Referência SQL

AWS Clean Rooms

AWS Clean Rooms: Referência SQL

Table of Contents

Visão geral do SQL em AWS Clean Rooms

Bem-vindo à Referência SQL do AWS Clean Rooms.

AWS Clean Roomsé construído com base na Linguagem de Consulta Estruturada (SQL) padrão do setor, uma linguagem de consulta que consiste em comandos e funções que você usa para trabalhar com bancos de dados e objetos de banco de dados. SQL também impõe regras relativas ao uso de tipos de dados, expressões e literais.

Os tópicos a seguir fornecem informações gerais sobre as convenções e regras de nomenclatura usadas nesta Referência SQL.

Tópicos

As seções a seguir fornecem informações sobre literais, tipos de dados, comandos SQL, tipos de funções SQL e condições SQL nas quais você pode usar. AWS Clean Rooms

Para obter mais informações sobreAWS Clean Rooms, consulte o Guia AWS Clean Rooms do usuário e a Referência AWS Clean Rooms da API.

Convenções de referência do SQL

Esta seção explica as convenções usadas para escrever a sintaxe das expressões, comandos e funções SQL.

Caractere	Descrição
CAPS	Palavras em letras maiúsculas são palavras chave.
[ ]	Parênteses denotam argumentos opcionais. Vários argumentos em parênteses indicam que você pode escolher qualquer número de argumentos. Além disso,

Caractere	Descrição
	os argumentos entre colchetes em linhas separadas indicam que o analisador do espera que os argumento s estejam na ordem em que estão listados na sintaxe.
{ }	As chaves indicam que será necessário escolher um dos argumentos nelas.
|	Barras verticais indicam que você pode escolher entre os argumentos.
itálico	As palavras em itálico indicam os espaços reservado s. Você deve inserir o valor apropriado no lugar da palavra em itálico.
...	Uma elipse indica que você pode repetir o elemento anterior.
'	Palavras entre aspas simples indicam que você deve digitar as aspas.

Regras de nomeação de SQL

As seções a seguir explicam as regras de nomeação de SQL no AWS Clean Rooms.

Tópicos

- Nomes e colunas de associação de tabelas configurados
- Palavras reservadas

Nomes e colunas de associação de tabelas configurados

Os membros que podem consultar usam nomes de associação de tabela configurados como nomes de tabela nas consultas. Os nomes de associações de tabelas configurados e as colunas de tabelas configuradas podem ter aliases em consultas.

As regras de nomenclatura a seguir se aplicam a nomes de associação de tabela configurados, nomes de colunas de tabela configurados e aliases:

- Eles devem usar somente caracteres alfanuméricos, sublinhado (_) ou hífen (-), mas não podem começar ou terminar com um hífen.
  - (Somente regra de análise personalizada) Eles podem usar o cifrão ($), mas não podem usar um padrão que siga uma constante de string cotada em dólares.

    Uma constante de string cotada em dólares consiste em:

    - um cifrão
    - uma “tag” opcional de zero ou mais caracteres
    - outro cifrão
    - sequência arbitrária de caracteres que compõe o conteúdo da string
    - um cifrão
    - a mesma etiqueta que iniciou a cotação do dólar
    - um cifrão

      Por exemplo: `$$invalid$$`
- Eles não podem conter caracteres consecutivos de hífen (-).
- Eles não podem começar com nenhum dos seguintes prefixos:

  `padb_`, `pg_`, `stcs_`, `stl_`, `stll_`, `stv_`, `svcs_`, `svl_`, `svv_`, `sys_`, `systable_`
- Eles não podem conter caracteres de barra invertida (\), aspas (') ou espaços que não estejam entre aspas duplas.
- Se começarem com um caractere não alfabético, devem estar entre aspas duplas (" ").
- Se contiverem um caractere de hífen (-), devem estar entre aspas duplas (" ").
- Eles devem ter entre 1 e 127 caracteres.
- Palavras reservadas devem estar entre aspas duplas (" ").
- Os seguintes nomes de colunas são reservados e não podem ser usados em AWS Clean Rooms (mesmo com aspas):
  - oid
  - tableoid
  - xmin
  - cmin
  - xmax
  - cmax

- ctid

Palavras reservadas

A seguir está uma lista de palavras reservadas em AWS Clean Rooms.

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

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

Suporte de tipo de dados pelo mecanismo SQL

AWS Clean Rooms oferece suporte a vários mecanismos e dialetos SQL. Compreender os sistemas de tipos de dados nessas implementações é crucial para uma colaboração e análise de dados bem-sucedidas. As tabelas a seguir mostram os tipos de dados equivalentes em AWS Clean Rooms SQL, Snowflake SQL e Spark SQL.

Tipos de dados numéricos

Os tipos numéricos representam vários tipos de números, desde números inteiros precisos até valores aproximados de ponto flutuante. A escolha do tipo numérico afeta tanto os requisitos de armazenamento quanto a precisão computacional. Os tipos inteiros variam de acordo com o tamanho do byte, enquanto os tipos decimal e de ponto flutuante oferecem diferentes opções de precisão e escala.

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Inteiro de 8 bytes	BIGINT	Não compatível	GRANDE, LONGO	Números inteiros assinados de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.
Inteiro de 4 bytes	INT	Não compatível	INT, INTEGER	Números inteiros assinados de -2.147.483.648 a 2.147.483.647
Inteiro de 2 bytes	SMALLINT	Não compatível	PEQUENO, CURTO	Números inteiros assinados

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
				de -32.768 a 32.767
Inteiro de 1 byte	Não compatível	Sem compatibilidade	TINYINT, BYTE	Números inteiros assinados de -128 a 127
Flutuador de dupla precisão	DUPLA, DUPLA PRECISÃO	FLUTUA, FLOAT4, FLOAT8, DUPLA, DUPLA PRECISÃO, REAL	DOUBLE	Números de ponto flutuante de precisão dupla de 8 bytes
Flutuador de precisão única	REAL, FLUTUAR	Não compatível	FLOAT	Números de ponto flutuante de precisão única de 4 bytes

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Decimal (precisão fixa)	DECIMAL	DECIMAL, NUMÉRICO, NÚMERO **Note** O Snowflake atribui automaticamente o alias de tipos numéricos exatos de menor largura (INT, BIGINT, SMALLINT etc.) para NUMBER.	DECIMAL, NUMÉRICO,	Números decimais assinados com precisão arbitrária
Decimal (com precisão)	DECIMAL (p)	DECIMAL (p), NÚMERO (p)	DECIMAL (p)	Números decimais de precisão fixa

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Decimal (com escala)	DECIMAL (p,s)	DECIMAL (p, s), NÚMERO (p, s)	DECIMAL (p,s)	Números decimais de precisão fixa com escala

Tipos de dados booleanos

Os tipos booleanos representam valores true/false lógicos simples. Esses tipos são consistentes em todos os mecanismos SQL e são comumente usados para sinalizadores, condições e operações lógicas.

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Booleano	BOOLEAN	BOOLEAN	BOOLEAN	Represent a true/false valores

Tipos de dados de data e hora

Os tipos de data e hora lidam com dados temporais, com níveis variados de precisão e reconhecimento de fuso horário. Esses tipos oferecem suporte a formatos diferentes para armazenar datas, horas e carimbos de data/hora, com opções para incluir ou excluir informações de fuso horário.

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Data	DATE	DATE	DATE	Valores de data (ano, mês, dia) sem fuso horário

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Tempo	TIME	Não compatível	Sem compatibilidade	Hora do dia em UTC, sem fuso horário
Tempo com TZ	TIMETZ	Não compatível	Sem compatibilidade	Hora do dia em UTC, com fuso horário
Timestamp	TIMESTAMP	TIMESTAMP, TIMESTAMP_NTZ	TIMESTAMP_NTZ	Time stamp sem fuso horário **Note** NTZ indica “Sem fuso horário”
Timestamp com TZ	TIMESTAMPTZ	TIMESTAMP_LTZ	TIMESTAMP, TIMESTAMP_LTZ	Carimbo de data/hora com fuso horário local **Note** LTZ indica “fuso horário local”

Tipos de dados de caracteres

Os tipos de caracteres armazenam dados textuais, oferecendo opções de tamanho fixo e tamanho variável. Esses tipos lidam com cadeias de texto e dados binários, com especificações de comprimento opcionais para controlar a alocação de armazenamento.

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Caractere de tamanho fixo	CHAR	CHAR, CHARACTER	CHAR, CHARACTER	String de caracteres com comprimento fixo
Caractere de comprimento fixo com comprimento	CHAR(n)	CHAR(n), CHARACTER (n)	CHAR(n), CHARACTER (n)	Cadeia de caracteres de comprimento fixo com comprimento especificado
Caractere de comprimento variável	VARCHAR	VARCHAR, STRING, TEXTO	VARCHAR, CORDA	String de caracteres de comprimento variável
Caractere de comprimento variável com comprimento	VARCHAR(n)	VARCHAR (n), CADEIA DE CARACTERES (n), TEXTO (n)	VARCHAR(n)	Cadeia de caracteres de comprimento variável com limite de comprimento
Binário	VARBYTE	BINARY, VARBINARY	BINARY	Sequência binária de bytes

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Binário com comprimento	VARBYTE(n)	Não compatível	Sem compatibilidade	Sequência binária de bytes com limite de comprimento

Tipos de dados estruturados

Os tipos estruturados permitem uma organização complexa de dados combinando vários valores em campos únicos. Isso inclui matrizes para coleções ordenadas, mapas para pares de valores-chave e estruturas para criar estruturas de dados personalizadas com campos nomeados.

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Array	MATRIZ <type>	MATRIZ (tipo)	MATRIZ <type>	Sequência ordenada de elementos do mesmo tipo **Note** Os tipos de matriz devem conter elementos do mesmo tipo

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Mapa	MAPA<key, value>	MAP (chave, valor)	MAPA<key, value>	Coleção de pares de valores-chave **Note** Os tipos de mapa devem conter elementos do mesmo tipo

Tipo de dados	AWS Clean Rooms SQL	SQL do Snowflake	Spark SQL	Description
Struct	ESTRUTURA< field1: type1, field2: type2>	OBJETO (campo1 tipo1, campo2 tipo2)	ESTRUTURA < field1: type1, field2: type2 >	Estrutura com campos nomeados de tipos especificados **Note** A sintaxe do tipo estrutura do pode variar um pouco entre as implementações
Super	SUPER	Não compatível	Sem compatibilidade	Tipo flexível que suporta todos os tipos de dados, incluindo tipos complexos

AWS Clean Rooms SQL do Spark

AWS Clean Rooms O Spark SQL impõe regras relacionadas ao uso de tipos de dados, expressões e literais.

Para obter mais informações sobre o AWS Clean Rooms Spark SQL, consulte o Guia AWS Clean Rooms do usuário e a Referência da AWS Clean Rooms API.

Os tópicos a seguir fornecem informações sobre literais, tipos de dados, comandos, funções e condições compatíveis com o AWS Clean Rooms Spark SQL.

Tópicos

Literais

Um literal ou constante é um valor fixo de dados, composto de uma sequência de caracteres ou uma constante numérica.

AWS Clean Rooms O Spark SQL oferece suporte a vários tipos de literais, incluindo:

- Literais numéricos para números inteiros, decimais e números de ponto flutuante.
- Literais de caracteres, também chamados de cadeias de caracteres, cadeias de caracteres ou constantes de caracteres, usados para especificar um valor de cadeia de caracteres.
- Literais de data, de hora e de carimbo de data/hora, usados com tipos de dados de data e hora. Para obter mais informações, consulte Literais de data, hora e timestamp.
- Literais de intervalo. Para obter mais informações, consulte Literais de intervalo.
- Literais booleanos. Para obter mais informações, consulte Literais booleanos.
- Literais nulos, usados para especificar um valor nulo.
- Somente TAB, CARRIAGE RETURN (CR) e LINE FEED (LF) Caracteres de controle Unicode da categoria geral Unicode (Cc) são suportados.

AWS Clean Rooms O Spark SQL não suporta referências diretas a literais de string na cláusula SELECT, mas elas podem ser usadas em funções como CAST.

Operador + (Concatenação)

Concatena literais numéricos, literais de sequência de caracteres e/ou literais de data e hora e intervalo. Eles estão em ambos os lados do símbolo + e retornam tipos diferentes com base nas entradas em cada lado do símbolo +.

Sintaxe

```
numeric + string
```

```
date + time
```

```
date + timetz
```

A ordem dos argumentos pode ser invertida.

Argumentos

numeric literals

Literais ou constantes que representam números podem ser números inteiros ou de ponto flutuante.

string literals

Cadeias de caracteres, cadeias de caracteres ou constantes de caracteres

date

A DATE coluna ou uma expressão que se converte implicitamente em um DATE.

time

A TIME coluna ou uma expressão que se converte implicitamente em um TIME.

timetz

A TIMETZ coluna ou uma expressão que se converte implicitamente em um TIMETZ.

Exemplo

A tabela de exemplo a seguir TIME_TEST tem uma coluna TIME_VAL (tipo TIME) com três valores inseridos.

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

Tipos de dados

Cada valor que o AWS Clean Rooms Spark SQL armazena ou recupera tem um tipo de dados com um conjunto fixo de propriedades associadas. Os tipos de dados são declarados quando as tabelas são criadas. Um tipo de dado restringe um conjunto de valores que uma coluna ou argumento pode conter.

A tabela a seguir lista os tipos de dados que você pode usar no AWS Clean Rooms Spark SQL.

Nome do tipo de dados	Tipo de dados	Aliases	Description
ARRAY	the section called “Tipo aninhados”	Não aplicável	Tipo de dados de matriz aninhados
BIGINT	the section called “Tipos numéricos”	Não aplicável	Número inteiro de oito bytes assinado
BINARY	the section called “Tipo binário”	Não aplicável	Valores de sequência de bytes
BOOLEAN	the section called “Tipo booliano”	BOOL	Booleanos lógicos (verdadeiro/falso)
BYTE	the section called “Tipos numéricos”	Não aplicável	Números inteiros assinados de 1 byte, de -128 a 127
CHAR	the section called “Tipos de caracteres”	CHARACTER	String de caracteres com comprimento fixo

Nome do tipo de dados	Tipo de dados	Aliases	Description
DATE	the section called “Tipos de datetime”	Não aplicável	Data de calendário (ano, mês, dia)
DECIMAL	the section called “Tipos numéricos”	NUMERIC	Numérico exato com precisão selecionável
FLOAT	the section called “Tipos numéricos”	FLOAT8, PRECISÃO DUPLA	Número de ponto flutuante de precisão dupla
INTEGER	the section called “Tipos numéricos”	INT	Número inteiro de quatro bytes assinado
INTERVAL	the section called “Tipos de datetime”	Não aplicável	Duração do tempo em ordem diária ou ordem de ano a mês
LONG	the section called “Tipos numéricos”	Não aplicável	Números inteiros assinados de 8 bytes
MAP	the section called “Tipo aninhados”	Não aplicável	Tipo de dados de mapa aninhados
REAL	the section called “Tipos numéricos”	FLOAT4	Número de ponto flutuante de precisão simples
SHORT	the section called “Tipos numéricos”	Não aplicável	Números inteiros assinados de 2 bytes.
SMALLINT	the section called “Tipos numéricos”	Não aplicável	Número inteiro de dois bytes assinado
STRUCT	the section called “Tipo aninhados”	Não aplicável	Tipo de dados de estrutura aninhados

Nome do tipo de dados	Tipo de dados	Aliases	Description
TIMESTAMP_LTZ	the section called “Tipos de datetime”	Não aplicável	Hora do dia com fuso horário local
TIMESTAMP_NTZ	the section called “Tipos de datetime”	Não aplicável	Hora do dia sem fuso horário
TINYINT	the section called “Tipos numéricos”	Não aplicável	Números inteiros assinados de 1 byte, de -128 a 127
VARCHAR	the section called “Tipos de caracteres”	CARACTERE VARIÁVEL	String de caracteres de comprimento variável com limite definido pelo usuário

> **Note**
>
> Atualmente, os tipos de dados aninhados ARRAY, STRUCT e MAP só estão habilitados para a regra de análise personalizada. Para obter mais informações, consulte Tipo aninhados.

Caracteres multibyte

O tipo de dados VARCHAR é compatível com caracteres UTF-8 multibyte até um máximo de quatro bytes. Caracteres de cinco ou mais bytes são incompatíveis. Para calcular o tamanho de uma coluna VARCHAR que contenha caracteres multibyte, multiplique o número de caracteres pelo número de bytes por caractere. Por exemplo, se uma string contém quatro caracteres chineses e cada caractere possui três bytes de comprimento, você precisará de uma coluna VARCHAR(12) para armazenar a string.

O VARCHAR não é compatível com os seguintes pontos de código de caracteres UTF-8 inválidos:

`0xD800 – 0xDFFF` (Sequências de byte: `ED A0 80` – `ED BF BF`)

O tipo de dados CHAR não é compatível com caracteres multibyte.

Tipos numéricos

Os tipos de dados numéricos incluem números inteiros, decimais e números de ponto flutuante.

Tópicos

Tipos de inteiros

Use os seguintes tipos de dados para armazenar números inteiros de vários intervalos. Não é possível armazenar valores fora do intervalo permitido para cada tipo.

Nome	Armazenamento	Intervalo
SMALLINT	2 bytes	-32768 a +32767
SHORT	2 bytes	-32768 a +32767
INTEGER ou INT	4 bytes	-2147483648 a +2147483647
BIGINT	8 bytes	-9223372036854775808 a 9223372036854775807
LONG	8 bytes	-9223372036854775808 a 9223372036854775807

Tipo DECIMAL ou NUMERIC

Use o tipo de dados DECIMAL ou NUMERIC para armazenar valores com uma precisão definida pelo usuário. As palavras-chave DECIMAL e NUMERIC são intercambiáveis. Neste documento,

decimal é o termo preferido para esse tipo de dados. O termo numeric é usado genericamente para se referir aos tipos de dados de número inteiro, decimal e de ponto flutuante.

Armazenamento	Intervalo
Variável, até 128 bits para tipos DECIMAL não compactados.	Números inteiros assinados de 128 bits com até 38 dígitos de precisão.

Defina uma coluna DECIMAL em uma tabela especificando um *precision* e: *scale*

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

precision

O número total de dígitos significativos no valor inteiro: o número de dígitos em ambos os lados do ponto decimal. Por exemplo, o número `48.2891` tem precisão de 6 e uma escala de 4. A precisão padrão, quando não especificada, é 18. A precisão máxima é 38.

Se o número de dígitos à esquerda da vírgula decimal em um valor de entrada exceder a precisão da coluna menos sua escala, o valor não poderá ser copiado na coluna (ou inserido ou atualizado). Esta regra aplica-se a qualquer valor que caia fora do intervalo de definição da coluna. Por exemplo, o intervalo permitido de valores para uma coluna `numeric(5,2)` é `-999.99` a `999.99`.

scale

O número de dígitos decimais na parte fracionada do valor, à direita do ponto decimal. Números inteiros têm uma escala de zero. Em uma especificação de coluna, o valor de escala deve ser menor ou igual ao valor de precisão. A escala padrão, quando não especificada, é 0. A escala máxima é 37.

Se a escala de um valor de entrada carregado em uma tabela for maior do que a escala da coluna, o valor será arredondado para a escala especificada. Por exemplo, a coluna PRICEPAID na tabela SALES é uma coluna DECIMAL(8,2). Se um valor DECIMAL(8,4) é inserido na coluna PRICEPAID, o valor é arredondado para uma 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)
```

Contudo, os resultados de conversões explícitas dos valores selecionados a partir de tabelas não são arredondados.

> **Note**
>
> O valor positivo máximo que você pode inserir na coluna DECIMAL(19,0) é 9223372036854775807 (2^{63} -1). O valor negativo máximo é -9223372036854775807. Por exemplo, uma tentativa de inserir o valor 9999999999999999999 (19 noves) causará um erro de transbordamento. Independentemente do posicionamento do ponto decimal, a maior string que o AWS Clean Rooms pode representar como um número DECIMAL é 9223372036854775807. Por exemplo, o maior valor que você pode carregar em uma coluna DECIMAL(19,18) é 9.223372036854775807.
> Essas regras ocorrem por causa do seguinte:
>
> - Valores DECIMAL com 19 ou menos dígitos significativos de precisão são armazenados internamente como valores inteiros de 8 bytes.
> - Valores DECIMAL com 20 a 38 dígitos significativos de precisão são armazenados como valores inteiros de 16 bytes.

Observações sobre o uso de colunas do tipo DECIMAL ou NUMERIC de 128 bits

Não designe arbitrariamente a precisão máxima às colunas do tipo DECIMAL, a não ser que você esteja certo de que sua aplicação requer essa precisão. Valores de 128 bits usam duas vezes mais espaço em disco do que valores de 64 bits e podem retardar o tempo de execução da consulta.

Tipos de ponto flutuante

Use os tipos de dados REAL e DOUBLE PRECISION para armazenar valores numéricos com precisão variável. Esses tipos são inexatos, o que significa que alguns valores são armazenados como aproximações, de tal forma que o armazenamento e retorno de um valor específico pode

resultar em ligeiras discrepâncias. Se você precisar de armazenamento e cálculos precisos (como para quantidades monetárias), use o tipo de dados DECIMAL.

REAL representa o formato de ponto flutuante de precisão simples, de acordo com o padrão IEEE 754 para aritmética de ponto flutuante. Tem uma precisão de cerca de 6 dígitos e um intervalo de cerca de 1E-37 a 1E+37. Você também pode especificar esse tipo de dados como FLOAT4.

DOUBLE PRECISION representa o formato de ponto flutuante de precisão dupla, de acordo com o padrão 754 do IEEE para aritmética de ponto flutuante binário. Tem uma precisão de cerca de 15 dígitos e um intervalo de cerca de 1E-307 a 1E+308. Você também pode especificar esse tipo de dados como FLOAT ou FLOAT8.

Computações com valores numéricos

EmAWS Clean Rooms, computação se refere a operações matemáticas binárias: adição, subtração, multiplicação e divisão. Esta seção descreve os tipos de retorno previstos para essas operações, assim como a fórmula específica que é aplicada para determinar a precisão e escala quando dados do tipo DECIMAL estão envolvidos.

Quando os valores numéricos são computados durante o processamento da consulta, você pode encontrar casos onde o cálculo é impossível e a consulta retorna um erro de transbordamento numérico. Você também pode encontrar casos em que a escala de valores computados varia ou é inesperada. Para algumas operações, você pode usar a conversão explícita (promoção do tipo) ou parâmetros de configuração do AWS Clean Rooms para contornar esses problemas.

Para obter informações sobre os resultados de computações semelhantes com funções SQL, consulte AWS Clean Rooms Funções do Spark SQL.

Tipos de retorno para computações

Dado o conjunto de tipos de dados numéricos suportados emAWS Clean Rooms, a tabela a seguir mostra os tipos de retorno esperados para operações de adição, subtração, multiplicação e divisão. A primeira coluna do lado esquerdo da tabela representa o primeiro operando no cálculo e a linha superior representa o segundo operando.

Operando 1	Operando 2	Tipo de retorno
SMALLINT ou SHORT	SMALLINT ou SHORT	SMALLINT ou SHORT
SMALLINT ou SHORT	INTEGER	INTEGER

Operando 1	Operando 2	Tipo de retorno
SMALLINT ou SHORT	BIGINT	BIGINT
SMALLINT ou SHORT	DECIMAL	DECIMAL
SMALLINT ou SHORT	FLOAT4	FLOAT8
SMALLINT ou SHORT	FLOAT8	FLOAT8
INTEGER	INTEGER	INTEGER
INTEGER	BIGINT ou LONG	BIGINT ou LONG
INTEGER	DECIMAL	DECIMAL
INTEGER	FLOAT4	FLOAT8
INTEGER	FLOAT8	FLOAT8
BIGINT ou LONG	BIGINT ou LONG	BIGINT ou LONG
BIGINT ou LONG	DECIMAL	DECIMAL
BIGINT ou LONG	FLOAT4	FLOAT8
BIGINT ou LONG	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL
DECIMAL	FLOAT4	FLOAT8
DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8
FLOAT8	FLOAT8	FLOAT8

Precisão e escala de resultados de DECIMAL computados

A tabela a seguir resume as regras para precisão e escala resultantes de computação quando operações matemáticas retornam resultados DECIMAIS. Nesta tabela, `p1` e `s1` represente a precisão e a escala do primeiro operando em um cálculo. `p2`e `s2` representam a precisão e a escala do segundo operando. (Independentemente desses cálculos, a precisão máxima de resultado é 38 e a escala máxima de resultado é 38.)

Operation	Precisão e escala de resultados
+ ou -	Dimensionar = `max(s1,s2)` Precisão = `max(p1-s1,p2-s2)+1+scale`
*	Dimensionar = `s1+s2` Precisão = `p1+p2+1`
/	Dimensionar = `max(4,s1+p2-s2+1)` Precisão = `p1-s1+ s2+scale`

Por exemplo, as colunas PRICEPAID e COMMISSION na tabela SALES são ambas colunas do tipo DECIMAL(8,2). Se você dividir PRICEPAID pela COMMISSION (ou vice-versa), a fórmula será aplicada da seguinte forma:

```
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)
```

O seguinte cálculo é a regra geral para computação da precisão e escala resultantes para operações executadas em valores DECIMAIS com operadores de conjunto tais como UNION, INTERSECT e EXCEPT ou funções como COALESCE e DECODE:

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

Por exemplo, uma DEC1 tabela com uma coluna DECIMAL (7,2) é unida a uma DEC2 tabela com uma coluna DECIMAL (15,3) para criar uma tabela. DEC3 O esquema de DEC3 mostra que a coluna se torna uma coluna NUMÉRICA (15,3).

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

No exemplo acima, a fórmula é aplicada da seguinte forma:

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

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

Observações sobre operações de divisão

Para operações de divisão, divide-by-zero as condições retornam erros.

O limite de escala de 100 é aplicado após o cálculo da precisão e escala. Se a escala de resultados calculada for maior que 100, os resultados da divisão serão escalados da seguinte forma:

- Precisão = `precision - (scale - max_scale)`
- Dimensionar = `max_scale`

Se a precisão calculada for maior do que a precisão máxima (38), a precisão será reduzida para 38 e a escala será o resultado de: `max(38 + scale - precision), min(4, 100))`

Condições de transbordamento

O transbordamento é verificado para todas as computações numéricas. Dados DECIMAIS com uma precisão de 19 ou o menos são armazenados como números inteiros de 64 bits. Dados DECIMAIS com uma precisão maior que 19 são armazenados como números inteiros de 128 bits. A precisão máxima para todos os valores DECIMAIS é 38 e a escala máxima é 37. Erros de transbordamento ocorrem quando um valor excede esses limites, que se aplicam a conjuntos de resultados intermediário e final:

- A conversão explícita resulta em erros de estouro de tempo de execução quando valores de dados específicos não se ajustam à precisão ou escala solicitada especificada pela função de conversão.

Por exemplo, você não pode converter todos os valores da coluna PRICEPAID na tabela SALES (uma coluna DECIMAL(8,2)) e retornar um resultado DECIMAL(7,3):

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

Este erro ocorre porque alguns dos valores maiores na coluna PRICEPAID não podem ser convertidos.

- As operações de multiplicação produzem resultados em que a escala de resultados é a soma de escala de cada operando. Se ambos os operandos têm uma escala de 4, por exemplo, a escala de resultados é 8, deixando apenas 10 dígitos para o lado esquerdo do ponto decimal. Portanto, é relativamente fácil se deparar com condições de transbordamento ao multiplicar dois números grandes que possuem uma escala significativa.

Cálculos numéricos com os tipos INTEGER e DECIMAL

Quando um dos operandos em um cálculo tem um tipo de dados INTEGER e o outro operando é DECIMAL, o operando INTEGER é convertido implicitamente como DECIMAL.

- SMALLINT ou SHORT é convertido como DECIMAL (5,0)
- INTEGER é convertido como DECIMAL(10,0)
- BIGINT ou LONG é convertido como DECIMAL (19,0)

Por exemplo, se você multiplicar SALES.COMMISSION, uma coluna DECIMAL(8,2), e SALES.QTYSOLD, uma coluna SMALLINT, este cálculo será convertido como:

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

Tipos de caracteres

Os tipos de dados de caracteres incluem CHAR (caractere) e VARCHAR (caractere variável).

Tópicos

- CHAR ou CHARACTER
- VARCHAR ou CHARACTER VARYING
- Significância de espaços em branco

CHAR ou CHARACTER

Use uma coluna CHAR OU CHARACTER para armazenar strings de comprimento fixo. Essas strings são protegidas com espaços, portanto uma coluna CHAR(10) sempre ocupa 10 bytes de armazenamento.

