)、ISO 貨幣符號 (USD)，以及貨幣名稱美元。 ISO 貨幣符號 USD 和貨幣名稱美元區分大小寫。 AWS Clean Rooms 僅支持美元貨幣。輸入 expression 字串可以包含美元貨幣符號與數值之間的空格，例如「$123E2」或「123E2 $」。
L	貨幣符號。此貨幣符號字元只能在 format 詞語中出現一次。您無法指定重複的貨幣符號字元。

格式	描述
C	ISO 貨幣符號。此貨幣符號字元只能在 format 詞語中出現一次。您無法指定重複的貨幣符號字元。
N	貨幣完整名稱。此貨幣符號字元只能在 format 詞語中出現一次。您無法指定重複的貨幣符號字元。
O	雙重貨幣符號。您無法在 format 詞語中指定此字元。
U	雙 ISO 貨幣符號。您無法在 format 詞語中指定此字元。
A	完整的雙重貨幣名稱。您無法在 format 詞語中指定此字元。

已簽署區域十進位、Teradata 樣式數值資料格式的資料格式化字元

您可以在 TEXT_TO_INT_ALT 和 TEXT_TO_NUMERIC_ALT 函數的 format 詞語中使用以下字元來表示帶符號的分區十進位值。

輸入字串的最後一個字元	數值轉換
{ 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

輸入字串的最後一個字元	數值轉換
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
N	-n … 5
O	-n … 6
P	-n … 7
Q	-n … 8
R	-n … 9

日期和時間函數

AWS Clean Rooms 支援下列日期和時間函數：

主題

- DATEADD 函數
- DATEDIFF 函數
- DATE_PART 函數
- DATE_TRUNC 函數
- EXTRACT 函數
- GETDATE 函數
- SYSDATE 函數
- TIMEOFDAY 函數
- TO_TIMESTAMP 函數
- 日期或時間戳記函數的日期部分

日期和時間函數的摘要

下表提供中所使用之日期與時間函數的摘要 AWS Clean Rooms。

函數	語法	傳回值
+ (串連) 運算子 將日期與 + 符號兩側的時間串連起來，並傳回 TIMESTAMP 或 TIMESTAMPTZ。	date + time	TIMESTAMP 或 TIMESTAMPZ
ADD_MONTHS 將指定幾個月新增至日期或時間戳記。	ADD_MONTHS ({date|timestamp}, integer)	TIMESTAMP
CURRENT_DATE 函數 傳回目前工作階段時區中的日期 (預設為 UTC) 做為目前交易的開始。	CURRENT_DATE	DATE
DATEADD 透過指定間隔來增量日期或時間。	DATEADD (datepart, interval, {date|time|timetz|timestamp})	TIMESTAMP、TIME 或 TIMETZ

函數	語法	傳回值
DATEDIFF 傳回兩個日期或時間的差異，做為給定日期的部分 (例如日或月)。	DATEDIFF (datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp})	BIGINT
DATE_PART 從日期或時間擷取日期部分值。	DATE_PART (datepart, {date|timestamp})	DOUBLE
DATE_TRUNC 根據日期部分來截斷時間戳記。	DATE_TRUNC ('datepart', timestamp)	TIMESTAMP
EXTRACT 從 timestamp、timestamptz、time 或 timetz 中擷取日期或時間部分。	EXTRACT (datepart FROM source)	INTEGER or DOUBLE
GETDATE 函數 傳回目前工作階段時區中的目前日期和時間 (預設為 UTC)。括號是必要的。	GETDATE()	TIMESTAMP
SYSDATE 傳回日期和時間 (以 UTC 表示) 做為目前交易的開始。	SYSDATE	TIMESTAMP
TIMEOFDAY 傳回目前工作階段時區中的目前工作日 (預設為 UTC) 做為字串值。	TIMEOFDAY()	VARCHAR

函數	語法	傳回值
TO_TIMESTAMP 傳回含時區的時間戳記，做為指定時間戳記和時區格式。	TO_TIMESTAMP ('timestamp', 'format')	TIMESTAMP TZ

> **Note**
>
> 不會將閏秒視為歷經時間的計算中。

交易中日期與時間函數

當您在交易區塊 (BEGIN … END) 中執行下列函數時，函數會傳回目前交易的開始時間，而不是目前陳述式的開始。

- SYSDATE
- TIMESTAMP
- CURRENT_DATE

下列函數一律會傳回目前陳述式的開始日期或時間 (即使他們在交易區塊中)。

- GETDATE
- TIMEOFDAY

+ (串連) 運算子

連接數字文字，字符串文字和/或日期時間和間隔文字。它們位於 + 符號的兩側，並根據 + 符號任一側的輸入返回不同類型。

語法

```
numeric + string
```

```
date + time
```

```
date + timetz
```

引數的順序可以反轉。

引數

####

表示數字的常值或常數，可以是整數或浮點數。

#####

字串、字元字串或字元常數

date

資DATE料行或表示式，隱含地轉換為DATE.

time

資TIME料行或表示式，隱含地轉換為TIME.

timetz

資TIMETZ料行或表示式，隱含地轉換為TIMETZ.

範例

下列範例表格TIME_TEST有一個欄 TIME_VAL (typeTIME)，其中插入了三個值。

```
select date '2000-01-02' + time_val as ts from time_test;
```

ADD_MONTHS 函數

ADD_MONTHS 會將指定幾個月新增至日期或時間戳記值或運算式。DATEADD 函數提供類似功能。

語法

```
ADD_MONTHS( {date | timestamp}, integer)
```

引數

date | timestamp

隱含轉換為日期或時間戳記的日期或時間戳記資料行或運算式。如果日期是某個月的最後一天，或者如果產生的月份是較短的月份，則函數會在結果中傳回該月的最後一天。若是其他日期，結果會包含與日期表達式一樣的相同天數。

integer

正或負整數。使用負數可減少日期中的月份。

傳回類型

TIMESTAMP

範例

以下查詢會使用 TRUNC 函數中的 ADD_MONTHS 函數。TRUNC 函數會從 ADD_MONTHS 結果移除某日時間。ADD_MONTHS 函數會從 CALDATE 欄中新增 12 個月至每個值。

```
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)
```

下列範例示範當 ADD_MONTHS 函數對含有月份的日期進行操作的行為，而這些月份的天數皆不同時。

```
select add_months('2008-03-31',1);

add_months
---------------------
2008-04-30 00:00:00
```

```
(1 row)

select add_months('2008-04-30',1);

add_months
---------------------
2008-05-31 00:00:00
(1 row)
```

CONVERT_TIMEZONE 函數

CONVERT_TIMEZONE 可將時間戳記從一個時區轉換為另一個時區。此功能會根據日光節約時間自動調整。

語法

```
CONVERT_TIMEZONE ( ['source_timezone',] 'target_timezone', 'timestamp')
```

引數

source_timezone

(選用) 目前時間戳記的時區。預設值為 UTC。

target_timezone

新時間戳記的時區。

timestamp

時間戳記欄或運算式會隱性轉換為時間戳記。

傳回類型

TIMESTAMP

範例

以下範例會將時間戳記值從預設 UTC 時區轉換為 PST。

```
select convert_timezone('PST', '2008-08-21 07:23:54');
```

```
 convert_timezone
-----------------------
2008-08-20 23:23:54
```

以下範例會將 LISTTIME 欄位中的時間戳記值從預設 UTC 時區轉換為 PST。即使時間戳記是在日光節約時間期間，其會轉換為標準時間，因為目標時區是指定為縮寫 (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
```

以下範例會將時間戳記 LISTTIME 欄位從預設 UTC 時區轉換為 US/Pacific 時區。目標時區使用時區名稱，且時間戳記是在日光節約時間期間，因此函數會傳回日光時間。

```
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
```

以下範例會將時間戳記字串從 EST 轉換為 PST：

```
select convert_timezone('EST', 'PST', '20080305 12:25:29');

 convert_timezone
-------------------
2008-03-05 09:25:29
```

以下範例會將時間戳記轉換為美國東部標準時間，因為目標時區使用時區名稱 (America/New_York) 且時間戳記是在標準時間期間。

```
select convert_timezone('America/New_York', '2013-02-01 08:00:00');

 convert_timezone
---------------------
2013-02-01 03:00:00
(1 row)
```

以下範例會將時間戳記轉換為美國東部日光時間，因為目標時區使用時區名稱 (America/New_York) 且時間戳記是在日光時間期間。

```
select convert_timezone('America/New_York', '2013-06-01 08:00:00');

 convert_timezone
---------------------
2013-06-01 04:00:00
(1 row)
```

以下範例示範的是偏移的使用。

```
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 函數

CURRENT_DATE 傳回目前工作階段時區中的日期 (預設為 UTC)，使用預設格式：YYYY-MM-DD。

> **Note**
>
> CURRENT_DATE 傳回目前交易的日期 (而不是目前陳述式的開始)。考慮以下場景：您在 10/01/08 23:59 啟動包含多個陳述式的交易，而包含 CURRENT_DATE 的陳述式在 10/02/08 00:00 執行。CURRENT_DATE 傳回 10/01/08，而不是 10/02/08。

語法

```
CURRENT_DATE
```

傳回類型

DATE

範例

下列範例會傳回目前的日期 (函式執行的 AWS 區域 位置)。

```
select current_date;

   date
------------
2008-10-01
```

DATEADD 函數

透過指定間隔來增量 DATE、TIME、TIMETZ 或 TIMESTAMP 值。

語法

```
DATEADD( datepart, interval, {date|time|timetz|timestamp} )
```

引數

datepart

函數對其執行的日期部分 (例如，年、月、日或小時)。如需詳細資訊，請參閱 日期或時間戳記函數的日期部分。

間隔

指定要新增至目標表達式之間隔 (例如，天數) 的整數。負整數會減去間隔。

date|time|timetz|timestamp

DATE、TIME、TIMETZ 或 TIMESTAMP 欄或隱含轉換為 DATE、TIME、TIMETZ 或 TIMESTAMP 的運算式。DATE、TIME、TIMETZ 或 TIMESTAMP 運算式必須包含指定的日期部分。

傳回類型

TIMESTAMP 或 TIME 或 TIMETZ 取決於輸入資料類型。

具有 DATE 欄的範例

下列範例會向 DATE 資料表中存在的 11 月份的每個日期增加 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)
```

下列範例會將 18 個月增加至常值日期值。

```
select dateadd(month,18,'2008-02-28');

date_add
---------------------
2009-08-28 00:00:00
(1 row)
```

DATEADD 函數的預設欄名稱為 DATE_ADD。日期值的預設時間戳記為 00:00:00。

下列範例會將 30 分鐘增加至未指定時間戳記的日期值。

```
select dateadd(m,30,'2008-02-28');

date_add
---------------------
2008-02-28 00:30:00
(1 row)
```

您可以用全名或縮寫來表示日期部分。在這種情況下，m 代表分鐘，而不是月。

具有 TIME 欄的範例

下列範例資料表 TIME_TEST 有一個 TIME_VAL 欄 (類型為 TIME)，其中插入了三個值。

```
select time_val from time_test;

time_val
```

```
---------------------
20:00:00
00:00:00.5550
00:58:00
```

下列範例會將 5 分鐘增加至 TIME_TEST 資料表中的每個 TIME_VAL。

```
select dateadd(minute,5,time_val) as minplus5 from time_test;

minplus5
---------------
20:05:00
00:05:00.5550
01:03:00
```

下列範例會將 8 小時增加至常值時間值。

```
select dateadd(hour, 8, time '13:24:55');

date_add
---------------
21:24:55
```

下列範例會顯示時間超過 24:00:00 或在 00:00:00 以下的時間。

```
select dateadd(hour, 12, time '13:24:55');

date_add
---------------
01:24:55
```

具有 TIMTZ 欄的範例

這些範例中的輸出值是以 UTC 為預設時區。

下列範例資料表 TIMETZ_TEST 有一個 TIMETZ_VAL 欄 (類型為 TIMETZ)，其中插入了三個值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
```

```
00:00:00.5550+00
05:58:00+00
```

下列範例會將 5 分鐘增加至 TIMETZ_TEST 資料表中的每個 TIMETZ_VAL。

```
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
```

下列範例會將 2 小時加入常值 timetz 值。

```
select dateadd(hour, 2, timetz '13:24:55 PST');

date_add
---------------
23:24:55+00
```

具有 TIMESTAMP 欄的範例

這些範例中的輸出值是以 UTC 為預設時區。

下列範例資料表 TIMESTAMP_TEST 有一個 TIMESTAMP_VAL 欄 (類型為 TIMESTAMP)，其中插入了三個值。

```
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
```

以下範例僅將 20 年增加至 2000 年之前的 TIMESTAMP_TEST 中的 TIMESTAMP_VAL 值。

```
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
```

下列範例將 5 秒加入不帶秒指示器的常值時間戳記值。

```
SELECT dateadd(second, 5, timestamp '2001-06-06');

date_add
---------------
2001-06-06 00:00:05
```

使用須知

DATEADD(month, ...) 和 ADD_MONTHS 函數處理落在月底之日期的方式會所有不同：

- ADD_MONTHS：如果您要新增的日期是某個月的最後一天，則產生的月份總是產生月份的最後一天，不論該月份的長短。例如，4 月 30 號 + 1 個月是 5 月 31 號。

  ```
  select add_months('2008-04-30',1);

  add_months
  ---------------------
  2008-05-31 00:00:00
  (1 row)
  ```

- DATEADD：如果您要新增之日期中的天數少於結果月份，則結果將會是結果月份的相對應天數，而不會是該月的最後一天。例如，4 月 30 號 + 1 個月是 5 月 30 號。

  ```
  select dateadd(month,1,'2008-04-30');

  date_add
  ---------------------
  2008-05-30 00:00:00
  (1 row)
  ```

DATEADD 函數使用 dateadd (月份，12...) 或 dateadd (年份，1...) 時，處理閏年日期 02-29 的方式會不同。

```
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 函數

DATEDIFF 傳回兩個日期或時間表達式之日期部分的差異。

語法

```
DATEDIFF ( datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp} )
```

引數

datepart

函數運作的日期或時間值 (年、月或日、小時、分鐘、秒、毫秒或微秒) 的特定部分。如需詳細資訊，請參閱 日期或時間戳記函數的日期部分。

特別是，DATEDIFF 會決定兩個運算式間相距的日期部分邊界數。例如，假設您正在計算兩個日期 12-31-2008 和 01-01-2009 之間的年度差異。在這種情況下，函數傳回 1 年，儘管這些日期只相隔一天。如果您正在尋找兩個時間戳記間時數的差異，01-01-2009 8:30:00 和 01-01-2009 10:00:00，結果為 2 小時。如果您正在尋找兩個時間戳記間時數的差異，8:30:00 和 10:00:00，結果為 2 小時。

date|time|timetz|timestamp

DATE、TIME、TIMETZ 或 TIMESTAMP 欄，或隱含轉換為 DATE、TIME、TIMETZ 或 TIMESTAMP 的運算式。運算式必須同時包含指定的日期部分或時間部分。如果第二個日期或時間晚於第一個日期或時間，結果為正值。如果第二個日期或時間早於第一個日期或時間，結果為負值。

傳回類型

BIGINT

具有 DATE 欄的範例

下列範例會尋找在兩個常值日期值間週次的差異。

```
select datediff(week,'2009-01-01','2009-12-31') as numweeks;

numweeks
----------
52
(1 row)
```

下列範例會尋找兩個常值日期值之間的差異 (以小時為單位)。如果您未提供日期的時間值，則預設值為 00:00:00。

```
select datediff(hour, '2023-01-01', '2023-01-03 05:04:03');

date_diff
----------
53
(1 row)
```

下列範例會尋找兩個常值 TIMESTAMETZ 值之間的差異 (以天為單位)。

```
Select datediff(days, 'Jun 1,2008  09:59:59 EST', 'Jul 4,2008  09:59:59 EST')

date_diff
----------
33
```

以下範例找出資料表中同一列兩個日期之間的差異 (以天為單位)。

```
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)
```

下列範例會尋找在過去和今日日期中常值間季次的差異。此範例假設目前日期為 2008 年 6 月 5 日。您可以用全名或縮寫來表示日期部分。DATEDIFF 函數的預設欄名稱為 DATE_DIFF。

```
select datediff(qtr, '1998-07-01', current_date);

date_diff
-----------
40
(1 row)
```

下列範例會聯結 SALES 和 LISTING 資歷表，來計算在他們列出和門票售出後所經的天數：清單 1000 到 1005。這些清單銷售的最長等待時間是 15 天，而最短時間是不到一天 (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)
```

此範例會計算賣家等待所有門票特價的平均小時數。

```
select avg(datediff(hours, listtime, saletime)) as avgwait
from sales, listing
where sales.listid = listing.listid;

avgwait
```

```
---------
465
(1 row)
```

具有 TIME 欄的範例

下列範例資料表 TIME_TEST 有一個 TIME_VAL 欄 (類型為 TIME)，其中插入了三個值。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

下列範例會尋找 TIME_VAL 欄與時間常值之間的小時數差異。

```
select datediff(hour, time_val, time '15:24:45') from time_test;

 date_diff
-----------
        -5
        15
        15
```

下列範例會尋找兩個常值時間值之間的分鐘數差異。

```
select datediff(minute, time '20:00:00', time '21:00:00') as nummins;

nummins
----------
60
```

具有 TIMTZ 欄的範例

下列範例資料表 TIMETZ_TEST 有一個 TIMETZ_VAL 欄 (類型為 TIMETZ)，其中插入了三個值。

```
select timetz_val from timetz_test;

timetz_val
```

```
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

下列範例會尋找 TIMETZ 常值與 timetz_val 之間的小時數差異。

```
select datediff(hours, timetz '20:00:00 PST', timetz_val) as numhours from timetz_test;

numhours
----------
0
-4
1
```

下列範例會尋找兩個常值 TIMTZ 值之間的小時數差異。

```
select datediff(hours, timetz '20:00:00 PST', timetz '00:58:00 EST') as numhours;

numhours
----------
1
```

DATE_PART 函數

DATE_PART 會從運算式擷取日期部分值。DATE_PART 是 PGDATE_PART 函數的同義詞。

語法

```
DATE_PART(datepart, {date|timestamp})
```

引數

datepart

函數所操作的日期值的特定部分 (例如年、月或日) 的識別碼常值或字串。如需詳細資訊，請參閱 日期或時間戳記函數的日期部分。

{date|timestamp}

日期欄、時間戳記欄，或會隱性轉換為日期或時間戳記的運算式。Date 或 timestamp 中的欄或運算式必須包含 datepart 中指定的日期部分。

傳回類型

DOUBLE

範例

DATE_PART 函數的預設欄名稱為 pgdate_part。

下列範例會從時間戳記常值尋找分鐘。

```
SELECT DATE_PART(minute, timestamp '20230104 04:05:06.789');

pgdate_part
-----------
          5
```

下列範例會從時間戳記常值中尋找週數。週數計算遵循 ISO 8601 標準。如需詳細資訊，請參閱 Wikipedia 中的 ISO 8601。

```
SELECT DATE_PART(week, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         18
```

下列範例會從時間戳記常值尋找月份中的日期。

```
SELECT DATE_PART(day, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
          2
```

下列範例會從時間戳記常值尋找星期幾。週數計算遵循 ISO 8601 標準。如需詳細資訊，請參閱 Wikipedia 中的 ISO 8601。

```
SELECT DATE_PART(dayofweek, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
          1
```

下列範例會從時間戳記常值中尋找世紀。世紀計算遵循 ISO 8601 標準。如需詳細資訊，請參閱 Wikipedia 中的 ISO 8601。

```
SELECT DATE_PART(century, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         21
```

下列範例會從時間戳記常值中尋找千年。千年計算遵循 ISO 8601 標準。如需詳細資訊，請參閱 Wikipedia 中的 ISO 8601。

```
SELECT DATE_PART(millennium, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         3
```

下列範例會從時間戳記常值中尋找微秒。微秒計算遵循 ISO 8601 標準。如需詳細資訊，請參閱 Wikipedia 中的 ISO 8601。

```
SELECT DATE_PART(microsecond, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
    789000
```

下列範例會從日期常值中尋找月份。

```
SELECT DATE_PART(month, date '20220502');

pgdate_part
-----------
         5
```

下列範例會將 DATE_PART 函數套用至資料表中的欄。

```
SELECT date_part(w, listtime) AS weeks, listtime
FROM listing
WHERE listid=10
```

```
weeks |      listtime
------+---------------------
 25   | 2008-06-17 09:44:54
(1 row)
```

您可以將日期部分以全名或縮寫表示，在此情況下，w 代表週。

週日期部分的天會傳回從 0–6 的整數，從星期日開始。使用 DATE_PART 含小數點 (DAYOFWEEK)，來檢視星期六的事件。

```
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_TRUNC 函數

DATE_TRUNC 函數會根據您指定的日期部分 (例如小時、天或月) 來截斷時間戳記運算式或常值。

語法

```
DATE_TRUNC('datepart', timestamp)
```

引數

datepart

時間戳記值截斷目標的日期部分。輸入 timestamp 被截斷為輸入 datepart 的精確度。例如，`month` 截斷為每個月的第一天。有效格式如下：

- 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

如需某些格式縮寫的相關資訊，請參閱 日期或時間戳記函數的日期部分

timestamp

時間戳記欄或運算式會隱性轉換為時間戳記。

傳回類型

TIMESTAMP

範例

將輸入時間戳記截斷為秒。

```
SELECT DATE_TRUNC('second', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:05:06
```

將輸入時間戳記截斷為分鐘。

```
SELECT DATE_TRUNC('minute', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:05:00
```

將輸入時間戳記截斷為小時。

```
SELECT DATE_TRUNC('hour', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:00:00
```

將輸入時間戳記截斷為天。

```
SELECT DATE_TRUNC('day', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 00:00:00
```

將輸入時間戳記截斷為一個月的第一天。

```
SELECT DATE_TRUNC('month', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

將輸入時間戳記截斷為季度的第一天。

```
SELECT DATE_TRUNC('quarter', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

將輸入時間戳記截斷為一年的第一天。

```
SELECT DATE_TRUNC('year', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-01-01 00:00:00
```

將輸入時間戳記截斷為世紀的第一天。

```
SELECT DATE_TRUNC('millennium', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2001-01-01 00:00:00
```

將輸入時間戳記截斷為一週的星期一。

```
select date_trunc('week', TIMESTAMP '20220430 04:05:06.789');
date_trunc
2022-04-25 00:00:00
```

在下列範例中，DATE_TRUNC 函數會使用「週」日期部分來傳回每週星期一的日期。

```
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 函數

EXTRACT 函數從 TIMESTAMP、TIMESTAMPTZ、TIME 或 TIMETZ 值傳回日期或時間部分。範例包括時間戳記中的日、月、年、時、分、秒、毫秒或微秒。

語法

```
EXTRACT(datepart FROM source)
```

引數

datepart

要擷取的日期或時間的分欄，例如日、月、年、小時、分鐘、秒、毫秒或微秒。對於可能的值，請參閱 日期或時間戳記函數的日期部分。

source

評估為 TIMESTAMP、TIMESTAMPTZ、TIME 或 TIMETZ 資料類型的欄或運算式。

傳回類型

如果 source 值的計算結果為資料類型 TIMESTAMP、TIME 或 TIMETZ，則為 INTEGER。

如果 source 值計算為資料類型 TIMESTAMPTZ，則為 DOUBLE PRECISION。

TIMESTAMP 範例

下列範例會判斷特價時，售價是 10,000 USD 或更多的週次。

