Referencia de SQL

AWS Clean Rooms

AWS Clean Rooms: Referencia de SQL

Table of Contents

Descripción general de SQL en AWS Clean Rooms

Le damos la bienvenida a Referencia de SQL en AWS Clean Rooms.

AWS Clean Roomsse basa en el lenguaje de consulta estructurado (SQL) estándar del sector, un lenguaje de consulta que consta de comandos y funciones que se utilizan para trabajar con bases de datos y objetos de bases de datos. SQL también aplica reglas relativas al uso de tipos de datos, expresiones y literales.

En los temas siguientes se proporciona información general sobre las convenciones y las reglas de nomenclatura utilizadas en esta referencia de SQL.

Temas

- Convenciones de referencia a SQL
- Reglas de nomenclatura de SQL
- Soporte de tipos de datos mediante el motor SQL

En las siguientes secciones se proporciona información sobre los literales, los tipos de datos, los comandos SQL, los tipos de funciones SQL y las condiciones SQL en AWS Clean Rooms las que puede utilizar.

- AWS Clean Rooms Spark SQL

Para obtener más informaciónAWS Clean Rooms, consulte la Guía del AWS Clean Rooms usuario y la Referencia de la AWS Clean Rooms API.

Convenciones de referencia a SQL

En esta sección se explican las convenciones que se utilizan para escribir la sintaxis de las expresiones, los comandos y las funciones SQL.

Carácter	Descripción
CAPS	Las palabras en mayúscula son palabras clave.
[ ]	Los corchetes denotan argumentos opcionales. Varios argumentos entre corchetes indican que

Carácter	Descripción
	puede seleccionar cualquier cantidad de argumento s. Además, los argumentos entre corchetes en líneas separadas indican que el analizador de espera que los argumentos estén en el orden que aparecen en la sintaxis.
{ }	Las llaves indican que debe seleccionar uno de los argumentos contenidos en las llaves.
|	Las barras verticales indican que puede seleccionar entre los argumentos.
cursiva	Las palabras en cursiva indican marcadores de posición. Debe insertar el valor adecuado en lugar de la palabra en cursiva.
...	Los puntos suspensivos indican que puede repetir el elemento anterior.
'	Las palabras entre comillas simples indican que debe escribir las comillas.

Reglas de nomenclatura de SQL

En las siguientes secciones se explican las reglas de nomenclatura de SQL de AWS Clean Rooms.

Temas

Columnas y nombres de asociación de tablas configuradas

Los miembros que pueden realizar consultas usan nombres de asociación de tablas configuradas como nombres de tabla en las consultas. Los nombres de asociación de tablas configuradas y las columnas de tablas configuradas pueden designarse por un alias en las consultas.

Las siguientes reglas de nomenclatura se aplican a los nombres de asociación de tablas configuradas, a los nombres de columnas de tablas configuradas y a los alias:

- Deben utilizar únicamente caracteres alfanuméricos, de subrayado (_) o de guión (-), pero no pueden empezar ni terminar con un guion.
  - (Solo para reglas de análisis personalizadas) Pueden usar el signo de dólar ($), pero no pueden usar un patrón que siga una constante de cadena cotizada en dólares.

    Una constante de cadena citada entre dólares consta de:
    - un símbolo de dólar ($)
    - una "etiqueta" opcional de cero o más caracteres
    - otro símbolo de dólar
    - secuencia arbitraria de caracteres que componen el contenido de la cadena
    - un símbolo de dólar ($)
    - la misma etiqueta con la que comenzó la citación entre dólares
    - un símbolo de dólar

    Por ejemplo: `$$invalid$$`
- No pueden contener guiones (-) consecutivos.
- No pueden empezar con ninguno de los siguientes prefijos:

  `padb_`, `pg_`, `stcs_`, `stl_`, `stll_`, `stv_`, `svcs_`, `svl_`, `svv_`, `sys_`, `systable_`
- No pueden contener caracteres de barra invertida (\), comillas (') ni espacios que no estén entre comillas dobles.
- Si comienzan con un carácter no alfabético, deben estar entre comillas dobles (" ").
- Si contienen un carácter de guion (-), deben estar entre comillas dobles (" ").
- Deben tener una longitud de entre 1 y 127 caracteres.
- Las palabras reservadas deben estar entre comillas dobles (" ").
- Los siguientes nombres de columna están reservados y no se pueden usar AWS Clean Rooms (ni siquiera entre comillas):
  - oid
  - tableoid
  - xmin
  - cmin

- xmax
- cmax
- ctid

Palabras reservadas

La siguiente es una lista de palabras reservadas en AWS Clean Rooms.

AES128	DELTA32KDESC	LEADING	PRIMARY
AES256ALL	DISTINCT	LEFTLIKE	RAW
ALLOWOVER WRITEANALYSE	DO	LIMIT	READRATIO
ANALYZE	DISABLE	LOCALTIME	RECOVERRE FERENCES
AND	ELSE	LOCALTIMESTAMP	REJECTLOG
ANY	EMPTYASNU LLENABLE	LUN	RESORT
ARRAY	ENCODE	LUNS	RESPECT
AS	ENCRYPT	LZO	RESTORE
ASC	ENCRYPTIONEND	LZOP	RIGHTSELECT
AUTHORIZATION	EXCEPT	MINUS	SESSION_USER
AZ64	EXPLICITFALSE	MOSTLY16	SIMILAR
BACKUPBETWEEN	FOR	MOSTLY32	SNAPSHOT
BINARY	FOREIGN	MOSTLY8NATURAL	SOME
BLANKSASN ULLBOTH	FREEZE	NEW	SYSDATESYSTEM

BYTEDICT	FROM	NOT	TABLE
BZIP2CASE	FULL	NOTNULL	TAG
CAST	GLOBALDICT256	NULL	TDES
CHECK	GLOBALDIC T64KGRANT	NULLSOFF	TEXT255
COLLATE	GROUP	OFFLINEOFFSET	TEXT32KTHEN
COLUMN	GZIPHAVING	OID	TIMESTAMP
CONSTRAINT	IDENTITY	OLD	TO
CREATE	IGNOREILIKE	ON	TOPTRAILING
CREDENTIA LSCROSS	IN	ONLY	TRUE
CURRENT_DATE	INITIALLY	OPEN	TRUNCATEC OLUMNSUNION
CURRENT_TIME	INNER	OR	UNIQUE
CURRENT_T IMESTAMP	INTERSECT	ORDER	UNNEST
CURRENT_USER	INTERVAL	OUTER	USING
CURRENT_U SER_IDDEFAULT	INTO	OVERLAPS	VERBOSE
DEFERRABLE	IS	PARALLELP ARTITION	WALLETWHEN
DEFLATE	ISNULL	PERCENT	WHERE
DEFRAG	JOIN	PERMISSIONS	WITH
DELTA	LANGUAGE	PIVOTPLACING	WITHOUT

Soporte de tipos de datos mediante el motor SQL

AWS Clean Rooms admite varios motores y dialectos de SQL. Comprender los sistemas de tipos de datos en estas implementaciones es crucial para el éxito de la colaboración y el análisis de los datos. En las siguientes tablas se muestran los tipos de datos equivalentes en AWS Clean Rooms SQL, Snowflake SQL y Spark SQL.

Tipos de datos numéricos

Los tipos numéricos representan varios tipos de números, desde números enteros precisos hasta valores aproximados de punto flotante. La elección del tipo numérico afecta tanto a los requisitos de almacenamiento como a la precisión computacional. Los tipos de enteros varían según el tamaño del byte, mientras que los tipos decimales y de punto flotante ofrecen diferentes opciones de precisión y escala.

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
Entero de 8 bytes	BIGINT	No compatible	BIGINT, LARGO	Enteros firmados comprendidos entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.
Entero de 4 bytes	INT	No compatible	INT, INTEGER	Enteros con signo de -2.147.483.648 a 2.147.483.647
Entero de 2 bytes	SMALLINT	No compatible	SMALLINT, CORTO	Números enteros firmados de

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
				-32.768 a 32.767
Entero de 1 byte	No admitido	No admitido	TINYINT, BYTE	Enteros con signo del -128 al 127
Flotador de doble precisión	DOBLE, DOBLE PRECISIÓN	FLOTANTE FLOAT4 FLOAT8, DOBLE, DOBLE PRECISIÓN, REAL	DOUBLE	Números de coma flotante de doble precisión de 8 bytes
Flotador de precisión única	REAL, FLOTANTE	No compatible	FLOAT	números de coma flotante de precisión única de 4 bytes

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
Decimal (precisión fija)	DECIMAL	DECIMAL, NUMÉRICO, NÚMERO **Note** Snowflake asigna automáticamente el alias NUMBER a los tipos numéricos exactos de menor ancho (INT, BIGINT, SMALLINT, etc.).	DECIMAL, NUMÉRICO,	Números decimales con signo de precisión arbitraria
Decimal (con precisión)	DECIMAL (p)	DECIMAL (p), NÚMERO (p)	DECIMAL (p)	Números decimales de precisión fija
Decimal (con escala)	DECIMAL(p,s)	DECIMAL (p, s), NÚMERO (p, s)	DECIMAL(p,s)	Números decimales de precisión fija con escala

Tipos de datos booleanos

Los tipos booleanos representan valores lógicos simples. true/false Estos tipos son coherentes en todos los motores de SQL y se suelen utilizar para indicadores, condiciones y operaciones lógicas.

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
Booleano	BOOLEAN	BOOLEAN	BOOLEANO	Representa valores true/ false

Tipos de datos de fecha y hora

Los tipos de fecha y hora gestionan datos temporales, con distintos niveles de precisión y reconocimiento de la zona horaria. Estos tipos admiten diferentes formatos para almacenar fechas, horas y marcas horarias, con opciones para incluir o excluir información sobre la zona horaria.

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
Fecha	DATE	DATE	DATE	Valores de fecha (año, mes, día) sin zona horaria
Time	TIME	No admitido	No admitido	Hora del día en UTC, sin zona horaria
Hora con TZ	TIMETZ	No admitido	No admitido	Hora del día en UTC, con zona horaria
Timestamp	TIMESTAMP	TIMESTAMP , TIMESTAMP _NTZ	TIMESTAMP _NTZ	TIMESTAMP sin zona horaria

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
				Note NTZ indica «Sin zona horaria»
Marca de tiempo con TZ	TIMESTAMPTZ	TIMESTAMP_LTZ	TIMESTAMP, TIMESTAMP_LTZ	Marca de tiempo con zona horaria local Note LTZ indica «zona horaria local»

Tipos de datos de caracteres

Los tipos de caracteres almacenan datos textuales y ofrecen opciones de longitud fija y longitud variable. Estos tipos manejan cadenas de texto y datos binarios, con especificaciones de longitud opcionales para controlar la asignación del almacenamiento.

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
Carácter de longitud fija	CHAR	CHAR, CHARACTER	CHAR, CHARACTER	Cadena de caracteres de longitud fija
Carácter de longitud fija con longitud	CHAR(n)	CHAR(n), CHARACTER (n)	CHAR(n), CHARACTER (n)	Cadena de caracteres de longitud fija con una longitud especificada
Carácter de longitud variable	VARCHAR	VARCHAR, CADENA, TEXTO	VARCHAR, CADENA	Cadena de caracteres de longitud variable
Carácter de longitud variable con longitud	VARCHAR(n)	VARCHAR (n), STRING (n), TEXT (n)	VARCHAR(n)	Cadena de caracteres de longitud variable con límite de longitud
Binario	VARBYTE	BINARY, VARBINARY	BINARIO	Secuencia de bytes binarios
Binario con longitud	VARBYTE(n)	No admitido	No admitido	Secuencia binaria de bytes con límite de longitud

Tipos de datos estructurados

Los tipos estructurados permiten una organización de datos compleja al combinar varios valores en campos únicos. Estos incluyen matrices para recopilaciones ordenadas, mapas para pares clave-valor y estructuras para crear estructuras de datos personalizadas con campos con nombres.

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
Matriz	MATRIZ <type>	ARRAY (tipo)	MATRIZ <type>	Secuencia ordenada de elementos del mismo tipo **Note** Los tipos de matriz deben contener elementos del mismo tipo
Asignación	MAPA<key, value>	MAP (clave, valor)	MAPA<key, value>	Colección de pares clave-valor **Note** Los tipos de mapas

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
				deben contener elementos del mismo tipo
Struct	ESTRUCTURA< field1: type1, field2: type2>	OBJETO (campo1 tipo1, campo2 tipo2)	ESTRUCTURA< field1: type1, field2: type2 >	Estructura con campos con nombre de tipos específicos **Note** La sintaxis de los tipos estructurados puede variar ligeramente de una implementación a otra

Tipo de datos:	AWS Clean Rooms SQL	SQL Snowflake	Spark SQL	Description (Descripción)
Super	SUPER	No admitido	No admitido	Tipo flexible que admite todos los tipos de datos, incluidos los tipos complejos

AWS Clean Rooms Spark SQL

AWS Clean Rooms Spark SQL aplica las reglas relativas al uso de tipos de datos, expresiones y literales.

Para obtener más información sobre AWS Clean Rooms Spark SQL, consulta la Guía del AWS Clean Rooms usuario y la Referencia de la AWS Clean Rooms API.

Los siguientes temas proporcionan información sobre los literales, los tipos de datos, los comandos, las funciones y las condiciones compatibles con AWS Clean Rooms Spark SQL.

Temas

Literales

Un literal o una constante es un valor de dato fijo que está compuesto por una secuencia de caracteres o una constante numérica.

AWS Clean Rooms Spark SQL admite varios tipos de literales, entre ellos:

- Literales numéricos para enteros, decimales y números en coma flotante.
- Los literales de caracteres, también denominados cadenas, cadenas de caracteres o constantes de caracteres, se utilizan para especificar el valor de una cadena de caracteres.
- Literales de fecha, hora y marca temporal, utilizados como tipos datos de fecha y hora. Para obtener más información, consulte Literales de fecha, hora y marca temporal.
- Literales de intervalo. Para obtener más información, consulte Literales de intervalo.
- Literales booleanos. Para obtener más información, consulte Literales booleanos.
- Literales nulos que se utilizan para especificar un valor nulo.
- Solo TAB, CARRIAGE RETURN (CR), y LINE FEED (LF) Se admiten los caracteres de control Unicode de la categoría general de Unicode (Cc).

AWS Clean Rooms Spark SQL no admite referencias directas a cadenas literales en la cláusula SELECT, pero se pueden usar en funciones como CAST.

+ Operador (concatenación)

Concatena literales numéricos, literales de cadena y/o literales de fecha y hora e intervalo. Están a ambos lados del símbolo + y devuelven diferentes tipos en función de las entradas a cada lado del símbolo +.

Sintaxis

```
numeric + string
```

```
date + time
```

```
date + timetz
```

El orden de los argumentos se puede invertir.

Argumentos

numeric literals

Los literales o las constantes que representan números pueden ser enteros o números en coma flotante.

string literals

Cadenas, cadenas de caracteres o constantes de caracteres

date

A DATE columna o expresión que se convierte implícitamente en DATE.

time

A TIME columna o expresión que se convierte implícitamente en TIME.

timetz

A TIMETZ columna o expresión que se convierte implícitamente en TIMETZ.

Ejemplo

La siguiente tabla de ejemplo TIME_TEST tiene una columna TIME_VAL (tipo TIME) con tres valores insertados.

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

Tipos de datos

Cada valor que AWS Clean Rooms Spark SQL almacena o recupera tiene un tipo de datos con un conjunto fijo de propiedades asociadas. Los tipos de datos se declaran cuando se crean las tablas. Un tipo de datos limita el conjunto de valores que una columna o un argumento puede contener.

La siguiente tabla muestra los tipos de datos que puedes usar en AWS Clean Rooms Spark SQL.

Nombre del tipo de datos	Tipo de datos:	Alias	Description (Descripción)
ARRAY	the section called “Tipo anidado”	No aplicable	Tipo de datos anidados de matriz
BIGINT	the section called “Tipos numéricos”	No aplicable	Entero firmado de ocho bytes
BINARIO	the section called “Tipo binario”	No aplicable	Valores de secuencia de bytes
BOOLEANO	the section called “Tipo booleano”	BOOL	Booleano lógico (true/false)
BYTE	the section called “Tipos numéricos”	No aplicable	Números enteros con signo de 1 byte, de -128 a 127
CHAR	the section called “Tipos de caracteres”	CHARACTER	Cadena de caracteres de longitud fija

Nombre del tipo de datos	Tipo de datos:	Alias	Description (Descripción)
DATE	the section called “Tipos de fecha y hora”	No aplicable	Fecha de calendario (año, mes, día)
DECIMAL	the section called “Tipos numéricos”	NUMERIC	Numérico exacto de precisión seleccionable
FLOAT	the section called “Tipos numéricos”	FLOAT8, DOBLE PRECISIÓN	Número en coma flotante de precisión doble
INTEGER	the section called “Tipos numéricos”	INT	Entero firmado de cuatro bytes
INTERVAL	the section called “Tipos de fecha y hora”	No aplicable	Duración del tiempo en orden de día a día o de año a mes
LONG	the section called “Tipos numéricos”	No aplicable	Números enteros con signo de 8 bytes
MAP	the section called “Tipo anidado”	No aplicable	Tipo de datos anidados de mapa
REAL	the section called “Tipos numéricos”	FLOAT4	Número en coma flotante de precisión única
SHORT	the section called “Tipos numéricos”	No aplicable	Números enteros con signo de 2 bytes.
SMALLINT	the section called “Tipos numéricos”	No aplicable	Entero firmado de dos bytes

Nombre del tipo de datos	Tipo de datos:	Alias	Description (Descripción)
STRUCT	the section called “Tipo anidado”	No aplicable	Tipo de datos anidados de estructura
TIMESTAMP_LTZ	the section called “Tipos de fecha y hora”	No aplicable	Hora del día con zona horaria local
TIMESTAMP_NTZ	the section called “Tipos de fecha y hora”	No aplicable	Hora del día sin zona horaria
TINYINT	the section called “Tipos numéricos”	No aplicable	Números enteros con signo de 1 byte, de -128 a 127
VARCHAR	the section called “Tipos de caracteres”	CHARACTER VARYING	Cadena de caracteres de longitud variable con un límite definido por el usuario

> **Note**
>
> Los tipos de datos anidados ARRAY, STRUCT y MAP actualmente solo están habilitados para la regla de análisis personalizada. Para obtener más información, consulte Tipo anidado.

Caracteres multibyte

El tipo de datos VARCHAR es compatible con caracteres multibyte UTF-8 de hasta un máximo de cuatro bytes. Los caracteres de cinco bytes o más no son compatibles. Para calcular el tamaño de una columna VARCHAR que contiene caracteres multibyte, multiplique el número de caracteres por

el número de bytes por carácter. Por ejemplo, si una cadena tiene cuatro caracteres chinos y cada carácter tiene tres bytes, necesitará una columna VARCHAR(12) para almacenar la cadena.

El tipo de datos VARCHAR no es compatible con los siguientes valores de punto UTF-8 no válidos:

`0xD800 – 0xDFFF` (Secuencias de bytes: `ED A0 80` a `ED BF BF`)

El tipo de datos CHAR no es compatible con los caracteres multibyte.

Tipos numéricos

Los tipos de datos numéricos incluyen enteros, decimales y números en coma flotante.

Temas

- Tipos de enteros
- Tipo DECIMAL o NUMERIC
- Tipos de números en coma flotante
- Cómputos con valores numéricos

Tipos de enteros

Usa los siguientes tipos de datos para almacenar números enteros de varios rangos. No puede almacenar valores fuera del rango permitido para cada tipo.

Name	Almacenamiento	Range
SMALLINT	2 bytes	De -32768 a +32767
SHORT	2 bytes	De -32768 a +32767
INTEGER o INT	4 bytes	De -2147483648 a 2147483647
BIGINT	8 bytes	De -92233720 36854775808 a 922337203 6854775807

Name	Almacenamiento	Range
LONG	8 bytes	De -92233720 36854775808 a 922337203 6854775807

Tipo DECIMAL o NUMERIC

Use el tipo de datos DECIMAL o NUMERIC para almacenar valores con una precisión definida por el usuario. Las palabras clave DECIMAL y NUMERIC son intercambiables. En este documento, decimal es el término preferido para este tipo de datos. El término numérico se utiliza genéricamente para referirse a tipos de datos enteros, decimales y con coma flotante.

Almacenamiento	Range
Variable, hasta 128 bits para tipos DECIMAL sin comprimir.	Los enteros firmados de 128 bits con hasta 38 dígitos de precisión.

Defina una columna DECIMAL en una tabla especificando un *precision* y*scale*:

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

precision

El número total de dígitos significativos en todo el valor: la cantidad de dígitos de ambos lados del punto decimal. Por ejemplo, el número `48.2891` tiene una precisión de 6 y una escala de 4. La precisión predeterminada es 18, si no se especifica. La precisión máxima es 38.

Si el número de dígitos a la izquierda del punto decimal en un valor de entrada supera la precisión de la columna menos su escala, no se puede copiar (ni insertar ni actualizar) el valor en la columna. Esta regla se aplica a cualquier valor que caiga fuera del rango de la definición de la columna. Por ejemplo, el rango permitido de valores para una columna `numeric(5,2)` es de `-999.99` a `999.99`.

scale

El número de dígitos decimales en la parte fraccional del valor, a la derecha del punto decimal. Los enteros tienen una escala de cero. En la especificación de una columna, el valor de la escala debe ser inferior que o igual al valor de precisión. La escala predeterminada es 0, si no se especifica. La escala máxima es 37.

Si la escala de un valor de entrada que se carga en una tabla es mayor que la escala de la columna, el valor se redondea a la escala especificada. Por ejemplo, la columna PRICEPAID de la tabla SALES es una columna DECIMAL(8,2). Si se inserta un valor DECIMAL(8,4) en la columna PRICEPAID, el valor se redondea a una escala de 2.

```
insert into sales
values (0, 8, 1, 1, 2000, 14, 5, 4323.8951, 11.00, null);

select pricepaid, salesid from sales where salesid=0;

pricepaid | salesid
-----------+---------
4323.90 |       0
(1 row)
```

Sin embargo, no se redondean los resultados de formas explícitas de los valores seleccionados de tablas.

Note

El valor positivo máximo que puede insertar en una columna DECIMAL(19,0) es `9223372036854775807` (2^{63} -1). El valor negativo máximo es `-9223372036854775807`. Por ejemplo, un intento de insertar el valor `9999999999999999` (19 nueves) provocará un error de desbordamiento. Independientemente de la ubicación del punto decimal, la cadena de mayor tamaño que AWS Clean Rooms puede representar como un número DECIMAL es `9223372036854775807`. Por ejemplo, el valor más grande que puede cargar en una columna DECIMAL(19,18) es `9.223372036854775807`.
Estas reglas se deben a los motivos siguientes:

- Los valores DECIMAL con 19 dígitos de precisión significativos o menos se almacenan internamente como enteros de 8 bytes.

- Los valores DECIMAL con entre 20 y 38 dígitos de precisión significativos se almacenan como enteros de 16 bytes.

Notas acerca del uso de las columnas DECIMAL o NUMERIC de 128 bits

No asigne arbitrariamente la precisión máxima de las columnas DECIMAL a menos que esté seguro de que la aplicación requiere esa precisión. Los valores de 128 bits utilizan el doble de espacio en el disco en comparación de los valores de 64 bits y pueden alargar el tiempo de ejecución de la consulta.

Tipos de números en coma flotante

Use el tipo de datos REAL o DOUBLE PRECISION para almacenar valores numéricos con precisión variable. Estos tipos son inexactos, lo que significa que algunos valores se almacenan como aproximaciones, por lo que puede haber pequeñas discrepancias al almacenar y devolver un valor específico. Si requiere almacenamiento y cálculos exactos (como para importes monetarios), use el tipo de datos DECIMAL.

REAL representa el formato de coma flotante de precisión simple, según la norma IEEE 754 de aritmética de coma flotante. Tiene una precisión de unos 6 dígitos y un intervalo de 1E-37 a 1E+37 aproximadamente. También puede especificar este tipo de datos como. FLOAT4

DOUBLE PRECISION representa el formato de coma flotante de doble precisión, según la norma IEEE 754 para la aritmética binaria de coma flotante. Tiene una precisión de unos 15 dígitos y un intervalo de 1E-307 a 1E+308 aproximadamente. También puede especificar este tipo de datos como FLOAT o FLOAT8.

Cómputos con valores numéricos

En AWS Clean Rooms, la computación se refiere a las operaciones matemáticas binarias: suma, resta, multiplicación y división. En esta sección se describen los tipos devueltos previstos para estas operaciones, así como la fórmula específica que se aplica para determinar la precisión y la escala cuando hay tipos de datos DECIMAL involucrados.

Cuando se computan los valores numéricos durante el procesamiento de consultas, puede encontrar casos donde el cómputo no es posible y la consulta devuelve un error de desbordamiento numérico. También puede encontrar casos donde una escala de valores computados varía o es inesperada.

Para algunas operaciones, puede usar formas explícitas (tipo de promoción) o parámetros de configuración de AWS Clean Rooms para solucionar estos problemas.

Para obtener más información acerca de los resultados de cálculo similares con funciones SQL, consulte AWS Clean Rooms Funciones de Spark SQL.

Tipos devueltos para cómputos

Dado el conjunto de tipos de datos numéricos admitidos AWS Clean Rooms, la siguiente tabla muestra los tipos de rendimiento esperados para las operaciones de suma, resta, multiplicación y división. La primera columna del lado izquierdo de la tabla representa el primer operando del cálculo, y la fila superior representa el segundo operando.

Operando 1	Operando 2	Tipo de devolución
SMALLINT o SHORT	SMALLINT o SHORT	SMALLINT o SHORT
SMALLINT o SHORT	INTEGER	INTEGER
SMALLINT o SHORT	BIGINT	BIGINT
SMALLINT o SHORT	DECIMAL	DECIMAL
SMALLINT o SHORT	FLOAT4	FLOAT8
SMALLINT o SHORT	FLOAT8	FLOAT8
INTEGER	INTEGER	INTEGER
INTEGER	GRANDE o LARGO	BIGINT o LONG
INTEGER	DECIMAL	DECIMAL
INTEGER	FLOAT4	FLOAT8
INTEGER	FLOAT8	FLOAT8
BIGINT o LONG	BIGINT o LONG	BIGINT o LONG
BIGINT o LONG	DECIMAL	DECIMAL
BIGINT o LONG	FLOAT4	FLOAT8

Operando 1	Operando 2	Tipo de devolución
BIGINT o LONG	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL
DECIMAL	FLOAT4	FLOAT8
DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8
FLOAT8	FLOAT8	FLOAT8

Precisión y escala de resultados DECIMAL computados

En la siguiente tabla se resumen las reglas para computar la precisión y la escala resultantes cuando las operaciones matemáticas devuelven resultados DECIMAL. En esta tabla, `p1` `s1` represente la precisión y la escala del primer operando de un cálculo. `p2`y `s2` representan la precisión y la escala del segundo operando. (Independientemente de estos cálculos, la precisión de resultados máxima es 38 y la escala de resultados máxima es 38).

Operación	Precisión y escala del resultado
+ o bien -	Escalado = `max(s1,s2)` Precisión = `max(p1-s1,p2-s2)+1+scale`
*	Escalado = `s1+s2` Precisión = `p1+p2+1`
/	Escalado = `max(4,s1+p2-s2+1)` Precisión = `p1-s1+ s2+scale`

Por ejemplo, las columnas PRICEPAID y COMMISSION de la tabla SALES son columnas DECIMAL(8,2). Si divide PRICEPAID por COMMISSION (o viceversa), la fórmula se aplica de la siguiente manera:

```
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)
```

El siguiente cálculo es la regla general para computar la precisión y la escala resultantes para operaciones realizadas en valores DECIMAL con operadores como UNION, INTERSECT o EXCEPT, o funciones como COALESCE y DECODE:

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

Por ejemplo, una DEC1 tabla con una columna DECIMAL (7,2) se une a una DEC2 tabla con una columna DECIMAL (15,3) para crear una tabla. DEC3 El esquema de DEC3 muestra que la columna se convierte en una columna NUMÉRICA (15,3).

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

En el ejemplo anterior, la fórmula se aplica de la siguiente manera:

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

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

Notas sobre las operaciones de división

En las operaciones de división, divide-by-zero las condiciones devuelven errores.

El límite de escala de 100 se aplica después de que se calculan la precisión y la escala. Si la escala resultante calculada es superior a 100, los resultados de la división están escalados de la siguiente manera:

- Precisión = `precision - (scale - max_scale)`

- Escalado = `max_scale`

Si la precisión calculada es superior a la precisión máxima (38), la precisión se reduce a 38 y la escala se convierte en el resultado de: `max(38 + scale - precision), min(4, 100))`

Condiciones de desbordamiento

Se revisa el desbordamiento para todos los cómputos numéricos. Los datos DECIMAL con una precisión de 19 o menos se almacenan como enteros de 64 bits. Los datos DECIMAL con una precisión superior a 19 se almacenan como enteros de 128 bits. La precisión máxima para todos los valores DECIMAL es 38 y la escala máxima es 37. Los errores de desbordamiento ocurren cuando un valor supera estos límites, que se aplican en los conjuntos de resultados intermedios y finales:

- La conversión explícita provoca errores de desbordamiento del tiempo de ejecución cuando valores de datos específicos no se ajustan a la precisión o escala solicitadas especificadas por la función de conversión. Por ejemplo, no se puede transformar todos los valores de la columna PRICEPAID de la tabla SALES (una columna DECIMAL(8,2)) y devolver un resultado DECIMAL(7,3):

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

  Este error se produce porque algunos de los valores más grandes de la columna PRICEPAID no se pueden transformar.
- Las operaciones de multiplicación producen resultados en los que la escala de resultados es la suma de la escala de cada operando. Si ambos operandos tienen una escala de 4, por ejemplo, la escala resultante es 8, dejando solo 10 dígitos para el lado izquierdo del punto decimal. Por lo tanto, es relativamente fácil encontrarse con condiciones de desbordamiento cuando multiplica dos números grandes que tienen escalas significativas.

Cálculos numéricos con tipos INTEGER y DECIMAL

Cuando uno de los operandos de un cálculo tiene un tipo de datos INTEGER y el otro operando es DECIMAL, el operando INTEGER se forma implícitamente como DECIMAL.

- SMALLINT o SHORT se convierten en DECIMAL (5,0)
- INTEGER se forma como DECIMAL(10,0)

- BIGINT o LONG se convierte en DECIMAL (19,0)

Por ejemplo, si multiplica SALES.COMMISSION, una columna DECIMAL(8,2), y SALES.QTYSOLD, una columna SMALLINT, este cálculo se forma de la siguiente manera:

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

Tipos de caracteres

Los tipos de datos de caracteres incluyen CHAR (carácter) y VARCHAR (carácter variable).

Temas

- CHAR o CHARACTER
- VARCHAR o CHARACTER VARYING
- Importancia de los espacios en blancos anteriores y posteriores

CHAR o CHARACTER

Utilice una columna CHAR o CHARACTER para almacenar cadenas de longitud fija. Estas cadenas está rellenadas con espacios en blanco, por lo que una columna CHAR(10) siempre ocupa 10 bytes de almacenamiento.

```
char(10)
```

Una columna CHAR sin una especificación de longitud resulta en una columna CHAR(1).

Los tipos de datos CHAR y VARCHAR se definen en términos de bytes, no de caracteres. Una columna CHAR solo puede contener caracteres de un byte, por lo que una columna CHAR(10) puede contener una cadena con una longitud máxima de 10 bytes.

Name	Almacenamiento	Rango (ancho de columna)
CHAR o CHARACTER	Longitud de la cadena, incluidos espacios en	4 096 bytes

Name	Almacenamiento	Rango (ancho de columna)
	blanco anteriores o posteriores (si corresponde)	

VARCHAR o CHARACTER VARYING

Utilice una columna VARCHAR o VARYING CHARACTER para almacenar cadenas de longitud variable con un límite fijo. Estas cadenas no se rellenan con espacios en blancos, por lo que una columna VARCHAR(120) consta de un máximo de 120 caracteres de un byte, 60 caracteres de dos bytes, 40 caracteres de tres bytes o 30 caracteres de cuatro bytes.

```
varchar(120)
```

Los tipos de datos de VARCHAR se definen en términos de bytes, no de caracteres. Un VARCHAR puede contener caracteres multibyte de hasta un máximo de cuatro bytes por carácter. Por ejemplo, una columna VARCHAR(12) puede contener 12 caracteres de un byte, 6 caracteres de dos bytes, 4 caracteres de tres bytes o 3 caracteres de cuatro bytes.

Name	Almacenamiento	Rango (ancho de columna)
VARCHAR o CHARACTER VARYING	4 bytes + bytes totales por caracteres, donde cada carácter puede tener entre 1 y 4 bytes.	65 535 bytes (64K -1)

Importancia de los espacios en blancos anteriores y posteriores

Los tipos de datos CHAR y VARCHAR almacenan cadenas de hasta n bytes de longitud. Si se intenta almacenar una cadena más larga en una columna de estos tipos, se obtiene un error. Sin embargo, si los caracteres adicionales son todos espacios (en blanco), la cadena se trunca hasta alcanzar la longitud máxima. Si la cadena es más corta que la longitud máxima, los valores CHAR se

rellenan con espacios en blanco, pero los valores VARCHAR almacenan la cadena sin espacios en blanco.

Los espacios en blanco anteriores o posteriores en valores CHAR no tienen importancia semántica. Se omiten cuando compara dos valores CHAR, no se incluyen en cálculos LENGTH y se eliminan cuando convierte un valor CHAR a otro tipo de cadena.

Los espacios anteriores o posteriores en los valores VARCHAR y CHAR no tienen importancia semántica cuando se comparan valores.

Los cálculos de longitud devuelven la longitud de cadenas de caracteres VARCHAR con espacios anteriores o posteriores incluidos en la longitud. Los espacios anteriores o posteriores no cuentan en la longitud para cadenas de caracteres de longitud fija.

Tipos de fecha y hora

Los tipos de datos de fecha y hora incluyen DATE, TIME, TIMESTAMP_LTZ y TIMESTAMP_NTZ.

Temas

DATE

Utilice el tipo de datos DATE para almacenar fechas de calendario simples sin marcas temporales.

Name	Almacenamiento	Range	Resolución
DATE	4 bytes	De 4713 a.C. a 294276 d.C.	1 día

TIMESTAMP_LTZ

Usa el tipo de datos TIMESTAMP_LTZ para almacenar valores de marca de tiempo completos que incluyan la fecha, la hora del día y la zona horaria local.

TIMESTAMP representa valores compuestos por los valores de los camposyear,,, y month dayhour, minute con la zona horaria local de la sesión. second El timestamp valor representa un punto absoluto en el tiempo.

TIMESTAMP en Spark es un alias especificado por el usuario asociado a una de las variantes TIMESTAMP_LTZ y TIMESTAMP_NTZ. Puedes establecer el tipo de marca de tiempo predeterminado como TIMESTAMP_LTZ (valor predeterminado) o TIMESTAMP_NTZ a través de la configuración. spark.sql.timestampType

TIMESTAMP_NTZ

Utilice el tipo de datos TIMESTAMP_NTZ para almacenar valores de marca de tiempo completos que incluyan la fecha y la hora del día, sin incluir la zona horaria local.

TIMESTAMP representa valores compuestos por los valores de los campos,,, y. year month day hour minute second Todas las operaciones se realizan sin tener en cuenta ninguna zona horaria.

TIMESTAMP en Spark es un alias especificado por el usuario asociado a una de las variantes TIMESTAMP_LTZ y TIMESTAMP_NTZ. Puedes establecer el tipo de marca de tiempo predeterminado como TIMESTAMP_LTZ (valor predeterminado) o TIMESTAMP_NTZ a través de la configuración. spark.sql.timestampType

Ejemplos con tipos de fecha y hora

En los siguientes ejemplos se muestra cómo usar los tipos de fecha y hora que se admiten en AWS Clean Rooms.

Ejemplos de fecha

Los siguientes ejemplos insertan fechas que tienen diferentes formatos y muestran la salida.

```
select * from datetable order by 1;

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

Si inserta un valor de marca temporal en una columna DATE, se ignora la parte de la hora y solo se carga la fecha.

Ejemplos de tiempo

Los siguientes ejemplos insertan los valores TIME y TIMETZ que tienen diferentes formatos y muestran la salida.

```
select * from timetable order by 1;
start_time |  end_time
-----------------------
 19:11:19  | 20:41:19+00
 19:11:19  | 20:41:19+00
```

Literales de fecha, hora y marca temporal

Las siguientes son las reglas para trabajar con literales de fecha, hora y marca horaria compatibles con Spark SQL. AWS Clean Rooms

Fechas

La siguiente tabla muestra las fechas de entrada que son ejemplos válidos de valores de fecha literales que puedes cargar en tablas. AWS Clean Rooms Se supone que el modo predeterminado `MDY DateStyle` está en vigor. Este modo significa que el valor del mes precede al valor del día en las cadenas, como `1999-01-08` y `01/02/00`.

> **Note**
>
> Un literal de marca temporal o fecha debe encerrarse entre comillas cuando lo carga a la tabla.

Fecha de entrada	Fecha completa
8 de enero de 1999	8 de enero de 1999
1999-01-08	8 de enero de 1999
1/8/1999	8 de enero de 1999

Fecha de entrada	Fecha completa
01/02/00	2 de enero de 2000
2000-Ene-31	31 de enero de 2000
Ene-31-2000	31 de enero de 2000
31-Ene-2000	31 de enero de 2000
20080215	15 de febrero de 2008
080215	15 de febrero de 2008
2008.366	31 de diciembre de 2008 (la parte de tres dígitos de la fecha debe tener un valor del rango 001-366).

Times

En la siguiente tabla se muestran las horas de entrada que son ejemplos válidos de valores de hora literales que se pueden cargar en AWS Clean Rooms las tablas.

Horas de entrada	Descripción (de la parte de la hora)
04:05:06789	4:05 a. m. y 6789 segundos
04:05:06	4:05 a. m. y 6 segundos
04:05	4:05 a. m. exactamente
04-0506	4:05 a. m. y 6 segundos
04:05 a. m.	4:05 a. m. exactamente; a. m. es opcional
04:05. p. m.	4:05 p. m. exactamente; el valor de la hora debe ser menor que 12
16:05	4:05 p. m. exactamente

Valores de fecha y hora especiales

La siguiente tabla muestra valores especiales que se pueden usar como literales de fecha y hora y como argumentos para funciones de fecha. Requieren comillas simples y se convierten en valores de marca temporal regulares durante el procesamiento de consultas.

Valor especial	Description (Descripción)
now	Evalúa la hora de inicio de la transacción actual y devuelve una marca temporal con precisión de microsegundo.
today	Toma el valor de la fecha adecuada y devuelve una marca temporal con ceros en las partes de la hora.
tomorrow	Toma el valor de la fecha adecuada y devuelve una marca temporal con ceros en las partes de la hora.
yesterday	Toma el valor de la fecha adecuada y devuelve una marca temporal con ceros en las partes de la hora.

Los siguientes ejemplos muestran cómo today funciona now la función DATE_ADD.

```
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)
```

Literales de intervalo

A continuación, se muestran las reglas para trabajar con literales de intervalo compatibles con AWS Clean Rooms Spark SQL.

Use un literal de intervalo para identificar períodos específicos de tiempo, como `12 hours` o `6 weeks`. Puede usar estos literales de intervalo en condiciones y cálculos que involucran expresiones de fecha y hora.

> **Note**
>
> No puedes usar el tipo de datos INTERVAL para las columnas de las AWS Clean Rooms tablas.

Un intervalo se expresa como una combinación de la palabra clave INTERVAL con una cantidad numérica y una parte de fecha compatible, por ejemplo, `INTERVAL '7 days'` o `INTERVAL '59 minutes'`. Puede conectar varias cantidades y unidades para formar un intervalo más preciso, por ejemplo: `INTERVAL '7 days, 3 hours, 59 minutes'`. También se admiten abreviaturas y plurales de cada unidad; por ejemplo: `5 s`, `5 second` y `5 seconds` son intervalos equivalentes.

Si no especifica una parte de fecha, el valor de intervalo representa segundos. Puede especificar el valor de cantidad como una fracción (por ejemplo: `0.5 days`).

Ejemplos

En los siguientes ejemplos se muestra una serie de cálculos con diferentes valores de intervalo.

En el siguiente ejemplo se agrega 1 segundo a la fecha especificada.

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

En el siguiente ejemplo se agrega 1 minuto a la fecha especificada.

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

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

En el siguiente ejemplo se agregan 3 horas y 35 minutos a la fecha especificada.

```
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)
```

En el siguiente ejemplo se agregan 52 semanas a la fecha especificada.

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

En el siguiente ejemplo se agrega 1 semana, 1 hora, 1 minuto y 1 segundo a la fecha especificada.

```
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)
```

En el siguiente ejemplo se agregan 12 horas (medio día) a la fecha especificada.

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

En el siguiente ejemplo se restan 4 meses desde el 31 de marzo de 2023 y el resultado es el 30 de noviembre de 2022. El cálculo tiene en cuenta el número de días de un mes.

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

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

Literales y tipos de datos de intervalo

Puede usar un tipo de datos de intervalo para almacenar duraciones de tiempo en unidades como `seconds`, `minutes`, `hours`, `days`, `months` y `years`. Los literales y los tipos de datos de intervalo se pueden usar en los cálculos de fecha y hora, por ejemplo, agregar intervalos a fechas y marcas temporales, sumar intervalos y restar un intervalo de una fecha o marca temporal. Los literales de intervalo se pueden usar como valores de entrada para las columnas de tipos de datos de intervalos de una tabla.

Sintaxis del tipo de datos de intervalo

Para especificar un tipo de datos de intervalo para almacenar una duración de tiempo en años y meses:

```
INTERVAL year_to_month_qualifier
```

Para especificar un tipo de datos de intervalo para almacenar una duración en días, horas, minutos y segundos:

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

Sintaxis de literal de intervalo

Para especificar un literal de intervalo para definir una duración de tiempo en años y meses:

```
INTERVAL quoted-string year_to_month_qualifier
```

Para especificar un literal de intervalo para definir una duración en días, horas, minutos y segundos:

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

Argumentos

quoted-string

Especifica un valor numérico positivo o negativo especificando una cantidad y la unidad de fecha y hora como cadena de entrada. Si la cadena entre comillas contiene solo un número, AWS Clean Rooms determina las unidades del calificador year_to_month_qualifier o day_to_second_qualifier. Por ejemplo, `'23' MONTH` representa `1 year 11 months`, `'-2' DAY` representa `-2 days 0 hours 0 minutes 0.0 seconds`, `'1-2' MONTH` representa `1 year 2 months` y `'13 day 1 hour 1 minute 1.123 seconds' SECOND` representa `13 days 1 hour 1 minute 1.123 seconds`. Para obtener más información acerca de los formatos de salida de un intervalo, consulte Estilos de intervalo.

year_to_month_qualifier

Especifica el rango del intervalo. Si usa un calificador y crea un intervalo con unidades de tiempo más pequeñas que el calificador, trunca y descarta las partes más pequeñas del intervalo. AWS Clean Rooms Los valores válidos para year_to_month_qualifier son:

- `YEAR`
- `MONTH`
- `YEAR TO MONTH`

day_to_second_qualifier

Especifica el rango del intervalo. Si usa un calificador y crea un intervalo con unidades de tiempo más pequeñas que el calificador, AWS Clean Rooms trunca y descarta las partes más pequeñas del intervalo. Los valores válidos para day_to_second_qualifier son:

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

El resultado del literal INTERVAL se trunca al componente INTERVAL más pequeño especificado. Por ejemplo, al utilizar un calificador MINUTE, AWS Clean Rooms descarta las unidades de tiempo inferiores a MINUTE.

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

El valor resultante se trunca en `'1 day 01:01:00'`.

fractional_precision

Parámetro opcional que especifica el número de dígitos fraccionales permitidos en el intervalo. El argumento fractional_precision solo se debe especificar si el intervalo contiene SECOND. Por ejemplo, `SECOND(3)` crea un intervalo que permite solo tres dígitos fraccionales, como 1234 segundos. El número máximo de dígitos fraccionales es seis.

La configuración de la sesión `interval_forbid_composite_literals` determina si se devuelve un error cuando se especifica un intervalo con las partes YEAR TO MONTH y DAY TO SECOND.

Aritmética de intervalos

Puede utilizar valores de intervalo con otros valores de fecha y hora para realizar operaciones aritméticas. En las siguientes tablas se describen las operaciones disponibles y los resultados de tipo de datos de cada operación.

> **Note**
>
> Las operaciones que pueden producir resultados `date` y `timestamp` lo hacen en función de la unidad de tiempo más pequeña implicada en la ecuación. Por ejemplo, cuando se agrega un `interval` a una `date` el resultado es una `date` si es un intervalo YEAR TO MONTH y una marca temporal si es un intervalo DAY TO SECOND.

Las operaciones en las que el primer operando es un `interval` producen los siguientes resultados para el segundo operando dado:

Operador	Date	Timestamp	Interval	Numérico
-	N/A	N/A	Interval	N/A

Operador	Date	Timestamp	Interval	Numérico
+	Date	Date/Timestamp	Interval	N/A
*	N/A	N/A	N/A	Interval
/	N/A	N/A	N/A	Interval

Las operaciones en las que el primer operando es una `date` producen los siguientes resultados para el segundo operando dado:

Operador	Date	Timestamp	Interval	Numérico
-	Numérico	Interval	Date/Timestamp	Date
+	N/A	N/A	N/A	N/A

Las operaciones en las que el primer operando es una `timestamp` producen los siguientes resultados para el segundo operando dado:

Operador	Date	Timestamp	Interval	Numérico
-	Numérico	Interval	Timestamp	Timestamp
+	N/A	N/A	N/A	N/A

Estilos de intervalo

- `postgres`: sigue el estilo de PostgreSQL. Es el valor predeterminado.
- `postgres_verbose`: sigue el estilo detallado de PostgreSQL.
- `sql_standard`: sigue el estilo de literales de intervalo estándar de SQL.

El siguiente comando establece el estilo de intervalo en `sql_standard`.

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

Formato de salida postgres

A continuación, se muestra el formato de salida del estilo de intervalo `postgres`. Cada valor numérico puede ser negativo.

```
'<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
```

Formato de salida postgres_verbose

La sintaxis de postgres_verbose es similar a la de postgres, pero las salidas de postgres_verbose también contienen la unidad de tiempo.

```
'[@] <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
```

Formato de salida sql_standard

Los valores del intervalo de año a mes tienen el siguiente formato. Si se especifica un signo negativo antes del intervalo, eso indica que el intervalo es un valor negativo y se aplica a todo el intervalo.

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

Los valores del intervalo de día a segundo tienen el siguiente formato.

```
'[-]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
```

Ejemplos de tipo de datos de intervalo

En los siguientes ejemplos, se muestra cómo usar tipos de datos INTERVAL con tablas.

```
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
-----+-----
```

Ejemplos de literales de intervalo

Los siguientes ejemplos se ejecutan con el estilo de intervalo establecido en `postgres`.

En el siguiente ejemplo, se muestra cómo crear un literal INTERVAL de 1 año.

```
select INTERVAL '1' YEAR

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

Si especifica una quoted-string que supere el calificador, las unidades de tiempo restantes se truncan con respecto al intervalo. En el ejemplo siguiente, un intervalo de 13 meses se convierte en 1 año y 1 mes, pero el mes restante se omite debido al calificador YEAR.

```
select INTERVAL '13 months' YEAR

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

Si utiliza un calificador inferior a la cadena de intervalos, se incluyen las unidades sobrantes.

```
select INTERVAL '13 months' MONTH

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

Al especificar una precisión en el intervalo, se trunca el número de dígitos fraccionarios hasta alcanzar la precisión especificada.

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

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

Si no especifica una precisión, AWS Clean Rooms utiliza la precisión máxima de 6.

```
select INTERVAL '1.23456789' SECOND

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

En el siguiente ejemplo, se muestra cómo crear un intervalo con rangos.

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

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

Los calificadores dictan las unidades que se especifican. Por ejemplo, aunque en el ejemplo siguiente se utiliza la misma cadena entrecomillada de «2:2» que en el ejemplo anterior, se AWS Clean Rooms reconoce que se utilizan unidades de tiempo diferentes debido al calificador.

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

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

También se admiten las abreviaturas y los plurales de cada unidad. Por ejemplo, `5s`, `5 second` y `5 seconds` son intervalos equivalentes. Las unidades admitidas son años, meses, horas, minutos y segundos.

