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 Rooms se 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, las reglas de nomenclatura y los tipos de datos:

Temas

Para comprender los comandos SQL, los tipos de funciones SQL y las condiciones SQL que puede utilizar AWS Clean Rooms, consulte los siguientes temas:

- Comandos SQL en AWS Clean Rooms
- Funciones SQL en AWS Clean Rooms
- Condiciones SQL en AWS Clean Rooms

Para obtener más información AWS 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.

Carácter	Descripción
[ ]	Los corchetes denotan argumentos opcionales. Varios argumentos entre corchetes indican que 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.

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:

- Admiten solo caracteres alfanuméricos, guion bajo (_) o guion medio (-), pero no pueden empezar ni terminar con un guion.
  - (Solo para la regla de análisis personalizada) Pueden usar el signo de dólar ($), pero no pueden usar un patrón que siga una constante de cadena citada entre 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 caracteres de guion (-) consecutivos.
- No pueden empezar por ninguno de los siguiente 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 en AWS Clean Rooms (ni siquiera entre comillas):
  - oid
  - tableoid
  - xmin
  - cmin
  - xmax
  - cmax

- ctid

Literales

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

A continuación se indican las reglas de nomenclatura que se aplican a los literales en AWS Clean Rooms:

- Se admiten literales numéricos, de caracteres, de fecha y hora y de marca temporal.
- Los se admiten los caracteres de control de Unicode TAB, CARRIAGE RETURN (CR) y LINE FEED (LF) de la categoría general de Unicode (Cc).
- La instrucción SELECT no admite referencias directas a literales de la lista de proyección.

  Por ejemplo:

```
SELECT 'test', consumer.first_purchase_day
FROM consumer
INNER JOIN provider2
ON consumer.hashed_email = provider2.hashedemail
```

Palabras reservadas

A continuación se ofrece una lista de las palabras reservadas en AWS Clean Rooms.

AES128	DELTA32KDESC	LEADING	PRIMARY
AES256ALL	DISTINCT	LEFTLIKE	RAW
ALLOWOVER WRITEANALYSE	DO	LIMIT	READRATIO
ANALYZE	DISABLE	LOCALTIME	RECOVERREFERENCES
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

Tipos de datos

Cada valor que AWS Clean Rooms 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.

En la siguiente tabla se enumeran los tipos de datos que puede usar en AWS Clean Rooms las tablas.

Tipo de datos	Alias	Descripción
ARRAY	No aplicable	Tipo de datos anidados de matriz
BIGINT	No aplicable	Entero firmado de ocho bytes
BOOLEAN	BOOL	Booleano lógico (true/false)

Tipo de datos	Alias	Descripción
CHAR	CHARACTER	Cadena de caracteres de longitud fija
FECHA	No aplicable	Fecha de calendario (año, mes, día)
DECIMAL	NUMERIC	Numérico exacto de precisión seleccionable
DOUBLE PRECISION	FLOAT8, FLOAT	Número en coma flotante de precisión doble
INTEGER	INT	Entero firmado de cuatro bytes
MAP	No aplicable	Tipo de datos anidados de mapa
REAL	FLOAT4	Número en coma flotante de precisión única
SMALLINT	No aplicable	Entero firmado de dos bytes
STRUCT	No aplicable	Tipo de datos anidados de estructura
SUPER	No aplicable	Tipo de datos de superconj unto que abarca todos los tipos escalares AWS Clean Rooms , incluidos los tipos complejos, como ARRAY y STRUCTS.
HORA	No aplicable	Hora del día
TIMETZ	No aplicable	Hora del día con zona horaria
VARBYTE	VARBINARY, BINARY VARYING	Valor binario de longitud variable

Tipo de datos	Alias	Descripción
VARCHAR	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

Temas

- Tipos de enteros
- Tipo DECIMAL o NUMERIC
- Notas acerca del uso de las columnas DECIMAL o NUMERIC de 128 bits
- Tipos de números en coma flotante
- Cómputos con valores numéricos

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

Tipos de enteros

Use los tipos de datos SMALLINT, INTEGER y BIGINT para almacenar los números completos de varios rangos. No puede almacenar valores fuera del rango permitido para cada tipo.

Nombre	Almacenamiento	Range
SMALLINT	2 bytes	De -32768 a +32767
INTEGER o INT	4 bytes	De -2147483648 a 2147483647
BIGINT	8 bytes	De -9223372036854775808 a 9223372036854775807

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 una *precisión* y una *escala*:

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

precisión

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 `9999999999999999999` (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 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 Funciones SQL en AWS Clean Rooms.

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.

	SMALLINT	INTEGER	BIGINT	DECIMAL	FLOAT4	FLOAT8
SMALLINT	SMALLINT	INTEGER	BIGINT	DECIMAL	FLOAT8	FLOAT8
INTEGER	INTEGER	INTEGER	BIGINT	DECIMAL	FLOAT8	FLOAT8
BIGINT	BIGINT	BIGINT	BIGINT	DECIMAL	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT4	FLOAT8
FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8

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` y `s1` representan la precisión y escala del primer operando en un cálculo, y `p2` y `s2` representan la precisión y 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 tabla DEC1 con una columna DECIMAL(7,2) se combina con una tabla DEC2 con una columna DECIMAL(15,3) para crear una tabla DEC3. En el esquema de DEC3 se muestra que la columna se convierte en una columna NUMERIC(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 el caso de 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:

- Las formas explícitas generan errores de desbordamiento de tiempo de ejecución cuando los valores de datos específicos no se ajustan a la precisión o escala solicitada y especificada

por la función de formación. 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 se forma como DECIMAL(5,0)
- INTEGER se forma como DECIMAL(10,0)
- BIGINT se forma como 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).

Almacenamiento y rangos

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. 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.

Nombre	Almacenamiento	Rango (ancho de columna)
CHAR o CHARACTER	Longitud de la cadena, incluidos espacios en blanco anteriores o posteriores (si corresponde)	4 096 bytes
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)

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).

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)
```

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, TIMETZ, TIMESTAMP y TIMESTAMPTZ.

Temas

- Literales de fecha, hora y marca temporal
- Literales de intervalo

Almacenamiento y rangos

Nombre	Almacenamiento	Range	Resolución
DATE	4 bytes	De 4713 a.C. a 294276 d.C.	1 día
HORA	8 bytes	00:00:00 a 24:00:00	1 microsegundo
TIMETZ	8 bytes	00:00:00+1459 a 00:00:00+1459	1 microsegundo
MARCA DE TIEMPO	8 bytes	De 4713 a.C. a 294276 d.C.	1 microsegundo
TIMESTAMPTZ	8 bytes	De 4713 a.C. a 294276 d.C.	1 microsegundo

FECHA

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

HORA

Utilice el tipo de datos TIME para almacenar la hora del día.

Las columnas TIME almacenan valores con un máximo de seis dígitos de precisión para las fracciones de segundos.

De manera predeterminada, los valores TIME están en Coordinated Universal Time (UTC, hora universal coordinada) en las tablas del usuario y las tablas del sistema de AWS Clean Rooms .

TIMETZ

Utilice el tipo de datos TIMETZ para almacenar la hora del día con una zona horaria.

Las columnas TIMETZ almacenan valores con un máximo de seis dígitos de precisión para las fracciones de segundos.

De forma predeterminada, los valores de TIMETZ son UTC tanto en las tablas de usuario como en las tablas AWS Clean Rooms del sistema.

MARCA DE TIEMPO

Use el tipo de datos TIMESTAMP para almacenar valores de marca temporal completa que incluyen la fecha y la hora del día.

Las columnas TIMESTAMP almacenan valores con un máximo de seis dígitos de precisión para las fracciones de segundos.

Si inserta una fecha en una columna TIMESTAMP o una fecha con un valor de marca temporal parcial, el valor se convierte de manera implícita a un valor de marca temporal completa. Este valor de marca temporal completo tiene valores predeterminados (00) para las horas, los minutos y los segundos que faltan. Se ignoran los valores de zona horaria en cadenas de entrada.

De forma predeterminada, los valores de TIMESTAMP son UTC tanto en las tablas de usuario como en las tablas del sistema. AWS Clean Rooms

TIMESTAMPTZ

Use el tipo de datos TIMESTAMPTZ para introducir valores de marca temporal completa que incluyan la fecha, la hora del día y una zona horaria. Cuando un valor de entrada incluye una zona horaria, AWS Clean Rooms utiliza la zona horaria para convertir el valor a UTC y almacena el valor de UTC.

Para ver una lista de los nombres de zonas horarias compatibles, ejecute el siguiente comando.

```
select my_timezone_names();
```

Para ver una lista de las abreviaturas de zonas horarias compatibles, ejecute el siguiente comando.

```
select my_timezone_abbrevs();
```

Puede obtener información actualizada acerca de las zonas horarias en la base de datos de zonas horarias de IANA.

La siguiente tabla tiene ejemplos de formatos de zonas horarias.

Formato	Ejemplo
dd mon hh:mi:ss yyyy tz	17 dic 07:37:16 1997 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 EE. UU./Pacífico
yyyy-mm-dd hh:mi:ss+/-tz	1997-12-17 07:37:16-08
dd.mm.yyyy hh:mi:ss tz	17.12.1997 07:37:16.00 PST

Las columnas TIMESTAMPTZ almacenan valores con un máximo de seis dígitos de precisión para las fracciones de segundos.

Si inserta una fecha en una columna TIMESTAMPTZ o una fecha con una marca temporal parcial, el valor se convierte de manera implícita a un valor de marca temporal completa. Este valor de marca temporal completo tiene valores predeterminados (00) para las horas, los minutos y los segundos que faltan.

Los valores TIMESTAMPTZ son UTC en tablas del usuario.

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
```

Ejemplos de marca temporal

Si inserta una fecha en una columna TIMESTAMP o TIMESTAMPTZ, la hora es, por defecto, la medianoche. Por ejemplo, si inserta el literal `20081231`, el valor almacenado es `2008-12-31 00:00:00`.

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

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

Literales de fecha, hora y marca temporal

A continuación se indican las reglas para trabajar con literales de fecha, hora y marca horaria compatibles con. AWS Clean Rooms

Fechas

En la siguiente tabla se muestran las fechas de entrada que son ejemplos válidos de valores de fecha literales que se pueden cargar en las 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
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

Horas de entrada	Descripción (de la parte de la hora)
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

Marcas temporales

La siguiente tabla muestra las marcas de tiempo de entrada que son ejemplos válidos de valores de tiempo literales que se pueden cargar en las tablas. AWS Clean Rooms Todos los literales de fecha válidos pueden combinarse con los siguientes literales de hora.

Marcas temporales de entrada (fechas y horas concatenadas)	Descripción (de la parte de la hora)
20080215 04:05:06.789	4:05 a. m. y 6789 segundos
20080215 04:05:06	4:05 a. m. y 6 segundos
20080215 04:05	4:05 a. m. exactamente
20080215 040506	4:05 a. m. y 6 segundos
20080215 04:05 AM	4:05 a. m. exactamente; a. m. es opcional
20080215 04:05 PM	4:05 p. m. exactamente; el valor de la hora debe ser menor que 12
20080215 16:05	4:05 p. m. exactamente
20080215	Medianoche (por defecto)

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	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.

En los siguientes ejemplos, se muestra cómo now y today trabajan en conjunto con la función DATEADD.

```
select dateadd(day,1,'today');

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

select dateadd(day,1,'now');

date_add
----------------------------
2009-11-17 10:45:32.021394
```

```
(1 row)
```

Literales de intervalo

A continuación se muestran reglas para trabajar con literales que se admiten en AWS Clean Rooms.

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 puede 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 15 de febrero de 2023 y el resultado es el 15 de octubre de 2022.

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

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

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
```

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    |
```

Tipo SUPER

Utilice el tipo de datos SUPER para almacenar datos o documentos semiestructurados como valores.

Los datos semiestructurados no se ajustan a la estructura rígida y tabular del modelo de datos relacionales utilizado en las bases de datos SQL. El tipo de datos SUPER contiene etiquetas que hacen referencia a entidades diferenciadas dentro de los datos. Los tipos de datos SUPER pueden contener valores complejos, como matrices, estructuras anidadas y otras estructuras complejas, que están asociadas a formatos de serialización, como JSON. El tipo de datos SUPER es un conjunto de valores de estructura y matriz sin esquema que abarcan todos los demás tipos escalares de AWS Clean Rooms.

El tipo de datos SUPER admite hasta 1 MB de datos para un campo o un objeto SUPER individuales.

El tipo de datos SUPER presenta las siguientes propiedades:

- Un valor AWS Clean Rooms escalar:
  - un nulo
  - un booleano
  - un número, como smallint, entero, bigint, decimal o de coma flotante (como float4 o float8)
  - un valor de cadena, como varchar o char
- un valor complejo:
  - una matriz de valores, incluidos los escalares o los complejos
  - una estructura, también conocida como tupla u objeto, que es un mapa de nombres y valores de atributos (escalar o complejo)

Cualquiera de los dos tipos de valores complejos contiene sus propios valores escalares o complejos sin ninguna restricción de regularidad.

El tipo de datos SUPER admite la persistencia de datos semiestructurados en un formato sin esquema. Aunque el modelo de datos jerárquico puede cambiar, las versiones antiguas de los datos pueden coexistir en la misma columna SUPER.

Tipo anidado

AWS Clean Rooms admite consultas que incluyan datos con tipos de datos anidados, específicamente los tipos de AWS Glue estructura, matriz y columna del mapa. 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.

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

Tipo VARBYTE

Utilice una columna VARBYTE, VARBINARY o VARYING BINARY para almacenar valores binarios de longitud variable con un límite fijo.

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

El número máximo de bytes (n) puede varias entre 1 a 1 024 000. El valor predeterminado es 64 000.

Algunos ejemplos en los que podría ser conveniente utilizar un tipo de datos VARBYTE son los siguientes:

- Puede unir tablas en columnas VARBYTE.
- Puede crear vistas materializadas que contengan columnas VARBYTE. Se admite la actualización progresiva de las vistas materializadas que contienen columnas VARBYTE. Sin embargo, a diferencia de las funciones agregadas COUNT, MIN y MAX y GROUP BY en columnas VARBYTE, no son compatibles con la actualización progresiva.

Para garantizar que todos los bytes sean caracteres imprimibles, AWS Clean Rooms utiliza el formato hexadecimal para imprimir los valores de VARBYTE. Por ejemplo, el siguiente SQL convierte la cadena hexadecimal 6162 en un valor binario. Aunque el valor devuelto es un valor binario, los resultados aparecen como hexadecimal 6162.

```
select from_hex('6162');

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

AWS Clean Rooms admite la conversión entre VARBYTE y los siguientes tipos de datos:

- CHAR
- VARCHAR
- SMALLINT
- INTEGER
- BIGINT

La siguiente instrucción SQL convierte un string VARCHAR en un VARBYTE. Aunque el valor devuelto es un valor binario, los resultados aparecen como hexadecimal `616263`.

```
select 'abc'::varbyte;

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

La siguiente instrucción SQL convierte un valor CHAR de una columna e un VARBYTE. En este ejemplo, se crea una tabla con una columna CHAR (10) (c), se insertan valores de caracteres más cortos que la longitud de 10. La conversión obtenida completa el resultado con un espacio de caracteres (hex “20”) hasta el tamaño de columna definido. Aunque el valor devuelto es un valor binario, los resultados se muestran como hexadecimal.

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

La siguiente instrucción SQL convierte una cadena SMALLINT en un VARBYTE. Aunque el valor devuelto es un valor binario, los resultados aparecen como hexadecimal `0005`, es decir, dos bytes o cuatro caracteres hexadecimales.

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

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

La siguiente instrucción SQL convierte un INTEGER en un VARBYTE. Aunque el valor devuelto es un valor binario, los resultados aparecen como hexadecimal `00000005`, es decir, cuatro bytes u ocho caracteres hexadecimales.

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

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

La siguiente instrucción SQL convierte un BIGINT en un VARBYTE. Aunque el valor devuelto es un valor binario, los resultados aparecen como hexadecimal `0000000000000005`, es decir, ocho bytes o 16 caracteres hexadecimales.

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

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

Limitaciones a la hora de utilizar el tipo de datos VARBYTE con AWS Clean Rooms

Las siguientes son limitaciones al usar el tipo de datos VARBYTE con: AWS Clean Rooms

- AWS Clean Rooms admite el tipo de datos VARBYTE solo para archivos Parquet y ORC.
- AWS Clean Rooms el editor de consultas aún no es totalmente compatible con el tipo de datos VARBYTE. Por ello, se debe utilizar un cliente SQL diferente cuando trabaje con expresiones VARBYTE.

  Una solución para utilizar el editor de consultas es que, si la longitud de sus datos es inferior a 64 KB y el contenido es un UTF-8 válido, se pueden convertir los valores VARBYTE en VARCHAR, por ejemplo:

  ```
  select to_varbyte('6162', 'hex')::varchar;
  ```
- No se pueden utilizar tipos de datos VARBYTE con funciones definidas por el usuario (UDF) de Python o Lambda.
- No se puede crear una columna HLLSKETCH a partir de una columna VARBYTE ni utilizar APPROXIMATE COUNT DISTINCT en una columna VARBYTE.

Conversión y compatibilidad de tipos

El siguiente análisis describe cómo funcionan las reglas de conversión de tipos y la compatibilidad de tipos de datos en AWS Clean Rooms.

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 determinado valor y un determinado tipo de datos. 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.
- Al convertir DATE o TIMESTAMP a TIMESTAMPTZ, o bien, convertir TIME a TIMETZ, la zona horaria se establece como la zona de la sesión actual. Por defecto, la zona horaria de la sesión es UTC.

- De manera similar, TIMESTAMPTZ se convierte a DATE, TIME o TIMESTAMP en función de la zona horaria de la sesión actual. Por defecto, la zona horaria de la sesión es UTC. Después de la conversión, se elimina la información de la zona horaria.
- Las cadenas de caracteres que representan una marca temporal con zona horaria especificada se convierten a TIMESTAMPTZ con la zona horaria de la sesión actual, que es la UTC de forma predeterminada. Del mismo modo, las cadenas de caracteres que representan una hora con zona horaria especificada se convierten a TIMETZ con la zona horaria de la sesión actual, que es la UTC de forma predeterminada.

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	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
CHAR	VARCHAR
FECHA	CHAR
	VARCHAR
	MARCA DE TIEMPO
	TIMESTAMPTZ

Del tipo	Al tipo
DECIMAL (NUMERIC)	BIGINT
	CHAR
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
DOUBLE PRECISION (FLOAT8)	BIGINT
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
INTEGER (INT)	BIGINT
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	REAL (FLOAT4)

Del tipo	Al tipo
	SMALLINT
	VARCHAR
REAL (FLOAT4)	BIGINT
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	SMALLINT
	VARCHAR
SMALLINT	BIGINT
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT)
	REAL (FLOAT4)
	VARCHAR
MARCA DE TIEMPO	CHAR
	FECHA
	VARCHAR
	TIMESTAMPTZ

Del tipo	Al tipo
	HORA
TIMESTAMPTZ	CHAR
	FECHA
	VARCHAR
	MARCA DE TIEMPO
	TIMETZ
HORA	VARCHAR
	TIMETZ
TIMETZ	VARCHAR
	HORA

> **Note**
>
> Las conversiones implícitas entre TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ 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.

Comandos SQL en AWS Clean Rooms

Los siguientes comandos SQL son compatibles con AWS Clean Rooms:

Temas

SELECT

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

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

Temas

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

Parámetros

Número TOP

TOP toma como argumento un entero positivo que define el número de filas que se devuelven al cliente. El comportamiento de la cláusula TOP es idéntico al de la cláusula LIMIT. El número de filas que devuelve es fijo, pero el conjunto de filas no lo es. Para que se devuelva un conjunto de filas constante, use TOP o LIMIT en combinación con una cláusula ORDER BY.

DISTINCT

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

expresión

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. Puede usar un alias en otras expresiones definidas después de él en la misma lista de destino.

Notas de uso

TOP es una extensión SQL; TOP proporciona una alternativa al comportamiento de LIMIT. No se puede usar TOP y LIMIT en la misma consulta.

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 )
```

Parámetros

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 sea compatible. Consulte SELECT.

Notas de uso

Puede usar una cláusula WITH en las siguientes instrucciones SQL:

- SELECT, WITH, UNION, INTERSECT y EXCEPT

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
- Parámetros
- Notas de uso
- Ejemplos de JOIN

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
```

Parámetros

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.

Ejemplos de 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. Para obtener más información acerca de la sintaxis de la cláusula JOIN, consulte Parámetros.

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
```

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 LISTID 2 y 3 no tienen ventas.