```
char(10)
```

Uma coluna CHAR sem a especificação de comprimento resulta em uma coluna CHAR(1).

Os tipos de dados CHAR e VARCHAR são definidos em termos de bytes, não caracteres. Uma coluna CHAR pode ter somente caracteres de byte único, portanto a coluna CHAR(10) pode conter uma string com um comprimento máximo de 10 bytes.

Nome	Armazenamento	Intervalo (largura da coluna)
CHAR ou CHARACTER	Comprimento da string, incluindo espaços em branco (se houver)	4.096 bytes

VARCHAR ou CHARACTER VARYING

Use uma coluna VARCHAR ou CHARACTER VARYING para armazenar strings de tamanho variável com um limite fixo. Essas strings não são protegidas com espaços, portanto uma coluna VARCHAR(120) consistem em um máximo de 120 caracteres de único byte, 60 caracteres de dois bytes, 40 caracteres de três bytes ou 30 caracteres de quatro bytes.

```
varchar(120)
```

Os tipos de dados VARCHAR são definidos em termos de bytes, não de caracteres. Uma coluna VARCHAR pode conter caracteres de multibyte, até o máximo de quatro bytes por caractere. Por exemplo, uma coluna VARCHAR(12) pode conter 12 caracteres de único byte, 6 caracteres de dois bytes, 4 caracteres de três bytes ou 3 caracteres de quatro bytes.

Nome	Armazenamento	Intervalo (largura da coluna)
VARCHAR ou CHARACTER VARYING	4 bytes + total de bytes dos caracteres, onde cada caractere pode ser de 1 a 4 bytes.	65.535 bytes (64K -1)

Significância de espaços em branco

Tanto os tipos de dados CHAR quanto VARCHAR armazenam strings de até n bytes de comprimento. Uma tentativa de armazenar uma string mais longa em uma coluna desses tipos resulta em um erro. No entanto, se os caracteres extras forem todos espaços (espaços em branco), a string será truncada até o tamanho máximo. Se a string é mais curta do que o comprimento máximo, os valores CHAR são protegidos por espaços, mas valores CHAR armazenam a string sem os espaços.

Espaços em branco em valores CHAR são sempre semanticamente insignificantes. Eles são ignorados quando você compara dois valores CHAR, não são incluídos em cálculos de COMPRIMENTO e são removidos quando você converte um valor CHAR para outro tipo de string.

Os espaços em branco em valores VARCHAR e de CHAR são tratados como semanticamente insignificantes quando os valores são comparados.

Os cálculos de comprimento retornam o comprimento de strings de caracteres VARCHAR com espaços em branco incluídos no comprimento. Os espaços em branco não são contados no comprimento para strings de caracteres de comprimento fixo.

Tipos de datetime

Os tipos de dados de data e hora incluem DATE, TIME, TIMESTAMP_LTZ e TIMESTAMP_NTZ.

Tópicos

- Exemplos com tipos de datetime
- Literais de data, hora e timestamp
- Literais de intervalo
- Tipos de dados e literais de intervalo

DATE

Use o tipo de dados DATE para armazenar datas de calendário simples sem timestamps.

Nome	Armazenamento	Intervalo	Resolução
DATE	4 bytes	4713 AC a 294276 DC	1 dia

TIMESTAMP_LTZ

Use o tipo de dados TIMESTAMP_LTZ para armazenar valores completos de timestamp que incluem a data, a hora do dia e o fuso horário local.

TIMESTAMP representa valores que compreendem valores de campos`year`,,,`month`,`day`, e `hour` `minutesecond`, com o fuso horário local da sessão. O `timestamp` valor representa um ponto absoluto no tempo.

TIMESTAMP no Spark é um alias especificado pelo usuário associado a uma das variações TIMESTAMP_LTZ e TIMESTAMP_NTZ. Você pode definir o tipo de timestamp padrão como TIMESTAMP_LTZ (valor padrão) ou TIMESTAMP_NTZ por meio da configuração. `spark.sql.timestampType`

TIMESTAMP_NTZ

Use o tipo de dados TIMESTAMP_NTZ para armazenar valores completos de timestamp que incluem a data, a hora do dia, sem o fuso horário local.

TIMESTAMP representa valores que compreendem valores de campos`year`,`month`,, `dayhour`, e. `minute` `second` Todas as operações são realizadas sem levar em conta nenhum fuso horário.

TIMESTAMP no Spark é um alias especificado pelo usuário associado a uma das variações TIMESTAMP_LTZ e TIMESTAMP_NTZ. Você pode definir o tipo de timestamp padrão

como TIMESTAMP_LTZ (valor padrão) ou TIMESTAMP_NTZ por meio da configuração. `spark.sql.timestampType`

Exemplos com tipos de datetime

Os exemplos a seguir mostram como utilizar tipos de data e hora que são suportados pelo AWS Clean Rooms.

Exemplos de data

Os exemplos a seguir inserem datas que possuem formatos diferentes e exibem a saída.

```
select * from datetable order by 1;

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

Se você inserir um valor de timestamp em uma coluna DATE, a parte da hora será ignorada e apenas a data será carregada.

Exemplos de tempo

Os exemplos a seguir inserem os valores TIME e TIMETZ com formatos diferentes e exibem a saída.

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

Literais de data, hora e timestamp

A seguir estão as regras para trabalhar com literais de data, hora e timestamp compatíveis com o Spark SQL. AWS Clean Rooms

Datas

A tabela a seguir mostra datas de entrada que são exemplos válidos de valores de datas literais que você pode carregar em AWS Clean Rooms tabelas. O modo `MDY DateStyle` é considerado

em vigor. Este modo significa que o valor do mês precede o valor do dia em strings tais como `1999-01-08` e `01/02/00`.

> **Note**
>
> Um literal de data ou timestamp deve ser colocado entre aspas ao carregá-lo em uma tabela.

Data de entrada	Data completa
8 de janeiro de 1999	8 de janeiro de 1999
1999-01-08	8 de janeiro de 1999
1/8/1999	8 de janeiro de 1999
01/02/00	2 de janeiro de 2000
2000-Jan-31	31 de janeiro de 2000
Jan-31-2000	31 de janeiro de 2000
31-Jan-2000	31 de janeiro de 2000
20080215	15 de fevereiro de 2008
080215	15 de fevereiro de 2008
2008.366	31 de dezembro de 2008 (a parte de três dígitos da data deve estar entre 001 e 366)

Times

A tabela a seguir mostra os tempos de entrada que são exemplos válidos de valores de tempo literais que você pode carregar nas AWS Clean Rooms tabelas.

Tempos de entrada	Descrição (da parte da hora)
04:05:06.789	4:05 e 6,789 segundos

Tempos de entrada	Descrição (da parte da hora)
04:05:06	4:05 e 6 segundos
04:05	Exatamente 4:05
040506	4:05 e 6 segundos
04:05	Exatamente 4:05; AM é opcional
04:05	Exatamente 4:05; o valor de hora deve ser menor do que 12.
16:05	Exatamente 16:05

Valores especiais de datetime

A tabela a seguir mostra valores especiais que podem ser usados como literais de data e hora e como argumentos para funções de data. Eles exigem aspas simples e são convertidos em valores de timestamp regulares durante o processamento da consulta.

Valor especial	Description
`now`	Avalia para a hora de início da transação e retorna um timestamp com precisão de microssegundo.
`today`	Avalia para a data apropriada e retorna um timestamp com zeros para as partes do tempo.
`tomorrow`	Avalia para a data apropriada e retorna um timestamp com zeros para as partes do tempo.
`yesterday`	Avalia para a data apropriada e retorna um timestamp com zeros para as partes do tempo.

Os exemplos a seguir mostram como `now` e `today` funcionam com a função DATE_ADD.

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

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

select date_add('now', 1);

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

Literais de intervalo

A seguir estão as regras para trabalhar com literais de intervalo compatíveis com o AWS Clean Rooms Spark SQL.

Use um literal de intervalo para identificar períodos de tempo específicos, tais como `12 hours` ou `6 weeks`. Você pode usar esses literais de intervalo em condições e cálculos que envolvem expressões de data e hora.

> **Note**
>
> Você não pode usar o tipo de dados INTERVAL para colunas em AWS Clean Rooms tabelas.

Um intervalo é expressado como uma combinação da palavra-chave de INTERVAL com uma quantidade numérica e uma parte da data compatível; por exemplo `INTERVAL '7 days'` ou `INTERVAL '59 minutes'`. Você pode conectar várias quantidades e unidades para formar um intervalo mais preciso, por exemplo: `INTERVAL '7 days, 3 hours, 59 minutes'`. As abreviaturas e os plurais de cada unidade também são compatíveis; por exemplo: `5 s`, `5 second` e `5 seconds` são intervalos equivalentes.

Se você não especificar uma parte da data, o valor do intervalo representará os segundos. Você pode especificar o valor de quantidade como uma fração (por exemplo: `0.5 days`).

Exemplos

Os exemplos a seguir mostram uma série de cálculos com diferentes valores de intervalo.

O exemplo a seguir adiciona 1 segundo à data especificada.

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

O exemplo a seguir adiciona 1 minuto à data especificada.

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

O exemplo a seguir adiciona 3 horas e 35 minutos à data 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)
```

O exemplo a seguir adiciona 52 semanas à data especificada

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

O exemplo a seguir adiciona 1 semana, 1 hora, 1 minuto e 1 segundo à data 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)
```

O exemplo a seguir adiciona 12 horas (meio dia) à data 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)
```

O exemplo a seguir subtrai 4 meses de 31 de março de 2023 e o resultado é 30 de novembro de 2022. O cálculo considera o número de dias em um mês.

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

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

Tipos de dados e literais de intervalo

Você pode usar um tipo de dados de intervalo para armazenar espaços de tempo em unidades como `seconds`, `minutes`, `hours`, `days`, `months` e `years`. Tipos de dados e literais de intervalo podem ser usados em cálculos de data e hora, como adicionar intervalos a datas e carimbos de data e hora, somar intervalos e subtrair um intervalo de uma data ou carimbo de data e hora. Os literais de intervalo podem ser usados como valores de entrada para colunas de tipo de dados de intervalo em uma tabela.

Sintaxe do tipo de dados de intervalo

Para especificar um tipo de dados de intervalo para armazenar um espaço de tempo em anos e meses:

```
INTERVAL year_to_month_qualifier
```

Para especificar um tipo de dados de intervalo para armazenar uma duração em dias, horas, minutos e segundos:

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

Sintaxe de literal de intervalo

Para especificar um literal de intervalo para definir um espaço de tempo em anos e meses:

```
INTERVAL quoted-string year_to_month_qualifier
```

Para especificar um literal de intervalo para definir uma duração em dias, horas, minutos e segundos:

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

Argumentos

quoted-string

Determina um valor numérico positivo ou negativo especificando uma quantidade e a unidade de data e hora como uma string de entrada. Se a string entre aspas contiver somente um número, AWS Clean Rooms determinará as unidades do qualificador year_to_month_qualifier ou day_to_second_qualifier. Por exemplo, `'23' MONTH` representa `1 year 11 months`, `'-2' DAY` representa `-2 days 0 hours 0 minutes 0.0 seconds`, `'1-2' MONTH` representa `1 year 2 months` e `'13 day 1 hour 1 minute 1.123 seconds' SECOND` representa `13 days 1 hour 1 minute 1.123 seconds`. Para obter mais informações sobre formatos de saída de um intervalo, consulte Estilos de intervalo.

year_to_month_qualifier

Especifica a faixa do intervalo. Se você usar um qualificador e criar um intervalo com unidades de tempo menores que o qualificador, AWS Clean Rooms truncará e descartará as partes menores do intervalo. Os valores válidos para year_to_month_qualifier são:

- `YEAR`
- `MONTH`

- YEAR TO MONTH

day_to_second_qualifier

Especifica a faixa do intervalo. Se você usar um qualificador e criar um intervalo com unidades de tempo menores que o qualificador, AWS Clean Rooms truncará e descartará as partes menores do intervalo. Os valores válidos para year_to_month_qualifier são:

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

A saída do literal INTERVAL é truncada no menor componente INTERVAL especificado. Por exemplo, ao usar um qualificador MINUTE, AWS Clean Rooms descarta as unidades de tempo menores que MINUTE.

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

O valor resultante é truncado para `'1 day 01:01:00'`.

fractional_precision

Parâmetro opcional que especifica o número de dígitos fracionários permitidos no intervalo. O argumento fractional_precision só deverá ser especificado se seu intervalo contiver SECOND. Por exemplo, `SECOND(3)` cria um intervalo que permite somente três dígitos fracionários, como 1.234 segundos. O número máximo de dígitos fracionários é seis.

A configuração da sessão `interval_forbid_composite_literals` determina se um erro é retornado quando um intervalo é especificado com as partes YEAR TO MONTH e DAY TO SECOND.

Operações aritméticas de intervalo

É possível usar valores de intervalo com outros valores de data e hora para realizar operações aritméticas. As tabelas a seguir descrevem as operações disponíveis e qual tipo de dados resulta de cada operação.

> **Note**
>
> As operações que podem produzir resultados `date` e `timestamp` o fazem com base na menor unidade de tempo incluída na equação. Por exemplo, quando você adiciona `interval` a `date`, o resultado é `date`, se for um intervalo de YEAR TO MONTH, e um carimbo de data e hora, se for um intervalo de DAY TO SECOND.

As operações em que o primeiro operando é um `interval` produzem os seguintes resultados para o segundo operando fornecido:

Operador	Data	Marca de data e hora	Intervalo	Numérico
-	N/D	N/D	Intervalo	N/D
+	Data	Data/carimbo de data e hora	Intervalo	N/D
*	N/D	N/D	N/D	Intervalo
/	N/D	N/D	N/D	Intervalo

As operações em que o primeiro operando é um `date` produzem os seguintes resultados para o segundo operando fornecido:

Operador	Data	Marca de data e hora	Intervalo	Numérico
-	Numérico	Intervalo	Data/carimbo de data e hora	Data

Operador	Data	Marca de data e hora	Intervalo	Numérico
+	N/D	N/D	N/D	N/D

As operações em que o primeiro operando é um `timestamp` produzem os seguintes resultados para o segundo operando fornecido:

Operador	Data	Marca de data e hora	Intervalo	Numérico
-	Numérico	Intervalo	Registro de data e hora	Registro de data e hora
+	N/D	N/D	N/D	N/D

Estilos de intervalo

- `postgres`: segue o estilo do PostgreSQL. Esse é o padrão.
- `postgres_verbose`: segue o estilo detalhado do PostgreSQL.
- `sql_standard`: segue o estilo de literais de intervalo padrão do SQL.

O comando a seguir define o estilo de intervalo como `sql_standard`.

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

formato de saída postgres

Veja abaixo o formato de saída para o estilo de intervalo `postgres`. Cada valor numérico pode ser negativo.

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

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

varchar
```

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

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

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

formato de saída postgres_verbose

A sintaxe postgres_verbose é semelhante à do postgres, mas as saídas do postgres_verbose também contêm a unidade de tempo.

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

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

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

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

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

formato de saída sql_standard

Os valores do intervalo de ano para mês são formatados da maneira a seguir. Especificar um sinal negativo antes do intervalo indica que o intervalo é um valor negativo e se aplica a todo o intervalo.

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

Os valores do intervalo de dia para segundo são formatados da maneira a seguir.

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

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

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

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

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

Exemplos do tipo de dados de intervalo

Os exemplos a seguir demonstram como usar tipos de dados INTERVAL com tabelas.

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


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

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

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

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

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

Exemplos de literais de intervalo

Os exemplos a seguir são executados com o estilo de intervalo definido como `postgres`.

O exemplo a seguir demonstra como criar um literal INTERVAL de um ano.

```
select INTERVAL '1' YEAR

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

Se você especificar um quoted-string que exceda o qualificador, as unidades de tempo restantes serão truncadas do intervalo. No exemplo a seguir, um intervalo de 13 meses se torna um ano e um mês, mas o mês restante é omitido devido ao qualificador YEAR.

```
select INTERVAL '13 months' YEAR

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

Se você usar um qualificador inferior à string de intervalo, as unidades restantes serão incluídas.

```
select INTERVAL '13 months' MONTH

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

Especificar uma precisão no intervalo trunca o número de dígitos fracionários até a precisão especificada.

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

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

Se você não especificar uma precisão, AWS Clean Rooms usa a precisão máxima de 6.

```
select INTERVAL '1.23456789' SECOND

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

O exemplo a seguir demonstra como criar um intervalo em faixas.

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

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

Os qualificadores ditam as unidades que você está especificando. Por exemplo, embora o exemplo a seguir use a mesma string entre aspas de '2:2' do exemplo anterior, AWS Clean Rooms reconhece que ele usa unidades de tempo diferentes por causa do qualificador.

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

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

Abreviações e plurais de cada unidade também são aceitos. Por exemplo, `5s`, `5 second` e `5 seconds` são intervalos equivalentes. As unidades aceitas são anos, meses, horas, minutos e segundos.

```
select INTERVAL '5s' SECOND

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

```
select INTERVAL '5 HOURS' HOUR

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

```
select INTERVAL '5 h' HOUR

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

Exemplos de literais de intervalo sem sintaxe de qualificador

> **Note**
>
> Os exemplos a seguir demonstram o uso de um literal de intervalo sem um qualificador `YEAR TO MONTH` ou `DAY TO SECOND`. Para obter informações sobre como usar o literal de intervalo recomendado com um qualificador, consulte Tipos de dados e literais de intervalo.

Use um literal de intervalo para identificar períodos de tempo específicos, tais como `12 hours` ou `6 months`. Você pode usar esses literais de intervalo em condições e cálculos que envolvem expressões de data e hora.

Um literal de intervalo é expresso como uma combinação da palavra-chave de INTERVAL com uma quantidade numérica e uma parte da data compatível; por exemplo `INTERVAL '7 days'` ou `INTERVAL '59 minutes'`. Você pode conectar várias quantidades e unidades para formar um intervalo mais preciso, por exemplo: `INTERVAL '7 days, 3 hours, 59 minutes'`. As abreviaturas e os plurais de cada unidade também são compatíveis; por exemplo: `5 s`, `5 second` e `5 seconds` são intervalos equivalentes.

Se você não especificar uma parte da data, o valor do intervalo representará os segundos. Você pode especificar o valor de quantidade como uma fração (por exemplo: `0.5 days`).

Os exemplos a seguir mostram uma série de cálculos com diferentes valores de intervalo.

O exemplo a seguir adiciona 1 segundo à data especificada.

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

O exemplo a seguir adiciona 1 minuto à data especificada.

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

O exemplo a seguir adiciona 3 horas e 35 minutos à data 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)
```

O exemplo a seguir adiciona 52 semanas à data especificada.

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

O seguinte adiciona 1 semana, 1 hora, 1 minuto e 1 segundo à data 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)
```

O exemplo a seguir adiciona 12 horas (meio dia) à data 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)
```

O exemplo a seguir subtrai quatro meses de 15 de fevereiro de 2023 e o resultado é 15 de outubro de 2022.

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

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

O exemplo a seguir subtrai quatro meses de 31 de março de 2023 e o resultado é 30 de novembro de 2022. O cálculo considera o número de dias em um mês.

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

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

Tipo booliano

Use o tipo de dados BOOLEAN para armazenar valores verdadeiros e falsos em uma coluna de único byte. A tabela a seguir descreve os três estados possíveis para um valor booleano e os valores de literal que resultam naquele estado. Independente da string de entrada, a coluna booleana armazena e fornece "t" para verdadeiro e "f" para falso.

Estado	Valores válidos de literal	Armazenamento
Verdadeiro	`TRUE 't' 'true' 'y' 'yes' '1'`	1 byte
Falso	`FALSE 'f' 'false' 'n' 'no' '0'`	1 byte

Estado	Valores válidos de literal	Armazenamento
Desconhecido	NULL	1 byte

É possível usar uma comparação IS para verificar um valor booleano somente como um predicado na cláusula WHERE. Não é possível usar a comparação IS com um valor booleano na lista SELECT.

Exemplos

Você pode usar uma coluna BOOLEAN para armazenar um estado "Ativo/Inativo" para cada cliente em uma tabela CUSTOMER.

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

Neste exemplo, a consulta a seguir seleciona usuários da tabela USERS que gostam de esportes, mas não gostam de 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)
```

O exemplo a seguir seleciona usuários da tabela USERS para os quais não se sabe se eles gostam de 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)
```

O exemplo a seguir retorna um erro porque ele usa uma comparação IS na lista SELECT.

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

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

O exemplo a seguir é bem-sucedido porque usa uma comparação igual (=) na lista SELECT em vez da comparação 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    |
```

Literais booleanos

As regras a seguir são para trabalhar com literais booleanos compatíveis com o AWS Clean Rooms Spark SQL.

Use um literal booleano para especificar um valor booleano, como ou. `TRUE` `FALSE`

Sintaxe

```
TRUE | FALSE
```

Exemplo

O exemplo a seguir mostra uma coluna com um valor especificado de`TRUE`.

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

Tipo binário

Use o tipo de dados BINARY para armazenar e gerenciar dados binários de tamanho fixo e não interpretados, fornecendo recursos eficientes de armazenamento e comparação para casos de uso específicos.

O tipo de dados BINARY armazena um número fixo de bytes, independentemente do tamanho real dos dados que estão sendo armazenados. O tamanho máximo geralmente é de 255 bytes.

BINARY é usado para armazenar dados binários brutos e não interpretados, como imagens, documentos ou outros tipos de arquivos. Os dados são armazenados exatamente como são fornecidos, sem qualquer codificação ou interpretação de caracteres. Os dados binários

armazenados em colunas BINARY são comparados e classificados byte-by-byte com base nos valores binários reais, em vez de qualquer codificação de caracteres ou regras de agrupamento.

O exemplo de consulta a seguir mostra a representação binária da string`"abc"`. Cada caractere na string é representado por seu código ASCII em formato hexadecimal: “a” é 0x61, “b” é 0x62 e “c” é 0x63. Quando combinados, esses valores hexadecimais formam a representação binária. `"616263"`

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

Tipo aninhados

AWS Clean Roomssuporta consultas envolvendo dados com tipos de dados aninhados, especificamente os tipos de coluna AWS Glue STRUCT, ARRAY e MAP. Somente a regra de análise personalizada oferece suporte a tipos de dados aninhados.

Notavelmente, tipos de dados aninhados não estão em conformidade com a estrutura rígida e tabular do modelo de dados relacional de bancos de dados SQL.

Os tipos de dados aninhados contêm tags que fazem referência a entidades distintas nos dados. Eles podem conter valores complexos, como matrizes, estruturas aninhadas e outras estruturas complexas associadas a formatos de serialização, como JSON. Os tipos de dados aninhados são compatíveis com até 1 MB de dados aninhados em um campo ou objeto aninhados.

Tópicos

- Tipo de MATRIZ
- Tipo de MAP
- Tipo STRUCT
- Exemplos de tipos de dados aninhados

Tipo de MATRIZ

Use o tipo ARRAY para representar valores que compreendem uma sequência de elementos com o tipo de`elementType`.

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

Use `containsNull` para indicar se os elementos em um tipo ARRAY podem ter `null` valores.

Tipo de MAP

Use o tipo MAP para representar valores que compreendem um conjunto de pares de valores-chave.

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

`keyType`: o tipo de dados das chaves

`valueType`: o tipo de dados dos valores

Não é permitido que as chaves tenham `null` valores. Use `valueContainsNull` para indicar se os valores de um valor do tipo MAP podem ter `null` valores.

Tipo STRUCT

Use o tipo STRUCT para representar valores com a estrutura descrita por uma sequência de StructFields (campos).

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

StructField(nome, tipo de dados, anulável): representa um campo em a. StructType

`dataType`: os dados digitam um campo

`name`: o nome de um campo

Use `nullable` para indicar se os valores desses campos podem ter `null` valores.

Exemplos de tipos de dados aninhados

Para o tipo `struct<given:varchar, family:varchar>`, há dois nomes de atributos: `given` e `family`, cada um correspondendo a um valor `varchar`.

Para o tipo `array<varchar>`, a matriz é especificada como uma lista de `varchar`.

O tipo `array<struct<shipdate:timestamp, price:double>>` se refere a uma lista de elementos com tipo `struct<shipdate:timestamp, price:double>`.

O tipo de dados `map` se comporta como um `array` de `structs`, em que o nome do atributo de cada elemento na matriz é indicado por `key` e mapeado para um `value`.

Example

Por exemplo, o tipo `map<varchar(20), varchar(20)>` é tratado como `array<struct<key:varchar(20), value:varchar(20)>>`, onde `key` e `value` se referem aos atributos do mapa nos dados subjacentes.

Para obter informações sobre como AWS Clean Rooms habilitar a navegação em matrizes e estruturas, consulteNavegação.

Para obter informações sobre como AWS Clean Rooms habilitar a iteração em matrizes navegando na matriz usando a cláusula FROM de uma consulta, consulte. Desaninhar consultas

Compatibilidade e conversão dos tipos

Os tópicos a seguir descrevem como as regras de conversão de tipos e a compatibilidade de tipos de dados funcionam no AWS Clean Rooms Spark SQL.

Tópicos

- Compatibilidade
- Regras gerais de compatibilidade e conversão
- Tipos de conversão implícita

Compatibilidade

A correspondência de tipo de dados e de valores e constantes literais com tipos de dados ocorre durante diversas operações do banco de dados, incluindo o seguinte:

- Operações de linguagem de manipulação de dados (DML) em tabelas
- Consultas de UNION, INTERSECT e EXCEPT
- Expressões de CASOS
- Avaliação de predicados, tais como LIKE e IN
- Avaliação de funções SQL que fazem comparações ou extrações de dados
- Comparações com operadores matemáticos

Os resultados dessas operações dependem das regras de conversão de tipo e da compatibilidade dos tipos de dados. A compatibilidade implica que a one-to-one correspondência de um determinado valor e um determinado tipo de dados nem sempre é necessária. Como alguns tipos de dados são

compatíveis, uma conversão implícita, ou coerção, é possível. Para obter mais informações, consulte Tipos de conversão implícita. Quando os tipos de dados são incompatíveis, você pode às vezes converter um valor de um tipo de dados para outro usando uma função de conversão explícita.

Regras gerais de compatibilidade e conversão

Observe as seguintes regras de compatibilidade e conversão:

- Em geral, tipos de dados que se enquadram no mesmo tipo de categoria (como diferentes tipos de dados numéricos) são compatíveis e podem ser implicitamente convertidos.

  Por exemplo, com a conversão implícita você pode inserir um valor decimal em uma coluna de número inteiro. O decimal é arredondado para produzir um número inteiro. Ou você pode extrair um valor numérico, tal como `2008`, a partir de uma data e inserir este valor uma coluna de inteiro.
- Os tipos de dados numéricos impõem condições de estouro que ocorrem quando você tenta inserir valores. out-of-range Por exemplo, um valor decimal com uma precisão de 5 não se enquadra em uma coluna decimal que foi definida com uma precisão de 4. Um número inteiro ou a parte inteira de um decimal nunca é truncada. No entanto, a parte fracionária de um decimal pode ser arredondada para cima ou para baixo, conforme apropriado. Contudo, os resultados de conversões explícitas dos valores selecionados a partir de tabelas não são arredondados.
- Diferentes tipos de cadeias de caracteres são compatíveis. As sequências de colunas VARCHAR contendo dados de byte único e as sequências de colunas CHAR são comparáveis e implicitamente conversíveis. Strings VARCHAR que contêm dados de multibyte não são comparáveis. Além disso, você pode converter uma sequência de caracteres em uma data, hora, carimbo de data/hora ou valor numérico se a sequência for um valor literal apropriado. Todos os espaços à esquerda ou à direita são ignorados. Por outro lado, você pode converter uma data, hora, timestamp ou valor numérico em uma sequência de caracteres de comprimento fixo ou variável.

  > **Note**
  >
  > Uma string de caracteres que você queira converter em um tipo numérico deve conter uma representação de caractere de um número. Por exemplo, você pode converter as strings `'1.0'` ou `'5.9'` em valores decimais, mas não pode converter a string `'ABC'` em nenhum tipo numérico.
- Se você comparar valores DECIMAIS com cadeias de caracteres, AWS Clean Rooms tentará converter a cadeia de caracteres em um valor DECIMAL. Ao comparar todos os outros valores

numéricos com strings de caracteres, os valores numéricos são convertidos em strings de caracteres. Para impor a conversão oposta (por exemplo, converter strings de caracteres em números inteiros ou converter valores do tipo DECIMAL em strings de caracteres), use uma função explícita, como Função CAST.

- Para converter valores do tipo DECIMAL ou NUMERIC de 64 bits em uma precisão mais alta, você deve usar uma função de conversão explícita tal como CAST ou CONVERT.

Tipos de conversão implícita

Há dois tipos de conversão implícita:

- Conversões implícitas em atribuições, como definir valores em comandos INSERT ou UPDATE
- Conversões implícitas em expressões, como realizar comparações na cláusula WHERE

A tabela a seguir lista os tipos de dados que podem ser convertidos implicitamente em atribuições ou expressões. Você também pode usar uma função de conversão explícita para realizar essas conversões.

Do tipo	Para o tipo
BIGINT	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	PRECISÃO DUPLA (FLOAT8)
	INTEGER
	REAL (FLOAT4)
	SMALLINT ou SHORT
	VARCHAR
CHAR	VARCHAR
DATE	CHAR

Do tipo	Para o tipo
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT ou LONG
	CHAR
	PRECISÃO DUPLA (FLOAT8)
	INTEGER INT)
	REAL (FLOAT4)
	SMALLINT ou SHORT
	VARCHAR
PRECISÃO DUPLA (FLOAT8)	BIGINT ou LONG
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	REAL (FLOAT4)
	SMALLINT ou SHORT
	VARCHAR
INTEGER (INT)	BIGINT ou LONG
	BOOLEAN
	CHAR

Do tipo	Para o tipo
	DECIMAL (NUMERIC)
	PRECISÃO DUPLA (FLOAT8)
	REAL (FLOAT4)
	SMALLINT ou SHORT
	VARCHAR
REAL (FLOAT4)	BIGINT ou LONG
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	SMALLINT ou SHORT
	VARCHAR
SMALLINT	BIGINT ou LONG
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	PRECISÃO DUPLA (FLOAT8)
	INTEGER (INT)
	REAL (FLOAT4)
	VARCHAR
TIME	VARCHAR

Do tipo	Para o tipo
	TIMETZ

> **Note**
>
> As conversões implícitas entre DATE, TIME, TIMESTAMP_LTZ, TIMESTAMP_NTZ ou cadeias de caracteres usam o fuso horário da sessão atual.
> O tipo de dados VARBYTE não pode ser convertido explicitamente em outro tipo de dados. Para obter mais informações, consulte Função CAST.

AWS Clean Rooms Comandos do Spark SQL

Os seguintes comandos SQL são compatíveis com o AWS Clean Rooms Spark SQL:

Tópicos

- TABELA DE CACHE
- Dicas
- SELECT

TABELA DE CACHE

O comando CACHE TABLE armazena em cache os dados de uma tabela existente ou cria e armazena em cache uma nova tabela contendo os resultados da consulta.

> **Note**
>
> Os dados em cache persistem durante toda a consulta.

A sintaxe, os argumentos e alguns exemplos vêm da Referência SQL do Apache Spark.

Sintaxe

O comando CACHE TABLE suporta três padrões de sintaxe:

Com AS (sem parênteses): cria e armazena em cache uma nova tabela com base nos resultados da consulta.

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

Com AS e parênteses: funciona de forma semelhante à primeira sintaxe, mas usa parênteses para agrupar explicitamente a consulta.

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

Sem AS: armazena em cache uma tabela existente, usando a instrução SELECT para filtrar quais linhas devem ser armazenadas em cache.