```
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
```

Ejemplos de literales de intervalo sin sintaxis de calificador

> **Note**
>
> En los siguientes ejemplos se muestra el uso de un literal de intervalo sin un calificador `YEAR TO MONTH` o `DAY TO SECOND`. Para obtener información sobre el uso del literal de intervalo recomendado con un calificador, consulte Literales y tipos de datos de intervalo.

Use un literal de intervalo para identificar períodos específicos de tiempo, como `12 hours` o `6 months`. Puede usar estos literales de intervalo en condiciones y cálculos que involucran expresiones de fecha y hora.

Un literal de intervalo se expresa como una combinación de la palabra clave INTERVAL con una cantidad numérica y una parte de fecha compatible, por ejemplo, `INTERVAL '7 days'` o `INTERVAL '59 minutes'`. Puede conectar varias cantidades y unidades para formar un intervalo más preciso, por ejemplo: `INTERVAL '7 days, 3 hours, 59 minutes'`. También se admiten abreviaturas y plurales de cada unidad; por ejemplo: `5 s`, `5 second` y `5 seconds` son intervalos equivalentes.

Si no especifica una parte de fecha, el valor de intervalo representa segundos. Puede especificar el valor de cantidad como una fracción (por ejemplo: `0.5 days`).

En los siguientes ejemplos se muestra una serie de cálculos con diferentes valores de intervalo.

A continuación, se agrega 1 segundo a la fecha especificada.

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

A continuación, se agrega 1 minuto a la fecha especificada.

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

A continuación, se agregan 3 horas y 35 minutos a la fecha especificada.

```
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)
```

A continuación, se agregan 52 semanas a la fecha especificada.

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

A continuación, se agregan 1 semana, 1 hora, 1 minuto y 1 segundo a la fecha especificada.

```
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)
```

A continuación, se agregan 12 horas (medio día) a la fecha especificada.

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

Lo siguiente resta 4 meses al 15 de febrero de 2023 y el resultado es 15 de octubre de 2022.

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

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

Lo siguiente resta 4 meses al 31 de marzo de 2023 y el resultado es 30 de noviembre de 2022. El cálculo tiene en cuenta el número de días de un mes.

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

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

Tipo booleano

Use el tipo de dato BOOLEAN para almacenar valores verdaderos y falsos en una columna de un byte. En la siguiente tabla se describen los tres estados posibles para un valor booleano y los valores literales que generan ese estado. Independientemente de la cadena de entrada, una columna booleana almacena y produce "t" para verdadero y "f" para falso.

Estado	Valores literales válidos	Almacenamiento
True	`TRUE 't' 'true' 'y' 'yes' '1'`	1 byte
False	`FALSE 'f' 'false' 'n' 'no' '0'`	1 byte
Unknown	`NULL`	1 byte

Puede usar una comparación IS para comprobar un valor booleano solo como un predicado en la cláusula WHERE. No puede usar la comparación IS con un valor booleano en la lista SELECT.

Ejemplos

Puede usar una columna BOOLEAN para almacenar un estado "Activo/Inactivo" para cada cliente de una tabla CUSTOMER.

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

En este ejemplo, la siguiente consulta selecciona usuarios de la tabla USERS a los que les gustan los deportes, pero no el teatro:

```
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)
```

El siguiente ejemplo selecciona usuarios de la tabla USERS para los que se desconoce si les gusta el rock.

```
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)
```

El siguiente ejemplo devuelve un error porque usa una comparación IS en la lista SELECT.

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

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

El siguiente ejemplo es correcto porque usa una comparación igual (=) en la lista SELECT en lugar de la comparación IS.

```
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    |
```

Literales booleanos

Las siguientes reglas sirven para trabajar con literales booleanos compatibles con Spark SQL. AWS Clean Rooms

Usa un literal booleano para especificar un valor booleano, como o. `TRUE` `FALSE`

Sintaxis

```
TRUE | FALSE
```

Ejemplo

El siguiente ejemplo muestra una columna con un valor especificado de. `TRUE`

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

Tipo binario

Usa el tipo de datos BINARIO para almacenar y administrar datos binarios de longitud fija y sin interpretar, lo que proporciona capacidades eficientes de almacenamiento y comparación para casos de uso específicos.

El tipo de datos BINARIO almacena un número fijo de bytes, independientemente de la longitud real de los datos que se almacenan. La longitud máxima suele ser de 255 bytes.

BINARY se utiliza para almacenar datos binarios sin procesar y sin interpretar, como imágenes, documentos u otros tipos de archivos. Los datos se almacenan exactamente como se proporcionan, sin codificación ni interpretación de caracteres. Los datos binarios almacenados en las columnas BINARIAS se comparan y ordenan byte-by-byte en función de los valores binarios reales y no de ninguna regla de codificación o cotejo de caracteres.

En la siguiente consulta de ejemplo, se muestra la representación binaria de la cadena`"abc"`. Cada carácter de la cadena se representa mediante su código ASCII en formato hexadecimal: «a» es 0x61, «b» es 0x62 y «c» es 0x63. Cuando se combinan, estos valores hexadecimales forman la representación binaria. `"616263"`

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

Tipo anidado

AWS Clean Rooms admite consultas que incluyan datos con tipos de datos anidados, específicamente los tipos de columnas AWS Glue STRUCT, ARRAY y MAP. Solo la regla de análisis personalizada admite tipos de datos anidados.

En particular, los tipos de datos anidados no se ajustan a la estructura tabular estricta del modelo de datos relacionales de las bases de datos SQL.

Los tipos de datos anidados contienen etiquetas que hacen referencia a entidades diferenciadas dentro de los datos. Pueden contener valores complejos, como matrices, estructuras anidadas y otras estructuras complejas, que están asociadas a formatos de serialización, como JSON. Los tipos de datos anidados admiten hasta 1 MB de datos anidados por campo u objeto con tipo de datos anidados.

Temas

Tipo de matriz

Usa el tipo ARRAY para representar valores compuestos por una secuencia de elementos con el tipo de`elementType`.

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

Se utiliza `containsNull` para indicar si los elementos de un tipo ARRAY pueden tener `null` valores.

Tipo de mapa

Usa el tipo MAP para representar valores que comprenden un conjunto de pares clave-valor.

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

`keyType`: el tipo de datos de las claves

`valueType`: el tipo de datos de los valores

No se permite que las claves tengan `null` valores. Se utiliza `valueContainsNull` para indicar si los valores de un valor de tipo MAP pueden tener `null` valores.

Tipo de estructura

Usa el tipo STRUCT para representar valores con la estructura descrita por una secuencia de StructFields (campos).

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

StructField(nombre, tipo de datos, anulable): representa un campo en un. StructType

`dataType`: el tipo de datos: un campo

`name`: el nombre de un campo

Se utiliza `nullable` para indicar si los valores de estos campos pueden tener `null` valores.

Ejemplos de tipos de datos anidados

Para el tipo `struct<given:varchar, family:varchar>`, existen dos nombres de atributo: `given` y `family`, cada uno de los cuales corresponde a un valor `varchar`.

Para el tipo `array<varchar>`, la matriz se especifica como una lista de `varchar`.

El tipo `array<struct<shipdate:timestamp, price:double>>` hace referencia a una lista de elementos con el tipo `struct<shipdate:timestamp, price:double>`.

El tipo de datos `map` se comporta como una `array` de `structs`, donde el nombre del atributo de cada elemento de la matriz se indica con `key` y se asigna a un `value`.

Example

Por ejemplo, el tipo `map<varchar(20), varchar(20)>` se trata como`array<struct<key:varchar(20), value:varchar(20)>>`, dónde `key` y `value` hacen referencia a los atributos del mapa en los datos subyacentes.

Para obtener información sobre cómo se AWS Clean Rooms habilita la navegación en matrices y estructuras, consulte. Navegación

Para obtener información sobre cómo se AWS Clean Rooms habilita la iteración sobre matrices navegando por la matriz mediante la cláusula FROM de una consulta, consulte. Desanidar consultas

Conversión y compatibilidad de tipos

En los siguientes temas se describe cómo funcionan las reglas de conversión de tipos y la compatibilidad de tipos de datos en AWS Clean Rooms Spark SQL.

Temas

- Compatibilidad
- Reglas generales de conversión y compatibilidad
- Tipos de conversiones implícitas

Compatibilidad

La vinculación de tipos de datos y la vinculación de valores literales y constantes con tipos de datos ocurren durante varias operaciones de la base de datos, incluidas las siguientes:

- Operaciones de Data Manipulation Language (DML, Lenguaje de manipulación de datos) en tablas
- Consultas UNION, INTERSECT y EXCEPT
- Expresiones CASE
- Evaluación de predicados, como LIKE e IN
- La evaluación de funciones SQL que realizan comparaciones o extracciones de datos.
- Comparaciones con operadores matemáticos

Los resultados de estas operaciones dependen de las reglas de conversión de tipos y la compatibilidad de tipos de datos. La compatibilidad implica que no siempre es necesaria la one-to-one coincidencia de un valor determinado con un tipo de datos determinado. Dado que algunos tipos de datos son compatible, es posible una conversión implícita o coerción. Para obtener más información, consulte Tipos de conversiones implícitas. Cuando los tipos de datos no son compatibles, a menudo puede convertir un valor de un tipo de datos a otro al utilizar la función de conversión explícita.

Reglas generales de conversión y compatibilidad

Tenga en cuenta las siguientes reglas de conversión y compatibilidad:

- En general, los tipos de datos que caen en la misma categoría (como diferentes tipos de datos numéricos) son compatibles y se pueden convertir implícitamente.

  Por ejemplo, con la conversión implícita puede insertar un valor decimal en una columna de enteros. El decimal se redondea para producir un número entero. O bien, puede extraer un valor numérico, como `2008`, de una fecha e insertar ese valor en una columna de enteros.
- Los tipos de datos numéricos imponen condiciones de desbordamiento que se producen cuando se intenta insertar out-of-range valores. Por ejemplo, un valor decimal con una precisión de 5 no encaja en una columna decimal que se definió con una precisión de 4. Un entero o toda la parte de un decimal nunca se truncan. Sin embargo, la parte fraccionaria de un decimal se puede redondear hacia arriba o hacia abajo, según corresponda. Sin embargo, no se redondean los resultados de formas explícitas de los valores seleccionados de tablas.

- Los distintos tipos de cadenas de caracteres son compatibles. Las cadenas de la columna VARCHAR que contienen datos de un byte y las cadenas de la columna CHAR se pueden comparar y son convertibles de manera implícita. No se pueden comparar las cadenas VARCHAR que contienen datos multibyte. También puede convertir una cadena de caracteres a una fecha, una hora, una marca temporal o un valor numérico si la cadena es un valor literal adecuado. Se omiten los espacios anteriores o posteriores. En cambio, puede convertir una fecha, una hora, una marca temporal o un valor numérico a una cadena de caracteres de longitud fija o variable.

  > **Note**
  >
  > Una cadena de caracteres que desea transformar a un tipo numérico debe contener una representación de carácter de un número. Por ejemplo, puede transformar las cadenas `'1.0'` o `'5.9'` a valores decimales, pero no puede transformar la cadena `'ABC'` a ningún tipo numérico.

- Si compara valores DECIMALES con cadenas de caracteres, AWS Clean Rooms intenta convertir la cadena de caracteres en un valor DECIMAL. Al comparar todos los demás valores numéricos con cadenas de caracteres, los valores numéricos se convierten en cadenas de caracteres. Para aplicar la conversión opuesta (por ejemplo, convertir cadenas de caracteres en números enteros o convertir valores de tipo DECIMAL en cadenas de caracteres), utilice una función explícita, como Función CAST.
- Para convertir valores DECIMAL o NUMERIC de 64 bits a una precisión más grande, debe usar una función de conversión explícita, como las funciones CAST o CONVERT.

Tipos de conversiones implícitas

Existen dos tipos de conversiones implícitas:

- Conversiones implícitas en asignaciones, como establecer valores en comandos INSERT o UPDATE
- Conversiones implícitas en expresiones, como realizar comparaciones en la cláusula WHERE

En la siguiente tabla se enumeran los tipos de datos que pueden convertirse implícitamente en asignaciones o expresiones. También puede usar una función de conversión explícita para realizar estas conversiones.

Del tipo	Al tipo
BIGINT	BOOLEANO
	CHAR
	DECIMAL (NUMERIC)
	PRECISIÓN DOBLE (FLOAT8)
	INTEGER
	REAL (FLOAT4)
	SMALLINT o SHORT
	VARCHAR
CHAR	VARCHAR
DATE	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	GRANDE o LARGO
	CHAR
	DOBLE PRECISIÓN () FLOAT8
	INTEGER (INT)
	REAL (FLOAT4)
	SMALLINT o SHORT
	VARCHAR

Del tipo	Al tipo
DOBLE PRECISIÓN () FLOAT8	BIGINT o LONG
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	REAL () FLOAT4
	SMALLINT o SHORT
	VARCHAR
INTEGER (INT)	GRANDE o LARGO
	BOOLEANO
	CHAR
	DECIMAL (NUMERIC)
	DOBLE PRECISIÓN () FLOAT8
	REAL (FLOAT4)
	SMALLINT o SHORT
	VARCHAR
REAL () FLOAT4	BIGINT o LONG
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	MINÚSCULA o CORTA

Del tipo	Al tipo
	VARCHAR
SMALLINT	GRANDE o LARGO
	BOOLEANO
	CHAR
	DECIMAL (NUMERIC)
	DOBLE PRECISIÓN () FLOAT8
	INTEGER (INT)
	REAL (FLOAT4)
	VARCHAR
TIME	VARCHAR
	TIMETZ

> **Note**
>
> Las conversiones implícitas entre DATE, TIME, TIMESTAMP_LTZ, TIMESTAMP_NTZ o cadenas de caracteres utilizan la zona horaria de la sesión actual.
> El tipo de datos VARBYTE no se puede convertir de forma implícita en otros tipos de datos. Para obtener más información, consulte Función CAST.

AWS Clean Rooms Comandos SQL de Spark

Los siguientes comandos SQL son compatibles con AWS Clean Rooms Spark SQL:

Temas

- TABLA DE CACHÉ
- Sugerencias

- SELECT

TABLA DE CACHÉ

El comando CACHE TABLE almacena en caché los datos de una tabla existente o crea y almacena en caché una nueva tabla que contiene los resultados de la consulta.

> **Note**
>
> Los datos en caché se conservan durante toda la consulta.

La sintaxis, los argumentos y algunos ejemplos provienen de la referencia SQL de Apache Spark.

Sintaxis

El comando CACHE TABLE admite tres patrones de sintaxis:

Con AS (sin paréntesis): crea y almacena en caché una nueva tabla en función de los resultados de la consulta.

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

Con AS y paréntesis: funciona de forma similar a la primera sintaxis, pero utiliza paréntesis para agrupar la consulta de forma explícita.

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

Sin AS: almacena en caché una tabla existente mediante la instrucción SELECT para filtrar las filas que se van a almacenar en caché.

```
CACHE TABLE cache_table_identifier query;
```

Donde:

- Todas las sentencias deben terminar con punto y coma (;)
- `query`suele ser una sentencia SELECT

- Los paréntesis alrededor de la consulta son opcionales con AS
- La palabra clave AS es opcional

Parameters

cache_table_identifier

El nombre de la tabla en caché. Puede incluir un calificador de nombre de base de datos opcional.

AS

Palabra clave que se utiliza al crear y almacenar en caché una tabla nueva a partir de los resultados de una consulta.

consulta

Una instrucción SELECT u otra consulta que defina los datos que se van a almacenar en caché.

Ejemplos

En los ejemplos siguientes, la tabla en caché se conserva durante toda la consulta. Tras el almacenamiento en caché, las consultas posteriores a las que *cache_table_identifier* se haga referencia se leerán desde la versión en caché en lugar de volver a calcularse o leer desde ella. *sourceTable* Esto puede mejorar el rendimiento de las consultas para los datos a los que se accede con frecuencia.

Cree y almacene en caché una tabla filtrada a partir de los resultados de la consulta

El primer ejemplo muestra cómo crear y almacenar en caché una tabla nueva a partir de los resultados de una consulta. Este comando usa la AS palabra clave sin paréntesis alrededor de la SELECT sentencia. Crea una nueva tabla llamada 'cache_table_identifier' que contiene solo las filas de 'sourceTable' donde el estado es '. active' Ejecuta la consulta, almacena los resultados en la nueva tabla y guarda en caché el contenido de la nueva tabla. El 'sourceTable' original permanece sin cambios y las consultas posteriores deben hacer referencia a 'cache_table_identifier' para usar los datos en caché.

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

Almacene en caché los resultados de las consultas con sentencias SELECT entre paréntesis

El segundo ejemplo muestra cómo almacenar en caché los resultados de una consulta como una tabla nueva con un nombre específico (`cache_table_identifier`), utilizando paréntesis alrededor de la sentencia. `SELECT` Este comando crea una nueva tabla llamada '`cache_table_identifier`' que contiene solo las filas de '`sourceTable`' donde el estado es '. `active`' Ejecuta la consulta, almacena los resultados en la nueva tabla y guarda en caché el contenido de la nueva tabla. El '`sourceTable`' original permanece inalterado. Las consultas posteriores deben hacer referencia a `cache_table_identifier` «» para utilizar los datos en caché.

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

Almacene en caché una tabla existente con las condiciones del filtro

El tercer ejemplo muestra cómo almacenar en caché una tabla existente con una sintaxis diferente. Esta sintaxis, que omite la palabra clave `AS` 'y los paréntesis, normalmente almacena en caché las filas especificadas de una tabla existente denominada' `cache_table_identifier` 'en lugar de crear una tabla nueva. La `SELECT` sentencia actúa como un filtro para determinar qué filas se van a almacenar en caché.

> **Note**
>
> El comportamiento exacto de esta sintaxis varía según los sistemas de bases de datos. Compruebe siempre la sintaxis correcta para su AWS servicio específico.

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

Sugerencias

Las sugerencias para los análisis de SQL proporcionan directrices de optimización que guían las estrategias de ejecución de consultas AWS Clean Rooms, lo que te permite mejorar el rendimiento

de las consultas y reducir los costes de procesamiento. Las sugerencias sugieren cómo el motor de análisis de Spark debe generar su plan de ejecución.

Sintaxis

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

Las sugerencias se incluyen en las consultas SQL mediante una sintaxis similar a la de un comentario y deben colocarse directamente después de la palabra clave SELECT.

Tipos de sugerencias compatibles

AWS Clean Rooms admite dos categorías de sugerencias: sugerencias de unión y sugerencias de partición.

Temas

- Únase a las sugerencias
- Sugerencias de particionamiento

Únase a las sugerencias

Los consejos de unión sugieren estrategias de unión para la ejecución de consultas. La sintaxis, los argumentos y algunos ejemplos provienen de la referencia SQL de Apache Spark para obtener más información

EMISIÓN

Sugiere AWS Clean Rooms utilizar broadcast join. La parte de unión con la sugerencia se emitirá independientemente del autoBroadcastJoin umbral. Si ambos lados de la unión tienen las sugerencias emitidas, se emitirá la que tenga el tamaño más pequeño (según las estadísticas).

Alias: BROADCASTJOIN, MAPJOIN

Parámetros: identificadores de tabla (opcionales)

Ejemplos:

```
-- 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

Sugiere que se AWS Clean Rooms utilice la combinación, la ordenación, la combinación y la combinación.

Alias: SHUFFLE_MERGE, MERGEJOIN

Parámetros: identificadores de tabla (opcionales)

Ejemplos:

```
-- 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

Sugiere AWS Clean Rooms usar shuffle hash join. Si ambos lados tienen las sugerencias de mezcla aleatoria, el optimizador de consultas elige el lado más pequeño (según las estadísticas) como el lado de construcción.

Parámetros: identificadores de tabla (opcionales)

Ejemplos:

```
-- Use shuffle hash join
SELECT /*+ SHUFFLE_HASH(students) */ *
```

```
FROM employees e JOIN students s ON e.id = s.id;
```

SHUFFLE_REPLICATE_NL

Sugiere utilizar una unión de bucles anidada. AWS Clean Rooms shuffle-and-replicate

Parámetros: identificadores de tabla (opcionales)

Ejemplos:

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

Consejos para la solución de problemas en Spark SQL

La siguiente tabla muestra situaciones comunes en las que no se aplican sugerencias en SparkSQL. Para obtener información adicional, consulta the section called “Consideraciones y limitaciones”.

Caso de uso	Consulta de ejemplo
No se encontró la referencia de la tabla	`SELECT /*+ BROADCAST(fake_table) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
La tabla no participa en la operación de unión	`SELECT /*+ BROADCAST(s) */ * FROM students s WHERE s.age > 25;`
Referencia de tabla en una subconsulta anidada	`SELECT /*+ BROADCAST(s) */ * FROM employees e INNER JOIN (SELECT * FROM students s WHERE s.age > 20) sub ON e.eid = sub.sid;`
Nombre de columna en lugar de referencia de tabla	`SELECT /*+ BROADCAST(e.eid) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`

Caso de uso	Consulta de ejemplo
Sugerencia sin los parámetros necesarios	`SELECT /*+ BROADCAST */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Nombre de la tabla base en lugar del alias de la tabla	`SELECT /*+ BROADCAST(employees) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`

Sugerencias de particionamiento

Las sugerencias de particionamiento controlan la distribución de datos entre los nodos ejecutores. Cuando se especifican varias sugerencias de partición, se insertan varios nodos en el plan lógico, pero el optimizador selecciona la sugerencia situada más a la izquierda.

COALESCE

Reduce el número de particiones al número de particiones especificado.

Parámetros: valor numérico (obligatorio): debe ser un número entero positivo comprendido entre 1 y 2147483647

Ejemplos:

```
-- Reduce to 5 partitions
SELECT /*+ COALESCE(5) */ employee_id, salary
FROM employees;
```

REPARTICIÓN

Redivide los datos en el número especificado de particiones mediante las expresiones de partición especificadas. Utiliza una distribución por turnos.

Parámetros:

- Valor numérico (opcional): número de particiones; debe ser un entero positivo entre 1 y 2147483647

- Identificadores de columna (opcionales): columnas por las que realizar la partición; estas columnas deben existir en el esquema de entrada.
- Si se especifican ambos, el valor numérico debe ser lo primero

Ejemplos:

```
-- 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;
```

REPARTICIÓN_POR_RANGO

Redivide los datos en el número especificado de particiones mediante la partición por rangos en las columnas especificadas.

Parámetros:

- Valor numérico (opcional): número de particiones; debe ser un entero positivo entre 1 y 2147483647
- Identificadores de columna (opcionales): columnas por las que realizar la partición; estas columnas deben existir en el esquema de entrada.
- Si se especifican ambos, el valor numérico debe ser lo primero

Ejemplos:

```
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;
```

REEQUILIBRAR

Reequilibra las particiones de salida de los resultados de la consulta para que cada partición tenga un tamaño razonable (ni demasiado pequeña ni demasiado grande). Se trata de una operación que se realiza con el máximo esfuerzo: si hay sesgos, AWS Clean Rooms dividirá las particiones asimétricas para que no sean demasiado grandes. Esta sugerencia resulta útil cuando se necesita escribir el resultado de una consulta en una tabla para evitar archivos demasiado pequeños o demasiado grandes.

Parámetros:

- Valor numérico (opcional): número de particiones; debe ser un entero positivo entre 1 y 2147483647
- Identificadores de columna (opcionales): las columnas deben aparecer en la lista de resultados SELECT
- Si se especifican ambos, el valor numérico debe figurar primero

Ejemplos:

```
-- 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;
```

Combinar varias sugerencias

Puede especificar varias sugerencias en una sola consulta separándolas con comas:

```
-- 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;
```

Consideraciones y limitaciones

- Las sugerencias son sugerencias de optimización, no comandos. El optimizador de consultas puede ignorar las sugerencias en función de las restricciones de recursos o las condiciones de ejecución.
- Las sugerencias se incrustan directamente en las cadenas de consulta SQL para CreateAnalysisTemplate y StartProtectedQuery APIs.
- Las sugerencias deben colocarse directamente después de la palabra clave SELECT.
- Los parámetros con nombre no se admiten con sugerencias y generarán una excepción.
- Los nombres de las columnas de las sugerencias REPARTITION y REPARTITION_BY_RANGE deben existir en el esquema de entrada.
- Los nombres de las columnas de las sugerencias de REBALANCE deben aparecer en la lista de resultados SELECT.
- Los parámetros numéricos deben ser enteros positivos entre 1 y 2147483647. No se admiten anotaciones científicas como 1e1
- Las sugerencias no se admiten en las consultas SQL de privacidad diferencial.

- Los PySpark trabajos no admiten sugerencias para consultas SQL. Para proporcionar directrices para los planes de ejecución de un PySpark trabajo, utilice la API de marco de datos. Consulte los documentos de la DataFrame API de Apache Spark para obtener más información.

SELECT

El comando SELECT devuelve filas de tablas y funciones definidas por el usuario.

AWS Clean Rooms Spark SQL admite los siguientes comandos, cláusulas y operadores de conjuntos SELECT SQL:

Temas

- SELECT list
- Cláusula WITH
- Cláusula FROM
- Cláusula JOIN
- Cláusula WHERE
- cláusula VALUES
- Cláusula GROUP BY
- Cláusula HAVING
- Operadores de establecimiento
- Cláusula ORDER BY
- Ejemplos de subconsultas
- Subconsultas correlacionadas

La sintaxis, los argumentos y algunos ejemplos provienen de la Referencia SQL de Apache Spark.

SELECT list

La SELECT list designa las columnas, funciones y expresiones que se desea que devuelva la consulta. La lista representa el resultado de la consulta.

Sintaxis

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

Parameters

DISTINCT

Opción que elimina las filas duplicadas del conjunto de resultados basándose en los valores coincidentes de una o más columnas.

expression

Una expresión formada a partir de una o más columnas que existen en las tablas a las que hace referencia la consulta. Una expresión puede contener funciones SQL. Por ejemplo:

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

AS column_alias

Un nombre temporal para la columna que se utiliza en el conjunto de resultados finales. La palabra clave AS es opcional. Por ejemplo:

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

Si no se especifica un alias para una expresión que no sea un nombre de columna simple, el conjunto de resultados aplica un nombre predeterminado a esa columna.

> **Note**
>
> El alias se reconoce justo después de definirlo en la lista de destino. No puedes usar un alias en otras expresiones definidas después de este en la misma lista de objetivos.

Cláusula WITH

Una cláusula WITH es una cláusula opcional que precede a la lista SELECT en una consulta. La cláusula WITH define una o más common_table_expressions. Cada expresión común de tabla (CTE) define una tabla temporal, que es similar a la definición de una vista. Puede referenciar estas tablas temporales en la cláusula FROM. Solo se utilizan mientras se ejecuta la consulta a la que pertenecen. Cada CTE de la cláusula WITH especifica un nombre de tabla, una lista opcional de nombres de columnas y una expresión de consulta que toma el valor de una tabla (una instrucción SELECT).

Las subconsultas de la cláusula WITH son una manera eficiente de definir tablas que puede utilizarse al ejecutar una única consulta. En todos los casos, se pueden obtener los mismos resultados al utilizar subconsultas en el cuerpo principal de la instrucción SELECT, pero las subconsultas de la cláusula WITH pueden resultar más sencillas de escribir y leer. Cuando es posible, las subconsultas de la cláusula WITH a las que se hace referencia varias veces se optimizan como subexpresiones comunes; es decir, puede ser posible evaluar una subconsulta WITH una vez y reutilizar sus resultados (tenga en cuenta que las subexpresiones comunes no se limitan a aquellas definidas en la cláusula WITH).

Sintaxis

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

donde common_table_expression puede ser no recursiva. A continuación se presenta la forma no recursiva:

```
CTE_table_name AS ( query )
```

Parameters

common_table_expression

Define una tabla temporal a la que se puede referenciar en Cláusula FROM y se utiliza solo durante la ejecución de la consulta a la que pertenece.

CTE_table_name

Un nombre único para una tabla temporal que define los resultados de una subconsulta de la cláusula WITH. No se pueden usar nombres duplicados dentro de una cláusula WITH. Cada subconsulta debe tener un nombre de tabla al que se pueda hacer referencia en la Cláusula FROM.

consulta

Cualquier consulta SELECT que AWS Clean Rooms admita. Consulte SELECT.

Notas de uso

Puede usar una cláusula WITH en las siguientes instrucciones SQL:

- SELECCIONE, CON, UNION, UNION ALL, INTERSECT, INTERSECT ALL, EXCEPTO O EXCEPTO ALL

Si la cláusula FROM de una consulta que contiene una cláusula WITH no referencia ninguna de las tablas definidas por la cláusula WITH, se ignora la cláusula WITH y la consulta se ejecuta como siempre.

Se puede hacer referencia a una tabla definida por una subconsulta de la cláusula WITH solo en el alcance de la consulta SELECT que inicia la cláusula WITH. Por ejemplo, se puede hacer referencia a dicha tabla en la cláusula FROM de una subconsulta en la lista SELECT, la cláusula WHERE o la cláusula HAVING. No se puede usar una cláusula WITH en una subconsulta y hacer referencia a su tabla en la cláusula FROM de una consulta principal o de otra subconsulta. Este patrón de consulta provoca un mensaje de error `relation table_name doesn't exist` para la tabla de la cláusula WITH.

No se puede especificar otra cláusula WITH dentro de una subconsulta de la cláusula WITH.

No se pueden realizar referencias futuras a tablas definidas por las subconsultas de la cláusula WITH. Por ejemplo, la siguiente consulta devuelve un error debido a la referencia futura a la tabla W2 en la definición de la tabla W1:

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

Ejemplos

En el siguiente ejemplo, se muestra el caso posible más simple de una consulta que contiene una cláusula WITH. La consulta WITH denominada VENUECOPY selecciona todas las filas de la tabla VENUE. La consulta principal, a su vez, selecciona todas las filas de VENUECOPY. La tabla VENUECOPY existe solo durante esta consulta.

```
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)
```

En el siguiente ejemplo, se muestra una cláusula WITH que produce dos tablas, denominadas VENUE_SALES y TOP_VENUES. La segunda tabla de la consulta WITH selecciona desde la primera. A su vez, la cláusula WHERE del bloque de la consulta principal contiene una subconsulta que limita la tabla TOP_VENUES.

```
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)
```

En los siguientes dos ejemplos se muestran las reglas para el alcance de las referencias de la tabla en función de las subconsultas de la cláusula WITH. La primera consulta se ejecuta, pero en la segunda se produce un error inesperado. La primera consulta tiene una subconsulta de la cláusula WITH dentro de la lista SELECT de la consulta principal. Se hace referencia a la tabla definida por la cláusula WITH (HOLIDAYS) en la cláusula FROM de la subconsulta de la lista SELECT:

```
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)
```

La segunda consulta falla porque intenta hacer referencia a la tabla HOLIDAYS en la consulta principal, así como en la subconsulta de la lista SELECT. Las referencias de la consulta principal están fuera de alcance.

```
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
```

Cláusula FROM

La cláusula FROM en una consulta enumera las referencias de la tabla (tablas, vistas y subconsultas) desde las que se seleccionan los datos. Si se enumeran varias referencias de tabla, se deben combinar las tablas a través de la sintaxis adecuada en la cláusula FROM o en la cláusula WHERE. Si no se especifican criterios de combinación, el sistema procesa la consulta como una combinación cruzada (producto cartesiano).

Temas

- Sintaxis
- Parameters
- Notas de uso

Sintaxis

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

donde table_reference es uno de los siguientes:

```
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

Una tabla definida por una subconsulta en la Cláusula WITH.

table_name

Nombre de una tabla o vista.

alias

Nombre alternativo temporal para una tabla o vista. Se debe proporcionar un alias para una tabla obtenida de una subconsulta. En otras referencias de tabla, los alias son opcionales La palabra clave AS es siempre opcional. Los alias de la tabla brindan un acceso directo para identificar tablas en otras partes de una consulta, como la cláusula WHERE.

Por ejemplo:

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

Si hay un alias de tabla definido, se debe usar el alias para hacer referencia a esa tabla en la consulta.

Por ejemplo, si la consulta es `SELECT "tbl"."col" FROM "tbl" AS "t"`, la consulta dará error porque en este caso el nombre de la tabla básicamente se anula. Una consulta válida en este caso sería `SELECT "t"."col" FROM "tbl" AS "t"`.

column_alias

Nombre alternativo temporal para una columna en una tabla o vista.

subquery

Una expresión de consulta que toma el valor de una tabla. La tabla solo existe mientras dura la consulta y, por lo general, se le asigna un nombre o un alias. No obstante, no es obligatorio tener un alias. También puede definir nombres de columnas para tablas que derivan de subconsultas. Designar un nombre a los alias de las columnas es importante cuando desea combinar los resultados de las subconsultas con otras tablas y cuando desea seleccionar o limitar esas columnas en otros sitios de la consulta.

Una subconsulta puede contener una cláusula ORDER BY, pero es posible que esta cláusula no tenga ningún efecto si no se especifica también una cláusula OFFSET o LIMIT.

NATURAL

Define una combinación que utiliza automáticamente todos los pares de columnas con nombres idénticos en las dos tablas como las columnas de combinación. No se requiere una condición de combinación explícita. Por ejemplo, si las tablas CATEGORY y EVENT tienen columnas denominadas CATID, una combinación natural de estas tablas es una combinación de las columnas CATID.

> **Note**
>
> Si se especifica una combinación NATURAL, pero no existen pares de columnas con nombres idénticos en las tablas que deben combinarse, la consulta se establece en una combinación cruzada.

join_type

Especifique uno de los siguientes tipos de combinación:

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

Las combinaciones cruzadas son combinaciones no calificadas; devuelven el producto cartesiano de dos tablas.

Las combinaciones internas y externas son combinaciones calificadas. Están calificadas implícitamente (en combinaciones naturales), con la sintaxis ON o USING en la cláusula FROM, o con una condición WHERE.

Una combinación interna devuelve filas coincidentes únicamente en función a la condición de combinación o a la lista de columnas de combinación. Una combinación externa devuelve todas las filas que la combinación interna equivalente devolvería, además de filas no coincidentes de la tabla "izquierda", tabla "derecha" o ambas tablas. La tabla izquierda es la primera tabla de la lista, y la tabla derecha es la segunda tabla de la lista. Las filas no coincidentes contienen valores NULL para llenar el vacío de las columnas de salida.

ON join_condition

Especificación del tipo de combinación donde las columnas de combinación se establecen como una condición que sigue la palabra clave ON. Por ejemplo:

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

USING (join_column [, ...])

Especificación del tipo de combinación donde las columnas de combinación aparecen enumeradas entre paréntesis. Si se especifican varias columnas de combinación, se delimitan por comas. La palabra clave USING debe preceder a la lista. Por ejemplo:

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

Notas de uso

Las columnas de combinación deben tener tipos de datos comparables.

Una combinación NATURAL o USING retiene solo uno de cada par de columnas de combinación en el conjunto de resultados intermedios.

Una combinación con la sintaxis ON retiene ambas columnas de combinación en su conjunto de resultados intermedios.

Véase también Cláusula WITH.

Cláusula JOIN

Se utiliza una cláusula JOIN de SQL para combinar los datos de dos o más tablas en función de los campos comunes. Es posible que los resultados cambien o no cambien según el método de combinación especificado. Las combinaciones externas izquierdas y derechas conservan valores de una de las tablas combinadas cuando no se encuentra una coincidencia en la otra tabla.

La combinación del tipo JOIN y la condición de unión determina qué filas se incluyen en el conjunto de resultados final. A continuación, las cláusulas SELECT y WHERE controlan qué columnas se devuelven y cómo se filtran las filas. Comprender los diferentes tipos de JOIN y cómo utilizarlos de forma eficaz es una habilidad crucial en SQL, ya que permite combinar datos de varias tablas de forma flexible y eficaz.

Sintaxis

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

Parameters

SELECCIONE la columna 1, la columna 2,..., la columna N

Las columnas que desea incluir en el conjunto de resultados. Puede seleccionar columnas de una o de las dos tablas incluidas en la COMBINACIÓN.

DE LA TABLA 1

La primera tabla (izquierda) de la operación JOIN.

[UNIÓN | UNIÓN INTERIOR | UNIÓN IZQUIERDA [EXTERIOR] | UNIÓN DERECHA [EXTERIOR] UNIÓN | UNIÓN COMPLETA [EXTERIOR]] Tabla 2:

El tipo de UNIÓN que se va a realizar. JOIN o INNER JOIN devuelven solo las filas con valores coincidentes en ambas tablas.

LEFT [OUTER] JOIN devuelve todas las filas de la tabla de la izquierda, con las filas coincidentes de la tabla de la derecha.

RIGHT [OUTER] JOIN devuelve todas las filas de la tabla de la derecha, con las filas coincidentes de la tabla de la izquierda.

FULL [OUTER] JOIN devuelve todas las filas de ambas tablas, independientemente de si coinciden o no.

CROSS JOIN crea un producto cartesiano de las filas de las dos tablas.

EN la tabla1.columna = tabla2.columna

La condición de unión, que especifica cómo se hacen coincidir las filas de las dos tablas. La condición de unión se puede basar en una o más columnas.

Condición WHERE:

Cláusula opcional que se puede utilizar para filtrar aún más el conjunto de resultados en función de una condición específica.

Ejemplo

El ejemplo siguiente es una combinación entre dos tablas con la cláusula USING. En este caso, las columnas listid y eventid se utilizan como columnas de combinación. Los resultados tienen un límite de cinco filas.

```
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
```

Tipos de combinación

INNER

Este es el tipo de unión predeterminado. Devuelve las filas que tienen valores coincidentes en ambas referencias de tabla.

La combinación interna es el tipo de combinación más común que se utiliza en SQL. Es una forma eficaz de combinar datos de varias tablas en función de una columna o conjunto de columnas común.

Sintaxis:

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

La siguiente consulta devolverá todas las filas en las que haya un valor de custome_id coincidente entre las tablas de clientes y pedidos. El conjunto de resultados contendrá las columnas customer_id, name, order_id y order_date.

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

La siguiente consulta es una combinación interna (sin la palabra clave JOIN) entre la tabla LISTING y la tabla SALES, donde LISTID de la tabla LISTING está entre 1 y 5. Esta consulta relaciona los

valores de la columna LISTID en la tabla LISTING (la tabla izquierda) y la tabla SALES (la tabla derecha). Los resultados muestran que LISTID 1, 4 y 5 coinciden con los criterios.

```
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
```

El siguiente ejemplo es una combinación interna con la cláusula ON. En este caso, las filas NULL no se devuelven.

```
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
```

La siguiente consulta es una combinación interna de dos subconsultas en la cláusula FROM. La consulta busca la cantidad de tickets vendidos y sin vender para diferentes categorías de eventos (conciertos y espectáculos). Estas subconsultas de la cláusula FROM son subconsultas de tabla; pueden devolver varias columnas y filas.

```
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
```

IZQUIERDA [EXTERIOR]

Devuelve todos los valores de la referencia de la tabla izquierda y los valores coincidentes de la referencia de la tabla derecha, o añade NULL si no hay ninguna coincidencia. También se conoce como unión exterior izquierda.

Devuelve todas las filas de la tabla izquierda (primera) y las filas coincidentes de la tabla derecha (segunda). Si no hay ninguna coincidencia en la tabla de la derecha, el conjunto de resultados contendrá valores NULOS para las columnas de la tabla de la derecha. La palabra clave OUTER se puede omitir y la unión se puede escribir simplemente como LEFT JOIN. Lo opuesto a una unión exterior izquierda es una unión exterior derecha, que devuelve todas las filas de la tabla de la derecha y las filas coincidentes de la tabla de la izquierda.

Sintaxis:

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

La siguiente consulta devolverá todas las filas de la tabla de clientes, junto con las filas coincidentes de la tabla de pedidos. Si un cliente no tiene ningún pedido, el conjunto de resultados seguirá incluyendo la información del cliente, con valores NULOS para las columnas order_id y order_date.

```
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;
```

La siguiente consulta es una combinación externa izquierda. Las combinaciones externas izquierdas y derechas conservan valores de una de las tablas combinadas cuando no se encuentra una coincidencia en la otra tabla. Las tablas izquierda y derecha son la primera tabla y la segunda tabla que aparecen en la sintaxis. Los valores NULL se utilizan para rellenar los "espacios" en el conjunto de resultados. Esta consulta relaciona los valores de la columna LISTID en la tabla LISTING (la tabla izquierda) y la tabla SALES (la tabla derecha). Los resultados muestran que LISTIDs 2 y 3 no generaron ninguna venta.

```
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
```

DERECHA [EXTERIOR]

Devuelve todos los valores de la referencia de la tabla derecha y los valores coincidentes de la referencia de la tabla izquierda, o añade NULL si no hay ninguna coincidencia. También se conoce como unión exterior derecha.

Devuelve todas las filas de la tabla derecha (segunda) y las filas coincidentes de la tabla izquierda (primera). Si no hay ninguna coincidencia en la tabla de la izquierda, el conjunto de resultados contendrá valores NULOS para las columnas de la tabla de la izquierda. La palabra clave OUTER se puede omitir y la unión se puede escribir simplemente como RIGHT JOIN. Lo opuesto a una unión exterior derecha es una unión exterior izquierda, que devuelve todas las filas de la tabla izquierda y las filas coincidentes de la tabla derecha.

Sintaxis:

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

La siguiente consulta devolverá todas las filas de la tabla de clientes, junto con las filas coincidentes de la tabla de pedidos. Si un cliente no tiene ningún pedido, el conjunto de resultados seguirá incluyendo la información del cliente, con valores NULOS para las columnas order_id y order_date.

```
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;
```

La siguiente consulta es una combinación externa derecha. Esta consulta relaciona los valores de la columna LISTID en la tabla LISTING (la tabla izquierda) y la tabla SALES (la tabla derecha). Los resultados muestran que LISTIDs 1, 4 y 5 coinciden con los criterios.

```
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
```

COMPLETO [EXTERIOR]

Devuelve todos los valores de ambas relaciones, añadiendo valores NULL en el lado que no coincida. También se conoce como unión externa completa.

Devuelve todas las filas de las tablas izquierda y derecha, independientemente de si coinciden o no. Si no hay ninguna coincidencia, el conjunto de resultados contendrá valores NULOS para las columnas de la tabla que no tengan ninguna fila coincidente. La palabra clave OUTER se puede omitir y la unión se puede escribir simplemente como FULL JOIN. La combinación externa completa se usa con menos frecuencia que la unión externa izquierda o la unión externa derecha, pero puede

resultar útil en algunos escenarios en los que es necesario ver todos los datos de ambas tablas, incluso si no hay coincidencias.

Sintaxis:

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

La siguiente consulta devolverá todas las filas de las tablas de clientes y de pedidos. Si un cliente no tiene ningún pedido, el conjunto de resultados seguirá incluyendo la información del cliente, con valores NULOS para las columnas order_id y order_date. Si un pedido no tiene ningún cliente asociado, el conjunto de resultados incluirá ese pedido, con valores NULOS para las columnas customer_id y name.

```
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;
```

La siguiente consulta es una combinación completa. Las combinaciones completas retienen valores de las tablas combinadas cuando no se encuentra una coincidencia en la otra tabla. Las tablas izquierda y derecha son la primera tabla y la segunda tabla que aparecen en la sintaxis. Los valores NULL se utilizan para rellenar los "espacios" en el conjunto de resultados. Esta consulta relaciona los valores de la columna LISTID en la tabla LISTING (la tabla izquierda) y la tabla SALES (la tabla derecha). Los resultados muestran que LISTIDs 2 y 3 no generaron ninguna venta.

```
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
```

La siguiente consulta es una combinación completa. Esta consulta relaciona los valores de la columna LISTID en la tabla LISTING (la tabla izquierda) y la tabla SALES (la tabla derecha). En los resultados solo aparecen las filas que no generan ventas (LISTIDs 2 y 3).

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing full join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
and (listing.listid IS NULL or sales.listid IS NULL)
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     2 | NULL   | NULL
     3 | NULL   | NULL
```

[IZQUIERDA] SEMIRREMOLQUE

Devuelve los valores del lado izquierdo de la referencia de la tabla que coinciden con los de la derecha. También se conoce como semiunión izquierda.

Solo devuelve las filas de la tabla izquierda (primera) que tienen una fila coincidente en la tabla derecha (segunda). No devuelve ninguna columna de la tabla de la derecha, solo las columnas de la tabla de la izquierda. El comando LEFT SEMI JOIN es útil cuando se quieren buscar las filas de una tabla que coinciden con las de otra tabla, sin necesidad de devolver ningún dato de la segunda tabla. LEFT SEMI JOIN es una alternativa más eficaz que utilizar una subconsulta con una cláusula IN o EXISTS.

Sintaxis:

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

La siguiente consulta devolverá solo las columnas customer_id y name de la tabla de clientes, para los clientes que tengan al menos un pedido en la tabla de pedidos. El conjunto de resultados no incluirá ninguna columna de la tabla de pedidos.

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

CROSS JOIN

Devuelve el producto cartesiano de dos relaciones. Esto significa que el conjunto de resultados contendrá todas las combinaciones posibles de filas de las dos tablas, sin aplicar ninguna condición ni filtro.

El método CROSS JOIN resulta útil cuando se necesitan generar todas las combinaciones posibles de datos a partir de dos tablas, como en el caso de crear un informe que muestre todas las combinaciones posibles de información sobre clientes y productos. La COMBINACIÓN CRUZADA es diferente de otros tipos de combinación (COMBINACIÓN INTERIOR, UNIÓN IZQUIERDA, etc.) porque no tiene una condición de unión en la cláusula ON. La condición de unión no es obligatoria para una COMBINACIÓN CRUZADA.

Sintaxis:

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

La siguiente consulta devolverá un conjunto de resultados que contiene todas las combinaciones posibles de customer_id, customer_name, product_id y product_name de las tablas de clientes y productos. Si la tabla de clientes tiene 10 filas y la tabla de productos tiene 20 filas, el conjunto de resultados del CROSS JOIN contendrá 10 x 20 = 200 filas.

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

La siguiente consulta es una combinación cruzada o cartesiana de la tabla LISTING y la tabla SALES con un predicado para limitar los resultados. Esta consulta hace coincidir los valores de las columnas LISTID de la tabla VENTAS y los valores LISTIDs 1, 2, 3, 4 y 5 de la tabla LISTING de ambas tablas. Los resultados muestran que 20 filas coinciden con los criterios.

```
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
```

ANTIUNIÓN

Devuelve los valores de la referencia de la tabla izquierda que no coinciden con la referencia de la tabla derecha. También se conoce como antiunión izquierda.

La función ANTI JOIN es una operación útil cuando se quieren encontrar las filas de una tabla que no coinciden con las de otra.

Sintaxis:

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

La siguiente consulta mostrará todos los clientes que no han realizado ningún pedido.

```
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

Especifica que las filas de las dos relaciones coincidirán implícitamente en igualdad de condiciones en todas las columnas con nombres coincidentes.

Hace coincidir automáticamente las columnas con el mismo nombre y tipo de datos entre las dos tablas. No requiere que especifique explícitamente la condición de unión en la cláusula ON. Combina todas las columnas coincidentes de las dos tablas en el conjunto de resultados.

La combinación NATURAL es una forma abreviada práctica cuando las tablas que se van a unir tienen columnas con los mismos nombres y tipos de datos. Sin embargo, generalmente se recomienda usar la combinación interna más explícita... La sintaxis ON permite que las condiciones de unión sean más explícitas y fáciles de entender.

Sintaxis:

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

El siguiente ejemplo es una unión natural entre dos tablas `employees` y`departments`, con las siguientes columnas:

- `employees`tabla:`employee_id`,`first_name`,`last_name`, `department_id`
- `departments`mesa:`department_id`, `department_name`

La siguiente consulta devolverá un conjunto de resultados que incluye el nombre, los apellidos y el nombre del departamento de todas las filas coincidentes entre las dos tablas, según la `department_id` columna.

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
```

```
NATURAL JOIN departments d;
```

El ejemplo siguiente es una combinación natural entre dos tablas. En este caso, las columnas listid, sellerid, eventid y dateid tienen nombres y tipos de datos idénticos en ambas tablas y, por lo tanto, se utilizan como columnas de combinación. Los resultados tienen un límite de cinco filas.