```
select salesid, extract(week from saletime) as weeknum
from sales
where pricepaid > 9999
order by 2;

salesid | weeknum
--------+---------
 159073 |       6
 160318 |       8
 161723 |      26
```

下列範例會從常值 timestamp 值傳回分鐘值。

```
select extract(minute from timestamp '2009-09-09 12:08:43');

date_part
--
```

下列範例會從常值 timestamp 值傳回毫秒值。

```
select extract(ms from timestamp '2009-09-09 12:08:43.101');

date_part
-----------
101
```

TIMESTAMPTZ 範例

下列範例會從常值 timestamptz 傳回年份值。

```
select extract(year from timestamptz '1.12.1997 07:37:16.00 PST');

date_part
-----------
1997
```

TIME 範例

下列範例資料表 TIME_TEST 有一個 TIME_VAL 欄 (類型為 TIME)，其中插入了三個值。

```
select time_val from time_test;
```

```
time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

下列範例會擷取每個 time_val 的分鐘。

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

下列範例會擷取每個 time_val 中的小時數。

```
select extract(hour from time_val) as hours from time_test;

hours
-----------
         20
         0
         0
```

下列範例會從常值值擷取毫秒。

```
select extract(ms from time '18:25:33.123456');

 date_part
-----------
     123
```

TIMETZ 範例

下列範例資料表 TIMETZ_TEST 有一個 TIMETZ_VAL 欄 (類型為 TIMETZ)，其中插入了三個值。

```
select timetz_val from timetz_test;
```

```
timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

下列範例會擷取每個 timetz_val 的小時。

```
select extract(hour from timetz_val) as hours from time_test;

hours
-----------
         4
         0
         5
```

下列範例會從常值值擷取毫秒。在處理擷取之前，常值不會轉換為 UTC。

```
select extract(ms from timetz '18:25:33.123456 EST');

 date_part
-----------
     123
```

下列範例會從常值 timetz 值傳回 UTC 的時區偏移小時。

```
select extract(timezone_hour from timetz '1.12.1997 07:37:16.00 PDT');

date_part
-----------
-7
```

GETDATE 函數

該GETDATE函數返回當前會話時區的當前日期和時間（默認為 UTC）。

它會傳回目前陳述式的開始日期或時間，即使是在交易區塊中也一樣。

語法

```
GETDATE()
```

括號是必要的。

傳回類型

TIMESTAMP

範例

下列範例會使用GETDATE函數來傳回目前日期的完整時間戳記。

```
select getdate();
```

SYSDATE 函數

SYSDATE 傳回目前工作階段時區中的目前日期和時間 (預設為 UTC)。

> **Note**
>
> SYSDATE 傳回目前交易的日期和時間 (而不是目前陳述式的開始)。

語法

```
SYSDATE
```

此函數不需引數。

傳回類型

TIMESTAMP

範例

下列範例中會使用 SYSDATE 函數來傳回目前日期的完整時間戳記。

```
select sysdate;

timestamp
----------------------------
2008-12-04 16:10:43.976353
```

```
(1 row)
```

下列範例中會使用 TRUNC 函數中的 SYSDATE 函數來傳回不含時間的目前日期。

```
select trunc(sysdate);

trunc
------------
2008-12-04
(1 row)
```

在發出查詢且當日期早於 120 天時，下列查詢會傳回落於該日期間之日期的特價資訊。

```
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 函數

TIMEOFDAY 是特別的別名，會用來傳回週間日、日期和時間做為字串值。它會傳回目前陳述式的當日時間字串，即使是在交易區塊中也一樣。

語法

```
TIMEOFDAY()
```

傳回類型

VARCHAR

範例

下列範例會透過使用 TIMEOFDAY 函數來傳回目前日期和時間。

```
select timeofday();
timeofday
------------
Thu Sep 19 22:53:50.333525 2013 UTC
(1 row)
```

TO_TIMESTAMP 函數

TO_TIMESTAMP 會將 TIMESTAMP 字串轉換為 TIMESTAMPTZ。

語法

```
to_timestamp (timestamp, format)
```

```
to_timestamp (timestamp, format, is_strict)
```

引數

timestamp

字串，代表 format 指定的格式中的時間戳記值。如果此引數保留為空白，則時間戳記值預設為 `0001-01-01 00:00:00`。

format

字串常值，定義 timestamp 值的格式。不支援包含時區 (**TZ**、**tz** 或 **OF**) 的格式做為輸出。請參閱 日期時間格式字串 以取得有效的時間戳記格式。

is_strict

選用的 Boolean 值，指定如果輸入時間戳值超出範圍是否回傳錯誤。當 is_strict 設定為 TRUE 時，如果有超出範圍的值，就會傳回錯誤。當 is_strict 設定為 FALSE (預設值) 時，會接受溢位值。

傳回類型

TIMESTAMPTZ

範例

以下範例示範如何使用 TO_TIMESTAMP 函數，將 TIMESTAMP 字串轉換成 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
```

可以傳遞日期的 TO_TIMESTAMP 部分。其餘日期部分設定為預設值。時間包含在輸出中：

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp
--------------------------
2017-01-01 00:00:00+00
```

以下 SQL 陳述式將字串 '2011-12-18 24:38:15' 轉換為 TIMESTAMPTZ。結果是 TIMESTAMPTZ 落在第二天，因為小時數超過 24 小時：

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS');

to_timestamp
----------------------
2011-12-19 00:38:15+00
```

下列 SQL 陳述式會將字串「2011-12-18 24:38:15」轉換為 TIMESTAMPTZ。結果會產生錯誤，因為時間戳記中的時間值超過 24 小時：

```
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
```

日期或時間戳記函數的日期部分

下表識別日期部分和時間部分名稱和縮寫，系統接受它們做為以下函數的引數：

- DATEADD
- DATEDIFF
- DATE_PART
- EXTRACT

日期部分或時間部分	縮寫
millennium, millennia	mil, mils
century, centuries	c, cent, cents
decade, decades	dec, decs
epoch	epoch (由 EXTRACT 支援)
year, years	y, yr, yrs
quarter, quarters	qtr, qtrs
month, months	mon, mons
week, weeks	w
週中的日	dayofweek, dow, dw, weekday (由 DATE_PART 和 EXTRACT 函數 所支援) 傳回從 0–6 的整數，從星期日開始。 **Note** DOW 日期部分與週中的日 (日期部分用來表示日期時間格是字串) 表現不同 (D)。D 是基於整數 1–7，其中星期日為 1。如需詳細資訊，請參閱 日期時間格式字串。
年中的日	dayofyear, doy, dy, yearday (由 EXTRACT 支援)
day, days	d
hour, hours	h, hr, hrs
minute, minutes	m, min, mins
second, seconds	s, sec, secs

日期部分或時間部分	縮寫
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 函數僅針對含時區 (TIMESTAMPTZ) 的時間戳記所支援。

含秒、毫秒和微秒結果的差異

當不同日期函數指定秒、毫秒或微秒做為日期部分時，減去查詢結果的差異：

- EXTRACT 函數會傳回整數僅做為指定日期部分，而忽略更高或更低層級的日期部分。若指定的日期部分為秒，則毫秒或微秒不會包含在結果中。若指定的日期部分為豪秒，則秒或微秒不會包含在內。若指定的日期部分為微秒，則秒或豪秒不會包含在內。
- DATE_PART 函數會傳回時間戳記的完整秒部分，不論指定的日期部分為何，視需要傳回十進位值或整數。

CENTURY、EPOCH、DECADE 和 MIL 備註

CENTURY 或 CENTURIES

AWS Clean Rooms 解釋一個世紀以年 ## #1 開始，並以年###0結束：

```
select extract (century from timestamp '2000-12-16 12:21:13');
date_part
-----------
20
(1 row)

select extract (century from timestamp '2001-12-16 12:21:13');
date_part
-----------
21
(1 row)
```

EPOCH

EPOCH 的 AWS Clean Rooms 實作相對於 1970-01-01 00:00:00.000 與叢集所在的時區無關。您可能想要根據叢集所在的時區，彌補結果的時數差異。

DECADE 或 DECADES

AWS Clean Rooms 根據公共日曆解釋十年或幾十年 DATPART。例如，因為一般日曆是從年份 1 開始，第一個十年 (十年 1) 為 0001-01-01 至 0009-12-31，而第二個十年 (十年 2) 為 0010-01-01 到 0019-12-31。例如，十年 201 橫跨 2000-01-01 到 2009-12-31：

```
select extract(decade from timestamp '1999-02-16 20:38:40');
date_part
-----------
200
(1 row)

select extract(decade from timestamp '2000-02-16 20:38:40');
date_part
-----------
201
(1 row)

select extract(decade from timestamp '2010-02-16 20:38:40');
date_part
-----------
202
(1 row)
```

MIL 或 MILS

AWS Clean Rooms 解釋 MIL 從一年的第一天開始 #001 並結束一年的最後一天:#000

```
select extract (mil from timestamp '2000-12-16 12:21:13');
date_part
-----------
2
(1 row)

select extract (mil from timestamp '2001-12-16 12:21:13');
date_part
-----------
3
```

```
(1 row)
```

雜湊函數

哈希函數是一個數學函數，將數字輸入值轉換為另一個值。 AWS Clean Rooms支持以下哈希函數：

主題

- MD5 函數
- SHA 函數
- SHA1 函數
- SHA2 函數
- 雜音

MD5 函數

使用 MD5 加密雜湊函數，將可變長度字串轉換為 32 的字元的字串，此為 128 位元檢查總和之十六進位值的文字表示法。

語法

```
MD5(string)
```

引數

string

可變長度字串。

傳回類型

MD5 函數傳回 32 個字元的字串，此為 128 位元檢查總和之十六進位值的文字表示法。

範例

下列範例顯示字串 'AWS Clean Rooms' 的 128 位元值：

```
select md5('AWS Clean Rooms');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

SHA 函數

SHA1 函數的同義詞。

請參閱 SHA1 函數。

SHA1 函數

SHA1 函數使用 SHA1 加密雜湊函數，將可變長度字串轉換為 40 之字元的字串，即 160 位元檢查總和之十六進位值的文字表示法。

語法

SHA1 是的同義詞。SHA 函數

```
SHA1(string)
```

引數

string

可變長度字串。

傳回類型

SHA1 函數傳回 40 個字元的字串，此為 160 位元檢查總和之十六進位值的文字表示法。

範例

下列範例傳回單字 'AWS Clean Rooms' 的 160 位元值：

```
select sha1('AWS Clean Rooms');
```

SHA2 函數

SHA2 函數使用 SHA2 加密雜湊函數，將可變長度字串轉換為一個字元的字串。字串是檢查總和之十六進位值的文字表示法，具有指定的位元數。

語法

```
SHA2(string, bits)
```

引數

string

可變長度字串。

integer

雜湊函數中的位元數。有效值為 0 (與 256 相同)、224、256、384 和 512。

傳回類型

SHA2 函數傳回一個字元的字串，它是檢查總和之十六進位值的文字表示法，或是空字串 (如果位元數無效)。

範例

下列範例傳回單字 'AWS Clean Rooms' 的 256 位元值：

```
select sha2('AWS Clean Rooms', 256);
```

雜音

該 MURMUR3_32_HASH 函數計算所有常見的數據類型，包括數字和字符串類型的 32 位 Murmur3a 非加密散列。

語法

```
MURMUR3_32_HASH(value [, seed])
```

引數

值

輸入值散列。 AWS Clean Rooms散列輸入值的二進制表示。此行為類似於 FNV_HASH，但該值會轉換為 Apache 冰山 32 位元雜湊規範所指定的二進位表示法。

seed

雜湊函數的 INT 種子。此為選用引數。如果未指定，則AWS Clean Rooms使用預設種子 0。這會允許組合多個欄位的雜湊，而無須進行轉換或串連。

傳回類型

該函數返回一個 INT。

範例

下列範例會傳回數字的 Murmur3 雜湊值、字串 'AWS Clean Rooms'，以及兩者的串連。

```
select MURMUR3_32_HASH(1);

    MURMUR3_32_HASH
----------------------
 -5968735742475085980
(1 row)
```

```
select MURMUR3_32_HASH('AWS Clean Rooms');

    MURMUR3_32_HASH
----------------------
 7783490368944507294
(1 row)
```

```
select MURMUR3_32_HASH('AWS Clean Rooms', MURMUR3_32_HASH(1));

    MURMUR3_32_HASH
----------------------
 -2202602717770968555
(1 row)
```

使用須知

要計算具有多列的表的哈希值，可以計算第一列的 Murmur3 哈希值，並將其作為種子傳遞給第二列的散列。然後，它將第二列的 Murmur3 哈希作為種子傳遞給第三列的散列。

以下範例會建立種子來雜湊包含多個欄位的資料表。

```
select MURMUR3_32_HASH(column_3, MURMUR3_32_HASH(column_2, MURMUR3_32_HASH(column_1)))
 from sample_table;
```

相同屬性可以用來運算字串串連的雜湊。

```
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)
```

雜湊函數會使用輸入的類型來判斷要雜湊的位元組數。如有必要，其會使用轉換來強制使用特定類型。

下列範例使用不同的輸入類型來產生不同的結果。

```
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)
```

JSON 函數

當您需要存放相當小的一組金鑰值對時，您可以用 JSON 格式存放資料以節省空間。因為 JSON 字串可存放於單一欄，採用 JSON 可能比以資料表格式存放資料更有效率。

Example

例如，假設您有一個稀疏資料表，其中您需要有許多資料行才能完整呈現所有可能的屬性。但是，對於任何給定的行或任何給定的列，大多數列值都是 NULL。透過使用 JSON 進行儲存，您可以將索引鍵值配對中的資料儲存在單一 JSON 字串中，並排除稀少填入的資料表資料行。

此外，您可以輕鬆修改 JSON 字串來存放其他金鑰:值對，而不需要在資料表中新增欄。

建議少用 JSON。JSON 不是存儲較大數據集的好選擇，因為通過在單個列中存儲不同的數據，JSON 不使用AWS Clean Rooms列存儲架構。

JSON 使用 UTF-8 編碼的文字字串，所以 JSON 字串可儲存為 CHAR 或 VARCHAR 資料類型。如果字串包含多位元組字元，請使用 VARCHAR。

JSON 字串必須是符合下列規則的適當格式化 JSON：

- 根層級 JSON 可以是 JSON 物件或 JSON 陣列。JSON 物件是一組未排序的逗號分隔金鑰:值對 (以大括號括住)。

  例如：`{"one":1, "two":2}`
- JSON 陣列是一組已排序的逗號分隔值 (以方括號括住)。

  以下是範例：`["first", {"one":1}, "second", 3, null]`

- JSON 陣列採用以零開始的索引，陣列的第一個元素在位置 0。在 JSON 鍵：值對中，鍵是雙引號中的字符串。
- JSON 值可以是下列任何一種：
  - JSON 物件
  - JSON 陣列
  - 雙引號中的字符串
  - 數字 (整數和浮點數)
  - Boolean
  - Null
- 空物件和空陣列是有效的 JSON 值。
- JSON 欄位區分大小寫。
- 忽略 JSON 結構元素之間的空格 (例如 `{ }, [ ]`)。

AWS Clean Rooms JSON 函數和 AWS Clean Rooms COPY 命令使用相同方法來處理 JSON 格式的資料。

主題

- CAN_J-分析函數
- JSON_EXTRACT_ARRAY_ELEMENT_TEXT 函數
- JSON_EXTRACT_PATH_TEXT 函數
- 分析函數
- 序列化函數
- JSON_ 序列化到 _ 瓦字節函數

CAN_J-分析函數

該 CAN_JSON_PARSE 函數解析 JSON 格式的數據，`true`如果結果可以使用 JSON_PARSE 函數轉換為值返回。SUPER

語法

```
CAN_JSON_PARSE(json_string)
```

引數

json_string

傳回VARBYTE或VARCHAR表單中序列化 JSON 的運算式。

傳回類型

BOOLEAN

範例

若要查看 JSON 陣列是否[10001,10002,"abc"]可以轉換成資SUPER料類型，請使用下列範例。

```
SELECT CAN_JSON_PARSE('[10001,10002,"abc"]');

+----------------+
| can_json_parse |
+----------------+
| true           |
+----------------+
```

JSON_EXTRACT_ARRAY_ELEMENT_TEXT 函數

該 JSON_EXTRET_ARRAY_TEXT 函數返回一個 JSON 字符串的最外層數組中的 JSON 數組元素，使用從零開始的索引。陣列的第一個元素在位置 0。如果索引是負數或超出邊界，JSON_EXTRACT_ARRAY_ELEMENT_TEXT 會傳回空字串。如果 null_if_invalid 引數設為 `true`，且 JSON 字串無效，此函數會傳回 Null，而非傳回錯誤。

如需詳細資訊，請參閱 JSON 函數。

語法

```
json_extract_array_element_text('json string', pos [, null_if_invalid ] )
```

引數

json_string

格式正確的 JSON 字串。

pos

整數，代表要傳回之陣列元素的索引 (採用以零開始的陣列索引)。

null_if_invalid

布林值，指定輸入 JSON 字串無效時是否傳回 Null，而非傳回錯誤。若要在 JSON 無效時傳回 Null，請指定 `true` (`t`)。若要在 JSON 無效時傳回錯誤，請指定 `false` (`f`)。預設值為 `false`。

傳回類型

VARCHAR 字串，代表 pos 所參考的 JSON 陣列元素。

範例

下列範例傳回位置 2 的陣列元素，這是從零開始之陣列索引的第三個元素：

```
select json_extract_array_element_text('[111,112,113]', 2);

json_extract_array_element_text
-------------------------------
113
```

下列範例會傳回錯誤，因為 JSON 無效。

```
select json_extract_array_element_text('["a",["b",1,["c",2,3,null,]]]',1);

An error occurred when executing the SQL command:
select json_extract_array_element_text('["a",["b",1,["c",2,3,null,]]]',1)
```

下列範例將 null_if_invalid 設為 true，所以在 JSON 無效時，陳述式會傳回 Null，而非傳回錯誤。

```
select json_extract_array_element_text('["a",["b",1,["c",2,3,null,]]]',1,true);

json_extract_array_element_text
-------------------------------
```

JSON_EXTRACT_PATH_TEXT 函數

該 JSON_EXTRACT_PATH_TEXT 函數返回由一系列 JSON 字符串中的路徑元素引用的鍵：值對的值。JSON 路徑巢狀最深可達五層。路徑元素區分大小寫。如果 JSON 字串中沒有路徑元素，JSON_EXTRACT_PATH_TEXT 會傳回空字串。如果 `null_if_invalid` 引數設為 `true`，且 JSON 字串無效，此函數會傳回 Null，而非傳回錯誤。

如需有關其他 JSON 函數的資訊，請參閱JSON 函數。

語法

```
json_extract_path_text('json_string', 'path_elem' [,'path_elem'[, …] ]
 [, null_if_invalid ] )
```

引數

json_string

格式正確的 JSON 字串。

path_elem

JSON 字串中的路徑元素。需要一個路徑元素。可指定其他路徑元素，最深可達五層。

null_if_invalid

布林值，指定輸入 JSON 字串無效時是否傳回 Null，而非傳回錯誤。若要在 JSON 無效時傳回 Null，請指定 `true` (`t`)。若要在 JSON 無效時傳回錯誤，請指定 `false` (`f`)。預設值為 `false`。

在 JSON 字串中，AWS Clean Rooms 將 `\n` 視為新行字元，將 `\t` 視為 Tab 字元。若要載入反斜線，請加上反斜線 (`\\`) 來逸出。

傳回類型

VARCHAR 字串，代表路徑元素所參考的 JSON 值。

範例

下列範例會傳回路徑的值`'f4', 'f6'`。

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}','f4', 'f6');


json_extract_path_text
----------------------
star
```

下列範例會傳回錯誤，因為 JSON 無效。

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4', 'f6');

An error occurred when executing the SQL command:
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4', 'f6')
```

下列範例將 null_if_invalid 設為 true，所以在 JSON 無效時，陳述式會傳回 Null，而非傳回錯誤。

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4',
 'f6',true);

json_extract_path_text
------------------------------
NULL
```

下列範例會傳回路徑的值'farm', 'barn', 'color'，其中擷取的值位於第三層。此範例使用 JSON lint 工具進行格式化，以便於閱讀。

```
select json_extract_path_text('{
    "farm": {
        "barn": {
            "color": "red",
            "feed stocked": true
        }
    }
}', 'farm', 'barn', 'color');

json_extract_path_text
----------------------
red
```

下列範例會傳回 NULL，因為'color'元素遺失。此範例使用 JSON 棉絨工具進行格式化。

```
select json_extract_path_text('{
    "farm": {
        "barn": {}
    }
}', 'farm', 'barn', 'color');

json_extract_path_text
----------------------
NULL
```

如果 JSON 是有效的，嘗試提取缺少的元素返回 NULL。

下列範例會傳回路徑的值'house', 'appliances', 'washing machine', 'brand'。

```
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
```

分析函數

該 JSON_PARSE 函數解析 JSON 格式的數據，並將其轉換為超級表示。

若要擷取到使用插入或更新命令超級資料類型，請使用 JSON_PARSE 函式。當您使用 JSON_PARSE () 將 JSON 字串解析為超級值時，會套用某些限制。

語法

```
JSON_PARSE(json_string)
```

引數

json_string

傳回序列化 JSON 為可變位元組或 varchar 型別的運算式。

傳回類型

超級

範例

下面的例子是 JSON_PARSE 函數的一個例子。

```
SELECT JSON_PARSE('[10001,10002,"abc"]');
     json_parse
---------------------------
 [10001,10002,"abc"]
(1 row)
```

```
SELECT JSON_TYPEOF(JSON_PARSE('[10001,10002,"abc"]'));
 json_typeof
----------------
 array
(1 row)
```

序列化函數

該 JSON_SERIALIZE 函數序列化一個超級表達式成文本 JSON 表示遵循 RFC 8259。如需有關該 RFC 的詳細資訊，請參閱 JavaScript 物件標記法 (JSON) 資料交換格式。

超級大小限制與區塊限制大致相同，且 varchar 限制小於超級大小限制。因此，當 JSON 格式超過系統的 varchar 限制時，JSON_SERIALIZE 函數會傳回錯誤。

語法

```
JSON_SERIALIZE(super_expression)
```

引數

超級表達式

超級運算式或欄。

傳回類型

varchar

範例

下面的例子序列化一個 SUPER 值到一個字符串。

```
 SELECT JSON_SERIALIZE(JSON_PARSE('[10001,10002,"abc"]'));
   json_serialize
---------------------
 [10001,10002,"abc"]
(1 row)
```

JSON_ 序列化到 _ 瓦字節函數

該 JSON_SERIALIZE_TO_VARBYTE 函數將超級值轉換為類似於 JSON_SERIALIZE（ ）的 JSON 字符串，但存儲在一個 VARBYTE 值，而不是。

語法

