SQL 参考資料

AWS Clean Rooms

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Clean Rooms: SQL 參考資料

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

中的 SQL 概觀AWS Clean Rooms

歡迎使用 AWS Clean RoomsSQL 參考。

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，包含您用來處理資料庫和資料庫物件的命令和函數。SQL 也會強制執行有關使用資料類型、表達式和常值的規則。

下列主題提供有關此 SQL 參考中使用的慣例和命名規則的一般資訊。

主題

- SQL 參考慣例
- SQL 命名規則
- SQL 引擎的資料類型支援

下列各節提供您可以在其中使用的常值、資料類型、SQL 命令、SQL 函數類型和 SQL 條件的相關資訊AWS Clean Rooms。

- AWS Clean Rooms Spark SQL

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

SQL 參考慣例

本節說明用於撰寫 SQL 表達式、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。

字元	描述
|	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。
'	單引號中的字詞表示您必須輸入引用內容。

SQL 命名規則

下列各節說明 中的 SQL 命名規則 AWS Clean Rooms。

主題

設定的資料表關聯名稱和資料欄

可以查詢的成員使用設定的資料表關聯名稱做為查詢中的資料表名稱。已設定的資料表關聯名稱和已設定的資料表資料欄可以在查詢中別名。

下列命名規則適用於設定的資料表關聯名稱、設定的資料表資料欄名稱和別名：

- 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。
  - (僅限自訂分析規則) 他們可以使用美元符號 ($)，但不能使用遵循美元引用字串常數的模式。

    引用美元的字串常數包含：
    - 美元符號 ($)
    - 零個或多個字元的選用「標籤」
    - 另一個貨幣符號
    - 構成字串內容的任意字元序列
    - 美元符號 ($)
    - 開始美元引號的相同標籤

    - 美元符號

      例如：`$$invalid$$`

- 它們不能包含連續連字號 (-) 字元。
- 它們不能以下列任何字首開頭：

  `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，則必須在雙引號 (" ") 內。
- 如果它們包含連字號 (-) 字元，則必須在雙引號 (" ") 內。
- 長度必須介於 1 到 127 個字元之間。
- 預留單字必須在雙引號 (" ") 內。
- 下列資料欄名稱無法保留在 AWS Clean Rooms （即使是引號） 中使用：
  - oid
  - Tableoid
  - xmin
  - cmin
  - xmax
  - cmax
  - ctid

保留字

以下是 中的預留單字清單 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	TRUNCATECOLUMNSUNION
CURRENT_TIME	INNER	OR	UNIQUE
CURRENT_TIMESTAMP	INTERSECT	ORDER	UNNEST
CURRENT_USER	INTERVAL	OUTER	USING
CURRENT_USER_IDDEFAULT	INTO	OVERLAPS	VERBOSE
DEFERRABLE	IS	PARALLELPARTITION	WALLETWHEN
DEFLATE	ISNULL	PERCENT	WHERE
DEFRAG	JOIN	PERMISSIONS	WITH
DELTA	LANGUAGE	PIVOTPLACING	WITHOUT

SQL 引擎的資料類型支援

AWS Clean Rooms 支援多個 SQL 引擎和方言。了解這些實作的資料類型系統對於成功的資料協作和分析至關重要。下表顯示 AWS Clean Rooms SQL、Snowflake SQL 和 Spark SQL 之間的同等資料類型。

數值資料類型

數值類型代表各種類型的數字，從精確整數到近似浮點值。數值類型的選擇會影響儲存需求和運算精確度。整數類型因位元組大小而異，而小數和浮點類型提供不同的精確度和縮放選項。

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
8 位元組整數	BIGINT	不支援	BIGINT、LONG	從 -9，223，372，036，854

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
				，775，808 到 9，223，372 ，036，854，775，807 的帶正負號整數。
4 位元組整數	INT	不支援	INT、INTEGER	從 -2，147，483，648 到 2，147，483，647 的帶正負號整數
2 位元組整數	SMALLINT	不支援	SMALLINT，簡短	從 -32，768 到 32，767 的帶正負號整數
1 位元組整數	不支援	不支援	TINYINT、BYTE	從 -128 到 127 的帶正負號整數
雙精確度浮點數	雙精度、雙精度	FLOAT，FLOAT4，FLOAT8，DOUBLE，DOUBLE PRECISION，REAL	DOUBLE	8 位元組雙精度浮點數
單一精確度浮點數	REAL、FLOAT	不支援	FLOAT	4 位元組單精度浮點數

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
小數（固定精確度）	DECIMAL	DECIMAL、NUMERIC、NUMBER **Note** Snowflake 會自動將較小寬度的確切數值類型 (INT、BIGINT、SMALLINT 等）別名為 NUMBER。	DECIMAL、NUMERIC、	任意精確度帶正負號的小數位數
小數（含精確度）	DECIMAL(p)	DECIMAL(p)、NUMBER(p)	DECIMAL(p)	固定精確度小數位數
小數（使用擴展）	DECIMAL(p,s)	DECIMAL(p，s)、NUMBER(p，s)	DECIMAL(p,s)	具有擴展的固定精確度小數位數

布林值資料類型

布林值類型代表簡單的 true/false 邏輯值。這些類型在 SQL 引擎之間是一致的，通常用於旗標、條件和邏輯操作。

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
Boolean	BOOLEAN	BOOLEAN	BOOLEAN	代表 true/false 值

日期和時間資料類型

日期和時間類型處理時間資料，精確度和時區感知程度各不相同。這些類型支援儲存日期、時間和時間戳記的不同格式，包括或排除時區資訊的選項。

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
Date	DATE	DATE	DATE	不含時區的日期值（年、月、日）
時間	TIME	不支援	不支援	UTC 的當日時間，不含時區
使用 TZ 的時間	TIMETZ	不支援	不支援	UTC 中的當日時間，含時區
時間戳記	TIMESTAMP	TIMESTAMP、TIMESTAMP_NTZ	TIMESTAMP_NTZ	不含時區的時間戳記 **Note** NTZ 表示「無時區」
使用 TZ 的時間戳記	TIMESTAMPTZ	TIMESTAMP_LTZ	TIMESTAMP、TIMESTAMP_LTZ	具有本機時區的時間戳記

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
				Note LTZ 表示「當地時區」

字元資料類型

字元類型可存放文字資料，同時提供固定長度和可變長度選項。這些類型會處理文字字串和二進位資料，並選用長度規格來控制儲存配置。

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
固定長度字元	CHAR	CHAR、CHARACTER	CHAR、CHARACTER	固定長度的字元字串
長度為 的固定長度字元	CHAR(n)	CHAR(n), CHARACTER(n)	CHAR(n), CHARACTER(n)	指定長度的固定長度字元字串
可變長度字元	VARCHAR	VARCHAR、STRING、TEXT	VARCHAR、STRING	可變長度字元字串
可變長度字元與長度	VARCHAR(n)	VARCHAR(n)、STRING(n)、TEXT(n)	VARCHAR(n)	具有長度限制的可變長度字元字串
二進位	VARBYTE	BINARY、VARBINARY	BINARY	二進位位元組序列

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
具有長度的二進位	VARBYTE(n)	不支援	不支援	具有長度限制的二進位位元組序列

結構化資料類型

結構化類型可將多個值合併為單一欄位，以允許複雜的資料組織。這些包括排序集合的陣列、索引鍵/值對的映射，以及使用具名欄位建立自訂資料結構的結構。

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
陣列	ARRAY<type>	ARRAY（類型）	ARRAY<type>	相同類型的元素順序 **Note** 陣列類型必須包含相同類型的元素
Map	MAP<key，value>	MAP（索引鍵、值）	MAP<key，value>	鍵值對的集合 **Note** 映射類型必須包含相同類型的元素

資料類型	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Description
Struct	STRUCT< field1 : type1 , field2 : type2>	OBJECT(field1 type1 , field2 type2)	STRUCT< field1 : type1 , field2 : type2 >	具有指定類型之具名欄位的結構 **Note** 結構式類型語法在實作之間可能略有不同
超級	SUPER	不支援	不支援	彈性類型支援所有資料類型，包括複雜類型

AWS Clean Rooms Spark SQL

AWS Clean Rooms Spark SQL 會強制執行有關使用資料類型、表達式和常值的規則。

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

下列主題提供有關 AWS Clean Rooms Spark SQL 中支援的常值、資料類型、命令、函數和條件的資訊。

主題

文字

常值或常數是固定的資料值，由一系列的字元或數值常數構成。

AWS Clean Rooms Spark SQL 支援多種類型的常值，包括：

- 整數的數值常值、小數和符點數。
- 字元常值，也稱為字串、字元字串或字元常數，用於指定字元字串值。
- 日期、時間和時間戳記常值，與日期時間資料類型搭配使用。如需詳細資訊，請參閱日期、時間和時間戳記常值。
- 間隔常值。如需詳細資訊，請參閱間隔常值。
- 布林值常值。如需詳細資訊，請參閱布林值常值。
- Null 常值，用於指定 null 值。
- 僅支援 Unicode 一般類別 CARRIAGE RETURN(Cc) 中的 TAB、 LINE FEED(CR) 和 (LF) Unicode 控制字元。

AWS Clean Rooms Spark SQL 不支援直接參考 SELECT 子句中的字串常值，但可在 CAST 等函數中使用。

+ (串連) 運算子

串連數值常值、字串常值和/或日期時間和間隔常值。它們位於 + 符號的任一端，並根據 + 符號任一端的輸入傳回不同的類型。

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

隱含轉換為 的資料DATE欄或表達式DATE。

time

隱含轉換為 的資料TIME欄或表達式TIME。

timetz

隱含轉換為 的資料TIMETZ欄或表達式TIMETZ。

範例

下列範例資料表TIME_TEST具有一欄 TIME_VAL (類型 TIME)，並插入三個值。

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

資料類型

AWS Clean Rooms Spark SQL 存放或擷取的每個值都有具有一組固定關聯屬性的資料類型。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。

下表列出您可以在 AWS Clean Rooms Spark SQL 中使用的資料類型。

資料類型名稱	資料類型	別名	Description
ARRAY	the section called “巢狀類型”	不適用	陣列巢狀資料類型
BIGINT	the section called “數值類型”	不適用	帶正負號的 8 位元組整數
BINARY	the section called “二進位類型”	不適用	位元組序列值
BOOLEAN	the section called “布林值 (Boolean) 類型”	BOOL	邏輯布林值 (true/false)
BYTE	the section called “數值類型”	不適用	1 位元組帶正負號整數，從 -128 到 127
CHAR	the section called “字元類型”	CHARACTER	固定長度的字元字串
DATE	the section called “日期時間 (Datetime) 類型”	不適用	日曆日期 (年、月、日)
DECIMAL	the section called “數值類型”	NUMERIC	可選擇精確度 (有效位數) 的精確數值
FLOAT	the section called “數值類型”	FLOAT8，雙精度	雙精度浮點數

資料類型名稱	資料類型	別名	Description
INTEGER	the section called “數值類型”	INT	帶正負號的 4 位元組整數
INTERVAL	the section called “日期時間 (Datetime) 類型”	不適用	依時間順序或年份到月順序顯示的時間持續時間
LONG	the section called “數值類型”	不適用	8 位元組帶正負號整數
MAP	the section called “巢狀類型”	不適用	映射巢狀資料類型
REAL	the section called “數值類型”	FLOAT4	單精度浮點數
SHORT	the section called “數值類型”	不適用	2 位元組帶正負號整數。
SMALLINT	the section called “數值類型”	不適用	帶正負號的 2 位元組整數
STRUCT	the section called “巢狀類型”	不適用	結構巢狀資料類型
TIMESTAMP_LTZ	the section called “日期時間 (Datetime) 類型”	不適用	當地時區的當日時間
TIMESTAMP_NTZ	the section called “日期時間 (Datetime) 類型”	不適用	一天中沒有時區的時間
TINYINT	the section called “數值類型”	不適用	1 位元組帶正負號整數，從 -128 到 127

資料類型名稱	資料類型	別名	Description
VARCHAR	the section called “字元類型”	角色變動	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，請參閱巢狀類型。

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

數值資料類型包括整數、小數和符點數。

主題

- 整數類型
- DECIMAL 或 NUMERIC 類型
- 浮點類型
- 數值的計算

整數類型

使用下列資料類型來存放各種範圍的整數。您無法將值存放在每種類型的允許範圍之外。

名稱	儲存	範圍
SMALLINT	2 位元組	-32768 到 +32767
SHORT	2 位元組	-32768 到 +32767
INTEGER 或 INT	4 位元組	-2147483648 到 +2147483647
BIGINT	8 位元組	-9223372036854775808 到 9223372036854775807
LONG	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資料欄的精確度減去其比例，則無法將該值複製到資料欄 （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。
> 這些規則的原因如下：
>
> - 具有 19 個或更少有效位數精度的 DECIMAL 值會在內部儲存為 8 位元組整數。
> - 精確度為 20 到 38 個有效位數的 DECIMAL 值會儲存為 16 位元組整數。

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 代表根據浮點數運算的 IEEE 標準 754 的單精度浮點格式。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，運算是指二進位數學運算：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 AWS Clean Rooms Spark SQL 函數。

計算的傳回類型

鑑於 中支援的一組數值資料類型 AWS Clean Rooms，下表顯示新增、減去、乘法和分割操作的預期傳回類型。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。

運算元 1	運算元 2	傳回類型
SMALLINT 或 SHORT	SMALLINT 或 SHORT	SMALLINT 或 SHORT
SMALLINT 或 SHORT	INTEGER	INTEGER
SMALLINT 或 SHORT	BIGINT	BIGINT

運算元 1	運算元 2	傳回類型
SMALLINT 或 SHORT	DECIMAL	DECIMAL
SMALLINT 或 SHORT	FLOAT4	FLOAT8
SMALLINT 或 SHORT	FLOAT8	FLOAT8
INTEGER	INTEGER	INTEGER
INTEGER	BIGINT 或 LONG	BIGINT 或 LONG
INTEGER	DECIMAL	DECIMAL
INTEGER	FLOAT4	FLOAT8
INTEGER	FLOAT8	FLOAT8
BIGINT 或 LONG	BIGINT 或 LONG	BIGINT 或 LONG
BIGINT 或 LONG	DECIMAL	DECIMAL
BIGINT 或 LONG	FLOAT4	FLOAT8
BIGINT 或 LONG	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL
DECIMAL	FLOAT4	FLOAT8
DECIMAL	FLOAT8	FLOAT8
FLOAT4	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。在此表格中， p1 和 s1代表計算中第一個運算元的精確度和比例。 p2和 s2代表第二個運算元的精確度和比例。(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

如果是除法運算，除以 0 的情況會傳回錯誤。

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數指定的請求精確度或規模時，明確轉換會導致執行時間溢位錯誤。例如，您無法從 SALES 資料表 (DECIMAL(8，2) 欄） 中的 PRICEPAID 欄轉換所有值，並傳回 DECIMAL(7，3) 結果：

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

  發生此錯誤是因為 PRICEPAID 欄中某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，而另一個運算元是 DECIMAL 時，INTEGER 運算元會隱含轉換為 DECIMAL。

- SMALLINT 或 SHORT 會轉換為 DECIMAL(5，0)
- INTEGER 轉換為 DECIMAL(10，0)
- BIGINT 或 LONG 轉換為 DECIMAL(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)。

主題

- CHAR 或 CHARACTER
- VARCHAR 或 CHARACTER VARYING
- 多餘空格的意義

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。

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。

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

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)
```

VARCHAR 資料類型是以位元組而非字元定義。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)
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)

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。嘗試將較長的字串存放到這些類型的資料欄會導致錯誤。不過，如果額外的字元是所有空格 （空白），則字串會截斷為長度上限。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

日期時間資料類型包括 DATE、TIME、TIMESTAMP_LTZ 和 TIMESTAMP_NTZ。

主題

DATE

使用 DATE 資料類型來儲存不含時間戳記的簡單日曆日期。

名稱	儲存	範圍	Resolution
DATE	4 位元組	4713 BC 到 294276 AD	1 天

TIMESTAMP_LTZ

使用 TIMESTAMP_LTZ 資料類型來存放完整的時間戳記值，包括日期、日期時間和本機時區。

TIMESTAMP 代表包含欄位 `year`、`month`、、`minute`、 `day` `hour`和 的值`second`，具有工作階段本機時區。此`timestamp`值代表絕對時間點。

Spark 中的 TIMESTAMP 是使用者指定的別名，與其中一個 TIMESTAMP_LTZ 和 TIMESTAMP_NTZ 變化相關聯。您可以透過組態 將預設時間戳記類型設定為 TIMESTAMP_LTZ （預設值） 或 TIMESTAMP_NTZ`spark.sql.timestampType`。

TIMESTAMP_NTZ

使用 TIMESTAMP_NTZ 資料類型來存放完整的時間戳記值，其中包含日期、一天中的時間，不含本機時區。

TIMESTAMP day代表包含欄位 year、month、、hour、 minute和 值的值second。所有操作都會執行，而不會考慮任何時區。

Spark 中的 TIMESTAMP 是使用者指定的別名，與其中一個 TIMESTAMP_LTZ 和 TIMESTAMP_NTZ 變化相關聯。您可以透過組態 將預設時間戳記類型設定為 TIMESTAMP_LTZ （預設值） 或 TIMESTAMP_NTZspark.sql.timestampType。

日期時間 (Datetime) 類型範例

下列範例示範如何使用 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
```

日期、時間和時間戳記常值

以下是使用 AWS Clean Rooms Spark SQL 支援之日期、時間和時間戳記常值的規則。

日期

下表顯示輸入日期，這些日期是您可以載入 AWS Clean Rooms 資料表的常值日期值的有效範例。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
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

下表顯示您可以載入 AWS Clean Rooms 資料表之常值時間值的有效範例輸入時間。

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

特殊的日期時間 (Datetime) 值

下表顯示可用作日期時間常值和用作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	Description
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。

下列範例示範 now和 如何使用 today DATE_ADD 函數。

```
select date_add('today', 1);
```

```
date_add
---------------------
2009-11-17 00:00:00
(1 row)

select date_add('now', 1);

date_add
----------------------------
2009-11-17 10:45:32.021394
(1 row)
```

間隔常值

以下是使用 Spark SQL 支援的 AWS Clean Rooms 間隔常值的規則。

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**
>
> 您無法對 AWS Clean Rooms 資料表中的資料欄使用 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。

下列範例會將 1 秒新增至指定的日期。

```
select caldate + interval '1 second' as dateplus from date
where caldate='12-31-2008';
dateplus
```

```
---------------------
2008-12-31 00:00:01
(1 row)
```

下列範例會將 1 分鐘新增至指定的日期。

```
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 分鐘新增至指定的日期。

```
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 秒新增至指定的日期。

```
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)
```

下列範例會將 12 小時 （半天） 新增至指定的日期。

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

下列範例從 2023 年 3 月 31 日減去 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
```

間隔資料類型和常值

您可以使用間隔資料類型來儲存下列單位的持續時間：seconds、minutes、hours、days、months 和 years。間隔資料類型和常值可用於日期時間計算，例如，將間隔新增至日期和時間戳記、加總間隔，以及從日期或時間戳記減去間隔。間隔常值可作為資料表中間隔資料類型欄的輸入值。

間隔資料類型的語法

若要指定間隔資料類型，以年和月為單位儲存持續時間：

```
INTERVAL year_to_month_qualifier
```

若要指定間隔資料類型，以天、小時、分鐘和秒為單位儲存持續時間：

```
INTERVAL day_to_second_qualifier [ (fractional_precision) ]
```

間隔常值的語法

若要指定間隔常值，以年和月為單位定義持續時間：

```
INTERVAL quoted-string year_to_month_qualifier
```

若要指定間隔常值，以天、小時、分鐘和秒為單位定義持續時間：

```
INTERVAL quoted-string day_to_second_qualifier [ (fractional_precision) ]
```

引數

quoted-string

指定正或負數值，以將數量和日期時間單位指定為輸入字串。如果引號字串只包含數字，則 AWS Clean Rooms 會從 year_to_month_qualifier 或 day_to_second_qualifier 判斷單位。例如，'23' MONTH 代表 1 year 11 months、'-2' DAY 代表 -2 days 0 hours 0 minutes 0.0 seconds、'1-2' MONTH 代表 1 year 2 months，以及 '13 day 1 hour 1 minute 1.123 seconds' SECOND 代表 13 days 1 hour 1 minute 1.123 seconds。如需間隔的輸出格式的詳細資訊，請參閱 間隔樣式。

year_to_month_qualifier

指定間隔的範圍。如果您使用限定詞並以小於限定詞的時間單位建立間隔， 會 AWS Clean Rooms 截斷並捨棄間隔的較小部分。year_to_month_qualifier 的有效值為：

- YEAR
- MONTH
- YEAR TO MONTH

day_to_second_qualifier

指定間隔的範圍。如果您使用限定詞並以小於限定詞的時間單位建立間隔， 會 AWS Clean Rooms 截斷並捨棄間隔的較小部分。day_to_second_qualifier 的有效值為：

- DAY
- HOUR
- MINUTE
- SECOND
- DAY TO HOUR
- DAY TO MINUTE
- DAY TO SECOND
- HOUR TO MINUTE
- HOUR TO SECOND

- `MINUTE TO SECOND`

  INTERVAL 常值的輸出會截斷至指定的最小 INTERVAL 元件。例如，使用 MINUTE 限定詞時，AWS Clean Rooms 會捨棄小於 MINUTE 的時間單位。

```
select INTERVAL '1 day 1 hour 1 minute 1.123 seconds' MINUTE
```

  得到的值會截斷至 `'1 day 01:01:00'`。

fractional_precision

這是選用參數，用於指定間隔中允許的小數位數。只有在您的間隔包含 SECOND 時，才應指定 fractional_precision 引數。例如，`SECOND(3)` 會建立只允許三位小數的間隔，例如 1.234 秒。小數最多為 6 位。

當間隔同時指定了 YEAR TO MONTH 和 DAY TO SECOND 部分時，工作階段組態 `interval_forbid_composite_literals` 會決定是否傳回錯誤。

間隔算術

您可以使用間隔值搭配其他日期時間值來執行算術運算。下表說明可用的運算，以及每項運算產生的資料類型。

> **Note**
>
> 可以同時產生 `date` 和 `timestamp` 結果的運算會根據方程式中涉及的最小時間單位來進行。例如，當您將 `interval` 新增至 `date` 時，如果是 YEAR TO MONTH 間隔，則結果為 `date`，如果是 DAY TO SECOND 間隔，則結果為時間戳記。

第一個運算元為 `interval` 的運算會針對指定的第二個運算元產生下列結果：

運算子	Date	時間戳記	Interval	數值
-	N/A	N/A	Interval	N/A
+	Date	日期/時間戳記	Interval	N/A
*	N/A	N/A	N/A	Interval

運算子	Date	時間戳記	Interval	數值
/	N/A	N/A	N/A	Interval

第一個運算元為 date 的運算會針對指定的第二個運算元產生下列結果：

運算子	Date	時間戳記	Interval	數值
-	數值	Interval	日期/時間戳記	Date
+	N/A	N/A	N/A	N/A

第一個運算元為 timestamp 的運算會針對指定的第二個運算元產生下列結果：

運算子	Date	時間戳記	Interval	數值
-	數值	Interval	時間戳記	時間戳記
+	N/A	N/A	N/A	N/A

間隔樣式

- postgres - 遵循 PostgreSQL 樣式。這是預設值。
- postgres_verbose - 遵循 PostgreSQL 詳細樣式。
- sql_standard - 遵循 SQL 標準間隔常值樣式。

下列命令會將間隔樣式設定為 sql_standard。