```
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
```

Cláusula WHERE

La cláusula WHERE contiene condiciones que combinan tablas o que aplican predicados a columnas de las tablas. Las tablas pueden combinarse de manera interna a través de la sintaxis adecuada en la cláusula WHERE o en la cláusula FROM. Los criterios de combinación externa deben especificarse en la cláusula FROM.

Sintaxis

```
[ WHERE condition ]
```

condition

Cualquier condición de búsqueda con un resultado booleano, como una condición de combinación o un predicado en una columna de la tabla. Los siguientes ejemplos son condiciones de combinación válidas:

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

Los siguientes ejemplos son condiciones válidas de columnas en tablas:

```
catgroup like 'S%'
venueseats between 20000 and 50000
eventname in('Jersey Boys','Spamalot')
year=2008
length(catdesc)>25
date_part(month, caldate)=6
```

Las condiciones pueden ser simples o complejas. Para las condiciones complejas, puede utilizar paréntesis para aislar las unidades lógicas. En el siguiente ejemplo, la condición de combinación está entre paréntesis.

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

Notas de uso

Puede usar alias en la cláusula WHERE para hacer referencia a expresiones de listas de selección.

No puede limitar los resultados de las funciones de agregación en la cláusula WHERE; utilice la cláusula HAVING con este fin.

Las columnas que están limitadas en la cláusula WHERE deben derivar de referencias de tabla en la cláusula FROM.

Ejemplo

La siguiente consulta utiliza una combinación de diferentes restricciones de la cláusula WHERE, incluida una condición de combinación para las tablas SALES y EVENT, un predicado en la columna EVENTNAME y dos predicados en la columna STARTTIME.

```
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)
```

cláusula VALUES

La cláusula VALUES se usa para proporcionar un conjunto de valores de fila directamente en la consulta, sin necesidad de hacer referencia a una tabla.

La cláusula VALUES se puede utilizar en los siguientes escenarios:

- Puede usar la cláusula VALUES en una instrucción INSERT INTO para especificar los valores de las nuevas filas que se insertan en una tabla.
- Puede utilizar la cláusula VALUES por sí sola para crear un conjunto de resultados temporal, o una tabla en línea, sin necesidad de hacer referencia a una tabla.
- Puede combinar la cláusula VALUES con otras cláusulas SQL, como WHERE, ORDER BY o LIMIT, para filtrar, ordenar o limitar las filas del conjunto de resultados.

Esta cláusula resulta especialmente útil cuando se necesita insertar, consultar o manipular un conjunto pequeño de datos directamente en la sentencia SQL, sin necesidad de crear o hacer referencia a una tabla permanente. Le permite definir los nombres de las columnas y los valores correspondientes para cada fila, lo que le brinda la flexibilidad de crear conjuntos de resultados temporales o insertar datos sobre la marcha, sin la sobrecarga de administrar una tabla independiente.

Sintaxis

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

Parameters

expression

Expresión que especifica una combinación de uno o más valores, operadores y funciones SQL que da como resultado un valor.

table_alias

Un alias que especifica un nombre temporal con una lista de nombres de columnas opcional.

Ejemplo

El siguiente ejemplo crea una tabla en línea, un conjunto de resultados similar a una tabla temporal con dos columnas, y. `col1` `col2` La única fila del conjunto de resultados contiene los valores `"one"` y1, respectivamente. La `SELECT * FROM` parte de la consulta simplemente recupera todas las columnas y filas de este conjunto de resultados temporal. El sistema de base de datos genera automáticamente los nombres de las columnas (`col1ycol2`), ya que la cláusula VALUES no especifica explícitamente los nombres de las columnas.

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

Si desea definir nombres de columnas personalizados, puede hacerlo utilizando una cláusula AS después de la cláusula VALUES, de la siguiente manera:

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

Esto crearía un conjunto de resultados temporal con los nombres de las columnas `name` `yid`, en lugar del predeterminado `col1` `ycol2`.

Cláusula GROUP BY

La cláusula GROUP BY identifica las columnas de agrupación para la consulta. Las columnas de agrupación deben declararse cuando la consulta computa las agregaciones con funciones estándar como SUM, AVG y COUNT. Si hay una función de agregado en la expresión SELECT, cualquier columna de la expresión SELECT que no esté en una función de agregado debe estar en la cláusula GROUP BY.

Para obtener más información, consulte AWS Clean Rooms Funciones de Spark SQL.

Sintaxis

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

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

Parámetros

expr

La lista de columnas o expresiones debe coincidir con la lista de expresiones no agregadas en la lista de selección de la consulta. Por ejemplo, considere la siguiente consulta simple.

```
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)
```

En esta consulta, la lista de selección consta de dos expresiones agregadas. La primera usa la función SUM y la segunda usa la función COUNT. Las dos columnas restantes, LISTID y EVENTID, deben declararse como columnas de agrupación.

Las expresiones de la cláusula GROUP BY también pueden hacer referencia a la lista de selección a través de números ordinales. Por ejemplo, el caso anterior podría abreviarse de la siguiente manera.

```
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

Puede utilizar la extensión de agregación ROLLUP para realizar el trabajo de varias operaciones GROUP BY en una sola instrucción. Para obtener más información sobre las extensiones de agregación y las funciones relacionadas, consulte Extensiones de agregación.

Extensiones de agregación

AWS Clean Rooms admite extensiones de agregación para realizar el trabajo de varias operaciones GROUP BY en una sola sentencia.

GROUPING SETS

Calcula uno o más conjuntos de agrupación en una sola instrucción. Un conjunto de agrupación es el conjunto de una sola cláusula GROUP BY, un conjunto de 0 o más columnas mediante el que se puede agrupar el conjunto de resultados de una consulta. GROUP BY GROUPING SETS equivale a ejecutar una consulta UNION ALL en un conjunto de resultados agrupado por columnas diferentes. Por ejemplo, GROUP BY GROUPING SETS((a), (b)) equivale a GROUP BY a UNION ALL GROUP BY b.

En el siguiente ejemplo se devuelve el costo de los productos de la tabla de pedidos agrupados en función tanto de las categorías de los productos como del tipo de productos vendidos.

```
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

Se supone una jerarquía en la que las columnas anteriores se consideran las principales de las columnas posteriores. ROLLUP agrupa los datos por las columnas proporcionadas y devuelve filas de subtotales adicionales que representan los totales de todos los niveles de agrupación de columnas, además de las filas agrupadas. Por ejemplo, puede usar GROUP BY ROLLUP ((a), (b)) para devolver un conjunto de resultados agrupado primero por a y luego por b, suponiendo que b es una subsección de a. ROLLUP también devuelve una fila con todo el conjunto de resultados sin agrupar columnas.

GROUP BY ROLLUP((a), (b)) equivale a GROUP BY GROUPING SETS((a,b), (a), ()).

En el siguiente ejemplo se devuelve el costo de los productos de la tabla de pedidos agrupados primero por categoría y, a continuación, por producto, con el producto como una subdivisión de la categoría.

```
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

Agrupa los datos por las columnas proporcionadas y devuelve filas de subtotales adicionales que representan los totales de todos los niveles de agrupación de columnas, además de las filas agrupadas. CUBE devuelve las mismas filas que ROLLUP, a la vez que agrega filas de subtotales adicionales por cada combinación de columnas de agrupación no incluidas en ROLLUP. Por ejemplo, puede usar GROUP BY CUBE ((a), (b)) para devolver un conjunto de resultados agrupado primero por a y luego por b, suponiendo que b es una subsección de a. CUBE también devuelve una fila con todo el conjunto de resultados sin agrupar columnas.

GROUP BY CUBE((a), (b)) equivale a GROUP BY GROUPING SETS((a, b), (a), (b), ()).

En el siguiente ejemplo se devuelve el costo de los productos de la tabla de pedidos agrupados primero por categoría y, a continuación, por producto, con el producto como una subdivisión de la categoría. A diferencia del ejemplo anterior de ROLLUP, la instrucción devuelve resultados para cada combinación de columnas de agrupación.

```
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)
```

Cláusula HAVING

La cláusula HAVING aplica una condición al conjunto de resultados agrupado intermedio que una consulta devuelve.

Sintaxis

```
[ HAVING condition ]
```

Por ejemplo, puede limitar los resultados de una función SUM:

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

La condición HAVING se aplica después de que se aplican todas las condiciones de la cláusula WHERE y se completan todas las operaciones de GROUP BY.

La condición toma la misma forma que cualquier condición de la cláusula WHERE.

Notas de uso

- Cualquier columna a la que se haga referencia en una condición de la cláusula HAVING debe ser una columna de agrupación o una columna que haga referencia al resultado de una función agregada.
- En una cláusula HAVING, no se puede especificar:
  - Un número ordinal que hace referencia a un elemento de la lista de selección. Solo las cláusulas GROUP BY y ORDER BY aceptan números ordinales.

Ejemplos

La siguiente consulta calcula las ventas de tickets totales para todos los eventos por nombre y, luego, elimina eventos donde las ventas totales sean inferiores a $800 000. La condición HAVING se aplica a los resultados de la función agregada en la lista de selección: `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)
```

La siguiente consulta calcula un conjunto de resultados similar. No obstante, en este caso, la condición HAVING se aplica a una agregación que no se especifica en la lista de selección: `sum(qtysold)`. Los eventos que no vendieron más de 2 000 tickets se eliminan del resultado final.

```
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)
```

Operadores de establecimiento

Los operadores de conjunto se utilizan para comparar y combinar los resultados de dos expresiones de consulta distintas.

AWS Clean Rooms Spark SQL admite los siguientes operadores de conjunto que se muestran en la siguiente tabla.

Operador de conjunto
INTERSECT
INTERSECAR TODOS
EXCEPT

Operador de conjunto
EXCEPTO TODOS
UNION
UNION ALL

Por ejemplo, si desea saber qué usuarios de un sitio web son compradores y vendedores pero los nombres de usuario están almacenados en diferentes columnas o tablas, puede buscar la intersección de estos dos tipos de usuarios. Si desea saber qué usuarios de un sitio web son compradores pero no vendedores, puede usar el operador EXCEPT para buscar la diferencia entre las dos listas de usuarios. Si desea crear una lista de todos los usuarios, independientemente de la función, puede usar el operador UNION.

> **Note**
>
> Las cláusulas ORDER BY, LIMIT, SELECT TOP y OFFSET no se pueden utilizar en las expresiones de consulta combinadas por los operadores de conjunto UNION, UNION ALL, INTERSECT y EXCEPT.

Temas

Sintaxis

```
subquery1
```

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

Parameters

subconsulta1, subconsulta2

Expresión de consulta que corresponde, en forma de lista de selección, a una segunda expresión de consulta que sigue al operador UNION, UNION ALL, INTERSECT, INTERSECT ALL, EXCEPT o EXCEPT ALL. Las dos expresiones deben contener la misma cantidad de columnas de salida con tipos de datos compatibles; de lo contrario, no se podrán comparar ni fusionar los dos conjuntos de resultados. Las operaciones de conjunto no permiten la conversión implícita entre diferentes categorías de tipos de datos. Para obtener más información, consulte Conversión y compatibilidad de tipos.

Puede crear consultas que contengan una cantidad ilimitada de expresiones de consulta y vincularlas con operadores UNION, INTERSECT y EXCEPT en cualquier combinación. Por ejemplo, la siguiente estructura de consulta es válida, suponiendo que las tablas T1, T2 y T3 contienen conjuntos de columnas compatibles:

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

UNIÓN [TODOS | DISTINTOS]

Operación de conjunto que devuelve filas de dos expresiones de consulta, independientemente de si las filas provienen de una o ambas expresiones.

INTERSECAR [TODOS | DISTINTOS]

Operación de conjunto que devuelve filas que provienen de dos expresiones de consulta. Las filas que no se devuelven en las dos expresiones se descartan.

EXCEPTO [TODOS | DISTINTOS]

Operación de conjunto que devuelve filas que provienen de una de las dos expresiones de consulta. Para calificar para el resultado, las filas deben existir en la primera tabla de resultados, pero no en la segunda.

EXCEPT ALL no elimina los duplicados de las filas de resultados.

MINUS y EXCEPT son sinónimos exactos.

Orden de evaluación para los operadores de conjunto

Los operadores de conjunto UNION y EXCEPT se asocian por la izquierda. Si no se especifican paréntesis para establecer el orden de prioridad, los operadores se evalúan de izquierda a derecha. Por ejemplo, en la siguiente consulta, UNION de T1 y T2 se evalúa primero, luego se realiza la operación EXCEPT en el resultado de UNION:

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

El operador INTERSECT prevalece sobre los operadores UNION y EXCEPT cuando se utiliza una combinación de operadores en la misma consulta. Por ejemplo, la siguiente consulta evalúa la intersección de T2 y T3, y luego unirá el resultado con T1:

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

Al agregar paréntesis, puede aplicar un orden diferente de evaluación. En el siguiente caso, el resultado de la unión de T1 y T2 está intersectado con T3, y la consulta probablemente produzca un resultado diferente.

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

Notas de uso

- Los nombres de la columnas que se devuelven en el resultado de una consulta de operación de conjunto son los nombres (o alias) de la columnas de las tablas de la primera expresión de consulta. Debido a que estos nombres de columnas pueden ser confusos, porque los valores de la columna provienen de tablas de cualquier lado del operador de conjunto, se recomienda proporcionar alias significativos para el conjunto de resultados.
- Cuando las consultas del operador de conjunto devuelven resultados decimales, las columnas de resultado correspondientes se promueven a devolver la misma precisión y escala. Por ejemplo, en la siguiente consulta, donde T1.REVENUE es una columna DECIMAL(10,2) y T2.REVENUE es una columna DECIMAL(8,4), el resultado decimal se promueve a DECIMAL(12,4):

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

  La escala es 4 ya que es la escala máxima de las dos columnas. La precisión es 12 ya que T1.REVENUE requiere 8 dígitos a la izquierda del punto decimal (12 - 4 = 8). Este tipo de promoción garantiza que todos los valores de ambos lados de UNION encajen en el resultado. Para valores de 64 bits, la precisión de resultados máxima es 19 y la escala de resultados máxima es 18. Para valores de 128 bits, la precisión de resultados máxima es 38 y la escala de resultados máxima es 37.

  Si el tipo de datos resultante supera los límites de AWS Clean Rooms precisión y escala, la consulta devuelve un error.
- En el caso de las operaciones de conjunto, las dos filas se tratan como idénticas si, para cada par de columnas correspondiente, los dos valores de datos son iguales o NULL. Por ejemplo, si las tablas T1 y T2 contienen una columna y una fila, y esa fila es NULL en ambas tablas, una operación INTERSECT sobre esas tablas devuelve esa fila.

Ejemplo de consultas UNION

En la siguiente consulta UNION, las filas de la tabla SALES se fusionan con las filas de la tabla LISTING. Se seleccionan tres columnas compatibles de cada tabla. En este caso, las columnas correspondientes tienen los mismos nombres y tipos de datos.

```
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
```

En el siguiente ejemplo, se muestra cómo puede agregar un valor literal para el resultado de una consulta UNION para ver cuál expresión de consulta produjo cada fila en el conjunto de resultados. La consulta identifica filas de la primera expresión de consulta como "B" (por compradores, "buyers" en inglés) y filas de la segunda expresión de consulta como "S" (por vendedores, "sellers" en inglés).

La consulta identifica compradores y vendedores para transacciones de ticket que cuestan $10 000 o más. La única diferencia entre las dos expresiones de consulta de cualquier lado del operador UNION es la columna de combinación para la tabla SALES.

```
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
```

En el siguiente ejemplo, se utiliza un operador UNION ALL porque las filas duplicadas, si se encuentran, deben conservarse en el resultado. Para una serie de eventos específica IDs, la consulta devuelve 0 o más filas por cada venta asociada a cada evento y 0 o 1 fila por cada anuncio de ese

evento. IDs Los eventos son únicos para cada fila de las tablas LISTING y EVENT, pero es posible que haya varias ventas para la misma combinación de evento y anuncio IDs en la tabla VENTAS.

La tercera columna en el conjunto de resultados identifica la fuente de la fila. Si viene de la tabla SALES, se marca "Sí" en la columna SALESROW. (SALESROW es un alias para SALES.LISTID). Si la fila proviene de la tabla LISTING, se marca "No" en la columna SALESROW.

En este caso, el conjunto de resultados consta de tres filas de ventas para la lista 500, evento 7787. En otras palabras, se llevaron a cabo tres transacciones diferentes para esta combinación de lista y evento. Los otros dos anuncios, 501 y 502, no generaron ventas, por lo que la única fila que la consulta genera para estas listas IDs proviene de la tabla de anuncios (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
```

Si ejecuta la misma consulta sin la palabra clave ALL, el resultado conserva solo una de las transacciones de ventas.

```
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
```

Ejemplo de consultas UNION ALL

En el siguiente ejemplo, se utiliza un operador UNION ALL porque las filas duplicadas, si se encuentran, deben conservarse en el resultado. Para una serie de eventos específica IDs, la consulta devuelve 0 o más filas por cada venta asociada a cada evento y 0 o 1 fila por cada anuncio de ese evento. IDs Los eventos son únicos para cada fila de las tablas LISTING y EVENT, pero es posible que haya varias ventas para la misma combinación de evento y anuncio IDs en la tabla VENTAS.

La tercera columna en el conjunto de resultados identifica la fuente de la fila. Si viene de la tabla SALES, se marca "Sí" en la columna SALESROW. (SALESROW es un alias para SALES.LISTID). Si la fila proviene de la tabla LISTING, se marca "No" en la columna SALESROW.

En este caso, el conjunto de resultados consta de tres filas de ventas para la lista 500, evento 7787. En otras palabras, se llevaron a cabo tres transacciones diferentes para esta combinación de lista y evento. Los otros dos anuncios, 501 y 502, no generaron ventas, por lo que la única fila que la consulta genera para estas listas IDs proviene de la tabla de anuncios (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
```

Si ejecuta la misma consulta sin la palabra clave ALL, el resultado conserva solo una de las transacciones de ventas.

```
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
```

Ejemplo de consultas INTERSECT

Compare el siguiente ejemplo con el primer ejemplo de UNION. La única diferencia entre los dos ejemplos es el operador de conjunto que se utiliza, pero los resultados son muy diferentes. Solo una de las filas es igual:

```
235494 |    23875 |    8771
```

Esta es la única fila en el resultado limitado de 5 filas que se encontró en ambas tablas.

```
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
```

La siguiente consulta busca eventos (para los que se vendieron tickets) que ocurrieron en lugares en la Ciudad de Nueva York y Los Ángeles en marzo. La diferencia entre las dos expresiones de consulta es la restricción en la columna VENUECITY.

```
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
```

Ejemplo de consulta EXCEPT

La tabla CATEGORY de la base de datos contiene las siguientes 11 filas:

```
 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)
```

Supongamos que una tabla CATEGORY_STAGE (una tabla provisional) contiene una fila adicional:

```
 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)
```

Devuelve la diferencia entre las dos tablas. En otras palabras, devuelve filas que están en la tabla CATEGORY_STAGE pero no en la tabla CATEGORY:

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

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

La siguiente consulta equivalente usa el sinónimo MINUS.

```
select * from category_stage
minus
select * from category;

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

```
(1 row)
```

Si revierte el orden de las expresiones SELECT, la consulta no devuelve filas.

Cláusula ORDER BY

La cláusula ORDER BY ordena el conjunto de resultados de una consulta.

> **Note**
>
> La expresión ORDER BY más externa solo debe tener columnas que estén en la lista de selección.

Temas

- Sintaxis
- Parameters
- Notas de uso
- Ejemplos con ORDER BY

Sintaxis

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

Parameters

expression

Un marco que especifica el orden de clasificación de los resultados de las consultas. Consta de una o más columnas en la lista de selección. Los resultados se devuelven en función de la ordenación UTF-8 binaria. También puede especificar lo siguiente:

- Números ordinales que representan la posición de las entradas de la lista de selección (o la posición de columnas en la tabla si no existe una lista de selección)
- Alias que definen las entradas de la lista de selección

Cuando la cláusula ORDER BY contiene varias expresiones, el conjunto de resultados se ordena según la primera expresión, luego se aplica la segunda expresión a las filas que tienen valores coincidentes de la primera expresión, etc.

ASC | DESC

Opción que define el orden de ordenación para la expresión, de la siguiente manera:

- ASC: ascendente (por ejemplo, de menor a mayor para valores numéricos y de la A a la Z para cadenas con caracteres). Si no se especifica ninguna opción, los datos se ordenan, de manera predeterminada, en orden ascendente.
- DESC: descendente (de mayor a menor para valores numéricos y de la Z a la A para cadenas).

NULLS FIRST | NULLS LAST

Opción que especifica si los valores NULL se deben ordenar en primer lugar, antes de los valores no nulos, o al final, después de los valores no nulos. De manera predeterminada, los valores NULL se ordenan y clasificación al final en orden ASC, y se ordenan y se clasifican primero en orden DESC.

LIMIT number (número) | ALL

Opción que controla la cantidad de filas ordenadas que una consulta devuelve. El número LIMIT deber ser un entero positivo; el valor máximo es 2147483647.

LIMIT 0 no devuelve filas. Puede usar la sintaxis para realizar pruebas: para verificar que una consulta se ejecuta (sin mostrar filas) o para devolver una lista de columnas de una tabla. Una cláusula ORDER BY es redundante si está utilizando LIMIT 0 para devolver una lista de columnas. El predeterminado es LIMIT ALL.

OFFSET start (inicio)

Opción que especifica que se omita el número de filas que hay delante de start (inicio) antes de comenzar a devolver filas. El número OFFSET deber ser un entero positivo; el valor máximo es 2147483647. Cuando se utiliza con la opción LIMIT, las filas OFFSET se omiten antes de comenzar a contar las filas LIMIT que se devuelven. Si no se utiliza la opción LIMIT, la cantidad de filas del conjunto de resultados se reduce por la cantidad de filas que se omiten. Las filas omitidas por una cláusula OFFSET aún deben analizarse, por lo que puede ser ineficiente utilizar un valor OFFSET grande.

Notas de uso

Tenga en cuenta el siguiente comportamiento esperado con las cláusulas ORDER BY:

- Los valores NULL son considerados "superiores" a todos los otros valores. Con el orden ascendente predeterminado, los valores NULL se ordenan al final. Para cambiar este comportamiento, utilice la opción NULLS FIRST.
- Cuando una consulta no contiene una cláusula ORDER BY, el sistema devuelve conjuntos de resultados sin un orden predecible de las filas. Si se ejecuta la misma consulta dos veces, puede devolver el conjunto de resultados en un orden diferente.
- Las opciones LIMIT y OFFSET pueden utilizarse sin una cláusula ORDER BY; no obstante, para devolver un conjunto consistente de filas, use estas opciones junto con ORDER BY.
- En cualquier sistema paralelo AWS Clean Rooms, por ejemplo, cuando ORDER BY no produce un orden único, el orden de las filas no es determinista. Es decir, si la expresión ORDER BY produce valores duplicados, el orden de retorno de esas filas puede variar de un sistema a otro o de una serie AWS Clean Rooms a otra.
- AWS Clean Rooms no admite cadenas literales en las cláusulas ORDER BY.

Ejemplos con ORDER BY

Devuelva todas las 11 filas de la tabla CATEGORY, ordenadas por la segunda columna, CATGROUP. Para los resultados que tienen el mismo valor CATGROUP, ordene los valores de la columna CATDESC por la longitud de la cadena de caracteres. Ordene, a continuación, por columna CATID y CATNAME.

```
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)
```

Devuelva las columnas seleccionadas de la tabla SALES, ordenadas por los valores QTYSOLD más altos. Limite el resultado a las primeras 10 filas:

```
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
```

Devuelve una lista de columnas y ninguna fila a través de la sintaxis LIMIT 0:

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

Ejemplos de subconsultas

En los siguientes ejemplos se muestran diferentes maneras en que las subconsultas encajan en las consultas SELECT. Para ver otro ejemplo del uso de las subconsultas, consulte Ejemplo.

Subconsulta de la lista SELECT

En el siguiente ejemplo, se observa una subconsulta en la lista SELECT. Esta subconsulta es escalar: devuelve solamente una columna y un valor, que se repite en el resultado de cada fila que se devuelve desde la consulta externa. La consulta compara el valor Q1SALES que la subconsulta computa con valores de venta de otros dos trimestres (2 y 3) en 2008, como la consulta externa lo define.

```
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)
```

Subconsulta de la cláusula WHERE

En el siguiente ejemplo, se observa una subconsulta de tabla en la cláusula WHERE. Esta subconsulta produce varias filas. En este caso, las filas contienen solo una columna, pero las subconsultas de la tabla pueden contener varias columnas y filas, como cualquier otra tabla.

La consulta busca los principales 10 vendedores en términos de cantidad máxima de tickets vendidos. La lista de los 10 principales está limitada por la subconsulta, que elimina usuarios que viven en ciudades donde hay lugares de venta de tickets. Esta consulta puede escribirse en diferentes maneras; por ejemplo, se puede volver a escribir la subconsulta como una combinación dentro de la consulta principal.

```
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)
```

Subconsultas de la cláusula WITH

Consulte Cláusula WITH.

Subconsultas correlacionadas

En el siguiente ejemplo, se observa una subconsulta correlacionada en la cláusula WHERE; este tipo de subconsulta contiene una o varias correlaciones entre sus columnas y las columnas producidas por la consulta externa. En este caso, la correlación es `where s.listid=l.listid`. Para cada fila que la consulta externa produce, se ejecuta la subconsulta para calificar o descalificar la fila.

```
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)
```

Patrones de subconsultas correlacionadas que no se admiten

El planificador de consultas usa un método de reescritura de consulta denominado descorrelación de subconsulta para optimizar varios patrones de subconsultas correlacionadas para la ejecución en un entorno MPP. Algunos tipos de subconsultas correlacionadas siguen patrones que no AWS Clean Rooms pueden decorrelacionarse ni son compatibles. Las consultas que contienen las siguientes referencias de correlación devuelven errores:

- Referencias de correlación que omiten un bloque de consulta, también conocidas como "referencias de correlación con nivel omitido". Por ejemplo, en la siguiente consulta, el bloque que contiene la referencia de correlación y el bloque omitido están conectados por un predicado NOT EXISTS:

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

  En este caso, el bloque omitido es la subconsulta que se ejecuta contra la tabla LISTING. La referencia de correlación correlaciona las tablas EVENT y SALES.

- Referencias de correlación de una subconsulta que es parte de una cláusula ON en una consulta externa:

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

  La cláusula ON contiene una referencia de correlación de SALES en la subconsulta a EVENT en la consulta externa.

- Referencias de correlación sensibles a valores nulos a una tabla del sistema. AWS Clean Rooms Por ejemplo:

```
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);
```

- Referencias de correlación de una subconsulta que contiene una función de ventana.

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

- Referencias en una columna GROUP BY a los resultados de una subconsulta correlacionada. Por ejemplo:

```
select listing.listid,
(select count (sales.listid) from sales where sales.listid=listing.listid) as list
from listing
group by list, listing.listid;
```

- Referencias de correlación de una subconsulta con una función agregada y una cláusula GROUP BY, conectadas a la consulta externa por un predicado IN. (Esta restricción no se aplica a las funciones agregadas MIN y MAX). Por ejemplo:

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

AWS Clean Rooms Funciones de Spark SQL

AWS Clean Rooms Spark SQL admite las siguientes funciones de SQL:

Temas

- Funciones de cadena
- Funciones relacionadas con la privacidad
- Funciones de ventana

Funciones de agregación

Las funciones agregadas de AWS Clean Rooms Spark SQL se utilizan para realizar cálculos u operaciones en un grupo de filas y devolver un único valor. Son esenciales para las tareas de análisis y resumen de datos.

AWS Clean Rooms Spark SQL admite las siguientes funciones de agregación:

Temas

- Función ANY_VALUE
- Función APPROX COUNT_DISTINCT
- Función APROX. PERCENTIL
- Función de AVG
- Función BOOL_AND
- Función BOOL_OR
- Función CARDINALIDAD
- Función COLLECT_LIST
- Función COLLECT_SET
- Funciones COUNT y COUNT DISTINCT
- Función COUNT
- Función MAX
- Función MEDIAN
- Función MIN
- Función PERCENTIL
- Función de ASIMETRÍA
- Funciones STDDEV_SAMP y STDDEV_POP
- Funciones SUM y SUM DISTINCT
- Funciones VAR_SAMP y VAR_POP

Función ANY_VALUE

La función ANY_VALUE devuelve cualquier valor de los valores de expresión de entrada de una manera que no sea determinista. Esta función puede devolver un valor NULL si el resultado de la expresión de entrada no implica que se devuelva ninguna fila.

Sintaxis

```
ANY_VALUE (expression[, isIgnoreNull] )
```

Argumentos

expression

La columna o la expresión de destino en la que opera la función. La expresión corresponde a uno de los siguientes tipos de datos:

isIgnoreNull

Un booleano que determina si la función debe devolver únicamente valores no nulos.

Devuelve

Devuelve el mismo tipo de datos que expresión.

Notas de uso

Si una instrucción que especifica la función ANY_VALUE para una columna también incluye una segunda referencia de columna, la segunda columna debe aparecer en una cláusula GROUP BY o debe incluirse en una función de agrupación.

Ejemplos

El siguiente ejemplo devuelve una instancia de cualquier `dateid` donde `eventname` es `Eagles`.

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

A continuación, se muestran los resultados.

```
dateid | eventname
-------+---------------
```

```
 1878  | Eagles
```

El siguiente ejemplo devuelve una instancia de cualquier `dateid` donde `eventname` es `Eagles` o `Cold War Kids`.

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

A continuación, se muestran los resultados.

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

Función APPROX COUNT_DISTINCT

APPROX COUNT_DISTINCT proporciona una forma eficaz de estimar el número de valores únicos en una columna o conjunto de datos.

Sintaxis

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

Argumentos

expr

La expresión o columna para la que desea estimar el número de valores únicos.

Puede ser una sola columna, una expresión compleja o una combinación de columnas.

Relativo D.

Parámetro opcional que especifica la desviación estándar relativa deseada de la estimación.

Es un valor entre 0 y 1, que representa el error relativo máximo aceptable de la estimación. Un valor de RelativeSD más pequeño dará como resultado una estimación más precisa pero más lenta.

Si no se proporciona este parámetro, se utiliza un valor predeterminado (normalmente alrededor del 0,05 o el 5%).

Devuelve

Devuelve la cardinalidad estimada en HyperLogLog ++. RelativeSD define la desviación estándar relativa máxima permitida.

Ejemplo

La siguiente consulta estima el número de valores únicos de la `col1` columna, con una desviación estándar relativa del 1% (0,01).

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

La siguiente consulta estima que hay 3 valores únicos en la `col1` columna (los valores 1, 2 y 3).

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

Función APROX. PERCENTIL

El PERCENTIL APROXIMADO se usa para estimar el valor percentil de una expresión o columna determinada sin tener que ordenar todo el conjunto de datos. Esta función resulta útil en situaciones en las que es necesario comprender rápidamente la distribución de un conjunto de datos grande o realizar un seguimiento de las métricas basadas en percentiles, sin la sobrecarga computacional que supone realizar un cálculo de percentil exacto. Sin embargo, es importante entender las ventajas y desventajas entre velocidad y precisión, y elegir la tolerancia de errores adecuada en función de los requisitos específicos de cada caso de uso.

Sintaxis

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

Argumentos

expr

La expresión o columna para la que desea estimar el valor del percentil.

Puede ser una sola columna, una expresión compleja o una combinación de columnas.

percentil

El valor percentil que desea estimar, expresado como un valor entre 0 y 1.

Por ejemplo, 0,5 correspondería al percentil 50 (mediana).

precisión

Parámetro opcional que especifica la precisión deseada de la estimación del percentil. Es un valor entre 0 y 1, que representa el error relativo máximo aceptable de la estimación. Un `accuracy` valor menor dará como resultado una estimación más precisa pero más lenta. Si no se proporciona este parámetro, se utiliza un valor predeterminado (normalmente alrededor del 0,05 o el 5%).

Devuelve

Devuelve el percentil aproximado de la columna de intervalo numérico o ANSI col, que es el valor más pequeño de los valores de columna ordenados (ordenados de menor a mayor), de modo que no más del porcentaje de valores de col sea inferior o igual a ese valor.

El valor del porcentaje debe estar comprendido entre 0,0 y 1,0. El parámetro de precisión (predeterminado: 10000) es un literal numérico positivo que controla la precisión de la aproximación a costa de la memoria.

Un valor de precisión más alto produce una mejor precisión, `1.0/accuracy` es el error relativo de la aproximación.

Cuando el porcentaje es una matriz, cada valor de la matriz porcentual debe estar entre 0.0 y 1.0. En este caso, devuelve la matriz de percentiles aproximada de la columna col en la matriz de porcentajes dada.

Ejemplos

La siguiente consulta estima el percentil 95 de la `response_time` columna, con un error relativo máximo del 1% (0,01).

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

La siguiente consulta estima los valores de los percentiles 50, 40 y 10 de la columna de la tabla. `col tab`

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

La siguiente consulta estima el percentil 50 (mediana) de los valores de la columna col.

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

Función de AVG

La función AVG devuelve el promedio (media aritmética) de los valores de la expresión de entrada. La función AVG funciona con valores numéricos e ignora los valores NULL.

Sintaxis

```
AVG (column)
```

Argumentos

column

La columna de destino sobre la que opera la función. La columna corresponde a uno de los siguientes tipos de datos:

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

Tipos de datos

Los tipos de argumentos que admite la función AVG son SMALLINT INTEGERBIGINT,DECIMAL, yDOUBLE.

Los tipos de retorno que admite la función AVG son los siguientes:

- BIGINT para cualquier argumento de tipo entero
- DOUBLE para un argumento de punto flotante
- Devuelve el mismo tipo de datos como expresión para cualquier otro tipo de argumento

La precisión predeterminada para un resultado de la función AVG con un argumento DECIMAL de es 38. La escala del resultado es la misma que la escala del argumento. Por ejemplo, una AVG de una columna DEC(5,2) devuelve un tipo de datos DEC(38,2).

Ejemplo

Encontrar la cantidad promedio vendida por transacción en la tabla SALES:

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

Función BOOL_AND

La función BOOL_AND funciona en una sola columna o expresión con valores booleanos o enteros. Esta función aplica una lógica similar a las funciones BIT_AND y BIT_OR. Para esta función, el tipo de retorno es un valor booleano (`true` o `false`).

Si todos los valores de un conjunto son verdaderos, la función BOOL_AND devuelve `true` (`t`). Si todo valor es falso, la función devuelve `false` (`f`).

Sintaxis

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

Argumentos

expression

La columna o expresión de destino sobre la que opera la función. Esta expresión debe tener un tipo de datos booleano o entero. El tipo de retorno de la función es booleano.

DISTINCT | ALL

Con el argumento DISTINCT, la función elimina todos los valores duplicados para la expresión especificada antes de calcular el resultado. Con el argumento ALL, la función retiene todos los valores duplicados. El valor predeterminado es ALL.

Ejemplos

Puede utilizar funciones booleanas con expresiones booleanas o expresiones enteras.

Por ejemplo, la siguiente consulta devuelve resultados de la tabla estándar USERS en la base de datos TICKIT, que tiene varias columnas con valores booleanos.

La función BOOL_AND devuelve `false` para las cinco filas. A no todos los usuarios en cada uno de esos estados les gusta deportes.

```
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)
```

Función BOOL_OR

La función BOOL_OR funciona en una única columna o expresión booleana o entera. Esta función aplica una lógica similar a las funciones BIT_AND y BIT_OR. Para esta función, el tipo de retorno es un valor booleano (`true`, `false` o `NULL`).

Si un valor en un conjunto es `true`, la función BOOL_OR devuelve `true` (`t`). Si un valor en un conjunto es `false`, la función devuelve `false` (`f`). Se puede devolver NULL si se desconoce el valor.

Sintaxis

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

Argumentos

expression

La columna o expresión de destino sobre la que opera la función. Esta expresión debe tener un tipo de datos booleano o entero. El tipo de retorno de la función es booleano.

DISTINCT | ALL

Con el argumento DISTINCT, la función elimina todos los valores duplicados para la expresión especificada antes de calcular el resultado. Con el argumento ALL, la función retiene todos los valores duplicados. El valor predeterminado es ALL.

Ejemplos

Puede utilizar las funciones booleanas con expresiones booleanas o expresiones enteras. Por ejemplo, la siguiente consulta devuelve resultados de la tabla estándar USERS en la base de datos TICKIT, que tiene varias columnas con valores booleanos.

La función BOOL_OR devuelve `true` para las cinco filas. A al menos un usuario en cada uno de esos estados les gusta deportes.

```
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)
```

El ejemplo siguiente devuelve NULL.

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

Función CARDINALIDAD

La función CARDINALIDAD devuelve el tamaño de una expresión ARRAY o MAP (expr).

Esta función es útil para encontrar el tamaño o la longitud de una matriz.

Sintaxis

```
cardinality(expr)
```

Argumentos

expr

Expresión matricial o MAP.

Devuelve

Devuelve el tamaño de una matriz o un mapa (INTEGER).

La función devuelve `NULL` una entrada nula si `sizeOfNull` se establece en `false` o `enabled` se establece en`true`.

De lo contrario, la función devuelve `-1` una entrada nula. Con la configuración predeterminada, la función vuelve `-1` para una entrada nula.

Ejemplo

La siguiente consulta calcula la cardinalidad, o el número de elementos, de la matriz dada. La matriz (`'b', 'd', 'c', 'a'`) tiene 4 elementos, por lo que el resultado de esta consulta sería4.

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

Función COLLECT_LIST

La función COLLECT_LIST recopila y devuelve una lista de elementos no únicos.

Este tipo de función resulta útil cuando se desean recopilar varios valores de un conjunto de filas en una única estructura de datos de matriz o lista.

> **Note**
>
> La función no es determinista porque el orden de los resultados recopilados depende del orden de las filas, que puede no ser determinista tras realizar una operación de barajado.

Sintaxis

```
collect_list(expr)
```

Argumentos

expr

Expresión de cualquier tipo.

Devuelve

Devuelve un ARRAY del tipo argumento. El orden de los elementos de la matriz no es determinista.

Se excluyen los valores NULL.

Si se especifica DISTINCT, la función recopila solo valores únicos y es sinónimo de función `collect_set` agregada.

Ejemplo

La siguiente consulta recopila todos los valores de la columna col en una lista. La `VALUES` cláusula se utiliza para crear una tabla en línea con tres filas, donde cada fila tiene una columna única con los valores 1, 2 y 1, respectivamente. Luego, la `collect_list()` función se usa para agregar todos los valores de la columna col en una sola matriz. El resultado de esta sentencia SQL sería la matriz`[1,2,1]`, que contiene todos los valores de la columna col en el orden en que aparecen en los datos de entrada.

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

Función COLLECT_SET

La función COLLECT_SET recopila y devuelve un conjunto de elementos únicos.

Esta función resulta útil cuando se desean recopilar todos los valores distintos de un conjunto de filas en una sola estructura de datos, sin incluir ningún duplicado.

> **Note**
>
> La función no es determinista porque el orden de los resultados recopilados depende del orden de las filas, que puede no ser determinista tras realizar una operación de barajado.

Sintaxis

```
collect_set(expr)
```

Argumentos

expr

Expresión de cualquier tipo excepto MAP.

Devuelve

Devuelve un ARRAY del tipo argumento. El orden de los elementos de la matriz no es determinista.

Se excluyen los valores NULL.

Ejemplo

La siguiente consulta recopila todos los valores únicos de la columna col en un conjunto. La `VALUES` cláusula se utiliza para crear una tabla en línea con tres filas, donde cada fila tiene una columna única con los valores 1, 2 y 1, respectivamente. Luego, la `collect_set()` función se usa para agregar todos los valores únicos de la columna col en un solo conjunto. El resultado de esta sentencia SQL sería el conjunto`[1,2]`, que contiene los valores únicos de la columna col. El valor duplicado de 1 solo se incluye una vez en el resultado.

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

Funciones COUNT y COUNT DISTINCT

La función COUNT cuenta las filas definidas por la expresión. La función COUNT DISTINCT calcula el número de valores que no son NULL diferentes en una columna o expresión. Elimina todos los valores duplicados de la expresión especificada antes de realizar el recuento.

Sintaxis

```
COUNT (DISTINCT column)
```

Argumentos

column

La columna de destino sobre la que opera la función.

Tipos de datos

La función COUNT y la función COUNT DISTINCT admite todos los tipos de datos de argumentos.

La función COUNT DISTINCT devuelve BIGINT.

Ejemplos

Cuente todos los usuarios del estado de Florida.

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

Cuenta todos los espacios únicos IDs desde la EVENT mesa.

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

Función COUNT

La función COUNT cuenta las filas definidas por la expresión.

La función COUNT tiene las siguientes variaciones.

- COUNT (*) cuenta todas las filas en la tabla destino, incluya o no valores nulos.
- COUNT (expresión) calcula el número de filas con valores no NULL de una determinada columna o expresión.
- COUNT (DISTINCT expresión) calcula el número de valores no NULL diferentes de una columna o expresión.

Sintaxis

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

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

Argumentos

expression

La columna o expresión de destino sobre la que opera la función. La función COUNT admite todos los tipos de datos de argumentos.

DISTINCT | ALL

Con el argumento DISTINCT, la función elimina todos los valores duplicados de la expresión especificada antes de hacer el conteo. Con el argumento ALL, la función retiene todos los valores duplicados de la expresión para el conteo. El valor predeterminado es ALL.

Tipo de devolución

La función COUNT devuelve BIGINT.

Ejemplos

Cuente todos los usuarios del estado de Florida:

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

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

Cuenta todos los nombres de evento de la tabla EVENT:

```
select count(eventname) from event;

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

Cuenta todos los nombres de evento de la tabla EVENT:

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

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

Cuenta todos los lugares únicos IDs de la tabla de EVENTOS:

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

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

Contar la cantidad de veces que cada vendedor indicó lotes de más de cuatro tickets para venta. Agrupar los resultados según ID de vendedor:

```
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
...
```

Función MAX

La función MAX devuelve el valor máximo en un conjunto de filas. Es posible utilizar DISTINCT o ALL pero no influye en el resultado.

Sintaxis

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

Argumentos

expression

La columna o expresión de destino sobre la que opera la función. La expresión es cualquier tipo de dato numérico.

DISTINCT | ALL

Con el argumento DISTINCT, la función elimina todos los valores duplicados de la expresión especificada antes de calcular el máximo. Con el argumento ALL, la función retiene todos los valores duplicados de la expresión especificada para calcular el máximo. El valor predeterminado es ALL.

Tipos de datos

Devuelve el mismo tipo de datos que expresión.

Ejemplos

Encontrar el precio más alto pagado de todas las ventas:

```
select max(pricepaid) from sales;

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

Encontrar el precio más alto pagado por ticket de todas las ventas:

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

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

Función MEDIAN

Sintaxis

```
MEDIAN ( median_expression )
```

Argumentos

expresión_de_mediana

La columna o expresión de destino sobre la que opera la función.

Función MIN

La función MIN devuelve el valor mínimo en un conjunto de filas. Es posible utilizar DISTINCT o ALL pero no influye en el resultado.

Sintaxis

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

Argumentos

expression

La columna o expresión de destino sobre la que opera la función. La expresión es cualquier tipo de datos numéricos.

DISTINCT | ALL

Con el argumento DISTINCT, la función elimina todos los valores duplicados de la expresión especificada antes de calcular el mínimo. Con el argumento ALL, la función retiene todos los valores duplicados de la expresión especificada para calcular el mínimo. El valor predeterminado es ALL.

Tipos de datos

Devuelve el mismo tipo de datos que expresión.

Ejemplos

Encontrar el precio más bajo pagado de todas las ventas:

```
select min(pricepaid) from sales;

min
-------
```

```
20.00
(1 row)
```

Encontrar el precio más bajo pagado por ticket de todas las ventas:

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

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

Función PERCENTIL

La función PERCENTIL se utiliza para calcular el valor percentil exacto ordenando primero los valores de la `col` columna y, a continuación, buscando el valor en el valor especificado. `percentage`

La función PERCENTIL es útil cuando necesita calcular el valor percentil exacto y el coste computacional es aceptable para su caso de uso. Proporciona resultados más precisos que la función APPROX_PERCENTILE, pero puede ser más lenta, especialmente para conjuntos de datos grandes.

Por el contrario, la función APPROX_PERCENTILE es una alternativa más eficiente que puede proporcionar una estimación del valor del percentil con una tolerancia de error específica, lo que la hace más adecuada para escenarios en los que la velocidad es una prioridad mayor que la precisión absoluta.

Sintaxis

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

Argumentos

col

La expresión o columna para la que desea calcular el valor del percentil.

porcentaje

El valor percentil que desea calcular, expresado como un valor entre 0 y 1.

Por ejemplo, 0,5 correspondería al percentil 50 (mediana).

frecuencia

Parámetro opcional que especifica la frecuencia o el peso de cada valor de la `col` columna. Si se proporciona, la función calculará el percentil en función de la frecuencia de cada valor.

Devuelve

Devuelve el valor percentil exacto de la columna de intervalo numérico o ANSI col en el porcentaje indicado.

El valor del porcentaje debe estar comprendido entre 0,0 y 1,0.

El valor de la frecuencia debe ser una integral positiva

Ejemplo

La siguiente consulta busca un valor mayor o igual al 30% de los valores de la `col` columna. Como los valores son 0 y 10, el percentil 30 es 3,0, porque es el valor que es mayor o igual al 30% de los datos.

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

Función de ASIMETRÍA

La función ASIMETRÍA devuelve el valor de asimetría calculado a partir de los valores de un grupo.

La asimetría es una medida estadística que describe la asimetría o la falta de simetría en un conjunto de datos. Proporciona información sobre la forma de la distribución de los datos.

Esta función puede resultar útil para comprender las propiedades estadísticas de un conjunto de datos y servir de base para futuros análisis o para la toma de decisiones.

Sintaxis

```
skewness(expr)
```

Argumentos

expr

Expresión que se evalúa como un valor numérico.

Devuelve

Devuelve DOUBLE.

Si se especifica DISTINCT, la función solo funciona con un conjunto único de valores de expr.

Ejemplos

La siguiente consulta calcula la asimetría de los valores de la columna. `col` En este ejemplo, la `VALUES` cláusula se usa para crear una tabla en línea con cuatro filas, donde cada fila tiene una sola columna `col` con los valores -10, -20, 100 y 1000. A continuación, la `skewness()` función se utiliza para calcular la asimetría de los valores de la columna. `col` El resultado, 1,1135657469022011, representa el grado y la dirección de la asimetría de los datos. Un valor de asimetría positivo indica que los datos están sesgados hacia la derecha y que la mayoría de los valores se concentran en el lado izquierdo de la distribución. Un valor de asimetría negativo indica que los datos están sesgados hacia la izquierda y que la mayoría de los valores se concentran en el lado derecho de la distribución.

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

La siguiente consulta calcula la asimetría de los valores de la columna col. Al igual que en el ejemplo anterior, la `VALUES` cláusula se usa para crear una tabla en línea con cuatro filas, donde cada fila tiene una sola columna `col` con los valores -1000, -100, 10 y 20. A continuación, la `skewness()` función se utiliza para calcular la asimetría de los valores de la columna. `col` El resultado, -1.1135657469022011, representa el grado y la dirección de la asimetría en los datos. En este caso, el valor de asimetría negativo indica que los datos están sesgados hacia la izquierda y que la mayoría de los valores se concentran en el lado derecho de la distribución.

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

Funciones STDDEV_SAMP y STDDEV_POP

Las funciones STDDEV_SAMP y STDDEV_POP devuelven la muestra y la desviación estándar de población de un conjunto de valores numéricos (entero, decimal o de punto flotante). El resultado de la función STDDEV_SAMP es equivalente a la raíz cuadrada de la varianza de muestra del mismo conjunto de valores.

STDDEV_SAMP y STDDEV son sinónimos para la misma función.

Sintaxis

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

La expresión debe tener un tipo de datos numérico. Independientemente del tipo de datos de la expresión, el tipo de retorno de esta función es un número de doble precisión.

> **Note**
>
> La desviación estándar se calcula utilizando aritmética de punto flotante, que puede dar como resultado una leve imprecisión.

Notas de uso

Cuando la desviación estándar de la muestra (STDDEV o STDDEV_SAMP) se calcula para una expresión que consta de un valor único, el resultado de la función es NULL no 0.