```
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
```

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 LISTID 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
```

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 LISTID 2 y 3 no tienen ventas.

```
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). Solo se encuentran en los resultados filas que no dan lugar a ninguna venta (LISTID 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
```

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 cruzada o cartesiana de la tabla LISTING y la tabla SALES con un predicado para limitar los resultados. Esta consulta coincide con los valores de columna LISTID de la tabla SALES y la tabla LISTING para LISTID 1, 2, 3, 4 y 5 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
```

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
```

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
```

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
```

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 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 Funciones SQL en AWS Clean Rooms.

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 de 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 UNION, INTERSECT y EXCEPT se utilizan para comparar y fusionar los resultados de dos expresiones de consulta diferentes. 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

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

Parámetros

consulta

Una expresión de consulta que corresponde, en la forma de su lista de selección, a una segunda expresión de consulta que sigue el operador UNION, INTERSECT o EXCEPT. 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 conversiones implícitas 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
```

UNION

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

INTERSECT

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.

EXCEPT | MINUS

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. MINUS y EXCEPT son sinónimos exactos.

ALL

La palabra clave ALL conserva cualquier fila duplicada que UNION produce. El comportamiento predeterminado cuando no se usa la palabra clave ALL es descartar todos estos duplicados. No se admiten las expresiones INTERSECT ALL, EXCEPT ALL y MINUS ALL.

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 específica de ID de eventos, la consulta devuelve 0 o más filas para cada venta asociada a cada evento, y 0 o 1 fila para cada lista de ese evento. Los ID de eventos son únicos en cada fila de las tablas LISTING y EVENT, pero es posible que haya varias ventas para la misma combinación de ID de lista y evento en la tabla SALES.

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. Las otras dos listas, 501 y 502, no produjeron ventas, por lo que la única fila que la consulta produce para estos ID de lista provienen de la tabla LISTING (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 específica de ID de eventos, la consulta devuelve 0 o más filas para cada venta asociada a cada evento, y 0 o 1 fila para cada lista de ese evento. Los ID de eventos son únicos en cada fila de las tablas LISTING y EVENT, pero es posible que haya varias ventas para la misma combinación de ID de lista y evento en la tabla SALES.

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. Las otras dos listas, 501 y 502, no produjeron ventas, por lo que la única fila que la consulta produce para estos ID de lista provienen de la tabla LISTING (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 11 filas siguientes:

```
 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
- Parámetros
- Notas de uso

- Ejemplos con ORDER BY

Sintaxis

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

Parámetros

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 Ejemplos de JOIN.

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 se AWS Clean Rooms pueden decorrelacionar y que no 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);
```

Funciones SQL en AWS Clean Rooms

AWS Clean Rooms admite las siguientes funciones de SQL:

Temas

Funciones de agregación

AWS Clean Rooms admite las siguientes funciones agregadas:

Temas

- Función MEDIAN
- Función MIN
- Función PERCENTILE_CONT
- 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 ( [ DISTINCT | ALL ] expression )
```

Argumentos

DISTINCT | ALL

Especifique DISTINCT u ALL para devolver cualquier valor de los valores de expresión de entrada. El argumento DISTINCT no tiene ningún efecto y se pasa por alto.

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:

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

- VARCHAR
- FECHA
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

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` lugar donde `eventname` esté`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` lugar donde `eventname` sea `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 APPROXIMATE PERCENTILE_DISC

APPROXIMATE PERCENTILE_DISC es una función de distribución inversa que asume un modelo de distribución discreta. Toma un valor percentil y una especificación de ordenación, y devuelve un elemento del conjunto dado. La aproximación permite que la función se ejecute mucho más rápido, con un margen de error relativamente bajo de alrededor del 0,5 %.

En un percentil dado, APPROXIMATE PERCENTILE_DISC utiliza un algoritmo de resumen de cuartiles para aproximar el percentil discreto de la expresión en la cláusula ORDER BY. APPROXIMATE PERCENTILE_DISC devuelve el menor valor de distribución acumulado (con respecto a la misma especificación de ordenación) que sea mayor o igual que el percentil.

APPROXIMATE PERCENTILE_DISC es una función específica del nodo de computación. La función devuelve un error si la consulta no hace referencia a una tabla definida por el usuario o a una tabla AWS Clean Rooms del sistema.

Sintaxis

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

Argumentos

percentil

Constante numérica entre 0 y 1. Los valores nulos se ignoran en el cálculo.

WITHIN GROUP (ORDER BY expr)

Cláusula que especifica valores numéricos o de fecha/hora para ordenar y calcular el percentil.

Devuelve

El mismo tipo de datos que la expresión ORDER BY en la cláusula WITHIN GROUP.

Notas de uso

Si la instrucción APPROXIMATE PERCENTILE_DISC incluye una cláusula GROUP BY, el conjunto de resultados es limitado. El límite varía según el tipo de nodo y la cantidad de nodos. Si se supera el límite, la función falla y devuelve el siguiente mensaje de error.

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

Si necesita evaluar más grupos que los permitidos, considere usar Función PERCENTILE_CONT.

Ejemplos

El siguiente ejemplo devuelve la cantidad de ventas, las ventas totales y el valor del décimo quinto percentil para las 10 primeras fechas.

```
select top 10 date.caldate,
count(totalprice), sum(totalprice),
approximate percentile_disc(0.5)
within group (order by totalprice)
from listing
join date on listing.dateid = date.dateid
group by date.caldate
order by 3 desc;

caldate    | count | sum        | percentile_disc
-----------+-------+------------+----------------
2008-01-07 |   658 | 2081400.00 |         2020.00
2008-01-02 |   614 | 2064840.00 |         2178.00
2008-07-22 |   593 | 1994256.00 |         2214.00
2008-01-26 |   595 | 1993188.00 |         2272.00
2008-02-24 |   655 | 1975345.00 |         2070.00
2008-02-04 |   616 | 1972491.00 |         1995.00
2008-02-14 |   628 | 1971759.00 |         2184.00
2008-09-01 |   600 | 1944976.00 |         2100.00
2008-07-29 |   597 | 1944488.00 |         2106.00
2008-07-23 |   592 | 1943265.00 |         1974.00
```

Función 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

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
```

Funciones COUNT y COUNT DISTINCT

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

Sintaxis

```
COUNT (column)
```

```
COUNT (DISTINCT column)
```

Argumentos

column

La columna de destino sobre la que opera la función.

Tipos de datos

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

La COUNT DISTINCT función regresaBIGINT.

Ejemplos

Cuente todos los usuarios del estado de Florida.

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

Cuente todos los ID únicos de los locales de la EVENT tabla.

```
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.
- APPROXIMATE COUNT DISTINCT realiza un cálculo aproximado del número de valores distintos de NULL diferentes de una columna o una expresión.

Sintaxis

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

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

```
APPROXIMATE COUNT ( DISTINCT 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.

APPROXIMATE

Cuando se usa con APPROXIMATE, la función COUNT DISTINCT usa un HyperLogLog algoritmo para aproximar el número de valores distintos que no son NULOS en una columna o expresión. Las consultas que utilizan la palabra clave APPROXIMATE se ejecutan mucho más rápido, con un margen de error relativamente bajo de alrededor del 2 %. La aproximación se justifica para consultas que devuelven una cantidad grande de valores distintos, millones o más por consulta, o por grupo, si es que hay una cláusula de grupo. Para conjuntos más pequeños de valores distintos, en miles, la aproximación puede ser más lenta que un conteo preciso. APPROXIMATE solo se puede usar con COUNT DISTINCT.

Tipo de retorno

La función COUNT devuelve BIGINT.

Ejemplos

Contar 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
```

Contar todos los ID únicos de lugares de la tabla EVENT:

```
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
...
```

En los siguientes ejemplos, se comparan los valores de retorno y los tiempos de ejecución para COUNT y APPROXIMATE COUNT.

```
select  count(distinct pricepaid) from sales;
```

```
count
-------
  4528


Time: 48.048 ms


select approximate count(distinct pricepaid) from sales;

count
-------
  4553


Time: 21.728 ms
```

Función LISTAGG

Para cada grupo de una consulta, la función de agregación LISTAGG ordena las filas de ese grupo según la expresión ORDER BY y, luego, concatena los valores en una sola cadena.

LISTAGG es una función específica del nodo de computación. La función devuelve un error si la consulta no hace referencia a una tabla definida por el usuario o a una tabla AWS Clean Rooms del sistema.

Sintaxis

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

Argumentos

DISTINCT

(Opcional) Una cláusula que elimina los valores duplicados de la expresión especificada antes de la concatenación. Se omiten los espacios posteriores y, por tanto, las cadenas `'a'` y `'a '` se tratan como duplicados. LISTAGG usa el primer valor que se encuentra. Para obtener más información, consulte Importancia de los espacios en blancos anteriores y posteriores.

expresión_de_agregación

Toda expresión válida (como un nombre de columna) que proporcione los valores para la agregación. Se ignoran los valores NULL y las cadenas vacías.

delimiter

(Opcional) La constante de cadena que separa los valores concatenados. El valor predeterminado es NULL.

AWS Clean Rooms admite cualquier cantidad de espacios en blanco iniciales o finales alrededor de una coma o dos puntos opcionales, así como una cadena vacía o cualquier número de espacios.

Estos son algunos ejemplos de valores válidos:

```
', '
': '
' '
```

WITHIN GROUP (ORDER BY order_list)

(Opcional) Una cláusula que especifica el orden de los valores agregados.

Devuelve

VARCHAR(MAX). Si el conjunto de resultados es mayor que el tamaño máximo VARCHAR (64K - 1 o 65535), LISTAGG devuelve el siguiente error:

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

Notas de uso

Si una instrucción incluye varias funciones LISTAGG que usan cláusulas WITHIN GROUP, cada cláusula WITHIN GROUP debe usar los mismos valores ORDER BY.

Por ejemplo, la siguiente instrucción devolverá un error.

```
select listagg(sellerid)
within group (order by dateid) as sellers,
```

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

Las siguientes instrucciones se ejecutarán correctamente.

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

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

Ejemplos

En el siguiente ejemplo, se agrega el ID de vendedores ordenados por ID de vendedor.

```
select listagg(sellerid, ', ') within group (order by sellerid) from sales
where eventid = 4337;
listagg

------------------------------------------------------------------------------------------------
380, 380, 1178, 1178, 1178, 2731, 8117, 12905, 32043, 32043, 32043, 32432, 32432,
 38669, 38750, 41498, 45676, 46324, 47188, 47188, 48294
```

En el siguiente ejemplo, se usa DISTINCT para devolver una lista de ID de vendedor distintos.

```
select listagg(distinct sellerid, ', ') within group (order by sellerid) from sales
where eventid = 4337;

listagg

-----------------------------------------------------------------------------------------
380, 1178, 2731, 8117, 12905, 32043, 32432, 38669, 38750, 41498, 45676, 46324, 47188,
 48294
```

En el siguiente ejemplo, se agrega el ID de vendedores ordenados por fecha.

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

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

El siguiente ejemplo devuelve una lista separada con barras de fechas de ventas para el comprador B.

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

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

El siguiente ejemplo devuelve una lista separada por comas de ID de ventas para cada ID de comprador.

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

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

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 corresponde a uno de los siguientes tipos de datos:

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

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

Calcula el valor mediano para el rango de valores. Se ignoran los valores NULL en el rango.

MEDIAN es una función de distribución inversa que asume un modelo de distribución continua.

MEDIAN es un caso especial de PERCENTILE_CONT(.5).

MEDIAN es una función específica del nodo de computación. La función devuelve un error si la consulta no hace referencia a una tabla definida por el usuario o a una tabla del sistema. AWS Clean Rooms

Sintaxis

```
MEDIAN ( median_expression )
```

Argumentos

expresión_de_mediana

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

Tipos de datos

El tipo de valor devuelto viene determinado por el tipo de datos de expresión_de_mediana. En la tabla siguiente, se muestra el tipo de valor devuelto para cada tipo de datos de expresión_de_mediana.

Tipo de entrada	Tipo de retorno
NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
FECHA	FECHA
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

Notas de uso

Si el argumento expresión_de_mediana es un tipo de dato DECIMAL definido con la precisión máxima de 38 dígitos, es posible que MEDIAN devuelva un resultado impreciso o un error. Si el valor de retorno de la función MEDIAN supera los 38 dígitos, el resultado se trunca para adaptarse, lo que genera pérdida de precisión. Si, durante la interpolación, un resultado intermedio supera la precisión máxima, se produce un desbordamiento numérico y la función devuelve un error. Para evitar estas condiciones, recomendamos usar un tipo de dato con menor precisión o emitir el argumento median_expression con una precisión menor.

Si una instrucción incluye varias llamadas a funciones de agregación basadas en ordenación (LISTAGG, PERCENTILE_CONT o MEDIAN), todas deben usar los mismos valores ORDER BY. Tenga en cuenta que MEDIAN aplica un comando ORDER BY implícito en el valor de expresión.

Por ejemplo, la siguiente instrucción devuelve un error.

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

An error occurred when executing the SQL command:
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (pricepaid)
from sales group by salesid, pricepai...

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

La siguiente instrucción se ejecuta correctamente.

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

Ejemplos

En el siguiente ejemplo, se muestra que MEDIAN produce los mismos resultados que PERCENTILE_CONT(0.5).

```
select top 10  distinct sellerid, qtysold,
percentile_cont(0.5) within group (order by qtysold),
median (qtysold)
from sales
group by sellerid, qtysold;

sellerid | qtysold | percentile_cont | median
---------+---------+-----------------+-------
       1 |       1 |             1.0 |    1.0
       2 |       3 |             3.0 |    3.0
       5 |       2 |             2.0 |    2.0
       9 |       4 |             4.0 |    4.0
      12 |       1 |             1.0 |    1.0
      16 |       1 |             1.0 |    1.0
      19 |       2 |             2.0 |    2.0
      19 |       3 |             3.0 |    3.0
      22 |       2 |             2.0 |    2.0
```

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

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 corresponde a uno de los siguientes tipos de datos:

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

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 PERCENTILE_CONT

PERCENTILE_CONT es una función de distribución inversa que asume un modelo de distribución continua. Toma un valor percentil y una especificación de ordenación, y devuelve un valor interpolado que se encuadraría dentro de ese valor percentil dado respecto de la especificación de ordenación.

PERCENTILE_CONT computa una interpolación linear entre valores después de ordenarlos. Utilizando el valor percentil `(P)` y la cantidad de filas no nulas `(N)` en el grupo de agregación,

la función computa la cantidad de filas después de ordenar las filas según la especificación de ordenación. Esta cantidad de filas `(RN)` se computa según la fórmula `RN = (1+ (P*(N-1))`. El resultado final de la función de agregación se computa por interpolación lineal entre los valores de filas en números de filas `CRN = CEILING(RN)` y `FRN = FLOOR(RN)`.

El resultado final será el siguiente.

Si `(CRN = FRN = RN)`, entonces el resultado es `(value of expression from row at RN)`

De lo contrario, el resultado es el siguiente:

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

PERCENTILE_CONT es una función específica del nodo de computación. La función devuelve un error si la consulta no hace referencia a una tabla definida por el usuario o a una tabla AWS Clean Rooms del sistema.

Sintaxis

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

Argumentos

percentil

Constante numérica entre 0 y 1. Los valores nulos se ignoran en el cálculo.

WITHIN GROUP (ORDER BY expr)

Valores numéricos o de fecha/hora específicos para ordenar y calcular el percentil.

Devuelve

El tipo de retorno se determina por el tipo de datos de la expresión ORDER BY en la cláusula WITHIN GROUP. En la tabla siguiente, se muestra el tipo de retorno para cada tipo de datos de la expresión ORDER BY.

Tipo de entrada	Tipo de retorno
SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
FECHA	FECHA
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

Notas de uso

Si la expresión ORDER BY es un tipo de dato DECIMAL definido con la precisión máxima de 38 dígitos, es posible que PERCENTILE_CONT devuelva un resultado impreciso o un error. Si el valor de retorno de la función PERCENTILE_CONT supera los 38 dígitos, el resultado se trunca para adaptarse, lo que genera pérdida de precisión. Si, durante la interpolación, un resultado intermedio supera la precisión máxima, se produce un desbordamiento numérico y la función devuelve un error. Para evitar estas condiciones, recomendamos usar un tipo de dato con menor precisión o emitir la expresión ORDER BY a una precisión menor.

Si una instrucción incluye varias llamadas a funciones de agregación basadas en ordenación (LISTAGG, PERCENTILE_CONT o MEDIAN), todas deben usar los mismos valores ORDER BY. Tenga en cuenta que MEDIAN aplica un comando ORDER BY implícito en el valor de expresión.

Por ejemplo, la siguiente instrucción devuelve un error.

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

An error occurred when executing the SQL command:
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (pricepaid)
from sales group by salesid, pricepai...
```

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

La siguiente instrucción se ejecuta correctamente.

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

Ejemplos

En el siguiente ejemplo, se muestra que MEDIAN produce los mismos resultados que PERCENTILE_CONT(0.5).

```
select top 10  distinct sellerid, qtysold,
percentile_cont(0.5) within group (order by qtysold),
median (qtysold)
from sales
group by sellerid, qtysold;

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

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 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**
>
> 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 (column)
```

```
SUM (DISTINCT 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

Tipos de datos

Los tipos de argumentos que admite la función SUM son SMALLINT, INTEGER, BIGINT,DECIMAL y DOUBLE.

La función SUM admite los siguientes tipos de retorno:

- BIGINT para argumentos BIGINT, SMALLINT y INTEGER
- DOUBLE para argumentos 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 SUM con un argumento DECIMAL es 38. La escala del resultado es la misma que la escala del argumento. Por ejemplo, un SUM de una columna DEC(5,2) devuelve un tipo datos DEC(38,2).

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
- función array_concat
- Función array_flatten
- Función get_array_length
- Función split_to_array
- función de submatriz

Función array

Crea una matriz cuyo tipo de datos es SUPER.

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 array devuelve el tipo de datos SUPER.

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_concat

La función array_concat concatena dos matrices para crear una matriz que contiene todos los elementos de la primera matriz seguida de todos los elementos de la segunda matriz. Los dos argumentos deben ser matrices válidas.

Sintaxis

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

Argumentos

super_expr1

El valor que especifica la primera de las dos matrices que se van a concatenar.

super_expr2

El valor que especifica la segunda de las dos matrices que se van a concatenar.

Tipo de retorno

La función array_concat devuelve un valor de datos SUPER.

Ejemplo

El siguiente ejemplo muestra la concatenación de dos matrices del mismo tipo y la concatenación de dos matrices de distintos tipos.

```
-- concatenating two arrays
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY(10003,10004));
              array_concat
-----------------------------------
 [10001,10002,10003,10004]
(1 row)

-- concatenating two arrays of different types
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY('ab','cd'));
          array_concat
------------------------------
 [10001,10002,"ab","cd"]
(1 row)
```

Función array_flatten

Fusiona varias matrices en una sola matriz de tipo SUPER.

Sintaxis

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

Argumentos

super_expr1,super_expr2

Una expresión SUPER válida en forma de matriz.

Tipo de retorno

La función array_flatten devuelve un valor de datos SUPER.

Ejemplo

En el siguiente ejemplo, se muestra una función array_flatten.

```
SELECT ARRAY_FLATTEN(ARRAY(ARRAY(1,2,3,4),ARRAY(5,6,7,8),ARRAY(9,10)));
     array_flatten
------------------------
 [1,2,3,4,5,6,7,8,9,10]
(1 row)
```

Función get_array_length

Devuelve la longitud de la matriz especificada. La función GET_ARRAY_LENGTH devuelve la longitud de una matriz SUPER dada una ruta de objeto o matriz.

Sintaxis

```
get_array_length( super_expr )
```

Argumentos

super_expr

Una expresión SUPER válida en forma de matriz.

Tipo de retorno

La función get_array_length devuelve un BIGINT.

Ejemplo

En el siguiente ejemplo, se muestra una función get_array_length.

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

Función split_to_array

Utiliza un delimitador como parámetro opcional. Si ningún delimitador está presente, entonces el valor predeterminado es la coma.

Sintaxis

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

Argumentos

string

La cadena de entrada que se dividirá.

delimiter

Un valor opcional en el que se dividirá la cadena de entrada. El valor predeterminado es una coma.

Tipo de retorno

La función split_to_array devuelve un valor de datos SUPER.

Ejemplo

En el siguiente ejemplo, se muestra una función split_to_array.

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

función de submatriz

Manipula las matrices para devolver un subconjunto de las matrices de entrada.

Sintaxis

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

Argumentos

super_expr

Una expresión válida SUPER en forma de matriz.

start_position

La posición dentro de la matriz desde donde comienza la extracción, con punto de partida en la posición de índice 0. Con una posición negativa, se cuenta hacia atrás desde el final de la matriz.

longitud

La cantidad de elementos para extraer (la longitud de la subcadena).