```
JSON_SERIALIZE_TO_VARBYTE(super_expression)
```

引數

超級表達式

超級運算式或欄。

傳回類型

瓦字節

範例

下面的例子序列化一個 SUPER 值，並以 VARBYTE 格式返回結果。

```
SELECT JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'));
```

```
       json_serialize_to_varbyte
----------------------------------------
 5b31303030312c31303030322c22616263225d
```

下面的示例序列化一個 SUPER 值，並將結果轉換為 VARCHAR 格式。

```
SELECT JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'))::VARCHAR;
```

```
  json_serialize_to_varbyte
---------------------------
 [10001,10002,"abc"]
```

數學函數

本節描述 AWS Clean Rooms 中支援的數學運算子和函數。

主題

- COT 函數
- DEGREES 函數
- DEXP 函數
- DLOG1 函數
- DLOG10 函數
- EXP 函數
- FLOOR 函數
- LN 函數
- LOG 函數
- MOD 函數
- PI 函數
- POWER 函數
- RADIANS 函數
- RANDOM 函數
- ROUND 函數
- SIGN 函數
- SIN 函數
- SQRT 函數
- TRUNC 函數

數學運算子符號

下表列出支援的數學運算子。

支援的運算子

運算子	描述	範例	結果
+	加法	2 + 3	5
-	減法	2 - 3	-1
*	乘法	2 * 3	6

運算子	描述	範例	結果
/	除法	4 /2	2
%	模數	5 % 4	1
^	指數	2.0 ^ 3.0	8
|/	平方根	| / 25.0	5
||/	立方根	|| / 27.0	3
@	絕對值	@ -5.0	5

範例

計算支付的佣金加上給定交易的 $2.00 手續費：

```
select commission, (commission + 2.00) as comm
from sales where salesid=10000;

commission | comm
-----------+-------
28.05      | 30.05
(1 row)
```

針對給定交易計算售價的 20%：

```
select pricepaid, (pricepaid * .20) as twentypct
from sales where salesid=10000;

pricepaid | twentypct
----------+-----------
187.00    |    37.400
(1 row)
```

根據持續成長模式的預測門票銷售。在此範例中，子查詢傳回 2008 年銷售的門票數。在 10 年內，該結果將以 5% 的連續增長率成倍倍增長。

```
select (select sum(qtysold) from sales, date
```

```
where sales.dateid=date.dateid and year=2008)
^ ((5::float/100)*10) as qty10years;

qty10years
------------------
587.664019657491
(1 row)
```

找出日期 ID 大於或等於 2,000 的已付總價格和銷售佣金。然後從支付總價中減去佣金總計。

```
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
-----------+--------+----------+-----------
 364445.00 |   2044 | 54666.75 | 309778.25
 349344.00 |   2112 | 52401.60 | 296942.40
 343756.00 |   2124 | 51563.40 | 292192.60
 378595.00 |   2116 | 56789.25 | 321805.75
 328725.00 |   2080 | 49308.75 | 279416.25
 349554.00 |   2028 | 52433.10 | 297120.90
 249207.00 |   2164 | 37381.05 | 211825.95
 285202.00 |   2064 | 42780.30 | 242421.70
 320945.00 |   2012 | 48141.75 | 272803.25
 321096.00 |   2016 | 48164.40 | 272931.60
(10 rows)
```

ABS 函數

ABS 計算數字的絕對值，此數字可以是常值，或評估為數字的表達式。

語法

```
ABS (number)
```

引數

number

數字或評估為數字的表達式。它可以是小型，整數，大整數，十進制，FLOAT4 或 FLOAT8 類型。

傳回類型

ABS 傳回與其引數相同的資料類型。

範例

計算 -38 的絕對值：

```
select abs (-38);
abs
-------
38
(1 row)
```

計算 (14-76) 的絕對值：

```
select abs (14-76);
abs
-------
62
(1 row)
```

ACOS 函數

ACOS 是傳回數字反餘弦的三角函數。傳回值為弧度且介於 0 和 PI 之間。

語法

```
ACOS(number)
```

引數

number

輸入參數是DOUBLE PRECISION 數字。

傳回類型

DOUBLE PRECISION

範例

若要傳回 -1 的反餘弦，請使用下列範例。

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

ASIN 函數

ASIN 是傳回數字反正弦的三角函數。傳回值為弧度且介於 PI/2 和 -PI/2 之間。

語法

```
ASIN(number)
```

引數

number

輸入參數是DOUBLE PRECISION 數字。

傳回類型

DOUBLE PRECISION

範例

若要傳回 1 的正弦，請使用下列範例。

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
```

```
+--------------------+
```

ATAN 函數

ATAN 是傳回數字反正切的三角函數。傳回值為弧度且介於 -PI 和 PI 之間。

語法

```
ATAN(number)
```

引數

number

輸入參數是DOUBLE PRECISION 數字。

傳回類型

DOUBLE PRECISION

範例

若要傳回 1 的反正切再乘以 4，請使用下列範例。

```
SELECT ATAN(1) * 4 AS pi;

+--------------------+
|         pi         |
+--------------------+
| 3.141592653589793 |
+--------------------+
```

ATAN2 函數

ATAN2 是傳回兩個數字相除之反正切的三角函數。傳回值為弧度且介於 PI/2 和 -PI/2 之間。

語法

```
ATAN2(number1, number2)
```

引數

number1

DOUBLE PRECISION 數字。

number2

DOUBLE PRECISION 數字。

傳回類型

DOUBLE PRECISION

範例

若要傳回 1 的反正切再乘以 4，請使用下列範例。

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

CBRT 函數

CBRT 是計算數字立方根的數學函數。

語法

```
CBRT (number)
```

引數

CBRT 接受 DOUBLE PRECISION 數字做為引數。

傳回類型

CBRT 傳回 DOUBLE PRECISION 數字。

範例

計算給定交易之已付佣金的立方根：

```
select cbrt(commission) from sales where salesid=10000;

cbrt
------------------
3.03839539048843
(1 row)
```

CEILING (或 CEIL) 函數

CEILING 或 CEIL 函數用來將數字捨進到下一個整數。(FLOOR 函數 將數字捨去到下一個整數。)

語法

```
CEIL | CEILING(number)
```

引數

number

數字或評估為數字的運算式。它可以是小型，整數，大整數，十進制，FLOAT4 或 FLOAT8 類型。

傳回類型

CEILING 和 CEIL 傳回相同的資料類型作為它的引數。

範例

計算給定銷售交易之已付佣金的上限：

```
select ceiling(commission) from sales
where salesid=10000;

ceiling
---------
29
(1 row)
```

COS 函數

COS 是傳回數字餘弦的三角函數。傳回值為弧度且介於 -1 和 1 (含) 之間。

語法

```
COS(double_precision)
```

引數

number

輸入參數是雙精確度數字。

傳回類型

COS 函數傳回雙精確度數字。

範例

下列範例傳回 0 的餘弦：

```
select cos(0);
cos
-----
1
(1 row)
```

下列範例傳回 PI 的餘弦：

```
select cos(pi());
cos
-----
-1
(1 row)
```

COT 函數

COT 是傳回數字餘切的三角函數。輸入參數必須不是零。

語法

```
COT(number)
```

引數

number

輸入參數是DOUBLE PRECISION 數字。

傳回類型

DOUBLE PRECISION

範例

若要傳回 1 的餘切，請使用下列範例。

```
SELECT COT(1);

+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

DEGREES 函數

將角度的弧度轉換為其同等度數。

語法

```
DEGREES(number)
```

引數

number

輸入參數是DOUBLE PRECISION 數字。

傳回類型

DOUBLE PRECISION

範例

若要傳回 0.5 弧度的同等度數，請使用下列範例。

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
+-------------------+
```

若要將 PI 弧度轉換為度數，請使用下列範例。

```
SELECT DEGREES(pi());

+---------+
| degrees |
+---------+
|     180 |
+---------+
```

DEXP 函數

DEXP 函數以科學記號表示法傳回雙精確度數字的指數值。DEXP 和 EXP 函數的唯一差異是 DEXP 的參數必須為 DOUBLE PRECISION。

語法

```
DEXP(number)
```

引數

number

輸入參數是DOUBLE PRECISION 數字。

傳回類型

DOUBLE PRECISION

範例

```
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 函數

DLOG1 函數傳回輸入參數的自然對數。

DLOG1 函數是的一個同義詞。LN 函數

DLOG10 函數

DLOG10 傳回輸入參數的以 10 為底的對數。

DLOG10 函數是的一個同義詞。LOG 函數

語法

```
DLOG10(number)
```

引數

number

輸入參數是雙精確度數字。

傳回類型

DLOG10 函數傳回雙精確度數字。

範例

下列範例傳回數字 100 的以 10 為底的對數：

```
select dlog10(100);

dlog10
--------
2
(1 row)
```

EXP 函數

EXP 函數會實作數值表達式的指數函數，或是自然對數的底數，表達式的 e 次方。EXP 函數為 LN 函數 的倒數。

語法

```
EXP (expression)
```

引數

運算式

表達式必須為 INTEGER、DECIMAL 或 DOUBLE PRECISION 資料類型。

傳回類型

EXP 傳回 DOUBLE PRECISION 數字。

範例

使用 EXP 函數以根據持續成長模式來預測門票銷售。在此範例中，子查詢傳回 2008 年銷售的門票數。此結果乘以 EXP 函數的結果，而此函數指出過去 10 年的持續成長率為 7%。

```
select (select sum(qtysold) from sales, date
where sales.dateid=date.dateid
and year=2008) * exp((7::float/100)*10) qty2018;
```

```
qty2018
------------------
695447.483772222
(1 row)
```

FLOOR 函數

FLOOR 函數將數字捨去到下一個整數。

語法

```
FLOOR (number)
```

引數

number

數字或評估為數字的運算式。它可以是小型，整數，大整數，十進制，FLOAT4 或 FLOAT8 類型。

傳回類型

FLOOR 傳回與其引數相同的資料類型。

範例

該範例顯示使用 FLOOD 函數之前和之後，針對指定的銷售交易支付之佣金的數值。

```
select commission from sales
where salesid=10000;

floor
-------
28.05
(1 row)

select floor(commission) from sales
where salesid=10000;

floor
-------
28
```

```
(1 row)
```

LN 函數

LN 函數返回輸入參數的自然對數。

LN 函數是的同義詞。DLOG1 函數

語法

```
LN(expression)
```

引數

運算式

函數運算的目標欄或表達式。

> **Note**
>
> 如果運算式參考 AWS Clean Rooms 使用者建立的資料表或 AWS Clean Rooms STL 或 STV 系統資料表，此函數會傳回某些資料類型的錯誤。

如果具有下列資料類型的表達式參考使用者建立的資料表或系統資料表，則會產生錯誤。

- BOOLEAN
- CHAR
- DATE
- DECIMAL 或 NUMERIC
- TIMESTAMP
- VARCHAR

在使用者建立的資料表和 STL 或 STV 系統資料表上，具有下列資料類型的表達式可以成功執行：

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL

- SMALLINT

傳回類型

LN 函數傳回與表達式相同的類型。

範例

下列範例傳回數字 2.718281828 的自然對數，或以 e 為底的對數：

```
select ln(2.718281828);
ln
--------------------
0.9999999998311267
(1 row)
```

請注意，答案幾乎等於 1。

此範例傳回 USERS 資料表的 USERID 欄中一些值的自然對數：

```
select username, ln(userid) from users order by userid limit 10;

 username |        ln
----------+------------------
 JSG99FHE |                 0
 PGL08LJI | 0.693147180559945
 IFT66TXU |  1.09861228866811
 XDZ38RDD |  1.38629436111989
 AEB55QTM |   1.6094379124341
 NDQ15VBM |  1.79175946922805
 OWY35QYB |  1.94591014905531
 AZG78YIP |  2.07944154167984
 MSD36KVR |  2.19722457733622
 WKW41AIW |  2.30258509299405
(10 rows)
```

LOG 函數

傳回數字的以 10 為底的對數。

DLOG10 函數 的同義詞。

語法

```
LOG(number)
```

引數

number

輸入參數是雙精確度數字。

傳回類型

LOG 函數傳回雙精確度數字。

範例

下列範例傳回數字 100 的以 10 為底的對數：

```
select log(100);
dlog10
--------
2
(1 row)
```

MOD 函數

傳回兩個數字的餘數，也稱為模數運算。為了計算結果，第一個參數除以第二個參數。

語法

```
MOD(number1, number2)
```

引數

number1

第一個輸入參數是 INTEGER、SMALLINT、BIGINT 或 DECIMAL 數字。如果任一參數為 DECIMAL 類型，則另一個參數也必須為 DECIMAL 類型。如果任一參數為 INTEGER，則另一個參數可以是 INTEGER、SMALLINT 或 BIGINT。兩個參數也都可以是 SMALLINT 或 BIGINT，但如果一個參數是 BIGINT，則另一個參數不能是 SMALLINT。

number2

第二個參數是 INTEGER、SMALLINT、BIGINT 或 DECIMAL 數字。相同的資料類型規則適用於 number2 與 number1。

傳回類型

有效傳回值為 DECIMAL、INT、SMALLINT 及 BIGINT.。如果兩個輸入參數都是相同類型，則 MOD 函數的傳回類型與輸入參數的數值類型相同。不過，如果任一輸入參數為 INTEGER，則傳回類型也會是 INTEGER。

使用須知

您可以使用% 作為模運算子。

範例

下列範例會在數字除以另一個數字時傳回餘數：

```
SELECT MOD(10, 4);

 mod
------
 2
```

下列範例會傳回十進位結果：

```
SELECT MOD(10.5, 4);

 mod
------
 2.5
```

您可以轉換參數值：

```
SELECT MOD(CAST(16.4 as integer), 5);

 mod
------
 1
```

檢查第一個參數是否為偶數除以 2：

```
SELECT mod(5,2) = 0 as is_even;

 is_even
--------
 false
```

您可以使用% 作為模運算符：

```
SELECT 11 % 4 as remainder;

 remainder
-----------
 3
```

下列範例傳回 CATEGORY 資料表中奇數編號類別的資訊：

```
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

(6 rows)
```

PI 函數

PI 函數會傳回 pi 的值，精確到 14 位小數。

語法

```
PI()
```

傳回類型

DOUBLE PRECISION

範例

若要傳回 pi 的值，請使用下列範例。

```
SELECT PI();

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

POWER 函數

POWER 函數是將一個數值表達式乘上以第二個數值表達式為次方的指數函數。例如，2 的三次方以 POWER(2,3) 計算，結果為 8。

語法

```
{POW | POWER}(expression1, expression2)
```

引數

expression1

要乘以次方的數值表達式。必須是 INTEGER、DECIMAL 或 FLOAT 資料類型。

expression2

expression1 要乘以的次方。必須是 INTEGER、DECIMAL 或 FLOAT 資料類型。

傳回類型

DOUBLE PRECISION

範例

```
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 |
+-------------------+
```

RADIANS 函數

RADIANS 函數會將以度為單位的角度轉換為其同等弧度。

語法

```
RADIANS(number)
```

引數

number

輸入參數是DOUBLE PRECISION 數字。

傳回類型

DOUBLE PRECISION

範例

若要傳回 180 度的同等弧度，請使用下列範例。

```
SELECT RADIANS(180);

+-------------------+
|      radians      |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

RANDOM 函數

RANDOM 函數會產生介於 0.0 (包含) 到 1.0 (不包含) 間的隨機值。

語法

```
RANDOM()
```

傳回類型

RANDOM 傳回 DOUBLE PRECISION 數字。

範例

1. 計算介於 0 和 99 之間的隨機值。如果隨機數字為 0 到 1，此查詢會產生 0 到 100 的隨機數字：

   ```
   select cast (random() * 100 as int);

   INTEGER
   ------
   24
   (1 row)
   ```

2. 擷取 10 個商品的均勻隨機樣本：

   ```
   select *
   from sales
   order by random()
   limit 10;
   ```

   現在擷取 10 個商品的隨機樣本，但請依價格比例來選擇商品。例如，一個商品的價格如果是其他商品的兩倍，則出現在查詢結果的機率也是其他商品的兩倍：

   ```
   select *
   from sales
   order by log(1 - random()) / pricepaid
   limit 10;
   ```

3. 本範例使用 SET 命令來設定 SEED 值，以便 RANDOM 產生可預測的數字序列。

   首先傳回三個 RANDOM 整數，但不先設定 SEED 值：

```
select cast (random() * 100 as int);
INTEGER
------
6
(1 row)

select cast (random() * 100 as int);
INTEGER
------
68
(1 row)

select cast (random() * 100 as int);
INTEGER
------
56
(1 row)
```

現在，將 SEED 值設為 .25，並傳回三個以上的 RANDOM 數字：

```
set seed to .25;
select cast (random() * 100 as int);
INTEGER
------
21
(1 row)

select cast (random() * 100 as int);
INTEGER
------
79
(1 row)

select cast (random() * 100 as int);
INTEGER
------
12
(1 row)
```

最後，將 SEED 值重設為 .25，並驗證 RANDOM 是否傳回與前三個呼叫相同的結果：

```
set seed to .25;
select cast (random() * 100 as int);
INTEGER
------
21
(1 row)

select cast (random() * 100 as int);
INTEGER
------
79
(1 row)

select cast (random() * 100 as int);
INTEGER
------
12
(1 row)
```

ROUND 函數

ROUND 函數將數字四捨五入至最接近的整數或小數。

ROUND 函數可以選擇性地包含第二個參數作為整數，以指示在任一方向四捨五入的小數位數。當您未提供第二個引數時，函數會四捨五入為最接近的整數。當指定第二個引數 >n 時，函數會捨入為具有 n 個小數位數的最接近的數字。

語法

```
ROUND (number [ , integer ] )
```

引數

number

數字或評估為數字的運算式。它可以是十進制或 FLOAT8 類型。 AWS Clean Rooms 可以根據隱含轉換規則轉換其他數據類型。

整數 (選用)

整數；指出在任一方向上捨入的小數位數。

傳回類型

ROUND 傳回與輸入引數相同的數值資料類型。

範例

將給定交易的已付佣金四捨五入至最接近的整數。

```
select commission, round(commission)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    28
(1 row)
```

將給定交易的已付佣金四捨五入至第一位小數。

```
select commission, round(commission, 1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |  28.1
(1 row)
```

在相同的查詢中，反方向延伸精確度。

```
select commission, round(commission, -1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    30
(1 row)
```

SIGN 函數

SIGN 函數傳回數字的符號 (正或負)。SIGN 函數的結果為 1、-1 或 0，表示引數的符號。

語法

```
SIGN (number)
```

引數

number

數字或評估為數字的表達式。它可以是十進制或 FLOAT8 類型。 AWS Clean Rooms 可以根據隱含轉換規則轉換其他數據類型。

傳回類型

SIGN 會傳回與輸入引數相同的數值資料類型。如果輸入是十進制，則輸出為十進制（1,0）。

範例

若要確定從 SALES 表中為給定交易支付的佣金的符號，請使用以下範例。

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

SIN 函數

SIN 是傳回數字正弦的三角函數。傳回值介於 -1 和 1 之間。

語法

```
SIN(number)
```

引數

number

以弧度表示的 DOUBLE PRECISION 數字。

傳回類型

DOUBLE PRECISION

範例

若要傳回 -PI 的正弦，請使用下列範例。

```
SELECT SIN(-PI());

+-------------------------+
|           sin           |
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

SQRT 函數

SQRT 函數傳回數值的平方根。平方根是一個數字與其自身相乘以獲得給定值。

語法

```
SQRT (expression)
```

引數

運算式

表達式必須為整數、小數或浮點數資料類型。expression 可以包含函數。系統可能會執行隱含型別轉換。

傳回類型

SQRT 傳回 DOUBLE PRECISION 數字。

範例

下列範例會傳回數字的平方根。

```
select sqrt(16);
```

```
sqrt
---------------
4
```

下列範例會執行隱含型別轉換。

```
select sqrt('16');

sqrt
---------------
4
```

下列範例會嵌套函數以執行更複雜的工作。

```
select sqrt(round(16.4));

sqrt
---------------
4
```

下列範例會在指定圓的面積時產生半徑的長度。例如，當以平方英吋為單位指定面積時，它會以英吋為單位計算半徑。樣本中的面積為 20。

```
select sqrt(20/pi());
```

這會傳回值 5.046265044040321。

下列範例會從 SALES 資料表傳回傭金值的平方根。COMMISSION 欄是 DECIMAL 欄。此範例顯示如何在具有更複雜條件式邏輯的查詢中使用函數。

```
select sqrt(commission)
from sales where salesid < 10 order by salesid;

sqrt
------------------
10.4498803820905
3.37638860322683
7.24568837309472
5.1234753829798
```

```
...
```

下列查詢傳回同一組 COMMISSION 值的四捨五入平方根。

```
select salesid, commission, round(sqrt(commission))
from sales where salesid < 10 order by salesid;

salesid | commission | round
--------+------------+-------
      1 |     109.20 |    10
      2 |      11.40 |     3
      3 |      52.50 |     7
      4 |      26.25 |     5
...
```

如需中範例資料的詳細資訊 AWS Clean Rooms，請參閱範例資料庫。

TRUNC 函數

TRUNC 函數將數字截斷為先前的整數或小數。

TRUNC 函數可以選擇性地包含第二個參數作為整數，以指示在任一方向四捨五入的小數位數。當您未提供第二個引數時，函數會四捨五入為最接近的整數。當指定第二個引數 >n 時，函數會捨入為小數位數 >n 個小數位數的最接近的數字。該函數還截斷時間戳並返回一個日期。

語法

```
TRUNC (number [ , integer ] |
timestamp )
```

引數

number

數字或評估為數字的運算式。它可以是十進制或 FLOAT8 類型。 AWS Clean Rooms 可以根據隱含轉換規則轉換其他數據類型。

整數 (選用)

整數，表示精確度的小數位數 (任一方向)。如果未提供整數，數字會截斷為整數；如果指定整數，數字會截斷至指定的小數位數。

timestamp

該函數還可以從時間戳返回日期。（要返回時間戳值00:00:00作為時間，請將函數結果轉換為時間戳。）

傳回類型

TRUNC 會傳回與第一個輸入引數相同的資料類型。對於時間戳記，TRUNC 會傳回一個日期。

範例

截斷給定銷售交易的已付佣金。

```
select commission, trunc(commission)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   111

(1 row)
```

將同一個佣金值截斷至第一位小數。

```
select commission, trunc(commission,1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 | 111.1

(1 row)
```

以第二個引數的負值截斷佣金；111.15 捨去到 110。

```
select commission, trunc(commission,-1)
from sales where salesid=784;

commission | trunc
-----------+-------
```

```
    111.15 |   110
(1 row)
```

返回從 SYSDATE 函數（返回一個時間戳）的結果的日期部分：

```
select sysdate;

timestamp
----------------------------
2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc
------------
2011-07-21
(1 row)
```

要套用到 TIMESTAMP 欄位的 TRUNC 函數。傳回類型為日期。

```
select trunc(starttime) from event
order by eventid limit 1;

trunc
------------
2008-01-25
(1 row)
```

字串函數

主題

- || (串連) 運算子
- BTRIM 函數
- CHAR_LENGTH 函數
- CHARACTER_LENGTH 函數
- CHARINDEX 函數
- CONCAT 函數