Ejemplos

La siguiente consulta devuelve el promedio de valores en la columna VENUESEATS de la tabla VENUE, seguido de la desviación estándar de la muestra y la desviación estándar de la población del mismo conjunto de valores. VENUESEATS es una columna INTEGER. La escala del resultado se reduce a 2 dígitos.

```
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)
```

La siguiente consulta devuelve la desviación estándar de muestra para la columna COMMISSION en la tabla SALES. COMMISSION es una columna DECIMAL. La escala del resultado se reduce a 10 dígitos.

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

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

La siguiente consulta convierte la desviación estándar de muestra para la columna COMMISSION en un número entero.

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

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

La siguiente consulta devuelve tanto la desviación estándar de muestra y la raíz cuadrada de la varianza de muestra para la columna COMMISSION. Los resultados de estos cálculos son semejantes.

```
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)
```

Funciones SUM y SUM DISTINCT

La función SUM devuelve la suma de la columna de entrada o valores de la expresión. La función SUM funciona con valores numéricos e ignora los valores NULL.

La función SUM DISTINCT elimina todos los valores duplicados de la expresión especificada antes de calcular la suma.

Sintaxis

```
SUM (DISTINCT column )
```

Argumentos

column

La columna de destino sobre la que opera la función. La columna es cualquier tipo de datos numéricos.

Ejemplos

Encontrar la suma de todas las comisiones pagadas de la tabla SALES:

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

Encontrar la suma de todas las comisiones diferenciadas pagadas de la tabla SALES:

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

Funciones VAR_SAMP y VAR_POP

Las funciones VAR_SAMP y VAR_POP devuelven la muestra y la varianza de población de un conjunto de valores numéricos (entero, decimal o de punto flotante). El resultado de la función VAR_SAMP es equivalente a la desviación cuadrada estándar de la muestra del mismo conjunto de valores.

VAR_SAMP y VARIANCE son sinónimos para la misma función.

Sintaxis

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

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

La expresión debe ser un tipo de datos entero, decimal o de punto flotante. Independientemente del tipo de datos de la expresión, el tipo de retorno de esta función es un número de doble precisión.

> **Note**
>
> Los resultados de estas funciones pueden variar entre clústeres de data warehouse, según la configuración del clúster en cada caso.

Notas de uso

Cuando la varianza de la muestra (VARIANCE o VAR_SAMP) se calcula para una expresión que consta de un valor único, el resultado de la función es NULL no 0.

Ejemplos

La siguiente consulta devuelve la varianza redondeada de muestra y población de la columna NUMTICKETS en la tabla LISTING.

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

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

La siguiente consulta ejecuta los mismos cálculos pero convierte los resultados a valores decimales.

```
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)
```

Funciones de matriz

En esta sección se describen las funciones de matriz de SQL admitidas en AWS Clean Rooms.

Temas

Función ARRAY

Crea una matriz con los elementos dados.

Sintaxis

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

Argumento

expr1, expr2

Expresiones de cualquier tipo de datos, excepto los tipos de datos de fecha y hora. Los argumentos no tienen que ser del mismo tipo de datos.

Tipo de retorno

La función de matriz devuelve una MATRIZ con los elementos de la expresión.

Ejemplo

El siguiente ejemplos muestra una matriz de valores numéricos y una matriz de diferentes tipos de datos.

```
--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)
```

Función ARRAY_CONTAINS

La función ARRAY_CONTAINS se puede utilizar para realizar comprobaciones básicas de pertenencia en estructuras de datos de matrices. La función ARRAY_CONTAINS es útil cuando se necesita comprobar si un valor específico está presente en una matriz.

Sintaxis

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

Argumentos

array

Un ARRAY que se va a buscar.

value

Una expresión con un tipo que comparte un tipo menos común con los elementos de la matriz.

Tipo de retorno

La función ARRAY_CONTAINS devuelve un BOOLEANO.

Si el valor es NULL, el resultado es NULL.

Si algún elemento de la matriz es NULL, el resultado es NULL si el valor no coincide con ningún otro elemento.

Ejemplos

El siguiente ejemplo comprueba si la matriz `[1, 2, 3]` contiene el valor4. Como la matriz`[1, 2, 3]` no contiene el valor4, devuelve la función array_contains. `false`

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

En el siguiente ejemplo, se comprueba si la matriz `[1, 2, 3]` contiene el valor. 2 Como la matriz `[1, 2, 3]` contiene el valor2, la función array_contains devuelve el valor. `true`

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

Función ARRAY_DISTINCT

La función ARRAY_DISTINCT se puede usar para eliminar valores duplicados de una matriz. La función ARRAY_DISTINCT es útil cuando necesita eliminar los duplicados de una matriz y trabajar solo con los elementos únicos. Esto puede resultar útil en situaciones en las que desee realizar operaciones o análisis en un conjunto de datos sin la interferencia de valores repetidos.

Sintaxis

```
array_distinct(array)
```

Argumentos

array

Una expresión matricial.

Tipo de retorno

La función ARRAY_DISTINCT devuelve una MATRIZ que contiene solo los elementos únicos de la matriz de entrada.

Ejemplos

En este ejemplo, la matriz de entrada `[1, 2, 3, null, 3]` contiene un valor duplicado de. `3` La `array_distinct` función elimina este valor duplicado 3 y devuelve una nueva matriz con los elementos únicos:`[1, 2, 3, null]`.

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

En este ejemplo, la matriz de entrada `[1, 2, 2, 3, 3, 3]` contiene valores duplicados de 2 y3. La `array_distinct` función elimina estos duplicados y devuelve una nueva matriz con los elementos únicos:`[1, 2, 3]`.

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

Función ARRAY_EXCEPT

La función ARRAY_EXCEPT toma dos matrices como argumentos y devuelve una nueva matriz que contiene solo los elementos que están presentes en la primera matriz, pero no en la segunda.

La ARRAY_EXCEPT es útil cuando se necesitan encontrar los elementos que son exclusivos de una matriz en comparación con otra. Esto puede resultar útil en situaciones en las que es necesario realizar operaciones similares a las de un conjunto en matrices, como encontrar la diferencia entre dos conjuntos de datos.

Sintaxis

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

Argumentos

matriz1

Un ARRAY de cualquier tipo con elementos comparables.

matriz (2)

Un ARRAY de elementos que comparten un tipo menos común con los elementos de array1.

Tipo de retorno

La función ARRAY_EXCEPT devuelve una MATRIZ del tipo coincidente con la matriz 1 sin duplicados.

Ejemplos

En este ejemplo, la primera matriz `[1, 2, 3]` contiene los elementos 1, 2 y 3. La segunda matriz `[2, 3, 4]` contiene los elementos 2, 3 y 4. La `array_except` función elimina los elementos 2 y 3 de la primera matriz, ya que también están presentes en la segunda matriz. La salida resultante es la matriz`[1]`.

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

En este ejemplo, la primera matriz `[1, 2, 3]` contiene los elementos 1, 2 y 3. La segunda matriz `[1, 3, 5]` contiene los elementos 1, 3 y 5. La `array_except` función elimina los elementos 1 y 3 de la primera matriz, ya que también están presentes en la segunda matriz. La salida resultante es la matriz`[2]`.

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

Función ARRAY_INTERSECT

La función ARRAY_INTERSECT toma dos matrices como argumentos y devuelve una nueva matriz que contiene los elementos que están presentes en ambas matrices de entrada. Esta función resulta útil cuando se necesitan encontrar los elementos comunes entre dos matrices. Esto puede resultar útil en situaciones en las que es necesario realizar operaciones similares a las de un conjunto en matrices, como encontrar la intersección entre dos conjuntos de datos.

Sintaxis

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

Argumentos

matriz1

Un ARRAY de cualquier tipo con elementos comparables.

matriz (2)

Un ARRAY de elementos que comparten un tipo menos común con los elementos de array1.

Tipo de retorno

La función ARRAY_INTERSECT devuelve un ARRAY del tipo coincidente con el de matriz1, sin duplicados y con elementos contenidos tanto en matriz1 como en matriz2.

Ejemplos

En este ejemplo, la primera matriz contiene los elementos 1, 2 y 3. `[1, 2, 3]` La segunda matriz `[1, 3, 5]` contiene los elementos 1, 3 y 5. La función ARRAY_INTERSECT identifica los elementos comunes entre las dos matrices, que son 1 y 3. La matriz de salida resultante es. `[1, 3]`

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

Función ARRAY_JOIN

La función ARRAY_JOIN utiliza dos argumentos: el primer argumento es la matriz de entrada que se unirá. El segundo argumento es la cadena separadora que se utilizará para concatenar los elementos de la matriz. Esta función resulta útil cuando se necesita convertir una matriz de cadenas (o cualquier otro tipo de datos) en una sola cadena concatenada. Esto puede resultar útil en situaciones en las que desee presentar una matriz de valores como una sola cadena con formato, por ejemplo, con fines de visualización o para su uso en un procesamiento posterior.

Sintaxis

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

Argumentos

array

Cualquier tipo de matriz, pero sus elementos se interpretan como cadenas.

delimiter

Una CADENA que se utiliza para separar los elementos de la matriz concatenados.

Reemplazo nulo

Cadena que se utiliza para expresar un valor NULO en el resultado.

Tipo de retorno

La función ARRAY_JOIN devuelve una cadena en la que los elementos de la matriz se separan mediante un delimitador y se sustituyen por elementos nulos. `nullReplacement` Si `nullReplacement` se omite, `null` los elementos se filtran. Si hay algún argumento`NULL`, el resultado es`NULL`.

Ejemplos

En este ejemplo, la función ARRAY_JOIN toma la matriz `['hello', 'world']` y une los elementos mediante el separador `' '` (un carácter de espacio). El resultado es la cadena. `'hello world'`

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

En este ejemplo, la función ARRAY_JOIN toma la matriz `['hello', null, 'world']` y une los elementos mediante el separador `' '` (un carácter de espacio). El `null` valor se sustituye por la cadena de sustitución proporcionada `','` (una coma). El resultado es la cadena`'hello , world'`.

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

Función ARRAY_REMOVE

La función ARRAY_REMOVE utiliza dos argumentos: el primer argumento es la matriz de entrada de la que se eliminarán los elementos. El segundo argumento es el valor que se eliminará de la matriz. Esta función es útil cuando se necesitan eliminar elementos específicos de una matriz. Esto puede resultar útil en situaciones en las que es necesario realizar una limpieza de datos o un preprocesamiento de una matriz de valores.

Sintaxis

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

Argumentos

array

Un ARRAY.

element

Una expresión de un tipo que comparte un tipo menos común con los elementos de una matriz.

Tipo de retorno

La función ARRAY_REMOVE devuelve el tipo de resultado que coincide con el tipo de matriz. Si el elemento que se va a eliminar es`NULL`, el resultado es. `NULL`

Ejemplos

En este ejemplo, la función ARRAY_REMOVE toma la matriz `[1, 2, 3, null, 3]` y elimina todas las apariciones del valor 3. La salida resultante es la matriz. `[1, 2, null]`

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

Función ARRAY_UNION

La función ARRAY_UNION toma dos matrices como argumentos y devuelve una nueva matriz que contiene los elementos únicos de ambas matrices de entrada. Esta función resulta útil cuando se necesitan combinar dos matrices y eliminar cualquier elemento duplicado. Esto puede resultar útil en situaciones en las que es necesario realizar operaciones similares a las de un conjunto en matrices, como encontrar la unión entre dos conjuntos de datos.

Sintaxis

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

Argumentos

matriz1

Un ARRAY.

Matriz 2

Un ARRAY del mismo tipo que array1.

Tipo de retorno

La función ARRAY_UNION devuelve una MATRIZ del mismo tipo que una matriz.

Ejemplo

En este ejemplo, la primera matriz `[1, 2, 3]` contiene los elementos 1, 2 y 3. La segunda matriz `[1, 3, 5]` contiene los elementos 1, 3 y 5. La función ARRAY_UNION combina los elementos únicos de ambas matrices, lo que da como resultado la matriz de salida. `[1, 2, 3, 5]` T

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

Función EXPLODE

La función EXPLODE se utiliza para transformar una sola fila con una matriz o columna de mapa en varias filas, donde cada fila corresponde a un único elemento de la matriz o el mapa.

Sintaxis

```
explode(expr)
```

Argumentos

expr

Una expresión matricial o una expresión de mapa.

Tipo de retorno

La función EXPLODE devuelve un conjunto de filas, donde cada fila representa un único elemento de la matriz o mapa de entrada.

El tipo de datos de las filas de salida depende del tipo de datos de los elementos de la matriz o el mapa de entrada.

Ejemplos

El siguiente ejemplo toma la matriz de una sola fila [10, 20] y la transforma en dos filas independientes, cada una de las cuales contiene uno de los elementos de la matriz (10 y 20).

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

En el primer ejemplo, la matriz de entrada se pasó directamente como argumento a`explode()`. En este ejemplo, la matriz de entrada se especifica mediante la => sintaxis, donde el nombre de la columna (`collection`) se proporciona de forma explícita.

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

Ambos enfoques son válidos y permiten obtener el mismo resultado, pero la segunda sintaxis puede resultar más útil cuando se necesita desglosar una columna de un conjunto de datos más grande, en lugar de limitarse a un simple literal de matriz.

Función FLATTEN

La función FLATTEN se utiliza para «aplanar» una estructura de matriz anidada en una sola matriz plana.

Sintaxis

```
flatten(arrayOfArrays)
```

Argumentos

arrayOfArrays

Matriz de matrices.

Tipo de retorno

La función FLATTEN devuelve una matriz.

Ejemplo

En este ejemplo, la entrada es una matriz anidada con dos matrices internas y la salida es una matriz plana única que contiene todos los elementos de las matrices internas. La función FLATTEN toma la

matriz anidada `[[1, 2], [3, 4]]` y combina todos los elementos en una sola matriz. `[1, 2, 3, 4]`

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

Expresiones condicionales

En SQL, las expresiones condicionales se utilizan para tomar decisiones en función de determinadas condiciones. Permiten controlar el flujo de las sentencias SQL y devolver valores diferentes o realizar diferentes acciones en función de la evaluación de una o más condiciones.

AWS Clean Rooms admite las siguientes expresiones condicionales:

Temas

- Expresión condicional CASE
- expresión COALESCE
- Expresión máxima y mínima
- Expresión IF
- Expresión IS_NULL
- Expresión IS_NOT_NULL
- Funciones NVL y COALESCE
- NVL2 función
- Función NULLIF

Expresión condicional CASE

La expresión CASE es una expresión condicional, similar a if/then/else las sentencias que se encuentran en otros lenguajes. CASE se utiliza para especificar un resultado cuando hay condiciones múltiples. Utilice CASE cuando una expresión SQL sea válida, como en un comando SELECT.

Existen dos tipos de expresiones CASE: simple y buscada.

- En expresiones CASE simples, una expresión se compara con un valor. Cuando hay una coincidencia, se aplica la acción especificada en la cláusula THEN. Si no se encuentra coincidencia, se aplica la acción en la cláusula ELSE.

- En las expresiones CASE buscadas, cada CASE se evalúa según una expresión booleana, y la instrucción CASE devuelve el primer CASE que coincida. Si no hay ninguna coincidencia entre las cláusulas WHEN, se devuelve la acción en la cláusula ELSE.

Sintaxis

Instrucción CASE simple utilizada para hacer coincidir condiciones:

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

Instrucción CASE buscada utilizada para evaluar cada condición:

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

Argumentos

expresión

Un nombre de columna o cualquier expresión válida.

value

Valor con el que se compara la expresión, como una constante numérica o una cadena de caracteres.

result

El valor destino o la expresión que se devuelve cuando se evalúa una expresión o una condición booleana. Los tipos de datos de todas las expresiones de resultado deben poder convertirse a un único tipo de salida.

condition

Expresión booleana que se evalúa como true o false. Si el argumento condition es verdadero, el valor de la expresión CASE es el resultado que sigue a la condición y el resto de la expresión

CASE no se procesa. Si el argumento condition es falso, se evalúan las cláusulas WHEN subsiguientes. Si ningún resultado de la condición WHEN es verdadero, el valor de la expresión CASE será el resultado de la cláusula ELSE. Si se omite la cláusula ELSE y ninguna condición es verdadera, el resultado será nulo.

Ejemplos

Use una expresión CASE simple para reemplazar `New York City` por `Big Apple` en una consulta de la tabla VENUE. Reemplace todos los demás nombres de ciudad por `other`.

```
select venuecity,
  case venuecity
    when 'New York City'
    then 'Big Apple' else 'other'
  end
from venue
order by venueid desc;

venuecity        |   case
-----------------+-----------
Los Angeles      | other
New York City    | Big Apple
San Francisco    | other
Baltimore        | other
...
```

Utilice una expresión CASE buscada para asignar números de grupo según el valor PRICEPAID para ventas de tickets individuales:

```
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
...
```

expresión COALESCE

Una expresión COALESCE devuelve el valor de la primera expresión en la lista que no sea nulo. Si todas las expresiones son nulas, el resultado es nulo. Cuando se encuentra un valor no nulo, las expresiones restantes de la lista no se evalúan.

Este tipo de expresión es útil cuando desea devolver un valor de backup para algo cuando no hay un valor preferido o si este es nulo. Por ejemplo, una consulta puede devolver uno de tres números telefónicos (celular, hogar o trabajo, en ese orden), sea cual sea que encuentre primero en la tabla (no nulo).

Sintaxis

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

Ejemplos

Aplica la expresión COALESCE a dos columnas.

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

El nombre de columna predeterminado de una expresión NVL es COALESCE. La siguiente consulta devuelve los mismos resultados.

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

Expresión máxima y mínima

Devuelve el valor más grande o el más pequeño de una lista de cualquier cantidad de expresiones.

Sintaxis

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

Parámetros

expression_list

Una lista de expresiones separada por comas, como la columna nombres. Las expresiones deben ser todas convertibles a un tipo común de datos. Se ignoran los valores NULL en la lista. Si todas las expresiones toman el valor NULL, el resultado es NULL.

Devuelve

Devuelve el valor máximo (para GREATEST) o el mínimo (para LEAST) de la lista de expresiones proporcionada.

Ejemplo

El siguiente ejemplo devuelve el valor más alto alfabéticamente para `firstname` o `lastname`.

```
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)
```

Expresión IF

La función condicional IF devuelve uno de los dos valores en función de una condición.

Esta función es una sentencia de flujo de control común que se utiliza en SQL para tomar decisiones y devolver diferentes valores en función de la evaluación de una condición. Resulta útil para implementar una lógica simple de tipo if-else en una consulta.

Sintaxis

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

Argumentos

expr1

La condición o expresión que se evalúa. Si es así`true`, la función devolverá el valor de expr2. Si expr1 es*`false`*, la función devolverá el valor de expr3.

expr2

La expresión que se evalúa y devuelve si expr1 es. `true`

expr3

La expresión que se evalúa y devuelve si expr1 es. `false`

Devuelve

Si se `expr1` evalúa como, devuelve`expr2`; de `true` lo contrario, devuelve. `expr3`

Ejemplo

En el siguiente ejemplo, se utiliza la `if()` función para devolver uno de los dos valores en función de una condición. La condición que se está `1 < 2` evaluando es`true`, es decir, `'a'` se devuelve el primer valor.

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

Expresión IS_NULL

La expresión IS_NULL condicional se usa para comprobar si un valor es nulo.

Esta expresión es sinónimo deIS NULL.

Sintaxis

```
is_null(expr)
```

Argumentos

expr

Una expresión de cualquier tipo.

Devuelve

La expresión IS_NULL condicional devuelve un booleano. Si `expr1` es NULL, devuelve; de lo contrario`true`, devuelve. `false`

Ejemplos

El siguiente ejemplo comprueba si el valor 1 es nulo y devuelve el resultado booleano `true` porque 1 es un valor válido y no nulo.

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

En el siguiente ejemplo, se selecciona la `id` columna de la `squirrels` tabla, pero solo para las filas en las que se encuentra la columna de edad. `null`

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

Expresión IS_NOT_NULL

La expresión IS_NOT_NULL condicional se usa para comprobar si un valor no es nulo.

Esta expresión es sinónimo deIS NOT NULL.

Sintaxis

```
is_not_null(expr)
```

Argumentos

expr

Una expresión de cualquier tipo.

Devuelve

La expresión IS_NOT_NULL condicional devuelve un booleano. Si no `expr1` es NULL, devuelve; de lo contrario`true`, devuelve. `false`

Ejemplos

El siguiente ejemplo comprueba si el valor no 1 es nulo y devuelve el resultado booleano `true` porque 1 es un valor válido y no nulo.

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

En el siguiente ejemplo, se selecciona la `id` columna de la `squirrels` tabla, pero solo para las filas en las que no aparece la columna de edad. `null`

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

Funciones NVL y COALESCE

Devuelve el valor de la primera expresión que no es nula en una serie de expresiones. Cuando se encuentra un valor que no es nulo, las expresiones restantes de la lista no se evalúan.

NVL es idéntica a COALESCE. Son sinónimos. En este tema se explica la sintaxis y se incluyen ejemplos de ambas funciones.

Sintaxis

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

La sintaxis de COALESCE es la misma:

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

Si todas las expresiones son nulas, el resultado es nulo.

Estas funciones son útiles cuando se desea devolver un valor secundario si falta un valor primario o es nulo. Por ejemplo, una consulta puede devolver el primero de los tres números de teléfono disponibles: móvil, fijo o trabajo. El orden de las expresiones de la función determina el orden de evaluación.

Argumentos

expresión

Una expresión, como un nombre de columna, que evalúa estados nulos.

Tipo de devolución

AWS Clean Rooms determina el tipo de datos del valor devuelto en función de las expresiones de entrada. Si los tipos de datos de las expresiones de entrada no tienen un tipo común, se devuelve un error.

Ejemplos

Si la lista contiene expresiones de enteros, la función devuelve un entero.

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

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

Este ejemplo, que es igual al anterior, excepto que usa NVL, devuelve el mismo resultado.

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

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

En el siguiente ejemplo, se devuelve un tipo de cadena.

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

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

En el siguiente ejemplo, se produce un error porque los tipos de datos varían en la lista de expresiones. En este caso, hay un tipo de cadena y un tipo de número en la lista.

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

NVL2 función

Devuelve uno de los dos valores, en función de si una expresión especificada toma un valor NULL o NOT NULL.

Sintaxis

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

Argumentos

expresión

Una expresión, como un nombre de columna, que evalúa estados nulos.

not_null_return_value

El valor devuelto si la expression (expresión) toma un valor NOT NULL. El valor not_null_return_value debe tener los mismos tipos de datos que expression (expresión) o ser convertible implícitamente a ese tipo de datos.

null_return_value

El valor de retorno si expression (expresión) toma un valor NULL. El valor null_return_value debe tener los mismos tipos de datos que expression (expresión) o ser convertible implícitamente a ese tipo de datos.

Tipo de devolución

El tipo de NVL2 devolución se determina de la siguiente manera:

- Si alguno de los valores not_null_return_value o null_return_value es nulo, se devuelve el tipo de datos de la expresión no nula.

Si ninguno de los valores not_null_return_value y null_return_value es nulo:

- Si los valores not_null_return_value y null_return_value tienen el mismo tipo de datos, se devuelve ese tipo de datos.

- Si los valores not_null_return_value y null_return_value tienen tipos de datos numéricos diferentes, se devuelve el tipo de dato numérico compatible que sea menor.
- Si los valores not_null_return_value y null_return_value tienen tipos de datos de fecha y hora diferentes, se devuelve un tipo de dato de marca temporal.
- Si los valores not_null_return_value y null_return_value tienen tipos de datos de caracteres diferentes, se devuelve el tipo de dato de not_null_return_value.
- Si los valores not_null_return_value y null_return_value tienen tipos de datos numéricos y no numéricos mezclados, se devuelve el tipo de dato de not_null_return_value.

> **Important**
>
> En los últimos dos casos en los que se devuelve el tipo de dato not_null_return_value, null_return_value está vinculado implícitamente a ese tipo de dato. Si los tipos de datos son incompatibles, la función falla.

Notas de uso

En efecto NVL2, la devolución tendrá el valor del parámetro not_null_return_value o null_return_value, según lo que seleccione la función, pero tendrá el tipo de datos not_null_return_value.

Por ejemplo, si se asume que column1 es NULL, las siguientes consultas devolverán el mismo valor. Sin embargo, el tipo de datos del valor devuelto por NVL2 DECODE será INTEGER y el tipo de datos del valor devuelto será VARCHAR.

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

Ejemplo

En el siguiente ejemplo, se modifican algunos datos de muestra y, luego, se evalúan dos campos para proporcionar la información de contacto adecuada para los usuarios:

```
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
```

Función NULLIF

Compara dos argumentos y devuelve un valor nulo si los argumentos son iguales. Si no son iguales, se devuelve el primer argumento.

Sintaxis

La expresión NULLIF compara dos argumentos y devuelve un valor nulo si los argumentos son iguales. Si no son iguales, se devuelve el primer argumento. Esta expresión realiza lo contrario a lo que realiza la expresión NVL o COALESCE.

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

Argumentos

expresión1, expresión2

Las columnas o expresiones de destino que se comparan. El tipo de retorno es el mismo que el tipo de la primera expresión.

Ejemplos

En el ejemplo siguiente, la consulta devuelve la cadena `first` porque los argumentos no son iguales.

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

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

En el ejemplo siguiente, la consulta devuelve NULL porque los argumentos literales de la cadena son iguales.

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

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

En el ejemplo siguiente, la consulta devuelve 1 porque los argumentos de enteros no son iguales.

```
SELECT NULLIF(1, 2);

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

En el ejemplo siguiente, la consulta devuelve NULL porque los argumentos de enteros son iguales.

```
SELECT NULLIF(1, 1);

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

En el siguiente ejemplo, la consulta devuelve valores nulos cuando los valores LISTID y SALESID coinciden:

```
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)
```

Funciones del constructor

Una función constructora de SQL es una función que se utiliza para crear nuevas estructuras de datos, como matrices o mapas.

Toman algunos valores de entrada y devuelven un nuevo objeto de estructura de datos. Las funciones constructoras suelen tener el nombre del tipo de datos que crean, como ARRAY o MAP.

Las funciones constructoras son diferentes de las funciones escalares o agregadas, que funcionan con los datos existentes y devuelven un único valor. Las funciones constructoras se utilizan para crear nuevas estructuras de datos que luego se pueden utilizar en el procesamiento o análisis posterior de los datos.

AWS Clean Rooms admite las siguientes funciones constructoras:

Temas

- función constructora MAP
- Función constructora NAMED_STRUCT
- Función constructora STRUCT

función constructora MAP

La función constructora MAP crea un mapa con los pares clave/valor dados.

Las funciones constructoras como MAP son útiles cuando necesita crear nuevas estructuras de datos mediante programación dentro de sus consultas SQL. Permiten crear estructuras de datos complejas que se pueden utilizar en posteriores procesamientos o análisis de datos.

Sintaxis

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

Argumentos

clave0

Una expresión de cualquier tipo comparable. Todas las key0 deben compartir un tipo mínimo común.

valor0

Una expresión de cualquier tipo. Todos los valores EN deben compartir un tipo mínimo común.

Devuelve

La función MAP devuelve un MAPA con las claves escritas como el tipo menos común de clave0 y los valores escritos como el tipo menos común de valor0.

Ejemplos

El siguiente ejemplo crea un mapa nuevo con dos pares clave-valor: la clave está asociada al valor. `1.0` `'2'` La clave `3.0` está asociada al valor. `'4'` A continuación, el mapa resultante se devuelve como salida de la sentencia SQL.

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

Función constructora NAMED_STRUCT

La función constructora NAMED_STRUCT crea una estructura con los nombres y valores de campo dados.

Las funciones constructoras como NAMED_STRUCT son útiles cuando se necesita crear nuevas estructuras de datos mediante programación en las consultas SQL. Permiten crear estructuras de

datos complejas, como estructuras o registros, que se pueden utilizar en el procesamiento o análisis posterior de los datos.

Sintaxis

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

Argumentos

nombre1

Un campo de nomenclatura literal STRING 1.

val1

Expresión de cualquier tipo que especifique el valor del campo 1.

Devuelve

La función NAMED_STRUCT devuelve una estructura cuyo campo 1 coincide con el tipo de val1.

Ejemplos

En el siguiente ejemplo, se crea una nueva estructura con tres campos con nombre: Se asigna el valor al campo. `"a" 1 "b"`Se asigna el valor al campo. `"c"` Se le asigna `2.` el valor `3` al campo. A continuación, la estructura resultante se devuelve como salida de la sentencia SQL.

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

Función constructora STRUCT

La función constructora STRUCT crea una estructura con los valores de campo dados.

Las funciones constructoras como STRUCT son útiles cuando se necesita crear nuevas estructuras de datos mediante programación dentro de las consultas SQL. Permiten crear estructuras de datos complejas, como estructuras o registros, que se pueden utilizar en el procesamiento o análisis posterior de los datos.

Sintaxis

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

Argumentos

col. 1

Un nombre de columna o cualquier expresión válida.

Devuelve

La función STRUCT devuelve una estructura cuyo campo1 coincide con el tipo de expr1.

Si los argumentos son referencias denominadas, los nombres se utilizan para nombrar el campo. De lo contrario, los campos se denominan COLn, donde N es la posición del campo en la estructura.

Ejemplos

El siguiente ejemplo crea una nueva estructura con tres campos: al primer campo se le asigna el valor 1. Al segundo campo se le asigna el valor 2. Al tercer campo se le asigna el valor 3. De forma predeterminada, los campos de la estructura resultante se denominan`col1`, y `col2``col3`, en función de su posición en la lista de argumentos. A continuación, la estructura resultante se devuelve como salida de la sentencia SQL.

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

Funciones de formato de tipo de datos

El uso de una función de formato de tipos de datos le permite convertir valores de un tipo de datos a otro. En cada una de estas funciones, el primer argumento siempre es el valor al que se va a dar formato, mientras que el segundo argumento contiene la plantilla del formato nuevo.

AWS Clean Rooms Spark SQL admite varias funciones de formato de tipos de datos.

Temas

- BASE64 función
- Función CAST
- Función DECODE
- Función ENCODE
- Función HEX

- Función STR_TO_MAP
- TO_CHAR
- Función TO_DATE
- TO_NUMBER
- UNBASE64 función
- Función UNHEX
- Cadenas de formatos de fecha y hora
- Cadenas de formatos numéricos

BASE64 función

La BASE64 función convierte una expresión en una cadena de base 64 mediante la codificación de transferencia RFC2045 Base64 para MIME.

Sintaxis

```
base64(expr)
```

Argumentos

expr

Una expresión BINARIA o una CADENA que la función interpretará como BINARIA.

Tipo de devolución

`STRING`

Ejemplo

Para convertir la entrada de cadena dada en su representación codificada en Base64, utilice el siguiente ejemplo. El resultado es la representación codificada en Base64 de la cadena de entrada «Spark SQL», que es «U3bhcMsgU1fm».

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

Función CAST

La función CAST convierte un tipo de datos en otro tipo compatible. Por ejemplo, puede convertir una cadena en una fecha o un tipo numérico en una cadena. CAST realiza una conversión en tiempo de ejecución, lo que significa que la conversión no cambia el tipo de datos de un valor en una tabla de origen. Solo cambia en el contexto de la consulta.

Algunos tipos de datos requieren una conversión explícita a otros tipos de datos mediante la función CAST. Otros tipos de datos se pueden convertir implícitamente, como parte de otro comando, sin usar CAST. Consulte Conversión y compatibilidad de tipos.

Sintaxis

Utilice cualquiera de estas dos formas sintácticas equivalentes para convertir expresiones de un tipo de datos a otro.

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

Argumentos

expresión

Una expresión que toma el valor de uno o más valores, como un nombre de columna o un literal. La conversión de valores nulos devuelve valores nulos. La expresión no puede tener cadenas en blanco ni vacías.

type

Uno de los compatiblesTipos de datos, excepto los tipos de datos BINARY y BINARY VARIANT.

Tipo de devolución

CAST devuelve el tipo de datos especificado por el argumento type.

> **Note**
>
> AWS Clean Rooms devuelve un error si intenta realizar una conversión problemática, como una conversión DECIMAL que pierde precisión, como la siguiente:
>
> ```
> select 123.456::decimal(2,1);
> ```

o una conversión a un valor de INTEGER que genera un desbordamiento:

```
select 12345678::smallint;
```

Ejemplos

Las siguientes dos consultas son equivalentes. Ambas convierten un valor decimal en uno entero:

```
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)
```

Lo siguiente produce un resultado similar. No requiere datos de muestra para ejecutarse:

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

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

En este ejemplo, los valores de una columna de marca temporal se convierten en fechas, lo que elimina la hora de cada resultado:

```
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)
```

Si no utilizara CAST como se ilustra en el ejemplo anterior, los resultados incluirían la hora: 2008-02-18 02:36:48.

La siguiente consulta convierte los datos de caracteres variables en una fecha. No requiere datos de muestra para ejecutarse.

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

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

En este ejemplo, los valores en una columna de fecha se convierten en marcas temporales:

```
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)
```

En un caso como el del ejemplo anterior, puede obtener control adicional sobre el formato de salida mediante el uso de TO_CHAR.

En este ejemplo, un valor entero se convierte en una cadena de caracteres:

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

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

En este ejemplo, un valor DECIMAL (6,3) se convierte en un valor DECIMAL (4,1):

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

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

Este ejemplo muestra una expresión más compleja. La columna PRICEPAID (una columna DECIMAL(8,2)) de la tabla SALES se convierte en una columna DECIMAL(38,2) y los valores se multiplican por 100000000000000000000:

```
select salesid, pricepaid::decimal(38,2)*100000000000000000000
as value from sales where salesid<10 order by salesid;


 salesid |           value
---------+----------------------------
       1 | 72800000000000000000000.00
       2 |  76000000000000000000000.00
       3 | 35000000000000000000000.00
       4 | 17500000000000000000000.00
       5 | 15400000000000000000000.00
       6 | 39400000000000000000000.00
```

```
       7 | 78800000000000000000.00
       8 | 19700000000000000000.00
       9 | 59100000000000000000.00

(9 rows)
```

Función DECODE

La función DECODE es la contraparte de la función ENCODE, que se utiliza para convertir una cadena a un formato binario mediante una codificación de caracteres específica. La función DECODE toma los datos binarios y los vuelve a convertir a un formato de cadena legible mediante la codificación de caracteres especificada.

Esta función resulta útil cuando necesita trabajar con datos binarios almacenados en una base de datos y debe presentarlos en un formato legible para las personas, o cuando necesita convertir datos entre diferentes codificaciones de caracteres.

Sintaxis

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

Argumentos

expr

Expresión binaria codificada en charset.

juego de caracteres

Una expresión de cadena.

Codificaciones de juegos de caracteres compatibles (no distinguen entre mayúsculas y minúsculas):`'US-ASCII'`,`'ISO-8859-1'`, `'UTF-8''UTF-16BE'`, `'UTF-16LE'` y. `'UTF-16'`

Tipo de devolución

La función DECODE devuelve una CADENA.

Ejemplo

El siguiente ejemplo tiene una tabla llamada `messages` con una columna denominada `message_text` que almacena los datos de los mensajes en formato binario mediante la codificación

de caracteres UTF-8. La función DECODE convierte los datos binarios a un formato de cadena legible. El resultado de esta consulta es el texto legible del mensaje almacenado en la tabla de mensajes, con el ID123, convertido del formato binario a una cadena mediante la `'utf-8'` codificación.

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

Función ENCODE

La función ENCODE se utiliza para convertir una cadena en su representación binaria mediante una codificación de caracteres específica.

Esta función resulta útil cuando se necesita trabajar con datos binarios o cuando se necesita convertir entre diferentes codificaciones de caracteres. Por ejemplo, puede utilizar la función ENCODE cuando almacene datos en una base de datos que requiera almacenamiento binario o cuando necesite transferir datos entre sistemas que utilizan codificaciones de caracteres diferentes.

Sintaxis

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

Argumentos

str

Una expresión STRING que se va a codificar.

juego de caracteres

Una expresión STRING que especifica la codificación.

Codificaciones de juegos de caracteres compatibles (no distinguen mayúsculas de minúsculas):`'US-ASCII'`,`'ISO-8859-1'`, `'UTF-8''UTF-16BE'`, `'UTF-16LE'` y. `'UTF-16'`

Tipo de devolución

La función ENCODE devuelve un binario.

Ejemplo

El siguiente ejemplo convierte la cadena `'abc'` en su representación binaria mediante la `'utf-8'` codificación, lo que en este caso hace que se devuelva la cadena original. Esto se debe a que la `'utf-8'` codificación es una codificación de caracteres de ancho variable que puede representar todo el conjunto de caracteres ASCII (que incluye las letras `'a'` y`'c'`) utilizando un solo byte por carácter. `'b'` Por lo tanto, la representación binaria del `'abc'` uso `'utf-8'` es la misma que la de la cadena original.

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

Función HEX

La función HEX convierte un valor numérico (ya sea un número entero o un número de punto flotante) en su correspondiente representación de cadena hexadecimal.

El hexadecimal es un sistema numérico que utiliza 16 símbolos distintos (0-9 y A-F) para representar valores numéricos. Se utiliza habitualmente en informática y programación para representar datos binarios en un formato más compacto y legible para las personas.

Sintaxis

```
hex(expr)
```

Argumentos

expr

Expresión BIGINT, BINARIA o STRING.

Tipo de devolución

HEX devuelve una cadena. La función devuelve la representación hexadecimal del argumento.

Ejemplo

El siguiente ejemplo toma el valor entero 17 como entrada y le aplica la función HEX (). La salida es11, que es la representación hexadecimal del valor de entrada17.

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

```
 11
```

El siguiente ejemplo convierte la cadena `'Spark_SQL'` en su representación hexadecimal. El resultado es`537061726B2053514C`, que es la representación hexadecimal de la cadena de entrada`'Spark_SQL'`.

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

En este ejemplo, la cadena 'Spark_SQL' se convierte de la siguiente manera:

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

La concatenación de estos valores hexadecimales da como resultado el resultado final». `537061726B2053514C"`

Función STR_TO_MAP

La función STR_TO_MAP es una función de conversión. string-to-map Convierte una representación en cadena de un mapa (o diccionario) en una estructura de datos cartográfica real.

Esta función resulta útil cuando necesita trabajar con estructuras de datos de mapas en SQL, pero los datos se almacenan inicialmente como una cadena. Al convertir la representación de cadena en un mapa real, puede realizar operaciones y manipulaciones en los datos del mapa.

Sintaxis

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

Argumentos

texto

Una expresión STRING que representa el mapa.

PairDelim

Un literal STRING opcional que especifica cómo separar las entradas. El valor predeterminado es una coma ()`','`.

keyValueDelim

Un literal STRING opcional que especifica cómo separar cada par clave-valor. El valor predeterminado es dos puntos (). `':'`

Tipo de devolución

La función STR_TO_MAP devuelve un MAPA de CADENAS tanto para las claves como para los valores. Tanto PairDelim como yo se tratan como expresiones regulares keyValueDelim.

Ejemplo

El siguiente ejemplo toma la cadena de entrada y los dos argumentos delimitadores y convierte la representación de la cadena en una estructura de datos de mapa real. En este ejemplo específico, la cadena de entrada `'a:1,b:2,c:3'` representa un mapa con los siguientes pares clave-valor: `'a'` es la clave y `'1'` es el valor. `'b'`es la clave y `'2'` es el valor. `'c'`es la clave y `'3'` es el valor. El `','` delimitador se usa para separar los pares clave-valor y el `':'` delimitador se usa para separar la clave y el valor dentro de cada par. El resultado de esta consulta es:. `{"a":"1","b":"2","c":"3"}` Esta es la estructura de datos del mapa resultante, donde las claves están `'a' 'b''c'`, y, y los valores correspondientes son `'1''2'`, y`'3'`.

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

El siguiente ejemplo demuestra que la función STR_TO_MAP espera que la cadena de entrada esté en un formato específico, con los pares clave-valor delimitados correctamente. Si la cadena de entrada no coincide con el formato esperado, la función seguirá intentando crear un mapa, pero es posible que los valores resultantes no sean los esperados.

```
SELECT str_to_map('a');
```

```
{"a":null}
```

TO_CHAR

TO_CHAR convierte una marca temporal o una expresión numérica a un formato de datos de cadena de caracteres.

Sintaxis

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

Argumentos

timestamp_expression

Una expresión que da lugar a un valor de tipo TIMESTAMP o TIMESTAMPTZ, o bien, un valor que se pueda convertir de forma implícita en una marca temporal.

numeric_expression

Una expresión que de como resultado un valor de tipo de datos numérico o un valor que se pueda convertir implícitamente en un tipo numérico. Para obtener más información, consulte Tipos numéricos. TO_CHAR inserta un espacio a la izquierda de la cadena numérica.

> **Note**
>
> TO_CHAR no admite valores DECIMAL de 128 bits.

format

El formato para el valor nuevo. Para conocer los formatos válidos, consulte Cadenas de formatos de fecha y hora y Cadenas de formatos numéricos.

Tipo de devolución

VARCHAR

Ejemplos

En el ejemplo siguiente, se convierte una marca temporal en un valor con la fecha y la hora en un formato con el nombre del mes relleno con nueve caracteres, el nombre del día de la semana y el número de día del mes.

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

En el siguiente ejemplo, se convierte una marca temporal en un valor con el número de día del año.

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

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

En el siguiente ejemplo, se convierte una marca temporal en un número de día de ISO de la semana.

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

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

El siguiente ejemplo extrae el nombre del mes de una fecha.

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

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

En el siguiente ejemplo, se convierte cada valor STARTTIME en la tabla EVENT a una cadena que consta de horas, minutos y segundos.

```
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)
```

En el siguiente ejemplo, se convierte un valor completo de marca temporal a un formato diferente.

```
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)
```

En el siguiente ejemplo, se convierte un literal de marca temporal a una cadena de caracteres.

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

En el siguiente ejemplo se convierte un número a una cadena de caracteres con el signo negativo al final.

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

En el siguiente ejemplo se convierte un número a una cadena de caracteres con el símbolo de moneda.

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

```
to_char
---------
$-125.88
(1 row)
```

En el siguiente ejemplo, se convierte un número a una cadena de caracteres con corchetes angulares para números negativos.

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

En el siguiente ejemplo se convierte un número a una cadena de números romanos.

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

En el ejemplo siguiente se muestra el día de la semana.

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

En el ejemplo siguiente se muestra el sufijo de número ordinal de un número.

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

En el siguiente ejemplo, se resta la comisión del precio pagado en la tabla de ventas. La diferencia, luego, se redondea hacia arriba y se convierte en un número romano, que se muestra en la columna `to_char`:

```
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)
```

En el siguiente ejemplo, se agrega el símbolo de la moneda a los valores de diferencia que se muestran en la columna `to_char`:

```
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)
```

En el siguiente ejemplo, se indica el siglo en el que se realizó la venta.

```
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)
```

En el siguiente ejemplo, se convierte cada valor STARTTIME en la tabla EVENT en una cadena que consta de horas, minutos, segundos y zona horaria.

```
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)
```

En el siguiente ejemplo, se muestra el formato para segundos, milisegundos y microsegundos.

```
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
```

Función TO_DATE

TO_DATE convierte una fecha que se representa con una cadena de caracteres en un tipo de datos DATE.

Sintaxis

```
TO_DATE (date_str)
```

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

Argumentos

date_str

Una cadena de fecha o un tipo de datos que se puede convertir en una cadena de fecha.

format

Un literal de cadena que coincide con los patrones de fecha y hora de Spark. Para ver patrones de fecha y hora válidos, consulta Patrones de fecha y hora para formatear y analizar.

Tipo de devolución

TO_DATE devuelve un valor DATE, en función del valor de format.

Si la conversión a formato produce un error, se devuelve un error.

Ejemplos

La siguiente instrucción SQL convierte la fecha `02 Oct 2001` a un tipo de datos de fecha.

```
select to_date('02 Oct 2001', 'dd MMM yyyy');

to_date
------------
```

```
2001-10-02
(1 row)
```

La siguiente instrucción SQL convierte la cadena `20010631` en una fecha.

```
select to_date('20010631', 'yyyyMMdd');
```

La siguiente instrucción SQL convierte la cadena `20010631` en una fecha:

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

El resultado es un valor nulo porque solo hay 30 días en junio.

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

TO_NUMBER

TO_NUMBER convierte una cadena en un valor numérico (decimal).

Sintaxis

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

Argumentos

string

Cadena que se convertirá. El formato debe ser un valor literal.

format

El segundo argumento es una cadena de formato que indica cómo se debe analizar la cadena original para crear el valor numérico. Por ejemplo, el formato `'99D999'` especifica que la cadena que se convertirá consta de cinco dígitos con el punto decimal en la tercera posición. Por ejemplo, `to_number('12.345','99D999')` devuelve `12.345` como un valor numérico. Para obtener una lista de formatos válidos, consulte Cadenas de formatos numéricos.

Tipo de devolución

TO_NUMBER devuelve un número DECIMAL.

Si la conversión a formato produce un error, se devuelve un error.

Ejemplos

En el siguiente ejemplo, se convierte la cadena `12,454.8-` a un número:

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

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

En el siguiente ejemplo, se convierte la cadena `$ 12,454.88` a un número:

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

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

En el siguiente ejemplo, se convierte la cadena `$ 2,012,454.88` a un número:

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

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

UNBASE64 función

La UNBASE64 función convierte un argumento de una cadena de base 64 a una cadena binaria.

La codificación Base64 se suele utilizar para representar datos binarios (como imágenes, archivos o información cifrada) en un formato textual que sea seguro para su transmisión a través de varios canales de comunicación (como el correo electrónico, los parámetros de URL o el almacenamiento de bases de datos).

La UNBASE64 función permite invertir este proceso y recuperar los datos binarios originales. Este tipo de funcionalidad puede resultar útil en situaciones en las que necesite trabajar con datos codificados en formato Base64, como cuando se integran con sistemas externos o APIs cuando se utiliza Base64 como mecanismo de transferencia de datos.

Sintaxis

```
unbase64(expr)
```

Argumentos

expr

Expresión STRING en formato base64.

Tipo de devolución

`BINARY`

Ejemplo

En el ejemplo siguiente, la cadena codificada en base64 `'U3BhcmsgU1FM'` se convierte de nuevo en la cadena original. `'Spark SQL'`

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

Función UNHEX

La función UNHEX convierte una cadena hexadecimal a su representación de cadena original.

Esta función puede resultar útil en situaciones en las que necesite trabajar con datos que se hayan almacenado o transmitido en formato hexadecimal y necesite restaurar la representación de cadena original para su posterior procesamiento o visualización.

La función UNHEX es la contraparte de la función HEX.

Sintaxis

```
unhex(expr)
```

Argumentos

expr

Expresión de cadena de caracteres hexadecimales.

Tipo de devolución

UNHEX devuelve un binario.

Si la longitud de expr es impar, el primer carácter se descarta y el resultado se rellena con un byte nulo. Si expr contiene caracteres que no son hexadecimales, el resultado es nulo.

Ejemplo

El siguiente ejemplo convierte una cadena hexadecimal a su representación de cadena original mediante las funciones UNHEX () y DECODE () juntas. En la primera parte de la consulta, se utiliza la función UNHEX () para convertir la cadena hexadecimal '537061726B2053514C' en su representación binaria. En la segunda parte de la consulta, se utiliza la función DECODE () para volver a convertir los datos binarios obtenidos de la función UNHEX () en una cadena, mediante la codificación de caracteres «UTF-8». El resultado de la consulta es la cadena original, «Spark_SQL», que se convirtió a hexadecimal y, después, se volvió a convertir en cadena.

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

Cadenas de formatos de fecha y hora

Puede utilizar patrones de fecha y hora en los siguientes escenarios comunes:

- Cuando se trabaja con fuentes de datos CSV y JSON para analizar y formatear contenido de fecha y hora
- Al convertir entre tipos de cadenas y tipos de fecha o marca horaria mediante funciones como:
  - unix_timestamp
  - date_format
  - a _unix_timestamp
  - from_unixtime
  - to_date

- to_timestamp
- from_utc_timestamp
- a _utc_timestamp

Utilice las letras del patrón de la siguiente tabla para analizar y formatear la fecha y la marca de tiempo.