```
SET IntervalStyle to 'sql_standard';
```

postgres 輸出格式

以下是 postgres 間隔樣式的輸出格式。每個數值都可以是負數。

```
'<numeric> <unit> [, <numeric> <unit> ...]'
```

```
select INTERVAL '1-2' YEAR TO MONTH::text

varchar
---------------
1 year 2 mons
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
------------------
1 day 02:03:04.5678
```

postgres_verbose 輸出格式

postgres_verbose 語法類似 postgres，但 postgres_verbose 輸出還包含時間單位。

```
'[@] <numeric> <unit> [, <numeric> <unit> ...] [direction]'
```

```
select INTERVAL '1-2' YEAR TO MONTH::text

varchar
-----------------
@ 1 year 2 mons
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
---------------------------
@ 1 day 2 hours 3 mins 4.56 secs
```

sql_standard 輸出格式

間隔年到月的格式設定如下。在間隔之前指定負號表示間隔是負值，並套用至整個間隔。

```
'[-]yy-mm'
```

間隔天到秒的格式設定如下。

```
'[-]dd hh:mm:ss.ffffff'
```

```
SELECT INTERVAL '1-2' YEAR TO MONTH::text

varchar
-------
1-2
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
---------------
1 2:03:04.5678
```

間隔資料類型的範例

下列範例示範如何搭配資料表使用 INTERVAL 資料類型。

```
create table sample_intervals (y2m interval month, h2m interval hour to minute);
insert into sample_intervals values (interval '20' month, interval '2 days
 1:1:1.123456' day to second);
select y2m::text, h2m::text from sample_intervals;


      y2m      |       h2m
---------------+-----------------
 1 year 8 mons | 2 days 01:01:00
```

```
update sample_intervals set y2m = interval '2' year where y2m = interval '1-8' year to
 month;
select * from sample_intervals;

   y2m   |       h2m
---------+-----------------
 2 years | 2 days 01:01:00
```

```
delete from sample_intervals where h2m = interval '2 1:1:0' day to second;
select * from sample_intervals;

 y2m | h2m
-----+-----
```

間隔常值的範例

下列範例會在間隔樣式設定為 postgres 的情況下執行。

下列範例示範如何建立 1 年的 INTERVAL 常值。

```
select INTERVAL '1' YEAR

intervaly2m
---------------
1 years 0 mons
```

如果您指定的 quoted-string 超過限定詞，則會從間隔中截斷剩餘的時間單位。在下列範例中，13 個月的間隔會變成 1 年又 1 個月，但剩餘的 1 個月會因 YEAR 限定詞而被排除。

```
select INTERVAL '13 months' YEAR

intervaly2m
---------------
1 years 0 mons
```

如果您使用的限定詞低於間隔字串，則會將剩餘單位納入。

```
select INTERVAL '13 months' MONTH

intervaly2m
---------------
1 years 1 mons
```

在間隔中指定精確度會將小數位數截斷至指定的精確度。

```
select INTERVAL '1.234567' SECOND (3)

intervald2s
--------------------------------
0 days 0 hours 0 mins 1.235 secs
```

如果您未指定精確度， AWS Clean Rooms 會使用最大精確度 6。

```
select INTERVAL '1.23456789' SECOND
```

```
intervald2s
---------------------------------
0 days 0 hours 0 mins 1.234567 secs
```

下列範例示範如何建立限定範圍的間隔。

```
select INTERVAL '2:2' MINUTE TO SECOND

intervald2s
-----------------------------
0 days 0 hours 2 mins 2.0 secs
```

限定詞規定您要指定的單位。例如，即使下列範例使用與上一個範例相同的 '2：2' 引號字串， 仍會 AWS Clean Rooms 識別它使用不同的時間單位，因為 限定詞。

```
select INTERVAL '2:2' HOUR TO MINUTE

intervald2s
-----------------------------
0 days 2 hours 2 mins 0.0 secs
```

每個單位都支援縮寫和複數。例如，5s、5 second 和 5 seconds 是相等的間隔。支援的單位包括年、月、小時、分鐘和秒。

```
select INTERVAL '5s' SECOND

intervald2s
-----------------------------
0 days 0 hours 0 mins 5.0 secs
```

```
select INTERVAL '5 HOURS' HOUR

intervald2s
-----------------------------
0 days 5 hours 0 mins 0.0 secs
```

```
select INTERVAL '5 h' HOUR

intervald2s
-----------------------------
```

```
0 days 5 hours 0 mins 0.0 secs
```

未使用限定詞語法的間隔常值範例

> **Note**
>
> 下列範例示範使用未包含 YEAR TO MONTH 或 DAY TO SECOND 限定詞的間隔常值。如需使用包含限定詞的建議間隔常值的相關資訊，請參閱 間隔資料類型和常值。

使用間隔常值來表示指定的時間期間，例如 12 hours 或 6 months。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。

間隔常值的表示方式，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)。

下列範例顯示不同間隔值的一連串計算。

以下內容為指定日期增加 1 秒。

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

以下內容為指定日期增加 1 分鐘。

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

以下內容為指定日期增加 3 個小時又 35 分鐘。

```
select caldate + interval '3 hours, 35 minutes' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 03:35:00
(1 row)
```

以下內容為指定日期增加 52 週。

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

以下內容為指定日期增加 1 週 1 小時 1 分鐘又 1 秒。

```
select caldate + interval '1w, 1h, 1m, 1s' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2009-01-07 01:01:01
(1 row)
```

以下內容為指定指定日期增加 12 個小時 (半天)。

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

以下內容從 2023 年 2 月 15 日減去 4 個月，結果是 2022 年 10 月 15 日。

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

?column?
---------------------
```

```
2022-10-15 00:00:00
```

以下內容從 2023 年 3 月 31 日減去 4 個月，結果是 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
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，將每位客戶的「作用中/非作用中」狀態存放在 CUSTOMER 資料表中。

```
select * from customer;
custid | active_flag
```

```
-------+--------------
   100 | t
```

在此範例中，下列查詢會從 USERS 資料表中選取喜歡運動但不喜歡戲劇的使用者：

```
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
```

下列範例成功，因為它在 SELECT 清單中使用相等比較 (=)，而不是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    |
```

布林值常值

下列規則適用於使用 Spark SQL 支援的 AWS Clean Rooms 布林值常值。

使用布林值常值來指定布林值，例如 TRUE或 FALSE。

語法

```
TRUE | FALSE
```

範例

下列範例顯示指定值為 TRUE 的資料欄。

```
SELECT TRUE AS col;
+----+
| col|
+----+
|true|
+----+
```

二進位類型

使用 BINARY 資料類型來存放和管理固定長度、未解譯的二進位資料，為特定使用案例提供高效的儲存和比較功能。

無論要存放的資料實際長度為何，BINARY 資料類型都會存放固定的位元組數。長度上限為 255 個位元組。

BINARY 用於存放原始、未解譯的二進位資料，例如影像、文件或其他類型的檔案。資料的存放方式與提供的完全相同，不會有任何字元編碼或解譯。根據實際的二進位值，而不是任何字元編碼或定序規則，比較和排序存放在 BINARY 資料欄中的二進位資料byte-by-byte。

下列範例查詢顯示字串 的二進位表示法"abc"。字串中的每個字元都以十六進位格式的 ASCII 程式碼表示："a" 是 0x61，"b" 是 0x62，而 "c" 是 0x63。合併時，這些十六進位值會形成二進位表示法 "616263"。

```
SELECT 'abc'::binary;
binary
---------
 616263
```

巢狀類型

AWS Clean Rooms 支援涉及巢狀資料類型資料的查詢，特別是 AWS Glue STRUCT、ARRAY 和 MAP 資料欄類型。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式結構。

巢狀資料類型包含參考資料中不同實體的標籤。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。巢狀資料類型支援個別巢狀資料類型欄位或物件最多 1 MB 的資料。

主題

ARRAY 類型

使用 ARRAY 類型來表示包含具有 類型之元素序列的值elementType。

```
array(elementType, containsNull)
```

使用 containsNull 指示 ARRAY 類型中的元素是否可以有null值。

MAP 類型

使用 MAP 類型來表示包含一組鍵值對的值。

```
map(keyType, valueType, valueContainsNull)
```

keyType：金鑰的資料類型

valueType：值的資料類型

金鑰不允許具有null值。使用 valueContainsNull 指示 MAP 類型值的值是否可以有null值。

STRUCT 類型

使用 STRUCT 類型以 StructFields （ 欄位 ） 序列描述的結構來表示值。

```
struct(name, dataType, nullable)
```

StructField(name， dataType， nullable)：代表 StructType 中的欄位。

dataType：資料類型欄位

name：欄位的名稱

使用 nullable 指示這些欄位的值是否可以有null值。

巢狀資料類型的範例

對於 `struct<given:varchar, family:varchar>`類型，有兩個屬性名稱： `given`和 `family`，每個對應至一個`varchar`值。

對於 `array<varchar>`類型，陣列指定為 的清單`varchar`。

`array<struct<shipdate:timestamp, price:double>>` 類型是指具有 `struct<shipdate:timestamp, price:double>` 類型的元素清單。

`map` 資料類型的行為類似於 `array`的 `structs`，其中陣列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)>`類型視為 `array<struct<key:varchar(20), value:varchar(20)>>`，其中 `key`和 `value`參考基礎資料中映射的屬性。

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)。

如需如何使用查詢的 FROM 子句導覽陣列來 AWS Clean Rooms 啟用陣列上的反覆運算的資訊，請參閱 解除巢狀化查詢。

類型相容性與轉換

下列主題說明類型轉換規則和資料類型相容性如何在 AWS Clean Rooms Spark SQL 中運作。

主題

- 相容性
- 一般相容性與轉換規則
- 隱含轉換類型

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。相容性暗示並不一定需要針對某些值和某些資料類型，進行一對一的比對。由於某些資料類型相容，因此可能會有隱含轉換或強制。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。
- 當您試圖插入超出範圍的值時，數值資料類型會強制讓溢位狀況發生。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。整數或小數的整個部分永遠不會截斷。不過，適當時，小數的分數部分可以四捨五入或捨去。不過，從資料表所選取值的明確轉換結果，不會四捨五入。
- 不同類型的字元字串相容。包含單位元組資料和 CHAR 資料欄字串的 VARCHAR 資料欄字串相當且隱含可轉換。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、時間戳記或數值。會忽略任何開頭或結尾的空格。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'`或 `'5.9'` 轉換為十進位值，但無法將字串轉換為`'ABC'`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。

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
- 表達式中的隱含轉換，例如在 WHERE 子句中執行比較

下表列出可在指派或表達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 或 SHORT
	VARCHAR
CHAR	VARCHAR
DATE	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT 或 LONG
	CHAR

轉換前的類型	轉換後的類型
	DOUBLE PRECISION (FLOAT8)
	INTEGER INT)
	REAL (FLOAT4)
	SMALLINT 或 SHORT
	VARCHAR
DOUBLE PRECISION (FLOAT8)	BIGINT 或 LONG
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	REAL (FLOAT4)
	SMALLINT 或 SHORT
	VARCHAR
INTEGER (INT)	BIGINT 或 LONG
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	REAL (FLOAT4)
	SMALLINT 或 SHORT
	VARCHAR

轉換前的類型	轉換後的類型
REAL (FLOAT4)	BIGINT 或 LONG
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	SMALLINT 或 SHORT
	VARCHAR
SMALLINT	BIGINT 或 LONG
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT)
	REAL (FLOAT4)
	VARCHAR
TIME	VARCHAR
	TIMETZ

Note

DATE、TIME、TIMESTAMP_LTZ、TIMESTAMP_NTZ 或字元字串之間的隱含轉換會使用目前的工作階段時區。
VARBYTE 資料類型無法隱含轉換至任何其他資料類型。如需詳細資訊，請參閱CAST 函數。

AWS Clean Rooms Spark SQL 命令

Spark SQL 支援 AWS Clean Rooms 下列 SQL 命令：

主題

CACHE 資料表

CACHE TABLE 命令會快取現有資料表的資料，或建立和快取包含查詢結果的新資料表。

> **Note**
>
> 快取的資料會在整個查詢中保留。

語法、引數和一些範例來自 Apache Spark SQL 參考。

語法

CACHE TABLE 命令支援三種語法模式：

使用 AS （不含括號）：根據查詢結果建立和快取新資料表。

```
CACHE TABLE cache_table_identifier AS query;
```

使用 AS 和括號：函數與第一個語法類似，但使用括號明確分組查詢。

```
CACHE TABLE cache_table_identifier AS ( query );
```

不使用 AS：使用 SELECT 陳述式來篩選要快取的資料列，以快取現有的資料表。

```
CACHE TABLE cache_table_identifier query;
```

其中：

- 所有陳述式應以分號 (;) 結尾

- query 通常是 SELECT 陳述式
- 查詢的括號是選用的 AS
- AS 關鍵字為選用

Parameters

cache_table_identifier

快取資料表的名稱。可以包含選用的資料庫名稱限定詞。

AS

從查詢結果建立和快取新資料表時使用的關鍵字。

query

SELECT 陳述式或其他查詢，定義要快取的資料。

範例

在下列範例中，快取的資料表會針對整個查詢持續存在。快取後，參考 *cache_table_identifier* 的後續查詢將從快取版本讀取，而不是重新計算或讀取 *sourceTable*。這可以改善經常存取資料的查詢效能。

從查詢結果建立和快取篩選的資料表

第一個範例示範如何從查詢結果建立和快取新資料表。此命令使用AS關鍵字，而沒有SELECT陳述式的括號。它會建立一個名為 'cache_table_identifier' 的新資料表，只包含來自 'sourceTable' 的資料列，其中狀態為 'active'。它會執行查詢、將結果存放在新資料表中，以及快取新資料表的內容。原始 'sourceTable' 保持不變，後續查詢必須參考 'cache_table_identifier' 才能使用快取的資料。

```
CACHE TABLE cache_table_identifier AS
    SELECT * FROM sourceTable
    WHERE status = 'active';
```

使用括號 SELECT 陳述式快取查詢結果

第二個範例示範如何在 SELECT 陳述式周圍使用括號，將查詢的結果快取為具有指定名稱 (cache_table_identifier) 的新資料表。此命令會建立一個名為 'cache_table_identifier' 的新資料表，只包含來自 'sourceTable' 的資料列，其中狀態為 'active'。它會執行查詢、將結果

存放在新資料表中，以及快取新資料表的內容。原始 'sourceTable' 保持不變。後續查詢必須參考「cache_table_identifier」才能使用快取的資料。

```
CACHE TABLE cache_table_identifier AS (
    SELECT * FROM sourceTable
    WHERE status = 'active'
);
```

快取具有篩選條件的現有資料表

第三個範例示範如何使用不同的語法快取現有的資料表。此語法會省略 'AS' 關鍵字和括號，通常會從名為 'cache_table_identifier' 的現有資料表快取指定的資料列，而不是建立新的資料表。SELECT 陳述式做為篩選條件，以決定要快取的資料列。

> **Note**
>
> 此語法的確切行為因資料庫系統而異。請務必驗證特定 AWS 服務的正確語法。

```
CACHE TABLE cache_table_identifier
SELECT * FROM sourceTable
WHERE status = 'active';
```

提示

SQL 分析的提示提供最佳化指令，可引導 中的查詢執行策略 AWS Clean Rooms，讓您改善查詢效能並降低運算成本。提示會建議 Spark 分析引擎應如何產生其執行計畫。

語法

```
SELECT /*+ hint_name(parameters), hint_name(parameters) */ column_list
FROM table_name;
```

提示使用註解樣式語法內嵌在 SQL 查詢中，且必須直接放置在 SELECT 關鍵字之後。

支援的提示類型

AWS Clean Rooms 支援兩種類型的提示：聯結提示和分割提示。

主題

聯結提示

聯結提示會建議查詢執行的聯結策略。語法、引數和一些範例來自 Apache Spark SQL 參考，以取得詳細資訊

BROADCAST

建議使用 AWS Clean Rooms 廣播聯結。無論 autoBroadcastJoinThreshold 為何，都會廣播具有提示的聯結端。如果聯結的兩端都有廣播提示，則具有較小大小（根據統計資料）的那一側將廣播。

別名：BROADCASTJOIN、MAPJOIN

參數：資料表識別符（選用）

範例：

```
-- Broadcast a specific table
SELECT /*+ BROADCAST(students) */ e.name, s.course
FROM employees e JOIN students s ON e.id = s.id;

-- Broadcast multiple tables
SELECT /*+ BROADCASTJOIN(s, d) */ *
FROM employees e
JOIN students s ON e.id = s.id
JOIN departments d ON e.dept_id = d.id;
```

MERGE

建議使用 AWS Clean Rooms 隨機排序合併聯結。

別名：SHUFFLE_MERGE、MERGEJOIN

參數：資料表識別符（選用）

範例：

```
-- Use merge join for a specific table
SELECT /*+ MERGE(employees) */ *
FROM employees e JOIN students s ON e.id = s.id;
```

```
-- Use merge join for multiple tables
SELECT /*+ MERGEJOIN(e, s, d) */ *
FROM employees e
JOIN students s ON e.id = s.id
JOIN departments d ON e.dept_id = d.id;
```

SHUFFLE_HASH

建議使用 AWS Clean Rooms 隨機雜湊聯結。如果兩端都有隨機雜湊提示，則查詢最佳化工具會選擇較小的端（根據統計資料）做為建置端。

參數：資料表識別符（選用）

範例：

```
-- Use shuffle hash join
SELECT /*+ SHUFFLE_HASH(students) */ *
FROM employees e JOIN students s ON e.id = s.id;
```

SHUFFLE_REPLICATE_NL

建議使用 AWS Clean Rooms shuffle-and-replicate巢狀迴圈聯結。

參數：資料表識別符（選用）

範例：

```
-- Use shuffle-replicate nested loop join
SELECT /*+ SHUFFLE_REPLICATE_NL(students) */ *
FROM employees e JOIN students s ON e.id = s.id;
```

Spark SQL 中的提示疑難排解

下表顯示未在 SparkSQL 中套用提示的常見案例。如需其他資訊，請參閱 the section called “考量和限制”。

使用案例	查詢範例
找不到資料表參考	`SELECT /*+ BROADCAST(fake_table) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`

使用案例	查詢範例
資料表未參與聯結操作	`SELECT /*+ BROADCAST(s) */ * FROM students s WHERE s.age > 25;`
巢狀子查詢中的資料表參考	`SELECT /*+ BROADCAST(s) */ * FROM employees e INNER JOIN (SELECT * FROM students s WHERE s.age > 20) sub ON e.eid = sub.sid;`
資料欄名稱而非資料表參考	`SELECT /*+ BROADCAST(e.eid) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
沒有必要參數的提示	`SELECT /*+ BROADCAST */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
基礎資料表名稱而非資料表別名	`SELECT /*+ BROADCAST(employees) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`

分割提示

分割提示控制執行器節點之間的資料分佈。指定多個分割提示時，會將多個節點插入邏輯計畫，但最左邊的提示是由最佳化工具挑選。

COALESCE

將分割區數量減少為指定的分割區數量。

參數：數值 （必要） - 必須是介於 1 和 2147483647 之間的正整數

範例：

```
-- Reduce to 5 partitions
```

```
SELECT /*+ COALESCE(5) */ employee_id, salary
FROM employees;
```

保留

使用指定的分割表達式，將資料重新分割至指定的分割區數量。使用循環配置分佈。

參數：

- 數值 （選用） - 分割區數量；必須是介於 1 和 2147483647 之間的正整數
- 資料欄識別符 （選用） - 要分割的資料欄；這些資料欄必須存在於輸入結構描述中。
- 如果同時指定兩者，數值必須首先出現

範例：

```
-- Repartition to 10 partitions
SELECT /*+ REPARTITION(10) */ *
FROM employees;

-- Repartition by column
SELECT /*+ REPARTITION(department) */ *
FROM employees;

-- Repartition to 8 partitions by department
SELECT /*+ REPARTITION(8, department) */ *
FROM employees;

-- Repartition by multiple columns
SELECT /*+ REPARTITION(8, department, location) */ *
FROM employees;
```

REPARTITION_BY_RANGE

使用指定資料欄上的範圍分割，將資料重新分割至指定數量的分割區。

參數：

- 數值 （選用） - 分割區數量；必須是介於 1 和 2147483647 之間的正整數
- 資料欄識別符 （選用） - 要分割的資料欄；這些資料欄必須存在於輸入結構描述中。
- 如果同時指定兩者，數值必須首先出現

範例：

```
SELECT /*+ REPARTITION_BY_RANGE(10) */ *
FROM employees;

-- Repartition by range on age column
SELECT /*+ REPARTITION_BY_RANGE(age) */ *
FROM employees;

-- Repartition to 5 partitions by range on age
SELECT /*+ REPARTITION_BY_RANGE(5, age) */ *
FROM employees;

-- Repartition by range on multiple columns
SELECT /*+ REPARTITION_BY_RANGE(5, age, salary) */ *
FROM employees;
```

REBALANCE

重新平衡查詢結果輸出分割區，讓每個分割區的大小合理（不會太小也不會太大）。這是最佳嘗試操作：如果有扭曲， AWS Clean Rooms 會分割扭曲的分割區，使其不會太大。當您需要將查詢結果寫入資料表，以避免檔案太小或太大時，此提示很有用。

參數：

- 數值（選用）- 分割區數量；必須是介於 1 和 2147483647 之間的正整數
- 資料欄識別符（選用）- 資料欄必須出現在 SELECT 輸出清單中
- 如果同時指定兩者，數值必須首先出現

範例：

```
-- Rebalance to 10 partitions
SELECT /*+ REBALANCE(10) */ employee_id, name
FROM employees;

-- Rebalance by specific columns in output
SELECT /*+ REBALANCE(employee_id, name) */ employee_id, name
FROM employees;

-- Rebalance to 8 partitions by specific columns
SELECT /*+ REBALANCE(8, employee_id, name) */ employee_id, name, department
```

```
FROM employees;
```

結合多個提示

您可以在單一查詢中指定多個提示，方法是使用逗號分隔它們：

```
-- Combine join and partitioning hints
SELECT /*+ BROADCAST(d), REPARTITION(8) */ e.name, d.dept_name
FROM employees e JOIN departments d ON e.dept_id = d.id;

-- Multiple join hints
SELECT /*+ BROADCAST(s), MERGE(d) */ *
FROM employees e
JOIN students s ON e.id = s.id
JOIN departments d ON e.dept_id = d.id;