Tipo de retorno

La función de submatriz devuelve un valor de datos SUPER.

Ejemplo

El siguiente es un ejemplo de una función de submatriz.

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

Expresiones condicionales

AWS Clean Rooms admite las siguientes expresiones condicionales:

Temas

Expresión condicional CASE

La expresión CASE es una expresión condicional similar a las instrucciones if/then/else 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;
```

Funciones GREATEST y LEAST

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)
```

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 retorno

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"
```

Función NVL2

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 retorno

El tipo de retorno NVL2 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 el caso de NVL2, el valor devuelto tendrá el valor del parámetro not_null_return_value o del parámetro null_return_value, el que haya seleccionado la función, pero tendrá el tipo de datos de not_null_return_value.

Por ejemplo, si se asume que column1 es NULL, las siguientes consultas devolverán el mismo valor. No obstante, el tipo de datos de valor de retorno DECODE será INTEGER y el tipo de datos del valor de retorno NVL2 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

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. El nombre predeterminado de la columna del resultado NULLIF es el nombre de columna 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 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. Para cada una de estas funciones, el primer argumento es siempre el valor que se va a formatear y el segundo argumento contiene la plantilla para el nuevo formato. AWS Clean Rooms admite varias funciones de formato de tipos de datos.

Temas

- Función CAST
- Función CONVERT
- TO_CHAR
- Función TO_DATE
- TO_NUMBER
- Cadenas de formatos de fecha y hora
- Cadenas de formatos numéricos

- Caracteres de formato de estilo Teradata para datos numéricos

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.

La función CAST es muy similar a the section called “CONVERT”, ya que ambas convierten un tipo de datos a otro, pero tienen nombres distintos.

Algunas tipos de datos requieren una conversión explícita a otros tipos de datos utilizando las funciones CAST o CONVERT. Otros tipos de datos se pueden convertir implícitamente, como parte de otro comando, sin utilizar CAST ni CONVERT. 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 )
expression :: 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 contener cadenas en blanco o vacías.

type

Uno de los tipos de datos admitidosTipos de datos, excepto para los tipos de datos VARBYTE, BINARY y BINARY VARIING.

Tipo de retorno

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 un control adicional sobre el formato de salida utilizandoTO_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 | 78800000000000000000000.00
       8 | 19700000000000000000000.00
       9 | 59100000000000000000000.00

(9 rows)
```

Función CONVERT

Al igual que laFunción CAST, la función CONVERT convierte un tipo de datos en otro tipo de datos compatible. Por ejemplo, puede convertir una cadena en una fecha o un tipo numérico en una cadena. CONVERT 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.

Determinados tipos de datos requieren una conversión explícita a otros tipos de datos con la función CONVERT. Otros tipos de datos se pueden convertir implícitamente, como parte de otro comando, sin utilizar CAST ni CONVERT. Consulte Conversión y compatibilidad de tipos.

Sintaxis

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

Argumentos

type

Uno de los tipos de datos compatiblesTipos de datos, excepto los tipos de datos VARBYTE, BINARY y BINARY VARIANT.

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 contener cadenas en blanco o vacías.

Tipo de retorno

CONVERT 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 CONVERT(decimal(2,1), 123.456);
> ```
>
> o una conversión a un valor de INTEGER que genera un desbordamiento:
>
> ```
> SELECT CONVERT(smallint, 12345678);
> ```

Ejemplos

En la siguiente consulta, se utiliza la función CONVERT para convertir una columna de decimales en enteros

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

En este ejemplo, se convierte un entero en una cadena de caracteres.

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

En este ejemplo, la fecha y la hora actuales se convierten en un tipo de datos de carácter variable:

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

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

En este ejemplo, se convierte la columna saletime solo en la hora y se eliminan las fechas de cada fila.

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

En el siguiente ejemplo, se convierten datos de caracteres variable en un objeto datetime.

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

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 DECIMALES de 128 bits.

formato

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 retorno

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. A continuación, la diferencia se redondea hacia arriba y se convierte en un número romano, como 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 añade el símbolo de la divisa a los valores de diferencia que se muestran en la `to_char` columna:

```
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(string, format)
```

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

Argumentos

string

La cadena que se convertirá.

formato

Un literal de cadena que define el formato de la entrada cadena, en términos de sus partes de fecha. Para obtener una lista de los formatos válidos para día, mes y año, consulte Cadenas de formatos de fecha y hora.

is_strict

Un valor booleano opcional que especifica si se devuelve un error si un valor de fecha de entrada se encuentra fuera de rango. Cuando is_strict se configura como TRUE, se devuelve un error si hay un valor fuera de rango. Si is_strict se configura como FALSE, que es el valor predeterminado, se aceptan valores de desbordamiento.

Tipo de retorno

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 Mon YYYY');

to_date
------------
2001-10-02
(1 row)
```

La siguiente instrucción SQL convierte la cadena `20010631` en una fecha.

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

El resultado es 1.° de julio de 2001, ya que solo hay 30 días en junio.

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

La siguiente instrucción SQL convierte la cadena `20010631` en una fecha:

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

El resultado es de error, ya que solo hay 30 días en junio.

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

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.

formato

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 retorno

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
```

Cadenas de formatos de fecha y hora

Las siguientes cadenas con formato de fecha y hora se aplican a funciones como TO_CHAR. Estas cadenas pueden tener separadores de fecha y hora (como "`-`", "`/`" o "`:`·) y las siguientes "partes de fecha" y "partes de hora".

Para ver ejemplos de cómo formatear fechas como cadenas, consulte TO_CHAR.

Partes de fecha o de hora	Significado
BC o B.C., AD o A.D., b.c. o bc, ad o a.d.	Indicadores de era en mayúsculas o minúsculas
CC	Número de siglo en dos dígitos
YYYY, YYY, YY, Y	Número de año en 4 dígitos, 3 dígitos, 2 dígitos, 1 dígito
Y,YYY	Número de año en 4 dígitos con coma
IYYY, IYY, IY, I	Número de año según la Organización Internacional de Normalización (ISO), en 4 dígitos, 3 dígitos, 2 dígitos y 1 dígito
Q	Número de trimestre (1 a 4)
MONTH, Month, month	Nombre del mes (mayúsculas, mayúsculas y minúsculas, minúsculas, con un espacio en blanco de 9 caracteres)

Partes de fecha o de hora	Significado
MON, Mon, mon	Nombre del mes abreviado (mayúsculas, mayúsculas y minúsculas, minúsculas, con un espacio en blanco de 3 caracteres)
MM	Número de mes (01-12)
RM, rm	Número de mes en números romanos (I a XII, donde I es enero, ya sea en mayúsculas o minúsculas)
W	Semana del mes (1 a 5; la primera semana comienza el primer día del mes).
WW	Número de semana del año (1 a 53; la primera semana comienza el primer día del año).
IW	Número de semana del año según la ISO (el primer jueves del nuevo año está en la semana 1.)
DAY, Day, day	Nombre del día (mayúsculas, mayúsculas y minúsculas, minúsculas, con un espacio en blanco de 9 caracteres)
DY, Dy, dy	Nombre del día abreviado (mayúsculas, mayúsculas y minúsculas, minúsculas, con un espacio en blanco de 3 caracteres)
DDD	Día del año (001 a 366)
IDDD	Día del año según la numeración de semanas de ISO 8601 (001-371; el primer día del año es el lunes de la primera semana ISO)
DD	Día del mes como un número (01 a 31)

Partes de fecha o de hora	Significado
D	Día de la semana (1 a 7; el domingo es el día 1) **Note** La parte de fecha D se comporta de manera diferente a la parte de fecha día (DOW) que se usa para las funciones de fecha y hora DATE_PART y EXTRACT. DOW se basa en los números enteros de 0 a 6, con el domingo como 0. Para obtener más información, consulte Partes de fecha para funciones de fecha o marca temporal.
ID	Día de la semana de ISO 8601, de lunes (1) a domingo (7)
J	Día juliano (días desde el 1.º de enero de 4712 AC)
HH24	Hora (formato de 24 horas, de 00 a 23)
HH o HH12	Hora (formato de 12 horas, de 01 a 12)
MI	Minutos (00 a 59)
SS	Segundos (00 a 59)
MS	Milisegundos (0,000)
US	Microsegundos (0,000000)
AM o PM, A.M. o P.M., a.m. o p.m., am o pm	Indicadores meridianos en mayúsculas o minúsculas (para formato de 12 horas)

Partes de fecha o de hora	Significado
TZ, tz	Abreviación de la zona horaria en mayúsculas o minúsculas; válido solo para TIMESTAMPTZ
OF	Desplazamiento del huso UTC; válido solo para TIMESTAMPTZ

> **Note**
>
> Tiene que incluir los separadores de datetime (como '-', '/' o ':') entre comillas simples, pero los "dateparts" y los "timeparts" enumerados en la tabla anterior tienen que estar entre comillas dobles.

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	Descripción
9	Valor numérico con la cantidad especificada de dígitos.
0	Valor numérico con ceros a la izquierda.
. (period), D	Punto decimal.
, (coma)	Separador de miles.
CC	Código de siglo. Por ejemplo, el siglo XXI comenzó el 01/01/2001 (compatible solo con TO_CHAR).

Formato	Descripción
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.

Caracteres de formato de estilo Teradata para datos numéricos

A continuación, puede descubrir cómo las funciones TEXT_TO_INT_ALT y TEXT_TO_NUMERIC_ALT interpretan los caracteres de la cadena de expresión de entrada. En la siguiente tabla también encontrará una lista de los caracteres que puede especificar en la frase de formato. Además, encontrará una descripción de las diferencias entre el formato de estilo Teradata y la opción AWS Clean Rooms de formato.

Formato	Descripción
G	No se admite como separador de grupo en la cadena de expresión de entrada. No puede especificar este carácter en la frase de formato.
D	Símbolo de punto base. Puede especificar este carácter en la frase de formato. Este carácter equivale al . (punto). El símbolo de punto base no puede aparecer en una frase de formato que contenga cualquiera de los siguientes caracteres: • . (punto) • S (“s” mayúscula) • V (“v” mayúscula)
/ , : %	Caracteres de inserción: barra diagonal (/), coma (,), dos puntos (:) y signo de porcentaje (%). No puede incluir estos caracteres en la frase de formato. AWS Clean Rooms ignora estos caracteres en la cadena de expresión de entrada.
.	Un punto como un carácter de punto base, es decir, una coma decimal. Este carácter no puede aparecer en una frase de formato que contenga cualquiera de los siguientes caracteres: • D (“d” mayúscula) • S (“s” mayúscula) • V (“v” mayúscula)

Formato	Descripción
B	No puede incluir el carácter de espacio en blanco (B) en la frase de formato. En la cadena de expresión de entrada, los espacios anteriores y posteriores no se tienen en cuenta, y los espacios entre dígitos no están permitidos.
+ -	No puede incluir el signo más (+) o el signo menos (-) en la frase de formato. No obstante, el signo más (+) y el signo menos (-) se analizan de forma implícita como parte del valor numérico si aparecen en la cadena de expresión de entrada.
V	Indicador de la posición de la coma decimal. Este carácter no puede aparecer en una frase de formato que contenga cualquiera de los siguientes caracteres: • D (“d” mayúscula) • . (punto)
Z	Dígito decimal suprimido con cero. AWS Clean Rooms recorta los ceros iniciales. El carácter Z no puede seguir a un carácter 9. El carácter Z debe estar a la izquierda del carácter de punto base si la parte de la fracción contiene el carácter 9.
9	Dígito decimal.

Formato	Descripción
CHAR(n)	Para este formato, puede especificar lo siguiente: • CHAR consta de Z o 9 caracteres. AWS Clean Rooms no admite un signo + (más) o - (menos) en el valor CHAR. • n es una constante entera, I o F. Para I, es el número de caracteres necesarios para mostrar la parte entera de los datos numéricos o enteros. Para F, es el número de caracteres necesarios para mostrar la parte de fracción de los datos numéricos.
-	Carácter de guion (-). No puede incluir este carácter en la frase de formato. AWS Clean Rooms omite este carácter de la cadena de expresión de entrada.

Formato	Descripción
S	Decimal zonificado firmado. El carácter S debe ubicarse después del último dígito decimal de la frase de formato. El último carácter de la cadena de expresión de entrada y la conversió n numérica correspondiente se mencionan en Caracteres de formato de datos para un decimal zonificado firmado, formato de datos numérico de estilo Teradata . El carácter S no puede aparecer en una frase de formato que contenga cualquiera de los siguientes caracteres: • + (signo más) • . (punto) • D (“d” mayúscula) • Z (“z” mayúscula) • F (“f” mayúscula) • E (“e” mayúscula)
E	Notación exponencial. La cadena de expresión de entrada puede incluir el carácter de exponente. No se puede especificar E como un carácter de exponente en la frase de formato.
FN9	No se admite en AWS Clean Rooms.
FNE	No se admite en AWS Clean Rooms.

Formato	Descripción
$, USD, dólares estadounidenses	Signo de dólar ($), código ISO de la moneda (USD) y nombre de la moneda dólares estadounidenses. El símbolo ISO de la divisa USD y el nombre de la divisa Dólares estadounidenses distingue n mayúsculas de minúsculas. AWS Clean Rooms solo admite la divisa USD. La entrada de expresión de entrada puede incluir espacios entre el símbolo de la moneda USD y el valor numérico; por ejemplo, “$ 123E2” o “123E2 $”.
L	Símbolo de la moneda. Este carácter de símbolo de moneda solo puede aparecer una vez en la frase de formato. No se pueden especificar caracteres de símbolo de moneda repetidos.
C	Código de la moneda ISO. Este carácter de símbolo de moneda solo puede aparecer una vez en la frase de formato. No se pueden especificar caracteres de símbolo de moneda repetidos.
N	Nombre completo de la moneda. Este carácter de símbolo de moneda solo puede aparecer una vez en la frase de formato. No se pueden especificar caracteres de símbolo de moneda repetidos.
O	Símbolo de moneda doble. No puede especific ar este carácter en la frase de formato.
U	Código de moneda ISO doble. No puede especificar este carácter en la frase de formato.

Formato	Descripción
A	Nombre completo doble de la moneda. No puede especificar este carácter en la frase de formato.

Caracteres de formato de datos para un decimal zonificado firmado, formato de datos numérico de estilo Teradata

Puede utilizar los siguientes caracteres en la frase de formato de las funciones TEXT_TO_INT_ALT y TEXT_TO_NUMERIC_ALT para un valor decimal, zonificado y firmado.

Último carácter de la cadena de entrada	Conversión numérica
{ o 0	n … 0
A o 1	n … 1
B o 2	n … 2
C o 3	n … 3
D o 4	n … 4
E o 5	n … 5
F o 6	n … 6
G o 7	n … 7
H u 8	n … 8
I o 9	n … 9
}	-n … 0
J	-n … 1
K	-n … 2

Último carácter de la cadena de entrada	Conversión numérica
L	-n … 3
M	-n … 4
N	-n … 5
O	-n … 6
P	-n … 7
Q	-n … 8
R	-n … 9

Funciones de fecha y hora

AWS Clean Rooms admite las siguientes funciones de fecha y hora:

Temas

- Función TO_TIMESTAMP
- Partes de fecha para funciones de fecha o marca temporal

Resumen de las funciones de fecha y hora

La siguiente tabla ofrece un resumen de las funciones de fecha y hora que se utilizan en AWS Clean Rooms.

Función	Sintaxis	Devuelve
+ Operador (concatenación) Concatena una fecha a una hora a cada lado del símbolo + y devuelve TIMESTAMP o TIMESTAMPTZ.	date + time	`TIMESTAMP` o `TIMESTAMPZ`
ADD_MONTHS Agrega la cantidad de meses especificada a una fecha o marca temporal.	ADD_MONTHS ({date|timestamp}, integer)	`TIMESTAMP`
Función CURRENT_DATE Devuelve una fecha en la zona horaria de la sesión actual (que es UTC de manera predeterminada) para el comienzo de la transacción actual.	CURRENT_DATE	`DATE`
DATEADD Aumenta una fecha o una hora según un intervalo especificado.	DATEADD (datepart, interval, {date|time|timetz|timestamp})	`TIMESTAMP` o `TIME` o `TIMETZ`
DATEDIFF Devuelve la diferencia entre dos fechas u horas para una parte de fecha dada, como un día o mes.	DATEDIFF (datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp})	`BIGINT`

Función	Sintaxis	Devuelve
DATE_PART Extrae el valor de la parte de una fecha a partir de una fecha u hora.	DATE_PART (datepart, {date|timestamp})	DOUBLE
DATE_TRUNC Trunca una marca temporal en función de una parte de fecha.	DATE_TRUNC ('datepart', timestamp)	TIMESTAMP
EXTRACT Extrae una parte de una fecha o una hora a partir de timestamp, timestamptz, time o timetz.	EXTRACT (parte de fecha FROM origen)	INTEGER or DOUBLE
Función GETDATE Devuelve la fecha y hora actual en la zona horaria de la sesión actual (que es UTC de manera predeterminada). Los paréntesis son obligatorios.	GETDATE()	TIMESTAMP
SYSDATE Devuelve la fecha y hora según la zona horaria UTC para el comienzo de la transacción actual.	SYSDATE	TIMESTAMP
TIMEOFDAY Devuelve el día, la fecha y la hora actuales en la zona horaria de la sesión actual (que es UTC de manera predeterminada) como un valor de cadena.	TIMEOFDAY()	VARCHAR

Función	Sintaxis	Devuelve
TO_TIMESTAMP Devuelve una marca temporal con zona horaria para el formato de marca temporal y zona horaria especificado.	TO_TIMESTAMP ('timestamp', 'format')	`TIMESTAMP TZ`

> **Note**
>
> Los segundos de salto no se consideran en los cálculos de tiempo transcurrido.

Funciones de fecha y hora en transacciones

Cuando se ejecutan las siguientes funciones dentro de un bloque de transacción (BEGIN ... END), la función devuelve la fecha u hora de inicio de la transacción actual, no de la instrucción actual.

- SYSDATE
- TIMESTAMP
- CURRENT_DATE

Las siguientes funciones siempre devuelven la fecha u hora de comienzo de la instrucción actual, incluso cuando se encuentran dentro de un bloque de transacción.

- GETDATE
- TIMEOFDAY

+ 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

literales numéricos

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

literales de cadena

Cadenas, cadenas de caracteres o constantes de caracteres

date

Una columna DATE o una expresión que, implícitamente, se convierte en un valor DATE.

time

Una columna TIME o una expresión que, implícitamente, se convierte en un valor TIME.

timetz

Una columna TIMETZ o una expresión que, implícitamente, se convierte en un valor 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;
```

Función ADD_MONTHS

ADD_MONTHS agrega la cantidad de meses especificada a una expresión o un valor de fecha o marca temporal. La función DATEADD 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 retorno

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 retorno

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
```

En el siguiente ejemplo, la marca temporal que aparece en la columna LISTTIME se convierte de la zona horaria predeterminada UTC a la 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 (que es UTC de manera predeterminada) en el formato predeterminado: AAAA-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 retorno

FECHA

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 DATEADD

Aumenta un valor de DATE, TIME, TIMETZ o TIMESTAMP según un intervalo especificado.

Sintaxis

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

Argumentos

datepart

La parte de la fecha (año, mes, día u hora, por ejemplo) sobre la que opera la función. Para obtener más información, consulte Partes de fecha para funciones de fecha o marca temporal.

interval

Un número entero que especifica el intervalo (cantidad de días, por ejemplo) por agregar a la expresión objetivo. Un número entero negativo resta al intervalo.

date|time|timetz|timestamp

Una columna DATE, TIME, TIMETZ o TIMESTAMP o una expresión que, de forma implícita, se convierte a una DATE, TIME, TIMETZ o TIMESTAMP. La expresión de DATE, TIME, TIMETZ o TIMESTAMP debe contener la parte de fecha especificada.

Tipo de retorno

TIMESTAMP, TIME o TIMETZ según el tipo de datos de entrada.

Ejemplos con una columna DATE

En el siguiente ejemplo, se agregan 30 días a cada fecha en noviembre que se encuentra en la tabla de DATE.

```
select dateadd(day,30,caldate) as novplus30
from date
where month='NOV'
order by dateid;

novplus30
---------------------
2008-12-01 00:00:00
2008-12-02 00:00:00
2008-12-03 00:00:00
```

```
...
(30 rows)
```

En el siguiente ejemplo, se agregan 18 meses a un valor de fecha literal.

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

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

El nombre predeterminado de la columna para una función DATEADD es DATE_ADD. La marca temporal predeterminada para un valor de fecha es `00:00:00`.