Partes de fecha o de hora	Significado	Ejemplos
a	A la mañana o a la tarde del día, presentadas como a.m. o p.m.	PM
D	Día del año, presentado como un número de 3 dígitos	189
d	Día del mes, presentado como un número de 2 dígitos	28
E	Día de la semana, presentado como texto	¿Verdad Martes
F	Día de la semana del mes alineado, presentado como un número de 1 dígito	3
G	Indicador de era, presentado como texto	AD Anno Domini
h	La hora del reloj de la mañana o la tarde, presentada como un número de 2 dígitos	12
H	Hora del día, presentada como un número de 2 dígitos del 0 al 23	0

Partes de fecha o de hora	Significado	Ejemplos
k	Hora del reloj del día, presentada como un número de 2 dígitos del 1 al 24	1
K	Hora de la mañana o de la tarde, presentada como un número de 2 dígitos del 0 al 11	0
m	Minuto de la hora, presentado como un número de 2 dígitos	30
M/L	Mes del año, presentado como mes	7 07 Julio julio
O	Desfase de zona localizada con respecto a UTC	GMT+8 GMT+ 8:00 UTC- 08:00
Q/q	Trimestre del año, presentado como número (del 1 al 4) o texto	3 03 Q3 3er trimestre
s	Segundo del minuto, presentado como un número de 2 dígitos	55

Partes de fecha o de hora	Significado	Ejemplos
S	Fracción de segundo, presentada como fracción	978
V	Identificador de zona horaria, presentado como identificador de zona	America/Los_Angeles Z 08:30
x	Desplazamiento de zona con respecto a UTC (offset-X)	+0000 -08 -0830 - 08:30 -083015 - 08:30:15
X	Desfase de zona con respecto a UTC; donde Z es igual a cero	Z -08 -0830 - 08:30 -083015 - 08:30:15
y	Año, presentado como año	2020 20
z	Nombre de la zona horaria, presentado como texto	Hora estándar del Pacífico PASADO

Partes de fecha o de hora	Significado	Ejemplos
Z	Desplazamiento de zona con respecto a UTC (offset-Z)	+0000 -08:00 - 08:00
'	Escape para texto, presentado como delimitador	N/A
''	Comilla simple, presentada en forma literal	'
[	Inicio de sección opcional	N/A
]	Final de sección opcional	N/A

El número de letras del patrón determina el tipo de formato:

Formato de texto

- Utilice de 1 a 3 letras para la forma abreviada (por ejemplo, «Mon» para lunes)
- Use exactamente 4 letras para el formulario completo (por ejemplo, «lunes»)
- No utilices 5 o más letras, ya que se producirá un error

Formato numérico (n)

- El valor n representa el número máximo de letras permitido
- Para patrones de una sola letra:
  - La salida utiliza un mínimo de dígitos sin relleno
- Para patrones de letras múltiples:
  - La salida se rellena con ceros para que coincida con el ancho del recuento de letras
- Al analizar, la entrada debe contener el número exacto de dígitos

Formato de número/texto

- Para 3 o más letras, siga las reglas de formato de texto
- Para menos letras, sigue las reglas de formato numérico

Formato de fracción

- Utilice de 1 a 9 caracteres en forma de «S» (por ejemplo, SSSSSS)
- Para analizar:
  - Acepte fracciones entre 1 y el número de caracteres S
- Para formatear:
  - Rellene con ceros para que coincidan con el número de caracteres S
- Admite hasta 6 dígitos para una precisión de microsegundos
- Puede analizar nanosegundos pero trunca los dígitos adicionales

Formato de año

- El recuento de letras establece el ancho de campo mínimo para el relleno
- Para dos letras:
  - Imprime los dos últimos dígitos
  - Analiza los años entre 2000 y 2099
- Para menos de cuatro letras (excepto dos):
  - Muestra el signo solo para los años negativos
- No utilices 7 o más letras, ya que se producirá un error

Formato de mes

- Use «M» para el formulario estándar o «L» para el formulario independiente
- «M» o «L» simples:
  - Muestra los números de los meses del 1 al 12 sin relleno
- 'MM' o 'LL':
  - Muestra los números de mes del 1 al 12 con relleno
- 'MMM':
  - Muestra el nombre abreviado del mes en formato estándar

    - Debe formar parte de un patrón de fechas completo
- 'LLL':
    - Muestra el nombre abreviado del mes en forma independiente
    - Úselo solo para formatear por meses
- 'MMMM':
    - Muestra el nombre completo del mes en formato estándar
    - Úselo para fechas y marcas de tiempo
- «JAJAJA»:
    - Muestra el nombre completo del mes en formato independiente
    - Úselo para formatear solo por meses

Formatos de zonas horarias

- am-pm: Usa solo una letra
- ID de zona (V): utilice solo 2 letras
- Nombres de zona (z):
    - De 1 a 3 letras: muestra el nombre corto
    - 4 letras: muestra el nombre completo
    - No utilices 5 o más letras

Formatos offset

- X y x:
    - 1 letra: Muestra la hora (+01) o la hora-minuto (+0130)
    - 2 letras: muestra la hora y el minuto sin dos puntos (+0130)
    - 3 letras: muestra la hora y el minuto con dos puntos (+ 01:30)
    - 4 letras: se muestra hour-minute-second sin dos puntos (+013015)
    - 5 letras: se muestra hour-minute-second con dos puntos (+ 01:30:15)
    - X usa 'Z' para el desplazamiento a cero
    - x usa '+00', '+0000' o '+ 00:00 'para el desplazamiento a cero
- O:
    - 1 letra: muestra la forma abreviada (GMT+8)

  - 4 letras: muestra la forma completa (GMT+ 08:00)
- Z:
  - De 1 a 3 letras: muestra la hora y el minuto sin dos puntos (+0130)
  - 4 letras: muestra la forma completa y localizada
  - 5 letras: se muestra hour-minute-second con dos puntos

Secciones opcionales

- Utilice corchetes [] para marcar el contenido opcional
- Puede anidar secciones opcionales
- Todos los datos válidos aparecen en la salida
- La entrada puede omitir secciones opcionales enteras

> **Note**
>
> Los símbolos «E», «F», «q» y «Q» solo funcionan para formatear fecha y hora (como date_format). No los utilices para analizar fechas y horas (como to_timestamp).

Cadenas de formatos numéricos

Las siguientes cadenas de formato numérico se aplican a funciones como TO_NUMBER y TO_CHAR.

- Para ver ejemplos de cómo formatear cadenas como números, consulte TO_NUMBER.
- Para ver ejemplos de cómo formatear números como cadenas, consulte TO_CHAR.

Formato	Description (Descripción)
9	Valor numérico con la cantidad especificada de dígitos.
0	Valor numérico con ceros a la izquierda.
. (period), D	Punto decimal.

Formato	Description (Descripción)
, (coma)	Separador de miles.
CC	Código de siglo. Por ejemplo, el siglo XXI comenzó el 01/01/2001 (compatible solo con TO_CHAR).
FM	Modo de relleno. Suprime espacios de relleno y ceros.
PR	Valor negativo entre paréntesis.
S	Signo anclado a un número.
L	El símbolo de la moneda en la posición especificada.
G	Separador de grupo.
MI	Signo menos en la posición especificada para números menores que 0.
PL	Signo más en la posición especificada para números mayores que 0.
SG	Signo más o menos en la posición especific ada.
RN	Número romano entre 1 y 3999 (compatible solo con TO_CHAR).
TH o th	Sufijo de número ordinal. No convierte fracciones ni valores menores que cero.

Funciones de fecha y hora

Las funciones de fecha y hora le permiten realizar una amplia gama de operaciones con datos de fecha y hora, como extraer partes de una fecha, realizar cálculos de fecha, formatear fechas y horas

y trabajar con la fecha y hora actuales. Estas funciones son esenciales para tareas como el análisis de datos, la elaboración de informes y la manipulación de datos que implican datos temporales.

AWS Clean Rooms admite las siguientes funciones de fecha y hora:

Temas

Función ADD_MONTHS

ADD_MONTHS agrega la cantidad de meses especificada a una expresión o un valor de fecha o marca temporal. La función DATE_ADD ofrece una funcionalidad similar.

Sintaxis

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

Argumentos

date | timestamp

Una columna de marca temporal o fecha o una expresión que, implícitamente, se convierte en una marca temporal o fecha. Si la fecha es el último día del mes, o si el mes resultante es más corto, la función devuelve el último día del mes en el resultado. Para otras fechas, el resultado tiene el mismo número de día que la expresión de fecha.

integer

Un número entero positivo o negativo. Use un número negativo para restar meses de las fechas.

Tipo de devolución

TIMESTAMP

Ejemplo

La siguiente consulta utiliza la función ADD_MONTHS dentro de una función TRUNC. La función TRUNC quita la hora del día del resultado de ADD_MONTHS. La función ADD_MONTHS agrega 12 meses a cada valor de la columna CALDATE.

```
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)
```

En los ejemplos a continuación, se demuestra el comportamiento resultante cuando la función ADD_MONTHS opera sobre fechas con meses que tienen diferente cantidad de días.

```
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)
```

Función CONVERT_TIMEZONE

CONVERT_TIMEZONE convierte una marca temporal de una zona horaria a otra. La función se ajusta automáticamente al horario de verano.

Sintaxis

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

Argumentos

source_timezone

(Opcional) La zona horaria de la marca temporal actual. El valor predeterminado es UTC.

target_timezone

La zona horaria para la marca temporal nueva.

timestamp

Una columna de marca temporal o una expresión que, implícitamente, se convierte en una marca temporal.

Tipo de devolución

TIMESTAMP

Ejemplos

En el siguiente ejemplo, se convierte el valor de la marca temporal de la zona horaria UTC predeterminada a la zona horaria PST.

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

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

En el siguiente ejemplo, el valor de la marca temporal que aparece en la columna LISTTIME se convierte de la zona horaria UTC predeterminada a la zona horaria PST. Aunque la marca temporal se encuentra dentro del periodo de horario de verano, se convierte a horario estándar porque la zona horaria objetivo se especifica como una abreviatura (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
```

El siguiente ejemplo convierte una columna LISTTIME con una marca de tiempo de la zona horaria UTC predeterminada a una zona horaria. US/Pacific La zona horaria objetivo usa un nombre de zona horaria y la marca temporal se encuentra dentro del periodo de horario de verano, por lo que la función devuelve el horario de verano.

```
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
```

En el siguiente ejemplo, se convierte una cadena de marca temporal de EST a PST:

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

 convert_timezone
```

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

En el siguiente ejemplo, se convierte una marca temporal al horario del este de Estados Unidos estándar porque la zona horaria objetivo usa un nombre de zona horaria (America/New_York) y la marca temporal se encuentra dentro del periodo estándar.

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

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

En el siguiente ejemplo, se convierte la marca temporal al horario de verano del este de Estados Unidos porque la zona horaria objetivo usa un nombre de zona horaria (America/New_York) y la marca temporal se encuentra dentro del periodo de horario de verano.

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

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

En el siguiente ejemplo, se demuestra el uso de desplazamientos.

```
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)
```

Función CURRENT_DATE

CURRENT_DATE devuelve una fecha en la zona horaria de la sesión actual (UTC de forma predeterminada) en el formato predeterminado:. YYYY-MM-DD

> **Note**
>
> CURRENT_DATE devuelve la fecha de comienzo de la transacción actual, no de la instrucción actual. Pensemos en el escenario en el que se inicia una transacción con varias instrucciones el 10/01/08 a las 23:59 y la instrucción que contiene CURRENT_DATE se ejecuta el 10/02/08 a las 00:00. CURRENT_DATE devuelve `10/01/08`, no `10/02/08`.

Sintaxis

```
CURRENT_DATE
```

Tipo de devolución

DATE

Ejemplo

El siguiente ejemplo devuelve la fecha actual (en la que Región de AWS se ejecuta la función).

```
select current_date;

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

Función CURRENT_TIMESTAMP

CURRENT_TIMESTAMP devuelve la fecha y la hora actuales, incluidas la fecha, la hora y (opcionalmente) los milisegundos o microsegundos.

Esta función resulta útil cuando se necesita obtener la fecha y la hora actuales, por ejemplo, para registrar la marca de tiempo de un evento, realizar cálculos basados en el tiempo o rellenar columnas. date/time

Sintaxis

```
current_timestamp()
```

Tipo de devolución

La función CURRENT_TIMESTAMP devuelve una FECHA.

Ejemplo

El siguiente ejemplo devuelve la fecha y hora actuales en el momento en que se ejecuta la consulta, es decir, el 25 de abril de 2020 a las 15:49:11 914 (15:49:11 914 p.m.).

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

El siguiente ejemplo recupera la fecha y hora actuales de cada fila de la tabla. `squirrels`

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

Función DATE_ADD

Devuelve la fecha que es num_days después de la fecha de inicio.

Sintaxis

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

Argumentos

fecha_inicio

El valor de la fecha de inicio.

num_days

El número de días que se va a añadir (entero). Un número positivo suma días y un número negativo resta días.

Tipo de devolución

DATE

Ejemplos

En el ejemplo siguiente se suma un día a una fecha:

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

Result:
2016-07-31
```

El siguiente ejemplo agrega varios días.

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

Result:
2016-08-04
```

Notas de uso

Esta documentación es para la función DATE_ADD de Spark SQL, que proporciona una interfaz más sencilla para añadir días a las fechas en comparación con otras variantes de SQL. Para añadir otros intervalos, como meses o años, es posible que se necesiten diferentes funciones.

Función DATE_DIFF

DATE_DIFF devuelve la diferencia entre las partes de fecha de dos expresiones de fecha u hora.

Sintaxis

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

Argumentos

endDate

Una expresión de fecha.

startDate

Una expresión de fecha.

Tipo de devolución

BIGINT

Ejemplos con una columna DATE

En el siguiente ejemplo, se encuentra la diferencia, en cantidad de semanas, entre dos valores de fecha literales.

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

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

En el siguiente ejemplo, se encuentra la diferencia, en horas, entre dos valores de fecha literales. Cuando no se proporciona el valor de la hora para una fecha, de forma predeterminada es 00:00:00.

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

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

En el siguiente ejemplo se encuentra la diferencia, en días, entre dos valores TIMESTAMETZ literales.

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

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

En el siguiente ejemplo, se encuentra la diferencia, en días, entre dos fechas de la misma fila de una tabla.

```
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)
```

En el siguiente ejemplo, se encuentra la diferencia, en cantidad de trimestres, entre un valor literal del pasado y la fecha de hoy. En este ejemplo, se asume que la fecha actual es 5 de junio del 2008. Puede nombrar las partes de la fecha de manera completa o abreviada. El nombre de columna predeterminado de la función DATE_DIFF es DATE_DIFF.

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

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

En este ejemplo, se unen las tablas SALES y LISTING para calcular cuántos días después de indicarse se vendieron los tickets de los listados 1000 a 1005. La espera más prolongada para la venta de estos listados fue de 15 días, y la más corta, de menos de 1 día (0 días).

```
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)
```

En este ejemplo, se calculan las horas promedio esperadas por los vendedores para todas las ventas de tickets.

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

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

Ejemplos con una columna TIME

La siguiente tabla de ejemplo, TIME_TEST, tiene una columna TIME_VAL (tipo TIME) con tres valores insertados.

```
select time_val from time_test;

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

En el siguiente ejemplo, se encuentra la diferencia en cantidad de horas entre la columna TIME_VAL y un literal de tiempo.

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

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

En el siguiente ejemplo, se encuentra la diferencia en cantidad de minutos entre dos valores de tiempo literales.

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

nummins
```

```
----------
60
```

Ejemplos con una columna TIMETZ

La siguiente tabla de ejemplo, TIMETZ_TEST, tiene una columna TIMETZ_VAL (tipo TIMETZ) con tres valores insertados.

```
select timetz_val from timetz_test;

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

En el siguiente ejemplo, se encuentran las diferencias en la cantidad de horas, entre un literal TIMETZ y timetz_val.

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

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

En el siguiente ejemplo, se encuentra la diferencia en cantidad de horas entre dos valores TIMETZ literales.

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

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

Función DATE_PART

DATE_PART extrae los valores de parte de fecha a partir de una expresión. DATE_PART es sinónimo de la función PGDATE_PART.

Sintaxis

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

Argumentos

campo

Qué parte de la fuente debe extraerse y los valores de cadena admitidos son los mismos que los campos de la función equivalente EXTRACT.

origen

Una columna de fecha o intervalo de la que se debe extraer el campo.

Tipo de devolución

Si el campo es «SEGUNDO», un DECIMAL (8, 6). En todos los demás casos, un entero.

Ejemplo

El siguiente ejemplo extrae el día del año (DOY) de un valor de fecha. El resultado muestra que el día del año para la fecha «2019-08-12» es. 224 Esto significa que el 12 de agosto de 2019 es el día 224 del año 2019.

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

Función DATE_TRUNC

La función DATE_TRUNC trunca todo literal o expresión de marca temporal basado en la parte de fecha especificada, como la hora, la semana o el mes.

Sintaxis

```
date_trunc(format, datetime)
```

Argumentos

format

El formato que representa la unidad a la que se va a truncar. Los formatos válidos son los siguientes:

- «YEAR», «YYYY», «YY»: si se trunca hasta la primera fecha del año en que cae la ts, la parte temporal será igual a cero
- «TRIMESTRE»: trunca hasta la primera fecha del trimestre en el que cae la ts, la parte horaria será cero
- «MONTH», «MM», «MON»: si se trunca hasta la primera fecha del mes en que cae la ts, la parte horaria será cero
- «SEMANA»: si se trunca hasta el lunes de la semana en que cae la ts, la parte horaria será cero
- «DÍA», «DD»: pone a cero la parte horaria
- «HORA»: pone a cero el minuto y el segundo con la parte fraccionada
- «MINUTO»: pone a cero el segundo con la parte fraccionada
- «SEGUNDO»: pone a cero la segunda parte de la fracción
- «MILISEGUNDO»: reduce a cero los microsegundos
- «MICROSEGUNDO»: todo permanece

- Es

Un valor de fecha y hora

Tipo de devolución

Devuelve la marca de tiempo ts truncada a la unidad especificada por el modelo de formato

Ejemplos

En el siguiente ejemplo, se trunca el valor de una fecha hasta el principio del año. El resultado muestra que la fecha «2015-03-05" se ha truncado a «2015-01-01», que es el comienzo del año 2015.

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

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

Función DAY

La función DAY devuelve el día del mes de la fecha/marca horaria.

Las funciones de extracción de fecha son útiles cuando se necesita trabajar con componentes específicos de una fecha o marca de tiempo, como cuando se realizan cálculos basados en fechas, se filtran datos o se formatea valores de fecha.

Sintaxis

```
day(date)
```

Argumentos

date

Una expresión de fecha o marca de hora.

Devuelve

La función DAY devuelve un ENTERO.

Ejemplos

El siguiente ejemplo extrae el día del mes (`30`) de la fecha de entrada `'2009-07-30'`.

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

El siguiente ejemplo extrae el día del mes de la `birthday` columna de la `squirrels` tabla y devuelve los resultados como salida de la instrucción SELECT. El resultado de esta consulta será una lista de valores de día, uno para cada fila de la `squirrels` tabla, que representa el día del mes del cumpleaños de cada ardilla.

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

Función DAYOFMONTH

La función DAYOFMONTH devuelve el día del mes del date/timestamp (un valor entre 1 y 31, según el mes y el año).

La función DAYOFMONTH es similar a la función DAY, pero tienen nombres y comportamientos ligeramente diferentes. La función DAY es la más utilizada, pero la función DAYOFMONTH se puede utilizar como alternativa. Este tipo de consulta puede resultar útil cuando se necesita realizar un análisis basado en fechas o filtrar una tabla que contiene datos de fecha o marca horaria, como extraer componentes específicos de una fecha para su posterior procesamiento o elaboración de informes.

Sintaxis

```
dayofmonth(date)
```

Argumentos

date

Una expresión de fecha o marca de hora.

Devuelve

La función DAYOFMONTH devuelve un ENTERO.

Ejemplo

El siguiente ejemplo extrae el día del mes (`30`) de la fecha de entrada. `'2009-07-30'`

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

En el siguiente ejemplo, se aplica la función DAYOFMONTH a la `birthday` columna de la `squirrels` tabla. Para cada fila de la `squirrels` tabla, se extraerá el día del mes de la `birthday` columna y se devolverá como resultado de la instrucción SELECT. El resultado de esta consulta será una lista de valores de días, uno para cada fila de la `squirrels` tabla, que representa el día del mes del cumpleaños de cada ardilla.

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

Función DAYOFWEEK

La función DAYOFWEEK toma una fecha o marca horaria como entrada y devuelve el día de la semana en forma de número (1 para el domingo, 2 para el lunes,..., 7 para el sábado).

Esta función de extracción de fechas resulta útil cuando se necesita trabajar con componentes específicos de una fecha o marca de tiempo, como cuando se realizan cálculos basados en fechas, se filtran datos o se formatea valores de fecha.

Sintaxis

```
dayofweek(date)
```

Argumentos

date

Una expresión de fecha o marca de hora.

Devuelve

La función DAYOFWEEK devuelve un ENTERO donde

1 = domingo

2 = lunes

3 = martes

4 = miércoles

5 = jueves

6 = viernes

7 = sábado

Ejemplos

El siguiente ejemplo extrae el día de la semana de esta fecha, que es 5 (que representa el jueves).

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

El siguiente ejemplo extrae el día de la semana de la `birthday` columna de la `squirrels` tabla y devuelve los resultados como salida de la instrucción SELECT. El resultado de esta consulta será una lista de los valores del día de la semana, uno para cada fila de la `squirrels` tabla, que representa el día de la semana del cumpleaños de cada ardilla.

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

Función DAYOFYEAR

La función DAYOFYEAR es una función de extracción de fechas que toma una fecha o marca de tiempo como entrada y devuelve el día del año (un valor entre 1 y 366, dependiendo del año y de si es bisiesto).

Esta función resulta útil cuando se necesita trabajar con componentes específicos de una fecha o marca de tiempo, como cuando se realizan cálculos basados en fechas, se filtran datos o se da formato a valores de fecha.

Sintaxis

```
dayofyear(date)
```

Argumentos

date

Una expresión de fecha o marca de hora.

Devuelve

La función DAYOFYEAR devuelve un entero (entre 1 y 366, según el año y si se trata de un año bisiesto).

Ejemplos

El siguiente ejemplo extrae el día del año (`100`) de la fecha de entrada. `'2016-04-09'`

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

El siguiente ejemplo extrae el día del año de la `birthday` columna de la `squirrels` tabla y devuelve los resultados como salida de la instrucción SELECT.

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

Función EXTRACT

La función EXTRACT devuelve una parte de fecha u hora a partir de un valor TIMESTAMP, TIMESTAMPTZ, TIME o TIMETZ. Algunos ejemplos son día, mes, año, hora, minuto, segundo, milisegundo o microsegundo de una marca de tiempo.

Sintaxis

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

Argumentos

datepart

El subcampo de una fecha u hora que se va a extraer, como día, mes, año, hora, minuto, segundo, milisegundo o microsegundo. Para obtener los valores posibles, consulte Partes de fecha para funciones de fecha o marca temporal.

origen

Una columna o una expresión que se evalúa como un tipo de datos TIMESTAMP, TIMESTAMPTZ, TIME o TIMETZ.

Tipo de devolución

INTEGER si el valor de origen se evalúa como tipo de datos TIMESTAMP, TIME o TIMETZ.

DOUBLE PRECISION si el valor de origen se evalúa como el tipo de datos TIMESTAMPTZ.

Ejemplos con TIME

La siguiente tabla de ejemplo, TIME_TEST, tiene una columna TIME_VAL (tipo TIME) con tres valores insertados.

```
select time_val from time_test;

time_val
---------------------
20:00:00
```

```
00:00:00.5550
00:58:00
```

En el siguiente ejemplo, se extraen los minutos de cada time_val.

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

En el siguiente ejemplo, se extraen las horas de cada time_val.

```
select extract(hour from time_val) as hours from time_test;

hours
-----------
         20
         0
         0
```

Función FROM_UTC_TIMESTAMP

La función FROM_UTC_TIMESTAMP convierte la fecha de entrada de UTC (hora universal coordinada) a la zona horaria especificada.

Esta función resulta útil cuando necesitas convertir valores de fecha y hora de UTC a una zona horaria específica. Esto puede ser importante cuando se trabaja con datos que se originan en diferentes partes del mundo y deben presentarse en la hora local adecuada.

Sintaxis

```
from_utc_timestamp(timestamp, timezone
```

Argumentos

timestamp

Una expresión de marca de tiempo con una marca de tiempo UTC.

timezone

Una expresión STRING que es una zona horaria válida a la que se debe convertir la fecha o la marca de tiempo de entrada.

Devuelve

La función FROM_UTC_TIMESTAMP devuelve una MARCA DE TIEMPO.

Ejemplo

En el siguiente ejemplo, se convierte la fecha de entrada de UTC a la zona horaria especificada (`'Asia/Seoul'`), que en este caso está 9 horas por delante de la UTC. El resultado es la fecha y la hora de la zona horaria de Seúl, que es`2016-08-31 09:00:00`.

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

Función HOUR

La función HOUR es una función de extracción de tiempo que toma una hora o una marca de tiempo como entrada y devuelve el componente horario (un valor entre 0 y 23).

Esta función de extracción de tiempo resulta útil cuando se necesita trabajar con componentes específicos de una hora o una marca de tiempo, como cuando se realizan cálculos basados en el tiempo, se filtran datos o se formatea valores de hora.

Sintaxis

```
hour(timestamp)
```

Argumentos

timestamp

UNA EXPRESIÓN DE MARCA DE TIEMPO.

Devuelve

La función HORA devuelve un ENTERO.

Ejemplo

El siguiente ejemplo extrae el componente hour (12) de la marca de tiempo `'2009-07-30 12:58:59'` de entrada.

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

Función MINUTE

La función MINUTE es una función de extracción de tiempo que toma una hora o una marca de tiempo como entrada y devuelve el componente de minutos (un valor entre 0 y 60).

Sintaxis

```
minute(timestamp)
```

Argumentos

timestamp

Una expresión de marca de tiempo o una CADENA con un formato de marca de tiempo válido.

Devuelve

La función MINUTE devuelve un entero.

Ejemplo

El siguiente ejemplo extrae el componente minuto (58) de la marca de tiempo `'2009-07-30 12:58:59'` de entrada.

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

Función MONTH

La función MONTH es una función de extracción de tiempo que toma una hora o una marca de tiempo como entrada y devuelve el componente del mes (un valor entre 0 y 12).

Sintaxis

```
month(date)
```

Argumentos

date

Una expresión de marca de tiempo o una CADENA con un formato de marca de tiempo válido.

Devuelve

La función MONTH devuelve un entero.

Ejemplo

El siguiente ejemplo extrae el componente month (7) de la marca de tiempo `'2016-07-30'` de entrada.

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

SEGUNDA función

La función SECOND es una función de extracción de tiempo que toma una hora o una marca de tiempo como entrada y devuelve el segundo componente (un valor entre 0 y 60).

Sintaxis

```
second(timestamp)
```

Argumentos

timestamp

Una expresión de marca de tiempo.

Devuelve

La función SECOND devuelve un ENTERO.

Ejemplo

El siguiente ejemplo extrae el segundo componente (59) de la marca de tiempo `'2009-07-30 12:58:59'` de entrada.

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

Función TIMESTAMP

La función TIMESTAMP toma un valor (normalmente un número) y lo convierte en un tipo de datos de marca de tiempo.

Esta función resulta útil cuando se necesita convertir un valor numérico que representa una hora o una fecha en un tipo de datos de marca de tiempo. Esto puede resultar útil cuando se trabaja con datos almacenados en un formato numérico, como las marcas de tiempo de Unix o la hora de época.

Sintaxis

```
timestamp(expr)
```

Argumentos

expr

Cualquier expresión que se pueda convertir en TIMESTAMP.

Devuelve

La función TIMESTAMP devuelve una MARCA DE TIEMPO.

Ejemplo

El siguiente ejemplo convierte una marca de tiempo numérica de Unix (`1632416400`) en su tipo de datos de marca de tiempo correspondiente: 22 de septiembre de 2021 a las 12:00:00 p.m. UTC.

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

Función TO_TIMESTAMP

TO_TIMESTAMP convierte una cadena TIMESTAMP en TIMESTAMPTZ.

Sintaxis

```
to_timestamp (timestamp)
```

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

Argumentos

timestamp

Una cadena de marca de tiempo o un tipo de datos que se puede convertir en una cadena de marca de tiempo.

format

Un literal de cadena que coincide con los patrones de fecha y hora de Spark. Para ver patrones de fecha y hora válidos, consulta Patrones de fecha y hora para formatear y analizar.

Tipo de devolución

TIMESTAMP

Ejemplos

En el siguiente ejemplo, se muestra el uso de la función TO_TIMESTAMP para convertir una cadena TIMESTAMP en 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
```

Es posible pasar a TO_TIMESTAMP parte de una fecha. Las partes de fecha restantes se establecen a los valores predeterminados. La hora se incluye en el resultado:

```
SELECT TO_TIMESTAMP('2017','YYYY');
```

```
to_timestamp
--------------------------
2017-01-01 00:00:00+00
```

La siguiente instrucción SQL convierte la cadena '2011-12-18 24:38:15 'en una marca de tiempo. El resultado es una marca de tiempo que cae al día siguiente porque el número de horas es superior a 24 horas:

```
select to_timestamp('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS');

to_timestamp
---------------------
2011-12-19 00:38:15+00
```

Función YEAR

La función YEAR es una función de extracción de fechas que toma una fecha o marca de tiempo como entrada y devuelve el componente del año (un número de cuatro dígitos).

Sintaxis

```
year(date)
```

Argumentos

date

Una expresión de fecha o marca de hora.

Devuelve

La función AÑO devuelve un ENTERO.

Ejemplo

El siguiente ejemplo extrae el componente del año (2016) de la fecha de entrada'2016-07-30'.

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

El siguiente ejemplo extrae el componente de año de la `birthday` columna de la `squirrels` tabla y devuelve los resultados como salida de la instrucción SELECT. El resultado de esta consulta será una lista de valores anuales, uno para cada fila de la `squirrels` tabla, que representa el año del cumpleaños de cada ardilla.

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

Partes de fecha para funciones de fecha o marca temporal

En la siguiente tabla, se identifican los nombres y las abreviaturas de partes de fecha y de hora que se aceptan como argumentos para las siguientes funciones:

- DATE_ADD
- DATE_DIFF
- DATE_PART
- EXTRACT

Parte de la fecha o parte de la hora	Abreviaturas
milenio, milenios	mil, mils
siglo, siglos	c, cent, cents
década, décadas	dec, decs
tiempo Unix	fecha de inicio (compatible con EXTRACT)
año, años	y, yr, yrs
trimestre, trimestres	qtr, qtrs
mes, meses	mon, mons
semana, semanas	w
día de la semana	dayofweek, dow, dw, weekday (compatibles con DATE_PART y Función EXTRACT)

Parte de la fecha o parte de la hora	Abreviaturas
	Devuelve un número entero de 0 a 6, comenzando por domingo. **Note** La parte de la fecha DOW se comporta de manera diferente a la parte de fecha (D) que se usa para las cadenas de formato de fecha y hora. D se basa en los números enteros de 1 a 7, donde domingo es 1. Para obtener más información, consulte Cadenas de formatos de fecha y hora.
día del año	dayofyear, doy, dy, yearday (compatibles con EXTRACT)
día, días	d
hora, horas	h, hr, hrs
minuto, minutos	m, min, mins
segundo, segundos	s, sec, secs
milisegundo, milisegundos	ms, msec, msecs, msecond, mseconds, millisec, millisecs, millisecon
microsegundo, microsegundos	microsec, microsecs, microsecond, usecond, useconds, us, usec, usecs
zona horaria, timezone_hour, timezone_minute	Compatible solo con EXTRACT para marca temporal con zona horaria (TIMESTAMPTZ).

Variaciones en resultados con segundos, milisegundos y microsegundos

Cuando diferentes funciones de fechas especifican segundos, milisegundos o microsegundos como partes de fecha, se generan diferencias mínimas en los resultados de las consultas:

- La función EXTRACT devuelve números enteros solo para la parte de fecha especificada e ignora partes de fecha de niveles mayores y menores. Si la parte de fecha especificada es segundos, los milisegundos y los microsegundos no se incluyen en el resultado. Si la parte de fecha especificada es milisegundos, los segundos y los microsegundos no se incluyen. Si la parte de fecha especificada es microsegundos, los segundos y los milisegundos no se incluyen.
- La función DATE_PART devuelve la parte de segundos de la marca temporal completa, sin importar la parte de fecha especificada, por lo que devuelve un valor decimal o un número entero según se requiera.

Notas acerca de CENTURY, EPOCH, DECADE y MIL

CENTURY o CENTURIES

AWS Clean Rooms interpreta que un SIGLO comienza con el año ## #1 y termina con el año: ###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

La AWS Clean Rooms implementación de EPOCH es relativa a 1970-01-01 00:00:00.000 000, independientemente de la zona horaria en la que resida el clúster. Podría ser necesario desplazar los resultados de la diferencia en horas según la zona horaria donde se encuentre el clúster.

DECADE o DECADES

AWS Clean Rooms interpreta DECADE o DECADES DATEPART basándose en el calendario común. Por ejemplo, debido a que el calendario común comienza a partir del año 1, la primera década (década 1) es de 0001-01-01 a 0009-12-31 y la segunda década (década 2) es de 0010-01-01 a 0019-12-31. Por ejemplo, la década 201 se extiende de 01/01/2001 a 31/12/2009:

```
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 o MILS

AWS Clean Rooms interpreta que una MIL comienza con el primer día del año #001 y termina con el último día del año: #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)
```

Funciones de cifrado y descifrado

Las funciones de cifrado y descifrado ayudan a los desarrolladores de SQL a proteger los datos confidenciales contra el acceso no autorizado o el uso indebido al convertirlos de un formato legible de texto plano a uno de texto cifrado ilegible.

AWS Clean Rooms Spark SQL admite las siguientes funciones de cifrado y descifrado:

Temas

Función AES_ENCRYPT

La función AES_ENCRYPT se utiliza para cifrar datos mediante el algoritmo AES (Advanced Encryption Standard).

Sintaxis

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

Argumentos

expr

El valor binario que se va a cifrar.

clave

La contraseña que se utilizará para cifrar los datos.

Se admiten longitudes de clave de 16, 24 y 32 bits.

mode

Especifica qué modo de cifrado por bloques se debe utilizar para cifrar los mensajes.

Modos válidos: ECB (electrónico CodeBook), GCM (modo Galois/Counter) y CBC (encadenamiento de bloques cifrados).

acolchado

Especifica cómo rellenar los mensajes cuya longitud no sea un múltiplo del tamaño del bloque.

Valores válidos: PKCS, NONE, DEFAULT.

El relleno PREDETERMINADO significa PKCS (estándares de criptografía de clave pública) para ECB, NONE para GCM y PKCS para CBC.

Las combinaciones admitidas de (modo, relleno) son («ECB», «PKCS»), («GCM», «NONE») y («CBC», «PKCS»).

iv

Vector de inicialización opcional (IV). Solo se admite en los modos CBC y GCM.

Valores válidos: 12 bytes de longitud para GCM y 16 bytes para CBC.

aad

Datos autenticados adicionales (AAD) opcionales. Solo se admite en el modo GCM. Puede ser cualquier entrada de formato libre y debe proporcionarse tanto para el cifrado como para el descifrado.

Tipo de retorno

La función AES_ENCRYPT devuelve un valor cifrado de expr mediante AES en un modo determinado con el relleno especificado.

Ejemplos

El siguiente ejemplo muestra cómo utilizar la función AES_ENCRYPT de Spark SQL para cifrar de forma segura una cadena de datos (en este caso, la palabra «Spark») mediante una clave de cifrado específica. A continuación, el texto cifrado resultante se codifica en Base64 para facilitar su almacenamiento o transmisión.

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

El siguiente ejemplo muestra cómo utilizar la función AES_ENCRYPT de Spark SQL para cifrar de forma segura una cadena de datos (en este caso, la palabra «Spark») mediante una clave de cifrado específica. A continuación, el texto cifrado resultante se representa en formato hexadecimal, lo que puede resultar útil para tareas como el almacenamiento, la transmisión o la depuración de datos.

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

El siguiente ejemplo muestra cómo utilizar la función AES_ENCRYPT de Spark SQL para cifrar de forma segura una cadena de datos (en este caso, «Spark SQL») mediante una clave de cifrado,

un modo de cifrado y un modo de relleno específicos. A continuación, el texto cifrado resultante se codifica en Base64 para facilitar su almacenamiento o transmisión.

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

Función AES_DECRYPT

La función AES_DECRYPT se utiliza para descifrar datos mediante el algoritmo AES (Advanced Encryption Standard).

Sintaxis

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

Argumentos

expr

El valor binario que se va a descifrar.

clave

La contraseña que se utilizará para descifrar los datos.

La contraseña debe coincidir con la clave utilizada originalmente para generar el valor cifrado y tener una longitud de 16, 24 o 32 bytes.

mode

Especifica qué modo de cifrado por bloques se debe utilizar para descifrar los mensajes.

Modos válidos: ECB, GCM, CBC.

acolchado

Especifica cómo rellenar los mensajes cuya longitud no sea un múltiplo del tamaño del bloque.

Valores válidos: PKCS, NONE, DEFAULT.

El relleno PREDETERMINADO significa PKCS para ECB, NONE para GCM y PKCS para CBC.

triste

Datos autenticados adicionales (AAD) opcionales. Solo se admite en el modo GCM. Puede ser cualquier entrada de formato libre y debe proporcionarse tanto para el cifrado como para el descifrado.

Tipo de retorno

Devuelve un valor descifrado de expr utilizando AES en modo con relleno.

Ejemplos

El siguiente ejemplo muestra cómo utilizar la función AES_ENCRYPT de Spark SQL para cifrar de forma segura una cadena de datos (en este caso, la palabra «Spark») mediante una clave de cifrado específica. A continuación, el texto cifrado resultante se codifica en Base64 para facilitar su almacenamiento o transmisión.

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

En el siguiente ejemplo, se muestra cómo utilizar la función AES_DECRYPT de Spark SQL para descifrar datos previamente cifrados y codificados en Base64. El proceso de descifrado requiere la clave y los parámetros de cifrado correctos (modo de cifrado y modo de relleno) para recuperar correctamente los datos originales en texto plano.

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

Funciones hash

Una función hash es una función matemática que convierte un valor de entrada numérico en otro valor.

AWS Clean Rooms Spark SQL admite las siguientes funciones hash:

Temas

- MD5 función
- Función SHA

- SHA1 función
- SHA2 función
- función xx HASH64

MD5 función

Utiliza la función hash MD5 criptográfica para convertir una cadena de longitud variable en una cadena de 32 caracteres que es una representación textual del valor hexadecimal de una suma de control de 128 bits.

Sintaxis

```
MD5(string)
```

Argumentos

string

Una cadena de longitud variable.

Tipo de retorno

La MD5 función devuelve una cadena de 32 caracteres que es una representación textual del valor hexadecimal de una suma de comprobación de 128 bits.

Ejemplos

En el siguiente ejemplo, se muestra el valor de 128 bits para la cadena 'AWS Clean Rooms':

```
select md5('AWS Clean Rooms');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

Función SHA

Sinónimo de función. SHA1

Consulte SHA1 función.

SHA1 función

La SHA1 función utiliza la función hash SHA1 criptográfica para convertir una cadena de longitud variable en una cadena de 40 caracteres que es una representación textual del valor hexadecimal de una suma de verificación de 160 bits.

Sintaxis

SHA1 es sinónimo de. Función SHA

```
SHA1(string)
```

Argumentos

string

Una cadena de longitud variable.

Tipo de retorno

La SHA1 función devuelve una cadena de 40 caracteres que es una representación textual del valor hexadecimal de una suma de verificación de 160 bits.

Ejemplo

En el siguiente ejemplo, se devuelve el valor de 160 bits para la palabra 'AWS Clean Rooms':

```
select sha1('AWS Clean Rooms');
```

SHA2 función

La SHA2 función utiliza la función hash SHA2 criptográfica para convertir una cadena de longitud variable en una cadena de caracteres. La cadena de caracteres es una representación de texto del valor hexadecimal de la suma de comprobación con el número especificado de bits.

Sintaxis

```
SHA2(string, bits)
```

Argumentos

string

Una cadena de longitud variable.

integer

El número de bits en las funciones hash. Los valores válidos son 0 (igual que 256), 224, 256, 384 y 512.

Tipo de retorno

La SHA2 función devuelve una cadena de caracteres que es una representación textual del valor hexadecimal de la suma de comprobación o una cadena vacía si el número de bits no es válido.

Ejemplo

En el siguiente ejemplo, se devuelve el valor de 256 bits para la palabra 'AWS Clean Rooms':

```
select sha2('AWS Clean Rooms', 256);
```

función xx HASH64

La función xxhash64 devuelve un valor hash de 64 bits de los argumentos.

La función xxhash64 () es una función hash no criptográfica diseñada para ser rápida y eficiente. Suele utilizarse en aplicaciones de procesamiento y almacenamiento de datos, en las que se necesita un identificador único para un dato, pero no es necesario mantener en secreto el contenido exacto de los datos.

En el contexto de una consulta SQL, la función xxhash64 () podría usarse para varios propósitos, como:

- Generar un identificador único para una fila de una tabla
- Particionar los datos en función de un valor hash
- Implementación de estrategias personalizadas de indexación o distribución de datos

El caso de uso específico dependerá de los requisitos de la aplicación y de los datos que se procesen.

Sintaxis

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

Argumentos

expr1

Una expresión de cualquier tipo.

expr2

Una expresión de cualquier tipo.

Devuelve

Devuelve un valor hash de 64 bits de los argumentos (BIGINT). La velocidad del hash es 42.

Ejemplo

El siguiente ejemplo genera un valor hash de 64 bits (5602566077635097486) en función de la entrada proporcionada. El primer argumento es un valor de cadena, en este caso, la palabra «Spark». El segundo argumento es una matriz que contiene el valor entero único 123. El tercer argumento es un valor entero que representa la semilla de la función hash.

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

Funciones de hiperloglog

Las funciones HyperLogLog (HLL) de SQL proporcionan una forma de estimar de manera eficiente el número de elementos únicos (cardinalidad) en un conjunto de datos grande, incluso cuando el conjunto real de elementos únicos no está almacenado.

Las principales ventajas de utilizar las funciones HLL son:

- Eficiencia de la memoria: los bocetos HLL requieren mucha menos memoria que almacenar el conjunto completo de elementos únicos, lo que los hace adecuados para conjuntos de datos de gran tamaño.
- Computación distribuida: los bocetos HLL se pueden combinar en múltiples fuentes de datos o nodos de procesamiento, lo que permite una estimación eficiente y distribuida del recuento único.

- Resultados aproximados: el HLL proporciona una estimación aproximada del recuento único, con una compensación ajustable entre la precisión y el uso de memoria (mediante el parámetro de precisión).

Estas funciones son especialmente útiles en situaciones en las que es necesario estimar el número de elementos únicos, como en aplicaciones de análisis, almacenamiento de datos y procesamiento de transmisiones en tiempo real.

AWS Clean Rooms admite las siguientes funciones HLL.

Temas

función HLL_SKETCH_AGG

La función de agregado HLL_SKETCH_AGG crea un boceto HLL a partir de los valores de la columna especificada. Devuelve un tipo de datos HLLSKETCH que encapsula los valores de la expresión de entrada.

La función de agregado HLL_SKETCH_AGG funciona con cualquier tipo de datos e ignora los valores NULL.

Cuando no hay filas en una tabla o todas las filas son NULL, el boceto resultante no tiene pares índice-valor como `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}`.

Sintaxis

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

Argumento

expresión_de_agregación

Cualquier expresión de tipo INT, BIGINT, STRING o BINARY con la que se realizará un recuento único. Se ignoran todos NULL los valores.

lgConfigk

Una constante INT opcional entre 4 y 21, ambos incluidos, con el valor predeterminado 12. El log-base-2 de K, donde K es el número de cubos o ranuras del boceto.

Tipo de retorno

La función HLL_SKETCH_AGG devuelve un búfer BINARIO no nulo que contiene el HyperLogLog boceto calculado debido a que consume y agrega todos los valores de entrada del grupo de agregación.

Ejemplos

En los ejemplos siguientes se utiliza el algoritmo HyperLogLog (HLL) para estimar el recuento distinto de valores de la columna. `col` La `hll_sketch_agg(col, 12)` función agrega los valores de la columna de columnas y crea un boceto HLL con una precisión de 12. A continuación, la `hll_sketch_estimate()` función se utiliza para estimar el recuento distinto de valores en función del boceto HLL generado. El resultado final de la consulta es 3, que representa el recuento distinto estimado de valores de la `col` columna. En este caso, los valores distintos son 1, 2 y 3.

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

El siguiente ejemplo también utiliza el algoritmo HLL para estimar el recuento distinto de valores de la `col` columna, pero no especifica un valor de precisión para el boceto HLL. En este caso, utiliza la precisión por defecto de 14. La `hll_sketch_agg(col)` función toma los valores de la `col` columna y crea un boceto HyperLogLog (HLL), que es una estructura de datos compacta que se puede utilizar para estimar el recuento distinto de elementos. La `hll_sketch_estimate(hll_sketch_agg(col))` función toma el boceto HLL creado en el paso anterior y calcula una estimación del recuento distinto de valores de la `col` columna. El resultado final de la consulta es 3, que representa el recuento distinto estimado de valores de la `col` columna. En este caso, los valores distintos son 1, 2 y 3.

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

Función HLL_SKETCH_ESTIMATE

La función HLL_SKETCH_ESTIMATE toma un boceto HLL y estima el número de elementos únicos representados por el boceto. Utiliza el algoritmo HyperLogLog (HLL) para contar una aproximación probabilística del número de valores únicos de una columna determinada, consumiendo una representación binaria conocida como búfer de croquis generada previamente por la función HLL_SKETCH_AGG y devolviendo el resultado como un entero grande.

El algoritmo de boceto HLL proporciona una forma eficaz de estimar el número de elementos únicos, incluso en el caso de conjuntos de datos grandes, sin tener que almacenar todo el conjunto de valores únicos.

`hll_union_agg`Las funciones `hll_union` y también pueden combinar bocetos consumiendo y fusionando estos búferes como entradas.

Sintaxis

```
HLL_SKETCH_ESTIMATE (hllsketch_expression)
```

Argumento

hllsketch_expression

`BINARY`Expresión que contiene un boceto generado por HLL_SKETCH_AGG

Tipo de retorno

La función HLL_SKETCH_ESTIMATE devuelve un valor de BIGINT que es el recuento distinto aproximado representado por el boceto de entrada.

Ejemplos

Los ejemplos siguientes utilizan el algoritmo de boceto HyperLogLog (HLL) para estimar la cardinalidad (recuento único) de los valores de la columna. `col` La `hll_sketch_agg(col, 12)` función toma la `col` columna y crea un boceto HLL con una precisión de 12 bits. El boceto HLL es una estructura de datos aproximada que puede estimar de manera eficiente el número de elementos únicos de un conjunto. La `hll_sketch_estimate()` función toma el boceto HLL creado por el boceto `hll_sketch_agg` y estima la cardinalidad (recuento único) de los valores representados por el boceto. `FROM VALUES (1), (1), (2), (2), (3) tab(col);`Genera un conjunto de datos

de prueba con 5 filas, donde la `col` columna contiene los valores 1, 1, 2, 2 y 3. El resultado de esta consulta es el recuento único estimado de los valores de la `col` columna, que es 3.

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

La diferencia entre el ejemplo siguiente y el anterior es que el parámetro de precisión (12 bits) no se especifica en la llamada a la `hll_sketch_agg` función. En este caso, se utiliza la precisión predeterminada de 14 bits, lo que puede proporcionar una estimación más precisa del recuento único en comparación con el ejemplo anterior, que utilizaba 12 bits de precisión.

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

Función HLL_UNION

La función HLL_UNION combina dos bocetos HLL en un boceto único y unificado. Utiliza el algoritmo HyperLogLog (HLL) para combinar dos bocetos en un solo boceto. Las consultas pueden usar los búferes resultantes para calcular recuentos únicos aproximados como enteros largos con la función. `hll_sketch_estimate`

Sintaxis

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

Argumento

eXPRN

`BINARY`Expresión que contiene un boceto generado por HLL_SKETCH_AGG.

allowDifferentLgConfiguración

Una expresión BOOLEANA opcional que controla si se permite la fusión de dos bocetos con valores de lgConfigK diferentes. El valor predeterminado es `false`.