-- Hints within separate hint blocks within the same query
SELECT /*+ REPARTITION(100) */ /*+ COALESCE(500) */ /*+ REPARTITION_BY_RANGE(3, c) */ *
 FROM t;
```

考量和限制

- 提示是最佳化建議，而不是命令。查詢最佳化工具可能會根據資源限制或執行條件忽略提示。
- 提示直接內嵌在 CreateAnalysisTemplate 和 StartProtectedQuery APIs SQL 查詢字串中。
- 提示必須直接放在 SELECT 關鍵字後面。
- 使用提示不支援具名參數，並會擲回例外狀況。
- REPARTITION amd REPARTITION_BY_RANGE 提示中的資料欄名稱必須存在於輸入結構描述中。
- REBALANCE 提示中的資料欄名稱必須出現在 SELECT 輸出清單中。
- 數值參數必須是介於 1 和 2147483647 之間的正整數。不支援 1e1 等科學表示法
- 差異隱私權 SQL 查詢不支援提示。
- PySpark 任務不支援 SQL 查詢的提示。若要在 PySpark 任務中提供執行計畫的指令，請使用資料框架 API。如需詳細資訊，請參閱 Apache Spark DataFrame API 文件。

SELECT

SELECT 命令會從資料表和使用者定義的函數傳回資料列。

AWS Clean Rooms Spark SQL 支援下列 SELECT SQL 命令、子句和集合運算子：

主題

- SELECT list
- WITH 子句
- FROM 子句
- JOIN 子句
- WHERE 子句
- VALUES 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

語法、引數和一些範例來自 Apache Spark SQL 參考。

SELECT list

您要查詢傳回的資料欄、函數和表達式SELECT list名稱。清單查詢的輸出。

語法

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

Parameters

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。AS 關鍵字是選用的。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中定義的其他表達式中使用別名。

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 common_table_expression 可以是非遞迴的。以下是非遞迴形式：

```
CTE_table_name AS ( query )
```

Parameters

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

AWS Clean Rooms 支援的任何 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：

- SELECT、 WITH、UNION、UNION ALL、INTERSECT、INTERSECT ALL、EXCEPT 或 EXCEPT ALL

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
- Parameters
- 使用須知

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
```

Parameters

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 關鍵字一律為選用。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
```

如果您定義了資料表別名，則必須使用別名在查詢中參考該資料表。

例如，如果查詢是 SELECT "tbl"."col" FROM "tbl" AS "t"，則查詢會失敗，因為資料表名稱現在基本上已覆寫。在這種情況下，有效的查詢會是 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 子句

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。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。

JOIN 類型和聯結條件的組合會決定哪些資料列包含在最終結果集中。然後 SELECT 和 WHERE 子句會控制傳回哪些資料欄，以及如何篩選資料列。了解不同的 JOIN 類型以及如何有效地使用它們是 SQL 的關鍵技能，因為它可讓您以靈活且強大的方式結合來自多個資料表的資料。

語法

```
SELECT column1, column2, ..., columnn
FROM table1
join_type table2
ON table1.column = table2.column;
```

Parameters

SELECT 欄 1、欄 2、...、columnN

您要包含在結果集中的資料欄。您可以從 JOIN 中涉及的其中一個或兩個資料表中選取資料欄。

FROM 資料表 1

JOIN 操作中的第一個（左）資料表。

【加入 | 內部加入 | 左側 【外部】 加入 | 右側 【外部】 加入 | 完整 【外部】 加入】 資料表 2：

要執行的 JOIN 類型。JOIN 或 INNER JOIN 只會傳回兩個資料表中具有相符值的資料列。

LEFT 【OUTER】 JOIN 會傳回左側資料表中的所有資料列，以及右側資料表中的相符資料列。

RIGHT 【OUTER】 JOIN 會傳回右側資料表中的所有資料列，以及左側資料表中的相符資料列。

FULL 【OUTER】 JOIN 會傳回兩個資料表中的所有資料列，無論是否相符。

CROSS JOIN 會從兩個資料表建立資料列的笛卡爾產品。

ON table1.column = table2.column

聯結條件，指定如何比對兩個資料表中的資料列。聯結條件可以根據一個或多個資料欄。

WHERE 條件：

選用子句，可用於根據指定的條件進一步篩選結果集。

範例

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
```

聯結類型

INNER

這是預設聯結類型。傳回兩個資料表參考中具有相符值的資料列。

INNER JOIN 是 SQL 中使用的最常見聯結類型。這是根據通用資料欄或一組資料欄合併多個資料表資料的強大方式。

語法:

```
SELECT column1, column2, ..., columnn
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

下列查詢將傳回客戶和訂單資料表之間存在相符 customer_id 值的所有資料列。結果集將包含 customer_id、name、order_id 和 order_date 資料欄。

```
SELECT customers.customer_id, customers.name, orders.order_id, orders.order_date
FROM customers
INNER JOIN orders
ON customers.customer_id = orders.customer_id;
```

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
```

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
```

左側 【外部 】

傳回左側資料表參考的所有值，以及右側資料表參考的相符值，如果沒有相符項目，則附加 NULL。它也稱為左側外部聯結。

它會傳回左側（第一個）資料表的所有資料列，以及右側（第二個）資料表的相符資料列。如果右側資料表中沒有相符項目，則結果集將包含右側資料表中資料欄的 NULL 值。您可以省略 OUTER 關鍵字，而且可以直接將聯結寫入 LEFT JOIN。與 LEFT OUTER JOIN 相反的是 RIGHT OUTER JOIN，它會傳回右側資料表的所有資料列，以及左側資料表的相符資料列。

語法:

```
SELECT column1, column2, ..., columnn
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

下列查詢會傳回客戶資料表中的所有資料列，以及訂單資料表中的相符資料列。如果客戶沒有訂單，結果集仍會包含該客戶的資訊，以及 order_id 和 order_date 資料欄的 NULL 值。

```
SELECT customers.customer_id, customers.name, orders.order_id, orders.order_date
FROM customers
LEFT OUTER JOIN orders
ON customers.customer_id = orders.customer_id;
```

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 LISTIDs2 和 3 不會產生任何銷售。

```
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
```

右 【 外部 】

傳回右側資料表參考的所有值，以及左側資料表參考的相符值，如果沒有相符項目，則附加 NULL。它也稱為正確的外部聯結。

它會傳回右側（秒）資料表中的所有資料列，以及左側（第一個）資料表的相符資料列。如果左側資料表中沒有相符項目，則結果集將包含左側資料表中資料欄的 NULL 值。您可以省略 OUTER 關鍵字，而且可以直接將聯結寫入 RIGHT JOIN。RIGHT OUTER JOIN 與 RIGHT OUTER JOIN 相反是 LEFT OUTER JOIN，它會傳回左側資料表的所有資料列，以及右側資料表的相符資料列。

語法:

```
SELECT column1, column2, ..., columnn
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

下列查詢會傳回客戶資料表中的所有資料列，以及訂單資料表中的相符資料列。如果客戶沒有訂單，結果集仍會包含該客戶的資訊，以及 order_id 和 order_date 資料欄的 NULL 值。

```
SELECT orders.order_id, orders.order_date, customers.customer_id, customers.name
FROM orders
RIGHT OUTER JOIN customers
ON orders.customer_id = customers.customer_id;
```

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
```

完整 【外部】

傳回兩個關係的所有值，並在沒有相符項目的 端附加 NULL 值。它也稱為完整的外部聯結。

它會傳回左側和右側資料表中的所有資料列，無論是否相符。如果沒有相符項目，則結果集將包含資料表中沒有相符資料列之資料欄的 NULL 值。您可以省略 OUTER 關鍵字，而且可以直接將聯結寫入 FULL JOIN。FULL OUTER JOIN 比 LEFT OUTER JOIN 或 RIGHT OUTER JOIN 更不常見，但在您需要查看兩個資料表中所有資料的某些情況下，即使沒有相符項目，它也很有用。

語法:

```
SELECT column1, column2, ..., columnn
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

下列查詢將傳回來自客戶和訂單資料表的所有資料列。如果客戶沒有訂單，結果集仍會包含該客戶的資訊，以及 order_id 和 order_date 資料欄的 NULL 值。如果訂單沒有相關聯的客戶，結果集將包含該訂單，其中包含 customer_id 和名稱欄的 NULL 值。

```
SELECT customers.customer_id, customers.name, orders.order_id, orders.order_date
FROM customers
FULL OUTER JOIN orders
ON customers.customer_id = orders.customer_id;
```

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 LISTIDs2 和 3 不會產生任何銷售。

```
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
```

【 左側 】 SEMI

從與右側相符的資料表參考左側傳回值。它也稱為左半聯結。

它只會傳回左側 （第一個） 資料表中具有右側 （第二個） 資料表中相符資料列的資料列。它不會傳回右側資料表中的任何資料欄 - 只會傳回左側資料表中的資料欄。當您想要在一個資料表中找到符合另一個資料表的資料列時，不需要從第二個資料表傳回任何資料，LEFT SEMI JOIN 會很有用。LEFT SEMI JOIN 是將子查詢與 IN 或 EXISTS 子句搭配使用時更有效率的替代方案。

語法:

```
SELECT column1, column2, ..., columnn
FROM table1
LEFT SEMI JOIN table2
ON table1.column = table2.column;
```

對於訂單資料表中至少有一個訂單的客戶，下列查詢只會傳回客戶資料表中的 customer_id 和名稱資料欄。結果集不會包含訂單資料表中的任何資料欄。

```
SELECT customers.customer_id, customers.name
FROM customers
LEFT SEMI JOIN orders
```

```
ON customers.customer_id = orders.customer_id;
```

CROSS JOIN

傳回兩個關係的笛卡爾產品。這表示結果集將包含來自兩個資料表的所有可能資料列組合，而不會套用任何條件或篩選條件。

當您需要從兩個資料表產生所有可能的組合資料時，例如建立報告以顯示客戶和產品資訊的所有可能組合時，CROSS JOIN 非常有用。CROSS JOIN 與其他聯結類型 (INNER JOIN、LEFT JOIN 等） 不同，因為它在 ON 子句中沒有聯結條件。CROSS JOIN 不需要加入條件。

語法:

```
SELECT column1, column2, ..., columnn
FROM table1
CROSS JOIN table2;
```

下列查詢會傳回結果集，其中包含來自客戶和產品資料表的所有可能 customer_id、Customer_name、product_id 和 product_name 組合。如果客戶資料表有 10 個資料列，而產品資料表有 20 個資料列，則 CROSS JOIN 的結果集將包含 10 x 20 = 200 個資料列。

```
SELECT customers.customer_id, customers.name, products.product_id,
 products.product_name
FROM customers
CROSS JOIN products;
```

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
```

ANTI 加入

傳回左側資料表參考中與右側資料表參考不相符的值。它也稱為左反聯結。

當您想要在一個資料表中尋找資料列，而另一個資料表中沒有相符項目時，ANTI JOIN 是一項有用的操作。

語法:

```
SELECT column1, column2, ..., columnn
FROM table1
LEFT ANTI JOIN table2
ON table1.column = table2.column;
```

下列查詢將傳回所有尚未下訂單的客戶。

```
SELECT customers.customer_id, customers.name
FROM customers
LEFT ANTI JOIN orders
ON customers.customer_id = orders.customer_id
WHERE orders.order_id IS NULL;
```

NATURAL

指定兩個關係中的資料列將隱含地與具有相符名稱的所有資料欄的相等性相符。

它會自動比對兩個資料表之間具有相同名稱和資料類型的資料欄。它不需要您在 ON 子句中明確指定聯結條件。它將兩個資料表之間的所有相符資料欄合併為結果集。

當您要聯結的資料表具有具有相同名稱和資料類型的資料欄時，NATURAL JOIN 非常方便。不過，通常建議使用更明確的 INNER JOIN ... ON 語法可讓聯結條件更明確且更容易理解。

語法:

```
SELECT column1, column2, ..., columnn
FROM table1
NATURAL JOIN table2;
```

下列範例是兩個資料表 employees和 之間的自然聯結departments，具有下列資料欄：

- employees 資料表：employee_id、first_name、last_name、 department_id
- departments 資料表：department_id、 department_name

下列查詢會根據資料department_id欄傳回結果集，其中包含兩個資料表之間所有相符資料列的名字、姓氏和部門名稱。

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
NATURAL JOIN departments d;
```

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
```

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)
```

VALUES 子句

VALUES 子句用於直接在查詢中提供一組資料列值，而不需要參考資料表。

VALUES 子句可用於下列案例：

- 您可以在 INSERT INTO 陳述式中使用 VALUES 子句來指定要插入資料表之新資料列的值。
- 您可以自行使用 VALUES 子句來建立暫時結果集或內嵌資料表，而不需要參考資料表。
- 您可以結合 VALUES 子句與其他 SQL 子句，例如 WHERE、ORDER BY 或 LIMIT，以篩選、排序或限制結果集中的資料列。

當您需要直接在 SQL 陳述式中插入、查詢或操作一小組資料，而不需要建立或參考永久資料表時，此子句特別有用。它可讓您定義資料欄名稱和每一列的對應值，讓您靈活地即時建立臨時結果集或插入資料，而無需管理個別資料表的額外負荷。

語法

```
VALUES ( expression [ , ... ] ) [ table_alias ]
```

Parameters

表達式

指定導致值的一或多個值、運算子和 SQL 函數的組合的表達式。

table_alias

指定具有選用資料欄名稱清單的暫時名稱的別名。

範例

下列範例會建立內嵌資料表、具有兩個資料欄的暫存資料表類似結果集，col1以及 col2。結果集中的單一資料列1分別包含值 "one"和 。查詢SELECT * FROM的部分只會從此暫時結果集中擷取所有資料欄和資料列。資料庫系統會自動產生資料欄名稱 (col1 和 col2)，因為 VALUES 子句不會明確指定資料欄名稱。

```
SELECT * FROM VALUES ("one", 1);
+----+----+
|col1|col2|
+----+----+
| one|   1|
+----+----+
```

如果您想要定義自訂資料欄名稱，您可以在 VALUES 子句後面使用 AS 子句來執行此操作，如下所示：

```
SELECT * FROM (VALUES ("one", 1)) AS my_table (name, id);
+------+----+
| name | id |
+------+----+
| one  |  1 |
+------+----+
```

這會建立具有資料欄名稱 name和 的暫時結果集id，而不是預設 col1和 col2。

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 表達式中存在彙總函數，則 SELECT 表達式中不在彙總函數中的任何資料欄都必須在 GROUP BY 子句中。

如需詳細資訊，請參閱AWS Clean Rooms Spark SQL 函數。

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 ROLLUP，在單一陳述式中執行多個 GROUP BY 操作的工作。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 操作的工作。

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

集合運算子用於比較和合併兩個個別查詢表達式的結果。

AWS Clean Rooms Spark SQL 支援下表所列的下列一組運算子。

設定運算子
INTERSECT
INTERSECT ALL
EXCEPT
全部除外
UNION
UNION ALL

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**
>
> ORDER BY、LIMIT、SELECT TOP 和 OFFSET 子句不能用於 UNION、UNION ALL、INTERSECT 和 EXCEPT 集運算子合併的查詢表達式。

主題

- 語法
- Parameters
- 集合運算子的評估順序

- 使用須知
- 範例 UNION 查詢
- 範例 UNION ALL 查詢
- 範例 INTERSECT 查詢
- 範例 EXCEPT 查詢

語法

```
subquery1
{ { UNION [ ALL | DISTINCT ] |
              INTERSECT [ ALL | DISTINCT ] |
              EXCEPT [ ALL | DISTINCT ] } subquery2 } [...] }
```

Parameters

subquery1、subquery2

查詢表達式，以其選取清單的形式對應至 UNION、UNION ALL、INTERSECT、INTERSECT ALL、EXCEPT 或 EXCEPT ALL 運算子後面的第二個查詢表達式。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。設定操作不允許在不同資料類型類別之間進行隱含轉換。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 【全部 | DISTINCT】

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。

交集 【全部 | 不同】

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 【全部 | DISTINCT】

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。

EXCEPT ALL 不會從結果列中移除重複項目。

MINUS 和 EXCEPT 是一模一樣的同義詞。

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 AWS Clean Rooms 精確度和比例限制，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

資料庫中的 CATEGORY 資料表包含下列 11 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
(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**
>
> 最外部的 ORDER BY 表達式只能包含選取清單中的資料欄。

主題

- 語法

- Parameters
- 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 ]
```

Parameters

表達式

定義查詢結果排序順序的表達式。它由選取清單中的一或多個資料欄組成。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。
- 在任何平行系統中 AWS Clean Rooms，例如，當 ORDER BY 不會產生唯一的排序時，資料列的順序是非確定性的。也就是說，如果 ORDER BY 表達式產生重複的值，則這些列的傳回順序可能會與其他系統或 的執行 AWS Clean Rooms 不同。
- AWS Clean Rooms 不支援 ORDER BY 子句中的字串常值。

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 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。一些相互關聯的子查詢類型遵循 AWS Clean Rooms 無法裝飾且不支援的模式。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。

- AWS Clean Rooms 系統資料表的 Null 敏感相互關聯參考。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);
```

AWS Clean Rooms Spark SQL 函數

AWS Clean Rooms Spark SQL 支援下列 SQL 函數：

主題

- 彙總函數
- 陣列函數
- 條件式運算式
- 建構子函數
- 資料類型格式化函數
- 日期和時間函數
- 加密和解密函數
- 雜湊函數
- Hyperloglog 函數
- JSON 函數
- 數學函數
- 純量函數
- 字串函數
- 隱私權相關函數
- 範圍函數

彙總函數

AWS Clean Rooms Spark SQL 中的彙總函數用於對一組資料列執行計算或操作，並傳回單一值。它們對於資料分析和摘要任務至關重要。

AWS Clean Rooms Spark SQL 支援下列彙總函數：

主題

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 (expression[, isIgnoreNull] )
```

引數

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：

isIgnoreNull

布林值，可判斷函數是否應僅傳回非空值。

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 dateid eventname為 之任何 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 dateid eventname為 Eagles或 之任何 的執行個體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
```

APPROX COUNT_DISTINCT 函數

APPROX COUNT_DISTINCT 提供有效的方法來估計資料欄或資料集中唯一值的數量。

語法

```
approx_count_distinct(expr[, relativeSD])
```

引數

expr

您要預估唯一值數目的表達式或資料欄。

它可以是單一資料欄、複雜表達式或資料欄的組合。

relativeSD

選用參數，指定預估值所需的相對標準差。

其值介於 0 和 1 之間，代表預估值可接受的相對誤差上限。較小的 relativeSD 值將產生更準確但較慢的估算。

如果未提供此參數，則會使用預設值 （通常約為 0.05 或 5%)。

傳回值

傳回 HyperLogLog++ 的預估基數。 relativeSD 定義允許的最大相對標準差。

範例

下列查詢會估計資料col1欄中唯一值的數量，相對標準差為 1% (0.01)。

```
SELECT approx_count_distinct(col1, 0.01)
```

下列查詢估計資料col1欄中有 3 個唯一值 （值 1、2 和 3)。

```
SELECT approx_count_distinct(col1) FROM VALUES (1), (1), (2), (2), (3) tab(col1)
```

APPROX PERCENTILE 函數

APPROX PERCENTILE 用於估計指定表達式或資料欄的百分位數值，而不必排序整個資料集。當您需要快速了解大型資料集的分佈或追蹤以百分位數為基礎的指標，而執行精確百分位數計算時，此函數非常有用。不過，請務必了解速度和準確性之間的權衡，並根據使用案例的特定需求選擇適當的容錯能力。

語法

```
APPROX_PERCENTILE(expr, percentile [, accuracy])
```

引數

expr

您要預估百分位數值的表達式或資料欄。

它可以是單一資料欄、複雜表達式或資料欄的組合。

percentile

您要預估的百分位數值，以介於 0 和 1 之間的值表示。

例如，0.5 會對應到第 50 個百分位數（中位數）。

準確度

選用參數，指定百分位數預估的所需準確性。其值介於 0 和 1 之間，代表預估值可接受的相對誤差上限。較小的accuracy值將產生更精確但較慢的估算。如果未提供此參數，則會使用預設值（通常約為 0.05 或 5%)。

傳回值

傳回數值或 ANSI 間隔資料欄 col 的近似百分位數，這是排序的 col 值中的最小值（從最小到最大排序），使得不超過 col 值的百分比小於值或等於該值。

百分比的值必須介於 0.0 和 1.0 之間。準確度參數（預設值：10000) 是正數值常值，可控制記憶體成本的近似準確度。

較高的準確度值可產生更好的準確度，1.0/accuracy是近似值的相對錯誤。

當百分比為陣列時，百分比陣列的每個值必須介於 0.0 和 1.0 之間。在此情況下， 會以指定的百分比陣列傳回資料欄 col 的近似百分位數陣列。

範例

下列查詢預估response_time資料欄的第 95 個百分位數，最大相對誤差為 1% (0.01)。

```
SELECT APPROX_PERCENTILE(response_time, 0.95, 0.01) AS p95_response_time
FROM my_table;
```

下列查詢預估tab資料表中col資料欄的第 50 個、第 40 個和第 10 個百分位數的值。

```
SELECT approx_percentile(col, array(0.5, 0.4, 0.1), 100) FROM VALUES (0), (1), (2),
 (10) AS tab(col)
```

下列查詢預估 col 欄中值的第 50 個百分位數 （ 中位數 ）。

```
SELECT approx_percentile(col, 0.5, 100) FROM VALUES (0), (6), (7), (9), (10) AS
 tab(col)
```

AVG 函數

AVG 函數會傳回輸入表達式值的平均值 （ 算術平均值 ）。AVG 函數使用數值並忽略 NULL 值。

語法

```
AVG (column)
```

引數

##

函數操作的目標欄。資料欄是下列其中一種資料類型：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- DOUBLE
- FLOAT

資料類型

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

AVG 函數支援的傳回類型為：

- BIGINT 適用於任何整數類型引數
- DOUBLE 浮點數引數
- 傳回與任何其他引數類型表達式相同的資料類型

具有DECIMAL引數之AVG函數結果的預設精確度為 38。結果的小數位數和引數的小數位數相同。例如，資料DEC(5,2)欄AVG的 會傳回DEC(38,2)資料類型。

範例

從SALES資料表尋找每筆交易的平均銷售數量。

```
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
```

CARDINALITY 函數

CARDINALITY 函數會傳回 ARRAY 或 MAP 表達式 (expr) 的大小。

此函數有助於尋找陣列的大小或長度。

語法

```
cardinality(expr)
```

引數

expr

ARRAY 或 MAP 表達式。

傳回值

傳回陣列或映射的大小 (INTEGER)。

如果 sizeOfNull 設定為 false或 enabled 設定為 ，則函數會傳回 NULL null 輸入true。

否則，函數會傳回 null -1 輸入。使用預設設定時，函數會傳回 null -1 輸入。

範例

下列查詢會計算指定陣列中的基數或元素數量。陣列 ('b', 'd', 'c', 'a') 有 4 個元素，因此此查詢的輸出會是 4。

```
SELECT cardinality(array('b', 'd', 'c', 'a'));
 4
```

COLLECT_LIST 函數

COLLECT_LIST 函數會收集並傳回非唯一元素的清單。

當您想要從一組資料列收集多個值到單一陣列或列出資料結構時，這種類型的函數很有用。

> **Note**
>
> 函數是非確定性的，因為收集結果的順序取決於資料列的順序，這在執行隨機播放操作之後可能是非確定性的。

語法

```
collect_list(expr)
```

引數

expr

任何類型的表達式。

傳回值

傳回引數類型的 ARRAY。陣列中元素的順序是非確定性的。

會排除 NULL 值。

如果指定 DISTINCT，則函數只會收集唯一值，並且是`collect_set` 彙總函數的同義詞。

範例

下列查詢會將 col 資料欄中的所有值收集到清單中。VALUES 子句用於建立具有三列的內嵌資料表，其中每一列都有一個值分別為 1、2 和 1 的單欄 col。然後， `collect_list()` 函數會用來將所有值從 col 資料欄彙總到單一陣列。此 SQL 陳述式的輸出會是陣列 `[1,2,1]`，其中包含來自 col 資料欄的所有值，依其在輸入資料中出現的順序排列。