- LEFT 和 RIGHT 函數
- LEN 函數
- LENGTH 函數
- LOWER 函數
- LPAD 和 RPAD 函數
- LTRIM 函數
- POSITION 函數
- REGEXP_COUNT 函數
- REGEXP_INSTR 函數
- REGEXP_REPLACE 函數
- REGEXP_SUBSTR 函數
- REPEAT 函數
- REPLACE 函數
- REPLICATE 函數
- REVERSE 函數
- RTRIM 函數
- SOUNDEX 函數
- SPLIT_PART 函數
- STRPOS 函數
- SUBSTR 函數
- SUBSTRING 函數
- TEXTLEN 函數
- TRANSLATE 函數
- TRIM 函數
- UPPER 函數

字串函數處理和操作字元字串，或評估為字元字串的表達式。當這些函數中的 string 引數是常值時，必須以單引號括住。支援的資料類型包括 CHAR 和 VARCHAR。

下節提供所支援函數的函數名稱、語法及描述。字串的所有偏移都是以一開始。

|| (串連) 運算子

串連 || 符號兩側的兩個運算式，並傳回串連後的運算式。

串連運算子類似於CONCAT 函數。

> **Note**
>
> 對於 CONCAT 函數和串連運算子，如果一個或兩個運算式為 Null，則串連的結果為 Null。

語法

```
expression1 || expression2
```

引數

expression1、expression2

兩個引數都可以是固定長度或可變長度的字元字串或表達式。

傳回類型

|| 運算子傳回字串。字串的類型與輸入引數相同。

範例

下列範例串連 USERS 資料表中的 FIRSTNAME 和 LASTNAME 欄位：

```
select firstname || ' ' || lastname
from users
order by 1
limit 10;

concat
-----------------
Aaron Banks
Aaron Booth
Aaron Browning
Aaron Burnett
```

```
Aaron Casey
Aaron Cash
Aaron Castro
Aaron Dickerson
Aaron Dixon
Aaron Dotson
(10 rows)
```

若要串連可能包含 Null 的欄，請使用 NVL 和 COALESCE 函數表達式。下列範例使用 NVL，只要遇到 NULL 就傳回 0。

```
select venuename || ' seats ' || nvl(venueseats, 0)
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
```

BTRIM 函數

BTRIM 函數修剪字串，包括移除開頭和結尾空格，或移除符合選用指定字串的開頭和結尾字元。

語法

```
BTRIM(string [, trim_chars ] )
```

引數

string

要修剪的輸入 VARCHAR 字串。

trim_chars

包含要比對之字元的 VARCHAR 字串。

傳回類型

BTRIM 函數傳回 VARCHAR 字串。

範例

下列範例從字串 ' abc ' 中修剪開頭和結尾空格：

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

下列範例從字串 'xyzaxyzbxyzcxyz' 中移除開頭和結尾 'xyz' 字串。開頭和結尾的 'xyz' 已移除，但出現在字串內的部分則未移除。

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

下列範例會從符合 trim _chars 清單 'tes' 中任何字元的字串 'setuphistorycassettes' 中移除開頭和結尾部分。任何出現在輸入字串開頭或結尾的 trim_chars 清單中另一個字元前的 t、e 或 s 都會被移除。

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
-----------------
 uphistoryca
```

CHAR_LENGTH 函數

LEN 函數的同義詞。

請參閱 LEN 函數。

CHARACTER_LENGTH 函數

LEN 函數的同義詞。

請參閱 LEN 函數。

CHARINDEX 函數

傳回指定子字串在一個字串內的位置。

如需相似函數，請參閱 POSITION 函數 和 STRPOS 函數。

語法

```
CHARINDEX( substring, string )
```

引數

substring

在 string 內要搜尋的子字串。

string

要搜尋的字串或欄。

傳回類型

CHARINDEX 函數傳回對應於子字串位置的整數 (以 1 開始，不是以零開始)。位置以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。

使用須知

如果在 `string` 內找不到子字串，CHARINDEX 會傳回 0：

```
select charindex('dog', 'fish');
```

```
charindex
----------
0
(1 row)
```

範例

下列範例顯示字串 fish 在單字 dogfish 內的位置：

```
select charindex('fish', 'dogfish');
 charindex
----------
         4
(1 row)
```

下列範例從 SALES 資料表中傳回 COMMISSION 超過 999.00 的銷售交易次數：

```
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
(1 row)
```

CONCAT 函數

CONCAT 函數會串連兩個運算式，並傳回產生的運算式。若要串連兩個以上的運算式，請使用巢狀 CONCAT 函數。兩個運算式之間的串連運算子 (||) 產生與 CONCAT 函數相同的結果。

> **Note**
>
> 對於 CONCAT 函數和串連運算子，如果一個或兩個運算式為 Null，則串連的結果為 Null。

語法

```
CONCAT ( expression1, expression2 )
```

引數

expression1、expression2

這兩個引數都可以是固定長度的字元字串、可變長度字串、二進位運算式或計算結果為這些輸入之一的運算式。

傳回類型

CONCAT 傳回一個運算式。運算式的資料類型與輸入引數的類型相同。

如果輸入運算式的類型不同， AWS Clean Rooms 會嘗試隱含型別轉換其中一個運算式。如果無法轉換數值，系統會傳回一個錯誤。

範例

下列範例串連兩個字元常值：

```
select concat('December 25, ', '2008');

concat
-------------------
December 25, 2008
(1 row)
```

下列查詢 (使用 || 運算子，而不是 CONCAT) 產生相同的結果：

```
select 'December 25, '||'2008';

concat
-------------------
December 25, 2008
(1 row)
```

下列範例使用兩個 CONCAT 函數來串連三個字元字串：

```
select concat('Thursday, ', concat('December 25, ', '2008'));

concat
----------------------------
Thursday, December 25, 2008
```

```
(1 row)
```

若要串連可能包含 Null 的欄，請使用 NVL 和 COALESCE 函數。下列範例使用 NVL，只要遇到 NULL 就傳回 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)
```

下列查詢串連 VENUE 資料表中的 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)
```

下列查詢使用巢狀 CONCAT 函數。此查詢串連 VENUE 資料表中的 CITY 和 STATE 值，但以逗號和空格來分隔產生的字串：

```
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)
```

LEFT 和 RIGHT 函數

這些函數從字元字串最左邊或最右邊傳回指定的字元數。

數目以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。

語法

```
LEFT ( string,  integer )

RIGHT ( string,  integer )
```

引數

string

任何字元字串，或任何評估為字元字串的表達式。

integer

正整數。

傳回類型

LEFT 和 RIGHT 傳回 VARCHAR 字串。

範例

下列範例從 ID 介於 1000 和 1005 之間的活動名稱中，傳回最左邊 5 個和最右邊 5 個字元：

```
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
(6 rows)
```

LEN 函數

以字元數傳回指定字串的長度。

語法

LEN 是 LENGTH 函數、CHAR_LENGTH 函數、CHARACTER_LENGTH 函數 及 TEXTLEN 函數 的同義詞。

```
LEN(expression)
```

引數

運算式

輸入參數是 CHAR 或 VARCHAR 或有效輸入類型之一的別名。

傳回類型

LEN 函數傳回整數，表示輸入字串中的字元數。

如果輸入字串是字串，則 LEN 函數會傳回多位元組字串中的實際字元數，而不是位元組數。例如，需要 VARCHAR(12) 欄來存放三個四位元組中文字元。LEN 函數針對此相同字串傳回 3。

使用須知

長度計算不包括固定長度字元字串的結尾空格，但計算可變長度字串的結尾空格。

範例

下列範例傳回字串 français 中的位元組數和字元數。

```
select octet_length('français'),
len('français');

octet_length  | len
--------------+-----
           9  |   8
```

下列範例傳回無結尾空格的字串 cat 和有三個結尾空格的 cat 中的字元數：

```
select len('cat'), len('cat   ');
 len | len
-----+-----
   3 |   6
```

下列範例傳回 VENUE 資料表中前十個最長的 VENUENAME 項目：

```
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 函數

LEN 函數的同義詞。

請參閱 LEN 函數。

LOWER 函數

將字串轉換成小寫。LOWER 支援 UTF-8 多位元組字元，每個字元最多 4 個位元組。

語法

```
LOWER(string)
```

引數

string

輸入參數是一個 VARCHAR 字符串（或任何其他可以隱式轉換為 VARCHAR 的數據類型，例如 CHAR）。

傳回類型

LOWER 函數返回與輸入字符串相同的數據類型的字符串。

範例

下列範例將 CATNAME 欄位轉換為小寫：

```
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
(11 rows)
```

LPAD 和 RPAD 函數

這些函數根據指定的長度，將字元附加到字串的前面或後面。

語法

```
LPAD (string1, length, [ string2 ])
```

```
RPAD (string1, length, [ string2 ])
```

引數

string1

字元字串或評估為字元字串的表達式，例如字元欄的名稱。

長度

整數，定義函數結果的長度。字串長度以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。如果 string1 比指定的長度更長，則會截斷 (從右邊)。如果 length 是負數，則函數結果為空字串。

string2

附加到 string1 前面或後面的一或多個字元。此為選用引數；如果不指定，則使用空格。

傳回類型

這些函數傳回 VARCHAR 資料類型。

範例

將一組指定的活動名稱截斷至 20 個字元，並在較短名稱的前面附加空格：

```
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
(5 rows)
```

將同一組活動名稱截斷至 20 個字元，但在較短名稱的後面附加 0123456789。

```
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
(5 rows)
```

LTRIM 函數

從字串開頭修剪字元。刪除只包含在修剪字元清單中的字元的最長字串。當修剪字元沒有出現在輸入字串中時，就會完成修剪。

語法

```
LTRIM( string [, trim_chars] )
```

引數

string

要修剪的字串資料行、運算式或字串常值。

trim_chars

字串欄、運算式或字串常值，代表要從 string 開頭修剪的字元。如果未指定，則會使用空格作為修剪字元。

傳回類型

LTRIM 函數會傳回與輸入 string 的資料類型相同的字元字串 (CHAR 或 VARCHAR)。

範例

下列範例從 listime 欄中擷取年份。字串常值 '2008-' 中的修剪字元表示要從左側修剪的字元。如果使用修剪字元 '028-'，則可以得到相同的結果。

```
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
```

當 trim_chars 中任何字元出現在 string 開頭時，LTRIM 會移除這些字元。下列範例修剪 VENUENAME (這是 VARCHAR 欄) 開頭出現的 'C'、'D' 和 'G' 字元。

```
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
```

下列範例使用修剪字元 2，這是從 venueid 欄擷取的。

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

下列範例不會修剪任何字元，因為在 '0' 修剪字元之前找到 2。

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

下列範例會使用預設的空格修剪字元，並從字串的開頭修剪兩個空格。

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

POSITION 函數

傳回指定子字串在一個字串內的位置。

如需相似函數，請參閱 CHARINDEX 函數 和 STRPOS 函數。

語法

```
POSITION(substring IN string )
```

引數

substring

在 string 內要搜尋的子字串。

string

要搜尋的字串或欄。

傳回類型

POSITION 函數傳回對應於子字串位置的整數 (以 1 開始，不是以零開始)。位置以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。

使用須知

如果在字串內找不到子字串，POSITION 會傳回 0：

```
select position('dog' in 'fish');

position
----------
 0
(1 row)
```

範例

下列範例顯示字串 fish 在單字 dogfish 內的位置：

```
select position('fish' in 'dogfish');

position
----------
 4
(1 row)
```

下列範例從 SALES 資料表中傳回 COMMISSION 超過 999.00 的銷售交易次數：

```
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
(1 row)
```

REGEXP_COUNT 函數

在字串中搜尋規則表達式模式，並傳回整數指出此模式出現在字串中的次數。如果找不到相符項目，則函數會傳回 0。

語法

```
REGEXP_COUNT ( source_string, pattern [, position [, parameters ] ] )
```

引數

source_string

要搜尋的字串表達式，例如欄名。

pattern

代表規則運算式模式的字串常值。

position

正整數，表示在 source_string 內開始搜尋的位置。位置以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。預設為 1。如果 position 小於 1，則從 source_string 的第一個字元開始搜尋。如果 position 大於 source_string 中的字元數，則結果為 0。

參數

一或多個字串常值，表示函數如何比對模式。可能值如下：

- c - 進行區分大小寫比對。預設是使用區分大小寫比對。
- i - 進行不區分大小寫比對。
- p - 使用 Perl 相容規則運算式 (PCRE) 方言解釋此模式。

傳回類型

Integer

範例

下列範例計算三字母序列出現的次數。

```
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');
```

```
 regexp_count
 --------------
            8
```

下列範例計算頂層網域名稱為 org 或 edu 的次數。

```
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
```

下面的例子計算字符串的出現FOX，使用不區分大小寫的匹配。

```
SELECT regexp_count('the fox', 'FOX', 1, 'i');

 regexp_count
 --------------
            1
```

下列範例會使用 PCRE 方言撰寫的模式來尋找至少包含一個數字和一個小寫字母的字詞。它使用 ?= 運算子，該運算子在 PCRE 中具有特定的前瞻內涵。此範例會計算此類字詞的出現次數，並使用區分大小寫的比對。

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

 regexp_count
 --------------
            2
```

下列範例會使用 PCRE 方言撰寫的模式來尋找至少包含一個數字和一個小寫字母的字詞。它使用 ?= 運算子，該運算子在 PCRE 中具有特定的內涵。此範例會計算此類字詞的出現次數，但與前一個範例不同，因為它使用不區分大小寫的比對。

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');
```

```
regexp_count
--------------
           3
```

REGEXP_INSTR 函數

在字串中搜尋規則表達式模式，並傳回整數指出相符子字串的開始位置或結尾位置。如果找不到相符項目，則函數會傳回 0。REGEXP_INSTR 類似於 POSITION 函數，但可讓您在字串中搜尋規則表達式模式。

語法

```
REGEXP_INSTR ( source_string, pattern [, position [, occurrence] [, option
 [, parameters ] ] ] ] )
```

引數

source_string

要搜尋的字串表達式，例如欄名。

pattern

代表規則運算式模式的字串常值。

position

正整數，表示在 source_string 內開始搜尋的位置。位置以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。預設為 1。如果 position 小於 1，則從 source_string 的第一個字元開始搜尋。如果 position 大於 source_string 中的字元數，則結果為 0。

occurrence

正整數，表示要使用哪一個出現的模式。REGEXP_INSTR 略過前 occurrence -1 個相符項目。預設為 1。如果 occurrence 小於 1 或大於 source_string 中的字元數，則忽略搜尋，且結果為 0。

option

此值指出要傳回相符項目第一個字元的位置 (0)，還是相符項目後第一個字元的位置 (1)。非零值與 1 相同。預設值為 0。

參數

一或多個字串常值，表示函數如何比對模式。可能值如下：

- c - 進行區分大小寫比對。預設是使用區分大小寫比對。
- i - 進行不區分大小寫比對。
- e - 使用子運算式擷取子字串。

  如果 pattern 包含子表達式，REGEXP_INSTR 使用 pattern 中的第一個子表達式來比對子字串。REGEXP_INSTR 只考慮第一個子表達式；忽略其他子表達式。如果模式沒有子表達式，REGEXP_INSTR 會忽略 'e' 參數。
- p - 使用 Perl 相容規則運算式 (PCRE) 方言解釋此模式。

傳回類型

Integer

範例

下列範例搜尋網域名稱開頭的 @，並傳回第一個相符項目的開始位置。

```
SELECT email, regexp_instr(email, '@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

                     email                      | regexp_instr
------------------------------------------------+--------------
 Etiam.laoreet.libero@example.com |           21
 Suspendisse.tristique@nonnisiAenean.edu       |           22
 amet.faucibus.ut@condimentumegetvolutpat.ca   |           17
 sed@lacusUtnec.ca                             |            4
```

下列範例搜尋單字 Center 的變體，並傳回第一個相符項目的開始位置。

```
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
```

下列範例會使用不區分大小寫的相符邏輯FOX，尋找字串第一次出現的起始位置。

```
SELECT regexp_instr('the fox', 'FOX', 1, 1, 0, 'i');

 regexp_instr
 --------------
            5
```

下列範例會使用 PCRE 方言撰寫的模式來尋找至少包含一個數字和一個小寫字母的字詞。它使用 ?= 運算子，該運算子在 PCRE 中具有特定的前瞻內涵。此範例會尋找第二個此類字詞的開始位置。

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

 regexp_instr
 --------------
           21
```

下列範例會使用 PCRE 方言撰寫的模式來尋找至少包含一個數字和一個小寫字母的字詞。它使用 ?= 運算子，該運算子在 PCRE 中具有特定的前瞻內涵。此範例會尋找第二個這類字詞的開始位置，但與前一個範例不同，因為它使用不區分大小寫的比對。

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');

 regexp_instr
 --------------
           15
```

REGEXP_REPLACE 函數

在字串中搜尋規則表達式模式，並以指定的字串取代每一個出現的模式。REGEXP_REPLACE 類似於 REPLACE 函數，但可讓您在字串中搜尋規則表達式模式。

REGEXP_REPLACE 類似於 TRANSLATE 函數 和 REPLACE 函數，但 TRANSLATE 會進行多次單一字元替換，REPLACE 會將一整個字串替換成另一個字串，而 REGEXP_REPLACE 可讓您在字串中搜尋規則表達式模式。

語法

```
REGEXP_REPLACE ( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

引數

source_string

要搜尋的字串表達式，例如欄名。

pattern

代表規則運算式模式的字串常值。

replace_string

字串表達式 (例如欄名)，用於搜尋每一個出現的模式。預設為空字串 ("")。

position

正整數，表示在 source_string 內開始搜尋的位置。位置以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。預設為 1。如果 position 小於 1，則從 source_string 的第一個字元開始搜尋。如果 position 大於 source_string 中的字元數，則結果為 source_string。

參數

一或多個字串常值，表示函數如何比對模式。可能值如下：

- c - 進行區分大小寫比對。預設是使用區分大小寫比對。
- i - 進行不區分大小寫比對。
- p - 使用 Perl 相容規則運算式 (PCRE) 方言解釋此模式。

傳回類型

VARCHAR

如果 pattern 或 replace_string 為 NULL，則結果為 NULL。

範例

下列範例從電子郵件地中刪除 @ 和網域名稱。

```
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
```

下列範例會以此值取代電子郵件地址的網域名稱：internal.company.com。

```
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
```

下列範例會使用不區分大小寫的比對quick brown fox，取代值FOX內所有出現的字串。

```
SELECT regexp_replace('the fox', 'FOX', 'quick brown fox', 1, 'i');

   regexp_replace
--------------------
 the quick brown fox
```

下列範例會使用 PCRE 方言撰寫的模式來尋找至少包含一個數字和一個小寫字母的字詞。它使用 ?= 運算子，該運算子在 PCRE 中具有特定的前瞻內涵。此範例會以值取代此類字詞的每個出現次數[hidden]。

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');
```

```
       regexp_replace
------------------------------
 [hidden] plain A1234 [hidden]
```

下列範例會使用 PCRE 方言撰寫的模式來尋找至少包含一個數字和一個小寫字母的字詞。它使用 ?= 運算子，該運算子在 PCRE 中具有特定的前瞻內涵。這個範例會以值取代這類字詞的每一次出現[hidden]，但與前一個範例不同，因為它使用不區分大小寫的比對。

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

         regexp_replace
---------------------------------
 [hidden] plain [hidden] [hidden]
```

REGEXP_SUBSTR 函數

搜尋規則運算式模式，傳回字串中的字元。REGEXP_SUBSTR 類似於 SUBSTRING 函數 函數，但可讓您在字串中搜尋規則表達式模式。如果函數不能比對規則運算式與字串中的任何字元，則傳回一個空字串。

語法

```
REGEXP_SUBSTR ( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

引數

source_string

要搜尋的字串運算式。

pattern

代表規則運算式模式的字串常值。

position

正整數，表示在 source_string 內開始搜尋的位置。位置以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。預設為 1。如果 position 小於 1，則從 source_string 的第一個字元開始搜尋。如果 position 大於 source_string 中的字元數，則結果為空字串 ("")。

occurrence

正整數，表示要使用哪一個出現的模式。REGEXP_SUBSTR 略過前 occurrence -1 個相符項目。預設為 1。如果 occurrence 小於 1 或大於 source_string 中的字元數，則忽略搜尋，且結果為 NULL。

參數

一或多個字串常值，表示函數如何比對模式。可能值如下：

- c - 進行區分大小寫比對。預設是使用區分大小寫比對。
- i - 進行不區分大小寫比對。
- e - 使用子運算式擷取子字串。

  如果 pattern 包含子表達式，REGEXP_SUBSTR 使用 pattern 中的第一個子表達式來比對子字串。子運算式是用括號括起來的模式中的運算式。例如，對於模式 `'This is a '`，會比對第一個運算式和字串 `'This is a (\\w+)'` 後跟一個單詞。具有 e 參數的 REGEXP_SUBSTR 不會傳回 pattern，而是僅傳回子運算式內的字串。

  REGEXP_SUBSTR 只考慮第一個子表達式；忽略其他子表達式。如果模式沒有子表達式，REGEXP_SUBSTR 會忽略 'e' 參數。
- p - 使用 Perl 相容規則運算式 (PCRE) 方言解釋此模式。

傳回類型

VARCHAR

範例

下列範例傳回電子郵件地址在 @ 和網域域名之間的部分。

```
SELECT email, regexp_substr(email,'@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

                     email                      |      regexp_substr
------------------------------------------------+--------------------------
 Etiam.laoreet.libero@sodalesMaurisblandit.edu | @sodalesMaurisblandit
 Suspendisse.tristique@nonnisiAenean.edu       | @nonnisiAenean
 amet.faucibus.ut@condimentumegetvolutpat.ca   | @condimentumegetvolutpat
```

```
 sed@lacusUtnec.ca                                | @lacusUtnec
```

下列範例會使用不區分大小寫的相符FOX，傳回對應於字串第一次出現的輸入部分。

```
SELECT regexp_substr('the fox', 'FOX', 1, 1, 'i');

 regexp_substr
---------------
 fox
```

下列範例會傳回以小寫字母開頭之輸入的第一部分。這在函數上與沒有 c 參數的相同 SELECT 陳述式完全相同。

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');

 regexp_substr
---------------
 abc
```

下列範例會使用 PCRE 方言撰寫的模式來尋找至少包含一個數字和一個小寫字母的字詞。它使用 ?= 運算子，該運算子在 PCRE 中具有特定的前瞻內涵。此範例會傳回對應於第二個這類字詞的輸入部分。

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

下列範例會使用 PCRE 方言撰寫的模式來尋找至少包含一個數字和一個小寫字母的字詞。它使用 ?= 運算子，該運算子在 PCRE 中具有特定的前瞻內涵。此範例會傳回與第二個此類字詞對應的輸入部分，但與前一個範例不同，因為它使用不區分大小寫的比對。

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
---------------
```

```
 A1234
```

下列範例會使用子運算式，使用不區分大小寫的比對來尋找符合模式 'this is a (\\w+)' 的第二個字串。它傳回括號內的子運算式。

```
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 函數

將字串重複指定的次數。如果輸入參數是數值，REPEAT 會將輸入參數視為字串。

REPLICATE 函數 的同義詞。

語法