Tipo de retorno

La función HLL_UNION devuelve un búfer BINARIO que contiene el HyperLogLog boceto calculado como resultado de la combinación de las expresiones de entrada. Cuando el `allowDifferentLgConfigK` parámetro es`true`, el boceto resultante utiliza el menor de los dos valores proporcionados. `lgConfigK`

Ejemplos

Los siguientes ejemplos utilizan el algoritmo de boceto HyperLogLog (HLL) para estimar el recuento único de valores en dos columnas `col1` y `col2` en un conjunto de datos.

La `hll_sketch_agg(col1)` función crea un boceto HLL para los valores únicos de la columna. `col1`

La `hll_sketch_agg(col2)` función crea un boceto HLL para los valores únicos de la columna col2.

La `hll_union(...)` función combina los dos bocetos HLL creados en los pasos 1 y 2 en un solo boceto HLL unificado.

La `hll_sketch_estimate(...)` función toma el boceto HLL combinado y estima el recuento único de valores entre ambas y. `col1` `col2`

La `FROM VALUES` cláusula genera un conjunto de datos de prueba con 5 filas, donde `col1` contiene los valores 1, 1, 2, 2 y 3, y `col2` contiene los valores 4, 4, 5, 5 y 6.

El resultado de esta consulta es el recuento único estimado de valores entre ambos `col1` y`col2`, que es 6. El algoritmo de boceto HLL proporciona una forma eficaz de estimar el número de elementos únicos, incluso en el caso de conjuntos de datos grandes, sin tener que almacenar todo el conjunto de valores únicos. En este ejemplo, la `hll_union` función se utiliza para combinar los bocetos HLL de las dos columnas, lo que permite estimar el recuento único en todo el conjunto de datos, en lugar de hacerlo solo para cada columna individualmente.

```
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
```

La diferencia entre el ejemplo siguiente y el anterior es que el parámetro de precisión (12 bits) no se especifica en la llamada a la `hll_sketch_agg` función. En este caso, se utiliza la precisión predeterminada de 14 bits, lo que puede proporcionar una estimación más precisa del recuento único en comparación con el ejemplo anterior, que utilizaba 12 bits de precisión.

```
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);
```

Función HLL_UNION_AGG

La función HLL_UNION_AGG combina varios bocetos HLL en un solo boceto unificado. Utiliza el algoritmo HyperLogLog (HLL) para combinar un grupo de bocetos en uno solo. Las consultas pueden usar los búferes resultantes para calcular recuentos únicos aproximados con la `hll_sketch_estimate` función.

Sintaxis

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

Argumento

expr

`BINARY`Expresión que contiene un boceto generado por HLL_SKETCH_AGG.

allowDifferentLgConfiguración

Una expresión BOOLEANA opcional que controla si se permite la fusión de dos bocetos con valores de lgConfigK diferentes. El valor predeterminado es `false`.

Tipo de retorno

La función HLL_UNION_AGG devuelve un búfer BINARIO que contiene el HyperLogLog boceto calculado como resultado de la combinación de las expresiones de entrada del mismo grupo. Cuando el `allowDifferentLgConfigK` parámetro es`true`, el boceto resultante utiliza el menor de los dos valores proporcionados. `lgConfigK`

Ejemplos

Los siguientes ejemplos utilizan el algoritmo de boceto HyperLogLog (HLL) para estimar el recuento único de valores en varios bocetos HLL.

El primer ejemplo estima el recuento único de valores de un conjunto de datos.

```
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
```

La consulta interna crea dos bocetos HLL:

- La primera instrucción SELECT crea un boceto a partir de un único valor de 1.
- La segunda instrucción SELECT crea un boceto a partir de otro valor único de 1, pero con una precisión de 20.

La consulta externa utiliza la función HLL_UNION_AGG para combinar los dos bocetos en un solo boceto. A continuación, aplica la función HLL_SKETCH_ESTIMATE a este boceto combinado para estimar el recuento único de valores.

El resultado de esta consulta es el recuento único estimado de los valores de la columna, que es. `col` 1 Esto significa que los dos valores de entrada de 1 se consideran únicos, aunque tengan el mismo valor.

El segundo ejemplo incluye un parámetro de precisión diferente para la función HLL_UNION_AGG. En este caso, ambos bocetos HLL se crean con una precisión de 14 bits, lo que permite combinarlos correctamente con el parámetro. `hll_union_agg` `true`

```
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
```

El resultado final de la consulta es el recuento único estimado, que en este caso también 1 lo es. Esto significa que los dos valores de entrada de 1 se consideran únicos, aunque tengan el mismo valor.

Funciones JSON

Cuando necesita almacenar un conjunto relativamente pequeño de pares clave-valor, puede ahorrar espacio al almacenar los datos en formato JSON. Debido a que las cadenas JSON se pueden almacenar en una única columna, utilizar JSON puede ser más eficiente que almacenar los datos en formato de tabla.

Example

Por ejemplo, piense en una tabla dispersa en la que necesita tener un gran número de columnas para representar completamente todos los atributos posibles. Sin embargo, la mayoría de los valores de las columnas son NULL para cualquier fila o columna determinada. Al usar JSON con fines de almacenamiento, puede almacenar los datos para una fila en pares de clave-valor en una única cadena JSON y eliminar las columnas de tabla pobladas de forma dispersa.

Además, puede modificar fácilmente las cadenas JSON para almacenar pares clave-valor adicionales sin necesidad de agregar columnas a una tabla.

Recomendamos utilizar JSON con moderación. JSON no es una buena alternativa para almacenar grandes conjuntos de datos porque, al almacenar datos dispersos en una única columna, JSON no utiliza la arquitectura de almacén de columnas de AWS Clean Rooms .

JSON utiliza cadenas de texto con cifrado UTF-8, por lo que las cadenas JSON se pueden almacenar como tipos de datos CHAR o VARCHAR. Utilice VARCHAR si las cadenas incluyen caracteres multibytes.

Las cadenas JSON deben tener el formato JSON adecuado, conforme a las siguientes reglas:

- El JSON a nivel raíz puede ser un objeto JSON o una matriz JSON. Un objeto JSON es un conjunto no ordenado de pares clave-valor separados por comas y delimitado con llaves.

  Por ejemplo, `{"one":1, "two":2}`
- Una matriz JSON es un conjunto ordenado de valores separados por comas delimitado entre corchetes.

  A continuación se muestra un ejemplo: `["first", {"one":1}, "second", 3, null]`
- Las matrices JSON utilizan un índice basado en cero; el primer elemento en una matriz está en la posición 0. En un par clave:valor de JSON, la clave es una cadena con comillas dobles.
- El valor JSON puede ser cualquiera de los siguientes valores:
  - Objeto JSON
  - matriz JSON
  - Cadena entre comillas dobles
  - Número (entero y flotante)
  - Booleano
  - Nulo
- Los objetos y las matrices vacíos son valores JSON válidos.
- Los campos JSON distinguen entre mayúsculas y minúsculas.
- Se ignoran los espacios en blanco entre los elementos estructurales de JSON (como `{ }, [ ]`).

Temas

Función GET_JSON_OBJECT

La función GET_JSON_OBJECT extrae un objeto json de. `path`

Sintaxis

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

Argumentos

json_txt

Una expresión STRING que contiene un JSON bien formado.

path

Un literal STRING con una expresión de ruta JSON bien formada.

Devuelve

Devuelve una cadena.

Si no se encuentra el objeto, se devuelve un valor NULL.

Ejemplo

El siguiente ejemplo extrae un valor de un objeto JSON. El primer argumento es una cadena JSON que representa un objeto simple con un único par clave-valor. El segundo argumento es una expresión de ruta JSON. El $ símbolo representa la raíz del objeto JSON y la `.a` parte especifica que queremos extraer el valor asociado a la clave a «». El resultado de la función es 'b', que es el valor asociado a la tecla «a» en el objeto JSON de entrada.

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

Función TO_JSON

La función TO_JSON convierte una expresión de entrada en una representación de cadena JSON. La función gestiona la conversión de diferentes tipos de datos (como números, cadenas y valores booleanos) en sus correspondientes representaciones JSON.

La función TO_JSON resulta útil cuando se necesitan convertir datos estructurados (como filas de bases de datos u objetos JSON) a un formato más portátil y autodescriptivo, como JSON. Esto puede resultar especialmente útil cuando necesitas interactuar con otros sistemas o servicios que esperan datos con formato JSON.

Sintaxis

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

Argumentos

expr

La expresión de entrada que desea convertir en una cadena JSON. Puede ser un valor, una columna o cualquier otra expresión SQL válida.

options

Un conjunto opcional de opciones de configuración que se puede utilizar para personalizar el proceso de conversión a JSON. Estas opciones pueden incluir aspectos como el manejo de valores nulos, la representación de valores numéricos y el tratamiento de los caracteres especiales.

Devuelve

Devuelve una cadena JSON con un valor de estructura determinado

Ejemplos

El siguiente ejemplo convierte una estructura con nombre (un tipo de datos estructurados) en una cadena JSON. El primer argumento (`named_struct('a', 1, 'b', 2)` () es la expresión de entrada que se pasa a la `to_json()` función. Crea una estructura con nombre con dos campos: «a» con un valor de 1 y «b» con un valor de 2. La función to_json () toma la estructura nombrada como argumento y la convierte en una representación de cadena JSON. El resultado es `{"a":1,"b":2}` una cadena JSON válida que representa la estructura nombrada.

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

El siguiente ejemplo convierte una estructura con nombre que contiene un valor de marca de tiempo en una cadena JSON, con un formato de marca de tiempo personalizado. El primer argumento (`named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd'))`) crea una estructura con nombre con un único campo «time» que contiene el valor de la marca de tiempo. El segundo argumento (`map('timestampFormat', 'dd/MM/yyyy')`) crea un mapa (diccionario clave-valor) con un único par clave-valor, donde la clave es 'TimestampFormat' y el valor es ". dd/MM/yyyy'. This map is used to specify the desired format for the timestamp value when converting it to JSON. The to_json() function converts the named struct into a JSON string. The second argument, the map, is used to customize the timestamp format to 'dd/MM/yyyy El resultado es

`{"time":"26/08/2015"}` una cadena JSON con un solo campo «time» que contiene el valor de la marca de tiempo en el formato «» deseado. dd/MM/yyyy

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

Funciones matemáticas

En esta sección se describen las funciones y los operadores matemáticos compatibles con AWS Clean Rooms Spark SQL.

Temas

- Función RADIANS
- Función RAND
- Función RANDOM
- Función ROUND
- Función SIGN
- Función SIN
- Función SQRT
- Función TRUNC

Símbolos de operadores matemáticos

En la tabla siguiente, se muestran los operadores matemáticos admitidos.

Operadores admitidos

Operador	Description (Descripción)	Ejemplo	Resultado
+	suma	2 + 3	5
-	resta	2 - 3	-1
*	multiplicación	2 * 3	6
/	división	4/2	2
%	módulo	5 % 4	1
^	potencia	2,0 ^ 3,0	8

Ejemplos

Se calcula la comisión pagada más una tarifa de manipulación de 2,00 $ para una determinada transacción:

```
select commission, (commission + 2.00) as comm
from sales where salesid=10000;

commission | comm
-----------+-------
28.05      | 30.05
(1 row)
```

Calcule el 20% del precio de venta para una transacción dada:

```
select pricepaid, (pricepaid * .20) as twentypct
from sales where salesid=10000;

pricepaid | twentypct
----------+-----------
187.00    |    37.400
(1 row)
```

Prevea la venta de tickets según un patrón de crecimiento continuo. En este ejemplo, la subconsulta devuelve la cantidad de tickets vendidos en 2008. El resultado se multiplica exponencialmente por un índice de crecimiento continuo del 5 % a 10 años.

```
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)
```

Se encuentra el precio total pagado y la comisión por ventas con un ID de fecha que sea mayor que o igual a 2000. Luego, se resta la comisión total del precio total pagado.

```
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)
```

Función ABS

ABS calcula el valor absoluto de un número, donde ese número puede ser un valor literal o una expresión que tome el valor de un número.

Sintaxis

```
ABS (number)
```

Argumentos

número

Número o expresión que toma el valor de un número. Puede ser SMALLINT, INTEGER, BIGINT FLOAT4, DECIMAL o type. FLOAT8

Tipo de devolución

ABS devuelve el mismo tipo de datos como su argumento.

Ejemplos

Calcular el valor absoluto de -38:

```
select abs (-38);
abs
-------
38
```

```
(1 row)
```

Calcular el valor absoluto de (14-76):

```
select abs (14-76);
abs
-------
62
(1 row)
```

Función ACOS

ACOS es una función trigonométrica que devuelve el arcocoseno de un número. El valor de retorno está en radianes y se encuentra entre `0` y `PI`.

Sintaxis

```
ACOS(number)
```

Argumentos

número

El parámetro de entrada es un número de `DOUBLE PRECISION`.

Tipo de devolución

`DOUBLE PRECISION`

Ejemplos

Para devolver el arcoseno de `-1`, use el siguiente ejemplo.

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Función ASIN

ASIN es una función trigonométrica que devuelve el arcoseno de un número. El valor de retorno está en radianes y se encuentra entre `PI/2` y `-PI/2`.

Sintaxis

```
ASIN(number)
```

Argumentos

número

El parámetro de entrada es un número de `DOUBLE PRECISION`.

Tipo de devolución

`DOUBLE PRECISION`

Ejemplos

Para devolver el arcoseno de 1, use el siguiente ejemplo.

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

Función ATAN

ATAN es una función trigonométrica que devuelve la arcotangente de un número. El valor de retorno está en radianes y se encuentra entre `-PI` y `PI`.

Sintaxis

```
ATAN(number)
```

Argumentos

número

El parámetro de entrada es un número de DOUBLE PRECISION.

Tipo de devolución

DOUBLE PRECISION

Ejemplos

Para devolver la arcotangente de 1 y multiplicarla por 4, use el siguiente ejemplo.

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

ATAN2 función

ATAN2 es una función trigonométrica que devuelve el arco tangente de un número dividido por otro número. El valor de retorno está en radianes y se encuentra entre PI/2 y -PI/2.

Sintaxis

```
ATAN2(number1, number2)
```

Argumentos

number1

Un número de DOUBLE PRECISION.

number2

Un número de DOUBLE PRECISION.

Tipo de devolución

`DOUBLE PRECISION`

Ejemplos

Para devolver la arcotangente de 2/2 y multiplicarla por 4, use el siguiente ejemplo.

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Función CBRT

La función CBRT es una función matemática que calcula la raíz cúbica de un número.

Sintaxis

```
CBRT (number)
```

Argumento

CBRT toma un número con un valor de DOUBLE PRECISION como argumento.

Tipo de devolución

CBRT devuelve un número con un valor de DOUBLE PRECISION.

Ejemplos

Calcular la raíz cúbica de la comisión pagada para una transacción dada:

```
select cbrt(commission) from sales where salesid=10000;

cbrt
------------------
3.03839539048843
(1 row)
```

Función CEILING (o CEIL)

La función CEILING o CEIL se usa para redondear un número hacia arriba hasta el próximo número entero. (La Función FLOOR redondea un número hacia abajo hasta el próximo número entero).

Sintaxis

```
CEIL | CEILING(number)
```

Argumentos

número

El número o la expresión que toma el valor de un número. Puede ser SMALLINT, INTEGER, BIGINT FLOAT4, DECIMAL o type. FLOAT8

Tipo de devolución

CEILING y CEIL devuelven el mismo tipo de datos como su argumento.

Ejemplo

Calcular el límite máximo de la comisión pagada para una transacción dada de ventas:

```
select ceiling(commission) from sales
where salesid=10000;

ceiling
---------
29
(1 row)
```

Función COS

COS es una función trigonométrica que devuelve el coseno de un número. El valor de retorno está en radianes y se encuentra entre -1 y 1, inclusive.

Sintaxis

```
COS(double_precision)
```

Argumento

número

El parámetro de entrada es un número de doble precisión.

Tipo de devolución

La función COS devuelve un número de doble precisión.

Ejemplos

El siguiente ejemplo devuelve el coseno de 0:

```
select cos(0);
cos
-----
1
(1 row)
```

El siguiente ejemplo devuelve el coseno de Pi:

```
select cos(pi());
cos
-----
-1
(1 row)
```

Función COT

COT es una función trigonométrica que devuelve la cotangente de un número. El parámetro de entrada debe ser distinto de cero.

Sintaxis

```
COT(number)
```

Argumento

número

El parámetro de entrada es un número de `DOUBLE PRECISION`.

Tipo de devolución

`DOUBLE PRECISION`

Ejemplos

Para devolver la cotangente de 1, use el siguiente ejemplo.

```
SELECT COT(1);

+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

Función DEGREES

Convierte un ángulo en radianes a su equivalente en grados.

Sintaxis

```
DEGREES(number)
```

Argumento

número

El parámetro de entrada es un número de `DOUBLE PRECISION`.

Tipo de devolución

`DOUBLE PRECISION`

Ejemplo

Para devolver el equivalente en grados de .5 radianes, use el siguiente ejemplo.

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
```

```
+-------------------+
| 28.64788975654116 |
+-------------------+
```

Para convertir radianes de Pi a grados, use el siguiente ejemplo.

```
SELECT DEGREES(pi());

+---------+
| degrees |
+---------+
|     180 |
+---------+
```

Función DIV

El operador DIV devuelve la parte integral de la división del dividendo por el divisor.

Sintaxis

```
dividend div divisor
```

Argumentos

dividendo

Expresión que se evalúa como un valor numérico o un intervalo.

divisor

Un tipo de intervalo coincidente si `dividend` es un intervalo, numérico en caso contrario.

Tipo de devolución

`BIGINT`

Ejemplos

En el siguiente ejemplo, se seleccionan dos columnas de la tabla de ardillas: la `id` columna, que contiene el identificador único de cada ardilla, y una `calculated` columna`age div 2`, que representa la división de enteros de la columna de edad entre 2. El `age div 2` cálculo divide los

enteros de la `age` columna, redondeando así la edad al entero par más próximo. Por ejemplo, si la `age` columna contiene valores como 3, 5, 7 y 10, contendrá los valores 1, 2, 3 y 5, respectivamente. `age div 2`

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

Esta consulta puede resultar útil en situaciones en las que necesite agrupar o analizar datos en función de los rangos de edad y desee simplificar los valores de edad redondeándolos al número entero par más cercano. El resultado resultante proporcionaría la edad `id` y la edad divididas por 2 para cada ardilla de la `squirrels` tabla.

Función EXP

La función EXP implementa la función exponencial para una expresión numérica, o la base del logaritmo natural, e, elevada a potencia de expresión. La función EXP es la operación inversa de Función LN.

Sintaxis

```
EXP (expression)
```

Argumento

expresión

La expresión debe ser un tipo de datos INTEGER, DECIMAL o DOUBLE PRECISION.

Tipo de devolución

EXP devuelve un número con un valor de DOUBLE PRECISION.

Ejemplo

Se utiliza la función EXP para prever las ventas de tickets según un patrón de crecimiento continuo. En este ejemplo, la subconsulta devuelve la cantidad de tickets vendidos en 2008. El resultado se multiplica por el resultado de la función EXP, que especifica un índice de crecimiento continuo del 7% durante 10 años.

```
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)
```

Función FLOOR

La función FLOOR redondea un número hacia abajo hasta el próximo número entero.

Sintaxis

```
FLOOR (number)
```

Argumento

número

El número o la expresión que toma el valor de un número. Puede ser SMALLINT, INTEGER, BIGINT, FLOAT4 DECIMAL o type. FLOAT8

Tipo de devolución

FLOOR devuelve el mismo tipo de datos como su argumento.

Ejemplo

En el ejemplo se muestra el valor de la comisión pagada por una transacción de ventas determinada antes y después de usar la función FLOOR.

```
select commission from sales
where salesid=10000;

floor
-------
28.05
(1 row)

select floor(commission) from sales
where salesid=10000;
```

```
floor
-------
28
(1 row)
```

Función LN

La función LN devuelve el logaritmo natural del parámetro de entrada.

Sintaxis

```
LN(expression)
```

Argumento

expresión

La columna o expresión de destino sobre la que opera la función.

> **Note**
>
> Esta función devuelve un error para algunos tipos de datos si la expresión hace referencia a una tabla AWS Clean Rooms creada por el usuario o a una tabla del sistema AWS Clean Rooms STL o STV.

Las expresiones con los siguientes tipos de datos producen un error si usa como referencia una tabla de sistema o creada por usuarios.

- BOOLEANO
- CHAR
- DATE
- DECIMAL o NUMERIC
- TIMESTAMP
- VARCHAR

Las expresiones con los siguientes tipos de datos se ejecutan con éxito en tablas creadas por usuarios y tablas de sistema STL o STV:

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

Tipo de devolución

La función LN devuelve el mismo tipo que la expresión.

Ejemplo

El siguiente ejemplo devuelve el logaritmo natural, o la base de logaritmo, del número 2,718281828:

```
select ln(2.718281828);
ln
--------------------
0.9999999998311267
(1 row)
```

Tenga en cuenta que la respuesta es casi igual a 1.

En este ejemplo, se devuelve el logaritmo natural de los valores en la columna USERID en la tabla USERS:

```
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)
```

Función LOG

Devuelve el logaritmo de `expr` con`base`.

Sintaxis

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

Argumento

expr

La expresión debe ser un tipo de datos entero, decimal o de punto flotante.

base

La base para el cálculo del logaritmo. Debe ser un número positivo (distinto de 1) del tipo de datos de doble precisión.

Tipo de devolución

La función LOG devuelve un número de doble precisión.

Ejemplo

El siguiente ejemplo devuelve el logaritmo de base 10 del número 100:

```
select log(10, 100);
--------
2
(1 row)
```

Función MOD

Devuelve el resto de dos números, también denominada operación de módulo. Para calcular el resultado, el primer parámetro se divide entre el segundo.

Sintaxis

```
MOD(number1, number2)
```

Argumentos

number1

El primer parámetro de entrada es un número con un valor de tipo INTEGER, SMALLINT, BIGINT o DECIMAL. Si cada parámetro es de tipo DECIMAL, el otro parámetro debe ser también un tipo DECIMAL. Si cada parámetro es un valor INTEGER, el otro parámetro puede ser INTEGER, SMALLINT o BIGINT. Ambos parámetros pueden ser SMALLINT o BIGINT, pero un parámetro no puede ser SMALLINT si el otro es BIGINT.

number2

El segundo parámetro de entrada es un número con un valor de tipo INTEGER, SMALLINT, BIGINT o DECIMAL. Se aplican las mismas reglas de tipo de datos en number2 y en number1.

Tipo de devolución

Los tipos de retorno válidos son DECIMAL, INT, SMALLINT y BIGINT. El tipo de retorno de la función MOD es el mismo tipo numérico que los parámetros de entrada, si ambos parámetros de entrada son del mismo tipo. No obstante, si algún parámetro de entrada es un valor INTEGER, el tipo de retorno también será INTEGER.

Notas de uso

Puede utilizar % como operador de módulo.

Ejemplos

En el siguiente ejemplo, se devuelve el resto cuando se divide un número entre otro:

```
SELECT MOD(10, 4);

 mod
------
 2
```

En el siguiente ejemplo, se devuelve un resultado decimal:

```
SELECT MOD(10.5, 4);

 mod
------
```

```
 2.5
```

Puede convertir valores de parámetro:

```
SELECT MOD(CAST(16.4 as integer), 5);

 mod
------
 1
```

Compruebe si el primer parámetro es par dividiéndolo entre 2:

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

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

Puede utilizar % como operador de módulo:

```
SELECT 11 % 4 as remainder;

 remainder
-----------
 3
```

El siguiente ejemplo devuelve la información para categorías con números impares en la tabla CATEGORY:

```
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)
```

Función PI

La función PI devuelve el valor de Pi a 14 lugares decimales.

Sintaxis

```
PI()
```

Tipo de devolución

`DOUBLE PRECISION`

Ejemplos

Para devolver el valor de pi, utilice el ejemplo siguiente.

```
SELECT PI();

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Función POWER

La función POWER es una función exponencial que eleva una expresión numérica a la potencia de una segunda expresión numérica. Por ejemplo, 2 a la tercera potencia se calcula como `POWER(2,3)`, con un resultado de 8.

Sintaxis

```
{POWER(expression1, expression2)
```

Argumentos

expression1

Expresión numérica que se elevará. Debe ser un tipo de datos `INTEGER`, `DECIMAL` o `FLOAT`.

expression2

Potencia a la que se va a elevar expression1. Debe ser un tipo de datos `INTEGER`, `DECIMAL` o `FLOAT`.

Tipo de devolución

`DOUBLE PRECISION`

Ejemplo

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

Función RADIANS

La función RADIANS convierte un ángulo en grados a su equivalente en radianes.

Sintaxis

```
RADIANS(number)
```

Argumento

número

El parámetro de entrada es un número de `DOUBLE PRECISION`.

Tipo de devolución

`DOUBLE PRECISION`

Ejemplo

Para devolver el equivalente en radianes de 180 grados, use el siguiente ejemplo.

```
SELECT RADIANS(180);

+-------------------+
|      radians      |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Función RAND

La función RAND genera un número aleatorio de punto flotante entre 0 y 1. La función RAND genera un nuevo número aleatorio cada vez que se llama.

Sintaxis

```
RAND()
```

Tipo de devolución

RANDOM devuelve un DOUBLE.

Ejemplo

El siguiente ejemplo genera una columna de números aleatorios de punto flotante entre 0 y 1 para cada fila de la tabla. `squirrels` El resultado sería una sola columna con una lista de valores decimales aleatorios, con un valor para cada fila de la tabla Squirrels.

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

Este tipo de consulta resulta útil cuando se necesitan generar números aleatorios, por ejemplo, para simular eventos aleatorios o para introducir la aleatoriedad en el análisis de datos. En el contexto de la `squirrels` tabla, podría usarse para asignar valores aleatorios a cada ardilla, que luego podrían usarse para su posterior procesamiento o análisis.

Función RANDOM

La función RANDOM genera un valor aleatorio entre 0,0 (inclusive) y 1,0 (exclusive).

Sintaxis

```
RANDOM()
```

Tipo de devolución

RANDOM devuelve un número con un valor de DOUBLE PRECISION.

Ejemplos

1. Se computa un valor aleatorio entre 0 y 99. Si el número aleatorio está comprendido entre 0 y 1, esta consulta produce un número aleatorio comprendido entre 0 y 100:

```
select cast (random() * 100 as int);

INTEGER
------
24
(1 row)
```

2. Recupera una muestra aleatoria uniforme de 10 objetos:

```
select *
from sales
order by random()
limit 10;
```

   Ahora recupera una muestra aleatoria de 10 objetos, pero elige los objetos en proporción a sus precios. Por ejemplo, un objeto que cuesta el doble del precio de otro tendría el doble de posibilidades de aparecer en los resultados de la búsqueda:

```
select *
from sales
order by log(1 - random()) / pricepaid
limit 10;
```

3. En este ejemplo se usa el comando SET para establecer un valor SEED de modo que RANDOM genere una secuencia predecible de números.

   Primero, se devuelven tres valores enteros RANDOM sin establecer antes el valor SEED:

```
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)
```

Ahora, establezca el valor SEED en `.25` y devuelva tres números RANDOM más:

```
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)
```

Finalmente, restablezca el valor SEED a `.25` y verifique que RANDOM devuelva los mismos resultados que en las tres ejecuciones anteriores:

```
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)
```

Función ROUND

La función ROUND redondea los números hasta el valor entero o decimal más cercano.

La función ROUND puede incluir, de forma opcional, un segundo argumento como un valor entero que indique la cantidad de lugares decimales para el redondeo, sea cual sea la dirección. Cuando no se proporciona el segundo argumento, la función redondea al número entero más cercano. Cuando se especifica el segundo argumento >n, la función redondea al número más cercano con una precisión de hasta n decimales.

Sintaxis

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

Argumento

número

Un número o una expresión que toma el valor de un número. Puede ser el DECIMAL o el FLOAT8 tipo. AWS Clean Rooms puede convertir otros tipos de datos según las reglas de conversión implícitas.

integer (opcional)

Un número entero que indica la cantidad de lugares decimales para el redondeo, sea cual sea la dirección.

Tipo de devolución

ROUND devuelve el mismo tipo de datos numérico como el argumento de entrada.

Ejemplos

Se redondea la comisión pagada para una transacción dada hasta el número entero más cercano.

```
select commission, round(commission)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    28
(1 row)
```

Se redondea la comisión pagada para una transacción dada hasta el primer lugar decimal.

```
select commission, round(commission, 1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |  28.1
(1 row)
```

Para la misma consulta, se extiende la precisión en la dirección opuesta.

```
select commission, round(commission, -1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    30
(1 row)
```

Función SIGN

La función SIGN devuelve el signo (positivo o negativo) de un número. El resultado de la función SIGN es `1`, `-1` o `0`, lo que indica el signo del argumento.

Sintaxis

```
SIGN (number)
```

Argumento

número

Número o expresión que toma el valor de un número. Puede ser del DECIMALor FLOAT8 tipo. AWS Clean Rooms puede convertir otros tipos de datos según las reglas de conversión implícitas.

Tipo de devolución

SIGN devuelve el mismo tipo de datos numérico como el argumento de entrada. Si la entrada es DECIMAL, la salida es DECIMAL(1,0).

Ejemplos

Para determinar el signo de la comisión pagada por una transacción determinada a partir de la tabla SALES, utilice el siguiente ejemplo.

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

Función SIN

SIN es una función trigonométrica que devuelve el seno de un número. El valor devuelto está comprendido entre -1 y 1.

Sintaxis

```
SIN(number)
```

Argumento

número

Un número de `DOUBLE PRECISION` en radianes.

Tipo de devolución

`DOUBLE PRECISION`

Ejemplo

Para devolver el seno de `-PI`, use el siguiente ejemplo.

```
SELECT SIN(-PI());

+-------------------------+
|           sin           |
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

Función SQRT

La función SQRT devuelve la raíz cuadrada de un valor numérico. La raíz cuadrada es un número multiplicado por sí mismo para obtener el valor dado.

Sintaxis

```
SQRT (expression)
```

Argumento

expresión

La expresión debe ser un tipo de datos entero, decimal o de punto flotante. La expresión puede incluir funciones. Es posible que el sistema realice conversiones de tipos implícitos.

Tipo de devolución

SQRT devuelve un número con valor de DOUBLE PRECISION.

Ejemplos

El siguiente ejemplo devuelve la raíz cuadrada de un número.

```
select sqrt(16);

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

El siguiente ejemplo realiza una conversión de tipo implícita.

```
select sqrt('16');

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

En el ejemplo siguiente se anidan las funciones para realizar una tarea más compleja.

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

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

El siguiente ejemplo da como resultado la longitud del radio si se da el área de un círculo. Calcula el radio en pulgadas, por ejemplo, cuando se le da el área en pulgadas cuadradas. El área del ejemplo es 20.

```
select sqrt(20/pi());
```

Esto devuelve el valor 5,046265044040321.

El siguiente ejemplo devuelve la raíz cuadrada para valores COMMISSION de la tabla SALES. La columna COMMISSION es una columna DECIMAL. En este ejemplo se muestra cómo se puede utilizar la función en una consulta con una lógica condicional más compleja.

```
select sqrt(commission)
from sales where salesid < 10 order by salesid;
```

```
sqrt
------------------
10.4498803820905
3.37638860322683
7.24568837309472
5.1234753829798
...
```

La siguiente consulta devuelve la raíz cuadrada redondeada para el mismo conjunto de valores COMMISSION.

```
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
...
```

Para obtener más información sobre los datos de muestra AWS Clean Rooms, consulte Base de datos de muestra.

Función TRUNC

La función TRUNC trunca los números hasta el valor entero o decimal anterior.

La función TRUNC puede incluir, de forma opcional, un segundo argumento como un valor entero que indique la cantidad de lugares decimales para el redondeo, sea cual sea la dirección. Cuando no se proporciona el segundo argumento, la función redondea al número entero más cercano. Cuando se especifica el segundo argumento >n, la función redondea al número más cercano con una precisión de hasta >n decimales. Esta función también trunca una marca temporal y devuelve una fecha.

Sintaxis

```
TRUNC (number [ , integer ] |
timestamp )
```

Argumentos

número

Un número o una expresión que toma el valor de un número. Puede ser el tipo DECIMAL o el FLOAT8 tipo. AWS Clean Rooms puede convertir otros tipos de datos según las reglas de conversión implícitas.

integer (opcional)

Un número entero que indica la cantidad de lugares decimales de precisión, sea cual sea la dirección. Si no se proporciona un valor entero, el número se trunca como un número entero; si se especifica un número entero, el número se trunca hasta el lugar decimal especificado.

timestamp

La función también devuelve la fecha de una marca temporal. (Para devolver un valor de marca temporal con `00:00:00` como la hora, convierta el resultado de la función en una marca temporal).

Tipo de devolución

TRUNC devuelve el mismo tipo de datos como el primer argumento de entrada. Para las marcas temporales, TRUNC devuelve una fecha.

Ejemplos

Se trunca la comisión pagada para una transacción dada de ventas.

```
select commission, trunc(commission)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   111

(1 row)
```

Se trunca el mismo valor de comisión hasta el primer lugar decimal.

```
select commission, trunc(commission,1)
```

```
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 | 111.1

(1 row)
```

Se trunca la comisión con un valor negativo para el segundo argumento; `111.15` se redondea hacia abajo hasta `110`.

```
select commission, trunc(commission,-1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   110
(1 row)
```

Se devuelve la parte de fecha desde el resultado de la función SYSDATE (que devuelve una marca temporal):

```
select sysdate;

timestamp
----------------------------
2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc
------------
2011-07-21
(1 row)
```

Se aplica la función TRUNC a una columna TIMESTAMP. El tipo de retorno es una fecha.

```
select trunc(starttime) from event
order by eventid limit 1;

trunc
```

```
------------
2008-01-25
(1 row)
```

Funciones escalares

En esta sección, se describen las funciones escalares compatibles con Spark SQL. AWS Clean Rooms Una función escalar es una función que toma uno o más valores como entrada y devuelve un único valor como salida. Las funciones escalares funcionan en filas o elementos individuales y producen un único resultado para cada entrada.

Las funciones escalares, como SIZE, son diferentes de otros tipos de funciones SQL, como las funciones de agregado (count, sum, avg) y las funciones generadoras de tablas (explode, flatten). Estos otros tipos de funciones funcionan en varias filas o generan varias filas, mientras que las funciones escalares funcionan en filas o elementos individuales.

Temas

- Función SIZE

Función SIZE

La función SIZE toma una matriz, un mapa o una cadena existente como argumento y devuelve un único valor que representa el tamaño o la longitud de esa estructura de datos. No crea una nueva estructura de datos. Se utiliza para consultar y analizar las propiedades de las estructuras de datos existentes, más que para crear estructuras nuevas.

Esta función es útil para determinar el número de elementos de una matriz o la longitud de una cadena. Puede resultar especialmente útil cuando se trabaja con matrices y otras estructuras de datos en SQL, ya que permite obtener información sobre el tamaño o la cardinalidad de los datos.

Sintaxis

```
size(expr)
```

Argumentos

expr

Una expresión ARRAY, MAP o STRING.

Tipo de retorno

La función SIZE devuelve un entero.

Ejemplo

En este ejemplo, la función SIZE se aplica a la matriz `['b', 'd', 'c', 'a']` y devuelve el valor4, que es el número de elementos de la matriz.

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

En este ejemplo, la función SIZE se aplica al mapa `{'a': 1, 'b': 2}` y devuelve el valor2, que es el número de pares clave-valor del mapa.

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

En este ejemplo, la función TAMAÑO se aplica a la cadena `'hello world'` y devuelve el valor11, que es el número de caracteres de la cadena.

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

Funciones de cadena

Las funciones de cadena procesan y administran cadenas de caracteres o expresiones que tomen el valor de cadenas de caracteres. Cuando el argumento string de estas funciones es un valor literal, debe incluirse entre comillas simples. Entre los tipos de datos compatibles, se incluyen CHAR y VARCHAR.

En la sección siguiente, se proporcionan los nombres de función, la sintaxis y las descripciones para las funciones compatibles. Todos los desplazamientos en cadenas se basan en uno.

Temas

- || Operador (concatenación)
- Función BTRIM
- Función CONCAT
- Función FORMAT_STRING

- Funciones LEFT y RIGHT
- Función LENGTH
- Función LOWER
- Funciones LPAD y RPAD
- Función LTRIM
- Función POSITION
- Función REGEXP_COUNT
- Función REGEXP_INSTR
- Función REGEXP_REPLACE
- Función REGEXP_SUBSTR
- Función REPEAT
- Función REPLACE
- Función REVERSE
- Función RTRIM
- Función SPLIT
- Función SPLIT_PART
- Función SUBSTRING
- Función TRANSLATE
- Función TRIM
- Función UPPER
- Función UUID

|| Operador (concatenación)

Concatena dos expresiones a ambos extremos del símbolo || y devuelve una expresión concatenada.

El operador de concatenación es similar a Función CONCAT.

> **Note**
>
> Para la función CONCAT y el operador de concatenación, si una o ambas expresiones son nulas, el resultado de la concatenación también lo será.

Sintaxis

```
expression1 || expression2
```

Argumentos

expression1, expression2

Ambos argumentos pueden ser cadenas de caracteres o expresiones de longitud fija o variable.

Tipo de devolución

El operador || devuelve una cadena. El tipo de cadena es el mismo que los argumentos de entrada.

Ejemplo

En el siguiente ejemplo, se concatenan los campos FIRSTNAME y LASTNAME de la tabla USERS:

```
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)
```

Para concatenar columnas que puedan llegar a tener valores nulos, use la expresión Funciones NVL y COALESCE. En el siguiente ejemplo, se usa NVL para devolver un 0 siempre que se encuentre un NULL.

```
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
```

Función BTRIM

La función BTRIM recorta una cadena al eliminar espacios o caracteres a la izquierda y a la derecha que coincidan con una cadena específica opcional.

Sintaxis

```
BTRIM(string [, trim_chars ] )
```

Argumentos

string

Es la cadena VARCHAR de entrada que se va a recortar.

trim_chars

Es la cadena VARCHAR que contiene los caracteres que deben coincidir.

Tipo de devolución

La función BTRIM devuelve una cadena VARCHAR.

Ejemplos

En el siguiente ejemplo, se recortan espacios a la izquierda y a la derecha de la cadena `' abc '`:

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

En el siguiente ejemplo, se eliminan las cadenas `'xyz'` a la izquierda y a la derecha de la cadena `'xyzaxyzbxyzcxyz'`. Las coincidencias a la izquierda y a la derecha de `'xyz'` se eliminan, pero las coincidencias internas dentro de la cadena no se eliminan.

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

En el siguiente ejemplo, se eliminan las partes a la izquierda y a la derecha de la cadena `'setuphistorycassettes'` que coinciden con cualquiera de los caracteres de la lista trim_chars `'tes'`. Cualquier `t`, `e` o `s` que aparezca antes de cualquier carácter que no esté en la lista trim_chars a la izquierda o a la derecha de la cadena de entrada se eliminará.

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
-----------------
 uphistoryca
```

Función CONCAT

La función CONCAT concatena dos expresiones y devuelve la expresión resultante. Para concatenar más de dos expresiones, utilice las funciones CONCAT anidadas. El operador de concatenación (`||`) entre dos expresiones produce los mismos resultados que la función CONCAT.

> **Note**
>
> Para la función CONCAT y el operador de concatenación, si una o ambas expresiones son nulas, el resultado de la concatenación también lo será.

Sintaxis

```
CONCAT ( expression1, expression2 )
```

Argumentos

expression1, expression2

Ambos argumentos pueden consistir en una cadena de caracteres de longitud fija, una cadena de caracteres de longitud variable, una expresión binaria o una expresión que tiene como valor una de estas entradas de datos.

Tipo de devolución

CONCAT devuelve una expresión. El tipo de datos de la expresión es igual al de los argumentos de entrada.

Si las expresiones de entrada son de tipos diferentes, AWS Clean Rooms intenta escribir implícitamente convierte una de las expresiones. Si no se pueden convertir los valores, se devuelve un error.

Ejemplos

En el siguiente ejemplo, se concatenan dos literales de caracteres:

```
select concat('December 25, ', '2008');

concat
-------------------
December 25, 2008
(1 row)
```

La siguiente consulta, utilizando el operador || en lugar de CONCAT, produce el mismo resultado:

```
select 'December 25, '||'2008';
```

```
concat
-------------------
December 25, 2008
(1 row)
```

En el siguiente ejemplo, se usan dos funciones CONCAT para concatenar tres cadenas de caracteres:

```
select concat('Thursday, ', concat('December 25, ', '2008'));

concat
----------------------------
Thursday, December 25, 2008
(1 row)
```

Para concatenar columnas que puedan llegar a tener valores nulos, use Funciones NVL y COALESCE. En el siguiente ejemplo, se usa NVL para devolver un 0 siempre que se encuentre un NULL.

```
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)
```

En la siguiente consulta, se concatenan valores CITY y STATE de la tabla VENUE:

```
select concat(venuecity, venuestate)
from venue
where venueseats > 75000
order by venueseats;

concat
```

```
-------------------
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

La siguiente consulta utiliza funciones CONCAT anidadas. La consulta concatena los valores CITY y STATE de la tabla VENUE pero delimita la cadena resultado con una coma y un espacio:

```
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)
```

Función FORMAT_STRING

La función FORMAT_STRING crea una cadena formateada sustituyendo los marcadores de posición de una cadena de plantilla por los argumentos proporcionados. Devuelve una cadena formateada a partir de cadenas de formato de estilo printf.

La función FORMAT_STRING funciona sustituyendo los marcadores de posición de la cadena de la plantilla por los valores correspondientes pasados como argumentos. Este tipo de formato de cadena puede resultar útil cuando se necesitan construir cadenas de forma dinámica que incluyan una combinación de texto estático y datos dinámicos, como cuando se generan mensajes de salida, informes u otros tipos de texto informativo. La función FORMAT_STRING proporciona una forma concisa y legible de crear estos tipos de cadenas formateadas, lo que facilita el mantenimiento y la actualización del código que genera la salida.

Sintaxis

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

Argumentos

strfmt

Una expresión de cadena.

obj

Una cadena o expresión numérica.

Tipo de devolución

FORMAT_STRING devuelve una cadena.

Ejemplo

El siguiente ejemplo contiene una cadena de plantilla que contiene dos marcadores de posición: `%d` para un valor decimal (entero) y `%s` para un valor de cadena. El `%d` marcador de posición se reemplaza por el valor decimal (entero) (`100`) y el marcador de posición %s se reemplaza por el valor de cadena (). `"days"` El resultado es una cadena de plantilla en la que los marcadores de posición se sustituyen por los argumentos proporcionados:. `"Hello World 100 days"`

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

Funciones LEFT y RIGHT

Estas funciones devuelven la cantidad especificada de caracteres más a la izquierda o más a la derecha de una cadena de caracteres.

La cantidad se basa en la cantidad de caracteres, no bytes, por lo que los caracteres multibyte se cuentan como caracteres simples.

Sintaxis

```
LEFT ( string,  integer )

RIGHT ( string,  integer )
```

Argumentos

string

Cualquier cadena de caracteres o cualquier expresión que tome como valor una cadena de caracteres.

integer

Un número entero.

Tipo de devolución

LEFT y RIGHT devuelven una cadena VARCHAR.

Ejemplo

El siguiente ejemplo devuelve los 5 caracteres situados más a la izquierda y los 5 más a la derecha de los nombres de eventos que tengan IDs entre 1000 y 1005:

```
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)
```

Función LENGTH

Función LOWER

Convierte una cadena de caracteres a minúsculas. LOWER admite caracteres multibyte UTF-8 de hasta un máximo de cuatro bytes por carácter.

Sintaxis

```
LOWER(string)
```

Argumento

string

El parámetro de entrada es una cadena VARCHAR (o cualquier otro tipo de datos, como CHAR, que se pueda convertir de forma implícita a VARCHAR).

Tipo de devolución

La función LOWER devuelve una cadena de caracteres que presenta el mismo tipo de datos que la cadena de entrada.

Ejemplos

En el siguiente ejemplo, se convierte el campo CATNAME a minúsculas:

```
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)
```

Funciones LPAD y RPAD

Estas funciones anteponen o anexan caracteres a una cadena, según una longitud especificada.

Sintaxis

```
LPAD (string1, length, [ string2 ])
```

```
RPAD (string1, length, [ string2 ])
```

Argumentos

string1

Una cadena de caracteres o una expresión toma el valor de una cadena de caracteres, como el nombre de una columna de caracteres.

longitud

Un valor entero que define la longitud del resultado de la función. La longitud de una cadena se basa en la cantidad de caracteres, no bytes, por lo que los caracteres multibyte se cuentan como caracteres simples. Si string1 (cadena1) tiene una longitud mayor que la especificada, se trunca (a la derecha). Si el valor de length (longitud) es un número negativo, el resultado de la función es una cadena vacía.

string2 (cadena2)

Uno o varios caracteres que se anteponen o anexan a string1 (cadena1). Este argumento es opcional; si no se especifica, se utilizan espacios.

Tipo de devolución

Estas funciones devuelven un tipo de datos VARCHAR.

Ejemplos

Truncar un conjunto especificado de nombres de eventos a 20 caracteres y anteponga espacios a los nombres más cortos:

```
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)
```

Truncar el mismo conjunto de nombres de eventos a 20 caracteres, pero anexar `0123456789` a los nombres más cortos.

```
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)
```

Función LTRIM

Recorta los caracteres desde el principio de una cadena. Elimina la cadena más larga que contiene solo caracteres de la lista de caracteres de recorte. El recorte se completa cuando no aparece ningún carácter de recorte en la cadena de entrada.

Sintaxis

```
LTRIM( string [, trim_chars] )
```

Argumentos

string

Una columna de cadena, una expresión o un literal de cadena que se va a recortar.

trim_chars

Una columna de cadena, expresión o literal de cadena que representa los caracteres que se van a recortar desde el principio de la cadena. Si no se especifica, se utiliza un espacio como carácter de recorte.

Tipo de devolución

La función LTRIM devuelve una cadena de caracteres con el mismo tipo de datos que la cadena de entrada (CHAR o VARCHAR).

Ejemplos

En el siguiente ejemplo, se recorta el año de la columna `listime`. Los caracteres de recorte del literal de cadena `'2008-'` indican los caracteres que se recortarán desde la izquierda. Si utiliza los caracteres de recorte `'028-'`, obtendrá el mismo resultado.

```
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
```

LTRIM elimina cualquiera de los caracteres de trim_chars cuando aparecen al principio de la cadena. En el siguiente ejemplo, se recortan los caracteres «C», «D» y «G» cuando aparecen al principio de VENUENAME, que es una columna VARCHAR.

```
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
```

En el siguiente ejemplo. se utiliza el carácter de recorte 2 que se recupera de la columna `venueid`.

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

En el siguiente ejemplo, no se recorta ningún carácter porque se encuentra un 2 antes del carácter de recorte `'0'`.

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

En el siguiente ejemplo, se utiliza el carácter de recorte de espacio predeterminado y se recortan los dos espacios desde el principio de la cadena.

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

Función POSITION

Devuelve la ubicación de la subcadena especificada dentro de una cadena.

Sintaxis

```
POSITION(substring IN string )
```

Argumentos

subcadena

Subcadena que se va a buscar dentro de la cadena.

string

La cadena o columna que se buscará.

Tipo de devolución

La función POSITION devuelve un valor entero correspondiente a la posición de la subcadena (basado en 1, no basado en cero). La posición se basa en la cantidad de caracteres, no bytes, por lo que los caracteres multibyte se cuentan como caracteres simples.

Notas de uso

POSITION devuelve 0 si no se encuentra subcadena dentro de la cadena:

```
select position('dog' in 'fish');

position
----------
 0
(1 row)
```

Ejemplos

En el siguiente ejemplo, se muestra la posición de la cadena `fish` dentro de la palabra `dogfish`:

```
select position('fish' in 'dogfish');

position
----------
 4
(1 row)
```

El siguiente ejemplo devuelve la cantidad de transacciones de venta con un parámetro COMMISSION que supere los 999,00 de la tabla SALES:

```
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)
```

Función REGEXP_COUNT

Busca una cadena para un patrón de expresión regular y devuelve un valor entero que indica la cantidad de veces que el patrón aparece en la cadena. Si no se encuentra coincidencia, la función devuelve 0.

Sintaxis

```
REGEXP_COUNT ( source_string, pattern [, position [, parameters ] ] )
```

Argumentos

source_string

Una expresión de cadena, como un nombre de columna, que se buscará.

pattern

Un literal de cadena que representa un patrón de expresión regular.

position

Valor entero positivo que indica la posición dentro de source_string (cadena_de_origen) para comenzar la búsqueda. La posición se basa en la cantidad de caracteres, no bytes, por lo que los caracteres multibyte se cuentan como caracteres simples. El valor predeterminado de es 1. Si el valor de position (posición) es menor que 1, la búsqueda comienza en el primer carácter de source-string (cadena_de_origen). Si el valor de position (posición) es mayor que el número de caracteres de source-string (cadena_de_origen), el resultado es 0.

parameters

Uno o varios literales de cadena que indican el grado de coincidencia de la función con el patrón. Los valores posibles son los siguientes:

- c: aplica la coincidencia que distingue entre mayúsculas y minúsculas. El comportamiento predeterminado es utilizar la coincidencia que distingue entre mayúsculas y minúsculas.

- i: aplica la coincidencia que no distingue entre mayúsculas y minúsculas.
- p: interpreta el patrón con el dialecto de expresión regular compatible con Perl (PCRE).

Tipo de devolución

Entero

Ejemplo

En el siguiente ejemplo, se cuenta la cantidad de veces en que aparece una secuencia de tres letras.