```
SELECT collect_list(col) FROM VALUES (1), (2), (1) AS tab(col);
 [1,2,1]
```

COLLECT_SET 函數

COLLECT_SET 函數會收集並傳回一組唯一元素。

當您想要從一組資料列收集所有相異的值到單一資料結構，但不包含任何重複項目時，此函數很有用。

> **Note**
>
> 函數是非確定性的，因為收集結果的順序取決於資料列的順序，這在執行隨機播放操作之後可能是非確定性的。

語法

```
collect_set(expr)
```

引數

expr

MAP 以外的任何類型的表達式。

傳回值

傳回引數類型的 ARRAY。陣列中元素的順序是非確定性的。

會排除 NULL 值。

範例

下列查詢會從 col 資料欄將所有唯一值收集到集合中。VALUES 子句用於建立具有三列的內嵌資料表，其中每一列都有一個值分別為 1、2 和 1 的單欄 col。然後， collect_set()函數會用來從 col 資料欄將所有唯一值彙總為單一集合。此 SQL 陳述式的輸出會是集合 [1,2]，其中包含 col 欄中的唯一值。重複值 1 只會包含在結果中一次。

```
SELECT collect_set(col) FROM VALUES (1), (2), (1) AS tab(col);
 [1,2]
```

COUNT 和 COUNT DISTINCT函數

COUNT 函數會計算運算式定義的資料列。COUNT DISTINCT 函數會計算資料欄或表達式中不同非 NULL 值的數量。它會先消除指定表達式中的所有重複值，再執行計數。

語法

```
COUNT (DISTINCT column)
```

引數

##

函數操作的目標欄。

資料類型

COUNT 函數和 COUNT DISTINCT函數支援所有引數資料類型。

COUNT DISTINCT 函數會傳回 BIGINT。

範例

計算佛羅里達州的所有使用者。

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

計算EVENT資料表中的所有唯一場地 IDs。

```
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。

語法

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

```
COUNT ( [ DISTINCT | ALL ] 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。

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
...
```

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。表達式是任何數值資料類型。

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

語法

```
MEDIAN ( median_expression )
```

引數

median_expression

函數運算的目標欄或表達式。

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。表達式是任何數值資料類型。

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 函數

PERCENTILE 函數用於計算確切的百分位數值，方法是先排序col欄中的值，然後在指定的 中尋找值percentage。

當您需要計算確切的百分位數值，且運算成本適用於您的使用案例時，PERCENTILE 函數非常有用。它提供比 APPROX_PERCENTILE 函數更準確的結果，但速度可能較慢，尤其是大型資料集。

相反地，APPROX_PERCENTILE 函數是更有效率的替代方案，可提供具有指定容錯能力的百分位數值估計值，使其更適合速度優先順序高於絕對精確度的情況。

語法

```
percentile(col, percentage [, frequency])
```

引數

col

您要計算百分位數值的表達式或資料欄。

百分比

您要計算的百分位數值，以介於 0 和 1 之間的值表示。

例如，0.5 會對應到第 50 個百分位數（中位數）。

頻率

選用參數，指定資料col欄中每個值的頻率或權重。如果提供，函數會根據每個值的頻率計算百分位數。

傳回值

以指定的百分比傳回數值或 ANSI 間隔資料欄 col 的確切百分位數值。

百分比的值必須介於 0.0 和 1.0 之間。

頻率的值應為正整數

範例

下列查詢會尋找大於或等於資料col欄中值 30% 的值。由於值為 0 和 10，因此第 30 個百分位數為 3.0，因為它是大於或等於資料 30% 的值。

```
SELECT percentile(col, 0.3) FROM VALUES (0), (10) AS tab(col);
 3.0
```

SKEWNESS 函數

SKEWNESS 函數會傳回從群組值計算的偏斜值。

偏斜是一種統計指標，描述資料集中的非對稱性或缺乏對稱性。它提供有關資料分佈形狀的資訊。

此函數有助於了解資料集的統計屬性，並通知進一步的分析或決策。

語法

```
skewness(expr)
```

引數

expr

評估為數值的表達式。

傳回值

傳回 DOUBLE。

如果指定 DISTINCT，則函數只會在一組唯一的 expr 值上執行。

範例

下列查詢會計算資料col欄中值的偏斜。在此範例中， VALUES子句用於建立具有四個資料列的內嵌資料表，其中每一列都有一個值col為 -10、-20、100 和 1000 的資料欄。然後， skewness()函數會用來計算資料col欄中值的偏斜。結果 1.1135657469022011 代表資料中扭曲的程度和方向。正偏斜值表示資料偏向右側，大量值集中在分佈的左側。負偏斜值表示資料偏向左側，大量值集中在分佈的右側。

```
SELECT skewness(col) FROM VALUES (-10), (-20), (100), (1000) AS tab(col);
 1.1135657469022011
```

下列查詢會計算 col 欄中值的偏斜。與先前的範例類似， VALUES子句可用來建立具有四列的內嵌資料表，其中每一列都有一個值col為 -1000、-100、10 和 20 的資料欄。然後， skewness()函數會用來計算資料col欄中值的偏斜。結果 -1.1135657469022011 代表資料中偏斜的程度和方向。在此情況下，負偏斜值表示資料偏向左側，並將大量值集中在分佈的右側。

```
SELECT skewness(col) FROM VALUES (-1000), (-100), (10), (20) AS tab(col);
 -1.1135657469022011
```

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)
```

表達式必須具有數值資料類型。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函數

SUM 函數會傳回輸入資料欄或表達式值的總和。SUM 函數使用數值並忽略NULL值。

在計算總和之前，SUM DISTINCT函數會從指定的表達式消除所有重複值。

語法

```
SUM (DISTINCT column )
```

引數

##

函數操作的目標欄。資料欄是任何數值資料類型。

範例

尋找從SALES資料表支付的所有佣金總和。

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

尋找從SALES資料表支付的所有不同佣金總和。

```
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

- ARRAY 函數
- ARRAY_CONTAINS 函數
- ARRAY_DISTINCT 函數
- ARRAY_EXCEPT 函數
- ARRAY_INTERSECT 函數
- ARRAY_JOIN 函數
- ARRAY_REMOVE 函數
- ARRAY_UNION 函數
- EXPLODE 函數
- FLATTEN 函數

ARRAY 函數

建立具有指定元素的陣列。

語法

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

引數

expr1, expr2

除了日期和時間類型之外的任何資料類型的表達式。引數不需要是相同的資料類型。

傳回類型

陣列函數會傳回包含表達式中元素的 ARRAY。

範例

下列範例顯示數值的陣列和不同資料類型的陣列。

```
--an array of numeric values
select array(1,50,null,100);
      array
------------------
 [1,50,null,100]
(1 row)

--an array of different data types
select array(1,'abc',true,3.14);
        array
-----------------------
 [1,"abc",true,3.14]
(1 row)
```

ARRAY_CONTAINS 函數

ARRAY_CONTAINS 函數可用於對陣列資料結構執行基本成員資格檢查。當您需要檢查陣列中是否存在特定值時，ARRAY_CONTAINS 函數很有用。

語法

```
array_contains(array, value)
```

引數

陣列

要搜尋的 ARRAY。

值

具有與陣列元素共用最小常見類型的 運算式。

傳回類型

ARRAY_CONTAINS 函數會傳回 BOOLEAN。

如果值為 NULL，則結果為 NULL。

如果陣列中的任何元素為 NULL，如果值與任何其他元素不相符，則結果為 NULL。

範例

下列範例會檢查陣列是否[1, 2, 3]包含值 4。由於陣列 [1, 2, 3】 不包含值 4，因此 array_contains 函數會傳回 false。

```
SELECT array_contains(array(1, 2, 3), 4)
false
```

下列範例會檢查陣列是否[1, 2, 3]包含值 2。由於陣列[1, 2, 3]確實包含值 2，因此 array_contains 函數會傳回 true。

```
SELECT array_contains(array(1, 2, 3), 2);
 true
```

ARRAY_DISTINCT 函數

ARRAY_DISTINCT 函數可用來從陣列中移除重複值。當您需要從陣列移除重複項目並僅使用唯一元素時，ARRAY_DISTINCT 函數非常有用。這在您想要在資料集上執行操作或分析，而不會干擾重複值的情況下很有用。

語法

```
array_distinct(array)
```

引數

陣列

ARRAY 表達式。

傳回類型

ARRAY_DISTINCT 函數會傳回只包含輸入陣列中唯一元素的 ARRAY。

範例

在此範例中，輸入陣列[1, 2, 3, null, 3]包含重複的 值3。array_distinct 函數會移除此重複值，3並傳回具有唯一元素的新陣列：[1, 2, 3, null]。

```
SELECT array_distinct(array(1, 2, 3, null, 3));
 [1,2,3,null]
```

在此範例中，輸入陣列[1, 2, 2, 3, 3, 3]包含 2和 的重複值3。array_distinct 函數會移除這些重複項目，並傳回具有唯一元素的新陣列：[1, 2, 3]。

```
SELECT array_distinct(array(1, 2, 2, 3, 3, 3))
  [1,2,3]
```

ARRAY_EXCEPT 函數

ARRAY_EXCEPT 函數會採用兩個陣列做為引數，並傳回僅包含存在於第一個陣列中但不包含第二個陣列之元素的新陣列。

當您需要尋找一個陣列與另一個陣列相比唯一的元素時，ARRAY_EXCEPT 非常有用。這在您需要在陣列上執行類似集合的操作時很有用，例如尋找兩組資料之間的差異。

語法

```
array_except(array1, array2)
```

引數

array1

具有類似元素的任何類型的 ARRAY。

array2

元素的 ARRAY，與 array1 元素共用最不常見的類型。

傳回類型

ARRAY_EXCEPT 函數會將符合類型的 ARRAY 傳回至 array1，而沒有重複項目。

範例

在此範例中，第一個陣列[1, 2, 3]包含元素 1、2 和 3。第二個陣列[2, 3, 4]包含元素 2、3 和 4。array_except 函數會從第一個陣列中移除元素 2 和 3，因為它們也存在於第二個陣列中。產生的輸出是陣列 [1]。

```
SELECT array_except(array(1, 2, 3), array(2, 3, 4))
  [1]
```

在此範例中，第一個陣列[1, 2, 3]包含元素 1、2 和 3。第二個陣列[1, 3, 5]包含元素 1、3 和 5。array_except 函數會從第一個陣列中移除元素 1 和 3，因為它們也存在於第二個陣列中。產生的輸出是陣列 [2]。

```
SELECT array_except(array(1, 2, 3), array(1, 3, 5));
 [2]
```

ARRAY_INTERSECT 函數

ARRAY_INTERSECT 函數採用兩個陣列做為引數，並傳回包含兩個輸入陣列中存在元素的新陣列。當您需要尋找兩個陣列之間的常見元素時，此函數很有用。這在您需要在陣列上執行類似集合的操作時很有用，例如尋找兩組資料之間的交集。

語法

```
array_intersect(array1, array2)
```

引數

array1

具有類似元素的任何類型的 ARRAY。

array2

元素的 ARRAY，與 array1 元素共用最不常見的類型。

傳回類型

ARRAY_INTERSECT 函數會將符合類型的 ARRAY 傳回至 array1，而 array1 和 array2 中不包含重複項目和元素。

範例

在此範例中，第一個陣列[1, 2, 3]包含元素 1、2 和 3。第二個陣列[1, 3, 5]包含元素 1、3 和 5。ARRAY_INTERSECT 函數會識別兩個陣列之間的常見元素，也就是 1 和 3。產生的輸出陣列為 [1, 3]。

```
SELECT array_intersect(array(1, 2, 3), array(1, 3, 5));
 [1,3]
```

ARRAY_JOIN 函數

ARRAY_JOIN 函數需要兩個引數：第一個引數是將聯結的輸入陣列。第二個引數是用來串連陣列元素的分隔符號字串。當您需要將字串陣列（或任何其他資料類型）轉換為單一串連字串時，此函數非常有用。這在您想要以單一格式化字串呈現一系列值的情況下很有用，例如用於顯示目的或用於進一步處理。

語法

```
array_join(array, delimiter[, nullReplacement])
```

引數

陣列

任何 ARRAY 類型，但其元素會解譯為字串。

delimiter

用來分隔串連陣列元素的 STRING。

nullReplacement

用於在結果中表達 NULL 值的 STRING。

傳回類型

ARRAY_JOIN 函數會傳回 STRING，其中陣列元素以分隔符號分隔，而 null 元素會替換為 nullReplacement。如果省略 nullReplacement ，則會篩選掉null元素。如果任何引數為 NULL，則結果為 NULL。

範例

在此範例中，ARRAY_JOIN 函數採用陣列['hello', 'world']，並使用分隔符號 ' '（空格字元）聯結元素。產生的輸出是字串 'hello world'。

```
SELECT array_join(array('hello', 'world'), ' ');
 hello world
```

在此範例中，ARRAY_JOIN 函數採用陣列['hello', null, 'world']，並使用分隔符號 ' '（空格字元）聯結元素。該null值會以提供的取代字串 ','（逗號）取代。產生的輸出是字串 'hello , world'。

```
SELECT array_join(array('hello', null ,'world'), ' ', ',');
 hello , world
```

ARRAY_REMOVE 函數

ARRAY_REMOVE 函數採用兩個引數：第一個引數是將元素從中移除的輸入陣列。第二個引數是從陣列中移除的值。當您需要從陣列中移除特定元素時，此函數很有用。這在您需要對 值陣列執行資料清理或預先處理的情況下很有用。

語法

```
array_remove(array, element)
```

引數

陣列

ARRAY。

元素

與陣列元素共用最小常見類型的類型表達式。

傳回類型

ARRAY_REMOVE 函數會傳回符合陣列類型的結果類型。如果要移除的元素為 NULL，則結果為 NULL。

範例

在此範例中，ARRAY_REMOVE 函數會取得陣列，[1, 2, 3, null, 3]並移除值 3 的所有出現次數。產生的輸出是陣列 [1, 2, null]。

```
SELECT array_remove(array(1, 2, 3, null, 3), 3);
 [1,2,null]
```

ARRAY_UNION 函數

ARRAY_UNION 函數採用兩個陣列做為引數，並傳回包含兩個輸入陣列中唯一元素的新陣列。當您需要結合兩個陣列並消除任何重複的元素時，此函數很有用。這在您需要在陣列上執行類似集合的操作的情況下很有用，例如尋找兩組資料之間的聯集。

語法

```
array_union(array1, array2)
```

引數

array1

ARRAY。

array2

與 array1 相同類型的 ARRAY。

傳回類型

ARRAY_UNION 函數會傳回與陣列相同類型的 ARRAY。

範例

在此範例中，第一個陣列[1, 2, 3]包含元素 1、2 和 3。第二個陣列[1, 3, 5]包含元素 1、3 和 5。ARRAY_UNION 函數結合了兩個陣列的唯一元素，進而產生輸出陣列 [1, 2, 3, 5]。T

```
SELECT array_union(array(1, 2, 3), array(1, 3, 5));
 [1,2,3,5]
```

EXPLODE 函數

EXPLODE 函數用於將具有陣列或映射資料欄的單一資料列轉換為多個資料列，其中每個資料列對應至陣列或映射中的單一元素。

語法

```
explode(expr)
```

引數

expr

陣列表達式或映射表達式。

傳回類型

EXPLODE 函數會傳回一組資料列，其中每一列代表輸入陣列或映射中的單一元素。

輸出列的資料類型取決於輸入陣列或映射中元素的資料類型。

範例

下列範例採用單列陣列 【10、20】，並將其轉換為兩個不同的列，每個列都包含其中一個陣列元素 (10 和 20)。

```
SELECT explode(array(10, 20));
```

在第一個範例中，輸入陣列直接做為引數傳遞給 explode()。在此範例中，使用語法指定輸入陣列，其中明確提供資料欄名稱 => (collection)。

```
SELECT explode(array(10, 20));
```

這兩種方法都是有效的，並達到相同的結果，但當您需要從較大的資料集分解資料欄時，第二個語法會更有用，而不只是簡單的陣列常值。

FLATTEN 函數

FLATTEN 函數用於將巢狀陣列結構「平面化」為單一平面陣列。

語法

```
flatten(arrayOfArrays)
```

引數

arrayOfArrays

陣列陣列。

傳回類型

FLATTEN 函數會傳回陣列。

範例

在此範例中，輸入是具有兩個內部陣列的巢狀陣列，而輸出是包含內部陣列中所有元素的單一扁平陣列。FLATTEN 函數採用巢狀陣列，`[[1, 2], [3, 4]]`並將所有元素合併為單一陣列`[1, 2, 3, 4]`。

```
SELECT flatten(array(array(1, 2), array(3, 4)));
 [1,2,3,4]
```

條件式運算式

在 SQL 中，條件式表達式用於根據特定條件做出決策。它們可讓您控制 SQL 陳述式的流程，並根據一或多個條件的評估傳回不同的值或執行不同的動作。

AWS Clean Rooms 支援下列條件式表達式：

主題

- CASE 條件式運算式
- COALESCE 表達式
- GREATEST 和 LEAST 表達式
- IF 運算式
- IS_NULL 表達式
- IS_NOT_NULL 表達式
- 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 表達式

COALESCE 表達式會傳回清單中不是 null 的第一個表達式的值。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表達式套用至兩個資料欄。

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

NVL 表達式的預設資料欄名稱為 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 表達式

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)
```

IF 運算式

IF 條件函數會根據條件傳回兩個值的其中之一。

此函數是 SQL 中使用的常見控制流程陳述式，可根據條件的評估做出決策並傳回不同的值。它適用於在查詢中實作簡單的 if-else 邏輯。

語法

```
if(expr1, expr2, expr3)
```

引數

expr1

評估的條件或表達式。如果為 `true`，則函數會傳回 expr2 的值。如果 expr1 為 `false`，則函數會傳回 expr3 的值。

expr2

如果 expr1 為 ，則評估和傳回的表達式true。

expr3

如果 expr1 為 ，則評估和傳回的表達式false。

傳回值

如果 expr1評估為 true，則 會傳回 expr2；否則 會傳回 expr3。

範例

下列範例使用 if()函數，根據條件傳回兩個值的其中之一。正在評估的條件是 1 < 2，也就是 true，因此'a'會傳回第一個值。

```
SELECT if(1 < 2, 'a', 'b');
 a]
```

IS_NULL 表達式

IS_NULL 條件式表達式用於檢查值是否為 null。

此表達式是 的同義詞IS NULL。

語法

```
is_null(expr)
```

引數

expr

任何類型的表達式。

傳回值

IS_NULL 條件式表達式會傳回布林值。如果 expr1是 NULL，則傳回 true，否則傳回 false。

範例

下列範例會檢查值是否為 1 Null，並傳回布林值結果，true因為 1 是有效的非 Null 值。

```
SELECT is not null(1);
 true
```

下列範例會從squirrels資料表中選取資料id欄，但僅適用於存留期資料欄為 的資料列null。

```
SELECT id FROM squirrels WHERE is_null(age)
```

IS_NOT_NULL 表達式

IS_NOT_NULL 條件式表達式用於檢查值是否不是 null。

此表達式是 的同義詞IS NOT NULL。

語法

```
is_not_null(expr)
```

引數

expr

任何類型的表達式。

傳回值

IS_NOT_NULL 條件式表達式會傳回布林值。如果 expr1 不是 NULL，則傳回 true，否則傳回 false。

範例

下列範例會檢查值是否1不是 null，並傳回布林值結果，true因為 1 是有效的非 Null 值。

```
SELECT is not null(1);
 true
```

下列範例會從squirrels資料表中選取資料id欄，但僅適用於存留期資料欄不是 的資料列null。

```
SELECT id FROM squirrels WHERE is_not_null(age)
```

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 根據輸入表達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

對於 NVL2，傳回的值將為 not_null_return_value 或 null_return_value 參數，以函數選取的值為準，但資料類型為 not_null_return_value。

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

比較兩個引數，如果引數相等，則傳回 Null。如果不相等，則會傳回第一個引數。

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，則會傳回第一個引數。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。

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)
```

建構子函數

SQL 建構函數是用來建立新的資料結構的函數，例如陣列或地圖。

它們會取得一些輸入值，並傳回新的資料結構物件。建構函式通常以其建立的資料類型命名，例如 ARRAY 或 MAP。

建構函式與純量函式或彙總函式不同，它們操作於現有資料並傳回單一值。建構函式用於建立新的資料結構，然後可用於進一步的資料處理或分析。

AWS Clean Rooms 支援下列建構函數：

主題

- MAP 建構函數
- NAMED_STRUCT 建構函數
- STRUCT 建構函數

MAP 建構函數

MAP 建構函式會使用指定的索引鍵/值對建立映射。

當您需要在 SQL 查詢中以程式設計方式建立新的資料結構時，像 MAP 這樣的建構器函數很有用。它們可讓您建置複雜的資料結構，可用於進一步的資料處理或分析。

語法

```
map(key0, value0, key1, value1, ...)
```

引數

key0

任何可比較類型的表達式。所有 key0 必須共用最低常見的類型。

value0

任何類型的表達式。所有 valueN 必須共用最小的常見類型。

傳回值

MAP 函數會傳回 MAP，其索引鍵輸入為最不常見的索引鍵 0 類型，而值輸入為最不常見的值 0 類型。

範例

下列範例會使用兩個索引鍵值對建立新的映射：索引鍵`1.0`與值 相關聯`'2'`。金鑰與值 `3.0`相關聯`'4'`。產生的對應接著會傳回為 SQL 陳述式的輸出。

```
SELECT map(1.0, '2', 3.0, '4');
 {1.0:"2",3.0:"4"}
```

NAMED_STRUCT 建構函數

NAMED_STRUCT 建構函數會建立具有指定欄位名稱和值的結構。

當您需要在 SQL 查詢中以程式設計方式建立新的資料結構時，像 NAMED_STRUCT 之類的建構器函數很有用。它們可讓您建置複雜的資料結構，例如結構或記錄，可用於進一步的資料處理或分析。

語法

```
named_struct(name1, val1, name2, val2, ...)
```

引數

name1

STRING 常值命名欄位 1。

val1

指定欄位 1 值的任何類型的表達式。

傳回值

NAMED_STRUCT 函數會傳回欄位 1 符合 val1 類型的結構。

範例

下列範例會建立新的結構，其中包含三個具名欄位： 欄位"a"會獲指派值 1。欄位"b"被指派值 2。欄位"c"被指派值 3。產生的結構接著會傳回為 SQL 陳述式的輸出。

```
SELECT named_struct("a", 1, "b", 2, "c", 3);
 {"a":1,"b":2,"c":3}
```

STRUCT 建構函數

STRUCT 建構函數會建立具有指定欄位值的結構。

當您需要在 SQL 查詢中以程式設計方式建立新的資料結構時，像 STRUCT 這樣的建構器函數非常有用。它們可讓您建置複雜的資料結構，例如結構或記錄，可用於進一步的資料處理或分析。