```
CACHE TABLE cache_table_identifier query;
```

Em que:

- Todas as declarações devem terminar com ponto e vírgula (;)
- `query`normalmente é uma instrução SELECT
- Os parênteses ao redor da consulta são opcionais com AS
- A palavra-chave AS é opcional

Parâmetros

identificador_tabela_cache

O nome da tabela em cache. Pode incluir um qualificador de nome de banco de dados opcional.

AS

Uma palavra-chave usada ao criar e armazenar em cache uma nova tabela a partir dos resultados da consulta.

query

Uma instrução SELECT ou outra consulta que define os dados a serem armazenados em cache.

Exemplos

Nos exemplos a seguir, a tabela em cache persiste durante toda a consulta. Após o armazenamento em cache, as consultas subsequentes que fazem referência *cache_table_identifier* serão lidas da versão em cache em vez de serem recalculadas ou lidas. *sourceTable* Isso pode melhorar o desempenho da consulta para dados acessados com frequência.

Crie e armazene em cache uma tabela filtrada a partir dos resultados da consulta

O primeiro exemplo demonstra como criar e armazenar em cache uma nova tabela a partir dos resultados da consulta. Esse comando usa a AS palavra-chave sem parênteses ao redor da declaração. SELECT Ele cria uma nova tabela chamada 'cache_table_identifier' contendo somente as linhas de 'sourceTable' onde o status é 'active'. Ele executa a consulta, armazena os resultados na nova tabela e armazena em cache o conteúdo da nova tabela. O 'sourceTable' original permanece inalterado e as consultas subsequentes devem fazer referência a 'cache_table_identifier' para usar os dados em cache.

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

Resultados da consulta em cache com instruções SELECT entre parênteses

O segundo exemplo demonstra como armazenar em cache os resultados de uma consulta como uma nova tabela com um nome especificado (cache_table_identifier), usando parênteses ao redor da instrução. SELECT Esse comando cria uma nova tabela chamada 'cache_table_identifier' contendo somente as linhas de 'sourceTable' em que o status é 'active'. Ele executa a consulta, armazena os resultados na nova tabela e armazena em cache o conteúdo da nova tabela. O 'sourceTable' original permanece inalterado. As consultas subsequentes devem fazer referência a 'cache_table_identifier' para usar os dados em cache.

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

Armazene em cache uma tabela existente com condições de filtro

O terceiro exemplo demonstra como armazenar em cache uma tabela existente usando uma sintaxe diferente. Essa sintaxe, que omite a palavra-chave 'AS' e os parênteses, normalmente armazena em cache as linhas especificadas de uma tabela existente chamada 'cache_table_identifier' em vez de criar uma nova tabela. A SELECT instrução atua como um filtro para determinar quais linhas devem ser armazenadas em cache.

> **Note**
>
> O comportamento exato dessa sintaxe varia entre os sistemas de banco de dados. Sempre verifique a sintaxe correta para seu AWS serviço específico.

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

Dicas

As dicas para análises de SQL fornecem diretivas de otimização que orientam as estratégias de execução de consultas AWS Clean Rooms, permitindo que você melhore o desempenho da consulta e reduza os custos de computação. As dicas sugerem como o mecanismo de análise do Spark deve gerar seu plano de execução.

Sintaxe

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

As dicas são incorporadas às consultas SQL usando a sintaxe de estilo de comentário e devem ser colocadas diretamente após a palavra-chave SELECT.

Tipos de dicas compatíveis

AWS Clean Rooms suporta duas categorias de dicas: dicas de junção e dicas de particionamento.

Tópicos

- Junte dicas

- Dicas de particionamento

Junte dicas

As dicas de junção sugerem estratégias de junção para execução de consultas. A sintaxe, os argumentos e alguns exemplos vêm da Referência SQL do Apache Spark para obter mais informações

TRANSMISSÃO

Sugere que AWS Clean Rooms use broadcast join. O lado de junção com a dica será transmitido independentemente do autoBroadcastJoin limite. Se os dois lados da junção tiverem as dicas de transmissão, aquele com o tamanho menor (com base nas estatísticas) será transmitido.

Pseudônimos: BROADCASTJOIN, MAPJOIN

Parâmetros: identificadores de tabela (opcional)

Exemplos:

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

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

MERGE

Sugere que AWS Clean Rooms use shuffle sort merge join.

Pseudônimos: SHUFFLE_MERGE, MERGEJOIN

Parâmetros: identificadores de tabela (opcional)

Exemplos:

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

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

SHUFFLE_HASH

Sugere que AWS Clean Rooms use shuffle hash join. Se os dois lados tiverem dicas de hash aleatórias, o otimizador de consultas escolherá o lado menor (com base nas estatísticas) como o lado da construção.

Parâmetros: identificadores de tabela (opcional)

Exemplos:

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

SHUFFLE_REPLICATE_NL

Sugere o AWS Clean Rooms uso de junção de loop shuffle-and-replicate aninhado.

Parâmetros: identificadores de tabela (opcional)

Exemplos:

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

Dicas de solução de problemas no Spark SQL

A tabela a seguir mostra cenários comuns em que as dicas não são aplicadas no SparkSQL. Para obter informações adicionais, consulte the section called “Considerações e limitações”.

Caso de uso	Exemplo de consulta
Referência de tabela não encontrada	`SELECT /*+ BROADCAST(fake_table) */ *`

Caso de uso	Exemplo de consulta
	`FROM employees e` `INNER JOIN students s ON e.eid = s.sid;`

Mesa que não participa da operação de junção	`SELECT /*+ BROADCAST(s) */ *`
	`FROM students s`
	`WHERE s.age > 25;`

Referência de tabela na subconsulta aninhada	`SELECT /*+ BROADCAST(s) */ *`
	`FROM employees e`
	`INNER JOIN (SELECT * FROM students s WHERE s.age > 20)`
	` sub`
	`ON e.eid = sub.sid;`

Nome da coluna em vez da referência da tabela	`SELECT /*+ BROADCAST(e.eid) */ *`
	`FROM employees e`
	`INNER JOIN students s ON e.eid = s.sid;`

Dica sem parâmetros obrigatórios	`SELECT /*+ BROADCAST */ *`
	`FROM employees e`
	`INNER JOIN students s ON e.eid = s.sid;`

Nome da tabela base em vez do alias da tabela	`SELECT /*+ BROADCAST(employees) */ *`
	`FROM employees e`
	`INNER JOIN students s ON e.eid = s.sid;`

Dicas de particionamento

As dicas de particionamento controlam a distribuição de dados entre os nós do executor. Quando várias dicas de particionamento são especificadas, vários nós são inseridos no plano lógico, mas a dica mais à esquerda é selecionada pelo otimizador.

AGLUTINAR

Reduz o número de partições para o número especificado de partições.

Parâmetros: valor numérico (obrigatório) - deve ser um número inteiro positivo entre 1 e 2147483647

Exemplos:

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

DISTRIBUIÇÃO

Reparticiona os dados para o número especificado de partições usando as expressões de particionamento especificadas. Usa distribuição round-robin.

Parâmetros:

- Valor numérico (opcional) - número de partições; deve ser um número inteiro positivo entre 1 e 2147483647
- Identificadores de coluna (opcional) - colunas pelas quais particionar; Essas colunas devem existir no esquema de entrada.
- Se ambos forem especificados, o valor numérico deverá vir primeiro

Exemplos:

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

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

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

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

REPARTIÇÃO_POR_INTERVALO

Reparticiona os dados para o número especificado de partições usando o particionamento de intervalo nas colunas especificadas.

Parâmetros:

- Valor numérico (opcional) - número de partições; deve ser um número inteiro positivo entre 1 e 2147483647
- Identificadores de coluna (opcional) - colunas pelas quais particionar; Essas colunas devem existir no esquema de entrada.
- Se ambos forem especificados, o valor numérico deverá vir primeiro

Exemplos:

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

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

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

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

REEQUILIBRAR

Reequilibra as partições de saída do resultado da consulta para que cada partição tenha um tamanho razoável (nem muito pequena nem muito grande). Esta é uma operação de melhor esforço: se houver inclinações, AWS Clean Rooms dividirá as partições inclinadas para que não sejam muito grandes. Essa dica é útil quando você precisa gravar o resultado de uma consulta em uma tabela para evitar arquivos muito pequenos ou muito grandes.

Parâmetros:

- Valor numérico (opcional) - número de partições; deve ser um número inteiro positivo entre 1 e 2147483647
- Identificadores de coluna (opcional) - as colunas devem aparecer na lista de saída SELECT
- Se ambos forem especificados, o valor numérico deverá vir primeiro

Exemplos:

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

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

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

Combinando várias dicas

Você pode especificar várias dicas em uma única consulta separando-as com vírgulas:

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

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

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

Considerações e limitações

- As dicas são sugestões de otimização, não comandos. O otimizador de consultas pode ignorar dicas com base nas restrições de recursos ou nas condições de execução.
- As dicas são incorporadas diretamente nas cadeias de caracteres de consulta SQL para e. CreateAnalysisTemplate StartProtectedQuery APIs
- As dicas devem ser colocadas diretamente após a palavra-chave SELECT.
- Os parâmetros nomeados não são compatíveis com dicas e gerarão uma exceção.

- Os nomes das colunas nas dicas REPARTITION e REPARTITION_BY_RANGE devem existir no esquema de entrada.
- Os nomes das colunas nas dicas de REBALANCE devem aparecer na lista de saída SELECT.
- Os parâmetros numéricos devem ser números inteiros positivos entre 1 e 2147483647. Não há suporte para notações científicas como 1e1
- As dicas não são suportadas em consultas SQL de privacidade diferencial.
- As dicas para consultas SQL não são suportadas em PySpark trabalhos. Para fornecer diretrizes para planos de execução em um PySpark trabalho, use a API de data frame. Consulte a documentação da DataFrame API Apache Spark para obter mais informações.

SELECT

O comando SELECT retorna linhas de tabelas e funções definidas pelo usuário.

Os seguintes comandos, cláusulas e operadores de conjunto do SELECT SQL são compatíveis com o AWS Clean Rooms Spark SQL:

Tópicos

A sintaxe, os argumentos e alguns exemplos vêm da Referência SQL do Apache Spark.

SELECT list

Os nomes SELECT list das colunas, funções e expressões que você deseja que a consulta retorne. A lista representa o resultado da consulta.

Sintaxe

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

Parâmetros

DISTINCT

Opção que elimina linhas duplicadas do conjunto de resultados, com base em valores correspondentes em uma ou mais colunas.

expression

Expressão formada por uma ou mais colunas que existem em tabelas referidas pela consulta. Uma expressão pode conter funções SQL. Por exemplo:

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

AS column_alias

Nome temporário da coluna que é usada no conjunto de resultados finais. A palavra-chave AS é opcional. Por exemplo:

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

Se você não especificar um alias para uma expressão que não for um nome de coluna simples, o resultado definido aplicará um nome padrão à coluna.

> **Note**
>
> O alias é reconhecido logo após ser definido na lista de destino. Você não pode usar um alias em outras expressões definidas depois dele na mesma lista de destinos.

Cláusula WITH

Uma cláusula WITH é uma cláusula opcional que precede a lista SELECT em uma consulta. A cláusula WITH define um ou mais common_table_expressions. Cada expressão de tabela comum (CTE) define uma tabela temporária, que é semelhante à definição de visualização. Você pode fazer referência a essas tabelas temporárias na cláusula FROM. Eles são usados apenas enquanto a consulta a que pertencem é executada. Cada CTE na cláusula WITH especifica um nome de tabela, uma lista opcional de nomes de coluna e uma expressão de consulta que é avaliada como uma tabela (uma instrução SELECT).

Subconsultas da cláusula WITH são uma forma eficiente de definir tabelas que podem ser usadas ao longo da execução de uma consulta. Em todos os casos, os mesmos resultados podem ser obtidos usando subconsultas no corpo principal da instrução SELECT, mas pode ser mais simples fazer leituras ou gravações de subconsultas da cláusula WITH. Sempre que possível, subconsultas da cláusula WITH por várias vezes referidas são aperfeiçoadas como subexpressões comuns, ou seja, é possível avaliar uma subconsulta WITH uma vez e reutilizar seus resultados. (Observe que subexpressões comuns não estão limitadas àquelas definidas na cláusula WITH.)

Sintaxe

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

onde common_table_expression pode ser não recursivo. Segue-se a forma não recursiva:

```
CTE_table_name AS ( query )
```

Parâmetros

common_table_expression

Define uma tabela temporária que você pode fazer referência no Cláusula FROM e é usado somente durante a execução da consulta a qual pertence.

CTE_table_name

Um nome exclusivo para uma tabela temporária que define os resultados da subconsulta de cláusula WITH. Você não pode usar nomes duplicados em uma única cláusula WITH. Cada subconsulta deve ter um nome de tabela que pode mencionado em Cláusula FROM.

query

Qualquer consulta SELECT que AWS Clean Rooms ofereça suporte. Consulte SELECT.

Observações de uso

Você pode usar uma cláusula WITH na seguinte instrução SQL:

- SELECIONAR, COM, UNIR, UNIR TUDO, INTERSETAR, INTERSETAR TUDO, EXCETO OU EXCETO TUDO

Se a cláusula FROM de uma consulta que contém a cláusula WITH não fizer referência a qualquer das tabelas definidas pela cláusula WITH, a cláusula WITH será ignorada e a consulta será executada como normal.

Uma tabela definida por uma subconsulta de cláusula WITH somente pode ser referida no escopo da consulta SELECT iniciada pela cláusula WITH. Por exemplo, você pode fazer referência a essa tabela na cláusula FROM da subconsulta na lista SELECT, na cláusula WHERE ou na cláusula HAVING. Você não pode usar a cláusula WITH em uma subconsulta e fazer referência à sua tabela na cláusula FROM da consulta principal ou de outra subconsulta. Este padrão de consulta resulta em uma mensagem de erro do formulário `relation table_name doesn't exist` para a tabela da cláusula WITH.

Você não pode especificar outra cláusula WITH em uma subconsulta de cláusula WITH.

Você não pode fazer referência antecipada a tabelas definidas por subconsultas da cláusula WITH. Por exemplo, a consulta a seguir retorna um erro devido à referência antecipada para a tabela W2 na definição da tabela W1:

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

Exemplos

O exemplo a seguir mostra o caso mais simples possível de uma consulta que contém uma cláusula WITH. A consulta WITH com o nome VENUECOPY seleciona todas as linhas da tabela VENUE. Por sua vez, a consulta principal seleciona todas as linhas de VENUECOPY. A tabela VENUECOPY existe somente durante a 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)
```

O exemplo a seguir mostra uma cláusula WITH que produz duas tabelas, chamadas VENUE_SALES e TOP_VENUES. A segunda tabela de consulta WITH seleciona a partir da primeira. Por sua vez, a cláusula WHERE do bloco principal de consulta contém um subconsulta que restringe a tabela 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)
```

Os dois exemplos a seguir demonstram as regras para o escopo de referências de tabela com base subconsultas da cláusula WITH. A primeira consulta é executada, mas a segunda falha com um erro esperado. A primeira consulta tem a subconsulta de cláusula WITH na lista SELECT da consulta principal. A tabela definida pela cláusula WITH (HOLIDAYS) é referida na cláusula FROM da subconsulta na 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)
```

A segunda consulta falha porque tenta fazer referência à tabela HOLIDAYS na consulta principal, assim como na subconsulta da lista SELECT. As referências principais da consulta estão fora do escopo.

```
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

A cláusula FROM em uma consulta lista as referências de tabela (tabelas, exibições e subconsultas) de onde os dados são selecionados. Se as referências de várias tabelas estiverem listadas, as tabelas devem ser juntadas, usando a sintaxe apropriada na cláusula FROM ou WHERE. Se nenhum critério de junção for especificado, o sistema processará a consulta como uma junção cruzada (produto cartesiano).

Tópicos

- Sintaxe
- Parâmetros
- Observações de uso

Sintaxe

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

onde referência_tabela é uma das seguintes:

```
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
```

Parâmetros

com_subconsulta_nome_tabela

Tabela definida por uma subconsulta em Cláusula WITH.

table_name

Nome de uma tabela ou exibição.

alias

Nome alternativo temporário para uma tabela ou exibição. Um alias deve ser fornecido para uma tabela derivada de uma subconsulta. Em outras referências de tabela, os alias são opcionais. A palavra-chave AS é sempre opcional. Os alias de tabela oferecem um atalho conveniente para tabelas de identificação em outras partes de uma consulta, como a cláusula WHERE.

Por exemplo:

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

Se você definir um alias de tabela definido, o alias deverá ser usado para referenciar essa tabela na consulta.

Por exemplo, se a consulta for `SELECT "tbl"."col" FROM "tbl" AS "t"`, a consulta falhará porque o nome da tabela está basicamente substituído agora. Uma consulta válida nesse caso seria `SELECT "t"."col" FROM "tbl" AS "t"`.

alias_coluna

Nome alternativo temporário para uma coluna em uma tabela ou exibição.

subconsulta

Uma expressão de consulta que avalia para uma tabela. A tabela existe somente pela duração da consulta e geralmente recebe um nome ou alias. No entanto, um alias não é necessário. Você também pode definir nomes de colunas para tabelas que derivam de subconsultas. Nomear aliases de coluna é importante quando você deseja participar dos resultados de subconsultas a outras tabelas e quando você deseja selecionar ou restringir essas colunas em outro lugar da consulta.

Uma subconsulta pode conter uma cláusula ORDER BY, mas essa cláusula poderá não ter qualquer efeito se uma cláusula LIMIT ou OFFSET também não estiver especificada.

NATURAL

Define um junção que usa automaticamente todos os pares de colunas com nomes idênticos em duas tabelas como colunas de junção. Nenhuma condição explícita de junção é necessária. Por

exemplo, se as tabelas CATEGORY e EVENT apresentam colunas com nome CATID, um junção natural dessas tabelas é um junção pelas colunas CATID.

> **Note**
>
> Se uma junção NATURAL for especificada mas não existirem pares de colunas com o mesmo nome nas tabelas a serem juntadas, a junção padrão da consulta usada será a junção cruzada.

join_type

Especifique um dos seguintes tipos de junção:

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

As junções cruzadas são junções não qualificadas; elas retornam o produto cartesiano das duas tabelas.

As junções internas e externas são junções qualificadas. Elas podem ser qualificadas implicitamente (em junções naturais); com a sintaxe ON ou USING na cláusula FROM; ou com a condição de cláusula WHERE.

Uma junção interna retorna somente linhas correspondentes, com base na condição de junção ou na lista de colunas de junção. Uma junção externa retorna todas as linhas que a junção interna equivalente deve retornar e linhas não correspondentes da tabela "esquerda", da tabela "direita" ou de ambas. A tabela esquerda é a primeira tabela listada, e a tabela direita é a segunda tabela listada. As linhas não correspondentes contêm valores NULL para preencher lacunas entre as colunas resultantes.

ON condição_junção

Tipo de especificação de junção em que as colunas a serem juntadas são exibidas como uma condição que acompanha a palavra-chave ON. Por exemplo:

```
sales join listing
```

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

USING (coluna_junção [, ...])

Tipo de especificação de junção em que as colunas a serem juntadas estão listadas entre parênteses. Se várias colunas a serem juntadas forem especificadas, elas serão separadas por vírgulas. A palavra-chave USING deve preceder a lista. Por exemplo:

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

Observações de uso

Colunas de junção devem ter tipos de dados comparáveis.

Uma junção NATURAL ou USING retém somente um de cada par de colunas de junção no conjunto de resultados intermediário.

Uma junção com a sintaxe ON retém ambas as colunas de junção em seu conjunto de resultados intermediário.

Consulte também Cláusula WITH.

Cláusula JOIN

Uma cláusula SQL JOIN é usada para combinar os dados de duas ou mais tabelas com base em campos comuns. Os resultados podem ou não mudar dependendo do método de junção especificado. Junções externas esquerdas e direitas retêm valores de uma das tabelas de junção quando nenhuma correspondência é encontrada na outra tabela.

A combinação do tipo JOIN e da condição de junção determina quais linhas são incluídas no conjunto de resultados final. As cláusulas SELECT e WHERE então controlam quais colunas são retornadas e como as linhas são filtradas. Compreender os diferentes tipos de JOIN e como usá-los de forma eficaz é uma habilidade crucial em SQL, pois permite combinar dados de várias tabelas de forma flexível e poderosa.

Sintaxe

```
SELECT column1, column2, ..., columnn
FROM table1
join_type table2
```

```
ON table1.column = table2.column;
```

Parâmetros

SELECIONE coluna1, coluna2,..., coluna N

As colunas que você deseja incluir no conjunto de resultados. Você pode selecionar colunas de uma ou de ambas as tabelas envolvidas no JOIN.

DA tabela 1

A primeira tabela (esquerda) na operação JOIN.

[JUNÇÃO | JUNÇÃO INTERNA | JUNÇÃO ESQUERDA [EXTERNA] | JUNÇÃO DIREITA [EXTERNA] | JUNÇÃO COMPLETA [EXTERNA]] tabela 2:

O tipo de JOIN a ser executado. JOIN ou INNER JOIN retorna somente as linhas com valores correspondentes em ambas as tabelas.

LEFT [OUTER] JOIN retorna todas as linhas da tabela à esquerda, com as linhas correspondentes da tabela à direita.

RIGHT [OUTER] JOIN retorna todas as linhas da tabela à direita, com as linhas correspondentes da tabela esquerda.

FULL [OUTER] JOIN retorna todas as linhas das duas tabelas, independentemente de haver uma correspondência ou não.

CROSS JOIN cria um produto cartesiano das linhas das duas tabelas.

NA tabela1.coluna = tabela2.coluna

A condição de junção, que especifica como as linhas nas duas tabelas são correspondidas. A condição de junção pode ser baseada em uma ou mais colunas.

Condição WHERE:

Uma cláusula opcional que pode ser usada para filtrar ainda mais o conjunto de resultados, com base em uma condição especificada.

Exemplo

O exemplo a seguir é uma junção entre duas tabelas com a cláusula USING. Nesse caso, as colunas listid e eventid são usadas como colunas de junção. Os resultados são limitados a cinco linhas.

```
select listid, listing.sellerid, eventid, listing.dateid, numtickets
from listing join sales
using (listid, eventid)
order by 1
limit 5;

listid | sellerid | eventid | dateid | numtickets
-------+----------+---------+--------+-----------
1      | 36861    | 7872    | 1850   | 10
4      | 8117     | 4337    | 1970   | 8
5      | 1616     | 8647    | 1963   | 4
5      | 1616     | 8647    | 1963   | 4
6      | 47402    | 8240    | 2053   | 18
```

Tipos de união

INNER

Esse é o tipo de junção padrão. Retorna as linhas que têm valores correspondentes nas duas referências da tabela.

O INNER JOIN é o tipo mais comum de junção usado em SQL. É uma forma poderosa de combinar dados de várias tabelas com base em uma coluna comum ou conjunto de colunas.

Sintaxe:

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

A consulta a seguir retornará todas as linhas em que há um valor de customer_id correspondente entre as tabelas de clientes e pedidos. O conjunto de resultados conterá as colunas customer_id, name, order_id e order_date.

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

A consulta a seguir é uma junção interna (sem a palavra-chave JOIN) entre a tabela LISTING e a tabela SALES, onde o LISTID da tabela LISTING está entre 1 e 5. Essa consulta corresponde aos

valores da coluna LISTID na tabela LISTING (a tabela à esquerda) e na tabela SALES (tabela à direita). Os resultados mostram que LISTID 1, 4 e 5 correspondem aos critérios.

```
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
```

O exemplo a seguir é uma junção interna com a cláusula ON. Nesse caso, as linhas NULL não são retornadas.