```
REPEAT(string, integer)
```

引數

string

第一個輸入參數是要重複的字串。

integer

第二個參數是整數，表示字串重複的次數。

傳回類型

REPEAT 函數傳回字串。

範例

下列範例將 CATEGORY 資料表中 CATID 欄的值重複三次：

```
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
(11 rows)
```

REPLACE 函數

以其他指定的字元取代一組字元在現有字串內出現的所有地方。

REPLACE 類似於 TRANSLATE 函數 和 REGEXP_REPLACE 函數，但 TRANSLATE 會進行多次單一字元替換，REGEXP_REPLACE 可讓您在字串中搜尋規則表達式模式，而 REPLACE 會將一整個字串替換成另一個字串。

語法

```
REPLACE(string1, old_chars, new_chars)
```

引數

string

要搜尋的 CHAR 或 VARCHAR 字串

old_chars

要取代的 CHAR 或 VARCHAR 字串。

new_chars

新的 CHAR 或 VARCHAR 字串，用來取代 old_string。

傳回類型

VARCHAR

如果 old_chars 或 new_chars 為 NULL，則結果為 NULL。

範例

下列範例將 CATGROUP 欄位中的字串 Shows 轉換為 Theatre：

```
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
(11 rows)
```

REPLICATE 函數

REPEAT 函數的同義詞。

請參閱 REPEAT 函數。

REVERSE 函數

REVERSE 函數操作字串並傳回相反順序的字元。例如，reverse('abcde') 傳回 edcba。此函數適用於數值和日期資料類型，以及字元資料類型；不過，在大部分情況下，字元字串有實用值。

語法

```
REVERSE ( expression )
```

引數

運算式

具有字元、日期、時間戳記或數值資料類型的運算式，代表字元反轉的目標。所有表達式都隱含地轉換為可變長度的字元字串。忽略固定長度字元字串中的結尾空格。

傳回類型

REVERSE 傳回 VARCHAR。

範例

從 USERS 資料表中選取五個不同城市名稱及其對應的反轉名稱：

```
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
(5 rows)
```

選取五個銷售 ID 及其對應的反轉 ID (轉換為字元字串)：

```
select salesid, reverse(salesid)::varchar
from sales order by salesid desc limit 5;

salesid | reverse
--------+---------
 172456 | 654271
 172455 | 554271
 172454 | 454271
```

```
 172453 | 354271
 172452 | 254271
(5 rows)
```

RTRIM 函數

RTRIM 函數從字串結尾修剪一組指定的字元。移除只包含修剪字元清單中字元的最長字串。當修剪字元沒有出現在輸入字串中時，就會完成修剪。

語法

```
RTRIM( string, trim_chars )
```

引數

string

要修剪的字串資料行、運算式或字串常值。

trim_chars

字串欄、運算式或字串常值，代表要從 string 結尾修剪的字元。如果未指定，則會使用空格作為修剪字元。

傳回類型

與 string 引數的資料類型相同的字串。

範例

下列範例從字串 ' abc ' 中修剪開頭和結尾空格：

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

下列範例從字串 'xyzaxyzbxyzcxyz' 中移除結尾 'xyz' 字串。結尾的 'xyz' 已移除，但出現在字串內的部分則未移除。

```
select 'xyzaxyzbxyzcxyz' as untrim,
```

```
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

下列範例會從符合 trim _chars 清單 'tes' 中任何字元的字串 'setuphistorycassettes' 中移除結尾部分。任何出現在輸入字串結尾的 trim_chars 清單中另一個字元前的 t、e 或 s 都會被移除。

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

下列範例修剪 VENUENAME 結尾出現的 'Park' 這幾個字元：

```
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
```

請注意，當 P、a、r 或 k 其中任何字元出現在 VENUENAME 的結尾時，RTRIM 會移除這些字元。

SOUNDEX 函數

SOUNDEX 函數會傳回由第一個字母後面接著代表您指定之字串英文發音的聲音的 3 位數編碼所組成的 American Soundex 值。

語法

```
SOUNDEX(string)
```

引數

string

您可以指定要轉換為美國聲音代碼值的 CHAR 或 VARCHAR 字符串。

傳回類型

SOUNDEX 函數返回一個 VARCHAR (4) 字符串由一個大寫字母後跟代表英語發音的聲音的三位數編碼。

使用須知

SOUNDEX 函數只會轉換英文字母小寫和大寫 ASCII 字元，包括 A-z 和 A-Z。SOUNDEX 會忽略其他字元。SOUNDEX 傳回由空格分隔的多個單詞的字串的單個 Soundex 值。

```
select soundex('AWS Amazon');
```

```
 soundex
---------
 A252
```

SOUNDEX 傳回一個空字串，如果輸入字串不包含任何英文字母。

```
select soundex('+-*/%');
```

```
  soundex
---------

```

範例

下面的示例返回的聲音 A525 這個詞 Amazon。

```
select soundex('Amazon');
```

```
 soundex
---------
 A525
```

SPLIT_PART 函數

在指定分隔符號之處分割字串，並傳回指定位置的部分。

語法

```
SPLIT_PART(string, delimiter, position)
```

引數

string

要分割的字串資料欄、運算式或字串常值。字串可以是 CHAR 或 VARCHAR。

delimiter

分隔符號字串，表示輸入 string 的部分。

如果 delimiter 是常值，請以單引號括住。

position

要傳回之字串部分的位置 (從 1 起算)。必須是大於 0 的整數。如果 position 大於字串部分的數目，SPLIT_PART 會傳回空字串。如果在 string 中找不到 delimiter，則傳回的值包含指定部分的內容，這可能是整個 string 或空值。

傳回類型

CHAR 或 VARCHAR 字串，與字串參數相同。

範例

下列範例會使用 $ 分隔符號將字串常值分割成多個部分，並傳回第二部分。

```
select split_part('abc$def$ghi','$',2)

split_part
----------
```

```
def
```

以下範例將使用 $ 分隔符號將字串常值分割成多個部分。它傳回一個空字串，因為沒有找到部分 4。

```
select split_part('abc$def$ghi','$',4)

split_part
----------
```

以下範例將使用 # 分隔符號將字串常值分割成多個部分。它傳回整個字串，這是第一部分，因為沒有找到分隔符號。

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

下列範例將時間戳記欄位 LISTTIME 分割成年、月、日元素。

```
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
```

下列範例選取 LISTTIME 時間戳記欄位，依 '-' 字元分割以取得月份 (LISTTIME 字串的第二部分)，然後計算每月的項目數：

```
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 函數

傳回子字串在指定字串內的位置。

如需相似函數，請參閱 CHARINDEX 函數 和 POSITION 函數。

語法

```
STRPOS(string, substring )
```

引數

string

第一個輸入參數是供進行搜尋的字串。

substring

第二個參數是在 string 內要搜尋的子字串。

傳回類型

STRPOS 函數傳回對應於子字串位置的整數 (以 1 開始，不是以零開始)。位置以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。

使用須知

如果在字符串中找不到子字符串，STRPOS 返回 0：

```
select strpos('dogfish', 'fist');
strpos
--------
0
(1 row)
```

範例

下列範例顯示字串 fish 在單字 dogfish 內的位置：

```
select strpos('dogfish', 'fish');
strpos
--------
4
(1 row)
```

下列範例從 SALES 資料表中傳回 COMMISSION 超過 999.00 的銷售交易次數：

```
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
(1 row)
```

SUBSTR 函數

SUBSTRING 函數的同義詞。

請參閱 SUBSTRING 函數。

SUBSTRING 函數

根據指定的開始位置傳回字串的子集。

如果輸入是字串，則提取的開始位置和字元數是以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。如果輸入是二進位運算式，則開始位置和提取的子字串是以位元組為基礎。您不能指定負長度，但可以指定負的開始位置。

語法

```
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 )
```

引數

character_string

供進行搜尋的字串。非字元資料類型視為字串。

start_position

在字串內要開始擷取的位置，從 1 開始。start_position 以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。這可以是負數。

number_characters

要擷取的字元數 (子字串的長度)。number_characters 以字元數為基礎，而不是位元組，所以多位元組字元視為單一字元。這不能是負數。

start_byte

在二進位運算式內要開始擷取的位置，從 1 開始。這可以是負數。

number_bytes

要擷取的位元組數，即子字串的長度。此數字不可以是負數。

傳回類型

VARCHAR

字元字串的使用注意事項

下列範例傳回從第六個字元開始的四個字元的字串。

```
select substring('caterpillar',6,4);
substring
-----------
pill
(1 row)
```

如果 start_position + number_characters 超過 string 的長度，SUBSTRING 會傳回從 start_position 開始到字串結尾的字串。例如：

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

如果 `start_position` 是負數或 0，SUBSTRING 函數會傳回從字串第一個字元開始且長度為 `start_position` + `number_characters` -1 的子字串。例如：

```
select substring('caterpillar',-2,6);
substring
-----------
cat
(1 row)
```

如果 `start_position` + `number_characters` -1 小於或等於零，SUBSTRING 會傳回空字串。例如：

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

範例

下列範例從 LISTING 資料表的 LISTTIME 字串中傳回月份：

```
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)
```

下列範例同上，但使用 FROM...FOR 選項：

```
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)
```

如果字串可能包含雙位元組字元，則您無法使用 SUBSTRING 來肯定地擷取字串的字首，因為您需要根據位元組數來指定雙位元組字串的長度，而不是字元數。若要根據位元組長度來擷取字串的開頭部分，您可以將字串 CAST 成為 VARCHAR(byte_length) 以截斷字串，其中 byte_length 是所需的長度。下列範例從 'Fourscore and seven' 字串中擷取前 5 個位元組。

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
Fours
```

下列範例會傳回輸入字串 Silva, Ana 中最後一個空格之後出現的名字 Ana。

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))

 reverse
-----------
 Ana
```

TEXTLEN 函數

LEN 函數的同義詞。

請參閱 LEN 函數。

TRANSLATE 函數

對於給定的表達式，以指定的替換值取代所有出現的指定字元。現有字元依位置映射至 characters_to_replace 和 characters_to_substitute 引數中的替換字元。如果 characters_to_replace 引數中指定的字元數比 characters_to_substitute 引數更多，傳回值中會省略 characters_to_replace 引數中額外的字元。

TRANSLATE 類似於 REPLACE 函數 和 REGEXP_REPLACE 函數，但 REPLACE 會將一整個字串替換成另一個字串，REGEXP_REPLACE 可讓您在字串中搜尋規則表達式模式，而 TRANSLATE 會進行多次單一字元替換。

如果任何引數為 Null，則傳回值為 NULL。

語法

```
TRANSLATE ( expression, characters_to_replace, characters_to_substitute )
```

引數

運算式

要轉換的表達式。

characters_to_replace

包含要取代之字元的字串。

characters_to_substitute

包含替換字元的字串。

傳回類型

VARCHAR

範例

下列範例取代字串中的幾個字元：

```
select translate('mint tea', 'inea', 'osin');

translate
-----------
most tin
```

下列範例對某欄的所有值，以點取代 at 符號 (@)：

```
select email, translate(email, '@', '.') as obfuscated_email
from users limit 10;

email                                           obfuscated_email
-------------------------------------------------------------------------------------------
Etiam.laoreet.libero@sodalesMaurisblandit.edu
 Etiam.laoreet.libero.sodalesMaurisblandit.edu
amet.faucibus.ut@condimentumegetvolutpat.ca
 amet.faucibus.ut.condimentumegetvolutpat.ca
```

```
turpis@accumsanlaoreet.org                turpis.accumsanlaoreet.org
ullamcorper.nisl@Cras.edu                ullamcorper.nisl.Cras.edu
arcu.Curabitur@senectusetnetus.com             arcu.Curabitur.senectusetnetus.com
ac@velit.ca                                 ac.velit.ca
Aliquam.vulputate.ullamcorper@amalesuada.org
 Aliquam.vulputate.ullamcorper.amalesuada.org
vel.est@velitegestas.edu                        vel.est.velitegestas.edu
dolor.nonummy@ipsumdolorsit.ca                  dolor.nonummy.ipsumdolorsit.ca
et@Nunclaoreet.ca                               et.Nunclaoreet.ca
```

下列範例對某欄的所有值，以點底線取代空格並剔除點：

```
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 函數

修剪字串，包括移除開頭和結尾空格，或移除符合選用指定字串的字元。

語法

```
TRIM( [ BOTH ] [ trim_chars FROM ] string
```

引數

trim_chars

(選用) 要從字串中修剪的字元。如果省略此參數，則會修剪空格。

string

要修剪的字串。

傳回類型

TRIM 函數傳回 VARCHAR 或 CHAR 字串。如果您將 TRIM 函數與 SQL 命令搭配使用，則 AWS Clean Rooms 會隱含地將結果轉換為 VARCHAR。如果您針對 SQL 函式使用 SELECT 清單中的 TRIM 函數，則 AWS Clean Rooms 不會隱含地轉換結果，而且您可能需要執行明確的轉換，以避免資料類型不符錯誤。如需明確轉換的相關資訊，請參閱 CAST 函數及 CONVERT 函數。

範例

下列範例從字串 ' abc ' 中修剪開頭和結尾空格：

```
select '     abc    ' as untrim, trim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

下列範例會移除字串周圍的雙引號"dog"：

```
select trim('"' FROM '"dog"');

btrim
-------
dog
```

TRIM 刪除任何在修剪字符的字符，當他們出現在字符串的開頭的字符。下列範例修剪 VENUENAME (這是 VARCHAR 欄) 開頭出現的 'C'、'D' 和 'G' 字元。

```
select venueid, venuename, trim(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
```

UPPER 函數

將字串轉換成大寫。UPPER 支援 UTF-8 多位元組字元，每個字元最多 4 個位元組。

語法

```
UPPER(string)
```

引數

string

輸入參數是一個 VARCHAR 字符串（或任何其他可以隱式轉換為 VARCHAR 的數據類型，例如 CHAR）。

傳回類型

UPPER 函數傳回與輸入字串的資料類型相同的字元字串。

範例

下列範例將 CATNAME 欄位轉換為大寫：

```
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
(11 rows)
```

超級類型信息函數

本節說明 SQL 的資訊函數，以從中支援的資SUPER料類型的輸入衍生動態資訊AWS Clean Rooms。

主題

- 十進制精度函數
- 十進制比例函數
- 陣列函數
- 大整特函數
- 字符函數
- 十進制函數
- 浮動函數
- 整數函數
- 物件函數
- 標量函數
- IS_ 斯密林特函數
- 瓦尔恰尔函数
- 函數類型

十進制精度函數

檢查要存儲的十進制位數的最大總數的精度。這個數字包括小數點的左右兩個數字。精確度的範圍是從 1 到 38，預設值為 38。

語法

```
DECIMAL_PRECISION(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

INTEGER

範例

若要將小數精確度函數套用至資料表 t，請使用下列範例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_PRECISION(s) FROM t;

+-------------------+
| decimal_precision |
+-------------------+
|                 6 |
+-------------------+
```

十進制比例函數

檢查要儲存在小數點右邊的小數位數。刻度的範圍是從 0 到精確度點，預設值為 0。

語法

```
DECIMAL_SCALE(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

INTEGER

範例

若要將十進制規模函數套用至資料表 t，請使用下列範例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_SCALE(s) FROM t;

+---------------+
| decimal_scale |
+---------------+
|             5 |
+---------------+
```

陣列函數

檢查一個變量是否是一個數組。true如果變量是一個數組，該函數返回。該函數還包括空數組。否則，函數會傳回false所有其他值，包括 null。

語法

```
IS_ARRAY(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

BOOLEAN

範例

若要檢查[1,2]是否是使用 IS_ARRAY 函數的陣列，請使用下列範例。

```
SELECT IS_ARRAY(JSON_PARSE('[1,2]'));

+----------+
| is_array |
+----------+
| true     |
+----------+
```

大整特函數

檢查一個值是否為BIGINT。該 IS_BIGINT 函數返回在 64 位範圍內的刻度true為 0 的數字。否則，函數會false傳回所有其他值，包括 null 和浮點數。

該 IS_BIGINT 函數是 IS_INTEGER 的超集。

語法

```
IS_BIGINT(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

BOOLEAN

範例

若要檢查5是否是BIGINT使用 IS_BIGINT 函數，請使用下列範例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_BIGINT(s) FROM t;

+---+-----------+
| s | is_bigint |
+---+-----------+
| 5 | true      |
+---+-----------+
```

字符函數

檢查一個值是否為CHAR。IS_CHAR 函數返回`true`只有 ASCII 字符的字符串，因為 CHAR 類型只能存儲在 ASCII 格式的字符串。該函數返回`false`任何其他值。

語法

```
IS_CHAR(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

BOOLEAN

範例

要檢查t是否是CHAR使用 IS_CHAR 函數，請使用下面的示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('t');

SELECT s, IS_CHAR(s) FROM t;

+-----+---------+
|  s  | is_char |
+-----+---------+
| "t" | true    |
+-----+---------+
```

十進制函數

檢查一個值是否為DECIMAL。該 IS_DECIMAL 函數返回true不是浮點數的數字。該函數返回false任何其他值，包括 null。

該小數函數是 IS_BIGINT 的超集。

語法

```
IS_DECIMAL(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

BOOLEAN

範例

要檢查1.22是否是DECIMAL使用 IS_DECIMAL 函數，請使用下面的示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (1.22);

SELECT s, IS_DECIMAL(s) FROM t;

+------+------------+
|  s   | is_decimal |
+------+------------+
| 1.22 | true       |
+------+------------+
```

浮動函數

檢查一個值是否是一個浮點數。該 IS_FLOAT 函數返回`true`浮點數（ FLOAT4和FLOAT8 ）。該函數返回`false`任何其他值。

IS_ 十進制集合 IS_FLOAT 的集合是不連接的。

語法

```
IS_FLOAT(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

BOOLEAN

範例

要檢查`2.22::`FLOAT是否是FLOAT使用 IS_FLOAT 函數，請使用下面的示例。

```
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     |
+---------+----------+
```

整數函數

傳回 true 32 位元範圍內刻度為 0 的數字，以及false其他任何項目 (包括 null 和浮點數)。

該整數函數是 IS_SMALLINT 函數的一個超集。

語法

```
IS_INTEGER(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

BOOLEAN

範例

要檢查5是否是INTEGER使用 IS_INTEGER 函數，請使用下面的示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_INTEGER(s) FROM t;
```

```
+---+------------+
| s | is_integer |
+---+------------+
| 5 | true       |
+---+------------+
```

物件函數

檢查一個變量是否是一個對象。該 IS_OBJECT 函數返回`true`對象，包括空對象。該函數返回`false`任何其他值，包括 null。

語法

```
IS_OBJECT(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

BOOLEAN

範例

若要檢查`{"name": "Joe"}`是否是使用 IS_OBJECT 函數的物件，請使用下列範例。

```
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      |
+----------------+-----------+
```

標量函數

檢查一個變量是否是一個標量。該 IS_SCALAR 函數返回不是一個數組或true對象的任何值。該函數返回false任何其他值，包括 null。

IS_ARRAY，IS_ 對象和 IS_純量的集合涵蓋除了空值以外的所有值。

語法

```
IS_SCALAR(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

BOOLEAN

範例

若要檢查{"name": "Joe"}是否是使用 IS_SCALAR 函數的純量，請使用下列範例。

```
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_ 斯密林特函數

檢查一個變量是否是一個SMALLINT。該 IS_SMALLINT 函數返回在 16 位範圍內的刻度true為 0 的數字。該函數返回false任何其他值，包括空和浮點數。

語法

```
IS_SMALLINT(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回

BOOLEAN

範例

要檢查5是否是SMALLINT使用 IS_SMALLINT 函數，請使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_SMALLINT(s) FROM t;

+---+-------------+
| s | is_smallint |
+---+-------------+
| 5 | true        |
+---+-------------+
```

瓦尔恰尔函数

檢查一個變量是否是一個VARCHAR。該 IS_VARCHAR 函數返回`true`所有字符串。該函數返回`false`任何其他值。

該 IS_VARCHAR 函數是 IS_CHAR 函數的超集。

語法

```
IS_VARCHAR(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

BOOLEAN

範例

若要檢查abc是否是VARCHAR使用 IS_VARCHAR 函數，請使用下列範例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('abc');

SELECT s, IS_VARCHAR(s) FROM t;

+-------+------------+
|   s   | is_varchar |
+-------+------------+
| "abc" | true       |
+-------+------------+
```

函數類型

JSON_TYPEOF 標量函數返回一個VARCHAR帶有布爾值，數字，字符串，對象，數組或空值，這取決於值的動態類型。SUPER

語法

```
JSON_TYPEOF(super_expression)
```

引數

超級表達式

表SUPER示式或欄。

傳回類型

VARCHAR

範例

若要使用 JSON_TYPEOF 函式檢查陣列的 JSON 類型，請[1,2]使用下列範例。

```
SELECT JSON_TYPEOF(ARRAY(1,2));

+-------------+
| json_typeof |
+-------------+
| array       |
+-------------+
```

瓦字節函數

AWS Clean Rooms支援以下 VARBYTE 函數。

主題

- 來自十六進位函數
- 從瓦字節函數
- TO_HEX 函數
- 到瓦字節函數

來自十六進位函數

FROM_HEX 將十六進制轉換為二進制值。

語法

```
FROM_HEX(hex_string)
```

引數

十六進制字符串

數據類型VARCHAR或TEXT要轉換的十六進制字符串。格式必須是文字值。

傳回類型

VARBYTE

範例

若要將的十六進位表示'6162'轉換為二進位值，請使用下列範例。結果會自動顯示為二進位值的十六進位表示。

```
SELECT FROM_HEX('6162');

+----------+
| from_hex |
+----------+
|     6162 |
+----------+
```

從瓦字節函數

FROM_VARBYTE 會將二進位值轉換成指定格式的字元字串。

語法

```
FROM_VARBYTE(binary_value, format)
```

引數

二進制值

資料類型的二進位值VARBYTE。

format

傳回字元字串的格式。不區分大小寫的有效值為hexbinary、utf-8、和utf8。

傳回類型

VARCHAR

範例

若要將二進位值轉換'ab'為十六進位，請使用下列範例。

```
SELECT FROM_VARBYTE('ab', 'hex');

+--------------+
| from_varbyte |
+--------------+
|         6162 |
+--------------+
```

TO_HEX 函數

TO_HEX 將數字或二進制值轉換為十六進制表示。

語法

```
TO_HEX(value)
```

引數

值

要轉換的數字或二進制值（VARBYTE）。

傳回類型

VARCHAR

範例

若要將數字轉換為其十六進位表示法，請使用下列範例。

```
SELECT TO_HEX(2147676847);

+----------+
|  to_hex  |
+----------+
| 8002f2af |
+----------+To create a table, insert the VARBYTE representation of 'abc' to a
 hexadecimal number, and select the column with the value, use the following example.
```

到瓦字節函數

TO_VARBYTE 會將指定格式的字串轉換為二進位值。

語法

```
TO_VARBYTE(string, format)
```

引數

string

一個CHAR或VARCHAR字符串。

format

輸入字符串的格式。不區分大小寫的有效值為hexbinary、utf-8、和utf8。

傳回類型

VARBYTE

範例

若要將十六進位6162轉換為二進位值，請使用下列範例。結果會自動顯示為二進位值的十六進位表示。

```
SELECT TO_VARBYTE('6162', 'hex');

+------------+
| to_varbyte |
+------------+
|       6162 |
+------------+
```