```
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');

 regexp_count
 --------------
            8
```

En el siguiente ejemplo, se cuenta la cantidad de veces en que el nombre del dominio de nivel superior es `org` o `edu`.

```
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
```

En el siguiente ejemplo, se cuenta cuántas veces aparece la cadena `FOX`, con una coincidencia que no distingue entre mayúsculas y minúsculas.

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

 regexp_count
 --------------
            1
```

En el siguiente ejemplo, se utiliza un patrón escrito en el dialecto de PCRE para localizar palabras que contengan al menos un número y una letra en minúsculas. Se utiliza el operador `?=`, que tiene

una connotación específica de anticipación en PCRE. En este ejemplo, se cuenta cuántas veces aparecen dichas palabras, con una coincidencia que distingue entre mayúsculas y minúsculas.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

 regexp_count
 --------------
            2
```

En el siguiente ejemplo, se utiliza un patrón escrito en el dialecto de PCRE para localizar palabras que contengan al menos un número y una letra en minúsculas. Se utiliza el operador ?=, que tiene una connotación específica en PCRE. En este ejemplo, se cuenta cuántas veces aparecen dichas palabras, pero difiere del ejemplo anterior, ya que se utiliza una coincidencia sin distinción entre mayúsculas y minúsculas.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');

 regexp_count
 --------------
            3
```

Función REGEXP_INSTR

Busca una cadena para un patrón de expresión regular y devuelve un valor entero que indica la posición de inicio o de finalización de la subcadena coincidente. Si no se encuentra coincidencia, la función devuelve 0. REGEXP_INSTR es similar a la función POSITION, pero le permite buscar un patrón de expresión regular en una cadena.

Sintaxis

```
REGEXP_INSTR ( source_string, pattern [, position [, occurrence] [, option
 [, parameters ] ] ] ] )
```

Argumentos

source_string

Una expresión de cadena, como un nombre de columna, que se buscará.

pattern

Un literal de cadena que representa un patrón de expresión regular.

position

Valor entero positivo que indica la posición dentro de source_string (cadena_de_origen) para comenzar la búsqueda. La posición se basa en la cantidad de caracteres, no bytes, por lo que los caracteres multibyte se cuentan como caracteres simples. El valor predeterminado de es 1. Si el valor de position (posición) es menor que 1, la búsqueda comienza en el primer carácter de source-string (cadena_de_origen). Si el valor de position (posición) es mayor que el número de caracteres de source-string (cadena_de_origen), el resultado es 0.

occurrence

Un número entero positivo que indica qué coincidencia del patrón se va a utilizar. REGEXP_INSTR omite las primeras coincidencias especificadas por el valor de occurrence menos uno. El valor predeterminado de es 1. Si occurrence es menor que 1 o mayor que el número de caracteres de source_string, la búsqueda se omite y el resultado es 0.

option

Valor que indica si se va a devolver la posición del primer carácter de la coincidencia (`0`) o la posición del primer carácter situado a continuación del final de la coincidencia (`1`). Un valor distinto de cero es lo mismo que 1. El valor predeterminado es 0.

parameters

Uno o varios literales de cadena que indican el grado de coincidencia de la función con el patrón. Los valores posibles son los siguientes:

- c: aplica la coincidencia que distingue entre mayúsculas y minúsculas. El comportamiento predeterminado es utilizar la coincidencia que distingue entre mayúsculas y minúsculas.
- i: aplica la coincidencia que no distingue entre mayúsculas y minúsculas.
- e: extrae una subcadena mediante una subexpresión.

  Si pattern incluye una subexpresión, REGEXP_INSTR realiza la comparación con una subcadena utilizando la primera subexpresión de pattern. REGEXP_INSTR solo tiene en cuenta la primera subexpresión; las subexpresiones adicionales se omiten. Si el patrón no incluye una subexpresión, REGEXP_INSTR omite el parámetro 'e'.
- p: interpreta el patrón con el dialecto de expresión regular compatible con Perl (PCRE).

Tipo de devolución

Entero

Ejemplo

En el siguiente ejemplo, se busca el carácter @ que comience un nombre de dominio y se devuelve la posición inicial de la primera coincidencia.

```
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
```

En el siguiente ejemplo, se buscan variantes de la palabra `Center` y se devuelve la posición inicial de la primera coincidencia.

```
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
```

En el siguiente ejemplo, se encuentra la posición inicial de la primera vez que aparece la cadena `FOX`, con una lógica que no distingue entre mayúsculas y minúsculas.

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

 regexp_instr
```

```
--------------
           5
```

En el siguiente ejemplo, se utiliza un patrón escrito en el dialecto de PCRE para localizar palabras que contengan al menos un número y una letra en minúsculas. Se utiliza el operador ?=, que tiene una connotación específica de anticipación en PCRE. En este ejemplo, se encuentra la posición inicial de la segunda palabra que reúne esas características.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

 regexp_instr
 --------------
           21
```

En el siguiente ejemplo, se utiliza un patrón escrito en el dialecto de PCRE para localizar palabras que contengan al menos un número y una letra en minúsculas. Se utiliza el operador ?=, que tiene una connotación específica de anticipación en PCRE. En este ejemplo, se encuentra la posición inicial de la segunda palabra que reúne esas características, pero difiere del ejemplo anterior, ya que se utiliza una coincidencia sin distinción entre mayúsculas y minúsculas.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');

 regexp_instr
 --------------
           15
```

Función REGEXP_REPLACE

Busca una cadena para un patrón de expresión regular y reemplaza cada coincidencia del patrón con una cadena especificada. REGEXP_REPLACE es similar a Función REPLACE, pero le permite buscar un patrón de expresión regular en una cadena.

REGEXP_REPLACE es similar a Función TRANSLATE y a Función REPLACE, salvo que TRANSLATE realiza varias sustituciones de caracteres únicos y REPLACE sustituye una cadena entera por otra cadena, mientras que REGEXP_REPLACE le permite buscar un patrón de expresión regular en una cadena.

Sintaxis

```
REGEXP_REPLACE ( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

Argumentos

source_string

Una expresión de cadena, como un nombre de columna, que se buscará.

pattern

Un literal de cadena que representa un patrón de expresión regular.

replace_string

Una expresión de cadena, como un nombre de columna, que reemplazará cada coincidencia del patrón. El valor predeterminado es una cadena vacía ("").

position

Valor entero positivo que indica la posición dentro de source_string (cadena_de_origen) para comenzar la búsqueda. La posición se basa en la cantidad de caracteres, no bytes, por lo que los caracteres multibyte se cuentan como caracteres simples. El valor predeterminado de es 1. Si el valor de position (posición) es menor que 1, la búsqueda comienza en el primer carácter de source-string (cadena_de_origen). Si el valor de position (posición) es mayor que la cantidad de caracteres de source-string (cadena_de_origen), el resultado es source_string (cadena_de_origen).

parameters

Uno o varios literales de cadena que indican el grado de coincidencia de la función con el patrón. Los valores posibles son los siguientes:

- c: aplica la coincidencia que distingue entre mayúsculas y minúsculas. El comportamiento predeterminado es utilizar la coincidencia que distingue entre mayúsculas y minúsculas.
- i: aplica la coincidencia que no distingue entre mayúsculas y minúsculas.
- p: interpreta el patrón con el dialecto de expresión regular compatible con Perl (PCRE).

Tipo de devolución

VARCHAR

Si el valor de pattern o la replace_string es NULL, el valor devuelto es NULL.

Ejemplo

En el siguiente ejemplo, se elimina el @ y el nombre de dominio de direcciones de correo electrónico.

```
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
```

En el siguiente ejemplo, se reemplazan los nombres de dominio de las direcciones de email con este valor: `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
```

En el siguiente ejemplo, se reemplazan todas las veces que aparece la cadena FOX en el valor `quick brown fox`, con una coincidencia que no distingue entre mayúsculas y minúsculas.

```
SELECT regexp_replace('the fox', 'FOX', 'quick brown fox', 1, 'i');

   regexp_replace
---------------------
```

```
 the quick brown fox
```

En el siguiente ejemplo, se utiliza un patrón escrito en el dialecto de PCRE para localizar palabras que contengan al menos un número y una letra en minúsculas. Se utiliza el operador ?=, que tiene una connotación específica de anticipación en PCRE. En este ejemplo, se reemplaza cada vez que aparece una palabra que reúne esas características con el valor [hidden].

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

       regexp_replace
------------------------------
 [hidden] plain A1234 [hidden]
```

En el siguiente ejemplo, se utiliza un patrón escrito en el dialecto de PCRE para localizar palabras que contengan al menos un número y una letra en minúsculas. Se utiliza el operador ?=, que tiene una connotación específica de anticipación en PCRE. En este ejemplo, se reemplaza cada vez que aparece una palabra que reúne esas características con el valor [hidden], pero difiere del ejemplo anterior, ya que se utiliza una coincidencia sin distinción entre mayúsculas y minúsculas.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

         regexp_replace
---------------------------------
 [hidden] plain [hidden] [hidden]
```

Función REGEXP_SUBSTR

Devuelve los caracteres de una cadena al buscar un patrón de expresión regular. REGEXP_SUBSTR es similar a la función Función SUBSTRING, pero le permite buscar un patrón de expresión regular en una cadena. Si la función no puede hacer coincidir la expresión regular con ningún carácter de la cadena, devuelve una cadena vacía.

Sintaxis

```
REGEXP_SUBSTR ( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

Argumentos

source_string

Una expresión de cadena que se va a buscar.

pattern

Un literal de cadena que representa un patrón de expresión regular.

position

Valor entero positivo que indica la posición dentro de source_string (cadena_de_origen) para comenzar la búsqueda. La posición se basa en la cantidad de caracteres, no bytes, por lo que los caracteres multibyte se cuentan como caracteres simples. El valor predeterminado de es 1. Si el valor de position (posición) es menor que 1, la búsqueda comienza en el primer carácter de source-string (cadena_de_origen). Si el valor de position (posición) es mayor que el número de caracteres de source-string (cadena_de_origen), el resultado es una cadena vacía ("").

occurrence

Un número entero positivo que indica qué coincidencia del patrón se va a utilizar. REGEXP_SUBSTR omite las primeras coincidencias especificadas por el valor de occurrence menos uno. El valor predeterminado de es 1. Si occurrence es menor que 1 o mayor que el número de caracteres de source_string, la búsqueda se omite y el resultado es NULL.

parameters

Uno o varios literales de cadena que indican el grado de coincidencia de la función con el patrón. Los valores posibles son los siguientes:

- c: aplica la coincidencia que distingue entre mayúsculas y minúsculas. El comportamiento predeterminado es utilizar la coincidencia que distingue entre mayúsculas y minúsculas.
- i: aplica la coincidencia que no distingue entre mayúsculas y minúsculas.
- e: extrae una subcadena mediante una subexpresión.

  Si pattern incluye una subexpresión, REGEXP_SUBSTR realiza la comparación con una subcadena utilizando la primera subexpresión de pattern. Una subexpresión es una expresión dentro del patrón que está entre paréntesis. Por ejemplo, para que el patrón `'This is a (\\w+)'` coincida con la primera expresión con la cadena `'This is a '` seguida de una palabra. En lugar de devolver el patrón, REGEXP_SUBSTR con el parámetro e devuelve solo la cadena dentro de la subexpresión.

REGEXP_SUBSTR solo tiene en cuenta la primera subexpresión; las subexpresiones adicionales se omiten. Si el patrón no incluye una subexpresión, REGEXP_SUBSTR omite el parámetro 'e'.

- p: interpreta el patrón con el dialecto de expresión regular compatible con Perl (PCRE).

Tipo de devolución

VARCHAR

Ejemplo

El siguiente ejemplo devuelve la parte de una dirección de correo electrónico entre el carácter @ y la extensión de dominio.

```
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
```

El siguiente ejemplo devuelve la parte de la entrada que corresponde a la primera vez que aparece la cadena F0X, con una coincidencia que no distingue entre mayúsculas y minúsculas.

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

 regexp_substr
---------------
 fox
```

El ejemplo siguiente devuelve la primera parte de la entrada que comienza en minúscula. Esto es funcionalmente idéntico a la misma instrucción SELECT sin el parámetro c.

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');
```

```
 regexp_substr
---------------
 abc
```

En el siguiente ejemplo, se utiliza un patrón escrito en el dialecto de PCRE para localizar palabras que contengan al menos un número y una letra en minúsculas. Se utiliza el operador ?=, que tiene una connotación específica de anticipación en PCRE. En este ejemplo, se devuelve la parte de la entrada que corresponde a la segunda palabra que reúne esas características.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

En el siguiente ejemplo, se utiliza un patrón escrito en el dialecto de PCRE para localizar palabras que contengan al menos un número y una letra en minúsculas. Se utiliza el operador ?=, que tiene una connotación específica de anticipación en PCRE. En este ejemplo, se devuelve la parte de la entrada que corresponde a la segunda palabra que reúne esas características, pero difiere del ejemplo anterior, ya que se utiliza una coincidencia sin distinción entre mayúsculas y minúsculas.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
---------------
 A1234
```

En el ejemplo siguiente se utiliza una subexpresión para buscar la segunda cadena que coincida con el patrón `'this is a (\\w+)'` con una coincidencia que no distingue entre mayúsculas y minúsculas. Devuelve la subexpresión entre paréntesis.

```
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
```

Función REPEAT

Repite una cadena la cantidad especificada de veces. Si el parámetro de entrada es numérico, REPEAT lo trata como una cadena.

Sintaxis

```
REPEAT(string, integer)
```

Argumentos

string

El primer parámetro de entrada es la cadena que se repetirá.

integer

El segundo parámetro es un valor entero que indica la cantidad de veces que se repite la cadena.

Tipo de devolución

La función REPEAT devuelve una cadena.

Ejemplos

En el siguiente ejemplo, se repite tres veces el valor de la columna CATID en la tabla CATEGORY:

```
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)
```

Función REPLACE

Reemplaza todas las coincidencias de un conjunto de caracteres dentro de una cadena existente con otros caracteres especificados.

REPLACE es similar a Función TRANSLATE y a Función REGEXP_REPLACE, salvo que TRANSLATE realiza varias sustituciones de caracteres únicos y REGEXP_REPLACE le permite buscar un patrón de expresión regular en una cadena, mientras que REPLACE sustituye una cadena entera por otra cadena.

Sintaxis

```
REPLACE(string1, old_chars, new_chars)
```

Argumentos

string

Cadena CHAR o VARCHAR que se buscará

old_chars

Cadena CHAR o VARCHAR que se reemplazará.

new_chars

Nueva cadena CHAR o VARCHAR que reemplaza a old_string (cadena_anterior).

Tipo de devolución

VARCHAR

Si old_chars o new_chars es NULL, el valor devuelto es NULL.

Ejemplos

En el siguiente ejemplo, se convierte la cadena `Shows` en `Theatre` en el campo CATGROUP:

```
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)
```

Función REVERSE

La función REVERSE opera en una cadena y devuelve los caracteres en orden inverso. Por ejemplo, `reverse('abcde')` devuelve `edcba`. Esta función trabaja sobre tipos de datos numéricos y de fecha, además de tipos de datos de caracteres; no obstante, en la mayoría de los casos, tiene valor práctico para las cadenas de caracteres.

Sintaxis

```
REVERSE ( expression )
```

Argumento

expresión

Una expresión con un tipo de datos de carácter, fecha, marca temporal o número que representa el destino de la reversión de carácter. Todas las expresiones se convierten implícitamente a cadenas de caracteres de longitud variable. Se ignoran los espacios a la derecha en cadenas de caracteres de ancho fijo.

Tipo de devolución

REVERSE devuelve un VARCHAR.

Ejemplos

Seleccione cinco nombres distintos de ciudades y sus correspondientes nombres invertidos de la tabla USERS:

```
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)
```

Seleccione cinco números de venta IDs y su correspondiente distribución invertida IDs como cadenas de caracteres:

```
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)
```

Función RTRIM

La función RTRIM recorta un conjunto especificado de caracteres desde el final de una cadena. Elimina la cadena más larga que contiene solo caracteres de la lista de caracteres de recorte. El recorte se completa cuando un carácter de recorte no aparece en la cadena de entrada.

Sintaxis

```
RTRIM( string, trim_chars )
```

Argumentos

string

Una columna de cadena, una expresión o un literal de cadena que se va a recortar.

trim_chars

Es una columna de cadena, expresión o literal de cadena que representa los caracteres que se deben recortar desde el final de string. Si no se especifica, se utiliza un espacio como carácter de recorte.

Tipo de devolución

Cadena que es del mismo tipo de datos que el argumento string.

Ejemplo

En el siguiente ejemplo, se recortan espacios a la izquierda y a la derecha de la cadena `' abc '`:

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

En el siguiente ejemplo, se eliminan las cadenas `'xyz'` a la derecha de la cadena `'xyzaxyzbxyzcxyz'`. Las coincidencias a la derecha de `'xyz'` se eliminan, pero las coincidencias internas dentro de la cadena no se eliminan.

```
select 'xyzaxyzbxyzcxyz' as untrim,
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

En el siguiente ejemplo, se eliminan las partes a la derecha de la cadena `'setuphistorycassettes'` que coinciden con cualquiera de los caracteres de la lista trim_chars `'tes'`. Cualquier `t`, `e` o `s` que aparezca antes de cualquier carácter que no esté en la lista trim_chars al final de la cadena de entrada se eliminará.

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

En el siguiente ejemplo, se recortan los caracteres "Park" del final de VENUENAME, cuando corresponde:

```
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
```

Tenga en cuenta que RTRIM elimina cualquiera de los caracteres `P`, `a`, `r` o `k` cuando aparecen al final de un VENUENAME.

Función SPLIT

La función SPLIT permite extraer subcadenas de una cadena más grande y trabajar con ellas como una matriz. La función DIVIDIR resulta útil cuando se necesita dividir una cadena en componentes individuales en función de un patrón o delimitador específico.

Sintaxis

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

Argumentos

estrella

Una expresión de cadena para dividir.

regex

Una cadena que representa una expresión regular. La cadena de expresiones regulares debe ser una expresión regular de Java.

limit

Una expresión entera que controla el número de veces que se aplica la expresión regular.

- límite **>** 0: la longitud de la matriz resultante no superará el límite y la última entrada de la matriz resultante contendrá todas las entradas más allá de la última expresión regular coincidente.
- límite **<=** 0: la expresión regular se aplicará tantas veces como sea posible y la matriz resultante puede ser de cualquier tamaño.

Tipo de devolución

<STRING**>**La función DIVIDIR devuelve una MATRIZ.

Si`limit > 0`: la longitud de la matriz resultante no superará el límite y la última entrada de la matriz resultante contendrá todas las entradas más allá de la última expresión regular coincidente.

Si`limit <= 0`: la expresión regular se aplicará tantas veces como sea posible y la matriz resultante puede ser de cualquier tamaño.

Ejemplo

En este ejemplo, la función SPLIT divide la cadena de entrada `'oneAtwoBthreeC'` siempre que encuentre los caracteres `'A'` o `'C'` (según lo especificado en el patrón de expresiones regulares). `'B' '[ABC]'` El resultado es una matriz de cuatro elementos:`"one"`, `"two"``"three"`, y una cadena `""` vacía.

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

Función SPLIT_PART

Divide una cadena en el delimitador especificado y devuelve la parte en la posición especificada.

Sintaxis

```
SPLIT_PART(string, delimiter, position)
```

Argumentos

string

Es una columna de cadena, una expresión o un literal de cadena que se va a dividir. La cadena puede ser CHAR o VARCHAR.

delimiter

Es la cadena delimitadora que indica las secciones del string de entrada.

Si el delimitador es un literal, enciérrelo entre comillas simples.

position

Posición de la porción de string a devolver (contando desde 1). Debe ser un número entero mayor que 0. Si position es mayor que la cantidad de porciones de la cadena, SPLIT_PART devuelve una cadena vacía. Si no se encuentra el delimitador en cadena, entonces el valor devuelto contiene el contenido de la parte especificado, que podría ser la cadena completa o un valor vacío.

Tipo de devolución

Una cadena CHAR o VARCHAR, igual que el parámetro string.

Ejemplos

En el siguiente ejemplo, se divide un literal de cadena en partes mediante el uso del delimitador $ que devuelve la segunda parte.

```
select split_part('abc$def$ghi','$',2)

split_part
----------
```

```
def
```

En el siguiente ejemplo, se divide un literal de cadena en partes mediante el uso del delimitador $ que devuelve la segunda parte. Devuelve una cadena vacía porque no se encuentra la parte 4.

```
select split_part('abc$def$ghi','$',4)

split_part
----------
```

En el siguiente ejemplo, se divide un literal de cadena en partes mediante el uso del delimitador # que devuelve la segunda parte. Devuelve la cadena completa, que es la primera parte, porque no se encuentra el delimitador.

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

En el siguiente ejemplo, se divide el campo de la marca temporal LISTTIME entre los componentes de año, mes y día.

```
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
```

En el siguiente ejemplo, se selecciona el campo de la marca temporal LISTTIME y se lo divide teniendo en cuenta el carácter `'-'` para obtener el mes (la segunda parte de la cadena LISTTIME). Luego, se cuenta la cantidad de entradas para cada mes:

```
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
```

Función SUBSTRING

Devuelve el subconjunto de una cadena basado en la posición inicial especificada.

Si la entrada es una cadena de caracteres, la posición inicial y el número de caracteres extraídos se basan en caracteres, y no en bytes, de modo tal que los caracteres de varios bytes se cuentan como si fueran simples. Si la entrada es una expresión binaria, la posición inicial y la subcadena extraída se basan en bytes. No puede especificar una longitud negativa, pero puede especificar una posición de inicio negativa.

Sintaxis

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

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

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

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

Argumentos

cadena de caracteres

La cadena que se buscará. Los tipos de datos que no son caracteres se tratan como una cadena.

start_position

La posición dentro de la cadena para comenzar la extracción, comenzando en 1. En valor de start_position (posición_de_inicio) se basa en la cantidad de caracteres, no bytes, por lo que los caracteres multibyte se cuentan como caracteres simples. Este número puede ser negativo.

numerar caracteres

La cantidad de caracteres para extraer (la longitud de la subcadena). El número de caracteres se basa en el número de caracteres, no en bytes, de modo que los caracteres de varios bytes se cuentan como caracteres individuales. Este número no puede ser negativo.

start_byte

La posición dentro de la expresión binaria desde donde comienza la extracción, con punto de partida en 1. Este número puede ser negativo.

bytes numéricos

La cantidad de bytes para extraer, es decir, la longitud de la subcadena. Este número no puede ser negativo.

Tipo de devolución

VARCHAR

Notas de uso de cadenas de caracteres

El siguiente ejemplo devuelve una cadena de cuatro caracteres comenzando con el sexto carácter.

```
select substring('caterpillar',6,4);
substring
-----------
pill
(1 row)
```

Si la posición inicial + el número de caracteres supera la longitud de la cadena, SUBSTRING devuelve una subcadena desde la posición inicial hasta el final de la cadena. Por ejemplo:

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

Si `start_position` es negativo o 0, la función SUBSTRING devuelve una cadena que comienza en el primer carácter de la cadena con una longitud de `start_position` + `numbecharacters` -1. Por ejemplo:

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

Si `start_position` + `numbecharacters` -1 es menor o igual a cero, SUBSTRING devuelve una cadena vacía. Por ejemplo:

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

Ejemplos

El siguiente ejemplo devuelve el mes de la cadena LISTTIME en la tabla LISTING:

```
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)
```

El siguiente ejemplo es igual al anterior, pero utiliza la opción FROM...FOR:

```
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)
```

No se puede utilizar SUBSTRING para extraer de forma predecible el prefijo de una cadena que pueda contener caracteres de varios bytes, ya que es necesario especificar la longitud de una cadena de varios bytes en función de la cantidad de bytes, y no de la cantidad de caracteres. Para extraer el segmento inicial de una cadena en función de la longitud en bytes, puede utilizar CAST y convertir la cadena en VARCHAR(byte_length) para truncarla, donde byte_length es la longitud requerida. En el siguiente ejemplo, se extraen los 5 primeros bytes de la cadena `'Fourscore and seven'`.

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
```

```
Fours
```

El ejemplo siguiente devuelve el nombre Ana que aparece después del último espacio de la cadena de entrada `Silva, Ana`.

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))

 reverse
-----------
 Ana
```

Función TRANSLATE

Para una expresión dada, reemplaza todas las coincidencias de caracteres especificados con sustitutos especificados. Los caracteres existentes se asignan a caracteres de reemplazo en función de su posición en los argumentos characters_to_replace y characters_to_substitute. Si se especifican más caracteres en el argumento characters_to_replace que en el argumento characters_to_substitute, los caracteres adicionales del argumento characters_to_replace se omiten en el valor devuelto.

TRANSLATE es similar a Función REPLACE y a Función REGEXP_REPLACE, salvo que REPLACE sustituye una cadena entera por otra cadena y REGEXP_REPLACE le permite buscar un patrón de expresión regular en una cadena para, mientras que TRANSLATE realiza varias sustituciones de caracteres únicos.

Si un argumento es nulo, el valor de retorno es NULL.

Sintaxis

```
TRANSLATE ( expression, characters_to_replace, characters_to_substitute )
```

Argumentos

expresión

La expresión que se traducirá.

characters_to_replace

Una cadena que tiene los caracteres que se reemplazarán.

characters_to_substitute

Una cadena que tiene los caracteres que se sustituirán.

Tipo de devolución

VARCHAR

Ejemplos

En el siguiente ejemplo, se reemplazan varios caracteres en una cadena:

```
select translate('mint tea', 'inea', 'osin');

translate
-----------
most tin
```

En el siguiente ejemplo, se reemplaza el signo (@) con un punto para todos los valores en una columna:

```
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
```

En el siguiente ejemplo, se reemplazan espacios con guiones bajos y se quitan los puntos para todos los valores en una columna:

```
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
```

Función TRIM

Recorta una cadena al eliminar espacios o caracteres a la izquierda y a la derecha que coincidan con una cadena específica opcional.

Sintaxis

```
TRIM( [ BOTH ] [ trim_chars FROM ] string
```

Argumentos

trim_chars

(Opcional) Los caracteres que se recortarán de la cadena. Si se omite este parámetro, se recortan los espacios en blanco.

string

La cadena que se recortará.

Tipo de devolución

La función TRIM devuelve una cadena VARCHAR o CHAR. Si utiliza la función TRIM con un comando SQL, convierte implícitamente los resultados en VARCHAR. AWS Clean Rooms Si utiliza la función TRIM de la lista SELECT para una función SQL, AWS Clean Rooms no convierte los resultados de forma implícita y es posible que necesite realizar una conversión explícita para evitar un error de discordancia en los tipos de datos. Consulte la Función CAST función para obtener información sobre las conversiones explícitas.

Ejemplo

En el siguiente ejemplo, se recortan espacios a la izquierda y a la derecha de la cadena `' abc '`:

```
select '     abc    ' as untrim, trim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

En el siguiente ejemplo, se eliminan las comillas dobles que rodean la cadena `"dog"`:

```
select trim('"' FROM '"dog"');

btrim
-------
dog
```

TRIM elimina cualquiera de los caracteres de trim_chars cuando aparecen al principio del string. En el siguiente ejemplo, se recortan los caracteres «C», «D» y «G» cuando aparecen al principio de VENUENAME, que es una columna VARCHAR.

```
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
```

Función UPPER

Convierte una cadena a mayúsculas. UPPER admite caracteres multibyte UTF-8 de hasta un máximo de cuatro bytes por carácter.

Sintaxis

```
UPPER(string)
```

Argumentos

string

El parámetro de entrada es una cadena VARCHAR (o cualquier otro tipo de datos, como CHAR, que se pueda convertir de forma implícita a VARCHAR).

Tipo de devolución

La función UPPER devuelve una cadena de caracteres que tiene el mismo tipo de datos que la cadena de entrada.

Ejemplos

El siguiente ejemplo convierte el campo CATNAME a mayúsculas:

```
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)
```

Función UUID

La función UUID genera un identificador único universal (UUID).

UUIDs son identificadores únicos a nivel mundial que se utilizan habitualmente para proporcionar identificadores únicos con diversos fines, como:

- Identificar registros de bases de datos u otras entidades de datos.
- Generar nombres o claves únicos para archivos, directorios u otros recursos.
- Rastrear y correlacionar datos en sistemas distribuidos.
- Proporcionar identificadores únicos para paquetes de red, componentes de software u otros activos digitales.

La función UUID genera un valor UUID que es único con una probabilidad muy alta, incluso en sistemas distribuidos y durante largos períodos de tiempo. UUIDs se generan normalmente mediante una combinación de la marca de tiempo actual, la dirección de red del ordenador y otros datos aleatorios o pseudoaleatorios, lo que garantiza que es muy poco probable que cada UUID generado entre en conflicto con cualquier otro UUID.

En el contexto de una consulta SQL, la función UUID se puede utilizar para generar identificadores únicos para los nuevos registros que se insertan en una base de datos, o para proporcionar claves únicas para la partición de datos, la indexación u otros fines en los que se requiera un identificador único.

> **Note**
>
> La función UUID no es determinista.

Sintaxis

```
uuid()
```

Argumentos

La función UUID no admite ningún argumento.

Tipo de devolución

El UUID devuelve una cadena de identificador único universal (UUID). El valor se devuelve como una cadena canónica de 36 caracteres del UUID.

Ejemplo

El siguiente ejemplo genera un identificador único universal (UUID). El resultado es una cadena de 36 caracteres que representa un identificador único universal.

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

Funciones relacionadas con la privacidad

AWS Clean Rooms proporciona funciones que le ayudan a cumplir con las normas relacionadas con la privacidad en relación con las siguientes especificaciones.

- Plataforma de privacidad global (GPP): especificación de la Oficina de Publicidad Interactiva (IAB) que establece un marco global y estandarizado para la privacidad en línea y el uso de los datos. Para obtener más información sobre las especificaciones técnicas de la GPP, consulte la documentación de la Plataforma de privacidad global en. GitHub
- Marco de transparencia y consentimiento (TCF): un componente clave de la GPP, lanzada en 2020, que proporciona un marco técnico estandarizado para ayudar a las empresas a cumplir con las normas de privacidad, como el Reglamento General de Protección de Datos (GDPR) de la UE. El TCF permite a los clientes conceder o denegar su consentimiento a la recopilación y el procesamiento de datos. Para obtener más información sobre las especificaciones técnicas del TCF, consulte la documentación del TCF en. GitHub

Temas

- función consent_gpp_v1_decode
- función consent_tcf_v2_decode

función consent_gpp_v1_decode

La `consent_gpp_v1_decode` función se utiliza para decodificar los datos de consentimiento de la versión 1 de la Global Privacy Platform (GPP). Toma la cadena de consentimiento codificada como entrada y devuelve los datos de consentimiento decodificados, que incluyen información sobre las preferencias de privacidad y las opciones de consentimiento del usuario. Esta función resulta útil cuando se trabaja con datos que incluyen información de consentimiento de la GPP v1, ya que permite acceder a los datos de consentimiento y analizarlos en un formato estructurado.

Sintaxis

```
consent_gpp_v1_decode(gpp_string)
```

Argumentos

gpp_string

La cadena de consentimiento codificada del GPP v1.

Devuelve

El diccionario devuelto incluye los siguientes pares clave-valor:

- `version`: La versión de la especificación GPP utilizada (actualmente 1).
- `cmpId`: el ID de la plataforma de gestión del consentimiento (CMP) que codificó la cadena de consentimiento.
- `cmpVersion`: la versión de la CMP que codificó la cadena de consentimiento.
- `consentScreen`: el ID de la pantalla de la interfaz de usuario de la CMP en la que el usuario dio su consentimiento.
- `consentLanguage`: El código de idioma de la información de consentimiento.
- `vendorListVersion`: La versión de la lista de proveedores utilizada.
- `publisherCountryCode`: El código de país del editor.

- `purposeConsent`: una lista de números enteros que representan los fines para los que el usuario ha dado su consentimiento.
- `purposeLegitimateInterest`: Una lista de propósitos IDs para los que se ha comunicado de forma transparente el interés legítimo del usuario.
- `specialFeatureOptIns`: una lista de números enteros que representan las funciones especiales que el usuario ha elegido.
- `vendorConsent`: una lista de proveedores a los IDs que el usuario ha dado su consentimiento.
- `vendorLegitimateInterest`: una lista de proveedores IDs para los que se ha comunicado de forma transparente el interés legítimo del usuario.

Ejemplo

El siguiente ejemplo utiliza un único argumento, que es la cadena de consentimiento codificada. Devuelve un diccionario que contiene los datos de consentimiento decodificados, incluida información sobre las preferencias de privacidad del usuario, las opciones de consentimiento y otros metadatos.

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

La estructura básica de los datos de consentimiento devueltos incluye información sobre la versión de la cadena de consentimiento, los detalles de la CMP (plataforma de gestión del consentimiento), las opciones de consentimiento e interés legítimo del usuario para los distintos fines y proveedores, y otros metadatos.

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

función consent_tcf_v2_decode

La `consent_tcf_v2_decode` función se utiliza para decodificar los datos de consentimiento del Marco de Transparencia y Consentimiento (TCF) v2. Toma la cadena de consentimiento codificada como entrada y devuelve los datos de consentimiento decodificados, que incluyen información sobre las preferencias de privacidad y las opciones de consentimiento del usuario. Esta función resulta útil cuando se trabaja con datos que incluyen información de consentimiento según el TCF v2, ya que permite acceder a los datos de consentimiento y analizarlos en un formato estructurado.

Sintaxis

```
consent_tcf_v2_decode(tcf_string)
```

Argumentos

tcf_string

La cadena de consentimiento codificada del TCF v2.

Devuelve

La `consent_tcf_v2_decode` función devuelve un diccionario que contiene los datos de consentimiento decodificados de una cadena de consentimiento del Marco de Transparencia y Consentimiento (TCF) v2.

El diccionario devuelto incluye los siguientes pares clave-valor:

Segmento principal

- `version`: La versión de la especificación TCF utilizada (actualmente 2).
- `created`: La fecha y la hora en que se creó la cadena de consentimiento.
- `lastUpdated`: la fecha y la hora en que se actualizó por última vez la cadena de consentimiento.
- `cmpId`: el ID de la plataforma de gestión del consentimiento (CMP) que codificó la cadena de consentimiento.
- `cmpVersion`: la versión de la CMP que codificó la cadena de consentimiento.
- `consentScreen`: el ID de la pantalla de la interfaz de usuario de la CMP en la que el usuario dio su consentimiento.

- `consentLanguage`: El código de idioma de la información de consentimiento.
- `vendorListVersion`: La versión de la lista de proveedores utilizada.
- `tcfPolicyVersion`: La versión de la política del TCF en la que se basa la cadena de consentimiento.
- `isServiceSpecific`: un valor booleano que indica si el consentimiento es específico de un servicio en particular o se aplica a todos los servicios.
- `useNonStandardStacks`: un valor booleano que indica si se utilizan pilas no estándar.
- `specialFeatureOptIns`: una lista de números enteros que representan las funciones especiales que el usuario ha elegido.
- `purposeConsent`: una lista de números enteros que representan los fines para los que el usuario ha dado su consentimiento.
- `purposesLITransparency`: una lista de números enteros que representan los fines para los que el usuario ha dado transparencia a sus intereses legítimos.
- `purposeOneTreatment`: Un valor booleano que indica si el usuario ha solicitado el «tratamiento con un único propósito» (es decir, todos los fines se tratan por igual).
- `publisherCountryCode`: el código de país del editor.
- `vendorConsent`: una lista de proveedores a los IDs que el usuario ha dado su consentimiento.
- `vendorLegitimateInterest`: una lista de proveedores IDs para los que se ha comunicado de forma transparente el interés legítimo del usuario.
- `pubRestrictionEntry`: una lista de restricciones para editores. Este campo contiene el identificador de propósito, el tipo de restricción y la lista de proveedores IDs sujetos a esa restricción de propósito.

Segmento de proveedores divulgado

- `disclosedVendors`: una lista de números enteros que representan los proveedores y que se ha revelado al usuario.

Segmento de fines editoriales

- `pubPurposesConsent`: una lista de números enteros que representan los fines específicos del editor para los que el usuario ha dado su consentimiento.
- `pubPurposesLITransparency`: una lista de números enteros que representan los fines específicos del editor para los que el usuario ha expresado su interés legítimo en la transparencia.

- `customPurposesConsent`: una lista de números enteros que representan los fines personalizados para los que el usuario ha dado su consentimiento.
- `customPurposesLITransparency`: una lista de números enteros que representan los fines personalizados para los que el usuario ha dado transparencia a sus intereses legítimos.

Estos datos detallados de consentimiento se pueden utilizar para comprender y respetar las preferencias de privacidad del usuario cuando trabaja con datos personales.

Ejemplo

El siguiente ejemplo utiliza un único argumento, que es la cadena de consentimiento codificada. Devuelve un diccionario que contiene los datos de consentimiento decodificados, incluida información sobre las preferencias de privacidad del usuario, las opciones de consentimiento y otros metadatos.

```
from aws_clean_rooms.functions import consent_tcf_v2_decode

consent_string = "CO1234567890abcdef"
consent_data = consent_tcf_v2_decode(consent_string)

print(consent_data)
```

La estructura básica de los datos de consentimiento devueltos incluye información sobre la versión de la cadena de consentimiento, los detalles de la CMP (plataforma de gestión del consentimiento), las opciones de consentimiento e interés legítimo del usuario para los distintos fines y proveedores, y otros metadatos.

```
    /** 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],
};
```

Funciones de ventana

Con las funciones de ventana, puede crear consultas empresariales analíticas de manera más eficiente. Las funciones de ventana operan en una partición o "ventana" de un conjunto de resultados y devuelven un valor para cada fila de esa ventana. Por el contrario, las funciones que no son de ventana realizan sus cálculos respecto de cada fila en el conjunto de resultados. A diferencia de las funciones de grupo que agregan las filas de resultados, las funciones de ventana retienen todas las filas de la expresión de tabla.

Los valores devueltos se calculan con los valores de los conjuntos de filas en esa ventana. Para cada fila en la tabla, la ventana define un conjunto de filas que se usan para computar atributos adicionales. Una ventana se define utilizando una especificación de ventana (la cláusula OVER) y se basa en tres conceptos principales:

- Particionamiento de ventana, que forma grupos de filas (cláusula PARTITION).
- Ordenación de ventana, que define un orden o una secuencia de filas dentro de cada partición (cláusula ORDER BY).
- Marcos de ventana, que se definen en función de cada fila para restringir aún más el conjunto de filas (especificación ROWS).

Las funciones de ventana son el último conjunto de operaciones realizadas en una consulta, excepto por la cláusula final ORDER BY. Todas las combinaciones y todas las cláusulas WHERE, GROUP BY y HAVING se completan antes de que se procesen las funciones de ventana. Por lo tanto, las funciones de ventana pueden figurar solamente en la lista SELECT o en la cláusula ORDER BY. Se pueden usar distintas funciones de ventana dentro de una única consulta con diferentes cláusulas de marco. También se pueden usar las funciones de ventana en otras expresiones escalares, como CASE.

Resumen de la sintaxis de la función de ventana

Las funciones de ventana siguen una sintaxis estándar, que es la que se indica a continuación.

```
function (expression) OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list [ frame_clause ] ] )
```

Aquí, function es una de las funciones que se describen en esta sección

La apariencia de expr_list es la siguiente.

```
expression | column_name [, expr_list ]
```

El order_list tiene la siguiente apariencia.

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

La frame_clause tiene la siguiente apariencia.

```
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 }}
```

Argumentos

función

La función de ventana. Para obtener más información, consulte las descripciones de las funciones individuales.

OVER

La cláusula que define la especificación de ventana. La cláusula OVER es obligatoria para las funciones de ventana y distingue funciones de ventana de otras funciones SQL.

PARTITION BY expr_list

(Opcional) La cláusula PARTITION BY subdivide el conjunto de resultado en particiones, muy similar a la cláusula GROUP BY. Si hay una cláusula de partición, la función se calcula para las filas en cada partición. Si no se especifica una cláusula de partición, una única partición tiene toda la tabla y la función se computa para la tabla completa.

Las funciones de clasificación, DENSE_RANK, NTILE, RANK y ROW_NUMBER, requieren una comparación global de todas las filas en el conjunto de resultados. Cuando se utiliza una cláusula PARTITION BY, el optimizador de consultas puede ejecutar cada agregación en paralelo mediante la distribución de la carga de trabajo entre distintos sectores, según las particiones. Si no hay cláusula PARTITION BY, el paso de agregación se debe ejecutar en serie en un único sector, lo que puede tener consecuencias negativas importantes en el rendimiento, especialmente en el caso de clústeres grandes.

AWS Clean Roomsno admite cadenas literales en las cláusulas PARTITION BY.

ORDER BY order_list

(Opcional) La función de ventana se aplica a las filas dentro de cada partición ordenada, según la especificación de orden en ORDER BY. Esta cláusula ORDER BY es distinta y no guarda relación alguna con las cláusulas ORDER BY de frame_clause. La cláusula ORDER BY se puede usar sin la cláusula PARTITION BY.

Para las funciones de clasificación, la cláusula ORDER BY identifica las medidas para los valores de clasificación. Para las funciones de agregación, las filas particionadas se deben ordenar antes de que la función de agregación se compute para cada marco. Para obtener más información acerca de los tipos de funciones de ventana, consulte Funciones de ventana.

Se requieren identificadores de columnas o expresiones que toman el valor de los identificadores de columnas en la lista de ordenación. No se pueden usar constantes ni expresiones constantes como sustitutos para los nombres de columnas.

Los valores NULLS se tratan como su propio grupo; se ordenan y se clasificación según la opción NULLS FIRST o NULLS LAST. De manera predeterminada, los valores NULL se ordenan y clasificación al final en orden ASC, y se ordenan y se clasifican primero en orden DESC.

AWS Clean Roomsno admite cadenas literales en las cláusulas ORDER BY.

Si se omite la cláusula ORDER BY, el orden de las filas no es determinista.

> **Note**
>
> En cualquier sistema paralelo, por ejemploAWS Clean Rooms, cuando una cláusula ORDER BY no produce un orden único y total de los datos, el orden de las filas no es determinista. Es decir, si la expresión ORDER BY produce valores duplicados (un orden parcial), el orden de retorno de esas filas puede variar de una serie AWS Clean Rooms a otra. A su vez, las funciones de ventana pueden devolver resultados inesperados o inconsistente. Para obtener más información, consulte Ordenación única de datos para funciones de ventana.

column_name

Nombre de una columna que se particionará u ordenará.

ASC | DESC

Opción que define el orden de ordenación para la expresión, de la siguiente manera:

- ASC: ascendente (por ejemplo, de menor a mayor para valores numéricos y de la A a la Z para cadenas con caracteres). Si no se especifica ninguna opción, los datos se ordenan, de manera predeterminada, en orden ascendente.
- DESC: descendente (de mayor a menor para valores numéricos y de la Z a la A para cadenas).

NULLS FIRST | NULLS LAST

Opción que especifica si los valores NULL se deben ordenar en primer lugar, antes de los valores no nulos, o al final, después de los valores no nulos. De manera predeterminada, los valores NULL se ordenan y clasificación al final en orden ASC, y se ordenan y se clasifican primero en orden DESC.

frame_clause

Para funciones de agregación, la cláusula de marco limita el conjunto de filas en una ventana de función al usar ORDER BY. Le permite incluir o excluir conjuntos de filas dentro del resultado ordenado. La cláusula de marco consta de la palabra clave ROWS y de los especificadores correspondientes.

La cláusula de marco no se aplica a las funciones de clasificación. Además, no es necesaria cuando no se utiliza una cláusula ORDER BY en la cláusula OVER para una función de agrupación. Si se utiliza una cláusula ORDER BY para una función de agregación, se necesita una cláusula de marco explícita.

Cuando no se especifica una cláusula ORDER BY, el marco implícito es ilimitado, lo que es equivalente a ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

ROWS

Esta cláusula define el marco de ventana especificando un desplazamiento físico desde la fila actual .

Esta cláusula especifica las filas en la ventana o partición actual que se combinará con el valor de la fila actual. Utiliza argumentos que especifican la posición de la fila, que puede ser antes o después de la fila actual. El punto de referencia para todos los marcos de ventana es la fila actual. Cada fila se convierte en la fila actual cuando el marco de ventana se desplaza hacia delante en la partición.

El marco puede ser un conjunto simple de filas hasta la fila actual, que se incluye.

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

O bien, puede ser un conjunto de filas entre dos límites.

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING indica que la ventana comienza en la primera fila de la partición; offset PRECEDING indica que la ventana comienza en un número de filas equivalente al valor de desplazamiento antes de la fila actual. UNBOUNDED PRECEDING es el valor predeterminado.

CURRENT ROW indica que la ventana comienza o finaliza en la fila actual.

UNBOUNDED FOLLOWING indica que la ventana finaliza en la última fila de la partición; offset FOLLOWING indica que la ventana finaliza en un número de filas equivalente al valor de desplazamiento después de la fila actual.

offset identifica un número físico de filas antes o después de la fila actual. En este caso, offset debe ser una constante que se evalúe como un valor numérico positivo. Por ejemplo, 5 FOLLOWING finaliza el marco de cinco filas después de la fila actual.

Cuando no se especifica BETWEEN, el marco se limita implícitamente a la fila actual. Por ejemplo, `ROWS 5 PRECEDING` equivale a `ROWS BETWEEN 5 PRECEDING AND CURRENT ROW`. Además, `ROWS UNBOUNDED FOLLOWING` equivale a `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`.

> **Note**
>
> No puede especificar un marco en el que el límite de inicio sea mayor que el límite final. Por ejemplo, no puede especificar ninguno de estos marcos.
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

Ordenación única de datos para funciones de ventana

Si una cláusula ORDER BY para una función de ventana no produce una ordenación total y única de los datos, el orden de las filas no es determinístico. Si la expresión ORDER BY produce valores duplicados (una ordenación parcial), el orden de retorno de esas filas puede variar en distintas ejecuciones. En este caso, las funciones de ventana también pueden devolver resultados inesperados o inconsistentes.

Por ejemplo, la siguiente consulta devuelve resultados diferentes con las múltiples ejecuciones. Estos resultados diferentes se producen porque `order by dateid` no genera una ordenación única de los datos para la función de ventana SUM.

```
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
...
```

En este caso, agregar una segunda columna ORDER BY a la función de ventana puede solucionar el problema.