語法

```
struct(col1, col2, col3, ...)
```

引數

col1

欄名或任何有效表達式。

傳回值

STRUCT 函數會傳回符合 expr1 類型的欄位 1 結構。

如果引數是具名參考，則名稱會用來命名 欄位。否則，欄位會命名為 colN，其中 N 是結構中欄位的位置。

範例

下列範例會建立新的結構，其中包含三個欄位：第一個欄位會獲指派值 1。第二個欄位會獲指派值 2。第三個欄位會獲指派值 3。根據預設，產生的結構中的欄位會根據其在引數清單中的位置命名為 col1col2、 col3和 。產生的結構接著會傳回為 SQL 陳述式的輸出。

```
SELECT struct(1, 2, 3);
 {"col1":1,"col2":2,"col3":3}
```

資料類型格式化函數

使用資料類型格式化函數，您可以將值從一個資料類型轉換為另一個資料類型。範本對於其中每一個函數，第一個引數一律為要格式化的值，第二個引數包含新格式的範本。

AWS Clean Rooms Spark SQL 支援多種資料類型格式化函數。

主題

- BASE64 函數
- CAST 函數
- DECODE 函數
- ENCODE 函數
- HEX 函數
- STR_TO_MAP 函數
- TO_CHAR
- TO_DATE 陣列
- TO_NUMBER
- UNBASE64 函數
- UNHEX 函數
- 日期時間格式字串
- 數值格式字串

BASE64 函數

BASE64 函數會使用 MIME 的 RFC2045 Base64 傳輸編碼，將表達式轉換為 base 64 字串。

語法

```
base64(expr)
```

引數

expr

函數將解譯為 BINARY 的 BINARY 表達式或 STRING。

傳回類型

STRING

範例

若要將指定的字串輸入轉換為其 Base64 編碼表示法。請使用下列範例。結果是輸入字串「Spark SQL」的 Base64 編碼表示法，也就是「U3BhcmsgU1FM」。

```
SELECT base64('Spark SQL');
 U3BhcmsgU1FM
```

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。

某些資料類型需要使用 CAST 函數明確轉換為其他資料類型。其他資料類型可以做為另一個命令的一部分隱含轉換，而無需使用 CAST。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 )
```

引數

表達式

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。表達式不能包含空白或空白字串。

type

支援的 之一，但 BINARY 資料類型 和 BINARY VARYING 資料類型除外。

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， 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)
```

在與上一個範例類似的情況下，您可以使用 來獲得對輸出格式的額外控制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)
```

DECODE 函數

DECODE 函數是 ENCODE 函數的複本，用於使用特定字元編碼將字串轉換為二進位格式。DECODE 函數會取得二進位資料，並使用指定的字元編碼將其轉換回可讀取的字串格式。

當您需要使用存放在資料庫中的二進位資料，並且需要以人類可讀的格式呈現，或者您需要在不同字元編碼之間轉換資料時，此函數非常有用。

語法

```
decode(expr, charset)
```

引數

expr

以字元集編碼的 BINARY 表達式。

字元集

STRING 表達式。

支援的字元集編碼（不區分大小寫）：'US-ASCII'、'ISO-8859-1'、'UTF-8''UTF-16BE'、、 'UTF-16LE'和 'UTF-16'。

傳回類型

DECODE 函數會傳回 STRING。

範例

下列範例有一個名為 的資料表messages，其中包含名為 的資料欄message_text，該資料欄使用 UTF-8 字元編碼以二進位格式存放訊息資料。DECODE 函數會將二進位資料轉換為可讀取的字串格式。此查詢的輸出是存放在訊息資料表中訊息的可讀取文字， ID 為 123，使用 'utf-8' 編碼從二進位格式轉換為字串。

```
SELECT decode(message_text, 'utf-8') AS message
FROM messages
WHERE message_id = 123;
```

ENCODE 函數

ENCODE 函數用於使用指定的字元編碼將字串轉換為其二進位表示法。

當您需要使用二進位資料或在不同的字元編碼之間轉換時，此函數非常有用。例如，您可以在將資料儲存在需要二進位儲存的資料庫中，或在使用不同字元編碼的系統之間傳輸資料時，使用 ENCODE 函數。

語法

```
encode(str, charset)
```

引數

str

要編碼的 STRING 表達式。

字元集

指定編碼的 STRING 表達式。

支援的字元集編碼（不區分大小寫）：'US-ASCII'、'ISO-8859-1'、'UTF-8''UTF-16BE'、、 'UTF-16LE'和 'UTF-16'。

傳回類型

ENCODE 函數會傳回 BINARY。

範例

下列範例使用 'utf-8'編碼將字串轉換為'abc'其二進位表示法，在此情況下，會導致傳回原始字串。這是因為'utf-8'編碼是變數寬度字元編碼，可使用每個字元的單一位元組來代表整個 ASCII 字元集（包括字母 'b'、 'a'和 'c')。因此，'abc'使用 的二進位表示'utf-8'法與原始字串相同。

```
SELECT encode('abc', 'utf-8');
 abc
```

HEX 函數

HEX 函數會將數值（整數或浮點數）轉換為對應的十六進位字串表示法。

十六進位是一種數字系統，使用 16 個不同的符號 (0-9 和 A-F) 來表示數值。它通常用於電腦科學和程式設計，以更精簡且人類可讀的格式表示二進位資料。

語法

```
hex(expr)
```

引數

expr

BIGINT、BINARY 或 STRING 表達式。

傳回類型

HEX 傳回 STRING。函數會傳回引數的十六進位表示法。

範例

下列範例採用整數值 17 做為輸入，並將 HEX() 函數套用到其中。輸出為 11，這是輸入值 的十六進位表示法17。

```
SELECT hex(17);
 11
```

下列範例會將字串轉換為'Spark_SQL'其十六進位表示法。輸出是 537061726B2053514C，這是輸入字串 的十六進位表示法'Spark_SQL'。

```
SELECT hex('Spark_SQL');
 537061726B2053514C
```

在此範例中，字串 'Spark_SQL' 的轉換方式如下：

- 'S' -> 53
- 'p' -> 70
- 'a' -> 61
- 'r' -> 72 '
- k' -> 6B
- '_' -> 20
- 'S' -> 53
- 'Q' -> 51
- 'L' -> 4C

這些十六進位值的串連會產生最終輸出 "537061726B2053514C"。

STR_TO_MAP 函數

STR_TO_MAP 函數是string-to-map轉換函數。它會將映射的字串表示法 （或字典） 轉換為實際的映射資料結構。

當您需要在 SQL 中使用映射資料結構，但資料最初儲存為字串時，此函數很有用。透過將字串表示法轉換為實際映射，您就可以對映射資料執行操作和操作。

語法

```
str_to_map(text[, pairDelim[, keyValueDelim]])
```

引數

文字

代表映射的 STRING 表達式。

pairDelim

選用 STRING 常值，指定如何分隔項目。預設為逗號 (',')。

keyValueDelim

選用 STRING 常值，指定如何分隔每個鍵值對。預設為冒號 (':')。

傳回類型

STR_TO_MAP 函數會針對索引鍵和值傳回 STRING 的 MAP。pairDelim 和 keyValueDelim 都視為規則表達式。

範例

下列範例採用輸入字串和兩個分隔符號引數，並將字串表示法轉換為實際的映射資料結構。在此特定範例中，輸入字串'a:1,b:2,c:3'代表具有下列索引鍵/值對的映射： 'a' 是索引鍵， '1' 是值。 'b' 是索引鍵， '2' 是值。 'c' 是索引鍵， 是值， '3'是值。',' 分隔符號用於分隔索引鍵/值對，而':'分隔符號用於分隔每一對中的索引鍵和值。此查詢的輸出為：{"a":"1","b":"2","c":"3"}。這是產生的映射資料結構，其中索引鍵為 'a'、 'b'和 'c'，對應值為 '2'、 '1'和 '3'。

```
SELECT str_to_map('a:1,b:2,c:3', ',', ':');
 {"a":"1","b":"2","c":"3"}
```

下列範例示範 STR_TO_MAP 函數預期輸入字串為特定格式，並正確分隔索引鍵/值對。如果輸入字串不符合預期的格式，函數仍會嘗試建立映射，但產生的值可能不如預期。

```
SELECT str_to_map('a');
 {"a":null}
```

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**
>
> TO_CHAR 不支援 128 位元的 DECIMAL 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，差異會四捨五入並轉換為羅馬數字，如 to_char 欄所示：

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'rn') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;

 salesid | pricepaid | commission | difference |     to_char
---------+-----------+------------+------------+-----------------
       1 |    728.00 |     109.20 |     618.80 |           dcxix
       2 |     76.00 |      11.40 |      64.60 |             lxv
       3 |    350.00 |      52.50 |     297.50 |        ccxcviii
       4 |    175.00 |      26.25 |     148.75 |           cxlix
       5 |    154.00 |      23.10 |     130.90 |           cxxxi
       6 |    394.00 |      59.10 |     334.90 |         cccxxxv
       7 |    788.00 |     118.20 |     669.80 |           dclxx
       8 |    197.00 |      29.55 |     167.45 |          clxvii
       9 |    591.00 |      88.65 |     502.35 |             dii
      10 |     65.00 |       9.75 |      55.25 |              lv
(10 rows)
```

下列範例會將貨幣符號新增至資料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 (date_str)
```

```
TO_DATE (date_str, format)
```

引數

date_str

日期字串或可轉換為日期字串的資料類型。

format

符合 Spark 日期時間模式的字串常值。如需有效的日期時間模式，請參閱格式化和剖析的日期時間模式。

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 MMM 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');
```

下列 SQL 陳述式會將字串 20010631 轉換為日期：

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

結果是 null 值，因為六月只有 30 天。

```
to_date
------------
NULL
```

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
```

UNBASE64 函數

UNBASE64 函數會將引數從基礎 64 字串轉換為二進位。

Base64 編碼通常用於以文字格式表示二進位資料（例如影像、檔案或加密資訊），可安全地透過各種通訊管道（例如電子郵件、URL 參數或資料庫儲存）傳輸。

UNBASE64 函數可讓您反轉此程序，並復原原始二進位資料。當您需要處理以 Base64 格式編碼的資料時，例如與使用 Base64 做為資料傳輸機制的外部系統或 APIs 整合時，這類功能非常有用。

語法

```
unbase64(expr)
```

引數

expr

base64 格式的 STRING 表達式。

傳回類型

BINARY

範例

在下列範例中，Base64-encoded字串'U3BhcmsgU1FM'會轉換回原始字串 'Spark SQL'。

```
SELECT unbase64('U3BhcmsgU1FM');
 Spark SQL
```

UNHEX 函數

UNHEX 函數會將十六進位字串轉換回其原始字串表示法。

此函數在您需要使用以十六進位格式存放或傳輸的資料，且需要還原原始字串表示以進行進一步處理或顯示的情況下非常有用。

UNHEX 函數是 HEX 函數的複本。

語法

```
unhex(expr)
```

引數

expr

十六進位字元的 STRING 表達式。

傳回類型

UNHEX 傳回 BINARY。

如果 expr 的長度是奇數，則會捨棄第一個字元，並將結果填入 null 位元組。如果 expr 包含非十六進位字元，則結果為 NULL。

範例

下列範例使用 UNHEX() 和 DECODE() 函數，將十六進位字串轉換回其原始字串表示法。查詢的第一部分使用 UNHEX() 函數，將十六進位字串 '537061726B2053514C' 轉換為其二進位表示法。查詢的第二部分使用 DECODE() 函數，使用 'UTF-8' 字元編碼，將從 UNHEX() 函數取得的二進位資料轉換為字串。查詢的輸出是原始字串 'Spark_SQL'，已轉換為十六進位，然後傳回字串。

```
SELECT decode(unhex('537061726B2053514C'), 'UTF-8');
 Spark SQL
```

日期時間格式字串

您可以在下列常見案例中使用日期時間模式：

- 使用 CSV 和 JSON 資料來源來剖析和格式化日期時間內容時
- 使用下列函數在字串類型和日期或時間戳記類型之間轉換時：
  - unix_timestamp
  - date_format
  - to_unix_timestamp
  - from_unixtime
  - to_date
  - to_timestamp

- from_utc_timestamp
- to_utc_timestamp

使用下表中的模式字母進行日期和時間戳記剖析和格式化。

日期部分或時間部分	意義	範例
a	當日上午或下午，以上午至下午顯示	PM
D	一年中的某一天，以 3 位數字顯示	189
d	當月日期，以 2 位數字顯示	28
E	星期幾，以文字顯示	星期二 週二
F	對齊當月的星期幾，以 1 位數字顯示	3
G	Era 指標，以文字顯示	.ade 安諾多米尼
h	AM 或 PM 的時鐘小時，以 2 位數字顯示	12
H	一天中的小時，以 0–23 的 2 位數字顯示	0
k	一天的時鐘小時，以 1–24 之間的 2 位數字顯示	1
K	上午或下午小時，以 0–11 之間的 2 位數字顯示	0
m	小時分鐘，以 2 位數字顯示	30

日期部分或時間部分	意義	範例
M/L	一年中的月份，以月份顯示	7 07 7 月 7 月
O	本地化區域偏離 UTC	GMT+8 GMT+8：00 UTC-08：00
Q/q	一年的季度，以數字 (1 到 4) 或文字顯示	3 03 Q3 第三季度
s	分鐘的秒數，以 2 位數字顯示	55
S	一秒的分數，以分數表示	978
V	時區識別符，以 zone-id 顯示	America/Los_Angeles Z 08：30

日期部分或時間部分	意義	範例
x	區域偏離 UTC (offset-X)	+0000 -08 -0830 -08：30 -083015 -08：30：15
X	區域偏離 UTC；其中 Z 為零	Z -08 -0830 -08：30 -083015 -08：30：15
y	年份，以年份顯示	2020 20
z	時區名稱，以文字顯示	太平洋標準時間 PST
Z	區域偏離 UTC (offset-Z)	+0000 -0800 -08：00
'	文字逸出，以分隔符號顯示	N/A
''	單引號，以常值呈現	'

日期部分或時間部分	意義	範例
[	選用區段開始	N/A
]	選用區段結束	N/A

模式字母的數量決定格式類型：

文字格式

- 縮寫格式請使用 1-3 個字母（例如，星期一的「星期一」）
- 完整格式只能使用 4 個字母（例如，「星期一」）
- 請勿使用 5 個或多個字母 - 這會導致錯誤

數字格式 (n)

- 值 n 代表允許的字母數目上限
- 對於單一字母模式：
  - 輸出使用不含填補的最小數字
- 對於多個字母模式：
  - 輸出以零填補，以符合字母計數寬度
- 剖析時，輸入必須包含確切的位數

數字/文字格式

- 對於 3 個或更多字母，請遵循文字格式規則
- 對於較少的字母，請遵循數字格式規則

分數格式

- 使用 1-9 個 'S' 字元（例如 SSSSSS)
- 對於剖析：
  - 接受介於 1 和 S 字元數之間的分數
- 針對格式化：

    - 以零填補以符合 S 字元數
- 支援高達 6 位數的微秒精確度
- 可以剖析奈秒，但會截斷額外的數字

年格式

- 字母計數會設定填補的最小欄位寬度
- 對於兩個字母：
    - 列印最後兩位數字
    - 2000-2099 之間的剖析年數
- 對於少於四個字母（兩個字母除外）：
    - 僅顯示負數年份的符號
- 請勿使用 7 個或更多字母 - 這會導致錯誤

月格式

- 將 'M' 用於標準表單，將 'L' 用於獨立表單
- 單一 'M' 或 'L'：
    - 顯示不含填補的月編號 1-12

- 'MM' 或 'LL'：
    - 顯示含填補的月份編號 01-12
- 'MMM'：
    - 以標準格式顯示縮寫月份名稱
    - 必須是完整日期模式的一部分
- 'LLL'：
    - 以獨立形式顯示縮寫月份名稱
    - 用於僅限月份的格式
- 'MMMM'：
    - 以標準格式顯示完整月份名稱
    - 將 用於日期和時間戳記
- 'LLLL'：

- 以獨立形式顯示完整月份名稱
- 用於僅限月份的格式

時區格式

- am-pm：僅使用 1 個字母
- 區域 ID (V)：僅使用 2 個字母
- 區域名稱 (z)：
  - 1-3 個字母：顯示簡短名稱
  - 4 個字母：顯示全名
  - 請勿使用 5 個或更多字母

位移格式

- X 和 x：
  - 1 個字母：顯示小時 (+01) 或小時/分鐘 (+0130)
  - 2 個字母：顯示不含冒號的小時/分鐘 (+0130)
  - 3 個字母：顯示含冒號的小時/分鐘 (+01：30)
  - 4 個字母：顯示不含冒號hour-minute-second (+013015)
  - 5 個字母：顯示含冒號的hour-minute-second (+01：30：15)
  - X 使用 'Z' 進行零位移
  - x 使用 '+00'、'+0000' 或 '+00：00' 進行零位移
- O:
  - 1 個字母：顯示簡短格式 (GMT+8)
  - 4 個字母：顯示完整格式 (GMT+08：00)
- Z：
  - 1-3 個字母：顯示不含冒號的小時/分鐘 (+0130)
  - 4 個字母：顯示完整的當地語系化表單
  - 5 個字母：以冒號顯示hour-minute-second

選用區段

- 使用方括號 【 】 標記選用內容
- 您可以將選用區段巢狀化
- 所有有效的資料都會顯示在輸出中
- 輸入可以省略整個選用區段

> **Note**
>
> 符號 'E'、'F'、'q' 和 'Q' 僅適用於日期時間格式 （ 例如 date_format)。請勿將它們用於日期時間剖析 （ 例如 to_timestamp)。

數值格式字串

下列數值格式字串適用於 TO_NUMBER 和 TO_CHAR 等函數。

- 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	Description
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。
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。

格式	Description
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。

日期和時間函數

日期和時間函數可讓您對日期和時間資料執行各種操作，例如擷取日期的一部分、執行日期計算、格式化日期和時間，以及使用目前的日期和時間。這些函數對於資料分析、報告和涉及暫時資料的資料處理等任務至關重要。

AWS Clean Rooms 支援下列日期和時間函數：

主題

- ADD_MONTHS 函數
- CONVERT_TIMEZONE 函數
- CURRENT_DATE 函數
- CURRENT_TIMESTAMP 函數
- DATE_ADD 函數
- DATE_DIFF 函數
- DATE_PART 函數
- DATE_TRUNC 函數
- DAY 函數
- DAYOFMONTH 函數
- DAYOFWEEK 函數

- DAYOFYEAR 函數
- EXTRACT 函數
- FROM_UTC_TIMESTAMP 函數
- HOUR 函數
- MINUTE 函數
- MONTH 函數
- SECOND 函數
- TIMESTAMP 函數
- TO_TIMESTAMP 函數
- YEAR 函數
- 日期或時間戳記函數的日期部分

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。DATE_ADD 函數提供類似功能。

語法

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

引數

date | timestamp

日期或時間戳記欄，或隱含轉換為日期或時間戳記的表達式。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。此函式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日期 （ 在函數執行 AWS 區域 所在的 中 ）。

```
select current_date;

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

CURRENT_TIMESTAMP 函數

CURRENT_TIMESTAMP 會傳回目前的日期和時間，包括日期、時間和 （ 選擇性 ） 毫秒或微秒。

當您需要取得目前的日期和時間時，例如記錄事件的時間戳記、執行以時間為基礎的計算，或填入日期/時間資料欄時，此函數很有用。

語法

```
current_timestamp()
```

傳回類型

CURRENT_TIMESTAMP 函數會傳回 DATE。

範例

下列範例會在執行查詢時傳回目前的日期和時間，即 2020 年 4 月 25 日 15：49：11.914 （下午 3：49：11.914)。

```
SELECT current_timestamp();
 2020-04-25 15:49:11.914
```

下列範例會擷取squirrels資料表中每一列的目前日期和時間。

```
SELECT current_timestamp() FROM squirrels
```

DATE_ADD 函數

傳回 start_date 之後 num_days 的日期。

語法

```
date_add(start_date, num_days)
```

引數

start_date

開始日期值。

num_days

要新增的天數 （整數）。正數加上天，負數減去天。

傳回類型

DATE

範例

下列範例會將一天新增至日期：

```
SELECT date_add('2016-07-30', 1);

Result:
2016-07-31
```

下列範例會新增多天。

```
SELECT date_add('2016-07-30', 5);

Result:
2016-08-04
```

使用須知

本文件適用於 Spark SQL 的 DATE_ADD 函數，相較於某些其他 SQL 變體，它提供了更簡單的界面，可新增日期天數。若要新增其他間隔，例如月或年，可能需要不同的函數。

DATE_DIFF 函數

DATE_DIFF 傳回兩個日期或時間表達式的日期部分之間的差異。

語法

```
date_diff(endDate, startDate)
```

引數

endDate

DATE 表達式。

startDate

DATE 表達式。

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 date_diff(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 date_diff(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 date_diff(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 date_diff(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。DATE_DIFF 函數的預設欄名稱為 DATE_DIFF。

```
select date_diff(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,
date_diff(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(date_diff(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 date_diff(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 date_diff(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 date_diff(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 date_diff(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

```
datepart(field, source)
```

引數

欄位

應擷取來源的哪個部分，且支援的字串值與對等函數 EXTRACT 的欄位相同。

source

應從中擷取欄位的 DATE 或 INTERVAL 資料欄。

傳回類型

如果欄位為「SECOND」，則為 DECIMAL(8， 6)。在所有其他情況下，INTEGER。

範例

下列範例會從日期值擷取一年中的日期 (DOY)。輸出顯示日期 "2019-08-12" 的年份日期是 224。這表示 2019 年 8 月 12 日是 2019 年第 224 天。

```
SELECT datepart('doy', DATE'2019-08-12');
 224
```

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(format, datetime)
```

引數

format

代表要截斷之單位的格式。有效格式如下：

- "YEAR"、"YYYY"、"YY" - 截斷至 ts 所在年份的第一天，時間部分將為零
- "QUARTER" - 截斷至 ts 所在季度的第一天，時間部分將為零
- "MONTH"、"MM"、"MON" - 截斷至 ts 所在月份的第一天，時間部分將為零
- "WEEK" - 截斷至 ts 所在一週的星期一，時間部分將為零
- "DAY"、"DD" - 零時間部分
- "HOUR" - 以分數部分將分鐘和秒歸零
- "MINUTE"- 以分數部分將秒數歸零

- "SECOND" - 將第二個分數部分歸零
- "MILLISECOND" - 微秒零
- "MICROSECOND" - 所有項目仍保留

ts

日期時間值

傳回類型

傳回截斷至格式模型所指定單位的時間戳記

範例

下列範例會將日期值截斷為年初。輸出顯示日期 "2015-03-05" 已截斷為 "2015-01-01"，這是 2015 年的開始。

```
SELECT date_trunc('YEAR', '2015-03-05');

 date_trunc
-----------
2015-01-01
```

DAY 函數

DAY 函數會傳回日期/時間戳記的月份日期。

當您需要使用日期或時間戳記的特定元件時，例如執行以日期為基礎的計算、篩選資料或格式化日期值時，日期擷取函數非常有用。

語法

```
day(date)
```

引數

date

DATE 或 TIMESTAMP 表達式。

傳回值

DAY 函數會傳回 INTEGER。

範例

下列範例會從輸入日期 擷取當月的日期 (30)'2009-07-30'。

```
SELECT day('2009-07-30');
 30
```

下列範例會從squirrels資料表的 birthday 欄中擷取當月的日期，並將結果傳回為 SELECT 陳述式的輸出。此查詢的輸出將是日值的清單，squirrels表格中每一列各一個，代表每個松鼠生日的月份日期。

```
SELECT day(birthday) FROM squirrels
```