範圍函數

使用範圍函數，您可以更有效地建立分析業務查詢。範圍函數對結果集的一個分割區或「視窗」執行運算，然後針對該視窗中的每一列傳回一個值。反之，非視窗函數對結果集的每一列執行計算。不同於彙總結果列的群組函數，範圍函數會保留運算式中的所有列。

傳回的值是利用該視窗中列集的值來計算。對於資料表的每一列，視窗會定義用於計算其他屬性的列集。視窗是以視窗規格 (OVER 子句) 並根據三個主要概念來定義：

- 視窗分割，其會形成列群組 (PARTITION 子句)
- 視窗排序，定義每一個分割區內列的順序或序列 (ORDER BY 子句)
- 視窗框，相對於每一列來定義，以進一步限制列組 (ROWS 規格)

範圍函數是查詢中最後執行的一組運算 (最後的 ORDER BY 子句除外)。所有聯結和所有 WHERE、GROUP BY 及 HAVING 子句都在範圍函數處理之前完成。因此，範圍函數只能出現在 select 清單或 ORDER BY 子句中。您可以在具有不同窗框子句的單一查詢中使用多個範圍函數。您也可以在其他純量運算式 (例如 CASE) 中使用範圍函數。

範圍函數語法摘要

範圍函數遵循標準語法，如下所示。

```
function (expression) OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list [ frame_clause ] ] )
```

其中，function 是本節所述其中一個函數。

expr_list 如下。

```
expression | column_name [, expr_list ]
```

order_list 如下。

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

frame_clause 如下。

```
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 }}
```

引數

函數

範圍函數。如需詳細資訊，請參閱個別函數描述。

OVER

此子句定義視窗規格。OVER 是範圍函數的必要子句，用於區分範圍函數和其他 SQL 函數。

PARTITION BY expr_list

(選用) PARTITION BY 子句將結果集細分為分割區，很像 GROUP BY 子句。如果有分割區子句，則會對每一個分割區的列來計算函數。如果未指定分割區子句，則單一分割區包含整個資料表，且會針對這整個資料表來計算函數。

排名函數 DENSE_RANK、NTILE、RANK 及 ROW_NUMBER 需要整體比較結果集的所有列。使用 PARTITION BY 子句時，查詢最佳化工具可以根據分割區將工作負載分散至多個配量，以平行執行每一個彙總。如果沒有 PARTITION BY 子句，則必須在單一配量上循序執行彙總步驟，這可能對效能造成嚴重的負面影響，尤其對於大型叢集。

AWS Clean Rooms 不支持分區 BY 子句中的字符串文字。

ORDER BY order_list

(選用) 範圍函數會套用至每一個分割區內根據 ORDER BY 中的順序規格所排序的列。此 ORDER BY 子句不同於且完全無關於 frame_clause 中的 ORDER BY 子句。使用 ORDER BY 子句可以不搭配 PARTITION BY 子句。

對於排名函數，ORDER BY 子句可辨識排名值的量值。對於彙總函數，在為每一個窗框計算彙總函數之前，分割的列必須排序。如需範圍函數的詳細資訊，請參閱範圍函數。

順序清單中需要欄識別碼或可評估為欄識別碼的欄表達式。常數或常數表達式都不能用來替代欄名。

NULLS 值自成一組，根據 NULLS FIRST 或 NULLS LAST 選項來排序和排名。根據預設，依 ASC 順序排序時，NULL 值排在最後面，而依 DESC 順序排序時，則排在最前面。

AWS Clean Rooms ORDER BY 子句中不支持字符串文字。

如果省略 ORDER BY 子句，則列的順序不確定。

> **Note**
>
> 在任何 parallel 系統中，例如 AWS Clean Rooms，當 ORDER BY 子句不會產生資料的唯一且總排序時，資料列的順序是不具決定性的。也就是說，如果 ORDER BY 運算式產生重複的值 (部分排序)，這些資料列的傳回順序可能會有所不同。 AWS Clean Rooms 於是，範圍函數可能傳回非預期或不一致的結果。如需詳細資訊，請參閱 範圍函數的資料唯一排序。

column_name

分割或排序所依據的欄名。

ASC | DESC

此選項會定義表達式的排序順序，如下所示：

- ASC：遞增 (例如，數值從低到高，字元字串 'A' 到 'Z')。若未指定選項，資料會預設為遞增排序。
- DESC：遞減 (數值從高到低，字串 'Z' 到 'A')。

NULLS FIRST | NULLS LAST

這些選項指定 NULLS 應該排序在最前 (在非 Null 值之前) 或排序在最後 (在非 Null 值之後)。根據預設，NULLS 在 ASC 排序中排序和排名最後，而在 DESC 排序中排序和排名最前。

frame_clause

對於彙總函數，使用 ORDER BY 時，窗框子句會進一步調整函數視窗中的一個列集。它可讓您在排序的結果內包含或排除資料列組。窗框子句包含 ROWS 關鍵字和相關的指定元。

窗框子句不適用於排名函數。此外，當彙總函數的 OVER 子句中未使用 ORDER BY 子句時，不需要使用窗框子句。如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。

未指定 ORDER BY 子句時，隱含的窗框無邊界：相當於 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING。

ROWS

此子句定義視窗框作法是指定相對於目前列的實體位移。

此子句指定目前視窗或分割區中的列，以便與目前列的值結合。此子句使用引數來指定列位置，可能在目前列之前或之後。所有視窗框都以目前列為參考點。隨著視窗框在分割區中向前滑動，每一列會輪流變成目前列。

窗框可能是一組簡單的列，最遠到達且包含目前列。

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

也可能是兩個邊界之間的一個列集。

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING 表示視窗從分割區的第一列開始；offset PRECEDING 表示視窗從目前列之前相當於 offset 值的列數開始。UNBOUNDED PRECEDING 是預設值。

CURRENT ROW 表示視窗在目前列開始或結束。

UNBOUNDED FOLLOWING 表示視窗在分割區的最後一列結束；offset FOLLOWING 表示視窗在目前列之後相當於 offset 值的列數結束。

offset 表示目前列之前或之後的實體列數。在此案例中，offset 必須是評估為正數值的常數。例如，5 FOLLOWING 會在目前列之後的 5 列結束窗框。

未指定 BETWEEN 時，窗框會隱含地以目前列為邊界。例如，ROWS 5 PRECEDING 等於 ROWS BETWEEN 5 PRECEDING AND CURRENT ROW。此外，ROWS UNBOUNDED FOLLOWING 等於 ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING。

> **Note**
>
> 您不能指定開始邊界大於結束邊界的窗框。例如，您不能指定下列任何窗框：
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

範圍函數的資料唯一排序

如果範圍函數的 ORDER BY 子句無法產生唯一且完全的資料排序時，列的順序不確定。如果 ORDER BY 運算式產生重複值 (局部排序)，則這些行的傳回順序在多次執行中可能會有所不同。在這種情況下，範圍函數也可能傳回非預期或不一致的結果。

例如，以下查詢會在多次執行中傳回不同的結果。發生這些不同的結果是因為 `order by dateid` 不會為 SUM 範圍函數產生唯一的資料排序。

```
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
...
```

在此情況下，將第二個 ORDER BY 欄新增至範圍函數可能會解決問題。

```
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
...
```

支援的函數

AWS Clean Rooms 支援兩種類型的視窗函式：彙總和排名。

以下是支援的彙總函數：

- AVG 範圍函數
- COUNT 範圍函數
- CUME_DIST 範圍函數
- DENSE_RANK 範圍函數
- FIRST_VALUE 範圍函數
- LAG 範圍函數
- LAST_VALUE 範圍函數
- LEAD 範圍函數
- LISTAGG 範圍函數
- MAX 範圍函數
- MEDIAN 範圍函數
- MIN 範圍函數
- NTH_VALUE 範圍函數
- PERCENTILE_CONT 範圍函數
- PERCENTILE_DISC 範圍函數
- RATIO_TO_REPORT 範圍函數
- STDDEV_SAMP 和 STDDEV_POP 範圍函數 (STDDEV_SAMP 和 STDDEV 是同義詞)
- SUM 範圍函數
- VAR_SAMP 和 VAR_POP 範圍函數 (VAR_SAMP 和 VARIANCE 是同義詞)

以下是支援的排名函數：

- DENSE_RANK 範圍函數
- NTILE 範圍函數
- PERCENT_RANK 範圍函數
- RANK 範圍函數

- ROW_NUMBER 範圍函數

範圍函數範例的範例資料表

您可以在每個函數說明中找到特定的範圍函數範例。部分範例會使用名為 WINSALES 的資料表，其中包含 11 個資料列，如下表所示。

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	

AVG 範圍函數

AVG 範圍函數傳回輸入表達式值的平均值 (算術平均數)。AVG 函數處理數值，且忽略 NULL 值。

語法

```
AVG ( [ALL ] expression ) OVER
```

```
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

引數

expression

函數運算的目標欄或表達式。

ALL

如果指定引數 ALL，則函數在計數時會保留表達式中的所有重複值。ALL 為預設值。不支援 DISTINCT。

OVER

指定彙總函數的視窗子句。OVER 子句區分視窗彙總函數和正常組彙總函數。

PARTITION BY expr_list

以一或多個表達式定義 AVG 函數的視窗。

ORDER BY order_list

排序每一個分割區內的列。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果內包含或排除列集，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

資料類型

AVG 函數支援的引數類型包括 SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 及 DOUBLE PRECISION。

AVG 函數支援的傳回類型如下：

- BIGINT 代表 SMALLINT 或 INTEGER 引數
- NUMERIC 代表 BIGINT 引數

- DOUBLE PRECISION 代表浮點數引數

範例

下列範例會依日期計算銷售數量的移動平均數，依日期 ID 和銷售 ID 排序結果：

```
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)
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

COUNT 範圍函數

COUNT 範圍函數會計算表達式所定義的列數。

COUNT 函數有兩種版本。COUNT(*) 計算目標資料表中的所有列數，而不論是否包含 Null。COUNT (表達式) 計算特定欄或表達式中不含 NULL 值的列數。

語法

```
COUNT ( * | [ ALL ] expression) OVER
(
[ PARTITION BY expr_list ]
```

```
[ ORDER BY order_list
                         frame_clause ]
)
```

引數

expression

函數運算的目標欄或表達式。

ALL

如果指定引數 ALL，則函數在計數時會保留表達式中的所有重複值。ALL 為預設值。不支援 DISTINCT。

OVER

指定彙總函數的視窗子句。OVER 子句區分視窗彙總函數和正常組彙總函數。

PARTITION BY expr_list

以一或多個表達式定義 COUNT 函數的視窗。

ORDER BY order_list

排序每一個分割區內的列。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果內包含或排除列集，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

資料類型

COUNT 函數支援所有引數資料類型。

COUNT 函數支援的傳回類型為 BIGINT。

範例

下列範例顯示從資料視窗開頭的所有列的銷售 ID、數量和計數：

```
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)
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

下列範例顯示如何從資料視窗開頭計算非 null 列的銷售 ID、數量和計數。(在 WINSALES 資料表中，QTY_SHIPPED 欄包含一些 NULL。)

```
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 範圍函數

計算視窗或分割區內值的累積分佈。假定為遞增排序，使用此公式來決定累積分佈：

```
count of rows with values <= x / count of rows in the window or partition
```

其中，x 等於 ORDER BY 子句所指定欄之目前列中的值。以下資料集示範此公式的使用：

```
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
```

傳回值範圍是 >0 至 1 (含)。

語法

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

引數

OVER

用於指定視窗分割的子句。OVER 子句不能包含視窗框規格。

PARTITION BY partition_expression

選用。此表達式針對 OVER 子句中的每一個群組，設定記錄範圍。

ORDER BY order_list

要計算累積分佈的表達式。表達式必須為數值資料類型，或可隱含地轉換為數值資料類型。如果省略 ORDER BY，所有列的傳回值為 1。

如果 ORDER BY 未產生唯一排序，則列的順序不確定。如需詳細資訊，請參閱 範圍函數的資料唯一排序。

傳回類型

FLOAT8

範例

以下範例計算每一個賣方的數量累積分佈：

```
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
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

DENSE_RANK 範圍函數

DENSE_RANK 範圍函數根據 OVER 子句中的 ORDER BY 表達式，決定一組值之中某個值的排名。如果有選用的 PARTITION BY 子句，則會重設每一組列的排名。在排名準則中有相等值的列獲得相同排名。DENSE_RANK 函數有一方面不同於 RANK：如果兩列以上繫結在一起，則排名值的序列中沒有間隙。例如，假設兩列都排名 1，則下一個排名為 2。

在相同查詢中，排名函數可以搭配不同的 PARTITION BY 和 ORDER BY 子句。

語法

```
DENSE_RANK () OVER
(
[ PARTITION BY expr_list ]
```

```
[ ORDER BY order_list ]
)
```

引數

( )

此函數不接受引數，但需要空括號。

OVER

DENSE_RANK 函數的視窗子句。

PARTITION BY expr_list

選用。一或多個用於定義視窗的表達式。

ORDER BY order_list

選用。排名值所根據的表達式。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。如果省略 ORDER BY，所有列的傳回值為 1。

如果 ORDER BY 未產生唯一排序，則列的順序不確定。如需詳細資訊，請參閱 範圍函數的資料唯一排序。

傳回類型

INTEGER

範例

下列範例會依售出的數量 (以遞減順序) 排序表格，並為每個資料列指定密集等級與一般順序。套用範圍函數結果之後排序結果。

```
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
(11 rows)
```

在相同查詢中同時使用 DENSE_RANK 和 RANK 函數時，請注意指派給相同列集的排名差異。如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

下列範例會依 SELLERID 對表格進行分割，並依數量 (以遞減順序) 對每個分割區進行排序，並為每個資料列指派密集等級。套用範圍函數結果之後排序結果。

```
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
(11 rows)
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

FIRST_VALUE 範圍函數

在一組已排序的列中，FIRST_VALUE 會針對視窗框中的第一列，傳回指定之表達式的值。

如需有關選取視窗框中最後一列的資訊，請參閱LAST_VALUE 範圍函數。

語法

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引數

運算式

函數運算的目標欄或表達式。

IGNORE NULLS

此選項與 FIRST_VALUE 一起使用時，函數會傳回窗框中第一個非 NULL (或如果所有值都是 NULL，則為 NULL) 的值。

RESPECT NULLS

指出在判斷要使用哪一列時， AWS Clean Rooms 應包含 null 值。如果您不指定 IGNORE NULLS，則預設支援 RESPECT NULLS。

OVER

引進函數的視窗子句。

PARTITION BY expr_list

以一或多個表達式定義函數的視窗。

ORDER BY order_list

排序每一個分割區內的列。如果未指定 PARTITION BY 子句，ORDER BY 會排序整個資料表。如果您指定 ORDER BY 子句，則還必須指定 frame_clause。

FIRST_VALUE 函數的結果取決於資料的排序。在下列情況中，結果不確定：

- 未指定 ORDER BY 子句，且分割區包含一個表達式的兩個不同值
- 表達式評估為不同值，而這些值對應於 ORDER BY 清單中的相同值。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果中包含或排除資料列組，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

傳回類型

這些函數支持使用原始 AWS Clean Rooms 數據類型的表達式。傳回類型與運算式的資料類型相同。

範例

下列範例傳回 VENUE 資料表中每個會場的座位容量，且結果依容量排序 (高到低)。會使用 FIRST_VALUE 函數來選取與窗框之第一列對應的會場名稱：在此案例中，即座位數最多的那一列。結果依州分割，所以當 VENUESTATE 值變更時，就會選取新的第一個值。視窗框無界限，對於每一個分割區的第一列，選取的第一個值都相同。

以加利佛尼亞來說，Qualcomm Stadium 的座位數最多 (70561)，因此，對於 CA 分割區中的所有列，此名稱是第一個值。

```
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
...
```

LAG 範圍函數

LAG 範圍函數傳回分割區中目前列上方 (之前) 給定位移那一列的值。

語法

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

引數

value_expr

函數運算的目標欄或表達式。

offset

選擇性參數，指定在目前列之前要傳回值的列數。位移可以是常數整數，或評估為整數的表達式。如果您不指定偏移， AWS Clean Rooms 會使用做1為預設值。位移 0 表示目前列。

IGNORE NULLS

選擇性規格，指出在判斷要使用哪一列時， AWS Clean Rooms 應略過 null 值。如果未列出 IGNORE NULLS，則會包含 Null 值。

> **Note**
>
> 您可以使用 NVL 或 COALESCE 表達式，將 Null 值換成另一個值。

RESPECT NULLS

指出在判斷要使用哪一列時， AWS Clean Rooms 應包含 null 值。如果您不指定 IGNORE NULLS，則預設支援 RESPECT NULLS。

OVER

指定視窗分割和排序。OVER 子句不能包含視窗框規格。

PARTITION BY window_partition

選擇性引數，針對 OVER 子句中的每一個群組，設定記錄範圍。

ORDER BY window_ordering

排序每一個分割區內的列。

LAG 視窗函數支援使用任何 AWS Clean Rooms 資料類型的運算式。傳回類型與 value_expr 的類型相同。

範例

下列範例顯示銷售給買方 ID 為 3 之買方的門票數量，以及買方 3 購買門票的時間。為了比較買方 3 的每次銷售與前次銷售，查詢會傳回每次銷售的前次銷售數量。因為 2008/1/16 之前沒有購買，所以第一個先前銷售數量值為 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 範圍函數

在一組已排序的列中，LAST_VALUE 函數針對窗框中的最後一列，傳回運算式的值。

如需有關選取框架中第一列的資訊，請參閱FIRST_VALUE 範圍函數。

語法

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引數

運算式

函數運算的目標欄或表達式。

IGNORE NULLS

函數會傳回窗框中非 NULL (或如果所有值都是 NULL，則為 NULL) 的最後一個值。

RESPECT NULLS

指出在判斷要使用哪一列時， AWS Clean Rooms 應包含 null 值。如果您不指定 IGNORE NULLS，則預設支援 RESPECT NULLS。

OVER

引進函數的視窗子句。

PARTITION BY expr_list

以一或多個表達式定義函數的視窗。

ORDER BY order_list

排序每一個分割區內的列。如果未指定 PARTITION BY 子句，ORDER BY 會排序整個資料表。如果您指定 ORDER BY 子句，則還必須指定 frame_clause。

結果取決於資料的順序。在下列情況中，結果不確定：

- 未指定 ORDER BY 子句，且分割區包含一個表達式的兩個不同值
- 表達式評估為不同值，而這些值對應於 ORDER BY 清單中的相同值。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果中包含或排除資料列組，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

傳回類型

這些函數支持使用原始 AWS Clean Rooms 數據類型的表達式。傳回類型與運算式的資料類型相同。

範例

下列範例傳回 VENUE 資料表中每個會場的座位容量，且結果依容量排序 (高到低)。LAST_VALUE 函數用於選取與窗框之最後一列對應的會場名稱：在此案例中，即座位數最少的那一列。結果依州分割，所以當 VENUESTATE 值變更時，就會選取新的最後一個值。視窗框無界限，對於每一個分割區的第一列，選取的最後一個值都相同。

以加利佛尼亞來說，分割區中的每一列列都傳回 Shoreline Amphitheatre，因為其座位數最少 (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 範圍函數

LEAD 範圍函數傳回分割區中目前列下方 (之後) 給定位移那一列的值。

語法

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

引數

value_expr

函數運算的目標欄或表達式。

offset

選擇性參數，指定在目前列下方要傳回值的列數。位移可以是常數整數，或評估為整數的表達式。如果您不指定偏移， AWS Clean Rooms 會使用做1為預設值。位移 0 表示目前列。

IGNORE NULLS

選擇性規格，指出在判斷要使用哪一列時， AWS Clean Rooms 應略過 null 值。如果未列出 IGNORE NULLS，則會包含 Null 值。

> **Note**
>
> 您可以使用 NVL 或 COALESCE 表達式，將 Null 值換成另一個值。

RESPECT NULLS

指出在判斷要使用哪一列時， AWS Clean Rooms 應包含 null 值。如果您不指定 IGNORE NULLS，則預設支援 RESPECT NULLS。

OVER

指定視窗分割和排序。OVER 子句不能包含視窗框規格。

PARTITION BY window_partition

選擇性引數，針對 OVER 子句中的每一個群組，設定記錄範圍。

ORDER BY window_ordering

排序每一個分割區內的列。

LEAD 視窗函數支援使用任何 AWS Clean Rooms 資料類型的運算式。傳回類型與 value_expr 的類型相同。

範例

下列範例提供 SALES 資料表中於 2008 年 1 月 1 日和 2008 年 1 月 2 日售出門票之活動的佣金，以及對隨後銷售之門票銷售支付的佣金。

```
select eventid, commission, saletime,
lead(commission, 1) over (order by saletime) as next_comm
from sales where saletime between '2008-01-01 00:00:00' and '2008-01-02 12:59:59'
order by saletime;

eventid | commission |      saletime       | next_comm
---------+------------+---------------------+-----------
6213 |      52.05 | 2008-01-01 01:00:19 |    106.20
7003 |     106.20 | 2008-01-01 02:30:52 |    103.20
8762 |     103.20 | 2008-01-01 03:50:02 |     70.80
1150 |      70.80 | 2008-01-01 06:06:57 |     50.55
1749 |      50.55 | 2008-01-01 07:05:02 |    125.40
8649 |     125.40 | 2008-01-01 07:26:20 |     35.10
2903 |      35.10 | 2008-01-01 09:41:06 |    259.50
6605 |     259.50 | 2008-01-01 12:50:55 |    628.80
6870 |     628.80 | 2008-01-01 12:59:34 |     74.10
6977 |      74.10 | 2008-01-02 01:11:16 |     13.50
4650 |      13.50 | 2008-01-02 01:40:59 |     26.55
4515 |      26.55 | 2008-01-02 01:52:35 |     22.80
5465 |      22.80 | 2008-01-02 02:28:01 |     45.60
5465 |      45.60 | 2008-01-02 02:28:02 |     53.10
7003 |      53.10 | 2008-01-02 02:31:12 |     70.35
4124 |      70.35 | 2008-01-02 03:12:50 |     36.15
1673 |      36.15 | 2008-01-02 03:15:00 |   1300.80
```

```
...
(39 rows)
```

LISTAGG 範圍函數

對於查詢中的每一組，LISTAGG 範圍函數依據 ORDER BY 表達式來排序該組的列，然後將這些值串連成單一字串。

LISTAGG 是僅限於運算節點的函數。如果查詢未參考使用者定義的資料表或 AWS Clean Rooms 系統資料表，函式會傳回錯誤。

語法

```
LISTAGG( [DISTINCT] expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
OVER ( [PARTITION BY partition_expression] )
```