```
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
...
```

Funciones compatibles

AWS Clean RoomsSpark SQL admite dos tipos de funciones de ventana: agregar y clasificar.

A continuación, se indican las funciones de agregado admitidas:

- Función de ventana CUME_DIST
- Función de ventana DENSE_RANK
- PRIMERA función de ventana
- Función de ventana FIRST_VALUE
- Función de ventana LAG
- Función de última ventana
- Función de ventana LAST_VALUE
- Función de ventana LEAD

A continuación, se indican las funciones de clasificación admitidas:

- Función de ventana DENSE_RANK
- Función de ventana PERCENT_RANK
- Función de ventana RANK
- Función de ventana ROW_NUMBER

Tabla de muestra para ejemplos de funciones de ventana

Puede encontrar ejemplos específicos de funciones de ventana con la descripción de cada función. Algunos de los ejemplos utilizan una tabla denominada WINSALES que tiene 11 filas, tal como se muestra a continuación.

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	

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
20001	2/12/2004	2	B	20	20
40005	2/12/2004	4	A	10	10
20002	2/16/2004	2	C	20	20
30003	4/18/2004	3	B	15	
30004	4/18/2004	3	B	20	
30007	9/7/2004	3	C	30	

Función de ventana CUME_DIST

Calcula la distribución acumulada de un valor dentro de una ventana o partición. Si se asume un orden ascendente, la distribución acumulada se determina utilizando esta fórmula:

```
count of rows with values <= x / count of rows in the window or partition
```

donde x equivale al valor en la fila actual de la columna especificada en la cláusula ORDER BY. El siguiente conjunto de datos ilustra el uso de esta fórmula:

```
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
```

El rango de valor de retorno es > 0 a 1, inclusive.

Sintaxis

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
```

```
)
```

Argumentos

OVER

Una cláusula que especifica la partición de ventana. La cláusula OVER no puede tener una especificación de marco de ventana.

PARTITION BY partition_expression

Opcional. Una expresión que establece el rango de registros para cada grupo en la cláusula OVER.

ORDER BY order_list

La expresión sobre la cual se calcula la distribución acumulada. La expresión debe tener un tipo de dato numérico o ser implícitamente convertible a un dato numérico. Si se omite ORDER BY, el valor de retorno es 1 para todas las filas.

Si ORDER BY no produce una ordenación única, el orden de las filas no es determinístico. Para obtener más información, consulte Ordenación única de datos para funciones de ventana.

Tipo de devolución

FLOAT8

Ejemplos

En el siguiente ejemplo, se calcula la distribución acumulada de la cantidad para cada vendedor:

```
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
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

Función de ventana DENSE_RANK

La función de ventana DENSE_RANK determina la clasificación de un valor en un grupo de valores, según la expresión ORDER BY en la cláusula OVER. Si hay una cláusula opcional PARTITION BY, las clasificaciones de restablecen para cada grupo de filas. Las filas con valores iguales para el criterio de clasificación reciben la misma clasificación. La función DENSE_RANK difiere de RANK en un aspecto: si se vinculan dos o más filas, no hay brecha en la secuencia de valores clasificados. Por ejemplo, si dos filas tienen clasificación 1, la siguiente clasificación es 2.

Puede tener funciones de clasificación con diferentes cláusulas PARTITION BY y ORDER BY en la misma consulta.

Sintaxis

```
DENSE_RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumentos

( )

La función no toma argumentos, pero se necesitan los paréntesis vacíos.

OVER

Las cláusulas de ventana para la función DENSE_RANK.

PARTITION BY expr_list

Opcional. Una o más expresiones que definen la ventana.

ORDER BY order_list

Opcional. La expresión en que se basan los valores de clasificación. Si no se especifica PARTITION BY, ORDER BY utiliza toda la tabla. Si se omite ORDER BY, el valor de retorno es 1 para todas las filas.

Si ORDER BY no produce una ordenación única, el orden de las filas no es determinístico. Para obtener más información, consulte Ordenación única de datos para funciones de ventana.

Tipo de devolución

INTEGER

Ejemplos

En el siguiente ejemplo, se ordena la tabla según la cantidad vendida (en orden descendiente) y se asigna a cada fila tanto una clasificación densa como una regular. Los resultados se ordenan después de que se apliquen los resultados de la función de ventana.

```
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)
```

Tenga en cuenta la diferencia entre las clasificaciones asignadas al mismo conjunto de filas cuando se usan las funciones DENSE_RANK y RANK en simultáneo en la misma consulta. Para ver una

descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

En el siguiente ejemplo, se divide la tabla según SELLERID, se ordena cada partición según la cantidad (en orden descendiente) y se asigna a cada fila una clasificación densa. Los resultados se ordenan después de que se apliquen los resultados de la función de ventana.

```
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)
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

PRIMERA función de ventana

Dado un conjunto ordenado de filas, FIRST devuelve el valor de la expresión especificada con respecto a la primera fila del marco de la ventana.

Para obtener información sobre cómo seleccionar la última fila del marco, consulte Función de última ventana.

Sintaxis

```
FIRST( expression )[ IGNORE NULLS | RESPECT NULLS ]
```

```
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumentos

expresión

La columna o expresión de destino sobre la que opera la función.

IGNORE NULLS

Cuando se utiliza esta opción con FIRST, la función devuelve el primer valor del marco que no sea NULL (o NULL si todos los valores son NULL).

RESPECT NULLS

Indica que se AWS Clean Rooms deben incluir valores nulos a la hora de determinar qué fila utilizar. De manera predeterminada, se admite RESPECT NULLS si no especifica IGNORE NULLS.

OVER

Introduce las cláusulas de ventana para la función.

PARTITION BY expr_list

Define la ventana para la función en términos de una o más expresiones.

ORDER BY order_list

Ordena las filas dentro de cada partición. Si no se especifica cláusula PARTITION BY, ORDER BY ordena toda la tabla. Si especifica una cláusula ORDER BY, también debe especificar una frame_clause (cláusula_de_marco).

Los resultados de la función FIRST dependen del orden de los datos. En los siguientes casos, los resultados son no determinísticos:

- Cuando no se especifica una cláusula ORDER BY y una partición tiene dos valores diferentes para una expresión
- Cuando la expresión toma valores diferentes que corresponden al mismo valor en la lista ORDER BY.

frame_clause

Si se utiliza una cláusula ORDER BY para una función de agregación, se necesita una cláusula de marco explícita. La cláusula de marco limita el conjunto de filas en una ventana de función e incluye o excluye conjuntos de filas en del resultado ordenado. La cláusula de marco consta de la palabra clave ROWS y de los especificadores correspondientes. Consulte Resumen de la sintaxis de la función de ventana.

Tipo de devolución

Estas funciones admiten expresiones que utilizan tipos de AWS Clean Rooms datos primitivos. El tipo de retorno es el mismo que el tipo de datos de la expression.

Ejemplos

El siguiente ejemplo devuelve la capacidad de asientos para cada lugar en la tabla VENUE, con los resultados ordenados por capacidad (de mayor a menor). La función FIRST se utiliza para seleccionar el nombre del lugar que corresponde a la primera fila del cuadro: en este caso, la fila con el mayor número de asientos. Los resultados se particionan por estado, por lo que cuando cambia el valor VENUESTATE, se selecciona un nuevo primer valor. El marco de ventana está ilimitado de modo que el primer valor se selecciona para cada fila en cada partición.

Para California, `Qualcomm Stadium` tiene la mayor cantidad de asientos (`70561`), por lo que nombre es el primer valor para todas las filas en la partición `CA`.

```
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
...
```

Función de ventana FIRST_VALUE

Dado un conjunto ordenado de filas, FIRST_VALUE devuelve el valor de la expresión especificada respecto de la primera fila en el marco de ventana.

Para obtener información sobre cómo seleccionar la última fila del marco, consulte Función de ventana LAST_VALUE.

Sintaxis

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumentos

expresión

La columna o expresión de destino sobre la que opera la función.

IGNORE NULLS

Cuando se utiliza esta opción con FIRST_VALUE, la función devuelve el primer valor en el marco que no sea NULL (o NULL si todos los valores son NULL).

RESPECT NULLS

Indica que se AWS Clean Rooms deben incluir valores nulos a la hora de determinar qué fila utilizar. De manera predeterminada, se admite RESPECT NULLS si no especifica IGNORE NULLS.

OVER

Introduce las cláusulas de ventana para la función.

PARTITION BY expr_list

Define la ventana para la función en términos de una o más expresiones.

ORDER BY order_list

Ordena las filas dentro de cada partición. Si no se especifica cláusula PARTITION BY, ORDER BY ordena toda la tabla. Si especifica una cláusula ORDER BY, también debe especificar una frame_clause (cláusula_de_marco).

Los resultados de la función FIRST_VALUE dependen del orden de los datos. En los siguientes casos, los resultados son no determinísticos:

- Cuando no se especifica una cláusula ORDER BY y una partición tiene dos valores diferentes para una expresión
- Cuando la expresión toma valores diferentes que corresponden al mismo valor en la lista ORDER BY.

frame_clause

Si se utiliza una cláusula ORDER BY para una función de agregación, se necesita una cláusula de marco explícita. La cláusula de marco limita el conjunto de filas en una ventana de función e incluye o excluye conjuntos de filas en del resultado ordenado. La cláusula de marco consta de la palabra clave ROWS y de los especificadores correspondientes. Consulte Resumen de la sintaxis de la función de ventana.

Tipo de devolución

Estas funciones admiten expresiones que utilizan tipos de AWS Clean Rooms datos primitivos. El tipo de retorno es el mismo que el tipo de datos de la expression.

Ejemplos

El siguiente ejemplo devuelve la capacidad de asientos para cada lugar en la tabla VENUE, con los resultados ordenados por capacidad (de mayor a menor). La función FIRST_VALUE se utiliza para seleccionar el nombre del lugar que corresponda a la primera fila en el marco: en este caso, la fila con la mayor cantidad de asientos. Los resultados se particionan por estado, por lo que cuando

cambia el valor VENUESTATE, se selecciona un nuevo primer valor. El marco de ventana está ilimitado de modo que el primer valor se selecciona para cada fila en cada partición.

Para California, `Qualcomm Stadium` tiene la mayor cantidad de asientos (`70561`), por lo que nombre es el primer valor para todas las filas en la partición `CA`.

```
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
...
```

Función de ventana LAG

La función de ventana LAG devuelve los valores para una fila en un desplazamiento dado arriba (antes) de la fila actual en la partición.

Sintaxis

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
```

```
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Argumentos

value_expr

La columna o expresión de destino sobre la que opera la función.

desplazamiento

Un parámetro opcional que especifica la cantidad de filas antes de la fila actual para la cual devolver valores. El desplazamiento puede ser un valor entero constante o una expresión que tome un valor entero. Si no especifica un desfase, 1 lo AWS Clean Rooms utiliza como valor por defecto. Un desplazamiento de `0` indica la fila actual.

IGNORE NULLS

Especificación opcional que indica que se AWS Clean Rooms deben omitir los valores nulos a la hora de determinar qué fila utilizar. Los valores nulos se incluyen si no se indica IGNORE NULLS.

> **Note**
>
> Puede usar una expresión NVL o COALESCE para reemplazar los valores nulos con otro valor.

RESPECT NULLS

Indica que se AWS Clean Rooms deben incluir valores nulos en la determinación de la fila que se debe utilizar. De manera predeterminada, se admite RESPECT NULLS si no especifica IGNORE NULLS.

OVER

Especifica la partición de ventana y el ordenamiento. La cláusula OVER no puede tener una especificación de marco de ventana.

PARTITION BY window_partition

Un argumento opcional que establece el rango de registros para cada grupo en la cláusula OVER.

ORDER BY window_ordering

Ordena las filas dentro de cada partición.

La función de ventana LAG admite expresiones que utilizan cualquiera de los tipos de AWS Clean Rooms datos. El tipo de valor devuelto es el mismo que el tipo de la value_expr (expresión_de_valor).

Ejemplos

En el siguiente ejemplo, se muestra la cantidad de tickets vendidos al comprador con un ID de comprador de 3 y la hora en que el comprador 3 compró los tickets. Para comparar cada venta con la venta anterior para el comprador 3, la consulta devuelve la cantidad anterior vendida para cada venta. Debido a que no hay compras antes del 01/16/2008, el primer valor de cantidad vendida anterior es nulo:

```
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)
```

Función de última ventana

Dado un conjunto ordenado de filas, la función LAST devuelve el valor de la expresión con respecto a la última fila del marco.

Para obtener información sobre cómo seleccionar la primera fila del marco, consulte PRIMERA función de ventana.

Sintaxis

```
LAST( expression )[ IGNORE NULLS | RESPECT NULLS ]
```

```
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumentos

expresión

La columna o expresión de destino sobre la que opera la función.

IGNORE NULLS

La función devuelve el último valor en el marco que no sea NULL (o NULL si todos los valores son NULL).

RESPECT NULLS

Indica que se AWS Clean Rooms deben incluir valores nulos a la hora de determinar qué fila utilizar. De manera predeterminada, se admite RESPECT NULLS si no especifica IGNORE NULLS.

OVER

Introduce las cláusulas de ventana para la función.

PARTITION BY expr_list

Define la ventana para la función en términos de una o más expresiones.

ORDER BY order_list

Ordena las filas dentro de cada partición. Si no se especifica cláusula PARTITION BY, ORDER BY ordena toda la tabla. Si especifica una cláusula ORDER BY, también debe especificar una frame_clause (cláusula_de_marco).

Los resultados dependen del orden de los datos. En los siguientes casos, los resultados son no deterministicos:

- Cuando no se especifica una cláusula ORDER BY y una partición tiene dos valores diferentes para una expresión
- Cuando la expresión toma valores diferentes que corresponden al mismo valor en la lista ORDER BY.

frame_clause

Si se utiliza una cláusula ORDER BY para una función de agregación, se necesita una cláusula de marco explícita. La cláusula de marco limita el conjunto de filas en una ventana de función e incluye o excluye conjuntos de filas en del resultado ordenado. La cláusula de marco consta de la palabra clave ROWS y de los especificadores correspondientes. Consulte Resumen de la sintaxis de la función de ventana.

Tipo de devolución

Estas funciones admiten expresiones que utilizan tipos de AWS Clean Rooms datos primitivos. El tipo de retorno es el mismo que el tipo de datos de la expression.

Ejemplos

El siguiente ejemplo devuelve la capacidad de asientos para cada lugar en la tabla VENUE, con los resultados ordenados por capacidad (de mayor a menor). La función LAST se utiliza para seleccionar el nombre del lugar que corresponde a la última fila del cuadro: en este caso, la fila con el menor número de asientos. Los resultados se particionan por estado, por lo que cuando cambia el valor VENUESTATE, se selecciona un nuevo último valor. El marco de la ventana está ilimitado de modo que el último valor se selecciona para cada fila en cada partición.

Para California, se devuelve `Shoreline Amphitheatre` para cada fila en la partición porque tiene la menor cantidad de asientos (`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
...
```

Función de ventana LAST_VALUE

Con un conjunto de filas ordenado, la función LAST_VALUE devuelve el valor de la expresión con respecto a la última fila del marco.

Para obtener información sobre cómo seleccionar la primera fila del marco, consulte Función de ventana FIRST_VALUE.

Sintaxis

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumentos

expresión

La columna o expresión de destino sobre la que opera la función.

IGNORE NULLS

La función devuelve el último valor en el marco que no sea NULL (o NULL si todos los valores son NULL).

RESPECT NULLS

Indica que se AWS Clean Rooms deben incluir valores nulos a la hora de determinar qué fila utilizar. De manera predeterminada, se admite RESPECT NULLS si no especifica IGNORE NULLS.

OVER

Introduce las cláusulas de ventana para la función.

PARTITION BY expr_list

Define la ventana para la función en términos de una o más expresiones.

ORDER BY order_list

Ordena las filas dentro de cada partición. Si no se especifica cláusula PARTITION BY, ORDER BY ordena toda la tabla. Si especifica una cláusula ORDER BY, también debe especificar una frame_clause (cláusula_de_marco).

Los resultados dependen del orden de los datos. En los siguientes casos, los resultados son no determinísticos:

- Cuando no se especifica una cláusula ORDER BY y una partición tiene dos valores diferentes para una expresión
- Cuando la expresión toma valores diferentes que corresponden al mismo valor en la lista ORDER BY.

frame_clause

Si se utiliza una cláusula ORDER BY para una función de agregación, se necesita una cláusula de marco explícita. La cláusula de marco limita el conjunto de filas en una ventana de función e incluye o excluye conjuntos de filas en del resultado ordenado. La cláusula de marco consta de la palabra clave ROWS y de los especificadores correspondientes. Consulte Resumen de la sintaxis de la función de ventana.

Tipo de devolución

Estas funciones admiten expresiones que utilizan tipos de AWS Clean Rooms datos primitivos. El tipo de retorno es el mismo que el tipo de datos de la expression.

Ejemplos

El siguiente ejemplo devuelve la capacidad de asientos para cada lugar en la tabla VENUE, con los resultados ordenados por capacidad (de mayor a menor). La función LAST_VALUE se utiliza para seleccionar el nombre del lugar que corresponda a la última fila en el marco: en este caso, la fila con la menor cantidad de asientos. Los resultados se particionan por estado, por lo que cuando cambia

el valor VENUESTATE, se selecciona un nuevo último valor. El marco de la ventana está ilimitado de modo que el último valor se selecciona para cada fila en cada partición.

Para California, se devuelve `Shoreline Amphitheatre` para cada fila en la partición porque tiene la menor cantidad de asientos (`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
...
```

Función de ventana LEAD

La función de ventana LEAD devuelve los valores para una fila en un desplazamiento dado abajo (después) de la fila actual en la partición.

Sintaxis

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
```

```
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Argumentos

value_expr

La columna o expresión de destino sobre la que opera la función.

desplazamiento

Un parámetro opcional que especifica la cantidad de filas debajo de la fila actual para la cual devolver valores. El desplazamiento puede ser un valor entero constante o una expresión que tome un valor entero. Si no especifica un desfase, 1 lo AWS Clean Rooms utiliza como valor por defecto. Un desplazamiento de `0` indica la fila actual.

IGNORE NULLS

Especificación opcional que indica que se AWS Clean Rooms deben omitir los valores nulos a la hora de determinar qué fila utilizar. Los valores nulos se incluyen si no se indica IGNORE NULLS.

> **Note**
>
> Puede usar una expresión NVL o COALESCE para reemplazar los valores nulos con otro valor.

RESPECT NULLS

Indica que se AWS Clean Rooms deben incluir valores nulos en la determinación de la fila que se debe utilizar. De manera predeterminada, se admite RESPECT NULLS si no especifica IGNORE NULLS.

OVER

Especifica la partición de ventana y el ordenamiento. La cláusula OVER no puede tener una especificación de marco de ventana.

PARTITION BY window_partition

Un argumento opcional que establece el rango de registros para cada grupo en la cláusula OVER.

ORDER BY window_ordering

Ordena las filas dentro de cada partición.

La función de ventana LEAD admite expresiones que utilizan cualquiera de los tipos de AWS Clean Rooms datos. El tipo de valor devuelto es el mismo que el tipo de la value_expr (expresión_de_valor).

Ejemplos

En el siguiente ejemplo, se proporciona la comisión para eventos en la tabla SALES para los cuales se vendieron tickets el 1 y el 2 de enero de 2008, y la comisión pagada por la venta de tickets de la venta subsiguiente.

```
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)
```

Función de ventana PERCENT_RANK

Calcula la clasificación de porcentaje de una fila dada. La clasificación de porcentaje se determina utilizando la siguiente fórmula:

```
(x - 1) / (the number of rows in the window or partition - 1)
```

donde x es la clasificación de la fila actual. El siguiente conjunto de datos ilustra el uso de esta fórmula:

```
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
```

El rango de valor de retorno es 0 a 1, inclusive. La primera fila en cualquier conjunto tiene un PERCENT_RANK de 0.

Sintaxis

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Argumentos

( )

La función no toma argumentos, pero se necesitan los paréntesis vacíos.

OVER

Una cláusula que especifica la partición de ventana. La cláusula OVER no puede tener una especificación de marco de ventana.

PARTITION BY partition_expression

Opcional. Una expresión que establece el rango de registros para cada grupo en la cláusula OVER.

ORDER BY order_list

Opcional. La expresión sobre la cual se calcula la clasificación de porcentaje. La expresión debe tener un tipo de dato numérico o ser implícitamente convertible a un dato numérico. Si se omite ORDER BY, el valor de retorno es 0 para todas las filas.

Si ORDER BY no produce un ordenamiento único, el orden de las filas no es determinístico. Para obtener más información, consulte Ordenación única de datos para funciones de ventana.

Tipo de devolución

FLOAT8

Ejemplos

En el siguiente ejemplo, se calcula la clasificación de porcentaje de las cantidades de ventas para cada vendedor:

```
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
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

Función de ventana RANK

La función de ventana RANK determina la clasificación de un valor en un grupo de valores, según la expresión ORDER BY en la cláusula OVER. Si hay una cláusula opcional PARTITION BY, las clasificaciones de restablecen para cada grupo de filas. Las filas con valores iguales para los criterios de clasificación reciben la misma clasificación. AWS Clean Roomssuma el número de filas empatadas a la clasificación empatada para calcular la siguiente clasificación y, por lo tanto, es

posible que las filas no sean números consecutivos. Por ejemplo, si dos filas tienen clasificación 1, la siguiente clasificación es 3.

RANK difiere de Función de ventana DENSE_RANK en un aspecto: para DENSE_RANK, si se vinculan dos o más filas, no hay brecha en la secuencia de valores clasificados. Por ejemplo, si dos filas tienen clasificación 1, la siguiente clasificación es 2.

Puede tener funciones de clasificación con diferentes cláusulas PARTITION BY y ORDER BY en la misma consulta.

Sintaxis

```
RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumentos

( )

La función no toma argumentos, pero se necesitan los paréntesis vacíos.

OVER

Las cláusulas de ventana para la función RANK.

PARTITION BY expr_list

Opcional. Una o más expresiones que definen la ventana.

ORDER BY order_list

Opcional. Define las columnas en que se basan los valores de clasificación. Si no se especifica PARTITION BY, ORDER BY utiliza toda la tabla. Si se omite ORDER BY, el valor de retorno es 1 para todas las filas.

Si ORDER BY no produce un ordenamiento único, el orden de las filas no es determinístico. Para obtener más información, consulte Ordenación única de datos para funciones de ventana.

Tipo de devolución

INTEGER

Ejemplos

En el siguiente ejemplo, se ordena la tabla por la cantidad vendida (orden ascendiente predeterminado) y se asigna una clasificación a cada fila. Un valor de 1 es la mejor clasificación. Los resultados se ordenan después de que se apliquen los resultados de la función de ventana:

```
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)
```

Tenga en cuenta que la cláusula ORDER BY externa de este ejemplo incluye las columnas 2 y 1 para garantizar que AWS Clean Rooms devuelva resultados ordenados de forma coherente cada vez que se ejecute la consulta. Por ejemplo, las filas con ventas IDs 10001 y 10006 tienen valores de QTY y RNK idénticos. Ordenar el resultado final por columna 1 garantiza que la fila 10001 siempre esté antes que la 10006. Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

En el siguiente ejemplo, la ordenación se invierte para la función de ventana (`order by qty desc`). Ahora, el valor más alto de clasificación se aplica al valor QTY más alto.

```
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)
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

En el siguiente ejemplo, se divide la tabla según SELLERID, se ordena cada partición según la cantidad (en orden descendiente) y se asigna una clasificación a cada fila. Los resultados se ordenan después de que se apliquen los resultados de la función de ventana.

```
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)
```

Función de ventana ROW_NUMBER

Determina el número ordinal de la fila actual dentro de un grupo de filas, contando desde 1, según la expresión ORDER BY en la cláusula OVER. Si hay una cláusula opcional PARTITION BY, los números ordinales se restablecen para cada grupo de filas. Las filas con valores iguales para las expresiones ORDER BY reciben los diferentes números de fila de manera no determinística.

Sintaxis

```
ROW_NUMBER () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumentos

( )

La función no toma argumentos, pero se necesitan los paréntesis vacíos.

OVER

Las cláusulas de ventana para la función ROW_NUMBER.

PARTITION BY expr_list

Opcional. Una o más expresiones que definen la función ROW_NUMBER.

ORDER BY order_list

Opcional. La expresión que define las columnas en que se basan los números de fila. Si no se especifica PARTITION BY, ORDER BY utiliza toda la tabla.

Si ORDER BY no produce una ordenación única o se omite, el orden de las filas no es determinístico. Para obtener más información, consulte Ordenación única de datos para funciones de ventana.

Tipo de devolución

BIGINT

Ejemplos

En el siguiente ejemplo, se particiona la tabla según SELLERID y se ordena cada partición según QTY (en orden ascendiente); luego, se asigna un número a cada fila. Los resultados se ordenan después de que se apliquen los resultados de la función de ventana.

```
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)
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

AWS Clean Rooms Condiciones de Spark SQL

Las condiciones son instrucciones de una o más expresiones y operadores lógicos que resuelven con un valor de verdadero, falso o desconocido. A las condiciones a veces se las denomina predicados.

Sintaxis

```
comparison_condition
| logical_condition
| range_condition
```

```
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

> **Note**
>
> Todas las comparaciones de cadenas y coincidencias del patrón LIKE distinguen entre mayúsculas y minúsculas. Por ejemplo, "A" y "a" no coinciden. Sin embargo, puede hacer una coincidencia de patrones que no distinga entre mayúsculas y minúsculas al utilizar el predicado ILIKE.

AWS Clean Rooms Spark SQL admite las siguientes condiciones de SQL.

Temas

Operadores de comparación

Las condiciones de comparación indican relaciones lógicas entre dos valores. Todas las condiciones de comparación son operadores binarios con un tipo devuelto booleano.

AWS Clean Rooms Spark SQL admite los operadores de comparación que se describen en la siguiente tabla.

Operador	Sintaxis	Descripción
!	`!expression`	El NOT operador lógico. Se usa para negar una expresión booleana, lo que significa que

Operador	Sintaxis	Descripción
		devuelve el valor opuesto al de la expresión. ¡El! El operador también se puede combinar con otros operadores lógicos, como AND y OR, para crear expresiones booleanas más complejas.
<	`a < b`	El operador de comparación menor que. Se utiliza para comparar dos valores y determinar si el valor de la izquierda es menor que el valor de la derecha.
>	`a > b`	El operador de comparación mayor que. Se utiliza para comparar dos valores y determinar si el valor de la izquierda es mayor que el valor de la derecha.
<=	`a <= b`	El operador de comparación menor o igual a. Se utiliza para comparar dos valores y devuelve `true` si el valor de la izquierda es menor o igual que el valor de la derecha, o si `false` no.

Operador	Sintaxis	Descripción
>=	`a >= b`	El operador de comparación mayor o igual a. Se utiliza para comparar dos valores y determinar si el valor de la izquierda es mayor o igual que el valor de la derecha.
=	`a = b`	El operador de comparación de igualdad, que compara dos valores y devuelve `true` si son iguales o si `false` no.
<> o **!=**	`a <> b` o `a != b`	El operador de comparación no igual, que compara dos valores y devuelve resultados `true` si no son iguales o si no `false` lo son.

Operador	Sintaxis	Descripción
==	`a == b`	El operador de comparación de igualdad estándar, que compara dos valores y devuelve `true` si son iguales o si `false` no lo son. **Note** El operador == distingue entre mayúsculas y minúsculas al comparar valores de cadenas. Si necesita realizar una comparación que no distinga entre mayúsculas y minúsculas, puede utilizar funciones como UPPER () o LOWER () para convertir los valores en mayúsculas y minúsculas antes de la comparación.

Ejemplos

A continuación se muestran algunos ejemplos sencillos de condiciones de comparación:

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

La siguiente consulta devuelve los valores de identificación de todas las ardillas que actualmente no están buscando alimento.

```
SELECT id FROM squirrels
WHERE !is_foraging
```

La siguiente consulta devuelve los lugares con más de 10 000 asientos de la tabla VENUE:

```
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)
```

Este ejemplo selecciona los usuarios (USERID) de la tabla USERS que les gusta el rock:

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
13
16
(5 rows)
```

Este ejemplo selecciona los usuarios (USERID) de la tabla USERS para los que se desconoce si les gusta el rock:

```
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
```

Ejemplos con una columna TIME

La siguiente tabla de ejemplo, TIME_TEST, tiene una columna TIME_VAL (tipo TIME) con tres valores insertados.

```
select time_val from time_test;

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

En el siguiente ejemplo, se extraen las horas de cada timetz_val.

```
select time_val from time_test where time_val < '3:00';
   time_val
---------------
 00:00:00.5550
 00:58:00
```

En el siguiente ejemplo, se comparan dos literales de tiempo.

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
 t
```

Ejemplos con una columna TIMETZ

La siguiente tabla de ejemplo, TIMETZ_TEST, tiene una columna TIMETZ_VAL (tipo TIMETZ) con tres valores insertados.

```
select timetz_val from timetz_test;

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

En el siguiente ejemplo, se seleccionan solo los valores TIMETZ menores que `3:00:00 UTC`. La comparación se realiza después de convertir el valor a la UTC.

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

En el siguiente ejemplo, se comparan dos literales TIMETZ. Para la comparación, se ignora la zona horaria.

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

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

Condiciones lógicas

Las condiciones lógicas combinan el resultado de dos condiciones para producir un único resultado. Todas las condiciones lógicas son operadores binarios con un tipo devuelto booleano.

Sintaxis

```
expression
{ AND | OR }
expression
NOT expression
```

Las condiciones lógicas utilizan un lógico booleano de tres valores donde el valor nulo representa una relación desconocida. En la siguiente tabla se describen los resultados de condiciones lógicas, donde `E1` y `E2` representan expresiones:

E1	E2	E1 AND E2	E1 OR E2	NOT E2
TRUE	TRUE	TRUE	TRUE	FALSO
TRUE	FALSO	FALSO	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSO	TRUE	FALSO	TRUE	
FALSO	FALSO	FALSO	FALSO	
FALSO	UNKNOWN	FALSO	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSO	FALSO	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

El operador NOT se evalúa antes de AND, y el operador AND se evalúa antes del operador OR. Cualquier paréntesis utilizado puede invalidar este orden de evaluación predeterminado.

Ejemplos

En el siguiente ejemplo se devuelve USERID y USERNAME de la tabla USERS donde al usuario le gusta Las Vegas y los deportes:

```
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)
```

En el siguiente ejemplo se devuelve el USERID y USERNAME de la tabla USERS donde al usuario le gusta Las Vegas o los deportes, o ambos. Esta consulta devuelve todos los resultados del ejemplo anterior además de los usuarios que solo les gustan Las Vegas o los deportes.

```
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)
```

La siguiente consulta usa paréntesis alrededor de la condición OR para encontrar lugares en Nueva York o California donde se realizó Macbeth:

```
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
...
```

Eliminar los paréntesis en este ejemplo cambia la lógica y los resultados de la consulta.

En el siguiente ejemplo se usa el operador NOT:

```
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
...
```

En el siguiente ejemplo se usa una condición NOT seguida de una condición AND:

```
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)
```

Condiciones de coincidencia de patrones

Un operador de coincidencia de patrones busca en una cadena un patrón especificado en la expresión condicional y devuelve verdadero o falso en función de si encuentra una coincidencia. AWS Clean Rooms Spark SQL utiliza los siguientes métodos para la coincidencia de patrones:

- Expresiones LIKE

  El operador LIKE compara una expresión de cadena, como el nombre de una columna, con un patrón que usa caracteres comodines % (porcentaje) y _ (guion bajo). La coincidencia de patrones LIKE siempre cubre la cadena completa. LIKE realiza una coincidencia que distingue entre mayúsculas y minúsculas.

Temas

- LIKE
- RLIKE

LIKE

El operador LIKE compara una expresión de cadena, como el nombre de una columna, con un patrón que usa caracteres comodines % (porcentaje) y _ (guion bajo). La coincidencia de patrones LIKE siempre cubre la cadena completa. Para relacionar una secuencia en cualquier lugar dentro de una cadena, el patrón debe comenzar y finalizar con un signo de porcentaje.

LIKE distingue entre mayúsculas y minúsculas.

Sintaxis

```
expression [ NOT ] LIKE | pattern [ ESCAPE 'escape_char' ]
```

Argumentos

expresión

Una expresión de carácter UTF-8 válido, como un nombre de columna.

LIKE

LIKE realiza una coincidencia de patrones que distingue entre mayúsculas y minúsculas. Para ejecutar una coincidencia de patrones sin distinguir entre mayúsculas y minúsculas con caracteres multibyte, utilice la función LOWER de expresión y patrón con una condición LIKE.

A diferencia de los predicados de comparación, como = y <>, los predicados LIKE no ignoran implícitamente los espacios finales. Para omitir los espacios finales, utilice RTRIM o convierta explícitamente una columna CHAR en VARCHAR.

El ~~ operador equivale a LIKE. Además, el !~~ operador equivale a NOT LIKE.

pattern

Una expresión de carácter UTF-8 válido con el patrón que se relacionará.

escape_char (carácter_de_escape)

Una expresión de carácter que aplicará escape a metacaracteres en el patrón. La predeterminada es dos barras diagonales invertidas ("\\").

Si el patrón no contiene metacaracteres, solo representa la propia cadena; en ese caso, LIKE actúa igual que el operador de igualdad.

Cualquiera de las expresiones de carácter pueden ser tipos de datos CHAR o VARCHAR. Si son diferentes, AWS Clean Rooms convierte el patrón al tipo de datos de la expresión.

LIKE admite los siguientes metacaracteres de coincidencia de patrón:

Operador	Descripción
%	Coincide con cualquier secuencia de cero o más caracteres.

Operador	Descripción
_	Coincide con cualquier carácter.

Ejemplos

En la tabla siguiente se muestran ejemplos de coincidencia de patrones a través de LIKE:

Expression	Devuelve
`'abc' LIKE 'abc'`	True
`'abc' LIKE 'a%'`	True
`'abc' LIKE '_B_'`	False
`'abc' LIKE 'c%'`	False

En el siguiente ejemplo se encuentran todas las ciudades cuyos nombres comienzan con "E":

```
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
...
```

En el siguiente ejemplo se encuentran usuarios cuyos apellidos contienen "ten":

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
-------------
```

```
Christensen
Wooten
...
```

En el siguiente ejemplo, se buscan ciudades cuyo tercer y cuarto carácter son «ea» . :

```
select distinct city from users where city like '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

En el siguiente ejemplo se usa la cadena de escape predeterminada (\\) para buscar cadenas que incluyan «start_» (el texto `start` seguido de un guion bajo _):

```
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)
```

En el siguiente ejemplo se especifica «^» como el carácter de escape y, luego, se utiliza el carácter de escape para buscar cadenas que incluyan «start_» (el texto `start` seguido de un guion bajo _):

```
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

El operador RLIKE permite comprobar si una cadena coincide con un patrón de expresión regular especificado.

Devuelve `true` si str coincide `regexp` o `false` no.

Sintaxis

```
rlike(str, regexp)
```

Argumentos

str

Una expresión de cadena

expresión regular

Una expresión de cadena. La cadena de expresiones regulares debe ser una expresión regular de Java.

Los literales de cadena (incluidos los patrones de expresiones regulares) no tienen escapes en nuestro analizador SQL. Por ejemplo, para que coincida con «\ abc», una expresión regular para expresiones regulares puede ser «^\ abc$».

Ejemplos

El siguiente ejemplo establece el valor del parámetro de configuración en`spark.sql.parser.escapedStringLiterals`. `true` Este parámetro es específico del motor SQL de Spark. El `spark.sql.parser.escapedStringLiterals` parámetro de Spark SQL controla la forma en que el analizador SQL gestiona los literales de cadena escapados. Cuando se

establece en`true`, el analizador interpretará los caracteres de barra invertida (\) de los literales de cadena como caracteres de escape, lo que te permitirá incluir caracteres especiales como líneas nuevas, tabulaciones y comillas dentro de los valores de la cadena.

```
SET spark.sql.parser.escapedStringLiterals=true;
spark.sql.parser.escapedStringLiterals  true
```

Por ejemplo, con`spark.sql.parser.escapedStringLiterals=true`, podrías usar el siguiente literal de cadena en tu consulta SQL:

```
SELECT 'Hello, world!\n'
```

El carácter de nueva línea se \n interpretaría como un carácter de nueva línea literal en la salida.

En el siguiente ejemplo, se realiza una coincidencia de patrones de expresiones regulares. El primer argumento se pasa al operador RLIKE. Es una cadena que representa la ruta de un archivo, donde el nombre de usuario real se sustituye por el patrón '****'. El segundo argumento es el patrón de expresión regular utilizado para la coincidencia. El resultado (`true`) indica que la primera cadena (`'%SystemDrive%\Users\****'`) coincide con el patrón de expresión regular (`'%SystemDrive%\\Users.*'`).

```
SELECT rlike('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

Condición de rango BETWEEN

Una condición `BETWEEN` prueba expresiones para incluirlas en un rango de valores, con las palabras clave `BETWEEN` y `AND`.

Sintaxis

```
expression [ NOT ] BETWEEN expression AND expression
```

Las expresiones pueden ser tipos de datos de fecha y hora, numéricos o caracteres, pero deben ser compatibles. El rango es inclusivo.

Ejemplos

El primer ejemplo cuenta cuántas transacciones registraron ventas de 2, 3 o 4 tickets:

```
select count(*) from sales
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

La condición de rango incluye los valores de inicio y final.

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

La primera expresión en una condición de rango debe ser el valor más bajo y la segunda expresión, el valor más alto. En el siguiente ejemplo SIEMPRE se devuelven cero filas debido a los valores de las expresiones:

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

Sin embargo, aplicar el modificador NOT invertirá la lógica y producirá un conteo de todas las filas:

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
172456
(1 row)
```

La siguiente consulta devuelve una lista de lugares que tienen entre 20 000 y 50 000 asientos:

```
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)
```

En el siguiente ejemplo, se demuestra el uso de BETWEEN para valores de fecha:

```
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
```

Tenga en cuenta que, aunque el intervalo de BETWEEN es inclusivo, las fechas tienen un valor de hora predeterminado de 00:00:00. La única fila válida del 3 de enero para la consulta de ejemplo sería una fila con un valor de saletime de `1/3/2008 00:00:00`.

Condición nula

La NULL la condición comprueba si hay valores nulos cuando falta un valor o se desconoce.

Sintaxis

```
expression IS [ NOT ] NULL
```

Argumentos

expresión

Cualquier expresión, como una columna.

IS NULL

Es true cuando el valor de la expresión es nulo y false cuando tiene un valor.

IS NOT NULL

Es false cuando el valor de la expresión es nulo y true cuando tiene un valor.

Ejemplo

Este ejemplo indica cuántas veces la tabla SALES contiene un valor nulo en el campo QTYSOLD:

```
select count(*) from sales
where qtysold is null;
count
-------
0
(1 row)
```

Condición EXISTS

Las condiciones EXISTS realizan pruebas en busca de la existencia de filas en una subconsulta, y devuelve true si una subconsulta devuelve al menos una fila. Si se especifica NOT, la condición devuelve true si una subconsulta no devuelve filas.

Sintaxis

```
[ NOT ] EXISTS (table_subquery)
```

Argumentos

EXISTS

Es true cuando table_subquery (subconsulta_de_tabla) devuelve al menos una fila.

NOT EXISTS

Es true cuando table_subquery (subconsulta_de_tabla) no devuelve filas.

table_subquery (subconsulta_de_tabla)

Una subconsulta que toma el valor de una tabla con una o más columnas y una o más filas.

Ejemplo

Este ejemplo devuelve todos los identificadores de fecha, uno a la vez, para cada fecha que tuvo una venta de cualquier tipo:

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
1827
1828
1829
...
```

Condición IN

Una condición IN prueba un valor de pertenencia en un conjunto de valores o en una subconsulta.

Sintaxis

```
expression [ NOT ] IN (expr_list | table_subquery)
```

Argumentos

expresión

Expresión temporal, de carácter o numérica que se compara con expr_list (lista_de_expresiones) o table_subquery (subconsulta_de_tabla) y debe ser compatible con el tipo de datos de esa lista o subconsulta.

expr_list (lista_de_expresiones)

Una o más expresiones separadas por comas o uno o más conjuntos de expresiones separados por comas entre paréntesis.

table_subquery (subconsulta_de_tabla)

Una subconsulta que toma el valor de una tabla con una o más filas, pero está limitada a una columna en su lista selecta.

IN | NOT IN

IN devuelve true si la expresión es un miembro de la consulta o lista de expresiones. NOT IN devuelve true si la expresión no es un miembro. IN y NOT IN devuelven NULL y no devuelven filas en los siguientes casos: si la expresión genera un valor nulo o si no hay valores de expr_list o table_subquery que coincidan y al menos una de estas filas de comparación genera un valor nulo.

Ejemplos

Las siguientes condiciones son true solo para esos valores enumerados:

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

Optimización para listas IN grandes

Para optimizar el rendimiento de la consulta, una lista IN que incluye más de 10 valores se evalúa internamente como una matriz escalar. Las listas IN con menos de 10 valores se evalúan como una serie de predicados OR. Esta optimización se admite para los tipos de datos SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP y TIMESTAMPTZ.

Observe el resultado de EXPLAIN de la consulta para ver el efecto de esta optimización. Por ejemplo:

```
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)
```

Consultar datos anidados

AWS Clean Rooms ofrece acceso compatible con SQL a datos relacionales y anidados.

AWS Clean Rooms utiliza la notación punteada y el subíndice matricial para navegar por las rutas al acceder a los datos anidados. También habilita la FROM los elementos de la cláusula se repiten sobre matrices y se utilizan en operaciones no anidadas. En los temas siguientes se describen los diferentes patrones de consulta que combinan el uso del tipo de array/struct/map datos con la navegación por rutas y matrices, el desanidamiento y las uniones.

Temas

Navegación

AWS Clean Rooms permite navegar por matrices y estructuras mediante la notación de `[...]` corchetes y puntos, respectivamente. Además, puede combinar la navegación en estructuras utilizando la notación con puntos y matrices con la notación con corchetes.

Example

Por ejemplo, en la siguiente consulta de ejemplo, se presupone que la columna de datos de matriz `c_orders` es una matriz con una estructura y que un atributo se denomina `o_orderkey`.

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

Puede utilizar las notaciones con puntos y corchetes en todos los tipos de consultas, como las de filtrado, combinación y agregación. También puede utilizar estas notaciones en una consulta en la que por lo general hay referencias de columnas.

Example

En el siguiente ejemplo, se utiliza una instrucción SELECT que filtra los resultados.

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0].o_orderkey IS NOT NULL;
```

Example

En el siguiente ejemplo, se utiliza la navegación con corchetes y puntos tanto en las cláusulas GROUP BY como ORDER BY.

```
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;
```

Desanidar consultas

Para deshacer las consultas, AWS Clean Rooms habilita la iteración sobre matrices. Para ello, navega por la matriz utilizando la cláusula FROM de una consulta.

Example

Continuando con el ejemplo anterior, el siguiente ejemplo itera los valores de atributo de `c_orders`.

```
SELECT o FROM customer_orders_lineitem c, c.c_orders o;
```

La sintaxis de desanidamiento es una extensión de la cláusula FROM. En SQL estándar, la cláusula FROM `x (AS) y` significa que y itera cada tupla en relación con `x`. En este caso, `x` hace referencia a una relación e y hace referencia a un alias de relación `x`. Del mismo modo, la sintaxis de desanidamiento con el elemento de cláusula FROM `x (AS) y` significa que y itera cada valor en la expresión de matriz `x`. En este caso, `x` es una expresión de matriz e y es un alias de `x`.

El operando izquierdo también puede utilizar la notación con puntos y corchetes para la navegación normal.

Example

En el ejemplo anterior:

- `customer_orders_lineitem c` es la iteración sobre la tabla base `customer_order_lineitem`
- `c.c_orders o` es la iteración sobre la `c.c_orders array`

Para iterar el atributo `o_lineitems`, que es una matriz dentro de otra matriz, debe añadir varias cláusulas.

```
SELECT o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

AWS Clean Rooms también admite un índice de matriz cuando se itera sobre la matriz mediante AT palabra clave. La cláusula `x AS y AT z` itera la matriz `x` y genera el campo `z`, que es el índice de la matriz.

Example

En el siguiente ejemplo se muestra cómo funciona un índice de matrices.

```
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

En el siguiente ejemplo se itera una matriz escalar.

```
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

En el siguiente ejemplo se itera una matriz de varios niveles. En el ejemplo se utilizan varias cláusulas de desanidamiento para iterar en las matrices más internas. la `f.multi_level_array`, AS la matriz se itera. `multi_level_array` La matriz AS el elemento es la iteración sobre las matrices que contiene. `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)
```

Semántica laxa

De manera predeterminada, las operaciones de navegación en valores de datos anidados devuelven valores nulos en lugar de devolver un error cuando la navegación no es válida. La navegación por objetos no es válida si el valor de datos anidado no es un objeto, o si el valor de datos anidado es un objeto, pero no contiene el nombre del atributo utilizado en la consulta.

Example

Por ejemplo, la siguiente accede a un nombre de atributo no válido de la columna de datos anidados `c_orders`:

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

La navegación por matrices devuelve el valor nulo si el valor de datos anidado no es una matriz o si el índice de la matriz está fuera de límites.

Example

La siguiente consulta devuelve el valor nulo porque `c_orders[1][1]` está fuera de límites.

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

Tipos de introspección

Las columnas de datos anidados admiten funciones de inspección que devuelven el tipo y otra información del tipo relativa al valor. AWS Clean Rooms admite las siguientes funciones booleanas para las columnas de datos anidados:

- 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

Todas estas funciones devuelven un valor false si el valor de entrada es nulo. IS_SCALAR, IS_OBJECT e IS_ARRAY son mutuamente excluyentes y cubren todos los valores posibles, excepto los nulos. Para deducir los tipos correspondientes a los datos, AWS Clean Rooms utiliza la función JSON_TYPEOF, que devuelve el tipo (el nivel superior) del valor de los datos anidados, como se muestra en el siguiente ejemplo:

```
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
```

Historial de documentos de la referencia AWS Clean Rooms de SQL

En la siguiente tabla se describen las versiones de la documentación de la Referencia AWS Clean Rooms SQL.

Para obtener notificaciones sobre las actualizaciones de esta documentación, puede suscribirse a la fuente RSS. Para suscribirse a las actualizaciones RSS, debe tener un complemento de RSS habilitado para el navegador que esté utilizando.

Cambio	Descripción	Fecha
Spark SQL es compatible con Hints	AWS Clean Rooms Spark SQL admite sugerencias de consulta para optimizar el rendimiento de las consultas y reducir los costes de procesamiento.	20 de enero de 2026
Spark SQL es compatible con CACHE TABLE	AWS Clean Rooms Spark SQL admite el comando CACHE TABLE, que permite a los clientes almacenar en caché las tablas existentes o crear y almacenar nuevas tablas a partir de los resultados de las consultas para mejorar el rendimiento de las consultas.	22 de octubre de 2025
Spark SQL admite las funciones FIRST y LAST Window	AWS Clean Rooms Spark SQL admite las siguientes funciones de ventana: FIRST y LAST.	12 de junio de 2025

Actualizaciones de la documentación de las funciones SQL de S	Actualización exclusiva de la documentación para reflejar con precisión las funciones de Spark SQL compatibles. Se ha eliminado la documentación de 25 funciones no compatibles, incluidas <=> operator, SIMILAR TO, LISTAGG y ARRAY_INSERT. Se corrigieron los nombres de las funciones de DATEADD a DATE_ADD, DATEDIFF a DATE_DIFF, ISNULL a IS_NULL e ISNOTNULL a IS_NOT_NULL. Se ha corregido un error tipográfico en los ejemplos de DATE_PART.	20 de mayo de 2025
AWS Clean Rooms Spark SQL	Los clientes ahora pueden ejecutar consultas utilizando algunas condiciones, funciones, comandos y convenciones de SQL compatibles con el motor de análisis SQL de Spark.	29 de octubre de 2024

Comandos y funciones SQL: actualización	Se han agregado ejemplos para la cláusula JOIN, operador de conjunto EXCEPT, expresión condicional CASE y las siguientes funciones: ANY_VALUE, NVL y COALESCE, NULLIF, CAST, CONVERT, CONVERT_TIMEZONE, EXTRACT, MOD, SIGN, CONCAT, FIRST_VALUE y LAST_VALUE.	28 de febrero de 2024
Funciones SQL: actualización	AWS Clean Rooms ahora es compatible con las siguientes funciones SQL: Array, SUPER y VARBYTE. Ahora se admiten las siguientes funciones matemáticas: ACOS, ASIN, ATAN, COT ATAN2, DEXP, PI, POW, RADIANS y SIN. Ahora se admiten las siguientes funciones JSON: CAN_JSON_PARSE, JSON_PARSE y JSON_SERIALIZE.	6 de octubre de 2023
Compatibilidad con tipos de datos anidados	AWS Clean Rooms ahora admite tipos de datos anidados.	30 de agosto de 2023
Reglas de nomenclatura de SQL: actualización	Cambios solo en la documentación para aclarar los nombres de columnas reservadas.	16 de agosto de 2023
Disponibilidad general	La referencia AWS Clean Rooms de SQL ahora está disponible de forma general.	31 de julio de 2023

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.