DAYOFMONTH 函數

DAYOFMONTH 函數會傳回日期/時間戳記的月份日期 (1 到 31 之間的值，取決於月份和年份）。

DAYOFMONTH 函數類似於 DAY 函數，但其名稱略有不同，行為略有不同。DAY 函數較常用，但 DAYOFMONTH 函數可作為替代函數。當您需要對包含日期或時間戳記資料的資料表執行日期型分析或篩選時，例如擷取日期的特定元件以進行進一步處理或報告時，這類查詢會很有用。

語法

```
dayofmonth(date)
```

引數

date

DATE 或 TIMESTAMP 表達式。

傳回值

DAYOFMONTH 函數會傳回 INTEGER。

範例

下列範例會從輸入日期 擷取當月的日期 (30)'2009-07-30'。

```
SELECT dayofmonth('2009-07-30');
 30
```

下列範例會將 DAYOFMONTH 函數套用至squirrels資料表的 birthday 欄。對於squirrels資料表中的每一列，將擷取資料birthday欄中當月的日期，並以 SELECT 陳述式的輸出傳回。此查詢的輸出將是日值清單，squirrels表格中每一列各一個，代表每個松鼠生日的月份日期。

```
SELECT dayofmonth(birthday) FROM squirrels
```

DAYOFWEEK 函數

DAYOFWEEK 函數以日期或時間戳記做為輸入，並以數字傳回星期幾 （ 星期日為 1，星期一為 2，...，星期六為 7)。

當您需要使用日期或時間戳記的特定元件時，例如執行以日期為基礎的計算、篩選資料或格式化日期值時，此日期擷取函數非常有用。

語法

```
dayofweek(date)
```

引數

date

DATE 或 TIMESTAMP 表達式。

傳回值

DAYOFWEEK 函數會傳回 INTEGER，其中

1 = 星期日

2 = 星期一

3 = 星期二

4 = 週三

5 = 星期四

6 = 星期五

7 = 週六

範例

下列範例會從這個日期擷取星期幾，即 5 （代表星期四）。

```
SELECT dayofweek('2009-07-30');
 5
```

下列範例會從squirrels資料表的 birthday 欄擷取星期幾，並將結果傳回為 SELECT 陳述式的輸出。此查詢的輸出將是星期幾值的清單，squirrels表格中每一列各一個，代表每個松鼠生日的星期幾。

```
SELECT dayofweek(birthday) FROM squirrels
```

DAYOFYEAR 函數

DAYOFYEAR 函數是一種日期擷取函數，採用日期或時間戳記做為輸入，並傳回一年中的某一天 （值介於 1 到 366 之間，取決於年份及其是否為閏年）。

當您需要使用日期或時間戳記的特定元件時，例如執行以日期為基礎的計算、篩選資料或格式化日期值時，此函數非常有用。

語法

```
dayofyear(date)
```

引數

date

DATE 或 TIMESTAMP 表達式。

傳回值

DAYOFYEAR 函數會傳回 INTEGER （介於 1 到 366 之間，取決於年份以及是否為閏年）。

範例

下列範例會從輸入日期 擷取一年中的日期 (100)'2016-04-09'。

```
SELECT dayofyear('2016-04-09');
 100
```

下列範例會從squirrels資料表的 birthday 欄擷取一年中的某一天，並將結果傳回為 SELECT 陳述式的輸出。

```
SELECT dayofyear(birthday) FROM squirrels
```

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。

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
```

FROM_UTC_TIMESTAMP 函數

FROM_UTC_TIMESTAMP 函數會將輸入日期從 UTC （國際標準時間） 轉換為指定的時區。

當您需要將日期和時間值從 UTC 轉換為特定時區時，此函數很有用。這在處理來自世界各地且需要在適當的當地時間呈現的資料時非常重要。

語法

```
from_utc_timestamp(timestamp, timezone
```

引數

timestamp

具有 UTC 時間戳記的 TIMESTAMP 表達式。

timezone

STRING 表達式是有效的時區，輸入日期或時間戳記應轉換為該時區。

傳回值

FROM_UTC_TIMESTAMP 函數會傳回 TIMESTAMP。

範例

下列範例會將輸入日期從 UTC 轉換為指定的時區 ('Asia/Seoul')，在此情況下為 UTC 之前的 9 小時。產生的輸出是首爾時區的日期和時間，即 2016-08-31 09:00:00。

```
SELECT from_utc_timestamp('2016-08-31', 'Asia/Seoul');
 2016-08-31 09:00:00
```

HOUR 函數

HOUR 函數是一種時間擷取函數，需要時間或時間戳記做為輸入，並傳回小時元件 （介於 0 和 23 之間的值）。

當您需要使用時間或時間戳記的特定元件時，例如執行以時間為基礎的計算、篩選資料或格式化時間值時，此時間擷取函數非常有用。

語法

```
hour(timestamp)
```

引數

timestamp

TIMESTAMP 表達式。

傳回值

HOUR 函數會傳回 INTEGER。

範例

下列範例會從輸入時間戳記 中擷取小時元件 (12)'2009-07-30 12:58:59'。

```
SELECT hour('2009-07-30 12:58:59');
 12
```

MINUTE 函數

MINUTE 函數是一種時間擷取函數，需要時間或時間戳記做為輸入，並傳回分鐘元件 （介於 0 和 60 之間的值）。

語法

```
minute(timestamp)
```

引數

timestamp

TIMESTAMP 表達式或有效時間戳記格式的 STRING。

傳回值

MINUTE 函數會傳回 INTEGER。

範例

下列範例會從輸入時間戳記 中擷取分鐘元件 (58)'2009-07-30 12:58:59'。

```
SELECT minute('2009-07-30 12:58:59');
 58
```

MONTH 函數

MONTH 函數是一種時間擷取函數，需要時間或時間戳記做為輸入，並傳回月份元件 （介於 0 和 12 之間的值）。

語法

```
month(date)
```

引數

date

TIMESTAMP 表達式或有效時間戳記格式的 STRING。

傳回值

MONTH 函數會傳回 INTEGER。

範例

下列範例會從輸入時間戳記 中擷取月份元件 (7)'2016-07-30'。

```
SELECT month('2016-07-30');
 7
```

SECOND 函數

SECOND 函數是一種時間擷取函數，需要時間或時間戳記做為輸入，並傳回第二個元件 （介於 0 和 60 之間的值）。

語法

```
second(timestamp)
```

引數

timestamp

TIMESTAMP 表達式。

傳回值

SECOND 函數會傳回 INTEGER。

範例

下列範例會從輸入時間戳記 中擷取第二個元件 (59)'2009-07-30 12:58:59'。

```
SELECT second('2009-07-30 12:58:59');
 59
```

TIMESTAMP 函數

TIMESTAMP 函數會取得值 （通常是數字），並將其轉換為時間戳記資料類型。

當您需要將代表時間或日期的數值轉換為時間戳記資料類型時，此函數很有用。當您使用以數字格式存放的資料時，例如 Unix 時間戳記或 epoch 時間，這會很有幫助。

語法

```
timestamp(expr)
```

引數

expr

可轉換為 TIMESTAMP 的任何表達式。

傳回值

TIMESTAMP 函數會傳回 TIMESTAMP。

範例

下列範例會將數字 Unix 時間戳記 (1632416400) 轉換為對應的時間戳記資料類型：2021 年 9 月 22 日下午 12：00：00 UTC。

```
SELECT timestamp(1632416400);
 2021-09-22 12:00:00 UTC
```

TO_TIMESTAMP 函數

TO_TIMESTAMP 會將 TIMESTAMP 字串轉換為 TIMESTAMPTZ。

語法

```
to_timestamp (timestamp)
```

```
to_timestamp (timestamp, format)
```

引數

timestamp

時間戳記字串或可轉換為時間戳記字串的資料類型。

format

符合 Spark 日期時間模式的字串常值。如需有效的日期時間模式，請參閱格式化和剖析的日期時間模式。

傳回類型

TIMESTAMP

範例

下列範例示範如何使用 TO_TIMESTAMP 函數將 TIMESTAMP 字串轉換為 TIMESTAMP。

```
select current_timestamp() as timestamp, to_timestamp( current_timestamp(), '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
```

下列 SQL 陳述式會將字串 '2011-12-18 24：38：15' 轉換為 TIMESTAMP。結果是 TIMESTAMP 落在第二天，因為時數超過 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
```

YEAR 函數

YEAR 函數是一種日期擷取函數，採用日期或時間戳記做為輸入，並傳回年份元件 （四位數字）。

語法

```
year(date)
```

引數

date

DATE 或 TIMESTAMP 表達式。

傳回值

YEAR 函數會傳回 INTEGER。

範例

下列範例會從輸入日期 擷取年份元件 (2016)'2016-07-30'。

```
SELECT year('2016-07-30');
 2016
```

下列範例會從squirrels資料表的 birthday 欄擷取年份元件，並將結果傳回為 SELECT 陳述式的輸出。此查詢的輸出將是年份值的清單，squirrels表格中每一列各一個，代表每個松鼠的生日年份。

```
SELECT year(birthday) FROM squirrels
```

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：

- DATE_ADD
- DATE_DIFF
- 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。

日期部分或時間部分	縮寫
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
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 解譯 CENTURY 以年份 ###1 開始，以年份 結束###0：

```
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 實作與 1970-01-01 00：00：00.00000 相關，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 根據常見行事曆解譯 DECADE 或 DECADES DATEPART。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 將 MIL 解譯為從年份 #001 的第一天開始，到年份的最後一天結束#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)
```

加密和解密函數

加密和解密函數可協助 SQL 開發人員在可讀取的純文字形式和不可讀取的加密文字形式之間轉換敏感資料，以防止未經授權的存取或濫用。

AWS Clean Rooms Spark SQL 支援下列加密和解密函數：

主題

- AES_ENCRYPT 函數
- AES_DECRYPT 函數

AES_ENCRYPT 函數

AES_ENCRYPT 函數用於使用進階加密標準 (AES) 演算法加密資料。

語法

```
aes_encrypt(expr, key[, mode[, padding[, iv[, aad]]]])
```

引數

expr

要加密的二進位值。

金鑰

用來加密資料的複雜密碼。

支援 16、24 和 32 位元的金鑰長度。

模式

指定應使用哪個區塊加密模式來加密訊息。

有效模式：ECB（電子 CodeBook)、GCM (Galois/Counter 模式）、CBC (Cipher-Block Chaining)。

填補

指定如何填補長度不是區塊大小倍數的訊息。

有效值：PKCS、NONE、DEFAULT。

DEFAULT 填補表示適用於 ECB 的 PKCS（公有金鑰密碼編譯標準）、適用於 GCM 的 NONE 和適用於 CBC 的 PKCS。

支援的 (模式、填充) 組合為 ('ECB'、'PKCS')、 ('GCM'、'NONE') 和 ('CBC'、'PKCS')。

iv

選用初始化向量 (IV)。僅支援 CBC 和 GCM 模式。

有效值：GCM 為 12 位元組長，CBC 為 16 位元組。

aad

選用的其他已驗證資料 (AAD)。僅支援 GCM 模式。這可以是任何自由格式的輸入，而且必須同時提供加密和解密。

傳回類型

AES_ENCRYPT 函數會在指定模式中使用 AES 傳回 expr 的加密值，並指定填充。

範例

下列範例示範如何使用 Spark SQL AES_ENCRYPT 函數，使用指定的加密金鑰安全地加密資料字串（在此案例中為「Spark」一詞）。產生的加密文字接著會經過 Base64-encoded，以便更輕鬆地儲存或傳輸。

```
SELECT base64(aes_encrypt('Spark', 'abcdefghijklmnop'));
```

```
  4A5jOAh9FNGwoMeuJukfllrLdHEZxA2DyuSQAWz77dfn
```

下列範例示範如何使用 Spark SQL AES_ENCRYPT 函數，使用指定的加密金鑰安全地加密資料字串（在此案例中為「Spark」一詞）。產生的密碼文字接著會以十六進位格式表示，對於資料儲存、傳輸或偵錯等任務很有用。

```
SELECT hex(aes_encrypt('Spark', '0000111122223333'));
 83F16B2AA704794132802D248E6BFD4E380078182D1544813898AC97E709B28A94
```

下列範例示範如何使用 Spark SQL AES_ENCRYPT 函數，使用指定的加密金鑰、加密模式和填充模式安全地加密資料字串（在此案例中為「Spark SQL」）。產生的加密文字接著會經過 Base64-encoded，以便更輕鬆地儲存或傳輸。

```
SELECT base64(aes_encrypt('Spark SQL', '1234567890abcdef', 'ECB', 'PKCS'));
 3lmwu+Mw0H3fi5NDvcu9lg==
```

AES_DECRYPT 函數

AES_DECRYPT 函數用於使用進階加密標準 (AES) 演算法解密資料。

語法

```
aes_decrypt(expr, key[, mode[, padding[, aad]]])
```

引數

expr

要解密的二進位值。

金鑰

用來解密資料的複雜密碼。

密碼短語必須符合最初用於產生加密值的金鑰，長度為 16、24 或 32 個位元組。

模式

指定應使用哪個區塊加密模式來解密訊息。

有效模式：ECB、GCM、CBC。

填補

指定如何填補長度不是區塊大小倍數的訊息。

有效值：PKCS、NONE、DEFAULT。

DEFAULT 填補表示適用於 ECB 的 PKCS、適用於 GCM 的 NONE 和適用於 CBC 的 PKCS。

aad

選用的其他已驗證資料 (AAD)。僅支援 GCM 模式。這可以是任何自由格式的輸入，而且必須同時提供加密和解密。

傳回類型

在 模式中使用 AES 搭配填充，傳回 expr 的解密值。

範例

下列範例示範如何使用 Spark SQL AES_ENCRYPT 函數，使用指定的加密金鑰安全地加密資料字串（在此案例中為「Spark」一詞）。產生的加密文字接著會經過 Base64-encoded，以便更輕鬆地儲存或傳輸。

```
SELECT base64(aes_encrypt('Spark', 'abcdefghijklmnop'));
  4A5jOAh9FNGwoMeuJukfllrLdHEZxA2DyuSQAWz77dfn
```

下列範例示範如何使用 Spark SQL AES_DECRYPT 函數來解密先前已加密和 Base64-encoded的資料。解密程序需要正確的加密金鑰和參數（加密模式和填補模式），才能成功復原原始純文字資料。

```
SELECT aes_decrypt(unbase64('3lmwu+Mw0H3fi5NDvcu9lg=='), '1234567890abcdef', 'ECB',
 'PKCS');
 Spark SQL
```

雜湊函數

雜湊函數是一種數學函數，可將數字輸入值轉換成另一個值。

AWS Clean Rooms Spark SQL 支援下列雜湊函數：

主題

- MD5 函數

- SHA 函數
- SHA1 函數
- SHA2 函數
- xxHASH64 函數

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);
```

xxHASH64 函數

xxhash64 函數會傳回引數的 64 位元雜湊值。

xxhash64() 函數是非加密雜湊函數，設計為快速且有效率。它通常用於資料處理和儲存應用程式，其中需要資料的唯一識別符，但資料的確切內容不需要保密。

在 SQL 查詢的內容中，xxhash64() 函數可用於各種用途，例如：

- 為資料表中的資料列產生唯一識別符
- 根據雜湊值分割資料
- 實作自訂索引或資料分佈策略

特定的使用案例取決於應用程式的需求和正在處理的資料。

語法

```
xxhash64(expr1, expr2, ...)
```

引數

expr1

任何類型的表達式。

expr2

任何類型的表達式。

傳回值

傳回引數 (BIGINT) 的 64 位元雜湊值。雜湊種子為 42。

範例

下列範例會根據提供的輸入產生 64 位元雜湊值 (5602566077635097486)。第一個引數是字串值，在此例中為「Spark」一詞。第二個引數是包含單一整數值 123 的陣列。第三個引數是整數值，代表雜湊函數的種子。

```
SELECT xxhash64('Spark', array(123), 2);
 5602566077635097486
```

Hyperloglog 函數

SQL 中的 HyperLogLog (HLL) 函數提供一種方式，可有效估計大型資料集中唯一元素 （基數） 的數量，即使未儲存一組實際的唯一元素也一樣。

使用 HLL 函數的主要優點如下：

- 記憶體效率：HLL 草圖比儲存整組唯一元素需要的記憶體要少得多，因此適合大型資料集。
- 分散式運算：HLL 草圖可以跨多個資料來源或處理節點組合，從而實現高效的分散式唯一計數估算。
- 近似結果：HLL 提供近似的唯一計數估算，在準確度和記憶體用量之間進行調整權衡 （透過精確度參數）。

這些函數在您需要估計唯一項目數量的情況下特別有用，例如分析、資料倉儲和即時串流處理應用程式。

AWS Clean Rooms 支援下列 HLL 函數。

主題

- HLL_SKETCH_AGG 函數
- HLL_SKETCH_ESTIMATE 函數

- HLL_UNION 函數
- HLL_UNION_AGG 函數

HLL_SKETCH_AGG 函數

HLL_SKETCH_AGG 彙總函數會從指定欄中的值建立 HLL 草圖。它會傳回封裝輸入表達式值的 HLLSKETCH 資料類型。

HLL_SKETCH_AGG 彙總函數適用於任何資料類型，並忽略 NULL 值。

如果表格中沒有列或所有列都為 NULL，則產生的草圖沒有索引-值對，例如 `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}`。

語法

```
HLL_SKETCH_AGG (aggregate_expression[, lgConfigK ] )
```

引數

aggregate_expression

任何類型 INT、BIGINT、STRING 或 BINARY 的表達式，其將發生唯一計數。會忽略任何NULL值。

lgConfigK

選用的 INT 常數，介於 4 到 21 之間，預設值為 12。K 的 log-base-2，其中 K 是草圖的儲存貯體或插槽數目。

傳回類型

HLL_SKETCH_AGG 函數會傳回非 Null BINARY 緩衝區，其中包含計算的 HyperLogLog 草圖，因為會耗用和彙總彙總群組中的所有輸入值。

範例

下列範例使用 HyperLogLog (HLL) 演算法來估計資料col欄中值的不同計數。`hll_sketch_agg(col, 12)` 函數會彙總 col 欄中的值，使用精確度 12 建立 HLL 草圖。然後，該hll_sketch_estimate()函數會用來根據產生的 HLL 草圖來估計不同的值計數。查詢的最終結果為 3，代表資料col欄中值的預估不同計數。在此情況下，不同的值為 1、2 和 3。

```
SELECT hll_sketch_estimate(hll_sketch_agg(col, 12))
    FROM VALUES (1), (1), (2), (2), (3) tab(col);
  3
```

下列範例也會使用 HLL 演算法來估算資料col欄中值的不同計數，但不會指定 HLL 草圖的精確度值。在這種情況下，它會使用預設精確度 14。hll_sketch_agg(col) 函數會取得資料col欄中的值，並建立 HyperLogLog (HLL) 草圖，這是可用來估計不同元素計數的精簡資料結構。hll_sketch_estimate(hll_sketch_agg(col)) 函數會採用上一個步驟中建立的 HLL 草圖，並計算資料col欄中不同值計數的估計值。查詢的最終結果為 3，代表資料col欄中值的預估不同計數。在此情況下，不同的值為 1、2 和 3。

```
SELECT hll_sketch_estimate(hll_sketch_agg(col))
FROM VALUES (1), (1), (2), (2), (3) tab(col);
3
```

HLL_SKETCH_ESTIMATE 函數

HLL_SKETCH_ESTIMATE 函數採用 HLL 草圖，並估計草圖所代表的唯一元素數量。它使用 HyperLogLog (HLL) 演算法來計算指定資料欄中唯一值數量的概率近似值，使用先前由 HLL_SKETCH_AGG 函數產生的草圖緩衝區二進位表示法，並將結果傳回為大整數。

HLL 素描演算法提供一種有效方法來估算唯一元素的數量，即使是大型資料集也一樣，而不必存放完整的一組唯一值。

hll_union 和 hll_union_agg函數也可以將這些緩衝區作為輸入耗用並合併，以將草圖結合在一起。

語法

```
HLL_SKETCH_ESTIMATE (hllsketch_expression)
```

引數

hllsketch_expression

保留 HLL_SKETCH_AGG 產生的草圖的BINARY表達式

傳回類型

HLL_SKETCH_ESTIMATE 函數會傳回 BIGINT 值，該值是輸入草圖表示的近似差異計數。

範例

下列範例使用 HyperLogLog (HLL) 素描演算法來估計col資料欄中值的基數 （唯一計數）。hll_sketch_agg(col, 12) 函數會取得 col資料欄，並使用 12 位元的精確度建立 HLL 草圖。HLL 草圖是一種近似資料結構，可有效估計集合中唯一元素的數量。hll_sketch_estimate() 函數會採用由 建立的 HLL 草圖，hll_sketch_agg並預估草圖所代表值的基數 （唯一計數）。FROM VALUES (1), (1), (2), (2), (3) tab(col); 會產生具有 5 列的測試資料集，其中資料col欄包含值 1、1、2、2 和 3。此查詢的結果是資料col欄中值的估計唯一計數，即 3。

```
SELECT hll_sketch_estimate(hll_sketch_agg(col, 12))
    FROM VALUES (1), (1), (2), (2), (3) tab(col);
  3
```

下列範例和上一個範例之間的差異在於未在hll_sketch_agg函數呼叫中指定精確度參數 (12 位元）。在此情況下，會使用 14 位元的預設精確度，相較於先前使用 12 位元精確度的範例，這可提供更準確的唯一計數預估。

```
SELECT hll_sketch_estimate(hll_sketch_agg(col))
FROM VALUES (1), (1), (2), (2), (3) tab(col);
3
```

HLL_UNION 函數

HLL_UNION 函數將兩個 HLL 草圖合併為單一的統一草圖。它使用 HyperLogLog (HLL) 演算法，將兩個草圖合併為單一草圖。查詢可以使用產生的緩衝區，以hll_sketch_estimate函數的長整數來計算近似的唯一計數。

語法

```
HLL_UNION (( expr1, expr2 [, allowDifferentLgConfigK ] ))
```

引數

exprN

保留 HLL_SKETCH_AGG 產生的草圖的BINARY表達式。

allowDifferentLgConfigK

選用的 BOOLEAN 表達式，可控制是否允許將兩個草圖與不同的 lgConfigK 值合併。預設值為false。

傳回類型

HLL_UNION 函數會傳回 BINARY 緩衝區，其中包含因合併輸入表達式而計算的 HyperLogLog 草圖。當 allowDifferentLgConfigK 參數為 時true，結果草圖會使用兩個所提供lgConfigK值中較小的值。

範例

下列範例使用 HyperLogLog (HLL) 素描演算法來估計資料集col2中兩欄 col1和 之間的唯一值計數。

hll_sketch_agg(col1) 函數會為資料col1欄中的唯一值建立 HLL 草圖。

hll_sketch_agg(col2) 函數會為 col2 欄中的唯一值建立 HLL 草圖。

hll_union(...) 函數將步驟 1 和 2 中建立的兩個 HLL 草圖合併為單一、統一的 HLL 草圖。

hll_sketch_estimate(...) 函數會採用合併的 HLL 草圖，並預估 col1和 中值的唯一計數col2。

FROM VALUES 子句會產生一個包含 5 列的測試資料集，其中 col1包含值 1、1、2、2 和 3，而 col2包含值 4、4、5、5 和 6。

此查詢的結果是跨 col1和 值的估計唯一計數col2，即 6。HLL 素描演算法提供一種有效方法來估算唯一元素的數量，即使是大型資料集也一樣，而不必存放完整的一組唯一值。在此範例中，hll_union函數會用來結合兩個資料欄的 HLL 草圖，這允許在整個資料集中估計唯一計數，而不只是針對每個資料欄個別估算。