```
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
```

A consulta a seguir é uma junção interna de duas subconsultas na cláusula FROM. A consulta encontra o número de ingressos vendidos e não vendidos para categorias diferentes de eventos (shows e apresentações). As subconsultas da cláusula FROM são subconsultas da tabela. Elas podem retornar várias colunas e linhas.

```
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
```

ESQUERDA [EXTERNA]

Retorna todos os valores da referência da tabela à esquerda e os valores correspondentes da referência da tabela à direita ou acrescenta NULL se não houver correspondência. Também é conhecida como junção externa esquerda.

Ele retorna todas as linhas da tabela esquerda (primeira) e as linhas correspondentes da tabela direita (segunda). Se não houver correspondência na tabela à direita, o conjunto de resultados conterá valores NULL para as colunas da tabela à direita. A palavra-chave OUTER pode ser omitida e a junção pode ser escrita simplesmente como LEFT JOIN. O oposto de um LEFT OUTER JOIN é um RIGHT OUTER JOIN, que retorna todas as linhas da tabela direita e as linhas correspondentes da tabela esquerda.

Sintaxe:

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

A consulta a seguir retornará todas as linhas da tabela de clientes, junto com as linhas correspondentes da tabela de pedidos. Se um cliente não tiver pedidos, o conjunto de resultados ainda incluirá as informações desse cliente, com valores NULL para as colunas order_id e order_date.

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

A consulta a seguir é uma junção externa à esquerda. Junções externas esquerdas e direitas retêm valores de uma das tabelas de junção quando nenhuma correspondência é encontrada na outra tabela. As tabelas esquerdas e direitas são a primeiras e a segunda listadas na sintaxe. Os valores NULL são usados para preencher "lacunas" no conjunto de resultados. Essa consulta corresponde aos valores da coluna LISTID na tabela LISTING (a tabela à esquerda) e na tabela SALES (tabela à direita). Os resultados mostram que LISTIDs 2 e 3 não resultaram em nenhuma venda.

```
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
```

DIREITO [EXTERNO]

Retorna todos os valores da referência da tabela à direita e os valores correspondentes da referência da tabela à esquerda ou acrescenta NULL se não houver correspondência. Também é conhecida como junção externa direita.

Ele retorna todas as linhas da tabela direita (segunda) e as linhas correspondentes da tabela esquerda (primeira). Se não houver correspondência na tabela à esquerda, o conjunto de resultados conterá valores NULL para as colunas da tabela à esquerda. A palavra-chave OUTER pode ser omitida e a junção pode ser escrita simplesmente como RIGHT JOIN. O oposto de um RIGHT OUTER JOIN é um LEFT OUTER JOIN, que retorna todas as linhas da tabela esquerda e as linhas correspondentes da tabela direita.

Sintaxe:

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

A consulta a seguir retornará todas as linhas da tabela de clientes, junto com as linhas correspondentes da tabela de pedidos. Se um cliente não tiver pedidos, o conjunto de resultados ainda incluirá as informações desse cliente, com valores NULL para as colunas order_id e order_date.

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

A consulta a seguir é uma junção externa à direita. Essa consulta corresponde aos valores da coluna LISTID na tabela LISTING (a tabela à esquerda) e na tabela SALES (tabela à direita). Os resultados mostram que LISTIDs 1, 4 e 5 correspondem aos critérios.

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

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

COMPLETO [EXTERNO]

Retorna todos os valores de ambas as relações, anexando valores NULL no lado que não tem correspondência. Também é conhecida como junção externa completa.

Ele retorna todas as linhas das tabelas esquerda e direita, independentemente de haver uma correspondência ou não. Se não houver correspondência, o conjunto de resultados conterá valores NULL para as colunas da tabela que não têm uma linha correspondente. A palavra-chave OUTER pode ser omitida e a junção pode ser escrita simplesmente como FULL JOIN. O FULL OUTER JOIN

é menos comumente usado do que o LEFT OUTER JOIN ou o RIGHT OUTER JOIN, mas pode ser útil em determinados cenários em que você precisa ver todos os dados das duas tabelas, mesmo que não haja correspondências.

Sintaxe:

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

A consulta a seguir retornará todas as linhas das tabelas de clientes e pedidos. Se um cliente não tiver pedidos, o conjunto de resultados ainda incluirá as informações desse cliente, com valores NULL para as colunas order_id e order_date. Se um pedido não tiver nenhum cliente associado, o conjunto de resultados incluirá esse pedido, com valores NULL para as colunas customer_id e name.

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

A consulta a seguir é uma junção completa. As junções completas retêm valores das tabelas unidas quando nenhuma correspondência é encontrada na outra tabela. As tabelas esquerdas e direitas são a primeiras e a segunda listadas na sintaxe. Os valores NULL são usados para preencher "lacunas" no conjunto de resultados. Essa consulta corresponde aos valores da coluna LISTID na tabela LISTING (a tabela à esquerda) e na tabela SALES (tabela à direita). Os resultados mostram que LISTIDs 2 e 3 não resultaram em nenhuma venda.

```
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
```

A consulta a seguir é uma junção completa. Essa consulta corresponde aos valores da coluna LISTID na tabela LISTING (a tabela à esquerda) e na tabela SALES (tabela à direita). Somente as linhas que não resultam em nenhuma venda (LISTIDs 2 e 3) estão nos resultados.

```
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
```

[ESQUERDA] SEMI

Retorna valores do lado esquerdo da referência da tabela que coincidem com o direito. Também é conhecida como junção semi esquerda.

Ele retorna somente as linhas da tabela esquerda (primeira) que têm uma linha correspondente na tabela direita (segunda). Ele não retorna nenhuma coluna da tabela à direita - somente as colunas da tabela à esquerda. O LEFT SEMI JOIN é útil quando você deseja encontrar as linhas em uma tabela que coincidem em outra tabela, sem precisar retornar nenhum dado da segunda tabela. O LEFT SEMI JOIN é uma alternativa mais eficiente ao uso de uma subconsulta com uma cláusula IN ou EXISTS.

Sintaxe:

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

A consulta a seguir retornará somente as colunas customer_id e name da tabela de clientes, para os clientes que têm pelo menos um pedido na tabela de pedidos. O conjunto de resultados não incluirá nenhuma coluna da tabela de pedidos.

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

CROSS JOIN

Retorna o produto cartesiano de duas relações. Isso significa que o conjunto de resultados conterá todas as combinações possíveis de linhas das duas tabelas, sem nenhuma condição ou filtro aplicado.

O CROSS JOIN é útil quando você precisa gerar todas as combinações possíveis de dados de duas tabelas, como no caso da criação de um relatório que exibe todas as combinações possíveis de informações do cliente e do produto. O CROSS JOIN é diferente de outros tipos de junção (INNER JOIN, LEFT JOIN etc.) porque não tem uma condição de junção na cláusula ON. A condição de junção não é necessária para um CROSS JOIN.

Sintaxe:

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

A consulta a seguir retornará um conjunto de resultados que contém todas as combinações possíveis de customer_id, customer_name, product_id e product_name das tabelas de clientes e produtos. Se a tabela de clientes tiver 10 linhas e a tabela de produtos tiver 20 linhas, o conjunto de resultados do CROSS JOIN conterá 10 x 20 = 200 linhas.

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

A consulta a seguir é uma junção cruzada ou junção cartesiana da tabela LISTING e da tabela SALES com um predicado para limitar os resultados. Essa consulta corresponde aos valores da coluna LISTID na tabela SALES e na tabela LISTING para LISTIDs 1, 2, 3, 4 e 5 em ambas as tabelas. Os resultados mostram que 20 linhas correspondem aos critérios.

```
select sales.listid as sales_listid, listing.listid as listing_listid
```

```
from sales cross join listing
where sales.listid between 1 and 5
and listing.listid between 1 and 5
order by 1,2;

sales_listid | listing_listid
-------------+---------------
1            | 1
1            | 2
1            | 3
1            | 4
1            | 5
4            | 1
4            | 2
4            | 3
4            | 4
4            | 5
5            | 1
5            | 1
5            | 2
5            | 2
5            | 3
5            | 3
5            | 4
5            | 4
5            | 5
5            | 5
```

ANTI-JUNÇÃO

Retorna os valores da referência da tabela à esquerda que não têm correspondência com a referência da tabela à direita. Também é conhecido como anti-junção esquerda.

O ANTI JOIN é uma operação útil quando você deseja encontrar as linhas em uma tabela que não coincidem em outra tabela.

Sintaxe:

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

A consulta a seguir retornará todos os clientes que não fizeram nenhum pedido.

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

NATURAL

Especifica que as linhas das duas relações serão correspondidas implicitamente em igualdade para todas as colunas com nomes correspondentes.

Ele combina automaticamente colunas com o mesmo nome e tipo de dados entre as duas tabelas. Não é necessário especificar explicitamente a condição de junção na cláusula ON. Ele combina todas as colunas correspondentes entre as duas tabelas no conjunto de resultados.

O NATURAL JOIN é uma abreviatura conveniente quando as tabelas que você está unindo têm colunas com os mesmos nomes e tipos de dados. No entanto, geralmente é recomendável usar o INNER JOIN mais explícito... Sintaxe ON para tornar as condições de junção mais explícitas e fáceis de entender.

Sintaxe:

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

O exemplo a seguir é uma junção natural entre duas tabelas `employees` e`departments`, com as seguintes colunas:

- `employees`tabela:`employee_id`,`first_name`,`last_name`, `department_id`
- `departments`tabela:`department_id`, `department_name`

A consulta a seguir retornará um conjunto de resultados que inclui o nome, o sobrenome e o nome do departamento de todas as linhas correspondentes entre as duas tabelas, com base na `department_id` coluna.

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

```
FROM employees e
NATURAL JOIN departments d;
```

O exemplo a seguir é uma junção natural entre duas tabelas. Nesse caso, as colunas listid, sellerid, eventid e dateid têm nomes e tipos de dados idênticos em ambas as tabelas e, portanto, são usadas como colunas de junção. Os resultados são limitados a cinco linhas.

```
select listid, sellerid, eventid, dateid, numtickets
from listing natural join sales
order by 1
limit 5;

listid | sellerid  | eventid | dateid | numtickets
-------+-----------+---------+--------+-----------
113    | 29704     | 4699    | 2075   | 22
115    | 39115     | 3513    | 2062   | 14
116    | 43314     | 8675    | 1910   | 28
118    | 6079      | 1611    | 1862   | 9
163    | 24880     | 8253    | 1888   | 14
```

Cláusula WHERE

A cláusula WHERE contém as condições que juntam as tabelas ou aplicam predicados às colunas nas tabelas. Tabelas internas que foram juntadas usando a sintaxe apropriada, seja com a cláusula WHERE ou com a cláusula FROM. Os critérios de junção externa devem ser especificados na cláusula FROM.

Sintaxe

```
[ WHERE condition ]
```

condição

Qualquer condição de pesquisa com um resultado booleano, como uma condição de junção ou um predicado em uma coluna de tabela. Os exemplos a seguir são condições de junção válidas:

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

Os exemplos a seguir são condições válidas nas colunas em tabelas:

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

As condições podem ser simples ou complexas; para condições complexas, você pode usar parênteses para isolar unidades lógicas. No exemplo a seguir, a condição de junção está entre parênteses.

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

Observações de uso

É possível usar aliases na cláusula WHERE para fazer referência a expressões da lista de seleção.

Não é possível restringir os resultados de funções agregadas na cláusula WHERE; use a cláusula HAVING para essa finalidade.

Colunas restringidas na cláusula WHERE devem ser derivadas de referências da tabela na cláusula FROM.

Exemplo

A consulta a seguir usa uma combinação de diferentes restrições da cláusula WHERE, incluindo uma condição de junção para as tabelas SALES e EVENT, um predicado na coluna EVENTNAME e dois predicados na coluna STARTTIME.

```
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Hannah Montana'
and date_part(quarter, starttime) in(1,2)
and date_part(year, starttime) = 2008
order by 3 desc, 4, 2, 1 limit 10;

eventname    |      starttime      |   costperticket   | qtysold
---------------+---------------------+-------------------+---------
Hannah Montana | 2008-06-07 14:00:00 |     1706.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |     1658.00000000 |       2
```

```
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       3
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       2
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       4
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       1
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       4
(10 rows)
```

Cláusula VALUES

A cláusula VALUES é usada para fornecer um conjunto de valores de linha diretamente na consulta, sem a necessidade de referenciar uma tabela.

A cláusula VALUES pode ser usada nos seguintes cenários:

- Você pode usar a cláusula VALUES em uma instrução INSERT INTO para especificar os valores das novas linhas que estão sendo inseridas em uma tabela.
- Você pode usar a cláusula VALUES sozinha para criar um conjunto de resultados temporário ou uma tabela embutida, sem a necessidade de referenciar uma tabela.
- Você pode combinar a cláusula VALUES com outras cláusulas SQL, como WHERE, ORDER BY ou LIMIT, para filtrar, classificar ou limitar as linhas no conjunto de resultados.

Essa cláusula é particularmente útil quando você precisa inserir, consultar ou manipular um pequeno conjunto de dados diretamente na instrução SQL, sem a necessidade de criar ou referenciar uma tabela permanente. Ele permite que você defina os nomes das colunas e os valores correspondentes para cada linha, oferecendo a flexibilidade de criar conjuntos de resultados temporários ou inserir dados dinamicamente, sem a sobrecarga de gerenciar uma tabela separada.

Sintaxe

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

Parâmetros

expressão

Uma expressão que especifica uma combinação de um ou mais valores, operadores e funções SQL que resulta em um valor.

apelido de tabela

Um alias que especifica um nome temporário com uma lista opcional de nomes de colunas.

Exemplo

O exemplo a seguir cria uma tabela embutida, um conjunto de resultados temporário semelhante a uma tabela com duas colunas e. `col1` `col2` A única linha no conjunto de resultados contém os valores `"one"` e`1`, respectivamente. A `SELECT * FROM` parte da consulta simplesmente recupera todas as colunas e linhas desse conjunto de resultados temporário. Os nomes das colunas (`col1`e`col2`) são gerados automaticamente pelo sistema de banco de dados, porque a cláusula VALUES não especifica explicitamente os nomes das colunas.

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

Se quiser definir nomes de colunas personalizados, você pode fazer isso usando uma cláusula AS após a cláusula VALUES, assim:

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

Isso criaria um conjunto de resultados temporário com os nomes das colunas `name` `eid`, em vez do padrão `col1` `col2` e.

Cláusula GROUP BY

A cláusula GROUP BY identifica as colunas de agrupamento para a consulta. As colunas de agrupamento devem ser declaradas quando a consulta computa agregadas com funções padrão como SUM, AVG e COUNT. Se uma função agregada estiver presente na expressão SELECT, qualquer coluna na expressão SELECT que não esteja em uma função agregada deverá estar na cláusula GROUP BY.

Para obter mais informações, consulte AWS Clean Rooms Funções do Spark SQL.

Sintaxe

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

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

Parâmetros

expr

A lista de colunas ou de expressões deve corresponder à lista de expressões não agregadas na lista de seleção da consulta. Por exemplo, considere a seguinte consulta simples.

```
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)
```

Nesta consulta, a lista de seleção consiste em duas expressões agregadas. A primeira usa a função SUM e a segunda usa a função COUNT. As duas colunas restantes, LISTID e EVENTID, devem ser declaradas como colunas de agrupamento.

As expressões na cláusula GROUP BY também podem fazer referência à lista de seleção usando números ordinais. O exemplo anterior poderia ser abreviado da seguinte forma.

```
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

Você pode usar a extensão de agregação ROLLUP para executar o trabalho de múltiplas operações GROUP BY em uma única instrução. Para obter mais informações sobre extensões de agregação e funções relacionadas, consulte Extensões de agregação.

Extensões de agregação

AWS Clean Roomssuporta extensões de agregação para realizar o trabalho de várias operações GROUP BY em uma única instrução.

GROUPING SETS

Calcula um ou mais conjuntos de agrupamento em uma única instrução. Um conjunto de agrupamento é o conjunto de uma única cláusula GROUP BY, um conjunto de 0 ou mais colunas pelo qual você pode agrupar o conjunto de resultados de uma consulta. GROUP BY GROUPING SETS é equivalente a executar uma consulta UNION ALL em um conjunto de resultados agrupado por colunas diferentes. Por exemplo, GROUP BY GROUPING SETS((a), (b)) é equivalente a GROUP BY a UNION ALL GROUP BY b.

O exemplo a seguir retorna o custo dos produtos da tabela de pedidos agrupados de acordo com as categorias de produtos e o tipo de produto vendido.

```
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

Assume uma hierarquia em que as colunas anteriores são consideradas pais das colunas subsequentes. ROLLUP agrupa os dados pelas colunas fornecidas, retornando linhas de subtotal extras representando os totais em todos os níveis de colunas de agrupamento, além das linhas agrupadas. Por exemplo, você pode usar GROUP BY ROLLUP((a), (b)) para retornar um conjunto de resultados agrupado primeiro por a, depois por b, assumindo que b é uma subseção de a. ROLLUP também retorna uma linha com todo o conjunto de resultados sem colunas de agrupamento.

GROUP BY ROLLUP((a), (b)) é equivalente a GROUP BY GROUPING SETS((a,b), (a), ()).

O exemplo a seguir retorna o custo dos produtos da tabela de pedidos agrupados primeiro por categoria, depois por produto, com o produto como uma subdivisão da categoria.

```
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 os dados pelas colunas fornecidas, retornando linhas de subtotal extras representando os totais em todos os níveis de colunas de agrupamento, além das linhas agrupadas. CUBE retorna as mesmas linhas que ROLLUP, enquanto inclui linhas de subtotal adicionais para cada combinação de coluna de agrupamento não contemplada por ROLLUP. Por exemplo, você pode usar GROUP BY CUBE((a), (b)) para retornar um conjunto de resultados agrupado primeiro por a, depois por b, assumindo que b é uma subseção de a, depois apenas por b. CUBE também retorna uma linha com todo o conjunto de resultados sem colunas de agrupamento.

GROUP BY CUBE((a), (b)) é equivalente a GROUP BY GROUPING SETS((a,b), (a), (b), ()).

O exemplo a seguir retorna o custo dos produtos da tabela de pedidos agrupados primeiro por categoria, depois por produto, com o produto como uma subdivisão da categoria. Ao contrário do exemplo anterior para ROLLUP, a instrução retorna resultados para cada combinação de coluna de agrupamento.

```
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

A cláusula HAVING aplica uma condição a um conjunto de resultados agrupados intermediários retornados por uma consulta.

Sintaxe

```
[ HAVING condition ]
```

Por exemplo, você pode restringir os resultados de uma função SUM:

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

A condição HAVING é aplicada depois que todas as condições da cláusula WHERE forem aplicadas e as operações GROUP BY concluídas.

A própria condição leva a mesma forma que qualquer condição da cláusula WHERE.

Observações de uso

- Qualquer coluna referida na condição da cláusula HAVING deve ser uma coluna de agrupamento ou uma coluna que faz referência ao resultado de uma função agregada.
- Em uma cláusula HAVING, você não pode especificar:
  - Número ordinal que se refere a um item na lista de seleção. Somente as cláusulas GROUP BY e ORDER BY aceitam números ordinais.

Exemplos

A consulta a seguir calcula as vendas de ingressos globais para todos os eventos por nome e depois elimina eventos em que as vendas globais tenham sido menos de $ 800.000. A condição HAVING é aplicada aos resultados da função agregada na lista de seleção: `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)
```

A consulta a seguir calcula um conjunto de resultados semelhante. Nesse caso, no entanto, a condição HAVING é aplicada a um valor agregado não especificado na lista de seleção: `sum(qtysold)`. Os eventos que não tenham vendido mais de 2.000 ingressos são eliminados dos resultados finais.

```
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)
```

Configurar operadores

Os operadores de conjunto são usados para comparar e mesclar os resultados de duas expressões de consulta separadas.

AWS Clean RoomsO Spark SQL é compatível com os seguintes operadores de conjunto listados na tabela a seguir.

Definir operador
INTERSECT
CRUZAR TUDO
EXCEPT

Definir operador
EXCETO TODOS
UNION
UNION ALL

Por exemplo, se você quiser saber quais usuários de um site compram e vendem, mas os nomes de usuários estiverem armazenados em colunas ou tabelas separadas, você pode encontrar a interseção desses dois tipos de usuários. Se você quiser saber quais usuários do site compram, mas não vendem, você pode usar o operador EXCEPT para encontrar a diferença entre as duas listas de usuários. Se quiser criar uma lista com todos os usuários, independentemente da função, use o operador UNION.

> **Note**
>
> As cláusulas ORDER BY, LIMIT, SELECT TOP e OFFSET não podem ser usadas nas expressões de consulta mescladas pelos operadores de conjunto UNION, UNION ALL, INTERSECT e EXCEPT.

Tópicos

Sintaxe

```
subquery1
```

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

Parâmetros

subconsulta1, subconsulta2

Uma expressão de consulta que corresponde, no formato de sua lista de seleção, a uma segunda expressão de consulta que segue o operador UNION, UNION ALL, INTERSECT, INTERSECT ALL, EXCEPT ou EXCEPT ALL. As duas expressões devem conter o mesmo número de colunas de saída com tipos de dados compatíveis. Caso contrário, os dois conjuntos de resultados não poderão ser comparados e mesclados. As operações de conjunto não permitem a conversão implícita entre diferentes categorias de tipos de dados. Para obter mais informações, consulte Compatibilidade e conversão dos tipos.

Você pode criar consultas contendo um número ilimitado de expressões de consulta e conectá-las aos operadores UNION, INTERSECT e EXCEPT em qualquer combinação. Por exemplo, a estrutura de consulta a seguir é válida, pressupondo que as tabelas T1, T2 e T3 contenham conjuntos compatíveis de colunas:

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

UNIÃO [TUDO | DISTINTO]

Operação de conjunto que retorna linhas de duas expressões de consulta, independentemente das linhas se derivarem de uma ou ambas as expressões.

CRUZAR [TUDO | DISTINTO]

Operação de conjunto que retorna linhas derivadas de duas expressões de consulta. As linhas que não forem retornadas por ambas as expressões serão descartadas.

EXCETO [TUDO | DISTINTO]

Operação de conjunto que retorna linhas derivadas de uma das duas expressões de consulta. Para se qualificar para o resultado, as linhas precisam existir na primeira tabela de resultados, mas não na segunda.

EXCEPT ALL não remove duplicatas das linhas de resultados.

MINUS e EXCEPT são sinônimos.

Ordem de avaliação para operadores de conjunto

Os operadores de conjunto UNION e EXCEPT se associam à esquerda. Se não houver parênteses especificados para influenciar a ordem de precedência, uma combinação desses operadores de conjunto será avaliada da esquerda para a direita. Por exemplo, na consulta a seguir, o operador UNION de T1 e T2 é avaliado primeiro, seguido pela operação EXCEPT, que é executada no resultado de UNION:

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

O operador INTERSECT tem precedência sobre os operadores UNION e EXCEPT quando uma combinação de operadores for usada na mesma consulta. Por exemplo, a consulta a seguir avalia a interseção de T2 e T3, e depois une o resultado com T1:

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

Adicionando parênteses, você pode aplicar uma ordem diferente de avaliação. No caso a seguir, o resultado da união de T1 e T2 é cruzado com T3, e a consulta provavelmente produzirá um resultado diferente.

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

Observações de uso

- Os nomes de colunas obtidos no resultado de uma consulta de operação de conjunto são os nomes de colunas (ou aliases) das tabelas na primeira expressão de consulta. Como esses nomes de coluna podem induzir a erros, os valores na coluna derivam de tabelas em ambos os lados do operador de conjunto, você pode querer fornecer aliases significativos para o conjunto de resultados.
- Quando as consultas do operador de conjunto retornam resultados decimais, as colunas de resultados correspondentes são promovidas para retornar a mesma precisão e escala. Por exemplo, na consulta a seguir, em que T1.REVENUE é uma coluna DECIMAL(10,2) e T2.REVENUE é uma coluna DECIMAL(8,4), o resultado decimal é atualizado para DECIMAL(12,4):

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

  A escala é 4 porque é a escala máxima das duas colunas. A precisão é 12 porque T1.REVENUE requer 8 dígitos à esquerda do ponto decimal (12 - 4 = 8). Essa promoção de tipo garante que todos os valores de ambos os lados de UNION se encaixem no resultado. Para valores de 64 bits, a precisão máxima de resultado é 19 e a escala máxima de resultado é 18. Para valores de 128-bits, a precisão máxima de resultado é 38 e a escala máxima de resultado é 37.

  Se o tipo de dados resultante exceder os limites AWS Clean Rooms de precisão e escala, a consulta retornará um erro.
- Para operações de conjunto, duas linhas são tratadas como idênticas se, para cada par de colunas correspondente, os dois valores de dados forem iguais ou ambos NULL. Por exemplo, se as tabelas T1 e T2 contiverem uma coluna e uma linha, e a linha for NULL em ambas as tabelas, uma operação INTERSECT sobre essas tabelas retornará essa linha.

Exemplos de consultas UNION

Na consulta UNION a seguir, as linhas na tabela SALES são mescladas com as linhas na tabela LISTING. Três colunas compatíveis de cada tabela são selecionadas. Nesse caso, as colunas correspondentes têm os mesmos nomes e tipos de dados.

```
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
```

O exemplo a seguir mostra como você pode adicionar um valor literal de saída de uma consulta UNION para ver qual expressão de consulta produziu cada linha no conjunto de resultados. A consulta identifica linhas da primeira expressão de consulta como “B” (para compradores) e linhas da segunda expressão de consulta como “S” (para vendedores).

A consulta identifica compradores e vendedores para as transações de ingressos que custem $10.000 ou mais. A única diferença entre as duas expressões de consulta em ambos os lados do operador UNION é a coluna de junção para a tabela 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
```

O exemplo a seguir usa um operador UNION ALL porque se forem encontradas linhas duplicadas, elas devem ser mantidas no resultado. Para uma série específica de eventos IDs, a consulta retorna 0 ou mais linhas para cada venda associada a cada evento e 0 ou 1 linha para cada anúncio desse

evento. IDs Os eventos são exclusivos para cada linha nas tabelas LISTING e EVENT, mas pode haver várias vendas para a mesma combinação de evento e anúncio IDs na tabela SALES.

A terceira coluna no conjunto de resultados identifica a origem da linha. Se vier da tabela SALES, “Yes” é marcado na coluna SALESROW. (SALESROW é um alias para SALES.LISTID.) Se a linha vier da tabela LISTING, “No” é marcado na coluna SALESROW.

Nesse caso, o conjunto de resultados consiste em três linhas de vendas para a lista 500, evento 7787. Em outras palavras, três transações diferentes ocorreram para essa combinação de lista e evento. As outras duas listagens, 501 e 502, não produziram nenhuma venda, então a única linha que a consulta produz para essas listas IDs vem da tabela LISTING (SALESROW = 'Não').

```
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
```

Se você executar a mesma consulta sem a palavra-chave ALL, o resultado manterá somente uma das transações de vendas.

```
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
```

Exemplos de consultas UNION ALL

O exemplo a seguir usa um operador UNION ALL porque se forem encontradas linhas duplicadas, elas devem ser mantidas no resultado. Para uma série específica de eventos IDs, a consulta retorna 0 ou mais linhas para cada venda associada a cada evento e 0 ou 1 linha para cada anúncio desse evento. IDs Os eventos são exclusivos para cada linha nas tabelas LISTING e EVENT, mas pode haver várias vendas para a mesma combinação de evento e anúncio IDs na tabela SALES.

A terceira coluna no conjunto de resultados identifica a origem da linha. Se vier da tabela SALES, “Yes” é marcado na coluna SALESROW. (SALESROW é um alias para SALES.LISTID.) Se a linha vier da tabela LISTING, “No” é marcado na coluna SALESROW.

Nesse caso, o conjunto de resultados consiste em três linhas de vendas para a lista 500, evento 7787. Em outras palavras, três transações diferentes ocorreram para essa combinação de lista e evento. As outras duas listagens, 501 e 502, não produziram nenhuma venda, então a única linha que a consulta produz para essas listas IDs vem da tabela LISTING (SALESROW = 'Não').

```
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
```

Se você executar a mesma consulta sem a palavra-chave ALL, o resultado manterá somente uma das transações de vendas.