En el siguiente ejemplo, se agregan 30 minutos a un valor de fecha que no especifica una marca temporal.

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

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

Puede nombrar las partes de la fecha de manera completa o abreviada. En este caso, m representa a los minutos y no a lo meses.

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 agregan 5 minutos a cada TIME_VAL de la tabla TIME_TEST.

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

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

En el siguiente ejemplo, se agregan 8 horas a un valor de fecha literal.

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

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

El siguiente ejemplo muestra cuando una hora supera las 24:00:00 o cuando es antes de las 00:00:00.

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

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

Ejemplos con una columna TIMETZ

Los valores de salida en estos ejemplos están en formato UTC, que es la zona horaria predeterminada.

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 agregan 5 minutos a cada TIMETZ_VAL de la tabla TIMETZ_TEST.

```
select dateadd(minute,5,timetz_val) as minplus5_tz from timetz_test;

minplus5_tz
---------------
04:05:00+00
00:05:00.5550+00
06:03:00+00
```

En el siguiente ejemplo, se agregan 2 horas a un valor de timetz literal.

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

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

Ejemplos con una columna TIMESTAMP

Los valores de salida en estos ejemplos están en formato UTC, que es la zona horaria predeterminada.

La siguiente tabla de ejemplo, TIMESTAMP_TEST tiene una columna TIMESTAMP_VAL (tipo TIMESTAMP) con tres valores insertados.

```
SELECT timestamp_val FROM timestamp_test;

timestamp_val
------------------
1988-05-15 10:23:31
2021-03-18 17:20:41
2023-06-02 18:11:12
```

El siguiente ejemplo agrega 20 años solo a los valores TIMESTAMP_VAL en TIMESTAMP_TEST anteriores al año 2000.

```
SELECT dateadd(year,20,timestamp_val)
```

```
FROM timestamp_test
WHERE timestamp_val < to_timestamp('2000-01-01 00:00:00', 'YYYY-MM-DD HH:MI:SS');

date_add
---------------
2008-05-15 10:23:31
```

En el siguiente ejemplo se agregan 5 segundos a un valor de marca temporal literal escrito sin un indicador de segundos.

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

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

Notas de uso

Las funciones DATEADD(month, ...) y ADD_MONTHS administran las fechas que caen en fin de mes de manera diferente:

- ADD_MONTHS: si la fecha que se agrega es el último día del mes, el resultado es siempre el último día del mes en el resultado, sin importar su longitud. Por ejemplo, 30 de abril + 1 mes es 31 de mayo.

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

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

- DATEADD: si en la fecha que se agrega hay menos días que en el mes del resultado, el resultado será el día correspondiente del mes del resultado, no el último día de ese mes. Por ejemplo, 30 de abril + 1 mes es 30 de mayo.

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

  date_add
  ---------------------
  2008-05-30 00:00:00
  ```

```
(1 row)
```

La función DATEADD administra la fecha 02-29 de año bisiesto de manera diferente con dateadd (month, 12,…) o dateadd (year, 1, …).

```
select dateadd(month,12,'2016-02-29');

date_add
---------------------
2017-02-28 00:00:00

select dateadd(year, 1, '2016-02-29');

date_add
---------------------
2017-03-01 00:00:00
```

Función DATEDIFF

DATEDIFF devuelve la diferencia entre las partes de fecha de dos expresiones de fecha u hora.

Sintaxis

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

Argumentos

datepart

La parte específica del valor de fecha u hora (año, mes o día; hora, minuto, segundo, milisegundo o microsegundo) sobre la que opera la función. Para obtener más información, consulte Partes de fecha para funciones de fecha o marca temporal.

En concreto, DATEDIFF determina la cantidad de límites de la parte de fecha que se cruzan entre las dos expresiones. Suponga, por ejemplo, que calcula la diferencia en años entre dos fechas, `12-31-2008` y `01-01-2009`. En este caso, la función devuelve 1 año a pesar de que estas fechas solo tienen un día de diferencia. Si busca la diferencia en horas entre dos marcas temporales, `01-01-2009 8:30:00` y `01-01-2009 10:00:00`, el resultado es 2 horas. Si busca la diferencia en horas entre dos marcas temporales, `8:30:00` y `10:00:00`, el resultado es 2 horas.

date|time|timetz|timestamp

Una columna o expresiones de DATE, TIME, TIMETZ o TIMESTAMP que se convierten de forma implícita en DATE, TIME, TIMETZ o TIMESTAMP. Ambas expresiones deben contener la parte de fecha u hora especificada. Si la segunda fecha u hora es posterior a la primera, el resultado es positivo. Si la segunda fecha u hora es anterior a la primera, el resultado es negativo.

Tipo de retorno

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 datediff(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 datediff(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 datediff(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 datediff(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 predeterminado de la columna para la función DATEDIFF es DATE_DIFF.

```
select datediff(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,
datediff(day, listtime, saletime) as wait
from sales, listing where sales.listid = listing.listid
```

```
and sales.listid between 1000 and 1005
order by wait desc, priceperticket desc;

priceperticket | wait
---------------+------
 96.00         |   15
 123.00        |   11
 131.00        |    9
 123.00        |    6
 129.00        |    4
 96.00         |    4
 96.00         |    0
(7 rows)
```

En este ejemplo, se calculan las horas promedio esperadas por los vendedores para todas las ventas de tickets.

```
select avg(datediff(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 datediff(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 datediff(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 datediff(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 datediff(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

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

Argumentos

datepart

Un literal o cadena de identificación de la parte específica del valor de la fecha (por ejemplo, año, mes o día) en la que actúa la función. Para obtener más información, consulte Partes de fecha para funciones de fecha o marca temporal.

{date|timestamp}

Una columna de fecha, una columna de marca de tiempo o una expresión que se convierte implícitamente en una fecha o una marca de tiempo. La columna o expresión en fecha o marca de tiempo debe contener la parte de fecha especificada en parte de fecha.

Tipo de retorno

DOUBLE

Ejemplos

El nombre predeterminado de la columna para la función DATE_PART es `pgdate_part`.

En el ejemplo siguiente se encuentra el minuto de un literal de marca temporal.

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

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

En el ejemplo siguiente, se encuentra el número de semana de un literal de marca temporal. El cálculo del número de semana sigue el estándar ISO 8601. Para obtener más información, consulte ISO 8601 en Wikipedia.

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

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

En el ejemplo siguiente se encuentra el día del mes de un literal de marca temporal.

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

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

En el ejemplo siguiente se encuentra el día de la semana de un literal de marca temporal. El cálculo del número de semana sigue el estándar ISO 8601. Para obtener más información, consulte ISO 8601 en Wikipedia.

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

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

En el ejemplo siguiente se encuentra el siglo de un literal de marca temporal. El cálculo del siglo sigue el estándar ISO 8601. Para obtener más información, consulte ISO 8601 en Wikipedia.

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

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

En el ejemplo siguiente, se encuentra el milenio de un literal de marca de tiempo. El cálculo del milenio sigue el estándar ISO 8601. Para obtener más información, consulte ISO 8601 en Wikipedia.

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

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

En el ejemplo siguiente, se encuentran los microsegundos de un literal de marca de tiempo. El cálculo de los microsegundos sigue el estándar ISO 8601. Para obtener más información, consulte ISO 8601 en Wikipedia.

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

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

En el ejemplo siguiente se encuentra el mes de un literal de fecha.

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

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

En el siguiente ejemplo, se aplica la función DATE_PART a una columna de una tabla.

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


weeks |      listtime
------+---------------------
```

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

Puede especificar las partes de fecha utilizando su nombre completo o una abreviatura; en este caso, w quiere decir semanas.

La parte de fecha de día de la semana devuelve un número entero de 0 a 6, comenzando por domingo. Use DATE_PART con dow (DAYOFWEEK) para visualizar los eventos que se llevan a cabo un sábado.

```
SELECT date_part(dow, starttime) AS dow, starttime
FROM event
WHERE date_part(dow, starttime)=6
ORDER BY 2,1;

 dow |      starttime
-----+---------------------
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
...
(1147 rows)
```

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('datepart', timestamp)
```

Argumentos

datepart

La parte de fecha que trunca el valor de una marca temporal. La timestamp de entrada se trunca en la precisión de la entrada datepart. Por ejemplo, `month` se trunca en el primer día del mes. Los formatos válidos son los siguientes:

- microsegundo, microsegundos
- milisegundo, milisegundos
- segundo, segundos
- minuto, minutos
- hora, horas
- día, días
- semana, semanas
- mes, meses
- trimestre, trimestres
- año, años
- década, décadas
- siglo, siglos
- milenio, milenios

Para obtener más información sobre las abreviaturas de algunos formatos, consulte Partes de fecha para funciones de fecha o marca temporal.

timestamp

Una columna de marca temporal o una expresión que, implícitamente, se convierte en una marca temporal.

Tipo de retorno

TIMESTAMP

Ejemplos

Trunque la marca temporal de entrada en el segundo.

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

Trunque la marca temporal de entrada al minuto.

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

Trunque la marca temporal de entrada a la hora.

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

Trunque la marca temporal de entrada al día.

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

Trunque la marca temporal de entrada al primer día de un mes.

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

Trunque la marca temporal de entrada al primer día de un trimestre.

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

Trunque la marca temporal de entrada al primer día de un año.

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

Trunque la marca temporal de entrada al primer día de un siglo.

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

```
2001-01-01 00:00:00
```

Trunque la marca temporal de entrada al lunes de un mes.

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

En el siguiente ejemplo, la función DATE_TRUNC usa la parte de fecha “week” (semana) para devolver la fecha del lunes de cada semana.

```
select date_trunc('week', saletime), sum(pricepaid) from sales where
saletime like '2008-09%' group by date_trunc('week', saletime) order by 1;

date_trunc  |    sum
------------+-------------
2008-09-01  | 2474899
2008-09-08  | 2412354
2008-09-15  | 2364707
2008-09-22  | 2359351
2008-09-29  |  705249
```

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 retorno

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 TIMESTAMP

En el siguiente ejemplo, se determinan los números de las semanas para las ventas en las que el precio pagado fue 10 000 USD o más.

```
select salesid, extract(week from saletime) as weeknum
from sales
where pricepaid > 9999
order by 2;

salesid | weeknum
--------+---------
 159073 |       6
 160318 |       8
 161723 |      26
```

En el siguiente ejemplo, se devuelve el valor de minutos de un valor de marca temporal literal.

```
select extract(minute from timestamp '2009-09-09 12:08:43');

date_part
--
```

En el siguiente ejemplo, se devuelve el valor de milisegundos de un valor timestamp literal.

```
select extract(ms from timestamp '2009-09-09 12:08:43.101');

date_part
-----------
```

```
101
```

Ejemplos con TIMESTAMPTZ

En el siguiente ejemplo, se devuelve el valor de año de un valor timestamptz literal.

```
select extract(year from timestamptz '1.12.1997 07:37:16.00 PST');

date_part
-----------
1997
```

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
```

En el siguiente ejemplo, se extraen los milisegundos de un valor literal.

```
select extract(ms from time '18:25:33.123456');

 date_part
-----------
     123
```

Ejemplos con 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 extraen las horas de cada timetz_val.

```
select extract(hour from timetz_val) as hours from time_test;

hours
-----------
         4
         0
         5
```

En el siguiente ejemplo, se extraen los milisegundos de un valor literal. Los literales no se convierten a UTC antes de que se procese la extracción.

```
select extract(ms from timetz '18:25:33.123456 EST');
```

```
 date_part
-----------
     123
```

En el siguiente ejemplo, se devuelve la hora de desfase horario respecto a UTC a partir de un valor timetz literal.

```
select extract(timezone_hour from timetz '1.12.1997 07:37:16.00 PDT');

date_part
-----------
-7
```

Función GETDATE

La función GETDATE devuelve la fecha y hora actuales en la zona horaria de la sesión actual (que es UTC de manera predeterminada).

Devuelve la fecha u hora de inicio de la instrucción actual, incluso cuando se encuentra dentro de un bloque de transacciones.

Sintaxis

```
GETDATE()
```

Los paréntesis son obligatorios.

Tipo de retorno

TIMESTAMP

Ejemplo

En el siguiente ejemplo, se usa la función GETDATE para devolver la marca temporal completa para la fecha actual.

```
select getdate();
```

Función SYSDATE

SYSDATE devuelve la fecha y hora actual en la zona horaria de la sesión actual (que es UTC de manera predeterminada).

> **Note**
>
> SYSDATE devuelve la fecha y hora de comienzo de la transacción actual, no de la instrucción actual.

Sintaxis

```
SYSDATE
```

Esta función no requiere argumentos.

Tipo de retorno

TIMESTAMP

Ejemplos

En el siguiente ejemplo, se usa la función SYSDATE para devolver la marca temporal completa para la fecha actual.

```
select sysdate;

timestamp
----------------------------
2008-12-04 16:10:43.976353
(1 row)
```

En el siguiente ejemplo, se usa la función SYSDATE dentro de la función TRUNC para devolver la fecha actual sin la hora.

```
select trunc(sysdate);

trunc
------------
```

```
2008-12-04
(1 row)
```

La siguiente consulta devuelve la información de ventas correspondiente a las fechas comprendidas entre la fecha en que se emite la consulta y la fecha que caiga 120 días antes.

```
select salesid, pricepaid, trunc(saletime) as saletime, trunc(sysdate) as now
from sales
where saletime between trunc(sysdate)-120 and trunc(sysdate)
order by saletime asc;

salesid | pricepaid |  saletime  |    now
---------+-----------+------------+------------
91535 |    670.00 | 2008-08-07 | 2008-12-05
91635 |    365.00 | 2008-08-07 | 2008-12-05
91901 |   1002.00 | 2008-08-07 | 2008-12-05
...
```

Función TIMEOFDAY

TIMEOFDAY es un alias especial que se usa para devolver el día, la fecha y la hora como un valor de cadena. Devuelve la cadena de hora del día para la instrucción actual, incluso cuando se encuentra dentro de un bloque de transacciones.

Sintaxis

```
TIMEOFDAY()
```

Tipo de retorno

VARCHAR

Ejemplos

En el siguiente ejemplo, se devuelve la fecha y la hora actual mediante el uso de la función TIMEOFDAY.

```
select timeofday();
timeofday
------------
Thu Sep 19 22:53:50.333525 2013 UTC
```

```
(1 row)
```

Función TO_TIMESTAMP

TO_TIMESTAMP convierte una cadena TIMESTAMP en TIMESTAMPTZ.

Sintaxis

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

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

Argumentos

timestamp

Una cadena que representa un valor de marca temporal en el formato especificado por format. Si este argumento se deja vacío, el valor predeterminado de la marca de tiempo es `0001-01-01 00:00:00`.

formato

Un literal de cadena que define el formato del valor de timestamp. Los formatos que incluyen una zona horaria (**`TZ`**, **`tz`** u **`OF`**) no son compatibles como entrada. Para obtener los formatos de marca temporal válidos, consulte Cadenas de formatos de fecha y hora.

is_strict

Un valor booleano opcional que especifica si se devuelve un error si un valor de marca temporal de entrada se encuentra fuera de rango. Cuando is_strict se configura como TRUE, se devuelve un error si hay un valor fuera de rango. Si is_strict se configura como FALSE, que es el valor predeterminado, se aceptan valores de desbordamiento.

Tipo de retorno

TIMESTAMPTZ

Ejemplos

En el siguiente ejemplo, se muestra cómo se usa la función TO_TIMESTAMP para convertir una cadena TIMESTAMP en TIMESTAMPTZ.

```
select sysdate, to_timestamp(sysdate, 'YYYY-MM-DD HH24:MI:SS') as second;

timestamp                  | second
---------------------------   ---------------------
2021-04-05 19:27:53.281812 | 2021-04-05 19:27:53+00
```

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 un valor TIMESTAMPTZ. El resultado es un valor TIMESTAMPTZ que se establece 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
```

La siguiente instrucción SQL convierte la cadena “2011-12-18 24:38:15” en un valor TIMESTAMPTZ. El resultado es un error porque el valor de tiempo en la marca de tiempo es superior a 24 horas:

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS', TRUE);

ERROR:  date/time field time value out of range: 24:38:15.0
```

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:

- DATEADD
- DATEDIFF

- 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) 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)

Parte de la fecha o parte de la hora	Abreviaturas
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 hash

Una función hash es una función matemática que convierte un valor de entrada numérico en otro valor. AWS Clean Rooms admite las siguientes funciones hash:

Temas

- Función MD5
- Función SHA
- Función SHA1
- Función SHA2
- MURMUR3_32_HASH

Función MD5

Utiliza la función hash criptográfica MD5 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 comprobación de 128 bits.

Sintaxis

```
MD5(string)
```

Argumentos

string

Una cadena de longitud variable.

Tipo de retorno

La función MD5 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 la función SHA1.

Consulte Función SHA1.

Función SHA1

La función SHA1 usa la función criptográfica hash SHA1 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 comprobación de 160 bits.

Sintaxis

SHA1 es sinónimo de Función SHA.

```
SHA1(string)
```

Argumentos

string

Una cadena de longitud variable.

Tipo de retorno

La función SHA1 devuelve una cadena de 40 caracteres que es una representación textual del valor hexadecimal de una suma de comprobación 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');
```

Función SHA2

La función SHA2 utiliza la función hash criptográfica SHA2 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 función SHA2 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);
```

MURMUR3_32_HASH

La función MURMUR3_32_HASH calcula el hash no criptográfico Murmur3A de 32 bits para todos los tipos de datos comunes, incluidos los tipos numéricos y de cadena.

Sintaxis

```
MURMUR3_32_HASH(value [, seed])
```

Argumentos

value

El valor de entrada al que se aplica el hash. AWS Clean Rooms realiza el hash de la representación binaria del valor de entrada. Este comportamiento es similar al de FNV_HASH, pero el valor se convierte a la representación binaria indicada por la especificación hash Murmur3 de 32 bits de Apache Iceberg.

valor de inicialización

El valor de inicialización INT de la función hash. Este argumento es opcional. Si no se da, AWS Clean Rooms utiliza el valor de inicio predeterminado 0. Esto permite combinar el hash de varias columnas sin conversiones ni concatenaciones.

Tipo de retorno

La función devuelve un valor INT.

Ejemplo

En los siguientes ejemplos se devuelve el hash Murmur3 de un número, la cadena "AWS Clean Rooms" y la concatenación de los dos.

```
select MURMUR3_32_HASH(1);

    MURMUR3_32_HASH
----------------------
 -5968735742475085980
(1 row)
```

```
select MURMUR3_32_HASH('AWS Clean Rooms');

    MURMUR3_32_HASH
----------------------
 7783490368944507294
(1 row)
```

```
select MURMUR3_32_HASH('AWS Clean Rooms', MURMUR3_32_HASH(1));

    MURMUR3_32_HASH
----------------------
 -2202602717770968555
(1 row)
```

Notas de uso

Para calcular el hash de una tabla con varias columnas, puede calcular el hash Murmur3 de la primera columna y pasarlo como valor de inicialización al hash de la segunda columna. A

continuación, pasa el hash Murmur3 de la segunda columna como valor de inicialización al hash de la tercera columna.

En el siguiente ejemplo, se crean valores de inicialización para aplicar un algoritmo hash a una tabla con varias columnas.

```
select MURMUR3_32_HASH(column_3, MURMUR3_32_HASH(column_2, MURMUR3_32_HASH(column_1)))
 from sample_table;
```

La misma propiedad se puede utilizar para calcular el hash de una concatenación de cadenas.

```
select MURMUR3_32_HASH('abcd');

   MURMUR3_32_HASH
---------------------
 -281581062704388899
(1 row)
```

```
select MURMUR3_32_HASH('cd', MURMUR3_32_HASH('ab'));

   MURMUR3_32_HASH
---------------------
 -281581062704388899
(1 row)
```

La función hash utiliza el tipo de entrada para determinar el número de bytes para aplicar un algoritmo hash. Utilice la conversión para aplicar un tipo específico, si es necesario.

En los ejemplos siguientes se utilizan diferentes tipos de entrada para producir resultados diferentes.