引數

DISTINCT

(選用) 此子句在串連值之前會從指定的表達式中消除重複值。結尾空格會忽略，所以字串 'a' 和 'a ' 視為重複值。LISTAGG 會使用第一個遇到的值。如需詳細資訊，請參閱 多餘空格的意義。

aggregate_expression

任何有效表達式 (例如欄名)，用於提供要彙總的值。忽略 NULL 值和空字串。

delimiter

(選用) 用來區隔串連值的字串常數。預設值為 NULL。

AWS Clean Rooms 支援可選逗號或冒號周圍任何數量的前導或尾隨空格，以及空字串或任意數量的空格。

有效值的範例如下：

", "

": "

" "

WITHIN GROUP (ORDER BY order_list)

(選用) 此子句指定彙總值的排序順序。只有在 ORDER BY 提供唯一排序時才可確定。預設是彙總所有列並傳回單一值。

OVER

用於指定視窗分割的子句。OVER 子句不能包含視窗排序或視窗框規格。

PARTITION BY partition_expression

(選用) 針對 OVER 子句中的每一個群組，設定記錄範圍。

傳回值

VARCHAR(MAX)。如果結果集大於 VARCHAR 大小上限 (64K – 1，或 65535)，則 LISTAGG 會傳回下列錯誤：

```
Invalid operation: Result size exceeds LISTAGG limit
```

範例

下列範例使用 WINSALES 資料表。如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

下列範例傳回賣方 ID 清單，依賣方 ID 排序。

```
select listagg(sellerid)
within group (order by sellerid)
over() from winsales;

  listagg
------------
 11122333344
...
...
 11122333344
 11122333344
   (11 rows)
```

下列範例傳回買方 B 的賣方 ID 清單，依日期排序。

```
select listagg(sellerid)
within group (order by dateid)
over () as seller
from winsales
where buyerid = 'b' ;

  seller
---------
    3233
    3233
    3233
    3233

(4 rows)
```

下列範例以逗號分隔清單傳回買方 B 的銷售日期。

```
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

(4 rows)
```

下列範例使用 DISTINCT 傳回買方 B 的唯一銷售日期清單。

```
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

(4 rows)
```

下列範例以逗號分隔清單傳回每個買方 ID 的銷售 ID。

```
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  |20001,30001,30004,30003
        b  |20001,30001,30004,30003
        b  |20001,30001,30004,30003
        b  |20001,30001,30004,30003
        c  |10001,20002,30007,10006
        c  |10001,20002,30007,10006
        c  |10001,20002,30007,10006
        c  |10001,20002,30007,10006
(11 rows)
```

MAX 範圍函數

MAX 範圍函數傳回輸入表達式值的最大值。MAX 函數處理數值，且忽略 NULL 值。

語法

```
MAX ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引數

expression

函數運算的目標欄或表達式。

ALL

如果指定引數 ALL，函數會保留表達式中的所有重複值。ALL 為預設值。不支援 DISTINCT。

OVER

此子句指定彙總函數的視窗子句。OVER 子句區分視窗彙總函數和正常組彙總函數。

PARTITION BY expr_list

以一或多個表達式定義 MAX 函數的視窗。

ORDER BY order_list

排序每一個分割區內的列。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果內包含或排除列集，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

資料類型

接受任何資料類型做為輸入。傳回與 expression 相同的資料類型。

範例

下列範例顯示資料視窗開頭的銷售 ID、數量和最大數量：

```
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)
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

下列範例顯示受限窗框中的 salesid、數量和最大數量：

```
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 範圍函數

計算視窗或分割區內值範圍的中位數。忽略範圍中的 NULL 值。

MEDIAN 是採用連續分佈模型的反向分佈函數。

MEDIAN 是僅限於運算節點的函數。如果查詢未參考使用者定義的資料表或 AWS Clean Rooms 系統資料表，函式會傳回錯誤。

語法

```
MEDIAN ( median_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

引數

median_expression

此表達式 (例如欄名) 提供要決定中位數的值。表達式必須為數值或日期時間資料類型，或可隱含地轉換為這種資料類型。

OVER

用於指定視窗分割的子句。OVER 子句不能包含視窗排序或視窗框規格。

PARTITION BY partition_expression

選用。此表達式針對 OVER 子句中的每一個群組，設定記錄範圍。

資料類型

傳回類型取決於 median_expression 的資料類型。下表顯示每一個 median_expression 資料類型的傳回類型。

輸入類型	傳回類型
數字，十進制	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE

使用須知

如果 median_expression 引數是以最大精確度 38 位數定義的 DECIMAL 資料類型，MEDIAN 可能會傳回不準確的結果或錯誤。如果 MEDIAN 函數的傳回值超過 38 位數，會將結果截斷為適合長度，導致精確度降低。在插補期間，如果中間結果超過最大精確度，則會發生數值溢位，且函數會傳回錯誤。為了避免這些情況，建議使用精確度較低的資料類型，或將 median_expression 引數轉換為較低精確度。

例如，搭配 DECIMAL 引數的 SUM 函數傳回的預設精確度為 38 位數。結果的小數位數和引數的小數位數相同。因此，例如，DECIMAL(5,2) 欄的 SUM 會傳回 DECIMAL(38,2) 資料類型。

下列範例在 MEDIAN 函數的 median_expression 引數中使用 SUM 函數。PRICEPAID 欄的資料類型是 DECIMAL (8,2)，所以 SUM 函數會傳回 DECIMAL(38,2)。

```
select salesid, sum(pricepaid), median(sum(pricepaid))
over() from sales where salesid < 10 group by salesid;
```

為了避免可能降低精確度或溢位錯誤，請將結果轉換為精確度較低的 DECIMAL 資料類型，如下列範例所示。

```
select salesid, sum(pricepaid), median(sum(pricepaid)::decimal(30,2))
over() from sales where salesid < 10 group by salesid;
```

範例

以下範例計算每一個賣方的銷售數量中位數：

```
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
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

MIN 範圍函數

MIN 範圍函數傳回輸入表達式值的最小值。MIN 函數處理數值，且忽略 NULL 值。

語法

```
MIN ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引數

expression

函數運算的目標欄或表達式。

ALL

如果指定引數 ALL，函數會保留表達式中的所有重複值。ALL 為預設值。不支援 DISTINCT。

OVER

指定彙總函數的視窗子句。OVER 子句區分視窗彙總函數和正常組彙總函數。

PARTITION BY expr_list

以一或多個表達式定義 MIN 函數的視窗。

ORDER BY order_list

排序每一個分割區內的列。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果內包含或排除列集，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

資料類型

接受任何資料類型做為輸入。傳回與 expression 相同的資料類型。

範例

下列範例顯示資料視窗開頭的銷售 ID、數量和最小數量：

```
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)
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

下列範例顯示受限窗框中的銷售 ID、數量和最小數量：

```
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 範圍函數

NTH_VALUE 範圍函數會相對於視窗的第一列，傳回視窗框之指定列的表達式值。

語法

```
NTH_VALUE (expr, offset)
[ IGNORE NULLS | RESPECT NULLS ]
OVER
( [ PARTITION BY window_partition ]
[ ORDER BY window_ordering
                        frame_clause ] )
```

引數

expr

函數運算的目標欄或表達式。

offset

相對於視窗中的第一列，決定要傳回表達式的列號。offset 可以是常數或表達式，且必須為大於 0 的正整數。

IGNORE NULLS

選擇性規格，指出在判斷要使用哪一列時， AWS Clean Rooms 應略過 null 值。如果未列出 IGNORE NULLS，則會包含 Null 值。

RESPECT NULLS

指出在判斷要使用哪一列時， AWS Clean Rooms 應包含 null 值。如果您不指定 IGNORE NULLS，則預設支援 RESPECT NULLS。

OVER

指定視窗分割、排序及視窗框。

PARTITION BY window_partition

針對 OVER 子句中的每一個群組，設定記錄範圍。

ORDER BY window_ordering

排序每一個分割區內的列。如果省略 ORDER BY，則預設窗框包含分割區中的所有列。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果中包含或排除資料列組，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

NTH_VALUE 視窗函數支援使用任何資料類型的運算式。 AWS Clean Rooms 傳回類型與 expr 的類型相同。

範例

下列範例顯示加利佛尼亞、佛羅里達及紐約的前三大會場的座位數，並對照這些州其他會場的座位數：

```
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 範圍函數

NTILE 範圍函數將分割區中排序的列劃分為指定數量的排名群組，且大小儘可能相等，然後傳回給定列所屬的群組。

語法

```
NTILE (expr)
OVER (
[ PARTITION BY expression_list ]
[ ORDER BY order_list ]
)
```

引數

expr

每一個分割區的排名群組數，且結果必須為整數值 (大於 0)。expr 引數必須不可為 Null。

OVER

用於指定視窗分割和排序的子句。OVER 子句不能包含視窗框規格。

PARTITION BY window_partition

選用。OVER 子句中每一個群組的記錄範圍。

ORDER BY window_ordering

選用。此表達式排序每一個分割區內的列。如果省略 ORDER BY 子句，則排名行為相同。

如果 ORDER BY 未產生唯一排序，則列的順序不確定。如需詳細資訊，請參閱 範圍函數的資料唯一排序。

傳回類型

BIGINT

範例

下列範例將 2008 年 8 月 26 日 Hamlet 門票的支付價格分成四個排名群組。結果集有 17 列，幾乎平均分散於排名 1 到 4：

```
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 範圍函數

計算給定資料列的百分比排行。使用此公式決定百分比排名：

```
(x - 1) / (the number of rows in the window or partition - 1)
```

其中 x 是目前列的排名。以下資料集示範此公式的使用：

```
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
```

傳回值範圍是 0 至 1 (含)。任何集的第一列的 PERCENT_RANK 為 0。

語法

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

引數

( )

此函數不接受引數，但需要空括號。

OVER

用於指定視窗分割的子句。OVER 子句不能包含視窗框規格。

PARTITION BY partition_expression

選用。此表達式針對 OVER 子句中的每一個群組，設定記錄範圍。

ORDER BY order_list

選用。要計算百分比排名的表達式。表達式必須為數值資料類型，或可隱含地轉換為數值資料類型。如果省略 ORDER BY，所有列的傳回值為 0。

如果 ORDER BY 未產生唯一排序，則列的順序不確定。如需詳細資訊，請參閱 範圍函數的資料唯一排序。

傳回類型

FLOAT8

範例

以下範例計算每一個賣方的銷售數量百分比排名：

```
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
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

PERCENTILE_CONT 範圍函數

PERCENTILE_CONT 是採用連續分佈模型的反向分佈函數。它採用百分位數值和排序規格，且會傳回插入值，該值將根據排序規格落入給定的百分位數值。

PERCENTILE_CONT 在值排序後計算值之間的線性插值。此函數在列根據排序規格來排序後，使用彙總群組中的百分位數值 (P) 和非 Null 列數 (N) 來計算列號。此列號 (RN) 是根據公式 RN = (1+(P*(N-1)) 來計算。彙總函數的最終結果是以列號 CRN = CEILING(RN) 到 FRN = FLOOR(RN) 各列的值之間的線性插值來計算。

最終結果如下。

如果 (CRN = FRN = RN)，則結果為 (value of expression from row at RN)

否則結果如下：

(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN).

您只能在 OVER 子句中指定 PARTITION 子句。如果指定 PARTITION，則對於每一列，PERCENTILE_CONT 會傳回在給定分割區內的一組值之中落在指定百分位數的值。

PERCENTILE_CONT 是僅限於運算節點的函數。如果查詢未參考使用者定義的資料表或 AWS Clean Rooms 系統資料表，函式會傳回錯誤。

語法

```
PERCENTILE_CONT ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

引數

percentile

介於 0 和 1 之間的數值常數。計算時會忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定要排序和計算百分位數的數值或日期/時間值。

OVER

指定視窗分割。OVER 子句不能包含視窗排序或視窗框規格。

PARTITION BY expr

選擇性引數，針對 OVER 子句中的每一個群組，設定記錄範圍。

傳回值

傳回類型取決於 WITHIN GROUP 子句中 ORDER BY 表達式的資料類型。下表顯示每一個 ORDER BY 表達式資料類型的傳回類型。

輸入類型	傳回類型
小型整數大數字, 十進制	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE

輸入類型	傳回類型
TIMESTAMP	TIMESTAMP

使用須知

如果 ORDER BY 表達式是以最大精確度 38 位數定義的 DECIMAL 資料類型，PERCENTILE_CONT 可能會傳回不準確的結果或錯誤。如果 PERCENTILE_CONT 函數的傳回值超過 38 位數，結果會截斷為適合長度，導致精確度降低。在插補期間，如果中間結果超過最大精確度，則會發生數值溢位，且函數會傳回錯誤。為了避免這些情況，建議使用精確度較低的資料類型，或將 ORDER BY 表達式轉換為較低精確度。

例如，搭配 DECIMAL 引數的 SUM 函數傳回的預設精確度為 38 位數。結果的小數位數和引數的小數位數相同。因此，例如，DECIMAL(5,2) 欄的 SUM 會傳回 DECIMAL(38,2) 資料類型。

下列範例在 PERCENTILE_CONT 函數的 ORDER BY 子句中使用 SUM 函數。PRICEPAID 欄的資料類型是 DECIMAL (8,2)，所以 SUM 函數會傳回 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;
```

為了避免可能降低精確度或溢位錯誤，請將結果轉換為精確度較低的 DECIMAL 資料類型，如下列範例所示。

```
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;
```

範例

下列範例使用 WINSALES 資料表。如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

```
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)
```

以下範例計算華盛頓州之賣方門票銷售的 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 範圍函數

PERCENTILE_DISC 是採用離散分佈模型的反向分佈函數。它採用百分位數值和排序規格，且會傳回給定集裡的一個元素。

針對給定的百分位數值 P，PERCENTILE_DISC 排序 ORDER BY 子句中的表達式值，且傳回的值具有大於或等於 P 的最小累積分佈值 (根據相同的排序規格)。

您只能在 OVER 子句中指定 PARTITION 子句。

PERCENTILE_DISC 是僅限於運算節點的函數。如果查詢未參考使用者定義的資料表或 AWS Clean Rooms 系統資料表，函式會傳回錯誤。

語法

```
PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

引數

percentile

介於 0 和 1 之間的數值常數。計算時會忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定要排序和計算百分位數的數值或日期/時間值。

OVER

指定視窗分割。OVER 子句不能包含視窗排序或視窗框規格。

PARTITION BY expr

選擇性引數，針對 OVER 子句中的每一個群組，設定記錄範圍。

傳回值

資料類型與 WITHIN GROUP 子句中的 ORDER BY 表達式相同。

範例

下列範例使用 WINSALES 資料表。如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over() as median from winsales;

sellerid | qty | median
----------+-----+--------
        1 |  10 |     20
        3 |  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
(11 rows)
```

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

sellerid | qty | median
----------+-----+--------
        2 |  20 |     20
        2 |  20 |     20
        4 |  10 |     10
        4 |  40 |     10
```

```
        1 |  10 |     10
        1 |  10 |     10
        1 |  30 |     10
        3 |  10 |     15
        3 |  15 |     15
        3 |  20 |     15
        3 |  30 |     15
(11 rows)
```

RANK 範圍函數

RANK 範圍函數根據 OVER 子句中的 ORDER BY 表達式，決定一組值之中某個值的排名。如果有選用的 PARTITION BY 子句，則會重設每一組列的排名。排名條件值相等的列會獲得相同的等級。AWS Clean Rooms 將綁定的行數添加到綁定的排名中以計算下一個排名，因此排名可能不是連續的數字。例如，假設兩列都排名 1，則下一個排名為 3。

RANK 函數有一方面不同於 DENSE_RANK 範圍函數：對於 DENSE_RANK，如果兩列以上繫結在一起，則排名值的序列中沒有間隙。例如，假設兩列都排名 1，則下一個排名為 2。

在相同查詢中，排名函數可以搭配不同的 PARTITION BY 和 ORDER BY 子句。

語法

```
RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

引數

( )

此函數不接受引數，但需要空括號。

OVER

RANK 函數的視窗子句。

PARTITION BY expr_list

選用。一或多個用於定義視窗的表達式。

ORDER BY order_list

選用。定義排名值所根據的欄。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。如果省略 ORDER BY，所有列的傳回值為 1。

如果 ORDER BY 未產生唯一排序，則列的順序不確定。如需詳細資訊，請參閱 範圍函數的資料唯一排序。

傳回類型

INTEGER

範例

下列範例會依銷售數量排序資料表 (預設為遞增)，並將排名指派給每一列。最高排名的值為 1。套用範圍函數結果之後排序結果：

```
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)
```

請注意，此範例中的外部 ORDER BY 子句包含資料行 2 和 1，以確保每次執行此查詢時都會 AWS Clean Rooms 傳回一致的排序結果。例如，銷售 ID 為 10001 和 10006 的列有相同的 QTY 和 RNK 值。依第 1 欄排序最終結果集可確保列 10001 一定位於 10006 之前。如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

在以下範例中，範圍函數反向排序 (order by qty desc)。現在，最高排名值套用至最大 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)
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

下列範例會依 SELLERID 分割資料表，並依數量排序每一個分割區 (以遞減順序)，然後指派排名給每一列。套用範圍函數結果之後排序結果。

```
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 範圍函數

計算視窗或分割區內值與值總和的比率。使用下列公式決定報告值的比率：

`value of` ratio_expression `argument for the current row / sum of` ratio_expression `argument for the window or partition`

以下資料集示範此公式的使用：

```
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
```

傳回值範圍是 0 至 1 (含)。如果 ratio_expression 為 NULL，則傳回值為 NULL。

語法

```
RATIO_TO_REPORT ( ratio_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

引數

ratio_expression

此表達式 (例如欄名) 提供要決定比率的值。表達式必須為數值資料類型，或可隱含地轉換為數值資料類型。

您不能在 ratio_expression 中使用其他任何分析函數。

OVER

用於指定視窗分割的子句。OVER 子句不能包含視窗排序或視窗框規格。

PARTITION BY partition_expression

選用。此表達式針對 OVER 子句中的每一個群組，設定記錄範圍。

傳回類型

FLOAT8

範例

以下範例計算每一個賣方的銷售數量比例：

```
select sellerid, qty, ratio_to_report(qty)
over (partition by sellerid)
from winsales;

sellerid qty  ratio_to_report
----------------------------------------
2  20.12312341      0.5
2  20.08630000      0.5
4  10.12414400      0.2
4  40.23000000      0.8
1  30.37262000      0.6
1  10.64000000      0.21
1  10.00000000      0.2
3  10.03500000      0.13
3  15.14660000      0.2
3  30.54790000      0.4
3  20.74630000      0.27
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

ROW_NUMBER 範圍函數

根據 OVER 子句中的 ORDER BY 表達式，決定一組列之內目前列的序數 (從 1 起算)。如果有選用的 PARTITION BY 子句，則會重設每一組列的序數。對於 ORDER BY 表達式，具有相等值的列會獲得非決定性的不同列號。

語法

```
ROW_NUMBER () OVER
```

```
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

引數

( )

此函數不接受引數，但需要空括號。

OVER

ROW_NUMBER 函數的視窗子句。

PARTITION BY expr_list

選用。一或多個用於定義 ROW_NUMBER 函數的表達式。

ORDER BY order_list

選用。此表達式定義列號所根據的欄。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。

如果 ORDER BY 未產生唯一排序或被省略，則列的順序不確定。如需詳細資訊，請參閱 範圍函數的資料唯一排序。

傳回類型

BIGINT

範例

下列範例依 SELLERID 分割資料表，並依 QTY 排序每一個分割區 (以遞增順序)，然後將列號指派給每一列。套用範圍函數結果之後排序結果。

```
select salesid, sellerid, qty,
row_number() over
(partition by sellerid
 order by qty asc) as row
from winsales
order by 2,4;
```

```
 salesid | sellerid | qty | row
---------+----------+-----+-----
   10006 |        1 |  10 |   1
   10001 |        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
(11 rows)
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

STDDEV_SAMP 和 STDDEV_POP 範圍函數

STDDEV_SAMP 和 STDDEV_POP 範圍函數傳回一組數值 (整數、小數或浮點數) 的樣本標準差和母體標準差。另請參閱 STDDEV_SAMP 和 STDDEV_POP 函數。

STDDEV_SAMP 和 STDDEV 是同一個函數的同義詞。

語法

```
STDDEV_SAMP | STDDEV | STDDEV_POP
( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

引數

expression

函數運算的目標欄或表達式。

ALL

如果指定引數 ALL，函數會保留表達式中的所有重複值。ALL 為預設值。不支援 DISTINCT。

OVER

指定彙總函數的視窗子句。OVER 子句區分視窗彙總函數和正常組彙總函數。

PARTITION BY expr_list

以一或多個表達式定義函數的視窗。

ORDER BY order_list

排序每一個分割區內的列。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果內包含或排除列集，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

資料類型

STDDEV 函數支援的引數類型包括 SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 及 DOUBLE PRECISION。

不論表達式的資料類型，STDDEV 函數的傳回類型都是雙精確度數字。

範例

下列範例顯示如何使用 STDDEV_POP 和 VAR_POP 函數做為範圍函數。查詢計算 SALES 資料表中 PRICEPAID 值的母體變異數和母體標準差。

```
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
...
```

樣本標準差和變異數函數可如法泡製。

SUM 範圍函數

SUM 範圍函數傳回輸入欄或表達式值的總和。SUM 函數處理數值，且忽略 NULL 值。

語法

```
SUM ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

引數

expression

函數運算的目標欄或表達式。

ALL

如果指定引數 ALL，函數會保留表達式中的所有重複值。ALL 為預設值。不支援 DISTINCT。

OVER

指定彙總函數的視窗子句。OVER 子句區分視窗彙總函數和正常組彙總函數。

PARTITION BY expr_list

以一或多個表達式定義 SUM 函數的視窗。

ORDER BY order_list

排序每一個分割區內的列。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果內包含或排除列集，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

資料類型

SUM 函數支援的引數類型包括 SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 及 DOUBLE PRECISION。

SUM 函數支援的傳回類型如下：

- BIGINT 代表 SMALLINT 或 INTEGER 引數
- NUMERIC 代表 BIGINT 引數
- DOUBLE PRECISION 代表浮點數引數

範例

下列範例會建立銷售數量的累積 (滾動) 總和，依日期和銷售 ID 排序：

```
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)
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

以下範例會依日期建立銷售數量的累積 (滾動) 總和、依賣方 ID 分割結果，然後在分割區內依日期和銷售 ID 排序結果：

```
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)
```

以下範例將結果集的所有列依序編號，依 SELLERID 和 SALESID 欄排序：

```
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)
```

如需 WINSALES 資料表的描述，請參閱範圍函數範例的範例資料表。

以下範例將結果集的所有列依序編號、依 SELLERID 分割結果，然後在分割區內依 SELLERID 和 SALESID 排序結果：

```
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 範圍函數

VAR_SAMP 和 VAR_POP 範圍函數傳回一組數值 (整數、小數或浮點數) 的樣本變異數和母體變異數。另請參閱 VAR_SAMP 和 VAR_POP 函數。