```
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
```

Exemplos de consultas INTERSECT

Compare o exemplo a seguir com o primeiro exemplo de UNION. A única diferença entre os dois exemplos é o operador de conjunto usado, mas os resultados são muito diferentes. Somente uma das linhas é a mesma:

```
235494 |    23875 |    8771
```

Essa é a única linha no resultado limitado de 5 linhas encontrada em ambas as tabelas.

```
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
```

A consulta a seguir encontra eventos (em que foram vendidos ingressos) que ocorreram em locais em Nova York e Los Angeles em março. A diferença entre as duas expressões de consulta é a restrição na coluna 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
```

Exemplos de consultas EXCEPT

A tabela CATEGORY no banco de dados contém as seguintes 11 linhas:

```
 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)
```

Pressuponha que uma tabela CATEGORY_STAGE (tabela de preparação) contém uma linha 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)
```

Retorne a diferença entre as duas tabelas. Em outras palavras, retorne as linhas que estão na tabela CATEGORY_STAGE, mas não na tabela CATEGORY:

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

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

A consulta equivalente a seguir usa o sinônimo MINUS.

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

catid | catgroup | catname |             catdesc
-------+----------+---------+---------------------------------
```

```
  12  | Concerts | Comedy  | All stand up comedy performances
(1 row)
```

Se você reverter a ordem das expressões SELECT, a consulta não retornará qualquer linha.

Cláusula ORDER BY

A cláusula ORDER BY classifica o conjunto de resultados de uma consulta.

> **Note**
>
> A expressão ORDER BY mais externa deve ter somente colunas que estejam na lista de seleção.

Tópicos

- Sintaxe
- Parâmetros
- Observações de uso
- Exemplos com ORDER BY

Sintaxe

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

Parâmetros

expressão

Expressão que define a ordem de classificação do resultado da consulta. Ela consiste em uma ou mais colunas da lista de seleção. Os resultados são obtidos com base na ordem binária UTF-8. Também é possível especificar o seguinte:

- Números ordinais que representam a posição de entradas da lista de seleção (ou a posição das colunas na tabela se não houver lista de seleção)

- Aliases que definem entradas da lista de seleção

Quando a cláusula ORDER BY tiver várias expressões, o conjunto de resultados será classificado de acordo com a primeira expressão, e a segunda expressão será aplicada a linhas que tenham valores correspondentes com os da primeira expressão, e assim por diante.

ASC | DESC

Opção que define a ordem de classificação para a expressão, da seguinte forma:

- ASC: ascendente (por exemplo, de valores numéricos menores para maiores e de "A" a "Z" para strings de caracteres). Se nenhuma opção é especificada, os dados são classificados na ordem ascendente por padrão.
- DESC: descendente (de valores numéricos maiores para menores; de "Z" a "A" para strings).

NULLS FIRST | NULLS LAST

Opção que especifica se valores NULL devem ser classificados primeiro, antes de valores não nulos, ou por último, depois de valores não nulos. Por padrão, os valores NULL são ordenados e classificados por último na ordem ASC e são ordenados e classificados primeiro na ordem DESC.

LIMIT number | ALL

Opção que controla o número de linhas classificadas que a consulta retorna. O número LIMIT deve ser um inteiro positivo. O valor máximo é `2147483647`.

LIMIT 0 não retorna linhas. Você pode usar essa sintaxe para fins de teste: para garantir que uma consulta seja executada (sem exibir qualquer linha) ou obter uma lista de colunas de uma tabela. Uma cláusula ORDER BY é redundante se você estiver usando LIMIT 0 para obter uma lista de colunas. O valor padrão é LIMIT ALL.

OFFSET start

Opção que especifica para ignorar o número de linhas antes de start antes de começar a retornar linhas. O número OFFSET deve ser um inteiro positivo. O valor máximo é `2147483647`. Quando usadas com a opção de LIMIT, as linhas OFFSET são ignoradas antes de iniciar a contagem de linhas LIMIT que são retornadas. Se a opção LIMIT não for usada, o número de linhas no conjunto de resultados será reduzido para o número de linhas ignoradas. As linhas ignoradas por uma cláusula OFFSET ainda precisam passar por varredura, e pode não ser eficiente usar um valor OFFSET grande.

Observações de uso

Observe o seguinte comportamento esperado com cláusulas ORDER BY:

- Os valores NULL são considerados "mais altos" que todos os demais valores. Com a ordem de classificação crescente padrão, os valores NULL são classificados no final. Para alterar esse comportamento, use a opção NULLS FIRST.
- Quando uma consulta não tiver uma cláusula ORDER BY, o sistema retornará conjuntos de resultados sem uma classificação previsível das linhas. A mesma consulta executada duas vezes pode retornar o conjunto de resultados em uma ordem diferente.
- As opções LIMIT e OFFSET podem ser usadas sem uma cláusula ORDER BY. No entanto, para obter um conjunto consistente de linhas, use essas opções em conjunto com ORDER BY.
- Em qualquer sistema paralelo, por exemploAWS Clean Rooms, quando ORDER BY não produz uma ordenação exclusiva, a ordem das linhas não é determinística. Ou seja, se a expressão ORDER BY produzir valores duplicados, a ordem de retorno dessas linhas poderá variar de outros sistemas ou de uma execução AWS Clean Rooms para a próxima.
- AWS Clean Roomsnão suporta literais de string nas cláusulas ORDER BY.

Exemplos com ORDER BY

Retorne todas as 11 linhas da tabela CATEGORY, classificada pela segunda coluna, CATGROUP. Para os resultados que têm o mesmo valor de CATGROUP, classifique os valores da coluna CATDESC pelo tamanho da string. Depois, organize pelas colunas CATID e 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)
```

Retorne colunas selecionadas da tabela SALES, classificada pelos valores mais altos de QTYSOLD. Limite o resultado às 10 primeiras linhas:

```
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
```

Retorne uma lista de colunas e nenhuma linha usando a sintaxe LIMIT 0:

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

Exemplos de subconsulta

Os exemplos a seguir mostram diferentes maneiras em que subconsultas se encaixam em consultas SELECT. Consulte Exemplo para obter outros exemplos de uso de subconsultas.

Subconsulta da lista SELECT

O exemplo a seguir contém um subconsulta na lista SELECT. Esta subconsulta é escalar: retorna somente uma coluna e um valor, que é repetido nos resultados para cada linha retornada da consulta exterior. A consulta compara o valor Q1SALES que a subconsulta computa com valores de vendas de outros dois trimestres (2 e 3) em 2008, como definido pela consulta externa.

```
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 da cláusula WHERE

O exemplo a seguir contém um subconsulta de tabela na cláusula WHERE. Essa subconsulta produz várias linhas. Nesse caso, as linhas contêm apenas uma coluna, mas as subconsultas da tabela podem conter várias colunas e linhas, assim como qualquer outra tabela.

A consulta encontra os 10 principais vendedores em termos quantidade máxima de ingressos vendidos. A lista dos 10 principais é restringida pela subconsulta, que remove usuários que vivem em cidades onde há locais de venda de ingressos. Essa consulta pode ser gravada de diferentes maneiras. Por exemplo, a subconsulta pode ser regravada como uma junção na 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 da cláusula WITH

Consulte Cláusula WITH.

Subconsultas correlacionadas

O exemplo a seguir contém uma subconsulta correlacionada na cláusula WHERE. Esse tipo de subconsulta contém uma ou mais correlações entre as colunas e as colunas produzidas pela consulta externa. Nesse caso, a correlação é `where s.listid=l.listid`. Para cada linha que a consulta externa produz, a subconsulta é executada para qualificar ou desqualificar a linha.

```
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)
```

Padrões de subconsultas correlacionadas não compatíveis

O planejador de consultas usa um método de regravação de consulta chamado decorrelação de subconsultas para otimizar vários padrões de subconsultas correlacionadas para execução em um ambiente de processamento paralelo massivo (MPP). Alguns tipos de subconsultas correlacionadas seguem padrões que não AWS Clean Rooms podem ser correlacionados e não são compatíveis. Consultas que contenham erros de retorno das seguintes referências de correlação:

- Referências de correlação que ignoram um bloco de consultas, também conhecidas como "referências de correlação para ignorar consultas". Por exemplo, na consulta a seguir, o bloco

contendo a referência de correlação e o bloco ignorado estão conectados por um 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));
```

O bloco ignorado nesse caso é a subconsulta na tabela LISTING. A referência de correlação correlaciona as tabelas EVENT e SALES.

- Referências de correlação de uma subconsulta que é parte de uma cláusula ON em uma 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);
```

A cláusula ON contém uma referência de correlação de SALES na subconsulta de EVENT na consulta externa.

- Referências de correlação sensíveis a nulos a uma tabela do sistema. AWS Clean Rooms Por exemplo:

```
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);
```

- Referências de correlação de dentro de uma subconsulta que contém uma função de janela.

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

- Referências em uma coluna GROUP BY para os resultados de um subconsulta correlacionada. Por exemplo:

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

- Referências de correlação de uma subconsulta com uma função agregada e uma cláusula GROUP BY, conectada à consulta externa por um predicado IN. (Essa restrição não se aplica a funções agregadas MIN e MAX.) Por exemplo:

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

AWS Clean Rooms Funções do Spark SQL

AWS Clean Rooms O Spark SQL é compatível com as seguintes funções SQL:

Tópicos

- Funções relacionadas à privacidade
- Funções de janela

Funções agregadas

As funções agregadas no AWS Clean Rooms Spark SQL são usadas para realizar cálculos ou operações em um grupo de linhas e retornar um único valor. Eles são essenciais para tarefas de análise e resumo de dados.

AWS Clean Rooms O Spark SQL é compatível com as seguintes funções agregadas:

Tópicos

- Função ANY_VALUE
- Função APPROX COUNT_DISTINCT
- Função PERCENTILE APROXIMADA
- Função do AVG
- Função BOOL_AND
- Função BOOL_OR
- Função CARDINALITY
- função COLLECT_LIST
- função COLLECT_SET
- Funções COUNT e COUNT DISTINCT
- Função COUNT
- Função MAX
- Função MEDIAN
- Função MIN
- Função PERCENTILE
- Função SKEWNESS
- Funções STDDEV_SAMP e STDDEV_POP
- Funções SUM e SUM DISTINCT
- Funções VAR_SAMP e VAR_POP

Função ANY_VALUE

A função ANY_VALUE retorna qualquer valor dos valores de expressão de entrada não deterministicamente. Esta função pode retornar NULL se a expressão de entrada não resultar no retorno de nenhuma linha.

Sintaxe

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

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera. A expressão é um destes tipos de dados:

isIgnoreNull

Um booleano que determina se a função deve retornar somente valores não nulos.

Retornos

Retorna o mesmo tipo de dados da expressão.

Observações de uso

Se uma instrução que especifica a função ANY_VALUE para uma coluna também incluir uma segunda referência de coluna, a segunda coluna deve aparecer em uma cláusula GROUP BY ou ser incluída em uma função agregada.

Exemplos

O exemplo a seguir exibe uma instância de qualquer `dateid` em que o `eventname` seja `Eagles`.

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

A seguir estão os resultados.

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

O exemplo a seguir exibe uma instância de qualquer `dateid` em que o `eventname` seja `Eagles` ou `Cold War Kids`.

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

A seguir estão os resultados.

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

Função APPROX COUNT_DISTINCT

APPROX COUNT_DISTINCT fornece uma maneira eficiente de estimar o número de valores exclusivos em uma coluna ou conjunto de dados.

Sintaxe

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

Argumentos

expr

A expressão ou coluna para a qual você deseja estimar o número de valores exclusivos.

Pode ser uma única coluna, uma expressão complexa ou uma combinação de colunas.

SD relativo

Um parâmetro opcional que especifica o desvio padrão relativo desejado da estimativa.

É um valor entre 0 e 1, representando o erro relativo máximo aceitável da estimativa. Um valor menor de RelativeSD resultará em uma estimativa mais precisa, porém mais lenta.

Se esse parâmetro não for fornecido, um valor padrão (geralmente em torno de 0,05 ou 5%) será usado.

Retornos

Retorna a cardinalidade estimada em HyperLogLog ++. RelativeSD define o desvio padrão relativo máximo permitido.

Exemplo

A consulta a seguir estima o número de valores exclusivos na `col1` coluna, com um desvio padrão relativo de 1% (0,01).

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

A consulta a seguir estima que há 3 valores exclusivos na `col1` coluna (os valores 1, 2 e 3).

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

Função PERCENTILE APROXIMADA

O PERCENTILE APROXIMADO é usado para estimar o valor percentual de uma determinada expressão ou coluna sem precisar classificar todo o conjunto de dados. Essa função é útil em cenários em que você precisa entender rapidamente a distribuição de um grande conjunto de dados ou rastrear métricas baseadas em percentis, sem a sobrecarga computacional de realizar um cálculo de percentil exato. No entanto, é importante entender as vantagens e desvantagens entre velocidade e precisão e escolher a tolerância de erro apropriada com base nos requisitos específicos do seu caso de uso.

Sintaxe

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

Argumentos

expr

A expressão ou coluna para a qual você deseja estimar o valor do percentil.

Pode ser uma única coluna, uma expressão complexa ou uma combinação de colunas.

percentil

O valor do percentil que você deseja estimar, expresso como um valor entre 0 e 1.

Por exemplo, 0,5 corresponderia ao 50º percentil (mediana).

precisão

Um parâmetro opcional que especifica a precisão desejada da estimativa do percentil. É um valor entre 0 e 1, representando o erro relativo máximo aceitável da estimativa. Um `accuracy` valor menor resultará em uma estimativa mais precisa, porém mais lenta. Se esse parâmetro não for fornecido, um valor padrão (geralmente em torno de 0,05 ou 5%) será usado.

Retornos

Retorna o percentil aproximado da coluna de intervalo numérico ou ANSI col, que é o menor valor nos valores de col ordenados (classificados do menor para o maior), de forma que não mais do que a porcentagem dos valores de col seja menor que o valor ou igual a esse valor.

O valor da porcentagem deve estar entre 0,0 e 1,0. O parâmetro de precisão (padrão: 10000) é um literal numérico positivo que controla a precisão da aproximação ao custo da memória.

Um valor mais alto de precisão gera melhor precisão, `1.0/accuracy` é o erro relativo da aproximação.

Quando a porcentagem é uma matriz, cada valor da matriz de porcentagem deve estar entre 0,0 e 1,0. Nesse caso, retorna a matriz de percentis aproximada da coluna col na matriz de porcentagem fornecida.

Exemplos

A consulta a seguir estima o 95º percentil da `response_time` coluna, com um erro relativo máximo de 1% (0,01).

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

A consulta a seguir estima os valores dos percentis 50, 40 e 10 da coluna na `col` tabela. `tab`

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

A consulta a seguir estima o 50º percentil (mediana) dos valores na coluna col.

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

Função do AVG

A função AVG retorna a média (média aritmética) dos valores da expressão de entrada. A função AVG trabalha com valores numéricos e ignora valores NULL.

Sintaxe

```
AVG (column)
```

Argumentos

column

A coluna de destino na qual a função opera. A coluna é um dos seguintes tipos de dados:

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

Tipos de dados

Os tipos de argumentos suportados pela função AVG são SMALLINT, INTEGER, BIGINT, DECIMAL e DOUBLE.

Os tipos de retorno suportados pela função AVG são:

- BIGINT para qualquer argumento de tipo inteiro
- DOUBLE para um argumento de ponto flutuante
- Retorna o mesmo tipo de dados que a expressão para qualquer outro tipo de argumento

A precisão padrão para um resultado de função AVG com um argumento DECIMAL é 38. A escala do resultado é a mesma que a escala do argumento. Por exemplo, um AVG de uma coluna DEC(5,2) de a retorna um tipo de dados DEC(38,2).

Exemplo

Encontre a quantidade média vendida por transação na tabela SALES.

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

Função BOOL_AND

A função BOOL_AND opera em uma única coluna ou expressão de boolianos ou inteiros. Essa função aplica lógica semelhante às funções BIT_AND e BIT_OR. Para essa função, o tipo de retorno é um valor booliano (`true` ou `false`).

Se todos os valores em um conjunto forem verdadeiros, a função BOOL_AND retorna `true` (`t`). Se qualquer valor for falso, a função retorna `false` (`f`).

Sintaxe

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

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera. Essa expressão deve ter um tipo de dados BOOLEAN ou de inteiros. O tipo de retorno da função é BOOLEAN.

DISTINCT | ALL

Com o argumento DISTINCT, a função elimina todos os valores duplicados para a expressão especificada antes de calcular o resultado. Com o argumento ALL, a função retém todos os valores duplicados. ALL é o padrão.

Exemplos

Você pode usar as funções booleanas com expressões booleanas ou expressões de inteiro.

Por exemplo, o seguinte retorno de consulta resultada da tabela USERS padrão no banco de dados TICKIT, que tem várias colunas booleanas.

A função BOOL_AND retorna `false` para todas as cinco linhas. Nem todos os usuários em cada um dos estados gostam de esportes.

```
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)
```

Função BOOL_OR

A função BOOL_OR opera em uma única coluna ou expressão de boolianos ou inteiros. Essa função aplica lógica semelhante às funções BIT_AND e BIT_OR. Para essa função, o tipo de retorno é um valor booliano (`true`, `false` ou NULL).

Se um valor de um conjunto for `true`, a função BOOL_OR retornará `true` (`t`). Se um valor de um conjunto for `false`, a função retornará `false` (`f`). NULL poderá ser retornado se o valor for desconhecido.

Sintaxe

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

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera. Essa expressão deve ter um tipo de dados BOOLEAN ou de inteiros. O tipo de retorno da função é BOOLEAN.

DISTINCT | ALL

Com o argumento DISTINCT, a função elimina todos os valores duplicados para a expressão especificada antes de calcular o resultado. Com o argumento ALL, a função retém todos os valores duplicados. ALL é o padrão.

Exemplos

Você pode usar as funções booleanas com expressões booleanas ou expressões de inteiro. Por exemplo, o seguinte retorno de consulta resultada da tabela USERS padrão no banco de dados TICKIT, que tem várias colunas booleanas.

A função BOOL_OR retorna `true` para todas as cinco linhas. Pelo menos um usuário em cada um dos estados gosta de esportes.

```
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)
```

O exemplo a seguir retorna NULL.

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

Função CARDINALITY

A função CARDINALITY retorna o tamanho de uma expressão ARRAY ou MAP (expr).

Essa função é útil para encontrar o tamanho ou o comprimento de uma matriz.

Sintaxe

```
cardinality(expr)
```

Argumentos

expr

Uma expressão ARRAY ou MAP.

Retornos

Retorna o tamanho de uma matriz ou mapa (INTEGER).

A função retorna `NULL` para entrada nula se `sizeOfNull` estiver definida como `false` ou `enabled` definida como. `true`

Caso contrário, a função retornará `-1` para entrada nula. Com as configurações padrão, a função retorna `-1` para entrada nula.

Exemplo

A consulta a seguir calcula a cardinalidade, ou o número de elementos, na matriz fornecida. O array `('b', 'd', 'c', 'a')` tem 4 elementos, então a saída dessa consulta seria4.

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

função COLLECT_LIST

A função COLLECT_LIST coleta e retorna uma lista de elementos não exclusivos.

Esse tipo de função é útil quando você deseja coletar vários valores de um conjunto de linhas em uma única matriz ou estrutura de dados de lista.

> **Note**
>
> A função não é determinística porque a ordem dos resultados coletados depende da ordem das linhas, que pode não ser determinística após a execução de uma operação aleatória.

Sintaxe

```
collect_list(expr)
```

Argumentos

expr

Uma expressão de qualquer tipo.

Retornos

Retorna um ARRAY do tipo de argumento. A ordem dos elementos na matriz não é determinística.

Valores NULL são excluídos.

Se DISTINCT for especificado, a função coletará somente valores exclusivos e será sinônimo de função `collect_set` agregada.

Exemplo

A consulta a seguir coleta todos os valores da coluna col em uma lista. A `VALUES` cláusula é usada para criar uma tabela embutida com três linhas, em que cada linha tem uma única coluna com os valores 1, 2 e 1, respectivamente. A `collect_list()` função é então usada para agregar todos os valores da coluna col em uma única matriz. A saída dessa instrução SQL seria a matriz`[1,2,1]`, que contém todos os valores da coluna col na ordem em que eles aparecem nos dados de entrada.

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

função COLLECT_SET

A função COLLECT_SET coleta e retorna um conjunto de elementos exclusivos.

Essa função é útil quando você deseja coletar todos os valores distintos de um conjunto de linhas em uma única estrutura de dados, sem incluir duplicatas.

> **Note**
>
> A função não é determinística porque a ordem dos resultados coletados depende da ordem das linhas, que pode não ser determinística após a execução de uma operação aleatória.

Sintaxe

```
collect_set(expr)
```

Argumentos

expr

Uma expressão de qualquer tipo, exceto MAP.

Retornos

Retorna um ARRAY do tipo de argumento. A ordem dos elementos na matriz não é determinística.

Valores NULL são excluídos.

Exemplo

A consulta a seguir coleta todos os valores exclusivos da coluna col em um conjunto. A `VALUES` cláusula é usada para criar uma tabela embutida com três linhas, em que cada linha tem uma única coluna com os valores 1, 2 e 1, respectivamente. A `collect_set()` função é então usada para agregar todos os valores exclusivos da coluna col em um único conjunto. A saída dessa instrução SQL seria o conjunto`[1,2]`, que contém os valores exclusivos da coluna col. O valor duplicado de 1 só é incluído uma vez no resultado.

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

Funções COUNT e COUNT DISTINCT

A função COUNT conta as linhas definidas pela expressão. A função COUNT DISTINCT calcula o número de valores distintos não NULL em uma coluna ou uma expressão. Ela elimina todos os valores duplicados da expressão especificada antes de realizar a contagem.

Sintaxe

```
COUNT (DISTINCT column)
```

Argumentos

column

A coluna de destino na qual a função opera.

Tipos de dados

As funções COUNT e COUNT DISTINCT comportam todos os tipos de dados de argumento.

A função COUNT DISTINCT exibe BIGINT.

Exemplos

Conte todos os usuários do Estado da Flórida.

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

Conte todo o local exclusivo IDs da EVENT mesa.

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

Função COUNT

A função COUNT conta as linhas definidas pela expressão.

A função COUNT tem as variações a seguir.

- COUNT (*) conta todas as linhas na tabela de destino independente se elas contêm nulls ou não.
- COUNT (expressão) computa o número de linhas com valores não NULL em uma coluna ou expressão específica.
- COUNT (expressão DISTINCT) computa o número de valores distintos não NULL em uma coluna ou expressão.

Sintaxe

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

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

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera. A função COUNT é compatível com todos os tipos de dados de argumento.

DISTINCT | ALL

Com o argumento DISTINCT, a função elimina todos os valores duplicados da expressão especificada antes realizar a contagem. Com o argumento ALL, a função retém todos os valores duplicados da expressão para contagem. ALL é o padrão.

Tipo de retorno

A função COUNT retorna BIGINT.

Exemplos

Conte todos os usuários do estado da Flórida:

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

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

Conte todos os nomes de eventos da tabela EVENT:

```
select count(eventname) from event;

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

Conte todos os nomes de eventos da tabela EVENT:

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

count
-------
```

```
8798
```

Conte todos os locais exclusivos IDs da tabela do EVENTO:

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

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

Conte o número de vezes que cada vendedor listou lotes de um ou mais ingressos para venda. Agrupe os resultados por 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
...
```

Função MAX

A função MAX retorna o valor máximo em um conjunto de linhas. DISTINCT ou ALL podem ser usadas, mas não afetam os resultados.

Sintaxe

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

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera. A expressão é qualquer tipo de dado numérico.

DISTINCT | ALL

Com o argumento DISTINCT, a função elimina todos os valores duplicados da expressão especificada antes de calcular o máximo. Com o argumento ALL, a função retém todos os valores duplicados da expressão para calcular o máximo. ALL é o padrão.

Tipos de dados

Retorna o mesmo tipo de dados da expressão.

Exemplos

Encontre o preço mais alto pago de todas as vendas:

```
select max(pricepaid) from sales;

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

Encontre o preço mais alto pago por ingresso em todas as vendas:

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

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

Função MEDIAN

Sintaxe

```
MEDIAN ( median_expression )
```

Argumentos

median_expression

A coluna ou expressão de destino na qual a função opera.

Função MIN

A função MIN retorna o valor mínimo em um conjunto de linhas. DISTINCT ou ALL podem ser usadas, mas não afetam os resultados.

Sintaxe

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

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera. A expressão é qualquer tipo de dado numérico.

DISTINCT | ALL

Com o argumento DISTINCT, a função elimina todos os valores duplicados da expressão especificada antes de calcular o mínimo. Com o argumento ALL, a função retém todos os valores duplicados da expressão para calcular o mínimo. ALL é o padrão.

Tipos de dados

Retorna o mesmo tipo de dados da expressão.

Exemplos

Encontre o preço mais baixo pago de todas as vendas:

```
select min(pricepaid) from sales;

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

Encontre o preço mais baixo pago por ingresso em todas as vendas:

```
select min(pricepaid/qtysold)as min_ticket_price
```

```
from sales;

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

Função PERCENTILE

A função PERCENTILE é usada para calcular o valor exato do percentil, primeiro classificando os valores na `col` coluna e, em seguida, localizando o valor no especificado. `percentage`

A função PERCENTILE é útil quando você precisa calcular o valor exato do percentil e o custo computacional é aceitável para seu caso de uso. Ela fornece resultados mais precisos do que a função APPROX_PERCENTILE, mas pode ser mais lenta, especialmente para grandes conjuntos de dados.

Por outro lado, a função APPROX_PERCENTILE é uma alternativa mais eficiente que pode fornecer uma estimativa do valor do percentil com uma tolerância de erro especificada, tornando-a mais adequada para cenários em que a velocidade é uma prioridade maior do que a precisão absoluta.

Sintaxe

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

Argumentos

resfriado

A expressão ou coluna para a qual você deseja calcular o valor do percentil.

porcentagem

O valor do percentil que você deseja calcular, expresso como um valor entre 0 e 1.

Por exemplo, 0,5 corresponderia ao 50º percentil (mediana).

frequência

Um parâmetro opcional que especifica a frequência ou o peso de cada valor na `col` coluna. Se fornecida, a função calculará o percentil com base na frequência de cada valor.

Retornos

Retorna o valor percentual exato da coluna numérica ou de intervalo ANSI col na porcentagem fornecida.

O valor da porcentagem deve estar entre 0,0 e 1,0.

O valor da frequência deve ser integral positiva

Exemplo

A consulta a seguir encontra o valor maior ou igual a 30% dos valores na `col` coluna. Como os valores são 0 e 10, o 30º percentil é 3,0, porque é o valor maior ou igual a 30% dos dados.

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

Função SKEWNESS

A função SKEWNESS retorna o valor de assimetria calculado a partir dos valores de um grupo.

A assimetria é uma medida estatística que descreve a assimetria ou a falta de simetria em um conjunto de dados. Ele fornece informações sobre a forma da distribuição de dados.

Essa função pode ser útil para entender as propriedades estatísticas de um conjunto de dados e informar análises adicionais ou tomadas de decisão.

Sintaxe

```
skewness(expr)
```

Argumentos

expr

Uma expressão que é avaliada como numérica.

Retornos

Retorna em DOBRO.

Se DISTINCT for especificado, a função operará somente em um conjunto exclusivo de valores expr.

Exemplos

A consulta a seguir calcula a assimetria dos valores na coluna. `col` Neste exemplo, a `VALUES` cláusula é usada para criar uma tabela embutida com quatro linhas, em que cada linha tem uma única coluna `col` com os valores -10, -20, 100 e 1000. A `skewness()` função é então usada para calcular a assimetria dos valores na `col` coluna. O resultado, 1.1135657469022011, representa o grau e a direção da distorção nos dados. Um valor de assimetria positivo indica que os dados estão inclinados para a direita, com a maior parte dos valores concentrados no lado esquerdo da distribuição. Um valor de distorção negativo indica que os dados estão inclinados para a esquerda, com a maior parte dos valores concentrados no lado direito da distribuição.