```
SELECT hll_sketch_estimate(
  hll_union(
    hll_sketch_agg(col1),
    hll_sketch_agg(col2)))
  FROM VALUES
    (1, 4),
    (1, 4),
    (2, 5),
    (2, 5),
    (3, 6) AS tab(col1, col2);
  6
```

下列範例和上一個範例之間的差異在於未在hll_sketch_agg函數呼叫中指定精確度參數 (12 位元）。在此情況下，會使用 14 位元的預設精確度，相較於先前使用 12 位元精確度的範例，這可提供更準確的唯一計數預估。

```
SELECT hll_sketch_estimate(
  hll_union(
    hll_sketch_agg(col1, 14),
    hll_sketch_agg(col2, 14)))
  FROM VALUES
    (1, 4),
    (1, 4),
    (2, 5),
    (2, 5),
    (3, 6) AS tab(col1, col2);
```

HLL_UNION_AGG 函數

HLL_UNION_AGG 函數將多個 HLL 草圖合併為單一的統一草圖。它使用 HyperLogLog (HLL) 演算法，將一組草圖合併為單一草圖。查詢可以使用產生的緩衝區，透過 `hll_sketch_estimate`函數計算近似的唯一計數。

語法

```
HLL_UNION_AGG ( expr [, allowDifferentLgConfigK ] )
```

引數

expr

保留 HLL_SKETCH_AGG 產生的草圖的`BINARY`表達式。

allowDifferentLgConfigK

選用的 BOOLEAN 表達式，可控制是否允許將兩個草圖與不同的 lgConfigK 值合併。預設值為 `false`。

傳回類型

HLL_UNION_AGG 函數會傳回 BINARY 緩衝區，其中包含 HyperLogLog 草圖，這是由於合併相同群組的輸入表達式所計算的結果。當 `allowDifferentLgConfigK` 參數為 時`true`，結果草圖會使用兩個所提供`lgConfigK`值中較小的值。

範例

下列範例使用 HyperLogLog (HLL) 草圖演算法來估計多個 HLL 草圖中值的唯一計數。

第一個範例預估資料集中值的唯一計數。

```
SELECT hll_sketch_estimate(hll_union_agg(sketch, true))
    FROM (SELECT hll_sketch_agg(col) as sketch
            FROM VALUES (1) AS tab(col)
          UNION ALL
          SELECT hll_sketch_agg(col, 20) as sketch
            FROM VALUES (1) AS tab(col));
  1
```

內部查詢會建立兩個 HLL 草圖：

- 第一個 SELECT 陳述式會從單一值 1 建立草圖。
- 第二個 SELECT 陳述式會從另一個單一值 1 建立草圖，但精確度為 20。

外部查詢使用 HLL_UNION_AGG 函數，將兩個草圖合併為單一草圖。然後，它會將 HLL_SKETCH_ESTIMATE 函數套用到此合併草圖，以估算值的唯一計數。

此查詢的結果是資料`col`欄中值的估計唯一計數，即 1。這表示 1 的兩個輸入值被視為唯一，即使它們具有相同的值。

第二個範例包含 HLL_UNION_AGG 函數的不同精確度參數。在這種情況下，兩個 HLL 草圖都是以 14 位元的精確度建立，這可讓它們成功`hll_union_agg`與 `true` 參數搭配使用。

```
SELECT hll_sketch_estimate(hll_union_agg(sketch, true))
    FROM (SELECT hll_sketch_agg(col, 14) as sketch
            FROM VALUES (1) AS tab(col)
          UNION ALL
          SELECT hll_sketch_agg(col, 14) as sketch
            FROM VALUES (1) AS tab(col));
  1
```

查詢的最終結果是估計的唯一計數，在這種情況下，它也是 1。這表示 1 的兩個輸入值被視為唯一，即使它們具有相同的值。

JSON 函數

當您需要儲存相當小的一組金鑰值對時，您可以用 JSON 格式儲存資料以節省空間。因為 JSON 字串可儲存於單一欄，採用 JSON 可能比以資料表格式儲存資料更有效率。

Example

例如，假設您有一個稀疏的資料表，其中您需要許多資料欄才能完全代表所有可能的屬性。不過，任何指定資料列或任何指定資料欄的大部分資料欄值都是 NULL。透過使用 JSON 進行儲存，您可以在單一 JSON 字串中將資料列的資料存放在鍵值對中，並消除稀疏填入的資料表資料欄。

此外，您可以輕鬆修改 JSON 字串來儲存其他金鑰:值對，而不需要在資料表中新增欄。

建議少用 JSON。JSON 不是存放較大資料集的好選擇，因為透過將不同的資料儲存在單一資料欄中，JSON 不會使用 AWS Clean Rooms 資料欄存放區架構。

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 金鑰:值對中，金鑰是雙引號括住的字串。
- JSON 值可以是下列任何值：
  - JSON 物件
  - JSON 陣列
  - 雙引號中的字串
  - 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 `{ }, [ ]`)。

主題

GET_JSON_OBJECT 函數

GET_JSON_OBJECT 函數會從 擷取 json 物件path。

語法

```
get_json_object(json_txt, path)
```

引數

json_txt

包含格式良好的 JSON 的 STRING 表達式。

路徑

具有格式良好的 JSON 路徑表達式的 STRING 常值。

傳回值

傳回 STRING。

如果找不到物件，則會傳回 NULL。

範例

下列範例會從 JSON 物件擷取值。第一個引數是 JSON 字串，代表具有單一鍵值對的簡單物件。第二個引數是 JSON 路徑表達式。$ 符號 代表 JSON 物件的根，而 .a部分指定我們要擷取與 "a" 金鑰相關聯的值。函數的輸出是 'b'，這是與輸入 JSON 物件中的「a」鍵相關聯的值。

```
SELECT get_json_object('{"a":"b"}', '$.a');
 b
```

TO_JSON 函數

TO_JSON 函數會將輸入表達式轉換為 JSON 字串表示法。函數會處理將不同資料類型 （例如數字、字串和布林值） 轉換為其對應的 JSON 表示法。

當您需要將結構化資料（例如資料庫資料列或 JSON 物件）轉換為更具可攜式、自我描述的格式，例如 JSON 時，TO_JSON 函數很有用。當您需要與預期 JSON 格式資料的其他系統或服務互動時，這特別有用。

語法

```
to_json(expr[, options])
```

引數

expr

您要轉換為 JSON 字串的輸入表達式。它可以是值、資料欄或任何其他有效的 SQL 表達式。

options

選用的組態選項集，可用於自訂 JSON 轉換程序。這些選項可能包括 Null 值的處理方式、數值的表示方式，以及特殊字元的處理方式。

傳回值

傳回具有指定結構值的 JSON 字串

範例

下列範例會將具名結構（結構化資料的類型）轉換為 JSON 字串。第一個引數 (named_struct('a', 1, 'b', 2)) 是傳遞給to_json()函數的輸入表達式。它會建立具有兩個欄位的具名結構：值為 1 的 "a" 和值為 2 的 "b"。to_json() 函數採用具名結構做為其引數，並將其轉換為 JSON 字串表示法。輸出為 {"a":1,"b":2}，這是代表具名結構的有效 JSON 字串。

```
SELECT to_json(named_struct('a', 1, 'b', 2));
 {"a":1,"b":2}
```

下列範例會將包含時間戳記值的具名結構轉換為具有自訂時間戳記格式的 JSON 字串。第一個引數 (named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd'))) 會使用包含時間戳記值的單一欄位 'time' 建立具名結構。第二個引數 (map('timestampFormat', 'dd/MM/yyyy')) 會使用單一索引鍵/值對建立映射（索引鍵/值字典），其中索引鍵為 'timestampFormat'，而值為 'dd/MM/yyyy'。此映射用於在轉換為 JSON 時指定時間戳記值的所需格式。to_json() 函數會將具名結構轉換為 JSON 字串。第二個引數映射用於自訂時間戳記格式為 'dd/MM/yyyy'。輸出為

{"time":"26/08/2015"}，這是具有單一欄位 'time' 的 JSON 字串，其中包含所需 'dd/MM/yyyy' 格式的時間戳記值。

```
SELECT to_json(named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd')),
 map('timestampFormat', 'dd/MM/yyyy'));
 {"time":"26/08/2015"}
```

數學函數

本節說明 AWS Clean Rooms Spark SQL 中支援的數學運算子和函數。

主題

- 數學運算子符號
- ABS 函數
- ACOS 函數
- ASIN 函數
- ATAN 函數
- ATAN2 函數
- CBRT 函數
- CEILING (或 CEIL) 函數
- COS 函數
- COT 函數
- DEGREES 函數
- DIV 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
- RAND 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	Description	範例	結果
+	加法	2 + 3	5
-	減法	2 - 3	-1
*	乘法	2 * 3	6
/	除法	4 /2	2
%	模數	5 % 4	1
^	指數	2.0 ^ 3.0	8

範例

計算已支付的佣金加上指定交易的 2.00 美元處理費：

```
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。該結果在 10 年間以指數方式乘以 5% 的持續成長率。

```
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)
```

尋找日期 ID 大於或等於 2，000 之銷售的已支付總價和佣金。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 SMALLINT、INTEGER、BIGINT、DECIMAL、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 SMALLINT、INTEGER、BIGINT、DECIMAL、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 |
+----------+
```

DIV 函數

DIV 運算子會傳回除以除數除以除以除以除以除法的積分部分。

語法

```
dividend div divisor
```

引數

股利

評估為數值或間隔的表達式。

除數

如果 dividend是間隔，則為數字，則為相符的間隔類型。

傳回類型

BIGINT

範例

下列範例會從 squirrels 資料表中選取兩個資料欄： id資料欄，其中包含每個 squirrel 的唯一識別符，以及 calculated資料欄age div 2，代表年齡欄的整數除以 2。age div 2 計算會在資料age欄上執行整數分割，有效地將存留期四捨五入至最接近的偶整數。例如，如果資料age欄包含 3、5、7 和 10 等值，則資料age div 2欄將分別包含值 1、2、3 和 5。

```
SELECT id, age div 2 FROM squirrels
```

此查詢在您需要根據存留期範圍分組或分析資料的情況下非常有用，而且您想要將存留期值四捨五入到最接近的偶整數。產生的輸出會提供squirrels資料表中每個 squirrel 的 id和存留期除以 2。

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

表達式

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 SMALLINT、INTEGER、BIGINT、DECIMAL、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 函數會傳回輸入參數的自然對數。

語法

```
LN(expression)
```

引數

表達式

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，則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

expr 使用 傳回 的對數base。

語法

```
LOG(base, expr)
```

引數

expr

表達式必須為整數、小數或浮點數資料類型。

base

對數計算的基礎。必須是雙精確度資料類型的正數 (不等於 1)。

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(10, 100);
--------
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 % 做為模數運算子。

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
```

透過將第一個參數除以 2 來檢查它是否為 ：

```
SELECT mod(5,2) = 0 as is_even;

 is_even
--------
 false
```

您可以使用 % 做為模數運算子：

```
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

```
{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 |
+-------------------+
```

RAND 函數

RAND 函數會產生介於 0 和 1 之間的隨機浮點數。RAND 函數會在每次呼叫時產生新的隨機數字。

語法

```
RAND()
```

傳回類型

RANDOM 傳回 DOUBLE。

範例

下列範例會為squirrels資料表中的每一列產生介於 0 到 1 之間的隨機浮點數字資料欄。產生的輸出會是包含隨機十進位值清單的單一資料欄，在 squirrels 資料表中，每一列各有一個值。

```
SELECT rand() FROM squirrels
```

當您需要產生隨機數字時，例如模擬隨機事件或將隨機性引入資料分析時，這類查詢非常有用。在squirrels資料表的內容中，它可能會用來為每個 squirrel 指派隨機值，然後可用於進一步處理或分析。

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. 此範例使用 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引數作為整數，以指出任一方向的四捨五入小數位數。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。指定第二個引數 >n 時，函數會四捨五入到具有小數位數 n 的最近數字。

語法

```
ROUND (number [ , integer ] )
```

引數

number

數字或評估為數字的運算式。它可以是 DECIMAL 或 FLOAT8 類型。 AWS Clean Rooms 可以根據隱含轉換規則轉換其他資料類型。

整數 (選用)

整數，指出朝任一方向四捨五入的小數位數。

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 DECIMALor FLOAT8 type。 AWS Clean Rooms 可以根據隱含轉換規則轉換其他資料類型。

傳回類型

SIGN 傳回與輸入引數相同的數值資料類型（輸入引數）。如果輸入是 DECIMAL，則輸出是 DECIMAL(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

表達式

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類型轉換。

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類型轉換。

```
select sqrt('16');

sqrt
---------------
4
```

下列範例會巢狀函數以執行更複雜的任務。

```
select sqrt(round(16.4));

sqrt
---------------
4
```

下列範例會在指定圓形區域時產生半徑的長度。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 COMMISSION 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引數作為整數，以指出任一方向的四捨五入小數位數。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。指定第二個引數 >n 時，函數會四捨五入至小數位數 >n 的最近數字。此函數也會截斷時間戳記並傳回日期。

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 DECIMAL 或 FLOAT8 類型。 AWS Clean Rooms 可以根據隱含轉換規則轉換其他資料類型。

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

函數也可以從時間戳記傳回日期。（若要以 傳回時間戳記值00:00:00作為時間，請將函數結果轉換為時間戳記。）

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)
```

從 SYSDATE 函數的結果傳回日期部分 （傳回時間戳記）：

```
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)
```

純量函數

本節說明 AWS Clean Rooms Spark SQL 中支援的純量函數。純量函數是將一或多個值作為輸入，並將單一值傳回為輸出的函數。純量函數在個別資料列或元素上運作，並為每個輸入產生單一結果。

純量函數，例如 SIZE，與其他類型的 SQL 函數不同，例如彙總函數（計數、總和、平均值）和表格產生函數（爆炸、平面）。這些其他函數類型可在多個資料列上操作或產生多個資料列，而純量函數則適用於個別的資料列或元素。

主題

- SIZE 函數

SIZE 函數

SIZE 函數會將現有的陣列、映射或字串做為引數，並傳回代表該資料結構大小或長度的單一值。它不會建立新的資料結構。它用於查詢和分析現有資料結構的屬性，而不是建立新的資料結構。

此函數有助於判斷陣列中的元素數量或字串長度。在 SQL 中使用陣列和其他資料結構時特別有用，因為它可讓您取得有關資料大小或基數的資訊。

語法

```
size(expr)
```

引數

expr

ARRAY、MAP 或 STRING 表達式。

傳回類型

SIZE 函數會傳回 INTEGER。

範例

在此範例中，SIZE 函數會套用至陣列 `['b', 'd', 'c', 'a']`，並傳回值 4，這是陣列中的元素數目。

```
SELECT size(array('b', 'd', 'c', 'a'));
 4
```

在此範例中，SIZE 函數會套用至映射 `{'a': 1, 'b': 2}`，並傳回值 2，這是映射中鍵值對的數量。

```
SELECT size(map('a', 1, 'b', 2));
 2
```

在此範例中，SIZE 函數會套用至字串 `'hello world'`，並傳回值 11，也就是字串中的字元數。

```
SELECT size('hello world');
11
```

字串函數

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。

主題

- || (串連) 運算子
- BTRIM 函數
- CONCAT 函數
- FORMAT_STRING 函數
- LEFT 和 RIGHT 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
- REVERSE 函數
- RTRIM 函數
- SPLIT 函數
- SPLIT_PART 函數
- SUBSTRING 函數
- TRANSLATE 函數
- TRIM 函數
- UPPER 函數
- UUID 函數

|| (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
```

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。

如果輸入表達式的類型不同， 會 AWS Clean Rooms 嘗試隱含輸入轉換其中一個表達式。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)
```

FORMAT_STRING 函數

FORMAT_STRING 函數會以提供的引數取代範本字串中的預留位置，以建立格式化字串。它會從 printf 格式格式字串傳回格式化字串。

FORMAT_STRING 函數的運作方式是將範本字串中的預留位置取代為以引數傳遞的對應值。當您需要動態建構包含靜態文字和動態資料的字串時，例如產生輸出訊息、報告或其他類型的資訊性文字時，這種類型的字串格式很有用。FORMAT_STRING 函數提供簡潔且可讀取的方式來建立這些類型的格式化字串，讓您更輕鬆地維護和更新產生輸出的程式碼。

語法

```
format_string(strfmt, obj, ...)
```

引數

strfmt

STRING 表達式。

obj

STRING 或數值表達式。

傳回類型

FORMAT_STRING 會傳回 STRING。

範例

下列範例包含一個範本字串，其中包含兩個預留位置：%d十進位 （整數） 值，以及%s字串值。%d 預留位置會取代為小數 （整數） 值 (100)，而 %s 預留位置則會取代為字串值 ("days")。輸出是範本字串，預留位置由提供的引數取代："Hello World 100 days"。

```
SELECT format_string("Hello World %d %s", 100, "days");
 Hello World 100 days
```

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)
```

LENGTH 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 VARCHAR 字串 （或任何其他資料類型，例如 CHAR，可隱含轉換為 VARCHAR)。

傳回類型

LOWER 函數會傳回與輸入字串相同資料類型的字元字串。

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。當修剪字元未出現在輸入字串中時，修剪即完成。

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。

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
```

下列範例會使用不區分大小寫的比對FOX，來計算字串 的出現次數。

```
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比對邏輯FOX，尋找字串 第一次出現的開始位置。

```
SELECT regexp_instr('the fox', 'FOX', 1, 1, 0, 'i');

 regexp_instr
 --------------
            5
```

下列範例使用以 PCRE 方言編寫的模式來尋找包含至少一個數字和一個小寫字母的單字。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
```

下列範例使用以 PCRE 方言編寫的模式來尋找包含至少一個數字和一個小寫字母的單字。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
```

下列範例會將電子郵件地址的網域名稱取代為此值：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內出現的所有字串。

```
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。此範例會將這類單字的每個出現次數取代為值 [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。此範例會將這類單字的每次出現取代為值 [hidden]，但與先前的範例不同，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
```

下列範例使用不區分大小寫的比對FOX，傳回與字串 第一次出現對應的輸入部分。

```
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。

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)
```

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。當修剪字元未出現在輸入字串中時，修剪即完成。

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。

SPLIT 函數

SPLIT 函數可讓您從較大的字串擷取子字串，並將其用作陣列。當您需要根據特定分隔符號或模式將字串分解為個別元件時，SPLIT 函數非常有用。

語法

```
split(str, regex, limit)
```

引數

str

要分割的字串運算式。

regex

代表規則表達式的字串。regex 字串應該是 Java 規則表達式。

limit

整數表達式，可控制套用 regex 的次數。

- 限制 > 0：產生的陣列長度不會超過限制，且產生的陣列最後一個項目將包含超過最後一個相符 regex 的所有輸入。
- limit <= 0：regex 將盡可能套用多次，且產生的陣列可以是任何大小。

傳回類型

SPLIT 函數會傳回 ARRAY<STRING>。

如果 `limit > 0`：產生的陣列長度不會超過限制，且產生的陣列最後一個項目將包含超過最後一個相符 regex 的所有輸入。

如果 `limit <= 0`：regex 將盡可能套用多次，而產生的陣列可以是任何大小。

範例

在此範例中，SPLIT 函數會在遇到字元 'A'、 'B'或 'C'（如規則表達式模式 所指定）'oneAtwoBthreeC'時分割輸入字串'[ABC]'。產生的輸出是四個元素的陣列："one"、"three"、"two"和空字串 ""。

```
SELECT split('oneAtwoBthreeC', '[ABC]');
 ["one","two","three",""]
```

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
```

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(charactestring FROM start_position [ FOR numbecharacters ] )
```

```
SUBSTRING(charactestring, start_position, numbecharacters )
```

```
SUBSTRING(binary_expression, start_byte, numbebytes )
```

```
SUBSTRING(binary_expression, start_byte )
```

引數

特徵化

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。

numbecharacters

要擷取的字元數 (子字串的長度)。numbecharacters 是根據字元數而非位元組數，因此多位元組字元會計為單一字元。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。

numbebyte

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

字元字串的使用備註

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 + numbecharacters 超過字串的長度，SUBSTRING 會從 start_position 傳回子字串，直到字串結束。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 + numbecharacters -1 的子字串。例如：

```
select substring('caterpillar',-2,6);
substring
-----------
cat
(1 row)
```

如果 start_position + numbecharacters -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
```

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。如果您搭配 SQL 命令使用 TRIM 函數， 會 AWS Clean Rooms 隱含地將結果轉換為 VARCHAR。如果您在 SQL 函數的 SELECT 清單中使用 TRIM 函數，AWS Clean Rooms 不會隱含地轉換結果，而且您可能需要執行明確轉換，以避免資料類型不相符錯誤。如需明確轉換的相關資訊，請參閱 CAST 函數函數。

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
```

當任何字元出現在字串開頭時，TRIM 會移除 trim_chars 中的任何字元。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, 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 VARCHAR 字串（或任何其他資料類型，例如 CHAR，可隱含轉換為 VARCHAR)。

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)
```

UUID 函數

UUID 函數會產生全域唯一識別符 (UUID)。

UUIDs是全域唯一識別符，通常用於為各種目的提供唯一識別符，例如：

- 識別資料庫記錄或其他資料實體。
- 產生檔案、目錄或其他資源的唯一名稱或金鑰。
- 跨分散式系統追蹤和關聯資料。

- 為網路封包、軟體元件或其他數位資產提供唯一識別符。

UUID 函數會產生具有極高機率的唯一 UUID 值，即使在分散式系統和長時間內也是如此。UUIDs通常使用目前時間戳記、電腦網路地址和其他隨機或虛擬隨機資料的組合產生，以確保每個產生的 UUID 都非常不可能與任何其他 UUID 衝突。

在 SQL 查詢的內容中，UUID 函數可用來為插入資料庫的新記錄產生唯一識別符，或為資料分割、索引或其他需要唯一識別符的目的提供唯一索引鍵。

> **Note**
>
> UUID 函數是非確定性的。

語法

```
uuid()
```

引數

UUID 函數不採用任何引數。

傳回類型

UUID 傳回通用唯一識別符 (UUID) 字串。值會以正式 UUID 36 字元字串傳回。

範例

下列範例會產生通用唯一識別符 (UUID)。輸出是代表全域唯一識別符的 36 個字元字串。

```
SELECT uuid();
 46707d92-02f4-4817-8116-a4c3b23e6266
```

隱私權相關函數

AWS Clean Rooms 提供 函數，可協助您符合下列規格的隱私權相關合規。

- 全球隱私權平台 (GPP) – 互動式廣告局 (IAB) 的規格，可建立適用於線上隱私權和資料使用的全球標準化架構。如需 GPP 技術規格的詳細資訊，請參閱 GitHub 上的全球隱私權平台文件。