```
select MURMUR3_32_HASH(1::smallint);

   MURMUR3_32_HASH
--------------------
 589727492704079044
(1 row)
```

```
select MURMUR3_32_HASH(1);

   MURMUR3_32_HASH
```

```
----------------------
 -5968735742475085980
(1 row)
```

```
select MURMUR3_32_HASH(1::bigint);

   MURMUR3_32_HASH
----------------------
 -8517097267634966620
(1 row)
```

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
  - Null
- 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 `{ }, [ ]`).

Las funciones JSON de AWS Clean Rooms y el comando COPY de AWS Clean Rooms utilizan los mismos métodos para trabajar con datos con formato JSON.

Temas

- Función CAN_JSON_PARSE
- Función JSON_EXTRACT_ARRAY_ELEMENT_TEXT
- Función JSON_EXTRACT_PATH_TEXT
- Función JSON_PARSE
- Función JSON_SERIALIZE
- Función JSON_SERIALIZE_TO_VARBYTE

Función CAN_JSON_PARSE

La función CAN_JSON_PARSE analiza los datos en formato JSON y devuelve `true` si el resultado se puede convertir en un valor SUPER mediante la función JSON_PARSE.

Sintaxis

```
CAN_JSON_PARSE(json_string)
```

Argumentos

json_string

Una expresión que devuelve objetos JSON serializados en el formato VARBYTE o VARCHAR.

Tipo de retorno

BOOLEAN

Ejemplo

Para ver si la matriz JSON `[10001,10002,"abc"]` se puede convertir en el tipo de datos SUPER, utilice el siguiente ejemplo.

```
SELECT CAN_JSON_PARSE('[10001,10002,"abc"]');

+----------------+
| can_json_parse |
+----------------+
| true           |
+----------------+
```

Función JSON_EXTRACT_ARRAY_ELEMENT_TEXT

La función JSON_EXTRACT_ARRAY_ELEMENT_TEXT devuelve un elemento de la matriz JSON en la matriz extrema de una cadena JSON utilizando un índice basado en cero. El primer elemento en una matriz está en posición 0. Si el índice es negativo o está fuera de límite, JSON_EXTRACT_ARRAY_ELEMENT_TEXT devuelve una cadena vacía. Si el argumento null_if_invalid está establecido en `true` y la cadena JSON no es válida, la función devuelve NULL en lugar de un error.

Para obtener más información, consulte Funciones JSON.

Sintaxis

```
json_extract_array_element_text('json string', pos [, null_if_invalid ] )
```

Argumentos

json_string

Una cadena JSON con formato adecuado.

pos

Un valor entero que representa el índice del elemento de matriz que se devolverá, utilizando un índice de matriz basado en cero.

null_if_invalid

Un valor booleano que especifica que se devuelva NULL si la cadena JSON de entrada no es válida en lugar de devolver un error. Para devolver NULL si la cadena JSON no es válida, especifique `true` (`t`). Para devolver un error si la cadena JSON no es válida, especifique `false` (`f`). El valor predeterminado es `false`.

Tipo de retorno

Cadena VARCHAR que representa el elemento de matriz JSON al que se hace referencia en pos.

Ejemplo

El siguiente ejemplo devuelve el elemento de matriz en la posición 2, que es el tercer elemento de un índice de matriz basado en cero:

```
select json_extract_array_element_text('[111,112,113]', 2);

json_extract_array_element_text
-------------------------------
113
```

El siguiente ejemplo devuelve un error porque la cadena JSON no es válida.

```
select json_extract_array_element_text('["a",["b",1,["c",2,3,null,]]]',1);

An error occurred when executing the SQL command:
select json_extract_array_element_text('["a",["b",1,["c",2,3,null,]]]',1)
```

El siguiente ejemplo establece null_if_invalid en true, por lo que la instrucción devuelve NULL en lugar de devolver un error para indicar que la cadena JSON no es válida.

```
select json_extract_array_element_text('["a",["b",1,["c",2,3,null,]]]',1,true);

json_extract_array_element_text
-------------------------------
```

Función JSON_EXTRACT_PATH_TEXT

La función JSON_EXTRACT_PATH_TEXT devuelve el valor del par clave:valor al que se referencia en una serie de elementos de ruta de una cadena JSON. La ruta JSON se puede anidar hasta un máximo de cinco niveles de profundidad. Los elementos de ruta distinguen entre mayúsculas y minúsculas. Si un elemento de ruta no existe en la cadena JSON, JSON_EXTRACT_PATH_TEXT devuelve una cadena vacía. Si el argumento `null_if_invalid` está establecido en `true` y la cadena JSON no es válida, la función devuelve NULL en lugar de un error.

Para obtener información sobre funciones JSON adicionales, consulte Funciones JSON.

Sintaxis

```
json_extract_path_text('json_string', 'path_elem' [,'path_elem'[, …] ]
 [, null_if_invalid ] )
```

Argumentos

json_string

Una cadena JSON con formato adecuado.

path_elem

Un elemento de ruta en una cadena JSON. Se necesita al menos un elemento de ruta. Se pueden especificar elementos de ruta adicionales, hasta un máximo de cinco niveles de profundidad.

null_if_invalid

Un valor booleano que especifica que se devuelva NULL si la cadena JSON de entrada no es válida en lugar de devolver un error. Para devolver NULL si la cadena JSON no es válida, especifique `true` (`t`). Para devolver un error si la cadena JSON no es válida, especifique `false` (`f`). El valor predeterminado es `false`.

En una cadena JSON, AWS Clean Rooms reconoce `\n` como un carácter de línea nueva y `\t` como un tabulador. Para cargar una barra inversa, aplique escape con una barra inversa (`\\`).

Tipo de retorno

Una cadena VARCHAR que representa el valor JSON al que se hace referencia en los elementos de ruta.

Ejemplo

El siguiente ejemplo devuelve el valor para la ruta `'f4', 'f6'`.

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}','f4', 'f6');


json_extract_path_text
----------------------
star
```

El siguiente ejemplo devuelve un error porque la cadena JSON no es válida.

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4', 'f6');

An error occurred when executing the SQL command:
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4', 'f6')
```

En el siguiente ejemplo, null_if_invalid se establece en true, por lo que la instrucción devuelve NULL en lugar de devolver un error para indicar que la cadena JSON no es válida.

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4',
 'f6',true);

json_extract_path_text
------------------------------
```

```
NULL
```

En el siguiente ejemplo, se devuelve el valor para la ruta `'farm', 'barn', 'color'`, donde el valor recuperado se encuentra en el tercer nivel. Este ejemplo está formateado con una herramienta de lint JSON para facilitar su lectura.

```
select json_extract_path_text('{
    "farm": {
        "barn": {
            "color": "red",
            "feed stocked": true
        }
    }
}', 'farm', 'barn', 'color');

json_extract_path_text
----------------------
red
```

El siguiente ejemplo devuelve NULL porque falta el elemento `'color'`. Este ejemplo está formateado con una herramienta de lint JSON.

```
select json_extract_path_text('{
    "farm": {
        "barn": {}
    }
}', 'farm', 'barn', 'color');

json_extract_path_text
----------------------
NULL
```

Si el JSON es válido, el intento de extracción de un elemento que falta devuelve NULL.

El siguiente ejemplo devuelve el valor para la ruta `'house', 'appliances', 'washing machine', 'brand'`.

```
select json_extract_path_text('{
  "house": {
    "address": {
      "street": "123 Any St.",
      "city": "Any Town",
```

```
      "state": "FL",
      "zip": "32830"
    },
    "bathroom": {
      "color": "green",
      "shower": true
    },
    "appliances": {
      "washing machine": {
        "brand": "Any Brand",
        "color": "beige"
      },
      "dryer": {
        "brand": "Any Brand",
        "color": "white"
      }
    }
  }
}', 'house', 'appliances', 'washing machine', 'brand');

json_extract_path_text
----------------------
Any Brand
```

Función JSON_PARSE

La función JSON_PARSE analiza los datos con formato JSON y los convierte en la representación SUPER.

Para capturar el tipo de datos SUPER mediante el comando INSERT o UPDATE, utilice la función JSON_PARSE. Cuando utiliza JSON_PARSE() para analizar cadenas JSON en valores SUPER, se aplican determinadas restricciones.

Sintaxis

```
JSON_PARSE(json_string)
```

Argumentos

json_string

Una expresión que devuelve JSON serializado como tipo varbyte o varchar.

Tipo de retorno

SUPER

Ejemplo

El siguiente es un ejemplo de la función JSON_PARSE.

```
SELECT JSON_PARSE('[10001,10002,"abc"]');
     json_parse
-------------------------
 [10001,10002,"abc"]
(1 row)
```

```
SELECT JSON_TYPEOF(JSON_PARSE('[10001,10002,"abc"]'));
 json_typeof
----------------
 array
(1 row)
```

Función JSON_SERIALIZE

La función JSON_SERIALIZE serializa una expresión SUPER en una representación JSON textual en función de RFC 8259. Para obtener más información acerca de dicho RFC, consulte Formato de intercambio de datos de notación de objetos de JavaScript (JSON).

El límite de tamaño de SUPER es aproximadamente el mismo que el límite de bloque, y el límite de VARCHAR es menor que el límite de tamaño de SUPER. Por lo tanto, la función JSON_SERIALIZE devuelve un error cuando el formato JSON excede el límite de VARCHAR del sistema.

Sintaxis

```
JSON_SERIALIZE(super_expression)
```

Argumentos

super_expression

Una superexpresión o columna.

Tipo de retorno

varchar

Ejemplo

En el siguiente ejemplo, se serializa un valor SUPER en una cadena.

```
 SELECT JSON_SERIALIZE(JSON_PARSE('[10001,10002,"abc"]'));
   json_serialize
--------------------
 [10001,10002,"abc"]
(1 row)
```

Función JSON_SERIALIZE_TO_VARBYTE

La función JSON_SERIALIZE_TO_VARBYTE convierte un valor SUPER en una cadena JSON similar a JSON_SERIALIZE(), pero se almacena en un valor VARBYTE en su lugar.

Sintaxis

```
JSON_SERIALIZE_TO_VARBYTE(super_expression)
```

Argumentos

super_expression

Una superexpresión o columna.

Tipo de retorno

varbyte

Ejemplo

En el siguiente ejemplo, se serializa un valor SUPER y devuelve el resultado en formato VARBYTE.

```
SELECT JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'));
```

```
       json_serialize_to_varbyte
```

```
----------------------------------------
 5b31303030312c31303030322c22616263225d
```

En el siguiente ejemplo, se serializa un valor SUPER y se convierte el resultado en formato VARCHAR.

```
SELECT JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'))::VARCHAR;
```

```
  json_serialize_to_varbyte
---------------------------
 [10001,10002,"abc"]
```

Funciones matemáticas

En esta sección, se describen los operadores y las funciones matemáticos compatibles con AWS Clean Rooms.

Temas

- Función LN
- Función LOG
- Función MOD
- Función PI
- Función POWER
- Función RADIANS
- 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	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
|/	raíz cuadrada	| / 25,0	5

Operador	Descripción	Ejemplo	Resultado
||/	raíz cúbica	|| / 27,0	3
@	valor absoluto	@ -5,0	5

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 el tipo SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT4 o FLOAT8.

Tipo de retorno

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 retorno

`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 retorno

`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 retorno

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

Función ATAN2

ATAN2 es una función trigonométrica que devuelve la arcotangente 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 retorno

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 retorno

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 el tipo SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT4 o FLOAT8.

Tipo de retorno

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 retorno

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 retorno

`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 retorno

`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 DEXP

La función DEXP devuelve el valor exponential en notación científica para un número de doble precisión. La única diferencia entre las funciones DEXP y EXP es que el parámetro para DEXP debe ser de `DOUBLE PRECISION`.

Sintaxis

```
DEXP(number)
```

Argumento

número

El parámetro de entrada es un número de DOUBLE PRECISION.

Tipo de retorno

DOUBLE PRECISION

Ejemplo

```
SELECT (SELECT SUM(qtysold)
FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * DEXP((7::FLOAT/100)*10) qty2010;

+-------------------+
|      qty2010      |
+-------------------+
| 695447.4837722216 |
+-------------------+
```

Función DLOG1

La función DLOG1 devuelve el logaritmo natural del parámetro de entrada.

La función DLOG1 es equivalente a Función LN.

Función DLOG10

DLOG10 devuelve el logaritmo de base 10 del parámetro de entrada.

La función DLOG10 es equivalente a Función LOG.

Sintaxis

```
DLOG10(number)
```

Argumento

número

El parámetro de entrada es un número de doble precisión.

Tipo de retorno

La función DLOG10 devuelve un número de doble precisión.

Ejemplo

El siguiente ejemplo devuelve el logaritmo de base 10 del número 100:

```
select dlog10(100);

dlog10
--------
2
(1 row)
```

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 retorno

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 el tipo SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT4 o FLOAT8.

Tipo de retorno

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.

La función LN es sinónimo de Función DLOG1.

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 AWS Clean Rooms sistema 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.

- BOOLEAN
- CHAR
- FECHA
- 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 retorno

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 base 10 de un número.

Sinónimo de Función DLOG10.

Sintaxis

```
LOG(number)
```

Argumento

número

El parámetro de entrada es un número de doble precisión.

Tipo de retorno

La función LOG devuelve un número de doble precisión.

Ejemplo

El siguiente ejemplo devuelve el logaritmo de base 10 del número 100:

```
select log(100);
dlog10
--------
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 retorno

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 retorno

`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

```
{POW | 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 retorno

`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 retorno

`DOUBLE PRECISION`

Ejemplo

Para devolver el equivalente en radianes de 180 grados, use el siguiente ejemplo.

```
SELECT RADIANS(180);

+-------------------+
|      radians      |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Función RANDOM

La función RANDOM genera un valor aleatorio entre 0,0 (inclusive) y 1,0 (exclusive).

Sintaxis

```
RANDOM()
```

Tipo de retorno

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 del tipo DECIMAL o FLOAT8. 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 retorno

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 tipo Decimal o FLOAT8. AWS Clean Rooms puede convertir otros tipos de datos según las reglas de conversión implícitas.

Tipo de retorno

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 retorno

`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 retorno

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 del tipo DECIMAL o FLOAT8. 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 retorno

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 de cadena

Temas

- || Operador (concatenación)
- Función BTRIM
- Función CHAR_LENGTH
- Función CHARACTER_LENGTH
- Función CHARINDEX
- Función CONCAT
- Funciones LEFT y RIGHT
- Función LEN
- 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 REPLICATE
- Función REVERSE
- Función RTRIM
- Función SOUNDEX
- Función SPLIT_PART
- Función STRPOS
- Función SUBSTR
- Función SUBSTRING
- Función TEXTLEN
- Función TRANSLATE
- Función TRIM
- Función UPPER

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.

|| 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 retorno

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 retorno

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 CHAR_LENGTH

Sinónimo de la función LEN.

Consulte Función LEN.

Función CHARACTER_LENGTH

Sinónimo de la función LEN.

Consulte Función LEN.

Función CHARINDEX

Devuelve la ubicación de la subcadena especificada dentro de una cadena.

Consulte Función POSITION y Función STRPOS para ver funciones similares.

Sintaxis

```
CHARINDEX( substring, string )
```

Argumentos

subcadena

Subcadena que se va a buscar dentro de la cadena.

string

La cadena o columna que se buscará.

Tipo de retorno

La función CHARINDEX 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

CHARINDEX devuelve 0 si no se encuentra una subcadena dentro de la `string`:

```
select charindex('dog', 'fish');

charindex
----------
0
(1 row)
```

Ejemplos

En el siguiente ejemplo, se muestra la posición de la cadena `fish` dentro de la palabra `dogfish`:

```
select charindex('fish', 'dogfish');
 charindex
----------
         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 charindex('.', commission), count (charindex('.', commission))
from sales where charindex('.', commission) > 4 group by charindex('.', commission)
order by 1,2;

charindex | count
----------+-------
 5        |    629
(1 row)
```

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 retorno

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)
```

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 retorno

LEFT y RIGHT devuelven una cadena VARCHAR.

Ejemplo

El siguiente ejemplo devuelve los 5 caracteres más a la izquierda y los 5 más a la derecha de nombres de eventos que tienen ID 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 LEN

Devuelve la longitud de la cadena especificada como el número de caracteres.

Sintaxis

LEN es sinónimo de Función LENGTH, Función CHAR_LENGTH, Función CHARACTER_LENGTH y Función TEXTLEN.

```
LEN(expression)
```

Argumento

expresión

El parámetro de entrada de datos es un CHAR, VARCHAR o un alias de uno de los tipos de entrada válidos.

Tipo de retorno

La función LEN devuelve un valor entero que indica la cantidad de caracteres en la cadena de entrada.

Si la cadena de entrada es una cadena de caracteres, la función LEN devuelve una cantidad real de caracteres en cadenas multibyte y no la cantidad de bytes. Por ejemplo, una columna VARCHAR (12) necesita almacenar tres caracteres chinos de cuatro bytes. La función LEN devolverá 3 para esa misma cadena.

Notas de uso

Los cálculos de longitud no cuentan los espacios a la derecha para cadenas de caracteres de longitud fija, pero sí los cuenta para las cadenas de longitud variable.

Ejemplo

El siguiente ejemplo devuelve el número de bytes y el número de caracteres de la cadena `français`.

```
select octet_length('français'),
len('français');

octet_length  | len
--------------+-----
           9  |   8
```

El siguiente ejemplo devuelve la cantidad de caracteres en las cadenas `cat` sin espacios a la derecha y `cat` con tres espacios a la derecha:

```
select len('cat'), len('cat   ');
 len | len
```

```
-----+-----
   3 |   6
```

El siguiente ejemplo devuelve las diez entradas más largas VENUENAME en la tabla VENUE:

```
select venuename, len(venuename)
from venue
order by 2 desc, 1
limit 10;

venuename                             | len
--------------------------------------+-----
Saratoga Springs Performing Arts Center |  39
Lincoln Center for the Performing Arts  |  38
Nassau Veterans Memorial Coliseum       |  33
Jacksonville Municipal Stadium          |  30
Rangers BallPark in Arlington           |  29
University of Phoenix Stadium           |  29
Circle in the Square Theatre            |  28
Hubert H. Humphrey Metrodome            |  28
Oriole Park at Camden Yards             |  27
Dick's Sporting Goods Park              |  26
```

Función LENGTH

Sinónimo de la función LEN.

Consulte Función LEN.

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 retorno

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 retorno

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 finaliza 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 retorno

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.

Consulte Función CHARINDEX y Función STRPOS para ver funciones similares.

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 retorno

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 retorno

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 retorno

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 retorno

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 retorno

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.

Sinónimo de Función REPLICATE.

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 retorno

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 retorno

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 REPLICATE

Sinónimo para la función REPEAT.

Consulte Función REPEAT.

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 retorno

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 ID de ventas y sus correspondientes ID invertidos vinculados convertidos a 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 finaliza cuando no aparece ningún carácter de recorte 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 retorno

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 SOUNDEX

La función SOUNDEX devuelve el valor American Soundex que consiste en la primera letra seguida de una codificación de 3 dígitos de los sonidos que representan la pronunciación en inglés de la cadena especificada.

Sintaxis

```
SOUNDEX(string)
```

Argumentos

string

Especifica una cadena CHAR o VARCHAR que desea convertir en un valor de código American Soundex.

Tipo de retorno

La función SOUNDEX devuelve una cadena VARCHAR(4) que consiste en una letra en mayúsculas seguida de una codificación de tres dígitos de los sonidos que representan la pronunciación en inglés.

Notas de uso

La función SOUNDEX solo convierte caracteres ASCII alfabéticos en minúsculas y mayúsculas en inglés, incluidas las letras a-z y A-Z. SOUNDEX omite otros caracteres. SOUNDEX devuelve un único valor Soundex para una cadena de varias palabras separadas por espacios.

```
select soundex('AWS Amazon');
```

```
 soundex
---------
```

```
 A252
```

SOUNDEX devuelve una cadena vacía si la cadena de entrada no contiene ninguna letra en inglés.

```
select soundex('+-*/%');
```

```
  soundex
---------

```

Ejemplo

El siguiente ejemplo devuelve el valor Soundex A525 para la palabra Amazon.

```
select soundex('Amazon');
```

```
 soundex
---------
 A525
```

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 retorno

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 STRPOS

Devuelve la posición de una subcadena dentro de una cadena especificada.

Consulte Función CHARINDEX y Función POSITION para ver funciones similares.

Sintaxis

```
STRPOS(string, substring )
```

Argumentos

string

El primer parámetro de entrada es la cadena que se buscará.

subcadena

El segundo parámetro es la subcadena que se va a buscar dentro de string (cadena).

Tipo de retorno

La función STRPOS 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

STRPOS devuelve 0 si no se encuentra substring (subcadena) dentro de la string (cadena):

```
select strpos('dogfish', 'fist');
strpos
--------
0
(1 row)
```

Ejemplos

En el siguiente ejemplo, se muestra la posición de la cadena `fish` dentro de la palabra `dogfish`:

```
select strpos('dogfish', 'fish');
strpos
--------
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 strpos(commission, '.'),
count (strpos(commission, '.'))
from sales
where strpos(commission, '.') > 4
group by strpos(commission, '.')
order by 1, 2;

strpos | count
-------+-------
 5     |    629
(1 row)
```

Función SUBSTR

Sinónimo de la función SUBSTRING.