VAR_SAMP 和 VARIANCE 是同一個函數的同義詞。

語法

```
VAR_SAMP | VARIANCE | VAR_POP
( [ ALL ] expression ) OVER
```

```
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

引數

expression

函數運算的目標欄或表達式。

ALL

如果指定引數 ALL，函數會保留表達式中的所有重複值。ALL 為預設值。不支援 DISTINCT。

OVER

指定彙總函數的視窗子句。OVER 子句區分視窗彙總函數和正常組彙總函數。

PARTITION BY expr_list

以一或多個表達式定義函數的視窗。

ORDER BY order_list

排序每一個分割區內的列。如果未指定 PARTITION BY，ORDER BY 會使用整個資料表。

frame_clause

如果彙總函數使用 ORDER BY 子句，則需要明確的窗框子句。窗框子句在排序的結果內包含或排除列集，以調整函數視窗中的一個列集。窗框子句包含 ROWS 關鍵字和相關的指定元。請參閱 範圍函數語法摘要。

資料類型

VARIANCE 函數支援的引數類型包括
SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 及 DOUBLE PRECISION。

不論表達式的資料類型，VARIANCE 函數的傳回類型都是雙精確度數字。

中的 SQL 條件 AWS Clean Rooms

條件是評估為 true、false 或未知之一或多個運算式和邏輯運算子的陳述式。條件有時也稱為述詞。

> **Note**
>
> 所有的字串比較和 LIKE 模式比對，都會區分大小寫。例如，「A」和「a」不符。不過，您可以使用 ILIKE 述詞，來進行不區分大小寫的模式比對。

中支援下列 SQL 條件 AWS Clean Rooms。

主題

比較條件

比較條件表示兩個值之間的邏輯關係。所有比較條件都是具有 Boolean 傳回類型的二元運算子。AWS Clean Rooms支援下表所述的比較運算子。

運算子	語法	描述
<	a < b	價值a小於值b。
>	a > b	價值a大於值b。
<=	a <= b	價值a小於或等於值b。

運算子	語法	描述
>=	`a >= b`	價值a大於或等於值b。
=	`a = b`	價值a等於值b。
<> 或 !=	`a <> b or a != b`	價值a不等於值b。
a = TRUE	`a IS TRUE`	價值a是布爾值TRUE。

使用須知

= ANY | SOME

任何和一些關鍵字是同義詞在條件。如果對於返回一個或多個值的子查詢返回的至少一個值的比較為 true，則 ANY 和 SOME 關鍵字返回 true。AWS Clean Rooms只支持任何和一些 = (等於) 條件。不支援不等式條件。

> **Note**
>
> 不支援 ALL 述詞。

<> ALL

ALL 關鍵字等同 NOT IN (請參閱 IN 條件 條件)，如果子查詢的結果中未包含表達式，將會傳回 true。AWS Clean Rooms 僅支援 ALL 的 <> 或 != (不等於)。不支援其他比較條件。

IS TRUE/FALSE/UNKNOWN

非 0 的值等於 TRUE、0 等於 FALSE，而 null 等於 UNKNOWN。請參閱 布林值 (Boolean) 類型 資料類型。

範例

下列是比較條件的一些簡單範例：

```
a = 5
a < b
```

```
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

以下查詢會傳回 VENUE 表格中擁有 10,000 個以上座位的場地：

```
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)
```

此範例會從 USERS 資料表中，選取喜歡搖滾樂的使用者 (USERID)：

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
13
16
(5 rows)
```

此範例會從 USERS 資料表中，選取不確定是否喜歡搖滾樂的使用者 (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
```

帶有 TIME 列的示例

下列範例表格 TIME_TEST 有一個資料行 TIME_VAL (類型為「時間」)，其中插入了三個值。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

下列範例會從每個 timetz_val 擷取小時數。

```
select time_val from time_test where time_val < '3:00';
   time_val
---------------
 00:00:00.5550
 00:58:00
```

下列範例會比較兩個時間常值。

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
```

```
 t
```

具有時間茲資料欄的範例

下列範例表格 TIMTZ_TEST 有一個資料行時間 TZ_VAL (類型為 TIMTZ)，其中插入了三個值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

下面的例子只選擇 TIMETZ 值小於3:00:00 UTC。將值轉換為 UTC 後進行比較。

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

下列範例會比較兩個 TIMTZ 文字。比較時會忽略時區。

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

 ?column?
----------
 t
```

邏輯條件

邏輯條件會合併兩個條件的結果，來產生單一結果。所有的邏輯條件都是二元運算子，具有 Boolean 傳回類型。

語法

```
expression
{ AND | OR }
expression
```

```
NOT expression
```

邏輯條件使用三種值的布林邏輯，其中 null 值代表未知的關係。下表說明邏輯條件的結果，其中 E1 和 E2 表示表達式：

E1	E2	E1 AND E2	E1 OR E2	NOT E2
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN (不明)	UNKNOWN (不明)	TRUE	UNKNOWN (不明)
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN (不明)	FALSE	UNKNOWN (不明)	
UNKNOWN (不明)	TRUE	UNKNOWN (不明)	TRUE	
UNKNOWN (不明)	FALSE	FALSE	UNKNOWN (不明)	
UNKNOWN (不明)	UNKNOWN (不明)	UNKNOWN (不明)	UNKNOWN (不明)	

NOT 運算子會在 AND 之前評估，而 AND 運算子會在 OR 運算子之前評估。如果使用任何括號，就可以覆蓋這個預設的評估順序。

範例

下列的範例會從 USERS 資料表，針對其中同時喜歡拉斯維加斯和運動的使用者，傳回其 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)
```

下一個範例會從 USERS 資料表，針對其中喜歡拉斯維加斯或運動，或是這兩者的使用者，傳回其 USERID 和 USERNAME。此查詢會傳回前一個範例的所有輸出資料，加上只喜歡拉斯維加斯或運動的使用者。

```
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)
```

下列的查詢在 OR 條件周圍加上括號，以找出在紐約或加州上演「馬克白」的場地：

```
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
...
```

如果移除此範例中的括號，將會改變查詢的邏輯和結果。

下列的範例使用 NOT 運算子：

```
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
...
```

下列範例使用 NOT 條件，後接 AND 條件：

```
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)
```

模式比對條件

模式匹配運算符搜索條件表達式中指定的模式的字符串，並根據它是否找到匹配返回 true 或 false。AWS Clean Rooms使用下列方法進行模式比對：

- LIKE 表達式

  LIKE 運算子會利用模式 (此模式使用萬用字元 % (百分比) 和 _ (底線)) 來比較字串表達式 (例如資料欄的名稱)。LIKE 模式比對的範圍一律涵蓋整個字串。LIKE 會進行區分大小寫的比對，ILIKE 則會進行不區分大小寫的比對。

- SIMILAR TO 規則表達式

  SIMILAR TO 運算子會利用 SQL 標準規則表達式的模式，來比對字串表達式，此模式包含一組模式比對中繼字元，其中包括 LIKE 運算子所支援的兩個字元。SIMILAR TO 會比對整個字串，並進行區分大小寫的比對。

主題

- LIKE
- SIMILAR TO

LIKE

LIKE 運算子會利用模式 (此模式使用萬用字元 % (百分比) 和 _ (底線)) 來比較字串表達式 (例如資料欄的名稱)。LIKE 模式比對的範圍一律涵蓋整個字串。若要比對字串中任意位置的序列，模式必須以百分比符號開頭和結尾。

LIKE 區分大小寫；ILIKE 不區分大小寫。

語法

```
expression [ NOT ] LIKE | ILIKE pattern [ ESCAPE 'escape_char' ]
```

引數

運算式

有效的 UTF-8 字元表達式，例如資料欄的名稱。

LIKE | ILIKE

LIKE 會進行區分大小寫的模式比對。ILIKE 會針對單位元組 UTF-8 (ASCII) 字元，進行不區分大小寫的模式比對。若要對多位元組字元執行不區分大小寫的模式比對，請使用較低功能開啟表達和模式有一個 LIKE 條件。

不同於比較述詞，例如 = 和 <>，LIKE 和 ILIKE 述詞並未隱含忽略結尾空格。若要忽略結尾空格，請 RTRIM 或將 CHAR 資料欄明確轉換為 VARCHAR。

該~~運算符等同於 LIKE，並且~~*相當於伊利克。也是!~~和!~~*運營商等同於不喜歡和不愛.

pattern

有效的 UTF-8 字元表達式，包含要比對的模式。

escape_char

字元表達式，將會用來逸出模式中的中繼字元。預設值為兩個反斜線 (「\\」)。

如果 pattern 未包含任何中繼字元，則模式只代表字串本身，此時 LIKE 的功用如同等於運算子。

兩個字元表達式都可以是 CHAR 或 VARCHAR 資料類型。如果不同，AWS Clean Rooms 會將 pattern 轉換為 expression 的資料類型。

LIKE 支援下列的模式比對中繼字元：

運算子	描述
%	比對任何 0 的序列或更多字元。
_	比對任一個單一字元。

範例

下表顯示範例，示範使用 LIKE 進行的模式比對：

表達式	傳回值
`'abc' LIKE 'abc'`	True
`'abc' LIKE 'a%'`	True
`'abc' LIKE '_B_'`	False
`'abc' ILIKE '_B_'`	True
`'abc' LIKE 'c%'`	False

下列範例會找出名稱以「E」開頭的所有城市：

```
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
...
```

下列範例會找出姓氏中包含「ten」的使用者：

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
-------------
Christensen
Wooten
```

```
...
```

下列範例會找出名稱中第 3 個和第 4 個字元為「ea」的城市。此指令使用 ILIKE 來示範不區分大小寫：

```
select distinct city from users where city ilike '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

下列範例會使用預設逸出字串 (\\) 來搜尋包含「start_」(文字) 的字串start後跟一個下劃線_):

```
select tablename, "column" from my_table_def


where "column" like '%start\\_%'
limit 5;

     tablename     |    column
-------------------+---------------
 my_s3client       | start_time
 my_tr_conflict    | xact_start_ts
 my_undone         | undo_start_ts
 my_unload_log     | start_time
 my_vacuum_detail | start_row
(5 rows)
```

下列範例會將「^」指定為逸出字元，然後使用逸出字元來搜尋包含「start_」(文字) 的字串start後跟一個下劃線_):

```
select tablename, "column" from my_table_def

where "column" like '%start^_%' escape '^'
limit 5;

     tablename     |    column
```

```
-------------------+---------------
 my_s3client       | start_time
 my_tr_conflict    | xact_start_ts
 my_undone         | undo_start_ts
 my_unload_log     | start_time
 my_vacuum_detail | start_row
(5 rows)
```

下列範例使用~~*運營商做一個不區分大小寫（ILIKE）搜索以「AG」開頭的城市。

```
select distinct city from users where city ~~* 'Ag%' order by city;

city
------------
Agat
Agawam
Agoura Hills
Aguadilla
```

SIMILAR TO

SIMILAR TO 運算子會利用 SQL 標準規則表達式的模式，來比對字串表達式 (例如資料欄的名稱)。SQL 標準規則表達式的模式可包含一組模式比對中繼字元，其中包括 LIKE 運算子所支援的兩個字元。

SIMILAR TO 運算子只會在其模式符合整個字串時傳回 true，不像 POSIX 規則表達式，其模式可以符合字串的任何部分。

SIMILAR TO 會進行區分大小寫的比對。

> **Note**
>
> 使用 SIMILAR TO 進行的規則表達式比對，其運算成本非常昂貴。我們建議盡可能使用 LIKE，尤其是在處理極為龐大的列數時。例如，下列查詢在功能上是相同的，但是使用 LIKE 的查詢執行速度比使用規則運算式的查詢快幾倍：
>
> ```
> select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
> select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die
> %';
> ```

語法

```
expression [ NOT ] SIMILAR TO pattern [ ESCAPE 'escape_char' ]
```

引數

運算式

有效的 UTF-8 字元表達式，例如資料欄的名稱。

SIMILAR TO

SIMILAR TO 會針對 expression 中的整個字串，進行區分大小寫的模式比對。

pattern

有效的 UTF-8 字元表達式，代表 SQL 標準規則表達式的模式。

escape_char

字元表達式，將會用來逸出模式中的中繼字元。預設值為兩個反斜線 (「\\」)。

如果模式未包含任何中繼字元，則模式只代表字串本身。

兩個字元表達式都可以是 CHAR 或 VARCHAR 資料類型。如果不同，AWS Clean Rooms 會將 pattern 轉換為 expression 的資料類型。

SIMILAR TO 支援下列的模式比對中繼字元：

運算子	描述
%	比對任何 0 的序列或更多字元。
_	比對任一個單一字元。
|	表示交替 (兩種替代選項中的任一種)。
*	重複前一個項目 0 次或更多次。
+	重複前一個項目 1 次或更多次。
?	重複前一個項目 0 次或 1 次。

運算子	描述
{m}	重複前一個項目 m 次。
{m,}	重複前一個項目 m 次或更多次。
{m,n}	重複前一個項目至少 m 次，而且不超過 n 次。
()	括號可將多個項目分組為單一邏輯項目。
[...]	方括號表達式表示字元類別，如同在 POSIX 規則表達式中的表示分式。

範例

下表顯示範例，示範使用 SIMILAR TO 進行的模式比對：

表達式	傳回值
'abc' SIMILAR TO 'abc'	True
'abc' SIMILAR TO '_b_'	True
'abc' SIMILAR TO '_A_'	False
'abc' SIMILAR TO '%(b|d)%'	True
'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

下列範例會尋找名稱包含「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
```

下列的範例使用預設的逸出字串 (「\\」)，來搜尋包含「_」的字串：

```
SELECT tablename, "column" FROM my_table_def
WHERE "column" SIMILAR TO '%start\\_%'

ORDER BY tablename, "column" LIMIT 5;

        tablename         |       column
--------------------------+---------------------
 my_abort_idle            | idle_start_time
 my_abort_idle            | txn_start_time
 my_analyze_compression | start_time
 my_auto_worker_levels  | start_level
 my_auto_worker_levels  | start_wlm_occupancy
```

下列的範例將「^」指定為逸出字元，然後使用該逸出字元來搜尋包含「_」的字串：

```
SELECT tablename, "column" FROM my_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
```

BETWEEN 範圍條件

BETWEEN 條件會使用關鍵字 BETWEEN 和 AND，來檢定表達式是否在一系列值的範圍內。

語法

```
expression [ NOT ] BETWEEN expression AND expression
```

表達式可以是數值、字元或日期時間 (datetime) 資料類型，但這些類型必須相容。範圍包含端點。

範例

第一個範例會計算有多少交易已登錄售出 2、3 或 4 張票券：

```
select count(*) from sales
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

範圍條件包含開頭值與結尾值。

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

範圍條件中第一個表達式的值，必須小於第二個表達式的值。由於表達式的值，下列的範例一律會傳回 0 列：

```
select count(*) from sales
where qtysold between 4 and 2;

count
```

```
-------
0
(1 row)
```

不過，套用 NOT 修飾符將會反轉邏輯，產生所有列的計數：

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
172456
(1 row)
```

下列的查詢會傳回擁有 20,000 到 50,000 個座位的場地清單：

```
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)
```

下面的例子演示了使用 BETWEEN 的日期值：

```
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
```

請注意，儘管 BETWEEN 的範圍包括在內，但日期默認為具有 00:00:00 的時間值。範例查詢的唯一有效 1 月 3 日資料列是銷售時間為的資料列`1/3/2008 00:00:00`。

Null 條件

該NULL當值丟失或未知時，對空值進行條件測試。

語法

```
expression IS [ NOT ] NULL
```

引數

運算式

任何表達式，例如資料欄。

IS NULL

表達式的值如果是 null 則為 true，表達式的值如果包含值，則為 false。

IS NOT NULL

表達式的值如果是 null 則為 false，表達式的值如果包含值，則為 true。

範例

此範例顯示 SALES 資料表的 QTYSOLD 欄位中有多少次包含 null：

```
select count(*) from sales
where qtysold is null;
count
-------
0
```

```
(1 row)
```

EXISTS 條件

EXISTS 條件會檢定在子查詢中是否存在列，如果子查詢傳回至少一列，則傳回 true。如果指定 NOT，則此條件會在子查詢未傳回任何列時傳回 true。

語法

```
[ NOT ] EXISTS (table_subquery)
```

引數

EXISTS

當 table_subquery 傳回至少一列時，其值為 true。

NOT EXISTS

當 table_subquery 未傳回任何列時，其值為 true。

table_subquery

子查詢，會評估包含一個或多個欄和一列或多列的資料表。

範例

此範例會針對具有任何類型銷售的日期，傳回所有的日期識別符，一次一個：

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
1827
1828
```

```
1829
...
```

IN 條件

一個INcondition 會測試一組值或子查詢中的成員資格值。

語法

```
expression [ NOT ] IN (expr_list | table_subquery)
```

引數

運算式

數值、字元或日期時間 (datetime) 表達式，會根據 expr_list 或 table_subquery 進行評估，而且必須與該清單或子查詢的資料類型相容。

expr_list

用英文逗號分隔的一個或多個表達式，或是用英文逗號分隔的一組或多組表達式 (用括號括住)。

table_subquery

子查詢，會評估包含一列或多列的資料表，但是其選擇清單中只限包含一個欄。

IN | NOT IN

如果表達式是表達式清單或查詢的成員，IN 會傳回 true。如果表達式不是成員，NOT IN 會傳回 true。在下列情況中，IN 和 NOT IN 會傳回 Null，而且不會傳回任何列：如果 expression 產生 null；或如果沒有符合的 expr_list 或 table_subquery 值，而且這些比較列其中至少有一列產生 null。

範例

只有這些列出的值，才會讓下列條件傳回 true：

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

大型 IN 清單的最佳化

為了實現最佳化的查詢效能，包含超過 10 個值的 IN 清單，會在內部轉換為純量陣列。包含不到 10 個值的 IN 清單，會轉換為一系列的 OR 述詞。SMALLINT、整數、大整數、實數、雙精度、布林值、字元、VARCHAR、日期、時間戳記和時間戳記資料類型支援此最佳化。

請檢視查詢的 EXPLAIN 輸出，以查看這個最佳化機制的效果。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)
```

語法

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

查詢會聯結資料

AWS Clean Rooms提供對關聯式和巢狀資料的 SQL 相容存取。

AWS Clean Rooms訪問嵌套數據時，使用虛線符號和數組下標進行路徑導航。它還使FROM子句項目遍歷數組並用於非巢狀操作。下列主題提供不同查詢模式的說明，這些模式將陣列/結構/對映資料類型與路徑和陣列導覽、取消巢狀和聯結結合在一起。

主題

Navigation (導覽)

AWS Clean Rooms啟用導覽至陣列和結構`[...]`括號和點符號分別。此外，您可以使用點符號和陣列使用括號表示法混合導航到結構中。

Example

例如，以下查詢會假設`c_orders`數組數據列是一個具有結構的數組和一個屬性被命名`o_orderkey`。

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

您可以在所有類型的查詢中使用點和括號表示法，例如篩選、聯結和彙總。您可以在查詢中使用這些符號，其中通常有列引用。

Example

下列範例會使用篩選結果的 SELECT 陳述式。

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0].o_orderkey IS NOT NULL;
```

Example

下列範例會在 GROUP BY 和 ORDER BY 子句中使用括號和點導覽。

```
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;
```

取消巢狀查詢

若要取消巢狀查詢，AWS Clean Rooms啟用對陣列的迭代。它通過使用查詢的 FROM 子句導航數組來實現這一點。

Example

使用上一個範例，下列範例會重複執行的屬性值c_orders。

```
SELECT o FROM customer_orders_lineitem c, c.c_orders o;
```

取消嵌套語法是 FROM 子句的擴展。在標準 SQL 中，FROM 子句x（AS）y意味著y迭代關係中的每個元組x。在這種情況下，x指的是一個關係和y指的是關係的別名x。同樣地，使用 FROM 子句項目取消嵌套的語法x（AS）y意味著y迭代數組表達式中的每個值x。在這種情況下，x是一個數組表達式和y是的別名x。

左操作數也可以使用點和括號表示法進行常規導航。

Example

在前列查詢會

- customer_orders_lineitem c是在迭代customer_order_lineitem基底資料表
- c.c_orders o是在迭代c.c_orders array

若要重複執行o_lineitems屬性，這是一個數組中的數組，你添加多個子句。

```
SELECT o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

AWS Clean Rooms在迭代使用數組時，還支持數組索引AT關鍵字。該條款x AS y AT z遍歷數組x並生成字段z，這是數組索引。

Example

以下查詢會如何運作的資料表。

```
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)
```

Example

下列範例會重複執行標量陣列。

```
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)
```

Example

下面的例子迭代了多個級別的數組。這個範例會使用多個 unnest 子句來重複到最內層的陣列。該f.multi_level_array AS數列查詢會multi_level_array。該陣列AS元素是對陣列的迭代multi_level_array。

```
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)
```

寬鬆的語義

依預設，巢狀資料值的導覽作業會傳回 null，而不是在導覽無效時傳回錯誤。如果巢狀資料值不是物件，或巢狀資料值是物件，但不包含查詢中使用的屬性名稱，則物件導覽無效。

Example

例如，下列查詢會存取巢狀資料行中的無效屬性名稱`c_orders`:

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

如果嵌套數據值不是數組或數組索引超出邊界，數組導航返回 null。

Example

以下查詢會傳回的資料表`c_orders[1][1]`超出範圍。

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

內省的種類

巢狀資料類型資料行支援檢查函數，這些函數會傳回類型以及值的其他類型資訊。AWS 清潔室針對巢狀資料欄支援下列布林函數：

- 十進制精度
- 小數比例
- 是陣列
- 是大

- 是字符
- 為十進位 (_)
- 是浮動
- 是 _ 整數
- 是物件
- 是 (_ 純量)
- 是斯莫林特
- 是 _ 瓦爾恰爾
- JSON_ 類型

如果輸入值為空，所有這些函數返回 false。IS_SCALAR，IS_OBJECT 和 IS_ARRAY 是相互排斥的，並涵蓋除了空值以外的所有可能值。若要推斷與資料對應的類型，AWS Clean Rooms使用 JSON_TYPEOF 函數，該函數會傳回巢狀資料值的類型 (最上層)，如下列範例所示：

```
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
```

AWS Clean Rooms SQL 參考的文件歷史記錄

下表說明「 AWS Clean Rooms SQL 參考」的文件版本。

如需有關此文件更新的通知，您可以訂閱 RSS 摘要。若要訂閱 RSS 更新，您必須為正在使用的瀏覽器啟用 RSS 外掛程式。

變更	描述	日期
SQL 命令和 SQL 函數-更新	已為 JOIN 子句，除了設置運算符，CASE 條件表達式和以下功能添加了示例：ANY_VALUE，NVL 和合併，NULLIF，轉換，轉換，轉換時區，提取，MOD，符號，連接，第一個值和最後值。	2024年2月28日
SQL 函數-更新	AWS Clean Rooms 現在支援下列 SQL 函數：陣列、超級和瓦位元組。現在支援下列數學函數：ACOS、ASIN、ATAN、ATAN2、COT、DEXP、PI、戰俘、弧度和罪。現在支持以下 JSON 函數：CAN_JSON_ 解析，JSON_ 解析和 JSON_ 序列化。	2023 年 10 月 6 日
嵌套數據類型支持	AWS Clean Rooms 現在支援巢狀資料類型。	2023 年 8 月 30 日
SQL 命名規則-更新	僅文檔更改以澄清保留列名。	2023 年 8 月 16 日
一般可用性	「 AWS Clean Rooms SQL 參考」現已正式推出。	2023 年 7 月 31 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。