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

A consulta a seguir calcula a assimetria dos valores na coluna col. Semelhante ao exemplo anterior, a `VALUES` cláusula é usada para criar uma tabela embutida com quatro linhas, em que cada linha tem uma única coluna `col` com os valores -1000, -100, 10 e 20. A `skewness()` função é então usada para calcular a assimetria dos valores na `col` coluna. O resultado, -1.1135657469022011, representa o grau e a direção da distorção nos dados. Nesse caso, o valor de assimetria negativa indica que os dados estão inclinados para a esquerda, com a maior parte dos valores concentrados no lado direito da distribuição.

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

Funções STDDEV_SAMP e STDDEV_POP

As funções STDDEV_SAMP e STDDEV_POP retornam o desvio padrão da amostra e da população de um conjunto de valores numéricos (número inteiro, decimal ou ponto flutuante). O resultado da função STDDEV_SAMP é equivalente à raiz quadrada da variação de amostra do mesmo conjunto de valores.

STDDEV_SAMP e STDDEV são sinônimos para a mesma função.

Sintaxe

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

A expressão deve ter um tipo de dados numérico. Independente do tipo de dados da expressão, o tipo de retorno desta função é um número de precisão dupla.

> **Note**
>
> O desvio padrão é calculado utilizando a aritmética de ponto flutuante, que pode resultar em uma ligeira imprecisão.

Observações de uso

Quando o desvio padrão de amostra (STDDEV ou STDDEV_SAMP) é calculado para uma expressão que consiste em um único valor, o resultado da função é NULL ou 0.

Exemplos

A seguinte consulta retorna a média dos valores na coluna VENUESEATS da tabela VENUE, seguida pelo desvio padrão de amostra e desvio padrão de população do mesmo conjunto de valores. VENUESEATS é uma coluna INTEGER. A escala do resultado é reduzida 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)
```

A seguinte consulta retorna o desvio padrão de amostra para a coluna COMMISSION na tabela SALES. COMMISSION é uma coluna DECIMAL. A escala do resultado é reduzida a 10 dígitos.

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

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

A seguinte consulta converte o desvio padrão de amostra para a coluna COMMISSION para um inteiro.

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

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

A seguinte consulta retorna o desvio padrão da amostra e a raiz quadrada da variação da amostra para a coluna COMMISSION. Os resultados desses cálculos são os mesmos.

```
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)
```

Funções SUM e SUM DISTINCT

A função SUM retorna a soma da coluna de entrada ou dos valores da expressão. A função SUM trabalha com valores numéricos e ignora valores NULL.

A função SUM DISTINCT elimina todos os valores duplicados da expressão especificada antes de calcular a soma.

Sintaxe

```
SUM (DISTINCT column )
```

Argumentos

column

A coluna de destino na qual a função opera. A coluna é qualquer tipo de dado numérico.

Exemplos

Encontre a soma de todas as comissões pagas na tabela SALES.

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

Encontre a soma de todas as comissões distintas pagas na tabela SALES.

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

Funções VAR_SAMP e VAR_POP

As funções VAR_SAMP e VAR_POP retornam a variação da amostra e da população de um conjunto de valores numéricos (número inteiro, decimal ou ponto flutuante). O resultado da função VAR_SAMP é equivalente à raiz quadrada do desvio padrão da amostra do mesmo conjunto de valores.

VAR_SAMP e VARIANCE são sinônimos para a mesma função.

Sintaxe

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

A expressão deve ter um tipo de dados de número inteiro, decimal ou ponto flutuante. Independente do tipo de dados da expressão, o tipo de retorno desta função é um número de precisão dupla.

> **Note**
>
> Os resultados dessas funções podem variar entre os clusters de data warehouse dependendo da configuração do cluster em cada caso.

Observações de uso

Quando a variação da amostra (VARIANCE ou VAR_SAMP) é calculada para uma expressão que consiste em um único valor, o resultado da função é NULL ou 0.

Exemplos

A seguinte consulta retorna a variação arredondada da amostra e da população para a coluna NUMTICKETS da tabela LISTING.

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

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

A seguinte consulta executa os mesmos cálculos, mas converte os resultados para valores decimais.

```
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)
```

Funções de array

Esta seção descreve as funções de matriz para SQL suportadas no AWS Clean Rooms.

Tópicos

- Função ARRAY
- função ARRAY_CONTAINS
- função ARRAY_DISTINCT
- função ARRAY_EXCEPT
- função ARRAY_INTERSECT
- função ARRAY_JOIN
- função ARRAY_REMOVE

- função ARRAY_UNION
- Função EXPLODE
- Função FLATTEN

Função ARRAY

Cria uma matriz com os elementos fornecidos.

Sintaxe

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

Argumento

expr1, expr2

Expressões de qualquer tipo de dados, exceto tipos de data e hora. Os argumentos não precisam ser do mesmo tipo de dado.

Tipo de retorno

A função de matriz retorna uma MATRIZ com os elementos na expressão.

Exemplo

O exemplo a seguir mostra uma matriz de valores numéricos e uma matriz de diferentes tipos de dados.

```
--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)
```

função ARRAY_CONTAINS

A função ARRAY_CONTAINS pode ser usada para realizar verificações básicas de associação em estruturas de dados de matriz. A função ARRAY_CONTAINS é útil quando você precisa verificar se um valor específico está presente em uma matriz.

Sintaxe

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

Argumentos

array

Uma MATRIZ a ser pesquisada.

value

Uma expressão com um tipo que compartilha um tipo menos comum com os elementos da matriz.

Tipo de retorno

A função ARRAY_CONTAINS retorna um BOOLEAN.

Se o valor for NULL, o resultado será NULL.

Se algum elemento na matriz for NULL, o resultado será NULL se o valor não corresponder a nenhum outro elemento.

Exemplos

O exemplo a seguir verifica se a matriz `[1, 2, 3]` contém o valor4. Como a matriz`[1, 2, 3]` não contém o valor4, a função array_contains retorna. `false`

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

O exemplo a seguir verifica se a matriz `[1, 2, 3]` contém o valor2. Como a matriz `[1, 2, 3]` contém o valor2, a função array_contains retorna. `true`

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

função ARRAY_DISTINCT

A função ARRAY_DISTINCT pode ser usada para remover valores duplicados de uma matriz. A função ARRAY_DISTINCT é útil quando você precisa remover duplicatas de uma matriz e trabalhar somente com os elementos exclusivos. Isso pode ser útil em cenários em que você deseja realizar operações ou análises em um conjunto de dados sem a interferência de valores repetidos.

Sintaxe

```
array_distinct(array)
```

Argumentos

array

Uma expressão ARRAY.

Tipo de retorno

A função ARRAY_DISTINCT retorna uma MATRIZ que contém somente os elementos exclusivos da matriz de entrada.

Exemplos

Neste exemplo, a matriz de entrada `[1, 2, 3, null, 3]` contém um valor duplicado de`3`. A `array_distinct` função remove esse valor duplicado `3` e retorna uma nova matriz com os elementos exclusivos:`[1, 2, 3, null]`.

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

Neste exemplo, a matriz de entrada `[1, 2, 2, 3, 3, 3]` contém valores duplicados de 2 e. 3 A `array_distinct` função remove essas duplicatas e retorna uma nova matriz com os elementos exclusivos:`[1, 2, 3]`.

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

função ARRAY_EXCEPT

A função ARRAY_EXCEPT usa duas matrizes como argumentos e retorna uma nova matriz que contém somente os elementos presentes na primeira matriz, mas não na segunda matriz.

O ARRAY_EXCEPT é útil quando você precisa encontrar os elementos que são exclusivos de uma matriz em comparação com outra. Isso pode ser útil em cenários em que você precisa realizar operações semelhantes a conjuntos em matrizes, como encontrar a diferença entre dois conjuntos de dados.

Sintaxe

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

Argumentos

matriz1

Uma MATRIZ de qualquer tipo com elementos comparáveis.

matriz2

Uma MATRIZ de elementos que compartilham um tipo menos comum com os elementos de array1.

Tipo de retorno

A função ARRAY_EXCEPT retorna um ARRAY do tipo correspondente ao array1 sem duplicatas.

Exemplos

Neste exemplo, a primeira matriz `[1, 2, 3]` contém os elementos 1, 2 e 3. A segunda matriz `[2, 3, 4]` contém os elementos 2, 3 e 4. A `array_except` função remove os elementos 2 e 3 da primeira matriz, pois eles também estão presentes na segunda matriz. A saída resultante é a matriz`[1]`.

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

Neste exemplo, a primeira matriz `[1, 2, 3]` contém os elementos 1, 2 e 3. A segunda matriz `[1, 3, 5]` contém os elementos 1, 3 e 5. A `array_except` função remove os elementos 1 e 3

da primeira matriz, pois eles também estão presentes na segunda matriz. A saída resultante é a matriz[2].

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

função ARRAY_INTERSECT

A função ARRAY_INTERSECT usa duas matrizes como argumentos e retorna uma nova matriz que contém os elementos presentes nas duas matrizes de entrada. Essa função é útil quando você precisa encontrar os elementos comuns entre duas matrizes. Isso pode ser útil em cenários em que você precisa realizar operações semelhantes a conjuntos em matrizes, como encontrar a interseção entre dois conjuntos de dados.

Sintaxe

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

Argumentos

matriz1

Uma MATRIZ de qualquer tipo com elementos comparáveis.

matriz2

Uma MATRIZ de elementos que compartilham um tipo menos comum com os elementos de array1.

Tipo de retorno

A função ARRAY_INTERSECT retorna uma MATRIZ do tipo correspondente à matriz1 sem duplicatas e elementos contidos na matriz1 e na matriz2.

Exemplos

Neste exemplo, a primeira matriz `[1, 2, 3]` contém os elementos 1, 2 e 3. A segunda matriz `[1, 3, 5]` contém os elementos 1, 3 e 5. A função ARRAY_INTERSECT identifica os elementos comuns entre as duas matrizes, que são 1 e 3. A matriz de saída resultante é`[1, 3]`.

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

```
 [1,3]
```

função ARRAY_JOIN

A função ARRAY_JOIN usa dois argumentos: o primeiro argumento é a matriz de entrada que será unida. O segundo argumento é a string separadora que será usada para concatenar os elementos da matriz. Essa função é útil quando você precisa converter uma matriz de strings (ou qualquer outro tipo de dados) em uma única string concatenada. Isso pode ser útil em cenários em que você deseja apresentar uma matriz de valores como uma única string formatada, por exemplo, para fins de exibição ou para uso em processamento posterior.

Sintaxe

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

Argumentos

array

Qualquer tipo de ARRAY, mas seus elementos são interpretados como strings.

delimitador

Uma STRING usada para separar os elementos concatenados da matriz.

Substituição nula

Uma STRING usada para expressar um valor NULL no resultado.

Tipo de retorno

A função ARRAY_JOIN retorna uma STRING em que os elementos da matriz são separados por delimitador e os elementos nulos são substituídos. `nullReplacement` Se `nullReplacement` for omitido, os `null` elementos serão filtrados. Se houver algum argumento`NULL`, o resultado é`NULL`.

Exemplos

Neste exemplo, a função ARRAY_JOIN pega a matriz `['hello', 'world']` e une os elementos usando o separador `' '` (um caractere de espaço). A saída resultante é a string`'hello world'`.

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

Neste exemplo, a função ARRAY_JOIN pega a matriz `['hello', null, 'world']` e une os elementos usando o separador `' '` (um caractere de espaço). O `null` valor é substituído pela string de substituição fornecida `','` (uma vírgula). A saída resultante é a string`'hello , world'`.

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

função ARRAY_REMOVE

A função ARRAY_REMOVE usa dois argumentos: o primeiro argumento é a matriz de entrada da qual os elementos serão removidos. O segundo argumento é o valor que será removido da matriz. Essa função é útil quando você precisa remover elementos específicos de uma matriz. Isso pode ser útil em cenários em que você precisa realizar a limpeza ou o pré-processamento de dados em uma matriz de valores.

Sintaxe

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

Argumentos

array

Um ARRAY.

Elemento

Uma expressão de um tipo que compartilha um tipo menos comum com os elementos da matriz.

Tipo de retorno

A função ARRAY_REMOVE retorna o tipo de resultado correspondente ao tipo da matriz. Se o elemento a ser removido for`NULL`, o resultado será`NULL`.

Exemplos

Neste exemplo, a função ARRAY_REMOVE pega a matriz `[1, 2, 3, null, 3]` e remove todas as ocorrências do valor 3. A saída resultante é a matriz`[1, 2, null]`.

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

```
 [1,2,null]
```

função ARRAY_UNION

A função ARRAY_UNION usa duas matrizes como argumentos e retorna uma nova matriz que contém os elementos exclusivos de ambas as matrizes de entrada. Essa função é útil quando você precisa combinar duas matrizes e eliminar quaisquer elementos duplicados. Isso pode ser útil em cenários em que você precisa realizar operações semelhantes a conjuntos em matrizes, como encontrar a união entre dois conjuntos de dados.

Sintaxe

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

Argumentos

matriz1

Um ARRAY.

matriz2

Um ARRAY do mesmo tipo que array1.

Tipo de retorno

A função ARRAY_UNION retorna uma MATRIZ do mesmo tipo da matriz.

Exemplo

Neste exemplo, a primeira matriz `[1, 2, 3]` contém os elementos 1, 2 e 3. A segunda matriz `[1, 3, 5]` contém os elementos 1, 3 e 5. A função ARRAY_UNION combina os elementos exclusivos de ambas as matrizes, resultando na matriz de saída. `[1, 2, 3, 5]` T

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

Função EXPLODE

A função EXPLODE é usada para transformar uma única linha com uma matriz ou coluna de mapa em várias linhas, onde cada linha corresponde a um único elemento da matriz ou do mapa.

Sintaxe

```
explode(expr)
```

Argumentos

expr

Uma expressão de matriz ou uma expressão de mapa.

Tipo de retorno

A função EXPLODE retorna um conjunto de linhas, em que cada linha representa um único elemento da matriz ou mapa de entrada.

O tipo de dados das linhas de saída depende do tipo de dados dos elementos na matriz de entrada ou no mapa.

Exemplos

O exemplo a seguir pega a matriz de linha única [10, 20] e a transforma em duas linhas separadas, cada uma contendo um dos elementos da matriz (10 e 20).

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

No primeiro exemplo, a matriz de entrada foi passada diretamente como argumento para`explode()`. Neste exemplo, a matriz de entrada é especificada usando a => sintaxe, em que o nome da coluna (`collection`) é fornecido explicitamente.

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

Ambas as abordagens são válidas e alcançam o mesmo resultado, mas a segunda sintaxe pode ser mais útil quando você precisa explodir uma coluna de um conjunto de dados maior, em vez de apenas uma simples matriz literal.

Função FLATTEN

A função FLATTEN é usada para “nivelar” uma estrutura de matriz aninhada em uma única matriz plana.

Sintaxe

```
flatten(arrayOfArrays)
```

Argumentos

arrayOfArrays

Uma matriz de matrizes.

Tipo de retorno

A função FLATTEN retorna uma matriz.

Exemplo

Neste exemplo, a entrada é uma matriz aninhada com duas matrizes internas e a saída é uma única matriz plana contendo todos os elementos das matrizes internas. A função FLATTEN pega a matriz aninhada `[[1, 2], [3, 4]]` e combina todos os elementos em uma única matriz. `[1, 2, 3, 4]`

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

Expressões condicionais

No SQL, expressões condicionais são usadas para tomar decisões com base em determinadas condições. Eles permitem que você controle o fluxo de suas instruções SQL e retorne valores diferentes ou execute ações diferentes com base na avaliação de uma ou mais condições.

AWS Clean Rooms suporta as seguintes expressões condicionais:

Tópicos

- Expressão condicional CASE
- expressão COALESCE
- MAIOR e MENOR expressão
- Expressão IF

- expressão IS_NULL
- expressão IS_NOT_NULL
- Funções NVL e COALESCE
- NVL2 função
- Função NULLIF

Expressão condicional CASE

A expressão CASE é uma expressão condicional, semelhante às if/then/else declarações encontradas em outras linguagens. CASE é usada para especificar um resultado onde há várias condições. Use CASE onde uma expressão SQL é válida, como em um comando SELECT.

Há dois tipos de expressões CASE: simples e pesquisada.

- Em expressões CASE simples, uma expressão é comparada a um valor. Quando uma correspondência é encontrada, a ação especificada na cláusula THEN é aplicada. Se nenhuma correspondência é encontrada, a ação especificada na cláusula ELSE é aplicada.
- Em expressões CASE pesquisadas, cada CASE é avaliado com base em uma expressão booleana e a instrução CASE retorna o primeiro CASE correspondente. Se nenhuma correspondência for encontrada entre as cláusulas WHEN, a ação na cláusula ELSE será retornada.

Sintaxe

Instrução CASE simples usada para correspondência de condições:

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

Instrução CASE pesquisada usada para avaliação de cada condição:

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

```
END
```

Argumentos

expressão

Um nome de coluna ou qualquer expressão válida.

value

Valor ao qual a expressão é comparada, tal como uma constante numérica ou string de caracteres.

resultado

O valor ou uma expressão de destino retornado quando uma expressão ou condição booleana é avaliada. Os tipos de dados de todas as expressões de resultados devem poder ser convertidos em um único tipo de saída.

condição

Uma expressão booliana que avalia como verdadeiro ou falso. Se a condição for verdadeira, o valor da expressão CASE será o resultado que segue a condição e o restante da expressão CASE não será processado. Se a condição for falsa, todas as cláusulas WHEN subsequentes serão avaliadas. Se nenhum resultado da condição WHEN for verdadeiro, o valor da expressão CASE será o resultado da cláusula ELSE. Se a cláusula ELSE for omitida e não nenhuma condição for verdadeira, o resultado será nulo.

Exemplos

Use uma expressão CASE simples para substituir `New York City` por `Big Apple` em uma consulta da tabela VENUE. Substitua todos os outros nomes de cidade 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
...
```

Use uma expressão CASE pesquisada para atribuir números de grupo com base no valor PRICEPAID para vendas individuais de ingresso:

```
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
...
```

expressão COALESCE

Uma expressão COALESCE retorna o valor da primeira expressão da lista que não é nula. Se todas as expressões forem nulas, o resultado será nulo. Quando um valor não nulo é localizado, as demais expressões na lista não são avaliadas.

Este tipo de expressão é útil quando você deseja retornar um valor de backup para algo quando o valor preferido está ausente ou é nulo. Por exemplo, uma consulta pode retornar um de três números de telefone (celular, residência ou comercial, nessa ordem), o que for localizado primeiro na tabela (não nulo).

Sintaxe

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

Exemplos

Aplique a expressão COALESCE em duas colunas.

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

O nome da coluna padrão para uma expressão NVL é COALESCE. A consulta a seguir retorna os mesmos resultados.

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

MAIOR e MENOR expressão

Retorna o maior ou menor valor de uma lista com qualquer número de expressões.

Sintaxe

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

Parâmetros

expression_list

Uma lista de expressões, tais como nomes de coluna, separadas por vírgula. As expressões devem ser conversíveis para um tipo de dados comum. Valores NULL na lista são ignorados. Se todas as expressões avaliarem para NULL, o resultado será NULL.

Retornos

Retorna o maior (para GREATEST) ou menor (para LEAST) valor da lista de expressões fornecida.

Exemplo

O seguinte exemplo retorna o valor mais alto alfabeticamente para `firstname` ou `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)
```

Expressão IF

A função condicional IF retorna um dos dois valores com base em uma condição.

Essa função é uma instrução de fluxo de controle comum usada em SQL para tomar decisões e retornar valores diferentes com base na avaliação de uma condição. É útil para implementar uma lógica if-else simples em uma consulta.

Sintaxe

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

Argumentos

expr 1

A condição ou expressão que é avaliada. Se for`true`, a função retornará o valor de expr2. Se expr1 for*`false`*, a função retornará o valor de expr3.

expr 2

A expressão que é avaliada e retornada se expr1 for. `true`

expr 3

A expressão que é avaliada e retornada se expr1 for. `false`

Retornos

Se for `expr1` avaliado como`true`, então retorna`expr2`; caso contrário, retorna`expr3`.

Exemplo

O exemplo a seguir usa a `if()` função para retornar um dos dois valores com base em uma condição. A condição que está sendo avaliada é`1 < 2`, ou seja`true`, o primeiro valor `'a'` é retornado.

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

expressão IS_NULL

A expressão IS_NULL condicional é usada para verificar se um valor é nulo.

Essa expressão é sinônimo de`IS NULL`.

Sintaxe

```
is_null(expr)
```

Argumentos

expr

Uma expressão de qualquer tipo.

Retornos

A expressão IS_NULL condicional retorna um booleano. Se `expr1` for NULL, retorna, caso contrário`true`, retorna`false`.

Exemplos

O exemplo a seguir verifica se o valor 1 é nulo e retorna o resultado booleano `true` porque 1 é um valor válido e não nulo.

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

```
 true
```

O exemplo a seguir seleciona a `id` coluna da `squirrels` tabela, mas somente para as linhas em que a coluna de idade está`null`.

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

expressão IS_NOT_NULL

A expressão IS_NOT_NULL condicional é usada para verificar se um valor não é nulo.

Essa expressão é sinônimo deIS NOT NULL.

Sintaxe

```
is_not_null(expr)
```

Argumentos

expr

Uma expressão de qualquer tipo.

Retornos

A expressão IS_NOT_NULL condicional retorna um booleano. Se não `expr1` for NULL, retorna, caso contrário`true`, retorna`false`.

Exemplos

O exemplo a seguir verifica se o valor não 1 é nulo e retorna o resultado booleano `true` porque 1 é um valor válido e não nulo.

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

O exemplo a seguir seleciona a `id` coluna da `squirrels` tabela, mas somente para as linhas em que a coluna de idade não `null` está.

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

Funções NVL e COALESCE

Retorna o valor da primeira expressão não nula em uma série de expressões. Quando um valor não nulo é encontrado, as demais expressões na lista não são avaliadas.

NVL é idêntica a COALESCE. São funções sinônimas. Este tópico explica a sintaxe e apresenta exemplos de ambas.

Sintaxe

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

A sintaxe de COALESCE é a mesma:

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

Se todas as expressões forem nulas, o resultado será nulo.

Essas funções são úteis para retornar um valor secundário quando um valor primário está ausente ou é nulo. Por exemplo, uma consulta pode retornar o primeiro dos três números de telefone disponíveis: celular, residencial ou profissional. A ordem das expressões na função determina a ordem de avaliação.

Argumentos

expressão

Uma expressão, tal como um nome de coluna, a ser avaliada quanto ao status nulo.

Tipo de retorno

AWS Clean Rooms determina o tipo de dados do valor retornado com base nas expressões de entrada. Se os tipos de dados das expressões de entrada não tiverem um tipo comum, um erro será retornado.

Exemplos

Se a lista contiver expressões do tipo inteiro, a função retornará um inteiro.

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

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

Esse exemplo, que é igual ao exemplo anterior, exceto pelo fato de usar NVL, retorna o mesmo resultado.

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

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

O exemplo a seguir retorna um tipo string.

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

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

O exemplo a seguir resulta em um erro porque os tipos de dados variam na lista de expressões. Nesse caso, há uma string e um número na lista.

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

NVL2 função

Retorna um de dois valores, dependendo se uma expressão especificada avalia para NULL ou NOT NULL.

Sintaxe

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

Argumentos

expressão

Uma expressão, tal como um nome de coluna, a ser avaliada quanto ao status nulo.

not_null_return_value

O valor retornado se a expressão avaliar para NOT NULL. O valor not_null_return_value deve ter o mesmo tipo de dados que a expressão ou ser implicitamente conversível para esse tipo de dados.

null_return_value

O valor retornado se a expressão avaliar para NULL. O valor null_return_value deve ter o mesmo tipo de dados que a expressão ou ser implicitamente conversível para esse tipo de dados.

Tipo de retorno

O tipo de NVL2 retorno é determinado da seguinte forma:

- Se not_null_return_value ou null_return_value for nulo, o tipo de dados da expressão não nula será retornado.

Se not_null_return_value e null_return_value não forem nulos:

- Se not_null_return_value e null_return_value tiverem o mesmo tipo de dados, esse tipo de dados será retornado.
- Se not_null_return_value e null_return_value tiverem diferentes tipos de dados numéricos, o menor tipo de dados numérico compatível será retornado.
- Se not_null_return_value e null_return_value tiverem diferentes tipos de dados datetime, um tipo de dados de timestamp será retornado.
- Se not_null_return_value e null_return_value tiverem diferentes tipos de dados de caracteres, o tipo de dados de not_null_return_value será retornado.
- Se not_null_return_value e null_return_value tiverem tipos de dados numéricos e não numéricos variados, o tipo de dados de not_null_return_value será retornado.

> **Important**
>
> Nos últimos dois casos onde o tipo de dados de not_null_return_value é retornado, o null_return_value é convertido implicitamente para esse tipo de dados. Se os tipos de dados forem incompatíveis, a função falhará.

Observações de uso

Pois NVL2, o retorno terá o valor do parâmetro not_null_return_value ou null_return_value, o que for selecionado pela função, mas terá o tipo de dados not_null_return_value.

Por exemplo, supondo que column1 seja NULL, as consultas seguintes retornarão o mesmo valor. No entanto, o tipo de dados do valor de retorno DECODE será INTEGER e o tipo de dados do valor de NVL2 retorno será VARCHAR.

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

Exemplo

O seguinte exemplo altera alguns dados de amostra e, então, avalia dois campos para fornecer as informações de contato apropriadas para usuários:

```
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
```

Função NULLIF

Compara dois argumentos e retorna nulo se os argumentos forem iguais. Se eles não forem iguais, o primeiro argumento será retornado.

Sintaxe

A expressão NULLIF compara dois argumentos e retorna nulo se os argumentos forem iguais. Se eles não forem iguais, o primeiro argumento será retornado. Essa expressão é o inverso da expressão NVL ou COALESCE.

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

Argumentos

expression1, expression2

As colunas ou expressões de destino que são comparadas. O tipo de retorno é igual ao tipo da primeira expressão.

Exemplos

No exemplo a seguir, a consulta retorna a string `first` porque os argumentos não são iguais.

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

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

No exemplo a seguir, a consulta retorna NULL porque os argumentos literais da string são iguais.

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

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

No exemplo a seguir, a consulta retorna 1 porque os argumentos inteiros não são iguais.