- 透明度和同意架構 (TCF) – GPP 的關鍵元件，於 2020 年推出，提供標準化的技術架構，協助公司遵守隱私權法規，例如歐盟一般資料保護法規 (GDPR)。TCF 可讓客戶授予或暫停對資料收集和處理的同意。如需 TCF 技術規格的詳細資訊，請參閱 GitHub 上的 TCF 文件。

主題

consent_gpp_v1_decode 函數

`consent_gpp_v1_decode` 函數用於解碼全球隱私權平台 (GPP) v1 同意資料。它以編碼的同意字串做為輸入，並傳回解碼的同意資料，其中包括使用者隱私權偏好設定和同意選擇的相關資訊。此函數在處理包含 GPP v1 同意資訊的資料時非常有用，因為它可讓您以結構化格式存取和分析同意資料。

語法

```
consent_gpp_v1_decode(gpp_string)
```

引數

gpp_string

編碼的 GPP v1 同意字串。

傳回值

傳回的字典包含下列鍵值對：

- `version`：使用的 GPP 規格版本 （ 目前為 1)。
- `cmpId`：編碼同意字串的同意管理平台 (CMP) ID。
- `cmpVersion`：編碼同意字串的 CMP 版本。
- `consentScreen`：使用者提供同意的 CMP UI 中的畫面 ID。
- `consentLanguage`：同意資訊的語言代碼。
- `vendorListVersion`：使用的廠商清單版本。
- `publisherCountryCode`：發佈者的國家/地區代碼。
- `purposeConsent`：整數清單，代表使用者同意的目的。

- purposeLegitimateInterest：已透明地傳達使用者合法利益的目的 IDs 清單。
- specialFeatureOptIns：整數清單，代表使用者已選擇加入的特殊功能。
- vendorConsent：使用者已同意的供應商 IDs清單。
- vendorLegitimateInterest：已透明地傳達使用者合法利益的供應商 IDs 清單。

範例

下列範例採用單一引數，也就是編碼的同意字串。它會傳回包含解碼同意資料的字典，包括使用者隱私權偏好設定、同意選擇和其他中繼資料的相關資訊。

```
SELECT * FROM consent_gpp_v1_decode('ABCDEFGHIJK');
```

傳回之同意資料的基本結構包括同意字串版本、CMP（同意管理平台）詳細資訊、使用者對不同用途和供應商的同意和合法利益選擇，以及其他中繼資料的相關資訊。

```
{
    "version": 1,
    "cmpId": 12,
    "cmpVersion": 34,
    "consentScreen": 5,
    "consentLanguage": "en",
    "vendorListVersion": 89,
    "publisherCountryCode": "US",
    "purposeConsent": [1],
    "purposeLegitimateInterests": [1],
    "specialFeatureOptins": [1],
    "vendorConsent": [1],
    "vendorLegitimateInterests": [1]}
}
```

consent_tcf_v2_decode 函數

consent_tcf_v2_decode 函數用於解碼透明度和同意架構 (TCF) v2 同意資料。它以編碼的同意字串做為輸入，並傳回解碼的同意資料，其中包括使用者隱私權偏好設定和同意選擇的相關資訊。此函數在處理包含 TCF v2 同意資訊的資料時非常有用，因為它可讓您以結構化格式存取和分析同意資料。

語法

```
consent_tcf_v2_decode(tcf_string)
```

引數

tcf_string

編碼的 TCF v2 同意字串。

傳回值

consent_tcf_v2_decode 函數會傳回字典，其中包含來自透明度和同意架構 (TCF) v2 同意字串的解碼同意資料。

傳回的字典包含下列鍵值對：

核心區段

- version：使用的 TCF 規格版本 （目前為 2)。
- created：建立同意字串的日期和時間。
- lastUpdated：上次更新同意字串的日期和時間。
- cmpId：編碼同意字串的同意管理平台 (CMP) ID。
- cmpVersion：編碼同意字串的 CMP 版本。
- consentScreen：使用者提供同意的 CMP UI 中的畫面 ID。
- consentLanguage：同意資訊的語言代碼。
- vendorListVersion：使用的廠商清單版本。
- tcfPolicyVersion：同意字串所依據的 TCF 政策版本。
- isServiceSpecific：布林值，指出同意是針對特定服務，還是適用於所有服務。
- useNonStandardStacks：布林值，指出是否使用非標準堆疊。
- specialFeatureOptIns：整數清單，代表使用者已選擇加入的特殊功能。
- purposeConsent：整數清單，代表使用者同意的目的。
- purposesLITransparency：整數清單，代表使用者已提供合法利益透明度的目的。
- purposeOneTreatment：布林值，指出使用者是否已請求「目的一處理」（也就是說，所有目的都會受到同等對待）。
- publisherCountryCode：發佈者的國家/地區代碼。
- vendorConsent：使用者已同意的供應商 IDs清單。

- vendorLegitimateInterest：已透明地傳達使用者合法利益的供應商 IDs 清單。
- pubRestrictionEntry：發佈者限制的清單。此欄位包含用途 ID、限制類型，以及該用途限制下的廠商 IDs 清單。

已公開的供應商客群

- disclosedVendors：整數清單，代表已向使用者公開的廠商。

發佈者用途區段

- pubPurposesConsent：整數清單，代表使用者已同意的發佈者特定目的。
- pubPurposesLITransparency：整數清單，代表使用者已提供合法利益透明度的發佈者特定目的。
- customPurposesConsent：整數清單，代表使用者已同意的自訂目的。
- customPurposesLITransparency：整數清單，代表使用者已提供合法利益透明度的自訂目的。

此詳細同意資料可用於了解和尊重使用者使用個人資料時的隱私權偏好設定。

範例

下列範例採用單一引數，也就是編碼的同意字串。它會傳回包含解碼同意資料的字典，包括使用者隱私權偏好設定、同意選擇和其他中繼資料的相關資訊。

```
from aws_clean_rooms.functions import consent_tcf_v2_decode

consent_string = "CO1234567890abcdef"
consent_data = consent_tcf_v2_decode(consent_string)

print(consent_data)
```

傳回之同意資料的基本結構包括同意字串版本、CMP（同意管理平台）詳細資訊、使用者對不同用途和供應商的同意和合法利益選擇，以及其他中繼資料的相關資訊。

```
    /** core segment **/
    version: 2,
    created: "2023-10-01T12:00:00Z",
    lastUpdated: "2023-10-01T12:00:00Z",
```

```
    cmpId: 1234,
    cmpVersion: 5,
    consentScreen: 1,
    consentLanguage: "en",
    vendorListVersion: 2,
    tcfPolicyVersion: 2,
    isServiceSpecific: false,
    useNonStandardStacks: false,
    specialFeatureOptIns: [1, 2, 3],
    purposeConsent: [1, 2, 3],
    purposesLITransparency: [1, 2, 3],
    purposeOneTreatment: true,
    publisherCountryCode: "US",
    vendorConsent: [1, 2, 3],
    vendorLegitimateInterest: [1, 2, 3],
    pubRestrictionEntry: [
        { purpose: 1, restrictionType: 2, restrictionDescription: "Example
 restriction" },
    ],

    /** disclosed vendor segment **/
    disclosedVendors: [1, 2, 3],

    /** publisher purposes  segment **/
    pubPurposesConsent: [1, 2, 3],
    pubPurposesLITransparency: [1, 2, 3],
    customPurposesConsent: [1, 2, 3],
    customPurposesLITransparency: [1, 2, 3],
};
```

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不支援 PARTITION BY 子句中的字串常值。

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不支援 ORDER BY 子句中的字串常值。

如果省略 ORDER BY 子句，則列的順序不確定。

> **Note**
>
> 在任何平行系統中AWS Clean Rooms，例如，當 ORDER BY 子句未產生資料的唯一和總排序時，資料列的順序是非確定性的。也就是說，如果 ORDER BY 表達式產生重複的值（部分排序），則這些資料列的傳回順序可能會因執行 而有所不同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 RoomsSpark SQL 支援兩種類型的視窗函數：彙總和排名。

以下是支援的彙總函數：

- CUME_DIST 範圍函數
- DENSE_RANK 範圍函數
- FIRST 視窗函數
- FIRST_VALUE 範圍函數

- LAG 範圍函數
- LAST 視窗函數
- LAST_VALUE 範圍函數
- LEAD 範圍函數

以下是支援的排名函數：

- DENSE_RANK 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。有些範例使用名為 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	

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
30004	4/18/2004	3	B	20	
30007	9/7/2004	3	C	30	

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

下列範例會依銷售數量排序資料表（以遞減順序），並將密集排名和一般排名指派給每一列。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 分割資料表，並依數量排序每個分割區 （以遞減順序），並將密集排名指派給每一列。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 視窗函數

根據一組列的排序，FIRST 會傳回指定表達式相對於視窗框架中第一列的值。

如需有關選取視窗框中最後一列的資訊，請參閱LAST 視窗函數。

語法

```
FIRST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引數

表達式

函數運算的目標欄或表達式。

IGNORE NULLS

當此選項與 FIRST 搭配使用時，函數會傳回框架中不是 NULL 的第一個值 （如果所有值都是 NULL，則為 NULL)。

RESPECT NULLS

指出 AWS Clean Rooms應該在決定要使用的資料列時包含 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 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

這些函數支援使用基本AWS Clean Rooms資料類型的表達式。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)。FIRST 函數用於選取與影格中第一列對應的場地名稱：在此情況下，為座位數量最高的列。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(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
-----------+------------+-------------------------------
+-----------------------------
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
...
```

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

表達式

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 AWS Clean Rooms應該在決定要使用的資料列時包含 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

這些函數支援使用基本AWS Clean Rooms資料類型的表達式。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。如果您未指定位移， AWS Clean Rooms會使用 1做為預設值。位移 0 表示目前列。

IGNORE NULLS

選用規格，指出 AWS Clean Rooms應該在決定要使用的資料列時略過 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 AWS Clean Rooms應該在決定要使用的資料列時包含 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資料類型的表達式。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 視窗函數

指定一組有序的資料列，LAST 函數會傳回相對於影格中最後一列的表達式值。

如需有關選取框架中第一列的資訊，請參閱FIRST 視窗函數。

語法

```
LAST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引數

表達式

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 AWS Clean Rooms應該在決定要使用的資料列時包含 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

這些函數支援使用基本AWS Clean Rooms資料類型的表達式。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 函數用於選取與影格中最後一列對應的場地名稱：在此情況下，為座位數最少的資料列。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(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
-----------+------------+--------------------------------
+------------------------------
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
...
```

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

表達式

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 AWS Clean Rooms應該在決定要使用的資料列時包含 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

這些函數支援使用基本AWS Clean Rooms資料類型的表達式。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。如果您未指定位移， AWS Clean Rooms會使用 1做為預設值。位移 0 表示目前列。

IGNORE NULLS

選用規格，指出 AWS Clean Rooms應該在決定要使用的資料列時略過 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 AWS Clean Rooms應該在決定要使用的資料列時包含 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資料類型的表達式。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)
```

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。

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。排名條件具有相同值的資料列會收到相同的排名。 會將繫結的資料列數目AWS Clean Rooms新增至繫結的排名，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 子句包含資料欄 2 和 1，以確保每次執行此查詢時都會AWS Clean Rooms傳回一致排序的結果。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)
```

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。

AWS Clean Rooms Spark SQL 條件

條件是一或多個表達式的陳述式，以及評估為 true、false 或 unknown 的邏輯運算子。條件有時也稱為述詞。

語法

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
```

```
| EXISTS_condition
| IN_condition
```

> **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。

AWS Clean Rooms Spark SQL 支援下列 SQL 條件。

主題

比較運算子

比較條件表示兩個值之間的邏輯關係。所有比較條件都是二元運算子，具有 Boolean 傳回類型。

AWS Clean Rooms Spark SQL 支援下表所述的比較運算子。

運算子	語法	描述
!	`!expression`	邏輯NOT運算子。用來否定布林表達式，表示傳回與表達式值相反的 。 ！ 運算子也可以與其他邏輯運算子結合，例如 AND 和 OR，以建立更複雜的布林表達式。

運算子	語法	描述
<	a < b	小於比較運算子。用來比較兩個值，並判斷左側的值是否小於右側的值。
>	a > b	大於比較運算子。用來比較兩個值，並判斷左側的值是否大於右側的值。
<=	a <= b	小於或等於比較運算子。用來比較兩個值，true如果左側的值小於或等於右側的值，則傳回 ，false否則傳回 。
>=	a >= b	大於或等於比較運算子。用來比較兩個值，並判斷左側的值是否大於或等於右側的值。
=	a = b	等式比較運算子，會比較兩個值，如果它們相等true則傳回 ，false否則傳回 。
<> 或 !=	a <> b 或 a != b	不等於比較運算子，它會比較兩個值，如果不相等true則傳回 ，false否則傳回 。

運算子	語法	描述
==	a == b	標準等式比較運算子，會比較兩個值，如果它們相等`true`則傳回 ，`false`否則傳回 。 **Note** == 運算子在比較字串值時區分大小寫。如果您需要執行不區分大小寫的比較，您可以使用 UPPER() 或 LOWER() 等函數，在比較之前將值轉換為相同的大小寫。

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
```

下列查詢會傳回目前未偽造之所有 squirrel 的 ID 值。

```
SELECT id FROM squirrels
WHERE !is_foraging
```

下列查詢會從 VENUE 資料表傳回超過 10，000 個座位的場地：

```
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
```

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
```

下列範例會擷取每個 timetz_val 中的小時數。

```
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 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
```

下列範例只會選取小於 3:00:00 UTC 的 TIMETZ 值。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 TIMETZ 常值。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

E1	E2	E1 AND E2	E1 OR E2	NOT E2
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

模式比對運算子會搜尋字串，尋找條件式表達式中指定的模式，並根據找到相符項目傳回 true 或 false。 AWS Clean Rooms Spark SQL 會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 會執行區分大小寫的配對。

主題

- LIKE
- RLIKE

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。

語法

```
expression [ NOT ] LIKE | 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

LIKE 會進行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，請在具有 LIKE 條件的 expression 和 pattern 上使用 LOWER 函數。

相較於比較述詞，例如 = 和 <>，LIKE 述詞不會隱含地忽略結尾空格。若要忽略結尾空格，請 RTRIM 或將 CHAR 資料欄明確轉換為 VARCHAR。

~~ 運算子等同於 LIKE。此外，運算!~~子相當於 NOT LIKE。

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' 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
...
```

下列範例會尋找第三個和第四個字元為「ea」的城市。：

```
select distinct city from users where city like '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

下列的範例使用預設的逸出字串 (\\)，來搜尋包含「_」的字串 (文字 start 後跟底線 _)：

```
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 (文字 start 後跟底線 _)：

```
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)
```

RLIKE

RLIKE 運算子可讓您檢查字串是否符合指定的規則表達式模式。

`true` 如果 str 符合 `regexp`，`false`否則傳回 。

語法

```
rlike(str, regexp)
```

引數

str

字串表達式

regexp

字串表達式。regex 字串應該是 Java 規則表達式。

SQL 剖析器中不會逸出字串常值 （包括 regex 模式）。例如，若要符合 "\abc"，regexp 的規則表達式可以是 "^\abc$"。

範例

下列範例會將spark.sql.parser.escapedStringLiterals組態參數的值設定為 true。此參數專屬於 Spark SQL 引擎。Spark SQL 中的 spark.sql.parser.escapedStringLiterals 參數控制 SQL 剖析器如何處理逸出的字串常值。設為 時true，剖析器會將字串常值中的反斜線字元 (\) 解譯為逸出字元，可讓您在字串值中包含特殊字元，例如換行符號、標籤和引號。

```
SET spark.sql.parser.escapedStringLiterals=true;
spark.sql.parser.escapedStringLiterals  true
```

例如，透過 spark.sql.parser.escapedStringLiterals=true，您可以在 SQL 查詢中使用下列字串常值：

```
SELECT 'Hello, world!\n'
```

換行字元\n會在輸出中解譯為常值換行字元。

下列範例會執行規則表達式模式比對。第一個引數會傳遞給 RLIKE 運算子。這是代表檔案路徑的字串，其中實際使用者名稱會以模式 '****' 取代。第二個引數是用於比對的規則表達式模式。輸出 (true) 表示第一個字串 ('%SystemDrive%\Users****') 符合規則表達式模式 ()'%SystemDrive%\\Users.*'。

```
SELECT rlike('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

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)
```

下列範例示範使用 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 的範圍包含在內，但日期預設的時間值為 00:00:00。範例查詢的唯一有效 1 月 3 日列是銷售時間為 1/3/2008 00:00:00 的列。

Null 條件

當值遺失或未知時，NULL條件會測試 null。

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碼，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

IN 條件會測試一組值或子查詢中成員資格的值。

語法

```
expression [ NOT ] IN (expr_list | table_subquery)
```

引數

表達式

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。支援這個最佳化功能的包括 SMALLINT、INTEGER、BIGINT、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP 和 TIMESTAMPTZ 等資料類型。

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)
```

查詢巢狀資料

AWS Clean Rooms 提供關聯式和巢狀資料的 SQL 相容存取。

AWS Clean Rooms 存取巢狀資料時， 會使用虛線符號和陣列下標進行路徑導覽。它還允許FROM子句項目在陣列上反覆運算，並用於不巢狀操作。下列主題提供不同查詢模式的描述，這些查詢模式將array/struct/map資料類型的使用與路徑和陣列導覽、取消巢狀和聯結結合結合。

主題

Navigation (導覽)

AWS Clean Rooms 可分別使用[...]括號和點符號來導覽陣列和結構。此外，您也可以使用點符號將導覽混合到結構中，以及使用括號符號將導覽混合到陣列中。

Example

例如，下列範例查詢假設c_orders陣列資料欄是具有 結構的陣列，而 屬性名為 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符號，例如篩選、聯結和彙總。您可以在查詢中使用這些符號，其中通常有欄參考。

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;
```

解除巢狀化查詢

若要巢狀查詢， AWS Clean Rooms 請啟用陣列的反覆運算。它透過使用查詢的 FROM 子句導覽陣列來實現此目的。

Example

使用上一個範例，下列範例會迭代 c_orders 的屬性值。

```
SELECT o FROM customer_orders_lineitem c, c.c_orders o;
```

解除巢狀化語法是 FROM 子句的擴充功能。在標準 SQL 中，FROM 子句 x （AS） y 代表 y 迭代關係 x 的每個元組。在這種情況下，x 指的是關係，而 y 指的是關係 x 的別名。同樣地，使用 FROM 子句項目取消巢狀的語法x （AS） y表示y反覆運算陣列表達式 中的每個值x。在此情況下， x 是陣列表達式，y也是 的別名x。

左運算元也可以使用點和括號符號進行常規導覽。

Example

在上一個範例中：

- customer_orders_lineitem c 是customer_order_lineitem基礎資料表上的反覆運算
- c.c_orders o 是對 的反覆運算 c.c_orders array

若要迭代 o_lineitems 屬性 (即陣列中的陣列)，您可以新增多個子句。

```
SELECT o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

AWS Clean Rooms 也支援使用 AT關鍵字在陣列上反覆運算的陣列索引。子句會反覆x AS y AT z運算陣列，x並產生欄位 z，也就是陣列索引。

Example

下列範例顯示陣列索引的運作方式。

```
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

下列範例會迭代純量陣列。

```
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

下列範例會迭代多個層級的陣列。這個範例會使用多個解除巢狀化子句來迭代到最內層的陣列。f.multi_level_array AS 陣列會透過 反覆運算multi_level_array。陣列AS元素是 內陣列的反覆運算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

根據預設，巢狀資料值的導覽操作會傳回 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，或者巢狀資料值是物件，但不包含查詢中使用的屬性名稱，則物件導覽會無效。

Example

例如，下列查詢會存取巢狀資料欄 中的無效屬性名稱`c_orders`：

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

如果巢狀資料值不是陣列或陣列索引超出範圍，則陣列導覽會傳回 null。

Example

下列查詢傳回 null`c_orders[1][1]`，因為 超出範圍。

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

自我檢查的種類

巢狀資料類型資料欄支援檢查函數，可傳回 類型和有關 值的其他類型資訊。AWS Clean Rooms 支援下列布林值函數的巢狀資料欄：

- DECIMAL_PRECISION
- DECIMAL_SCALE
- IS_ARRAY
- IS_BIGINT

- IS_CHAR
- IS_DECIMAL
- IS_FLOAT
- IS_INTEGER
- IS_OBJECT
- IS_SCALAR
- IS_SMALLINT
- IS_VARCHAR
- JSON_TYPEOF

如果輸入值為 null，所有這些函數傳回 false。IS_SCALAR、IS_OBJECT 和 IS_ARRAY 是互斥的，涵蓋除 null 之外的所有可能值。若要推斷與資料對應的類型， AWS Clean Rooms 會使用 JSON_TYPEOF 函數，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，必須為所使用的瀏覽器啟用 RSS 外掛程式。

變更	描述	日期
Spark SQL 支援提示	AWS Clean Rooms Spark SQL 支援查詢提示，以最佳化查詢效能並降低運算成本。	2026 年 1 月 20 日
Spark SQL 支援 CACHE TABLE	AWS Clean Rooms Spark SQL 支援 CACHE TABLE 命令，可讓客戶快取現有資料表，或從查詢結果建立和快取新資料表，以提升查詢效能。	2025 年 10 月 22 日
Spark SQL 支援 FIRST 和 LAST Window 函數	AWS Clean Rooms Spark SQL 支援下列視窗函數：FIRST 和 LAST。	2025 年 6 月 12 日
Spark SQL 函數文件更新	僅文件更新可準確反映支援的 Spark SQL 函數。已移除 25 個不支援函數的文件，包括 <=> 運算子、SIMILAR TO、LISTAGG 和 ARRAY_INSERT。將函數名稱從 DATEADD 更正為 DATE_ADD、DATEDIFF 更正為 DATE_DIFF、ISNULL 更正為 IS_NULL，ISNOTNULL 更正為 IS_NOT_NULL。修正 DATE_PART 範例中的錯別字。	2025 年 5 月 20 日

AWS Clean Rooms Spark SQL	客戶現在可以使用 Spark SQL 分析引擎支援的一些 SQL 條件、函數、命令和慣例來執行查詢。	2024 年 10 月 29 日
SQL 命令和 SQL 函數 – 更新	已新增 JOIN 子句、EXCEPT 設定運算子、CASE 條件式表達式和下列函數的範例： ANY_VALUE、NVL 和 COALESCE、NULLIF、CAST、CONVERT、CONVERT_TIMEZONE、EXTRACT、MOD、SIGN、CONCAT、FIRST_VALUE 和 LAST_VALUE。	2024 年 2 月 28 日
SQL 函數 - 更新	AWS Clean Rooms 現在支援下列 SQL 函數：Array、SUPER 和 VARBYTE。現在支援下列數學函數：ACOS、ASIN、ATAN、ATAN2、COT、DEXP、PI、POW、RADIANS 和 SIN。現在支援下列 JSON 函數：CAN_JSON_PARSE、JSON_PARSE 和 JSON_SERIALIZE。	2023 年 10 月 6 日
巢狀資料類型支援	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	僅文件變更以釐清預留資料欄名稱。	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。