Consulte Función SUBSTRING.

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(character_string FROM start_position [ FOR number_characters ] )
```

```
SUBSTRING(character_string, start_position, number_characters )
```

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

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

Argumentos

character_string

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.

number_characters

La cantidad de caracteres para extraer (la longitud de la subcadena). El valor de number_characters (número_de_caracteres) se basa en la cantidad de caracteres, no bytes, por lo que los caracteres multibyte se cuentan como caracteres simples. 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.

number_bytes

La cantidad de bytes para extraer, es decir, la longitud de la subcadena. Este número no puede ser negativo.

Tipo de retorno

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 start_position + number_characters supera la longitud de la cadena, SUBSTRING devuelve una subcadena que comienza en start_position y llega 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` + `number_characters` -1. Por ejemplo:

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

Si `start_position` + `number_characters` -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 TEXTLEN

Sinónimo de la función LEN.

Consulte Función LEN.

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 retorno

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 retorno

La función TRIM devuelve una cadena VARCHAR o CHAR. Si utiliza la función TRIM con un comando SQL, convierte AWS Clean Rooms implícitamente los resultados a VARCHAR. 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 las funciones Función CAST y Función CONVERT para obtener información acerca de 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 retorno

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)
```

Funciones de información acerca del tipo SUPER

En esta sección se describen las funciones de información para SQL que permiten obtener información dinámica de las entradas con el tipo de datos SUPER admitido en AWS Clean Rooms.

Temas

- Función DECIMAL_PRECISION

- Función DECIMAL_SCALE
- Función IS_ARRAY
- Función IS_BIGINT
- Función IS_CHAR
- Función IS_DECIMAL
- Función IS_FLOAT
- Función IS_INTEGER
- Función IS_OBJECT
- Función IS_SCALAR
- Función IS_SMALLINT
- Función IS_VARCHAR
- Función JSON_TYPEOF

Función DECIMAL_PRECISION

Verifica la precisión del número total máximo de dígitos decimales que se almacenarán. Este número incluye los dígitos a la izquierda y a la derecha de la coma decimal. El rango de precisión es de 1 a 38, con un valor predeterminado de 38.

Sintaxis

```
DECIMAL_PRECISION(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

INTEGER

Ejemplo

Para aplicar la función DECIMAL_PRECISION a la tabla t, use el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_PRECISION(s) FROM t;

+-------------------+
| decimal_precision |
+-------------------+
|                 6 |
+-------------------+
```

Función DECIMAL_SCALE

Verifica el número de dígitos decimales que se almacenarán a la derecha de la coma decimal. El rango de la escala va de 0 al punto de precisión, con un valor predeterminado de 0.

Sintaxis

```
DECIMAL_SCALE(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

INTEGER

Ejemplo

Para aplicar la función DECIMAL_SCALE a la tabla t, use el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);
```

```
SELECT DECIMAL_SCALE(s) FROM t;

+---------------+
| decimal_scale |
+---------------+
|             5 |
+---------------+
```

Función IS_ARRAY

Verifica si una variable está en una matriz. La función devuelve `true` si la variable es una matriz. La función también incluye matrices vacías. De lo contrario, la función devuelve `false` para todos los demás valores, incluido el valor nulo.

Sintaxis

```
IS_ARRAY(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

BOOLEAN

Ejemplo

Para comprobar si `[1,2]` es una matriz que utiliza la función IS_ARRAY, utilice el siguiente ejemplo.

```
SELECT IS_ARRAY(JSON_PARSE('[1,2]'));

+----------+
| is_array |
+----------+
| true     |
+----------+
```

Función IS_BIGINT

Verifica si un valor es `BIGINT`. La función IS_BIGINT devuelve `true` para los números de la escala 0 en el rango de 64 bits. De lo contrario, la función devuelve `false` para todos los demás valores, incluidos el valor nulo y los números con coma flotante.

La función IS_BIGINT es un superconjunto de IS_INTEGER.

Sintaxis

```
IS_BIGINT(super_expression)
```

Argumentos

super_expression

Una expresión o columna `SUPER`.

Tipo de retorno

`BOOLEAN`

Ejemplo

Para comprobar si `5` es `BIGINT` con la función IS_BIGINT, utilice el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_BIGINT(s) FROM t;

+---+-----------+
| s | is_bigint |
+---+-----------+
| 5 | true      |
+---+-----------+
```

Función IS_CHAR

Verifica si un valor es CHAR. La función IS_CHAR devuelve `true` para las cadenas que solo tienen caracteres ASCII, ya que el tipo CHAR solo puede almacenar caracteres con formato ASCII. La función devuelve `false` para todos los demás valores.

Sintaxis

```
IS_CHAR(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

BOOLEAN

Ejemplo

Para comprobar si `t` es CHAR con la función IS_CHAR, utilice el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('t');

SELECT s, IS_CHAR(s) FROM t;

+-----+---------+
|  s  | is_char |
+-----+---------+
| "t" | true    |
+-----+---------+
```

Función IS_DECIMAL

Verifica si un valor es DECIMAL. La función IS_DECIMAL devuelve `true` para los números que no sean de coma flotante. La función devuelve `false` para todos los demás valores, incluido el valor nulo.

La función IS_DECIMAL es un superconjunto de IS_BIGINT.

Sintaxis

```
IS_DECIMAL(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

BOOLEAN

Ejemplo

Para comprobar si 1.22 es DECIMAL con la función IS_DECIMAL, utilice el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (1.22);

SELECT s, IS_DECIMAL(s) FROM t;

+------+------------+
|  s   | is_decimal |
+------+------------+
| 1.22 | true       |
+------+------------+
```

Función IS_FLOAT

Verifica si un valor es un número de coma flotante. La función IS_FLOAT devuelve `true` para los números de coma flotante (FLOAT4 y FLOAT8). La función devuelve `false` para todos los demás valores.

El conjunto de IS_DECIMAL y el conjunto de IS_FLOAT son discontinuos.

Sintaxis

```
IS_FLOAT(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

BOOLEAN

Ejemplo

Para comprobar si 2.22::FLOAT es FLOAT con la función IS_FLOAT, utilice el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES(2.22::FLOAT);

SELECT s, IS_FLOAT(s) FROM t;

+---------+----------+
|    s    | is_float |
+---------+----------+
| 2.22e+0 | true     |
+---------+----------+
```

Función IS_INTEGER

Devuelve `true` para los números de la escala 0 en el rango de 32 bits y `false` para todo lo demás (incluidos los valores nulos y los números de coma flotante).

La función IS_INTEGER es un superconjunto de la función IS_SMALLINT.

Sintaxis

```
IS_INTEGER(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

BOOLEAN

Ejemplo

Para comprobar si 5 es INTEGER con la función IS_INTEGER, utilice el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_INTEGER(s) FROM t;

+---+------------+
| s | is_integer |
+---+------------+
| 5 | true       |
+---+------------+
```

Función IS_OBJECT

Verifica si una variable es un objeto. La función IS_OBJECT devuelve `true` para los objetos, incluidos los objetos vacíos. La función devuelve `false` para todos los demás valores, incluido el valor nulo.

Sintaxis

```
IS_OBJECT(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

BOOLEAN

Ejemplo

Para comprobar si {"name": "Joe"} es un objeto con la función IS_OBJECT, utilice el siguiente ejemplo.

```
CREATE TABLE t(s super);

INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));

SELECT s, IS_OBJECT(s) FROM t;

+----------------+-----------+
|       s        | is_object |
+----------------+-----------+
| {"name":"Joe"} | true      |
+----------------+-----------+
```

Función IS_SCALAR

Verifica si una variable es escalar. La función IS_SCALAR devuelve `true` para cualquier valor que no sea una matriz o un objeto. La función devuelve `false` para todos los demás valores, incluido el valor nulo.

El conjunto de IS_ARRAY, IS_OBJECT e IS_SCALAR cubre todos los valores, excepto los nulos.

Sintaxis

```
IS_SCALAR(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

BOOLEAN

Ejemplo

Para comprobar si {"name": "Joe"} es escalar con la función IS_SCALAR, utilice el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));

SELECT s, IS_SCALAR(s.name) FROM t;

+----------------+-----------+
|       s        | is_scalar |
+----------------+-----------+
| {"name":"Joe"} | true      |
+----------------+-----------+
```

Función IS_SMALLINT

Verifica si una variable es SMALLINT. La función IS_SMALLINT devuelve `true` para los números de la escala 0 en el rango de 16 bits. La función devuelve `false` para todos los demás valores, incluidos los valores nulos y los números con coma flotante.

Sintaxis

```
IS_SMALLINT(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Return

BOOLEAN

Ejemplo

Para comprobar si 5 es SMALLINT con la función IS_SMALLINT, utilice el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_SMALLINT(s) FROM t;

+---+-------------+
| s | is_smallint |
+---+-------------+
| 5 | true        |
+---+-------------+
```

Función IS_VARCHAR

Verifica si una variable es VARCHAR. La función IS_VARCHAR devuelve `true` para todas las cadenas. La función devuelve `false` para todos los demás valores.

La función IS_VARCHAR es un superconjunto de la función IS_CHAR.

Sintaxis

```
IS_VARCHAR(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

BOOLEAN

Ejemplo

Para comprobar si abc es VARCHAR con la función IS_VARCHAR, utilice el siguiente ejemplo.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('abc');

SELECT s, IS_VARCHAR(s) FROM t;

+-------+------------+
|   s   | is_varchar |
+-------+------------+
| "abc" | true       |
+-------+------------+
```

Función JSON_TYPEOF

La función escalar JSON_TYPEOF devuelve un VARCHAR con valores booleanos, números, cadenas, objetos, matrices o nulos, en función del tipo dinámico del valor SUPER.

Sintaxis

```
JSON_TYPEOF(super_expression)
```

Argumentos

super_expression

Una expresión o columna SUPER.

Tipo de retorno

VARCHAR

Ejemplo

Para comprobar el tipo de JSON de la matriz [1,2] con la función JSON_TYPEOF, utilice el siguiente ejemplo.

```
SELECT JSON_TYPEOF(ARRAY(1,2));

+-------------+
| json_typeof |
```

```
+-------------+
| array       |
+-------------+
```

Funciones VARBYTE

AWS Clean Rooms admite las siguientes funciones VARBYTE.

Temas

- Función FROM_HEX
- Función FROM_VARBYTE
- Función TO_HEX
- Función TO_VARBYTE

Función FROM_HEX

FROM_HEX convierte un hexadecimal en un valor binario.

Sintaxis

```
FROM_HEX(hex_string)
```

Argumentos

hex_string

Cadena hexadecimal del tipo de datos `VARCHAR` o `TEXT` que hay que convertir. El formato debe ser un valor literal.

Tipo de retorno

`VARBYTE`

Ejemplo

Para convertir la representación hexadecimal de `'6162'` a un valor binario, use el siguiente ejemplo. El resultado se muestra automáticamente como representación hexadecimal del valor binario.

```
SELECT FROM_HEX('6162');

+----------+
| from_hex |
+----------+
|     6162 |
+----------+
```

Función FROM_VARBYTE

FROM_VARBYTE convierte un valor binario en una cadena de caracteres en el formato especificado.

Sintaxis

```
FROM_VARBYTE(binary_value, format)
```

Argumentos

binary_value

Un valor binario del tipo de datos VARBYTE.

formato

Formato de la cadena de caracteres devuelta. Los valores válidos que no distinguen entre mayúsculas y minúsculas son `hex`, `binary`, `utf-8` y `utf8`.

Tipo de retorno

VARCHAR

Ejemplo

Para convertir el valor binario `'ab'` a hexadecimal, use el siguiente ejemplo.

```
SELECT FROM_VARBYTE('ab', 'hex');

+--------------+
| from_varbyte |
```

```
+--------------+
|         6162 |
+--------------+
```

Función TO_HEX

TO_HEX convierte un número o valor binario en una representación hexadecimal.

Sintaxis

```
TO_HEX(value)
```

Argumentos

value

Un número o un valor binario (VARBYTE) que hay que convertir.

Tipo de retorno

VARCHAR

Ejemplo

Para convertir un número a su representación hexadecimal, use el siguiente ejemplo.

```
SELECT TO_HEX(2147676847);

+----------+
|  to_hex  |
+----------+
| 8002f2af |
+----------+To create a table, insert the VARBYTE representation of 'abc' to a
 hexadecimal number, and select the column with the value, use the following example.
```

Función TO_VARBYTE

TO_VARBYTE convierte una cadena con un formato especificado en un valor binario.

Sintaxis

```
TO_VARBYTE(string, format)
```

Argumentos

string

Una cadena CHAR o VARCHAR.

formato

El formato de la cadena de entrada. Los valores válidos que no distinguen entre mayúsculas y minúsculas son `hex`, `binary`, `utf-8` y `utf8`.

Tipo de retorno

VARBYTE

Ejemplo

Para convertir el 6162 hexadecimal en un valor binario, use el siguiente ejemplo. El resultado se muestra automáticamente como representación hexadecimal del valor binario.

```
SELECT TO_VARBYTE('6162', 'hex');

+------------+
| to_varbyte |
+------------+
|       6162 |
+------------+
```

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 Rooms no 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 Rooms no 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 ejemplo AWS 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 Rooms admite dos tipos de funciones de ventana: agregar y clasificar.

A continuación, se indican las funciones de agregado admitidas:

- Función de ventana AVG
- Función de ventana COUNT
- Función de ventana CUME_DIST
- Función de ventana DENSE_RANK
- Función de ventana FIRST_VALUE
- Función de ventana LAG
- Función de ventana LAST_VALUE
- Función de ventana LEAD
- Función de ventana LISTAGG
- Función de ventana MAX
- Función de ventana MEDIAN
- Función de ventana MIN
- Función de ventana DENSE_NTH
- Función de ventana PERCENTILE_CONT
- Función de ventana PERCENTILE_DISC
- Función de ventana RATIO_TO_REPORT
- Funciones de ventana STDDEV_SAMP y STDDEV_POP (STDDEV_SAMP y STDDEV son sinónimos)
- Función de ventana SUM
- Funciones de ventana VAR_SAMP y VAR_POP (VAR_SAMP y VARIANCE son sinónimos)

A continuación, se indican las funciones de clasificación admitidas:

- Función de ventana DENSE_RANK
- Función de ventana NTILE
- 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	
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 AVG

La función de ventana 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 ( [ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

Argumentos

expression

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

ALL

Con el argumento ALL, la función retiene todos los valores duplicados de la expresión para el conteo. El valor predeterminado es ALL. DISTINCT no se admite.

OVER

Especifica las cláusulas de ventana para las funciones de agregación. La cláusula OVER distingue funciones de agregación de ventana de las funciones de agregación de conjuntos normales.

PARTITION BY expr_list

Define la ventana para la función AVG en términos de una o más expresiones.

ORDER BY order_list

Ordena las filas dentro de cada partición. Si no se especifica PARTITION BY, ORDER BY utiliza toda la tabla.

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 dentro 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.

Tipos de datos

Los tipos de argumento compatibles con la función AVG son SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL y DOUBLE PRECISION.

Los tipos de retorno compatibles con la función AVG son los siguientes:

- BIGINT para los argumentos SMALLINT o INTEGER
- NUMERIC para argumentos BIGINT
- DOUBLE PRECISION para argumentos de punto flotante

Ejemplos

En el siguiente ejemplo, se calcula un promedio móvil de las cantidades vendidas por fecha; ordene los resultados por ID de fecha e ID de ventas:

```
select salesid, dateid, sellerid, qty,
avg(qty) over
(order by dateid, salesid rows unbounded preceding) as avg
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | avg
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
10005 | 2003-12-24 |        1 |  30 |  16
40001 | 2004-01-09 |        4 |  40 |  22
10006 | 2004-01-18 |        1 |  10 |  20
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  18
20002 | 2004-02-16 |        2 |  20 |  18
30003 | 2004-04-18 |        3 |  15 |  18
30004 | 2004-04-18 |        3 |  20 |  18
30007 | 2004-09-07 |        3 |  30 |  19
(11 rows)
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

Función de ventana COUNT

La función de ventana COUNT cuenta las filas definidas por la expresión.

La función COUNT tiene dos variaciones. COUNT(*) cuenta todas las filas en la tabla destino, incluya o no valores nulos. COUNT (expresión) computa la cantidad de filas con valores no NULL en una columna o expresión específicas.

Sintaxis

```
COUNT ( * | [ ALL ] expression) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

Argumentos

expression

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

ALL

Con el argumento ALL, la función retiene todos los valores duplicados de la expresión para el conteo. El valor predeterminado es ALL. DISTINCT no se admite.

OVER

Especifica las cláusulas de ventana para las funciones de agregación. La cláusula OVER distingue funciones de agregación de ventana de las funciones de agregación de conjuntos normales.

PARTITION BY expr_list

Define la ventana para la función COUNT en términos de una o más expresiones.

ORDER BY order_list

Ordena las filas dentro de cada partición. Si no se especifica PARTITION BY, ORDER BY utiliza toda la tabla.

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 dentro 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.

Tipos de datos

La función COUNT admite todos los tipos de datos de argumentos.

El tipo de retorno compatible con la función COUNT es BIGINT.

Ejemplos

En el siguiente ejemplo, se muestran los ID de ventas, la cantidad y el conteo de todas las filas desde el comienzo de la ventana de datos:

```
select salesid, qty,
count(*) over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | count
---------+-----+-----
10001 |  10 |   1
10005 |  30 |   2
10006 |  10 |   3
20001 |  20 |   4
20002 |  20 |   5
30001 |  10 |   6
30003 |  15 |   7
30004 |  20 |   8
30007 |  30 |   9
40001 |  40 |   10
40005 |  10 |   11
(11 rows)
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

En el siguiente ejemplo, se muestran los ID de ventas, la cantidad y el conteo de las filas no nulas desde el comienzo de la ventana de datos. (En la tabla WINSALES, la columna QTY_SHIPPED tiene algunos valores NULL.)

```
select salesid, qty, qty_shipped,
count(qty_shipped)
over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | qty_shipped | count
---------+-----+-------------+-------
10001 |  10 |          10 |   1
10005 |  30 |             |   1
10006 |  10 |             |   1
20001 |  20 |          20 |   2
20002 |  20 |          20 |   3
30001 |  10 |          10 |   4
30003 |  15 |             |   4
30004 |  20 |             |   4
30007 |  30 |             |   4
40001 |  40 |             |   4
40005 |  10 |          10 |   5
(11 rows)
```

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 retorno

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 retorno

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.

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 retorno

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 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 al 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 retorno

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 LISTAGG

Para cada grupo de una consulta, la función de ventana LISTAGG ordena las filas de ese cada grupo según la expresión ORDER BY y luego concatena los valores en una sola cadena.

LISTAGG es una función específica del nodo de computación. La función devuelve un error si la consulta no hace referencia a una tabla definida por el usuario o a una tabla AWS Clean Rooms del sistema.

Sintaxis

```
LISTAGG( [DISTINCT] expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
OVER ( [PARTITION BY partition_expression] )
```

Argumentos

DISTINCT

(Opcional) Una cláusula que elimina los valores duplicados de la expresión especificada antes de la concatenación. Se omiten los espacios posteriores y, por tanto, las cadenas `'a'` y `'a '` se tratan como duplicados. LISTAGG usa el primer valor que se encuentra. Para obtener más información, consulte Importancia de los espacios en blancos anteriores y posteriores.

expresión_de_agregación

Toda expresión válida (como un nombre de columna) que proporcione los valores para la agregación. Se ignoran los valores NULL y las cadenas vacías.

delimiter

(Opcional) La constante de cadena que separa los valores concatenados. El valor predeterminado es NULL.

AWS Clean Rooms admite cualquier cantidad de espacios en blanco iniciales o finales alrededor de una coma o dos puntos opcionales, así como una cadena vacía o cualquier número de espacios.

Estos son algunos ejemplos de valores válidos:

```
', '
': '
' '
```

WITHIN GROUP (ORDER BY order_list)

(Opcional) Una cláusula que especifica el orden de los valores agregados. Determinístico solamente si ORDER BY proporciona un ordenamiento único. La opción predeterminada es agregar todas las filas y devolver un valor único.

OVER

Una cláusula que especifica la partición de ventana. La cláusula OVER no puede contener una especificación de marco de ventana u ordenamiento.

PARTITION BY partition_expression

(Opcional) Establece el rango de registros para cada grupo en la cláusula OVER.

Devuelve

VARCHAR(MAX). Si el conjunto de resultados es mayor que el tamaño máximo VARCHAR (64K - 1 o 65535), LISTAGG devuelve el siguiente error:

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

Ejemplos

En los siguientes ejemplos, se utiliza la tabla WINSALES. Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

El siguiente ejemplo devuelve una lista de ID de vendedores ordenados por ID de vendedor.