```
SELECT NULLIF(1, 2);

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

No exemplo a seguir, a consulta retorna NULL porque os argumentos inteiros são iguais.

```
SELECT NULLIF(1, 1);

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

No exemplo a seguir, a consulta retorna nulo quando há correspondência dos valores LISTID e SALESID:

```
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)
```

Funções do construtor

Uma função construtora SQL é uma função usada para criar novas estruturas de dados, como matrizes ou mapas.

Eles pegam alguns valores de entrada e retornam um novo objeto de estrutura de dados. As funções do construtor geralmente são nomeadas de acordo com o tipo de dados que elas criam, como ARRAY ou MAP.

As funções construtoras são diferentes das funções escalares ou agregadas, que operam com dados existentes e retornam um único valor. As funções do construtor são usadas para criar novas estruturas de dados que podem ser usadas em processamento ou análise de dados adicionais.

AWS Clean Rooms suporta as seguintes funções de construtor:

Tópicos

Função construtora MAP

A função construtora MAP cria um mapa com os pares chave/valor fornecidos.

Funções de construtor, como MAP, são úteis quando você precisa criar novas estruturas de dados programaticamente em suas consultas SQL. Eles permitem que você crie estruturas de dados complexas que podem ser usadas em processamento ou análise de dados adicionais.

Sintaxe

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

Argumentos

chave 0

Uma expressão de qualquer tipo comparável. Todas as key0 devem compartilhar um tipo menos comum.

valor0

Uma expressão de qualquer tipo. Todos os ValueN devem compartilhar um tipo menos comum.

Retornos

A função MAP retorna um MAP com chaves digitadas como o tipo menos comum de chave0 e valores digitados como o tipo menos comum de valor0.

Exemplos

O exemplo a seguir cria um novo mapa com dois pares de valores-chave: A chave `1.0` está associada ao valor. `'2'` A chave `3.0` está associada ao valor`'4'`. O mapa resultante é então retornado como a saída da instrução SQL.

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

Função construtora NAMED_STRUCT

A função do construtor NAMED_STRUCT cria uma estrutura com os nomes e valores de campo fornecidos.

Funções de construtor como NAMED_STRUCT são úteis quando você precisa criar novas estruturas de dados programaticamente em suas consultas SQL. Eles permitem que você crie estruturas de dados complexas, como estruturas ou registros, que podem ser usadas em processamento ou análise de dados adicionais.

Sintaxe

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

Argumentos

nome1

Um campo de nomenclatura literal STRING 1.

val1

Uma expressão de qualquer tipo especificando o valor do campo 1.

Retornos

A função NAMED_STRUCT retorna uma estrutura com o campo 1 correspondente ao tipo de val1.

Exemplos

O exemplo a seguir cria uma nova estrutura com três campos nomeados: O valor `"a"` 1 é atribuído ao campo. O campo `"b"` recebe o valor `2`. O campo `"c"` recebe o valor`3`. A estrutura resultante é então retornada como saída da instrução SQL.

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

Função construtora STRUCT

A função construtora STRUCT cria uma estrutura com os valores de campo fornecidos.

Funções de construtor como STRUCT são úteis quando você precisa criar novas estruturas de dados programaticamente em suas consultas SQL. Eles permitem que você crie estruturas de dados complexas, como estruturas ou registros, que podem ser usadas em processamento ou análise de dados adicionais.

Sintaxe

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

Argumentos

col1

Um nome de coluna ou qualquer expressão válida.

Retornos

A função STRUCT retorna uma estrutura com field1 correspondente ao tipo de expr1.

Se os argumentos forem referências nomeadas, os nomes serão usados para nomear o campo. Caso contrário, os campos são denominados colN, onde N é a posição do campo na estrutura.

Exemplos

O exemplo a seguir cria uma nova estrutura com três campos: O primeiro campo recebe o valor 1. O segundo campo recebe o valor 2. O terceiro campo recebe o valor 3. Por padrão, os campos na estrutura resultante são nomeados`col1`,, e `col2``col3`, com base em sua posição na lista de argumentos. A estrutura resultante é então retornada como saída da instrução SQL.

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

Funções de formatação de tipo de dados

Usando uma função de formatação de tipo de dados, você pode converter valores de um tipo de dados para outro. Para cada uma dessas funções, o primeiro argumento é sempre o valor a ser formatado e o segundo argumento contém o modelo para o novo formato.

AWS Clean Rooms O Spark SQL oferece suporte a várias funções de formatação de tipos de dados.

Tópicos

BASE64 função

A BASE64 função converte uma expressão em uma string de base 64 usando a codificação de transferência RFC2 045 Base64 para MIME.

Sintaxe

```
base64(expr)
```

Argumentos

expr

Uma expressão BINÁRIA ou uma STRING que a função interpretará como BINÁRIA.

Tipo de retorno

`STRING`

Exemplo

Para converter a entrada de string fornecida em sua representação codificada em Base64. use o exemplo a seguir. O resultado é a representação codificada em Base64 da string de entrada 'Spark SQL', que é 'u3bhcmsgu1fm'.

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

Função CAST

A função CAST converte um tipo de dados em outro compatível. Por exemplo, é possível converter uma string em uma data ou um tipo numérico em uma string. CAST executa uma conversão em tempo de execução, o que significa que a conversão não altera o tipo de dados de um valor em uma tabela de origem. Isso é alterado somente no contexto da consulta.

Alguns tipos de dados exigem uma conversão explícita para outros tipos de dados usando a função CAST. Outros tipos de dados podem ser convertidos implicitamente, como parte de outro comando, sem usar CAST. Consulte Compatibilidade e conversão dos tipos.

Sintaxe

Use uma dessas duas formas equivalentes de sintaxe para transmitir expressões de um tipo de dados para outro.

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

Argumentos

expressão

Uma expressão que avalia para um ou mais valores, tal como um nome de coluna ou um literal. A conversão de valores nulos retorna nulos. A expressão não pode conter strings em branco ou vazias.

tipo

Um dos suportadosTipos de dados, exceto para os tipos de dados BINARY e BINARY VARYING.

Tipo de retorno

CAST retorna o tipo de dados especificado pelo argumento type.

> **Note**
>
> AWS Clean Rooms retorna um erro se você tentar realizar uma conversão problemática, como uma conversão DECIMAL que perde a precisão, como a seguir:
>
> ```
> select 123.456::decimal(2,1);
> ```
>
> ou uma conversão INTEGER que causa um transbordamento:
>
> ```
> select 12345678::smallint;
> ```

Exemplos

As seguintes duas consultas são equivalentes. Ambas convertem um valor decimal em um número inteiro:

```
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)
```

O seguinte produz um resultado semelhante. Ele não exige dados de exemplo para ser executado:

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

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

Neste exemplo, os valores em uma coluna de carimbo de data/hora são transmitidos como datas, o que resulta na remoção do horário 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)
```

Se você não usasse CAST conforme ilustrado no exemplo anterior, os resultados incluiriam o horário: 2008-02-18 02:36:48.

A consulta a seguir converte dados de caracteres variáveis em uma data. Ele não exige dados de exemplo para ser executado.

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

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

Neste exemplo, os valores em uma coluna de data são convertidos como timestamps:

```
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)
```

Em um caso como no exemplo anterior, é possível ter controle adicional sobre a formatação de saída usando TO_CHAR.

Neste exemplo, um número inteiro é convertido como uma string de caracteres:

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

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

Neste exemplo, um valor DECIMAL(6,3) é convertido como um valor DECIMAL(4,1):

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

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

Este exemplo mostra uma expressão mais complexa. A coluna PRICEPAID (uma coluna DECIMAL(8,2)) na tabela SALES é convertida em uma coluna DECIMAL(38,2) e os valores são multiplicados por 100.000.000.000.000.000.000.

```
select salesid, pricepaid::decimal(38,2)*10000000000000000000
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)
```

Função DECODE

A função DECODE é a contrapartida da função ENCODE, que é usada para converter uma string em um formato binário usando uma codificação de caracteres específica. A função DECODE pega os dados binários e os converte novamente em um formato de string legível usando a codificação de caracteres especificada.

Essa função é útil quando você precisa trabalhar com dados binários armazenados em um banco de dados e apresentá-los em um formato legível por humanos ou quando precisa converter dados entre diferentes codificações de caracteres.

Sintaxe

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

Argumentos

expr

Uma expressão BINÁRIA codificada em charset.

conjunto de caracteres

Uma expressão STRING.

Codificações de conjuntos de caracteres compatíveis (sem distinção entre maiúsculas e minúsculas):`'US-ASCII'`,,`'ISO-8859-1'`,`'UTF-8'`, e. `'UTF-16BE' 'UTF-16LE' 'UTF-16'`

Tipo de retorno

A função DECODE retorna uma STRING.

Exemplo

O exemplo a seguir tem uma tabela chamada `messages` com uma coluna chamada `message_text` que armazena dados de mensagens em formato binário usando a codificação de caracteres UTF-8. A função DECODE converte os dados binários de volta em um formato de string legível. A saída dessa consulta é o texto legível da mensagem armazenada na tabela de mensagens, com o ID`123`, convertido do formato binário em uma string usando a `'utf-8'` codificação.

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

Função ENCODE

A função ENCODE é usada para converter uma string em sua representação binária usando uma codificação de caracteres especificada.

Essa função é útil quando você precisa trabalhar com dados binários ou quando precisa converter entre diferentes codificações de caracteres. Por exemplo, você pode usar a função ENCODE ao armazenar dados em um banco de dados que requer armazenamento binário ou quando precisar transferir dados entre sistemas que usam codificações de caracteres diferentes.

Sintaxe

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

Argumentos

str

Uma expressão STRING a ser codificada.

conjunto de caracteres

Uma expressão STRING especificando a codificação.

Codificações de conjuntos de caracteres compatíveis (sem distinção entre maiúsculas e minúsculas):`'US-ASCII',,'ISO-8859-1','UTF-8'`, e. `'UTF-16BE' 'UTF-16LE' 'UTF-16'`

Tipo de retorno

A função ENCODE retorna um BINÁRIO.

Exemplo

O exemplo a seguir converte `'abc'` a string em sua representação binária usando a `'utf-8'` codificação, o que, nesse caso, resulta no retorno da string original. Isso ocorre porque a `'utf-8'` codificação é uma codificação de caracteres de largura variável que pode representar todo o conjunto de caracteres ASCII (que inclui as letras `'a''b'`, `e'c'`) usando um único byte por caractere. Portanto, a representação binária do `'abc'` uso `'utf-8'` é a mesma da string original.

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

Função HEX

A função HEX converte um valor numérico (um número inteiro ou um número de ponto flutuante) em sua representação de string hexadecimal correspondente.

O hexadecimal é um sistema numérico que usa 16 símbolos distintos (0-9 e A-F) para representar valores numéricos. É comumente usado em ciência da computação e programação para representar dados binários em um formato mais compacto e legível por humanos.

Sintaxe

```
hex(expr)
```

Argumentos

expr

Uma expressão BIGINT, BINARY ou STRING.

Tipo de retorno

HEX retorna uma STRING. A função retorna a representação hexadecimal do argumento.

Exemplo

O exemplo a seguir usa o valor inteiro 17 como entrada e aplica a função HEX () a ele. A saída é11, que é a representação hexadecimal do valor de entrada. 17

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

O exemplo a seguir converte a string em sua `'Spark_SQL'` representação hexadecimal. A saída é537061726B2053514C, que é a representação hexadecimal da string de entrada. `'Spark_SQL'`

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

Neste exemplo, a string 'Spark_SQL' é convertida da seguinte forma:

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

- 'L' -> 4C

A concatenação desses valores hexadecimais resulta na saída final ". 537061726B2053514C"

função STR_TO_MAP

A função STR_TO_MAP é uma função de conversão. string-to-map Ele converte uma representação em cadeia de caracteres de um mapa (ou dicionário) em uma estrutura de dados real do mapa.

Essa função é útil quando você precisa trabalhar com estruturas de dados de mapas em SQL, mas os dados são inicialmente armazenados como uma string. Ao converter a representação da string em um mapa real, você pode então realizar operações e manipulações nos dados do mapa.

Sintaxe

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

Argumentos

texto

Uma expressão STRING que representa o mapa.

Par de LIM

Um literal STRING opcional que especifica como separar as entradas. O padrão é uma vírgula (). `','`

keyValueDelim

Um literal STRING opcional que especifica como separar cada par de valores-chave. O padrão é dois pontos (). `':'`

Tipo de retorno

A função STR_TO_MAP retorna um MAP de STRING para chaves e valores. Tanto o PairDelim quanto o PairDelim keyValueDelimsão tratados como expressões regulares.

Exemplo

O exemplo a seguir usa a string de entrada e os dois argumentos delimitadores e converte a representação da string em uma estrutura de dados de mapa real. Neste exemplo específico, a

string de entrada `'a:1,b:2,c:3'` representa um mapa com os seguintes pares de valores-chave: `'a'` é a chave e `'1'` é o valor. `'b'`é a chave e `'2'` é o valor. `'c'`é a chave e `'3'` é o valor. O `','` delimitador é usado para separar os pares de valores-chave e o `':'` delimitador é usado para separar a chave e o valor em cada par. A saída dessa consulta é:`{"a":"1","b":"2","c":"3"}`. Essa é a estrutura de dados do mapa resultante, onde as chaves são `'a''b','c'`, e, e os valores correspondentes são `'1''2'`, `'3'` e.

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

O exemplo a seguir demonstra que a função STR_TO_MAP espera que a string de entrada esteja em um formato específico, com os pares de valores-chave delimitados corretamente. Se a string de entrada não corresponder ao formato esperado, a função ainda tentará criar um mapa, mas os valores resultantes podem não ser os esperados.

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

TO_CHAR

TO_CHAR Converte uma expressão de timestamp ou numérica para o formato de dados de string de caracteres.

Sintaxe

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

Argumentos

timestamp_expression

Uma expressão que resulta em um valor do tipo TIMESTAMP ou TIMESTAMPTZ ou um valor que pode ser implicitamente forçado para um timestamp.

numeric_expression

Uma expressão que resulta em um valor de tipo de dados numérico ou em um valor que pode implicitamente ser convertido para tipo numérico. Para obter mais informações, consulte Tipos numéricos. TO_CHAR insere um espaço à esquerda da string numérica.

> **Note**
>
> TO_CHAR não permite valores do tipo DECIMAL de 128 bits.

format

O formato para o novo valor. Para obter os formatos válidos, consulte Strings de formato datetime e Strings de formato numérico.

Tipo de retorno

VARCHAR

Exemplos

O exemplo a seguir converte um carimbo de data/hora em um valor com a data e a hora em um formato com o nome do mês preenchido com nove caracteres, o nome do dia da semana e o número do dia do mês.

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

O exemplo a seguir converte um carimbo de data/hora em um valor com o número do dia do ano.

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

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

O exemplo a seguir converte um carimbo de data/hora em um número do dia da semana da norma ISO.

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

to_char
-------------------------
```

```
1
```

O exemplo a seguir extrai o nome do mês de uma data.

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

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

O seguinte exemplo converte cada valor de STARTTIME na tabela EVENT em uma string que consiste em horas, minutos e 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)
```

O exemplo a seguir converte um valor de timestamp inteiro em um 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)
```

O exemplo a seguir converte um literal de timestamp em uma string de caracteres.

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

```
23:15:59
(1 row)
```

O exemplo a seguir converte um número em uma string de caracteres com o sinal menos no final.

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

O exemplo a seguir converte um número em uma string de caracteres com o símbolo de moeda.

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

O exemplo a seguir converte um número em uma string de caracteres usando colchetes angulares para números negativos.

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

O exemplo a seguir converte um número em uma string de numerais romanos.

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

O exemplo a seguir exibe o dia da semana.

```
SELECT to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS');
               to_char
```

```
-----------------------
Wednesday, 31 09:34:26
```

O exemplo a seguir exibe o sufixo de número ordinal de um número.

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

O exemplo a seguir subtrai a comissão do preço pago na tabela de vendas. A diferença é então arredondada para cima e convertida em um numeral romano, exibido na coluna `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)
```

O seguinte exemplo adiciona o cifrão aos valores da diferença exibidos na coluna `to_char`:

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'l99999D99') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;

salesid | pricepaid | commission | difference |  to_char
--------+-----------+------------+------------+------------
      1 |    728.00 |     109.20 |     618.80 | $   618.80
```

```
      2 |     76.00 |      11.40 |      64.60 | $    64.60
      3 |    350.00 |      52.50 |     297.50 | $   297.50
      4 |    175.00 |      26.25 |     148.75 | $   148.75
      5 |    154.00 |      23.10 |     130.90 | $   130.90
      6 |    394.00 |      59.10 |     334.90 | $   334.90
      7 |    788.00 |     118.20 |     669.80 | $   669.80
      8 |    197.00 |      29.55 |     167.45 | $   167.45
      9 |    591.00 |      88.65 |     502.35 | $   502.35
     10 |     65.00 |       9.75 |      55.25 | $    55.25
(10 rows)
```

O seguinte exemplo lista o século em que cada venda foi efetuada.

```
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)
```

O seguinte exemplo converte cada valor de STARTTIME na tabela EVENT em uma string que consiste em horas, minutos, segundos e fuso horário:

```
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)
```

O seguinte exemplo exibe a formatação para segundos, milissegundos e microssegundos.

```
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
```

Função TO_DATE

TO_DATE converte uma data representada em uma string de caracteres para um tipo de dados DATE.

Sintaxe

```
TO_DATE (date_str)
```

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

Argumentos

data_str

Uma string de data ou um tipo de dados que pode ser convertido em uma string de data.

format

Uma string literal que corresponde aos padrões de data e hora do Spark. Para padrões de data e hora válidos, consulte Padrões de data e hora para formatação e análise.

Tipo de retorno

TO_DATE retorna uma DATE, dependendo do valor do format.

Se ocorrer falha na conversão no formato, um erro será gerado.

Exemplos

A instrução SQL a seguir converte a data `02 Oct 2001` em um tipo de dados de data.

```
select to_date('02 Oct 2001', 'dd MMM yyyy');

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

A instrução SQL a seguir converte a string `20010631` em uma data.

```
select to_date('20010631', 'yyyyMMdd');
```

A seguinte instrução SQL converte a string `20010631` em uma data:

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

O resultado é um valor nulo porque há apenas 30 dias em junho.

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

TO_NUMBER

TO_NUMBER converte uma string em um valor numérico (decimal).

Sintaxe

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

Argumentos

string

String a ser convertida. O formato deve ser um valor literal.

format

O segundo argumento é uma string de formato que indica como a string de caracteres deve ser analisada para criar o valor numérico. Por exemplo, o formato `'99D999'` especifica que a string a ser convertida consiste em cinco dígitos com o ponto decimal na terceira posição. Por exemplo, `to_number('12.345','99D999')` retorna `12.345` como um valor numérico. Para obter uma lista dos formatos válidos, consulte Strings de formato numérico.

Tipo de retorno

TO_NUMBER retorna um número DECIMAL.

Se ocorrer falha na conversão no formato, um erro será gerado.

Exemplos

O exemplo a seguir converte a string `12,454.8-` em um número:

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

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

O exemplo a seguir converte a string `$ 12,454.88` em um número:

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

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

O exemplo a seguir converte a string `$ 2,012,454.88` em um número:

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

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

UNBASE64 função

A UNBASE64 função converte um argumento de uma string de base 64 em um binário.

A codificação Base64 é comumente usada para representar dados binários (como imagens, arquivos ou informações criptografadas) em um formato textual seguro para transmissão por vários canais de comunicação (como e-mail, parâmetros de URL ou armazenamento de banco de dados).

A UNBASE64 função permite reverter esse processo e recuperar os dados binários originais. Esse tipo de funcionalidade pode ser útil em cenários em que você precisa trabalhar com dados codificados no formato Base64, como na integração com sistemas externos ou APIs que usam o Base64 como mecanismo de transferência de dados.

Sintaxe

```
unbase64(expr)
```

Argumentos

expr

Uma expressão STRING no formato base64.

Tipo de retorno

`BINARY`

Exemplo

No exemplo a seguir, a string codificada em Base64 `'U3BhcmsgU1FM'` é convertida novamente na string original. `'Spark SQL'`

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

Função UNHEX

A função UNHEX converte uma string hexadecimal de volta à sua representação de string original.

Essa função pode ser útil em cenários em que você precisa trabalhar com dados que foram armazenados ou transmitidos em formato hexadecimal e restaurar a representação da string original para processamento ou exibição adicionais.

A função UNHEX é a contrapartida da função HEX.

Sintaxe

```
unhex(expr)
```

Argumentos

expr

Uma expressão STRING de caracteres hexadecimais.

Tipo de retorno

UNHEX retorna um BINÁRIO.

Se o comprimento de expr for ímpar, o primeiro caractere será descartado e o resultado será preenchido com um byte nulo. Se expr contiver caracteres não hexadecimais, o resultado será NULL.

Exemplo

O exemplo a seguir converte uma string hexadecimal de volta à sua representação de string original usando as funções UNHEX () e DECODE () juntas. A primeira parte da consulta usa a função UNHEX () para converter a string hexadecimal '537061726B2053514C' em sua representação binária. A segunda parte da consulta usa a função DECODE () para converter os dados binários obtidos da função UNHEX () em uma string, usando a codificação de caracteres 'UTF-8'. A saída da consulta é a string original 'Spark_SQL' que foi convertida em hexadecimal e depois novamente em uma string.

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

Strings de formato datetime

Você pode usar padrões de data e hora nos seguintes cenários comuns:

- Ao trabalhar com fontes de dados CSV e JSON para analisar e formatar conteúdo de data e hora
- Ao converter entre tipos de string e tipos de data ou timestamp usando funções como:
  - unix_timestamp
  - date_format
  - carimbo de data/hora to_unix_
  - from_unixtime
  - to_date
  - to_timestamp
  - from_utc_timestamp
  - to_utc_timestamp

Use as letras padrão na tabela a seguir para análise e formatação de data e hora.

Datepart ou timepart	Significado	Exemplos
a	AM ou PM do dia, apresenta do como am-pm	PM
D	Dia do ano, apresentado como um número de 3 dígitos	189
d	Dia do mês, apresentado como um número de 2 dígitos	28
E	Dia da semana, apresentado como texto	Ter Terça-feira
F	Dia da semana alinhado no mês, apresentado como um número de 1 dígito	3
G	Indicador de era, apresentado como texto	AD Anno Domini

Datepart ou timepart	Significado	Exemplos
h	Horário da manhã ou da tarde, apresentado como um número de 2 dígitos	12
H	Hora do dia, apresentada como um número de 2 dígitos de 0 a 23	0
k	Hora do dia, apresentada como um número de 2 dígitos de 1 a 24	1
K	Hora da manhã ou da tarde, apresentada como um número de 2 dígitos de 0 a 11	0
m	Minuto de hora, apresentado como um número de 2 dígitos	30
M/L	Mês do ano, apresentado como um mês	7 07 jul Julho
O	Deslocamento de zona localizado em relação ao UTC	GMT +8 GMT+ 8:00 UTC- 08:00

Datepart ou timepart	Significado	Exemplos
Q/q	Trimestre do ano, apresentado como um número (1 a 4) ou texto	3 03 Q3 3º trimestre
s	Segundo do minuto, apresentado como um número de 2 dígitos	55
S	Fração de segundo, apresentada como uma fração	978
V	Identificador de fuso horário, apresentado como uma identificação de zona	America/Los_Angeles Z 08:30
x	Deslocamento de zona em relação ao UTC (offset-X)	+0000 -08 -0830 - 08:30 -083015 - 08:30:15

Datepart ou timepart	Significado	Exemplos
X	Deslocamento de zona em relação ao UTC; onde Z é zero	Z -08 -0830 - 08:30 -083015 - 08:30:15
y	Ano, apresentado como um ano	2020 20
z	Nome do fuso horário, apresentado como texto	Hora Oficial do Pacífico PST
Z	Deslocamento de zona em relação ao UTC (offset-Z)	+0000 -0800 - 08:00
'	Escape para texto, apresenta do como um delimitador	N/D
''	Citação única, apresentada como literal	'
[	Início da seção opcional	N/D
]	Fim de seção opcional	N/D

O número de letras padrão determina o tipo de formato:

Formato de texto

- Use de 1 a 3 letras para a forma abreviada (por exemplo, “Mon” para segunda-feira)
- Use exatamente 4 letras para o formulário completo (por exemplo, “segunda-feira”)
- Não use 5 ou mais letras - isso causará um erro

Formato numérico (n)

- O valor n representa o número máximo de letras permitido
- Para padrões de letra única:
  - A saída usa dígitos mínimos sem preenchimento
- Para vários padrões de letras:
  - A saída é preenchida com zeros para corresponder à largura da contagem de letras
- Ao analisar, a entrada deve conter o número exato de dígitos

Formato de número/texto

- Para 3 ou mais letras, siga as regras de formato de texto
- Para menos letras, siga as regras de formato numérico

Formato de fração

- Use de 1 a 9 caracteres 'S' (por exemplo, SSSSSS)
- Para análise:
  - Aceite frações entre 1 e o número de caracteres S
- Para formatação:
  - Almofada com zeros para corresponder ao número de caracteres S
- Suporta até 6 dígitos para precisão de microssegundos
- Pode analisar nanossegundos, mas trunca dígitos extras

Formato do ano

- A contagem de letras define a largura mínima do campo para preenchimento
- Para duas letras:
  - Imprime os dois últimos dígitos

  - Analisa anos entre 2000 e 2099
- Por menos de quatro letras (exceto duas):
  - Mostra o sinal apenas para anos negativos
- Não use 7 ou mais letras - isso causará um erro

Formato do mês

- Use 'M' para o formulário padrão ou 'L' para o formulário autônomo
- Único 'M' ou 'L':
  - Mostra os números dos meses de 1 a 12 sem preenchimento

- 'MM' ou 'LL':
  - Mostra os números dos meses de 01 a 12 com preenchimento
- 'MMM':
  - Mostra o nome abreviado do mês no formato padrão
  - Deve fazer parte de um padrão de data completo
- 'LLL':
  - Mostra o nome abreviado do mês em formato autônomo
  - Use para formatação somente por mês
- 'MMMM':
  - Mostra o nome completo do mês no formato padrão
  - Use para datas e carimbos de data e hora
- 'LLLL':
  - Mostra o nome completo do mês em formato independente
  - Use para formatação somente por mês

Formatos de fuso horário

- am-pm: use apenas 1 letra
- ID de zona (V): use apenas 2 letras
- Nomes de zonas (z):
  - 1-3 letras: mostra o nome curto

- 4 letras: mostra o nome completo
- Não use 5 ou mais letras

Formatos de offset

- X e x:
  - 1 letra: Mostra a hora (+01) ou hora-minuto (+0130)
  - 2 letras: Mostra hora-minuto sem dois pontos (+0130)
  - 3 letras: mostra hora-minuto com dois pontos (+ 01:30)
  - 4 letras: mostra hour-minute-second sem dois pontos (+013015)
  - 5 letras: mostra hour-minute-second com dois pontos (+ 01:30:15)
  - X usa 'Z' para compensação zero
  - x usa '+00', '+0000' ou '+ 00:00 'para compensação zero
- U:
  - 1 letra: mostra a forma abreviada (GMT+8)
  - 4 letras: Mostra o formulário completo (GMT+ 08:00)
- Z:
  - 1-3 letras: Mostra hora-minuto sem dois pontos (+0130)
  - 4 letras: Mostra a forma localizada completa
  - 5 letras: mostra hour-minute-second com dois pontos

Seções opcionais

- Use colchetes [] para marcar conteúdo opcional
- Você pode aninhar seções opcionais
- Todos os dados válidos aparecem na saída
- A entrada pode omitir seções opcionais inteiras

> **Note**
>
> Os símbolos 'E', 'F', 'q' e 'Q' funcionam somente para formatação de data e hora (como date_format). Não os use para análise de data e hora (como to_timestamp).

Strings de formato numérico

As sequências de formato numérico a seguir se aplicam a funções como TO_NUMBER e TO_CHAR.

- Para obter exemplos de strings de formatação como números, consulte TO_NUMBER.
- Para obter exemplos de números de formatação como strings, consulte TO_CHAR.

Formato	Description
9	Valor numérico com o número especificado de dígitos.
0	Valor numérico com zeros iniciais.
. (ponto final), D	Ponto decimal.
, (vírgula)	Separador de milhares.
CC	Código de século. Por exemplo, o século 21 começou em 2001-01-01 (compatível somente com TO_CHAR).
FM	Modo de preenchimento. Excluir espaços em branco e zeros.
PR	Valor negativo entre colchetes angulares.
S	Sinal ancorado a um número.
L	Símbolo de cifrão na posição especificada.
G	Separador de grupo.
MI	Sinal de menos na posição especificada para números menores que 0.
PL	Sinal de mais na posição especificada para números maiores que 0.

Formato	Description
SG	Sinal de mais ou menos na posição especific ada.
RN	Numeral romano entre 1 e 3.999 (compatível somente com TO_CHAR).
TH ou th	Sufixo de número ordinal. Não converte números ou valores fracionários menores que zero.

Perfis de data e hora

As funções de data e hora permitem que você execute uma ampla variedade de operações em dados de data e hora, como extrair partes de uma data, realizar cálculos de data, formatar datas e horas e trabalhar com a data e a hora atuais. Essas funções são essenciais para tarefas como análise de dados, relatórios e manipulação de dados envolvendo dados temporais.

AWS Clean Rooms suporta as seguintes funções de data e hora:

Tópicos

- Função EXTRACT
- função FROM_UTC_TIMESTAMP
- Função HOUR
- Função MINUTE
- Função MÊS
- SEGUNDA função
- Função TIMESTAMP
- Função TO_TIMESTAMP
- Função YEAR
- Partes da data para funções de data ou de timestamp

Função ADD_MONTHS

ADD_MONTHS adiciona o número especificado de meses a um valor ou expressão de data ou timestamp. A função DATE_ADD oferece funcionalidade semelhante.