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

```
  listagg
------------
 11122333344
...
...
 11122333344
 11122333344
   (11 rows)
```

El siguiente ejemplo devuelve una lista de ID de vendedores para el comprador B, ordenados por fecha.

```
select listagg(sellerid)
within group (order by dateid)
over () as seller
from winsales
where buyerid = 'b' ;

  seller
---------
    3233
    3233
    3233
    3233

(4 rows)
```

El siguiente ejemplo devuelve una lista separada por comas de fechas de ventas para el comprador B.

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

             dates
-----------------------------------------
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
```

```
(4 rows)
```

En el siguiente ejemplo, se usa DISTINCT para devolver una lista de fechas de venta distintas para el comprador B.

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

           dates
-------------------------------
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12

(4 rows)
```

El siguiente ejemplo devuelve una lista separada por comas de ID de ventas para cada ID de comprador.

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

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

```
        c  |10001,20002,30007,10006
(11 rows)
```

Función de ventana MAX

La función de ventana MAX devuelve el máximo de los valores de la expresión de entrada. La función MAX funciona con valores numéricos e ignora los valores NULL.

Sintaxis

```
MAX ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumentos

expression

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

ALL

Con el argumento ALL, la función retiene todos los valores duplicados de la expresión. El valor predeterminado es ALL. DISTINCT no se admite.

OVER

Una cláusula que especifica las cláusulas de ventana para las funciones de agregación. La cláusula OVER distingue funciones de agregación de ventana de las funciones de agregación de conjuntos normales.

PARTITION BY expr_list

Define la ventana para la función MAX en términos de una o más expresiones.

ORDER BY order_list

Ordena las filas dentro de cada partición. Si no se especifica PARTITION BY, ORDER BY utiliza toda la tabla.

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 dentro 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.

Tipos de datos

Acepta cualquier tipo de datos como entrada. Devuelve el mismo tipo de datos que expresión.

Ejemplos

En el siguiente ejemplo, se muestran los ID de ventas, la cantidad y la cantidad máxima desde el comienzo de la ventana de datos:

```
select salesid, qty,
max(qty) over (order by salesid rows unbounded preceding) as max
from winsales
order by salesid;

salesid | qty | max
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  30
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  30
30001 |  10 |  30
30003 |  15 |  30
30004 |  20 |  30
30007 |  30 |  30
40001 |  40 |  40
40005 |  10 |  40
(11 rows)
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

En el siguiente ejemplo, se muestran los ID de ventas, la cantidad y la cantidad máxima en un marco restringido:

```
select salesid, qty,
max(qty) over (order by salesid rows between 2 preceding and 1 preceding) as max
from winsales
order by salesid;

salesid | qty | max
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  20
30001 |  10 |  20
30003 |  15 |  20
30004 |  20 |  15
30007 |  30 |  20
40001 |  40 |  30
40005 |  10 |  40
(11 rows)
```

Función de ventana MEDIAN

Calcula el valor mediano para el rango de valores en una ventana o partición. Se ignoran los valores NULL en el rango.

MEDIAN es una función de distribución inversa que asume un modelo de distribución continua.

MEDIAN es una función específica del nodo de computación. La función devuelve un error si la consulta no hace referencia a una tabla definida por el usuario o a una tabla del sistema. AWS Clean Rooms

Sintaxis

```
MEDIAN ( median_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

Argumentos

expresión_de_mediana

Una expresión, como un nombre de columna, que proporciona un valor para los cuales determinar la media. La expresión debe tener un tipo de dato numérico o de fecha y hora o ser implícitamente convertible en uno.

OVER

Una cláusula que especifica la partición de ventana. La cláusula OVER no puede contener una especificación de marco de ventana u ordenamiento.

PARTITION BY partition_expression

Opcional. Una expresión que establece el rango de registros para cada grupo en la cláusula OVER.

Tipos de datos

El tipo de valor devuelto viene determinado por el tipo de datos de expresión_de_mediana. En la tabla siguiente, se muestra el tipo de valor devuelto para cada tipo de datos de expresión_de_mediana.

Tipo de entrada	Tipo de retorno
NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
FECHA	FECHA

Notas de uso

Si el argumento expresión_de_mediana es un tipo de dato DECIMAL definido con la precisión máxima de 38 dígitos, es posible que MEDIAN devuelva un resultado impreciso o un error. Si el valor de retorno de la función MEDIAN supera los 38 dígitos, el resultado se trunca para adaptarse, lo que genera pérdida de precisión. Si, durante la interpolación, un resultado intermedio supera la precisión máxima, se produce un desbordamiento numérico y la función devuelve un error.

Para evitar estas condiciones, recomendamos usar un tipo de dato con menor precisión o emitir el argumento median_expression con una precisión menor.

Por ejemplo, la función SUM con un argumento DECIMAL devuelve una precisión predeterminada de 38 dígitos. La escala del resultado es la misma que la escala del argumento. Entonces, por ejemplo, un SUM de una columna DECIMAL(5,2) devuelve un tipo de dato DECIMAL(38,2).

En el siguiente ejemplo, se utiliza una función SUM en el argumento median_expression (expresión_de_mediana) de una función MEDIAN. El tipo de datos de la columna PRICEPAID es DECIMAL (8,2), por lo que la función SUM devuelve un DECIMAL (38,2).

```
select salesid, sum(pricepaid), median(sum(pricepaid))
over() from sales where salesid < 10 group by salesid;
```

Para evitar una posible pérdida de precisión o un error de desbordamiento, convierta el resultado a un tipo de dato DECIMAL con menor precisión, como se muestra en el siguiente ejemplo.

```
select salesid, sum(pricepaid), median(sum(pricepaid)::decimal(30,2))
over() from sales where salesid < 10 group by salesid;
```

Ejemplos

En el siguiente ejemplo, se calcula la cantidad media de ventas para cada vendedor:

```
select sellerid, qty, median(qty)
over (partition by sellerid)
from winsales
order by sellerid;


sellerid qty median
----------------------------
1  10 10.0
1  10 10.0
1  30 10.0
2  20 20.0
2  20 20.0
3  10 17.5
3  15 17.5
3  20 17.5
3  30 17.5
```

```
4  10 25.0
4  40 25.0
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

Función de ventana MIN

La función de ventana MIN devuelve el mínimo de los valores de la expresión de entrada. La función MIN funciona con valores numéricos e ignora los valores NULL.

Sintaxis

```
MIN ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumentos

expression

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

ALL

Con el argumento ALL, la función retiene todos los valores duplicados de la expresión. El valor predeterminado es ALL. DISTINCT no se admite.

OVER

Especifica las cláusulas de ventana para las funciones de agregación. La cláusula OVER distingue funciones de agregación de ventana de las funciones de agregación de conjuntos normales.

PARTITION BY expr_list

Define la ventana para la función MIN en términos de una o más expresiones.

ORDER BY order_list

Ordena las filas dentro de cada partición. Si no se especifica PARTITION BY, ORDER BY utiliza toda la tabla.

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 dentro 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.

Tipos de datos

Acepta cualquier tipo de datos como entrada. Devuelve el mismo tipo de datos que expresión.

Ejemplos

En el siguiente ejemplo, se muestran los ID de ventas, la cantidad y la cantidad mínima desde el comienzo de la ventana de datos:

```
select salesid, qty,
min(qty) over
(order by salesid rows unbounded preceding)
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  10
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  10
40001 |  40 |  10
40005 |  10 |  10
(11 rows)
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

En el siguiente ejemplo, se muestran los ID de ventas, la cantidad y la cantidad mínima en un marco restringido:

```
select salesid, qty,
min(qty) over
(order by salesid rows between 2 preceding and 1 preceding) as min
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  20
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  15
40001 |  40 |  20
40005 |  10 |  30
(11 rows)
```

Función de ventana DENSE_NTH

La función de ventana NTH_VALUE devuelve el valor de la expresión de la fila especificada del marco de ventana relativo a la primera fila de la ventana.

Sintaxis

```
NTH_VALUE (expr, offset)
[ IGNORE NULLS | RESPECT NULLS ]
OVER
( [ PARTITION BY window_partition ]
[ ORDER BY window_ordering
                        frame_clause ] )
```

Argumentos

expr

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

desplazamiento

Determina el número de fila relativo a la primera fila en la ventana para la cual devolver la expresión. El desplazamiento puede ser una constante o una expresión y debe ser un valor entero positivo mayor que 0.

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.

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, el ordenamiento y el marco de ventana.

PARTITION BY window_partition

Establece el rango de registros para cada grupo en la cláusula OVER.

ORDER BY window_ordering

Ordena las filas dentro de cada partición. Si se omite ORDER BY, el marco predeterminado consta de todas las filas en la partición.

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.

La función de ventana NTH_VALUE admite expresiones que utilizan cualquiera de los AWS Clean Rooms tipos de datos. El valor devuelto es del mismo tipo que el valor de expr.

Ejemplos

En el siguiente ejemplo, se muestra la cantidad de asientos en el tercer lugar más grande de California, Florida y Nueva York, comparados con la cantidad de asientos en los demás lugares en esos estados:

```
select venuestate, venuename, venueseats,
nth_value(venueseats, 3)
ignore nulls
over(partition by venuestate order by venueseats desc
rows between unbounded preceding and unbounded following)
as third_most_seats
from (select * from venue where venueseats > 0 and
venuestate in('CA', 'FL', 'NY'))
order by venuestate;

venuestate |           venuename            | venueseats | third_most_seats
-----------+--------------------------------+------------+------------------
CA         | Qualcomm Stadium               |      70561 |            63026
CA         | Monster Park                   |      69843 |            63026
CA         | McAfee Coliseum                |      63026 |            63026
CA         | Dodger Stadium                 |      56000 |            63026
CA         | Angel Stadium of Anaheim       |      45050 |            63026
CA         | PETCO Park                     |      42445 |            63026
CA         | AT&T Park                      |      41503 |            63026
CA         | Shoreline Amphitheatre         |      22000 |            63026
FL         | Dolphin Stadium                |      74916 |            65647
FL         | Jacksonville Municipal Stadium |      73800 |            65647
FL         | Raymond James Stadium          |      65647 |            65647
FL         | Tropicana Field                |      36048 |            65647
NY         | Ralph Wilson Stadium           |      73967 |            20000
NY         | Yankee Stadium                 |      52325 |            20000
NY         | Madison Square Garden          |      20000 |            20000
(15 rows)
```

Función de ventana NTILE

La función de ventana NTILE divide las filas ordenadas en la partición en la cantidad especificada de grupos clasificados de igual tamaño en la medida que sea posible y devuelve el grupo en el que se inscribe una fila dada.

Sintaxis

```
NTILE (expr)
OVER (
[ PARTITION BY expression_list ]
[ ORDER BY order_list ]
)
```

Argumentos

expr

La cantidad de grupos clasificados y debe dar como resultado un valor entero positivo (mayor que 0) para cada partición. El argumento expr no debe admitir valores nulos.

OVER

Una cláusula que especifica la partición y ordenamiento de ventana. La cláusula OVER no puede tener una especificación de marco de ventana.

PARTITION BY window_partition

Opcional. El rango de registros para cada grupo en la cláusula OVER.

ORDER BY window_ordering

Opcional. Una expresión que ordena las filas dentro de cada partición. Si se omite la cláusula ORDER BY, el comportamiento de clasificación es el mismo.

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 retorno

BIGINT

Ejemplos

En el siguiente ejemplo, se clasifica en cuatro grupos de clasificación el precio pagado por los tickets de Hamlet el 26 de agosto de 2008. El conjunto de resultados es de 17 filas, divididas, casi uniformemente, entre las clasificaciones 1 a 4:

```
select eventname, caldate, pricepaid, ntile(4)
```

```
over(order by pricepaid desc) from sales, event, date
where sales.eventid=event.eventid and event.dateid=date.dateid and eventname='Hamlet'
and caldate='2008-08-26'
order by 4;

eventname |  caldate   | pricepaid | ntile
----------+------------+-----------+-------
Hamlet    | 2008-08-26 |   1883.00 |     1
Hamlet    | 2008-08-26 |   1065.00 |     1
Hamlet    | 2008-08-26 |    589.00 |     1
Hamlet    | 2008-08-26 |    530.00 |     1
Hamlet    | 2008-08-26 |    472.00 |     1
Hamlet    | 2008-08-26 |    460.00 |     2
Hamlet    | 2008-08-26 |    355.00 |     2
Hamlet    | 2008-08-26 |    334.00 |     2
Hamlet    | 2008-08-26 |    296.00 |     2
Hamlet    | 2008-08-26 |    230.00 |     3
Hamlet    | 2008-08-26 |    216.00 |     3
Hamlet    | 2008-08-26 |    212.00 |     3
Hamlet    | 2008-08-26 |    106.00 |     3
Hamlet    | 2008-08-26 |    100.00 |     4
Hamlet    | 2008-08-26 |     94.00 |     4
Hamlet    | 2008-08-26 |     53.00 |     4
Hamlet    | 2008-08-26 |     25.00 |     4
(17 rows)
```

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 retorno

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 PERCENTILE_CONT

PERCENTILE_CONT es una función de distribución inversa que asume un modelo de distribución continua. Toma un valor percentil y una especificación de ordenación, y devuelve un valor interpolado que se encuadraría dentro de ese valor percentil dado respecto de la especificación de ordenación.

PERCENTILE_CONT computa una interpolación linear entre valores después de ordenarlos. Utilizando el valor percentil `(P)` y la cantidad de filas no nulas `(N)` en el grupo de agregación, la función computa la cantidad de filas después de ordenar las filas según la especificación de ordenación. Esta cantidad de filas `(RN)` se computa según la fórmula `RN = (1+ (P*(N-1))`. El resultado final de la función de agregación se computa por interpolación lineal entre los valores de filas en números de filas `CRN = CEILING(RN)` y `FRN = FLOOR(RN)`.

El resultado final será el siguiente.

Si `(CRN = FRN = RN)`, entonces el resultado es `(value of expression from row at RN)`

De lo contrario, el resultado es el siguiente:

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

Puede especificar solamente la cláusula PARTITION en la cláusula OVER. Si se especifica PARTITION, para cada fila, PERCENTILE_CONT devuelve el valor que se encuadraría dentro de ese valor percentil especificado dentro de un conjunto de valores en una partición dada.

PERCENTILE_CONT es una función específica del nodo de computación. La función devuelve un error si la consulta no hace referencia a una tabla definida por el usuario o a una tabla del sistema. AWS Clean Rooms

Sintaxis

```
PERCENTILE_CONT ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

Argumentos

percentil

Constante numérica entre 0 y 1. Los valores nulos se ignoran en el cálculo.

WITHIN GROUP (ORDER BY expr)

Valores numéricos o de fecha/hora específicos para ordenar y calcular el percentil.

OVER

Especifica la partición de ventana. La cláusula OVER no puede contener una especificación de marco de ventana u ordenamiento.

PARTITION BY expr

Argumento opcional que establece el rango de registros para cada grupo en la cláusula OVER.

Devuelve

El tipo de retorno se determina por el tipo de datos de la expresión ORDER BY en la cláusula WITHIN GROUP. En la tabla siguiente, se muestra el tipo de retorno para cada tipo de datos de la expresión ORDER BY.

Tipo de entrada	Tipo de retorno
SMALLINTINTEGERBIGINT, NUMÉRICO, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
FECHA	FECHA
TIMESTAMP	TIMESTAMP

Notas de uso

Si la expresión ORDER BY es un tipo de dato DECIMAL definido con la precisión máxima de 38 dígitos, es posible que PERCENTILE_CONT devuelva un resultado impreciso o un error. Si el valor de retorno de la función PERCENTILE_CONT supera los 38 dígitos, el resultado se trunca para adaptarse, lo que genera pérdida de precisión. Si, durante la interpolación, un resultado intermedio supera la precisión máxima, se produce un desbordamiento numérico y la función devuelve un error. Para evitar estas condiciones, recomendamos usar un tipo de dato con menor precisión o emitir la expresión ORDER BY a una precisión menor.

Por ejemplo, la función SUM con un argumento DECIMAL devuelve una precisión predeterminada de 38 dígitos. La escala del resultado es la misma que la escala del argumento. Entonces, por ejemplo, un SUM de una columna DECIMAL(5,2) devuelve un tipo de dato DECIMAL(38,2).

En el siguiente ejemplo, se utiliza una función SUM en la cláusula ORDER BY de una función PERCENTILE_CONT. El tipo de datos de la columna PRICEPAID es DECIMAL (8,2), por lo que la función SUM devuelve un DECIMAL (38,2).

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid) desc) over()
from sales where salesid < 10 group by salesid;
```

Para evitar una posible pérdida de precisión o un error de desbordamiento, convierta el resultado a un tipo de dato DECIMAL con menor precisión, como se muestra en el siguiente ejemplo.

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid)::decimal(30,2) desc) over()
from sales where salesid < 10 group by salesid;
```

Ejemplos

En los siguientes ejemplos, se utiliza la tabla WINSALES. Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over() as median from winsales;

 sellerid | qty | median
----------+-----+--------
        1 |  10 |   20.0
        1 |  10 |   20.0
        3 |  10 |   20.0
        4 |  10 |   20.0
        3 |  15 |   20.0
        2 |  20 |   20.0
        3 |  20 |   20.0
        2 |  20 |   20.0
        3 |  30 |   20.0
        1 |  30 |   20.0
        4 |  40 |   20.0
(11 rows)
```

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

 sellerid | qty | median
----------+-----+--------
        2 |  20 |   20.0
        2 |  20 |   20.0
        4 |  10 |   25.0
        4 |  40 |   25.0
        1 |  10 |   10.0
```

```
        1 |  10 |   10.0
        1 |  30 |   10.0
        3 |  10 |   17.5
        3 |  15 |   17.5
        3 |  20 |   17.5
        3 |  30 |   17.5
(11 rows)
```

En el siguiente ejemplo, se calcula PERCENTILE_CONT y PERCENTILE_DISC de las ventas de tickets para vendedores en el estado de Washington.

```
SELECT sellerid, state, sum(qtysold*pricepaid) sales,
percentile_cont(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over(),
percentile_disc(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over()
from sales s, users u
where s.sellerid = u.userid and state = 'WA' and sellerid < 1000
group by sellerid, state;

 sellerid | state |  sales  | percentile_cont | percentile_disc
----------+-------+---------+-----------------+-----------------
      127 | WA    | 6076.00 |         2044.20 |         1531.00
      787 | WA    | 6035.00 |         2044.20 |         1531.00
      381 | WA    | 5881.00 |         2044.20 |         1531.00
      777 | WA    | 2814.00 |         2044.20 |         1531.00
       33 | WA    | 1531.00 |         2044.20 |         1531.00
      800 | WA    | 1476.00 |         2044.20 |         1531.00
        1 | WA    | 1177.00 |         2044.20 |         1531.00
(7 rows)
```

Función de ventana PERCENTILE_DISC

PERCENTILE_DISC es una función de distribución inversa que asume un modelo de distribución discreta. Toma un valor percentil y una especificación de ordenación, y devuelve un elemento del conjunto dado.

Para un valor percentil dado P, PERCENTILE_DISC ordena los valores de la expresión en la cláusula ORDER BY devuelve el valor con el menor valor de distribución acumulada (con respecto a la misma especificación de ordenamiento) que sea mayor que o igual al P.

Puede especificar solamente la cláusula PARTITION en la cláusula OVER.

PERCENTILE_DISC es una función específica del nodo de computación. La función devuelve un error si la consulta no hace referencia a una tabla definida por el usuario o a una tabla AWS Clean Rooms del sistema.

Sintaxis

```
PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

Argumentos

percentil

Constante numérica entre 0 y 1. Los valores nulos se ignoran en el cálculo.

WITHIN GROUP (ORDER BY expr)

Valores numéricos o de fecha/hora específicos para ordenar y calcular el percentil.

OVER

Especifica la partición de ventana. La cláusula OVER no puede contener una especificación de marco de ventana u ordenamiento.

PARTITION BY expr

Argumento opcional que establece el rango de registros para cada grupo en la cláusula OVER.

Devuelve

El mismo tipo de datos que la expresión ORDER BY en la cláusula WITHIN GROUP.

Ejemplos

En los siguientes ejemplos, se utiliza la tabla WINSALES. Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over() as median from winsales;
```

```
sellerid | qty | median
----------+-----+--------
        1 |  10 |     20
        3 |  10 |     20
        1 |  10 |     20
        4 |  10 |     20
        3 |  15 |     20
        2 |  20 |     20
        2 |  20 |     20
        3 |  20 |     20
        1 |  30 |     20
        3 |  30 |     20
        4 |  40 |     20
(11 rows)
```

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

sellerid | qty | median
----------+-----+--------
        2 |  20 |     20
        2 |  20 |     20
        4 |  10 |     10
        4 |  40 |     10
        1 |  10 |     10
        1 |  10 |     10
        1 |  30 |     10
        3 |  10 |     15
        3 |  15 |     15
        3 |  20 |     15
        3 |  30 |     15
(11 rows)
```

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 Rooms suma 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 retorno

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 ID de ventas 10001 y 10006 tienen valores 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 RATIO_TO_REPORT

Calcula la relación de un valor con la suma de los valores en una ventana o partición. La relación de un valor de informe se determina utilizando la fórmula:

`value of` ratio_expression `argument for the current row / sum of` ratio_expression `argument for the window or partition`

El siguiente conjunto de datos ilustra el uso de esta fórmula:

```
Row# Value Calculation RATIO_TO_REPORT
1 2500 (2500)/(13900) 0.1798
2 2600 (2600)/(13900) 0.1870
3 2800 (2800)/(13900) 0.2014
4 2900 (2900)/(13900) 0.2086
5 3100 (3100)/(13900) 0.2230
```

El rango de valor de retorno es 0 a 1, inclusive. Si ratio_expression es NULL, el valor devuelto es NULL.

Sintaxis

```
RATIO_TO_REPORT ( ratio_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

Argumentos

ratio_expression

Una expresión, como un nombre de columna, que proporciona el valor para el cual determinar la relación. La expresión debe tener un tipo de dato numérico o ser implícitamente convertible a un dato numérico.

No se puede usar ninguna otra función analítica en ratio_expression.

OVER

Una cláusula que especifica la partición de ventana. La cláusula OVER no puede contener una especificación de marco de ventana u ordenamiento.

PARTITION BY partition_expression

Opcional. Una expresión que establece el rango de registros para cada grupo en la cláusula OVER.

Tipo de retorno

FLOAT8

Ejemplos

En el siguiente ejemplo, se calculan las relaciones de porcentaje de las cantidades de ventas para cada vendedor:

```
select sellerid, qty, ratio_to_report(qty)
over (partition by sellerid)
from winsales;

sellerid qty  ratio_to_report
----------------------------------------
2  20.12312341      0.5
2  20.08630000      0.5
4  10.12414400      0.2
4  40.23000000      0.8
1  30.37262000      0.6
1  10.64000000      0.21
1  10.00000000      0.2
3  10.03500000      0.13
3  15.14660000      0.2
3  30.54790000      0.4
3  20.74630000      0.27
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

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 retorno

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.

Funciones de ventana STDDEV_SAMP y STDDEV_POP

Las funciones de ventana 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). Véase también Funciones STDDEV_SAMP y STDDEV_POP.

STDDEV_SAMP y STDDEV son sinónimos para la misma función.

Sintaxis

```
STDDEV_SAMP | STDDEV | STDDEV_POP
( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

Argumentos

expression

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

ALL

Con el argumento ALL, la función retiene todos los valores duplicados de la expresión. El valor predeterminado es ALL. DISTINCT no se admite.

OVER

Especifica las cláusulas de ventana para las funciones de agregación. La cláusula OVER distingue funciones de agregación de ventana de las funciones de agregación de conjuntos normales.

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 PARTITION BY, ORDER BY utiliza toda la tabla.

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 dentro 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.

Tipos de datos

Los tipos de argumento compatibles con las funciones STDDEV son SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL y DOUBLE PRECISION.

Independientemente del tipo de datos de la expresión, el tipo de retorno de una función STDDEV es un número de doble precisión.

Ejemplos

En el siguiente ejemplo, se muestra cómo usar las funciones STDDEV_POP y VAR_POP como funciones de ventana. La consulta computa la varianza de población y la desviación estándar de población para valores PRICEPAID en la tabla SALES.

```
select salesid, dateid, pricepaid,
round(stddev_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as stddevpop,
round(var_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as varpop
from sales
order by 2,1;

salesid | dateid | pricepaid | stddevpop | varpop
--------+--------+-----------+-----------+---------
  33095 |   1827 |    234.00 |         0 |       0
  65082 |   1827 |    472.00 |       119 |   14161
  88268 |   1827 |    836.00 |       248 |   61283
  97197 |   1827 |    708.00 |       230 |   53019
 110328 |   1827 |    347.00 |       223 |   49845
 110917 |   1827 |    337.00 |       215 |   46159
 150314 |   1827 |    688.00 |       211 |   44414
 157751 |   1827 |   1730.00 |       447 |  199679
 165890 |   1827 |   4192.00 |      1185 | 1403323
...
```

La desviación estándar de muestra y las funciones de varianza se pueden usar de la misma manera.

Función de ventana SUM

La función de ventana 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.

Sintaxis

```
SUM ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

Argumentos

expression

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

ALL

Con el argumento ALL, la función retiene todos los valores duplicados de la expresión. El valor predeterminado es ALL. DISTINCT no se admite.

OVER

Especifica las cláusulas de ventana para las funciones de agregación. La cláusula OVER distingue funciones de agregación de ventana de las funciones de agregación de conjuntos normales.

PARTITION BY expr_list

Define la ventana para la función SUM en términos de una o más expresiones.

ORDER BY order_list

Ordena las filas dentro de cada partición. Si no se especifica PARTITION BY, ORDER BY utiliza toda la tabla.

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 dentro 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.

Tipos de datos

Los tipos de argumento compatibles con la función SUM son SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL y DOUBLE PRECISION.

Los tipos de retorno compatibles con la función SUM son los siguientes:

- BIGINT para los argumentos SMALLINT o INTEGER
- NUMERIC para argumentos BIGINT
- DOUBLE PRECISION para argumentos de punto flotante

Ejemplos

En el siguiente ejemplo, se crea una suma acumulativa (continua) de cantidades de ventas ordenadas por fecha e ID de ventas:

```
select salesid, dateid, sellerid, qty,
sum(qty) over (order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  20
10005 | 2003-12-24 |        1 |  30 |  50
40001 | 2004-01-09 |        4 |  40 |  90
10006 | 2004-01-18 |        1 |  10 | 100
20001 | 2004-02-12 |        2 |  20 | 120
40005 | 2004-02-12 |        4 |  10 | 130
20002 | 2004-02-16 |        2 |  20 | 150
30003 | 2004-04-18 |        3 |  15 | 165
30004 | 2004-04-18 |        3 |  20 | 185
30007 | 2004-09-07 |        3 |  30 | 215
(11 rows)
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

En el siguiente ejemplo, se crea una suma acumulativa (continua) de cantidades de ventas por fecha, se dividen los resultados por ID de vendedor y se ordenan los resultados por fecha e ID de ventas dentro de la partición:

```
select salesid, dateid, sellerid, qty,
sum(qty) over (partition by sellerid
order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
```

```
10005 | 2003-12-24 |        1 |  30 |  40
40001 | 2004-01-09 |        4 |  40 |  40
10006 | 2004-01-18 |        1 |  10 |  50
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  50
20002 | 2004-02-16 |        2 |  20 |  40
30003 | 2004-04-18 |        3 |  15 |  25
30004 | 2004-04-18 |        3 |  20 |  45
30007 | 2004-09-07 |        3 |  30 |  75
(11 rows)
```

En el siguiente ejemplo, se enumeran en forma secuencial todas las filas del conjunto de resultados, ordenadas por las columnas SELLERID y SALESID:

```
select salesid, sellerid, qty,
sum(1) over (order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid |  qty | rownum
--------+----------+------+--------
10001 |        1 |   10 |     1
10005 |        1 |   30 |     2
10006 |        1 |   10 |     3
20001 |        2 |   20 |     4
20002 |        2 |   20 |     5
30001 |        3 |   10 |     6
30003 |        3 |   15 |     7
30004 |        3 |   20 |     8
30007 |        3 |   30 |     9
40001 |        4 |   40 |    10
40005 |        4 |   10 |    11
(11 rows)
```

Para ver una descripción de la tabla WINSALES, consulte Tabla de muestra para ejemplos de funciones de ventana.

En el siguiente ejemplo, se enumeran en forma secuencial todas las filas en el conjunto de resultados, se dividen los resultados por SELLERID y se ordenan los resultados por SELLERID y SALESID dentro de la partición:

```
select salesid, sellerid, qty,
```

```
sum(1) over (partition by sellerid
order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid | qty | rownum
---------+----------+-----+--------
10001 |        1 |  10 |      1
10005 |        1 |  30 |      2
10006 |        1 |  10 |      3
20001 |        2 |  20 |      1
20002 |        2 |  20 |      2
30001 |        3 |  10 |      1
30003 |        3 |  15 |      2
30004 |        3 |  20 |      3
30007 |        3 |  30 |      4
40001 |        4 |  40 |      1
40005 |        4 |  10 |      2
(11 rows)
```

Funciones de ventana VAR_SAMP y VAR_POP

Las funciones de ventana 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). Véase también Funciones VAR_SAMP y VAR_POP.

VAR_SAMP y VARIANCE son sinónimos para la misma función.

Sintaxis

```
VAR_SAMP | VARIANCE | VAR_POP
( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

Argumentos

expression

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

ALL

Con el argumento ALL, la función retiene todos los valores duplicados de la expresión. El valor predeterminado es ALL. DISTINCT no se admite.

OVER

Especifica las cláusulas de ventana para las funciones de agregación. La cláusula OVER distingue funciones de agregación de ventana de las funciones de agregación de conjuntos normales.

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 PARTITION BY, ORDER BY utiliza toda la tabla.

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 dentro 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.

Tipos de datos

Los tipos de argumento compatibles con las funciones VARIANCE son SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL y DOUBLE PRECISION.

Independientemente del tipo de datos de la expresión, el tipo de retorno de una función VARIANCE es un número de doble precisión.

Condiciones SQL en AWS Clean Rooms

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.

> **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.

En él se admiten las siguientes condiciones de SQL AWS Clean Rooms.

Temas

Condiciones de comparación

Las condiciones de comparación indican relaciones lógicas entre dos valores. Todas las condiciones de comparación son operadores binarios y devuelven un tipo de valor booleano. AWS Clean Rooms admite los operadores de comparación que se describen en la siguiente tabla:

Operador	Sintaxis	Descripción
<	a < b	El valor a es inferior al valor b.

Operador	Sintaxis	Descripción
>	`a > b`	El valor a es superior al valor b.
<=	`a <= b`	El valor a es inferior o igual al valor b.
>=	`a >= b`	El valor a es superior o igual al valor b.
=	`a = b`	El valor a es igual al valor b.
<> o !=	`a <> b or a != b`	El valor a no es igual al valor b.
a = TRUE	`a IS TRUE`	El valor a es TRUE booleano.

Notas de uso

= ANY | SOME

Las palabras clave ANY y SOME son sinónimos de la condición IN. Las palabras clave ANY y SOME devuelven el valor true si la comparación es verdadera para al menos un valor devuelto por una subconsulta que devuelve uno o más valores. AWS Clean Rooms solo admite la condición = (igual a) con ANY y SOME. No se admiten las condiciones de desigualdad.

> **Note**
>
> Se admite el predicado ALL.

<> ALL

La palabra clave ALL es sinónimo de NOT IN (consulte la condición Condición IN) y devuelve true si la expresión no está incluida en los resultados de una subconsulta. AWS Clean Rooms solamente admite la condición <> o != (no es igual) con ALL. No se admiten otras condiciones de comparación.

IS TRUE/FALSE/UNKNOWN

Los valores distintos de cero tienen un valor de TRUE, 0 tiene un valor de FALSE, y los valores nulos tienen un valor de UNKNOWN. Consulte el tipo de datos Tipo booleano.

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 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	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

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 el patrón especificado en la expresión condicional y devuelve true o false dependiendo de si encuentra o no una coincidencia. AWS Clean Rooms utiliza tres métodos de coincidencia de patrones siguientes:

- 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, mientras que ILIKE realiza una coincidencia que no distingue entre mayúsculas y minúsculas.
- Expresiones regulares SIMILAR TO

  El operador SIMILAR TO relaciona una expresión de cadena con un patrón de expresión regular de SQL estándar, que puede incluir un conjunto de metacaracteres de coincidencia de patrón que incluyen los dos admitidos por el operador LIKE. SIMILAR TO relaciona la cadena completa y realiza una coincidencia que distingue entre mayúsculas y minúsculas.

Temas

- LIKE
- SIMILAR TO

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; ILIKE no distingue entre mayúsculas y minúsculas.

Sintaxis

```
expression [ NOT ] LIKE | ILIKE pattern [ ESCAPE 'escape_char' ]
```

Argumentos

expression

Una expresión de carácter UTF-8 válido, como un nombre de columna.

LIKE | ILIKE

LIKE realiza una coincidencia de patrones que distingue entre mayúsculas y minúsculas. ILIKE ejecuta una coincidencia de patrones sin distinción entre mayúsculas y minúsculas para caracteres UTF-8 (ASCII) de un byte. 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.

Al contrario que los predicados de comparación, como = y <>, los predicados LIKE e ILIKE no omiten implícitamente los espacios finales. Para omitir los espacios finales, utilice RTRIM o convierta explícitamente una columna CHAR en VARCHAR.

El operador ~~ es equivalente a LIKE y ~~* es equivalente a ILIKE. Además, los operadores !~~ y !~~* son equivalentes a NOT LIKE y NOT ILIKE.

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:

"."	Descripción
%	Coincide con cualquier secuencia de cero o más caracteres.
_	Coincide con cualquier carácter.

Ejemplos

En la tabla siguiente se muestran ejemplos de coincidencia de patrones a través de LIKE:

Expresión	Devuelve
`'abc' LIKE 'abc'`	True
`'abc' LIKE 'a%'`	True
`'abc' LIKE '_B_'`	False
`'abc' ILIKE '_B_'`	True
`'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 encuentran todas las ciudades cuyos terceros y cuartos caracteres son "ea". El comando usa ILIKE para demostrar que no distingue entre mayúsculas y minúsculas:

```
select distinct city from users where city ilike '__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)
```

En el siguiente ejemplo, se utiliza el operador ~~* para realizar una búsqueda que no distinga mayúsculas de minúsculas (ILIKE) de ciudades que comiencen por "Ag".

```
select distinct city from users where city ~~* 'Ag%' order by city;

city
------------
Agat
Agawam
Agoura Hills
Aguadilla
```

SIMILAR TO

El operador SIMILAR TO relaciona una expresión de cadena, como el nombre de una columna, con un patrón de expresión regular de SQL estándar. Un patrón de expresión regular de SQL estándar puede incluir un conjunto de metacaracteres de coincidencia de patrón, incluidos los dos admitidos por el operador LIKE.

El operador SIMILAR TO devuelve true solo si su patrón coincide con la cadena completa, a diferencia del comportamiento de la expresión regular POSIX, donde el patrón puede coincidir con cualquier parte de la cadena.

SIMILAR TO realiza una coincidencia de patrones que distingue entre mayúsculas y minúsculas.

> **Note**
>
> La coincidencia de expresiones regulares a través de SIMILAR TO es costosa informáticamente. Recomendamos utilizar LIKE cuando sea posible, especialmente cuando procesa una gran cantidad de filas. Por ejemplo, las siguientes consultas son idénticas desde el punto de vista funcional, pero la consulta que utiliza LIKE se ejecuta varias veces más rápido que la consulta que utiliza una expresión regular:
>
> ```
> select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
> select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die
> %';
> ```

Sintaxis

```
expression [ NOT ] SIMILAR TO pattern [ ESCAPE 'escape_char' ]
```

Argumentos

expression

Una expresión de carácter UTF-8 válido, como un nombre de columna.

SIMILAR TO

SIMILAR TO ejecuta una coincidencia de patrones con distinción entre mayúsculas y minúsculas en la cadena completa de expresión.

pattern

Una expresión de carácter UTF-8 válido que representa un patrón de expresión regular de SQL estándar.

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, el patrón solo representa la propia cadena.

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.

SIMILAR TO admite los siguientes metacaracteres de coincidencia de patrón:

"."	Descripción
%	Coincide con cualquier secuencia de cero o más caracteres.
_	Coincide con cualquier carácter.
|	Denota una alternancia (cualquiera de dos alternativas).
*	Repite el elemento anterior cero o más veces.
+	Repite el elemento anterior una o más veces.
?	Repite el elemento anterior cero o una vez.
{m}	Repite el elemento anterior exactamente m veces.
{m,}	Repite el elemento anterior m o más veces.
{m,n}	Repite el elemento anterior al menos m y no más de n veces.
()	Los paréntesis agrupan elementos en un único elemento lógico.
[...]	Una expresión de corchetes especifica una clase de carácter, como en las expresiones regulares POSIX.

Ejemplos

En la siguiente tabla se muestran ejemplos de coincidencia de patrones a través de SIMILAR TO:

Expresión	Devuelve
`'abc' SIMILAR TO 'abc'`	True
`'abc' SIMILAR TO '_b_'`	True
`'abc' SIMILAR TO '_A_'`	False
`'abc' SIMILAR TO '%(b|d)%'`	True
`'abc' SIMILAR TO '(b|c)%'`	False
`'AbcAbcdefgefg12efgefg12' SIMILAR TO '((Ab)?c)+d((efg)+(12))+'`	True
`'aaaaaab11111xy' SIMILAR TO 'a{6}_[0-9]{5}(x|y){2}'`	True
`'$0.87' SIMILAR TO '$[0-9]+(.[0-9][0-9])?'`	True

En el siguiente ejemplo, se encuentran las ciudades cuyos nombres contienen “E” o “H”:

```
SELECT DISTINCT city FROM users
WHERE city SIMILAR TO '%E%|%H%' ORDER BY city LIMIT 5;

      city
-----------------
 Agoura Hills
 Auburn Hills
 Benton Harbor
 Beverly Hills
 Chicago Heights
```

En el siguiente ejemplo se usa la cadena de escape predeterminada ("\\") para buscar cadenas que incluyan "_":

```
SELECT tablename, "column" FROM my_table_def
WHERE "column" SIMILAR TO '%start\\_%'

ORDER BY tablename, "column" LIMIT 5;

        tablename         |       column
--------------------------+---------------------
 my_abort_idle           | idle_start_time
 my_abort_idle           | txn_start_time
 my_analyze_compression | start_time
 my_auto_worker_levels   | start_level
 my_auto_worker_levels   | start_wlm_occupancy
```

En el siguiente ejemplo se especifica "^" como la cadena de escape y, luego, se utiliza la cadena de escape para buscar cadenas que incluyan "_":

```
SELECT tablename, "column" FROM my_table_def

WHERE "column" SIMILAR TO '%start^_%' ESCAPE '^'
ORDER BY tablename, "column" LIMIT 5;

        tablename         |       column
--------------------------+---------------------
 stcs_abort_idle           | idle_start_time
 stcs_abort_idle           | txn_start_time
 stcs_analyze_compression | start_time
 stcs_auto_worker_levels   | start_level
 stcs_auto_worker_levels   | start_wlm_occupancy
```

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 condición NULL realiza una prueba en busca de valores nulos cuando hay un valor que falta o un valor desconocido.

Sintaxis

```
expression IS [ NOT ] NULL
```

Argumentos

expression

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

expression

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)
```

Sintaxis

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

Consultar datos anidados

AWS Clean Rooms ofrece acceso compatible con SQL a datos relacionales y anidados.

AWS Clean Rooms utiliza notación con puntos y subíndice de matriz para la navegación de rutas al acceder a datos anidados. También habilita los elementos de cláusula FROM para iterar matrices y utilizarlas para las operaciones de desanidamiento. Los siguientes temas ofrecen descripciones de los diferentes patrones de consulta que combinan el uso del tipo de datos matriz/estructura/mapa con la navegación, el desanidamiento y la combinación de rutas y matrices.

Temas

Navegación

AWS Clean Rooms permite la navegación en matrices y estructuras utilizando la notación con 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 desanidar consultas, AWS Clean Rooms permite 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 matrices al iterar la matriz usando la palabra clave AT. 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 matriz `f.multi_level_array` AS itera `multi_level_array`. El elemento AS de la matriz representa la iteración sobre las matrices en `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 inferir los tipos correspondientes a los datos, AWS Clean Rooms utiliza la función JSON_TYPEOF, que devuelve el tipo (el nivel superior) del valor de 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
Comandos y funciones SQL: actualización	Se han agregado ejemplos de la cláusula JOIN, EXCEPTO el operador set, la 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 admite las siguientes funciones de SQL: Array, SUPER y VARBYTE. Ahora se admiten las siguientes funciones matemáticas: ACOS, ASIN, ATAN, ATAN2, COT, DEXP, PI, POW, RADIANS y SIN. Ahora se admiten las siguientes funciones JSON: CAN_JSON_	6 de octubre de 2023

	PARSE, JSON_PARSE y JSON_SERIALIZE.	
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.