Sintaxe

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

Argumentos

date | timestamp

Uma coluna de data ou timestamp ou uma expressão que converta implicitamente em uma data ou timestamp. Se a data for o último dia do mês ou se o mês resultante for mais curto, a função retorna o último dia do mês nos resultados. Para outras datas, o resultado contém o mesmo número de dia que a expressão de data.

inteiro

Um número inteiro positivo ou negativo. Use um número negativo para subtrair meses de datas.

Tipo de retorno

TIMESTAMP

Exemplo

A seguinte consulta usa a função de ADD_MONTHS dentro de uma função TRUNC. A função TRUNC remove o horário do dia dos resultados de ADD_MONTHS. A função ADD_MONTHS adiciona 12 meses a cada valor da coluna 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)
```

Os seguintes exemplos demonstram o comportamento quando a função ADD_MONTHS opera em datas com meses que têm diferentes número de dias.

```
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)
```

Função CONVERT_TIMEZONE

CONVERT_TIMEZONE converte um timestamp de um fuso horário para outro. A função se ajusta automaticamente para o horário de verão.

Sintaxe

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

Argumentos

source_timezone

(Opcional) O fuso horário do timestamp atual. O padrão é UTC.

target_timezone

O fuso horário do novo timestamp.

timestamp

Uma coluna de timestamp ou uma expressão que converta implicitamente para um timestamp.

Tipo de retorno

TIMESTAMP

Exemplos

O exemplo a seguir converte o valor de carimbo de data/hora do fuso horário UTC padrão em PST.

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

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

O exemplo a seguir converte o valor do timestamp na coluna LISTTIME do fuso horário UTC padrão para PST. Embora o timestamp esteja no período de horário de verão, ele é convertido para o horário padrão, pois o fuso horário de destino é especificado como uma abreviação (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
```

O exemplo a seguir converte uma coluna LISTTIME de carimbo de data/hora do fuso horário UTC padrão em fuso horário. US/Pacific A fuso horário de destino usa um nome de fuso horário e o timestamp está no horário de verão, portanto a função retorna o horário de verão.

```
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
```

O exemplo a seguir converte uma string de timestamp de EST para PST:

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

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

O exemplo a seguir converte um timestamp para o horário padrão do Leste dos EUA, pois o fuso horário de destino usa um nome de fuso horário (America/New_York) e o timestamp está dentro do período de horário padrão.

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

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

O seguinte exemplo converte o timestamp para o horário de verão do Leste dos EUA, pois o fuso horário de destino usa um nome de fuso horário (America/New_York) e o timestamp está dentro do período do horário de verão.

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

 convert_timezone
---------------------
```

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

O seguinte exemplo demonstra o uso de deslocamentos.

```
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)
```

Função CURRENT_DATE

CURRENT_DATE retorna uma data no fuso horário da sessão atual (UTC por padrão) no formato padrão:. YYYY-MM-DD

> **Note**
>
> CURRENT_DATE retorna a data de início para a transação atual, não para o início da instrução atual. Considere o cenário em que você inicia uma transação contendo várias declarações em 10/01/08 23:59, e a declaração contendo CURRENT_DATE é executada em 10/02/08 00:00. CURRENT_DATE retorna `10/01/08`, não `10/02/08`.

Sintaxe

```
CURRENT_DATE
```

Tipo de retorno

DATE

Exemplo

O exemplo a seguir retorna a data atual (no local em Região da AWS que a função é executada).

```
select current_date;
```

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

função CURRENT_TIMESTAMP

CURRENT_TIMESTAMP retorna a data e a hora atuais, incluindo a data, a hora e (opcionalmente) os milissegundos ou microssegundos.

Essa função é útil quando você precisa obter a data e a hora atuais, por exemplo, para registrar a data e hora de um evento, realizar cálculos com base no tempo ou preencher colunas. date/time

Sintaxe

```
current_timestamp()
```

Tipo de retorno

A função CURRENT_TIMESTAMP retorna uma DATA.

Exemplo

O exemplo a seguir retorna a data e a hora atuais no momento em que a consulta é executada, que é 25 de abril de 2020, às 15:49:11.914 (15:49:11.914 PM).

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

O exemplo a seguir recupera a data e a hora atuais de cada linha na `squirrels` tabela.

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

função DATE_ADD

Retorna a data que é num_days após start_date.

Sintaxe

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

Argumentos

data_inicial

O valor da data inicial.

num_dias

O número de dias a serem adicionados (inteiro). Um número positivo soma dias, um número negativo subtrai dias.

Tipo de retorno

DATE

Exemplos

O exemplo a seguir adiciona um dia a uma data:

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

Result:
2016-07-31
```

O exemplo a seguir adiciona vários dias.

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

Result:
2016-08-04
```

Observações de uso

Esta documentação é para a função DATE_ADD do Spark SQL, que fornece uma interface mais simples para adicionar dias às datas em comparação com algumas outras variantes do SQL. Para adicionar outros intervalos, como meses ou anos, funções diferentes podem ser necessárias.

função DATE_DIFF

DATE_DIFF retorna a diferença entre as partes de data de duas expressões de data ou hora.

Sintaxe

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

Argumentos

endDate

Uma expressão DATE.

startDate

Uma expressão DATE.

Tipo de retorno

BIGINT

Exemplos com uma coluna DATE

O exemplo a seguir encontra a diferença, em número de semanas, entre dois valores de data literais.

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

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

O exemplo a seguir encontra a diferença, em horas, entre dois valores de data literais. Quando você não fornece o valor de hora para uma data, o padrão é 00:00:00.

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

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

O exemplo a seguir encontra a diferença, em dias, entre dois valores literais de TIMESTAMETZ.

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

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

O exemplo a seguir encontra a diferença, em dias, entre duas datas na mesma linha de uma tabela.

```
select * from date_table;

start_date |   end_date
-----------+-----------
2009-01-01 | 2009-03-23
2023-01-04 | 2024-05-04
(2 rows)

select date_diff(day, start_date, end_date) as duration from date_table;

duration
---------
      81
     486
(2 rows)
```

O exemplo a seguir encontra a diferença, em número de trimestres, entre um valor literal no passado e a data de hoje. Este exemplo presume que a data atual seja 5 de junho de 2008. Você pode nomear as partes da data por completo ou abreviá-las. O nome da coluna padrão para a função DATE_DIFF é DATE_DIFF.

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

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

O exemplo a seguir une as tabelas SALES e LISTING para calcular quantos dias os ingressos foram vendidos para as listagens 1000 a 1005 depois de serem listados. A espera mais longa para vendas dessas ofertas foi de 15 dias e a espera mais curta foi de menos de um dia (0 dias).

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

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

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

Este exemplo calcula o número médio de horas que os vendedores esperaram para todas as vendas de ingressos.

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

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

Exemplos com uma coluna TIME

O TIME_TEST da tabela a seguir tem uma coluna TIME_VAL (tipo TIME) com três valores inseridos.

```
select time_val from time_test;

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

O exemplo a seguir localiza a diferença no número de horas entre a coluna TIME_VAL e um literal de tempo.

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

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

O exemplo a seguir localiza a diferença no número de minutos entre dois valores de tempo literal.

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

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

Exemplos com uma coluna TIMETZ

O TIMETZ_TEST da tabela de exemplo a seguir tem uma coluna TIMETZ_VAL (tipo TIMETZ) com três valores inseridos.

```
select timetz_val from timetz_test;

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

O exemplo a seguir localiza as diferenças no número de horas, entre um literal TIMETZ e timetz_val.

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

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

O exemplo a seguir localiza a diferença no número de horas, entre dois valores TIMETZ literal.

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

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

Função DATE_PART

DATE_PART extrai os valores de parte da data de uma expressão. DATE_PART é sinônimo da função PGDATE_PART.

Sintaxe

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

Argumentos

Campo

Qual parte da fonte deve ser extraída e os valores de string suportados são os mesmos dos campos da função equivalente EXTRACT.

source

Uma coluna DATE ou INTERVAL de onde o campo deve ser extraído.

Tipo de retorno

Se o campo for 'SEGUNDO', um DECIMAL (8, 6). Em todos os outros casos, um INTEGER.

Exemplo

O exemplo a seguir extrai o dia do ano (DOY) de um valor de data. A saída mostra que o dia do ano para a data “2019-08-12" é. 224 Isso significa que 12 de agosto de 2019 é o 224º dia do ano de 2019.

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

Função DATE_TRUNC

A função DATE_TRUNC trunca uma expressão de timestamp ou literal com base na parte da data especificada, tal como hora, dia ou mês.

Sintaxe

```
date_trunc(format, datetime)
```

Argumentos

format

O formato que representa a unidade a ser truncada. Os formatos válidos são:

- “YEAR”, “YYYY”, “YY” - trunque até a primeira data do ano em que o ts cai, a parte do tempo será zero
- “TRIMESTRE” - trunque para a primeira data do trimestre em que o ts cai, a parte do tempo será zero
- “MÊS”, “MM”, “SEGUNDA-FEIRA” - trunque para a primeira data do mês em que o ts cai, a parte do tempo será zero
- “SEMANA” - trunque até a segunda-feira da semana em que o ts cai, a parte do tempo será zero
- “DAY”, “DD” - zerar a parte do tempo
- “HORA” - zerar o minuto e o segundo com parte fracionária
- “MINUTO” - zerar o segundo com parte fracionária
- “SEGUNDO” - zerar a segunda parte da fração
- “MILISECOND” - zerar os microssegundos
- “MICROSECOND” - tudo permanece

ts

Um valor de data e hora

Tipo de retorno

Retorna o timestamp ts truncado para a unidade especificada pelo modelo de formato

Exemplos

O exemplo a seguir trunca um valor de data para o início do ano. A saída mostra que a data “2015-03-05" foi truncada para “2015-01-01", que é o início do ano de 2015.

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

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

Função DAY

A função DAY retorna o dia do mês do carimbo de data/hora.

As funções de extração de data são úteis quando você precisa trabalhar com componentes específicos de uma data ou carimbo de data/hora, como ao realizar cálculos baseados em datas, filtrar dados ou formatar valores de data.

Sintaxe

```
day(date)
```

Argumentos

data

Uma expressão DATE ou TIMESTAMP.

Retornos

A função DAY retorna um INTEIRO.

Exemplos

O exemplo a seguir extrai o dia do mês (`30`) da data `'2009-07-30'` de entrada.

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

O exemplo a seguir extrai o dia do mês da `birthday` coluna da `squirrels` tabela e retorna os resultados como saída da instrução SELECT. O resultado dessa consulta será uma lista de valores de dias, um para cada linha na `squirrels` tabela, representando o dia do mês do aniversário de cada esquilo.

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

função DAYOFMONTH

A função DAYOFMONTH retorna o dia do mês do date/timestamp (um valor entre 1 e 31, dependendo do mês e do ano).

A função DAYOFMONTH é semelhante à função DAY, mas elas têm nomes e comportamentos ligeiramente diferentes. A função DAY é mais comumente usada, mas a função DAYOFMONTH pode ser usada como alternativa. Esse tipo de consulta pode ser útil quando você precisa realizar uma análise ou filtragem com base em datas em uma tabela que contém dados de data ou data e hora, como extrair componentes específicos de uma data para processamento ou geração de relatórios adicionais.

Sintaxe

```
dayofmonth(date)
```

Argumentos

data

Uma expressão DATE ou TIMESTAMP.

Retornos

A função DAYOFMONTH retorna um INTEIRO.

Exemplo

O exemplo a seguir extrai o dia do mês (`30`) da data `'2009-07-30'` de entrada.

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

O exemplo a seguir aplica a função DAYOFMONTH à `birthday` coluna da `squirrels` tabela. Para cada linha na `squirrels` tabela, o dia do mês da `birthday` coluna será extraído e retornado como saída da instrução SELECT. O resultado dessa consulta será uma lista de valores de dias, um para cada linha na `squirrels` tabela, representando o dia do mês do aniversário de cada esquilo.

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

Função DAYOFWEEK

A função DAYOFWEEK usa uma data ou timestamp como entrada e retorna o dia da semana como um número (1 para domingo, 2 para segunda-feira,..., 7 para sábado).

Essa função de extração de data é útil quando você precisa trabalhar com componentes específicos de uma data ou carimbo de data/hora, como ao realizar cálculos baseados em datas, filtrar dados ou formatar valores de data.

Sintaxe

```
dayofweek(date)
```

Argumentos

data

Uma expressão DATE ou TIMESTAMP.

Retornos

A função DAYOFWEEK retorna um INTEGER onde

1 = domingo

2 = segunda-feira

3 = terça-feira

4 = quarta-feira

5 = quinta-feira

6 = sexta-feira

7 = sábado

Exemplos

O exemplo a seguir extrai o dia da semana dessa data, que é 5 (representando quinta-feira).

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

```
 5
```

O exemplo a seguir extrai o dia da semana da `birthday` coluna da `squirrels` tabela e retorna os resultados como saída da instrução SELECT. O resultado dessa consulta será uma lista dos valores do dia da semana, um para cada linha na `squirrels` tabela, representando o dia da semana do aniversário de cada esquilo.

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

Função DAYOFYEAR

A função DAYOFYEAR é uma função de extração de data que usa uma data ou carimbo de data/hora como entrada e retorna o dia do ano (um valor entre 1 e 366, dependendo do ano e se é um ano bissexto).

Essa função é útil quando você precisa trabalhar com componentes específicos de uma data ou carimbo de data/hora, como ao realizar cálculos baseados em datas, filtrar dados ou formatar valores de data.

Sintaxe

```
dayofyear(date)
```

Argumentos

data

Uma expressão DATE ou TIMESTAMP.

Retornos

A função DAYOFYEAR retorna um INTEIRO (entre 1 e 366, dependendo do ano e se é um ano bissexto).

Exemplos

O exemplo a seguir extrai o dia do ano (`100`) da data `'2016-04-09'` de entrada.

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

O exemplo a seguir extrai o dia do ano da `birthday` coluna da `squirrels` tabela e retorna os resultados como saída da instrução SELECT.

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

Função EXTRACT

A função EXTRACT retorna a parte da data ou hora de um valor de TIMESTAMP, TIMESTAMPTZ, TIME ou TIMETZ. Os exemplos incluem um dia, mês, ano, hora, minuto, segundo, milissegundo ou microssegundo de um timestamp.

Sintaxe

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

Argumentos

datepart

O subcampo de uma data ou hora que será extraído, como dia, mês, ano, hora, minuto, segundo, milissegundo ou microssegundo. Para os possíveis valores, consulte Partes da data para funções de data ou de timestamp.

source

Uma coluna ou expressão que é avaliada como um tipo de dado TIMESTAMP, TIMESTAMPTZ, TIME ou TIMETZ.

Tipo de retorno

INTEGER se o valor de source for avaliado como TIMESTAMP, TIME ou TIMETZ.

DOUBLE PRECISION se o valor de source for avaliado como TIMESTAMPTZ.

Exemplos com TIME

O TIME_TEST da tabela a seguir tem uma coluna TIME_VAL (tipo TIME) com três valores inseridos.

```
select time_val from time_test;

time_val
---------------------
```

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

O exemplo a seguir extrai os minutos de cada time_val.

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

O exemplo a seguir extrai as horas de cada time_val.

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

hours
-----------
         20
         0
         0
```

função FROM_UTC_TIMESTAMP

A função FROM_UTC_TIMESTAMP converte a data de entrada de UTC (Tempo Universal Coordenado) para o fuso horário especificado.

Essa função é útil quando você precisa converter valores de data e hora do UTC para um fuso horário específico. Isso pode ser importante ao trabalhar com dados originários de diferentes partes do mundo e que precisam ser apresentados no horário local apropriado.

Sintaxe

```
from_utc_timestamp(timestamp, timezone
```

Argumentos

timestamp

Uma expressão TIMESTAMP com um timestamp UTC.

timezone

Uma expressão STRING que é um fuso horário válido no qual a data ou o timestamp de entrada devem ser convertidos.

Retornos

A função FROM_UTC_TIMESTAMP retorna um TIMESTAMP.

Exemplo

O exemplo a seguir converte a data de entrada de UTC para o fuso horário especificado (`'Asia/Seoul'`), que nesse caso é 9 horas antes do UTC. A saída resultante é a data e a hora no fuso horário de Seul, qual é`2016-08-31 09:00:00`.

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

Função HOUR

A função HOUR é uma função de extração de tempo que usa uma hora ou um carimbo de data/hora como entrada e retorna o componente de hora (um valor entre 0 e 23).

Essa função de extração de hora é útil quando você precisa trabalhar com componentes específicos de uma hora ou carimbo de data/hora, como ao realizar cálculos baseados em tempo, filtrar dados ou formatar valores de hora.

Sintaxe

```
hour(timestamp)
```

Argumentos

timestamp

Uma expressão TIMESTAMP.

Retornos

A função HOUR retorna um INTEIRO.

Exemplo

O exemplo a seguir extrai o componente hora (12) do `'2009-07-30 12:58:59'` timestamp de entrada.

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

Função MINUTE

A função MINUTE é uma função de extração de tempo que usa uma hora ou um carimbo de data/hora como entrada e retorna o componente minuto (um valor entre 0 e 60).

Sintaxe

```
minute(timestamp)
```

Argumentos

timestamp

Uma expressão TIMESTAMP ou uma STRING de um formato de timestamp válido.

Retornos

A função MINUTE retorna um INTEIRO.

Exemplo

O exemplo a seguir extrai o componente minute (58) do `'2009-07-30 12:58:59'` timestamp de entrada.

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

Função MÊS

A função MONTH é uma função de extração de hora que usa uma hora ou um carimbo de data/hora como entrada e retorna o componente do mês (um valor entre 0 e 12).

Sintaxe

```
month(date)
```

Argumentos

data

Uma expressão TIMESTAMP ou uma STRING de um formato de timestamp válido.

Retornos

A função MONTH retorna um INTEIRO.

Exemplo

O exemplo a seguir extrai o componente month (7) do `'2016-07-30'` timestamp de entrada.

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

SEGUNDA função

A função SECOND é uma função de extração de tempo que usa uma hora ou um timestamp como entrada e retorna o segundo componente (um valor entre 0 e 60).

Sintaxe

```
second(timestamp)
```

Argumentos

timestamp

Uma expressão TIMESTAMP.

Retornos

A função SECOND retorna um INTEIRO.

Exemplo

O exemplo a seguir extrai o segundo componente (59) do `'2009-07-30 12:58:59'` timestamp de entrada.

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

Função TIMESTAMP

A função TIMESTAMP pega um valor (normalmente um número) e o converte em um tipo de dados de carimbo de data/hora.

Essa função é útil quando você precisa converter um valor numérico representando uma hora ou data em um tipo de dados de carimbo de data/hora. Isso pode ser útil quando você está trabalhando com dados armazenados em um formato numérico, como carimbos de data/hora do Unix ou horário de época.

Sintaxe

```
timestamp(expr)
```

Argumentos

expr

Qualquer expressão que possa ser convertida em TIMESTAMP.

Retornos

A função TIMESTAMP retorna um TIMESTAMP.

Exemplo

O exemplo a seguir converte um timestamp numérico Unix (`1632416400`) em seu tipo de dados de timestamp correspondente: 22 de setembro de 2021 às 12:00:00 UTC.

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

Função TO_TIMESTAMP

TO_TIMESTAMP converte uma string de TIMESTAMP em TIMESTAMPTZ.

Sintaxe

```
to_timestamp (timestamp)
```

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

Argumentos

timestamp

Uma string de carimbo de data/hora ou um tipo de dados que pode ser convertido em uma string de carimbo de data/hora.

format

Uma string literal que corresponde aos padrões de data e hora do Spark. Para padrões de data e hora válidos, consulte Padrões de data e hora para formatação e análise.

Tipo de retorno

TIMESTAMP

Exemplos

O exemplo a seguir demonstra o uso da função TO_TIMESTAMP para converter uma string TIMESTAMP em TIMESTAMP.

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

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

É possível enviar parte de uma data com TO_TIMESTAMP. As partes restantes da data são definidas como valores padrão. A hora é incluída na saída:

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp
--------------------------
2017-01-01 00:00:00+00
```

A instrução SQL a seguir converte a string '2011-12-18 24:38:15 'em um TIMESTAMP. O resultado é um TIMESTAMP que cai no dia seguinte porque o número de horas é 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
```

Função YEAR

A função YEAR é uma função de extração de data que usa uma data ou timestamp como entrada e retorna o componente do ano (um número de quatro dígitos).

Sintaxe

```
year(date)
```

Argumentos

data

Uma expressão DATE ou TIMESTAMP.

Retornos

A função YEAR retorna um INTEIRO.

Exemplo

O exemplo a seguir extrai o componente ano (2016) da data `'2016-07-30'` de entrada.

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

```
2016
```

O exemplo a seguir extrai o componente ano da `birthday` coluna da `squirrels` tabela e retorna os resultados como saída da instrução SELECT. O resultado dessa consulta será uma lista dos valores do ano, um para cada linha na `squirrels` tabela, representando o ano do aniversário de cada esquilo.

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

Partes da data para funções de data ou de timestamp

A tabela a seguir identifica os nomes e abreviações da parte da data e da hora que são aceitos como argumentos para as seguintes funções:

- DATE_ADD
- DATE_DIFF
- DATE_PART
- EXTRACT

Parte da data ou parte da hora	Abreviações
milênio, milênios	mil, mils
século, séculos	c, cent, cents
década, décadas	dec, decs
epoch	epoch (compatível com EXTRACT)
ano, anos	y, yr, yrs
trimestre, trimestres	qtr, qtrs
mês, meses	mon, mons
semana, semanas	w

Parte da data ou parte da hora	Abreviações
dia da semana	dayofweek, dow, dw, weekday (compatível com DATE_PART e Função EXTRACT) Retorna um número inteiro de 0 a 6, começando com domingo. **Note** A parte da data DOW se comporta de maneira diferente da parte da data do dia da semana (D) usada para strings de formato de data e hora. D se baseia no números inteiros 1 a 7, onde domingo é 1. Para obter mais informações, consulte Strings de formato datetime.
dia do ano	dayofyear, doy, dy, yearday (compatível com EXTRACT)
dia, dias	d
hora, horas	h, hr, hrs
minuto, minutos	m, min, mins
segundo, segundos	s, sec, secs
milissegundo, milissegundos	ms, msec, msecs, msecond, mseconds, millisec, millisecs, millisecon
microssegundo, microssegundos	microsec, microsecs, microsecond, usecond, useconds, us, usec, usecs
timezone, timezone_hour, timezone_minute	Compatível com EXTRACT para timestamp somente com fuso horário (TIMESTAMPTZ).

Variações nos resultados com segundos, milissegundos e microssegundos

Pequenas diferenças nos resultados de consultas ocorrem quando diferentes funções de data especificam segundos, milissegundos ou microssegundos como partes da data:

- A função EXTRACT retorna números inteiros somente para a parte da data especificada, ignorando partes de data de níveis superiores e inferiores. Se a parte da data especificada é segundos, os milissegundos e os microssegundos não são incluídos no resultados. Se a parte da data especificada é milissegundos, segundos e microssegundos não são incluídos. Se a parte da data especificada é microssegundos, segundos e milissegundos não são incluídos.
- A função DATE_PART retorna a parte completa de segundos do timestamp, independente da parte de data especificada, retornando um valor decimal ou um número inteiro conforme necessário.

Observações de CENTURY, EPOCH, DECADE e MIL

CENTURY ou CENTURIES

AWS Clean Rooms interpreta um SÉCULO como começando com o ano ## #1 e terminando com o ano: ###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

A AWS Clean Rooms implementação do EPOCH é relativa a 1970-01-01 00:00:00.000 000, independente do fuso horário em que o cluster reside. Você pode precisar deslocar os resultados pela diferença em horas dependendo do fuso horário onde o cluster está localizado.

DECADE ou DECADES

AWS Clean Rooms interpreta o DECADE ou DECADES DATEPART com base no calendário comum. Por exemplo, como o calendário comum começa a partir do ano 1, a primeira década (década 1) é 0001-01-01 a 0009-12-31 e a segunda década (década 2) é 0010-01-01 a 0019-12-31. Por exemplo, a década 201 vai de 2000-01-01 a 2009-12-31:

```
select extract(decade from timestamp '1999-02-16 20:38:40');
date_part
-----------
200
(1 row)

select extract(decade from timestamp '2000-02-16 20:38:40');
date_part
-----------
201
(1 row)

select extract(decade from timestamp '2010-02-16 20:38:40');
date_part
-----------
202
(1 row)
```

MIL ou MILS

AWS Clean Rooms interpreta uma MIL para começar com o primeiro dia do ano #001 e terminar com o último dia do ano#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)
```

Funções de criptografia e descriptografia

As funções de criptografia e descriptografia ajudam os desenvolvedores de SQL a proteger dados confidenciais contra acesso não autorizado ou uso indevido, convertendo-os entre um formulário de texto simples legível e um formulário de texto cifrado ilegível.

AWS Clean Rooms O Spark SQL é compatível com as seguintes funções de criptografia e decodificação:

Tópicos

função AES_ENCRYPT

A função AES_ENCRYPT é usada para criptografar dados usando o algoritmo Advanced Encryption Standard (AES).

Sintaxe

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

Argumentos

expr

O valor binário a ser criptografado.

chave

A senha a ser usada para criptografar os dados.

Comprimentos de chave de 16, 24 e 32 bits são suportados.

modo

Especifica qual modo de criptografia de bloco deve ser usado para criptografar mensagens.

Modos válidos: ECB (eletrônico CodeBook), GCM (modo Galois/Counter), CBC (Cipher-Block Chaining).

acolchoamento

Especifica como preencher mensagens cujo tamanho não seja múltiplo do tamanho do bloco.

Valores válidos: PKCS, NONE, DEFAULT.

O preenchimento DEFAULT significa PKCS (Padrões de Criptografia de Chave Pública) para ECB, NONE para GCM e PKCS para CBC.

As combinações suportadas de (modo, preenchimento) são ('ECB', 'PKCS'), ('GCM', 'NONE') e ('CBC', 'PKCS').

iv

Vetor de inicialização opcional (IV). Compatível apenas com os modos CBC e GCM.

Valores válidos: 12 bytes para GCM e 16 bytes para CBC.

anúncio

Dados autenticados adicionais (AAD) opcionais. Compatível apenas com o modo GCM. Isso pode ser qualquer entrada de formato livre e deve ser fornecido tanto para criptografia quanto para decodificação.

Tipo de retorno

A função AES_ENCRYPT retorna um valor criptografado de expr usando AES em determinado modo com o preenchimento especificado.

Exemplos

O exemplo a seguir demonstra como usar a função AES_ENCRYPT do Spark SQL para criptografar com segurança uma sequência de dados (nesse caso, a palavra “Spark”) usando uma chave de criptografia especificada. O texto cifrado resultante é então codificado em Base64 para facilitar o armazenamento ou a transmissão.

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

O exemplo a seguir demonstra como usar a função AES_ENCRYPT do Spark SQL para criptografar com segurança uma sequência de dados (nesse caso, a palavra “Spark”) usando uma chave de criptografia especificada. O texto cifrado resultante é então representado em formato hexadecimal, o que pode ser útil para tarefas como armazenamento, transmissão ou depuração de dados.

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

O exemplo a seguir demonstra como usar a função AES_ENCRYPT do Spark SQL para criptografar com segurança uma sequência de dados (nesse caso, “Spark SQL”) usando uma chave de

criptografia, um modo de criptografia e um modo de preenchimento especificados. O texto cifrado resultante é então codificado em Base64 para facilitar o armazenamento ou a transmissão.

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

função AES_DECRYPT

A função AES_DECRYPT é usada para descriptografar dados usando o algoritmo Advanced Encryption Standard (AES).

Sintaxe

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

Argumentos

expr

O valor binário a ser decifrado.

chave

A frase secreta a ser usada para descriptografar os dados.

A frase secreta deve corresponder à chave originalmente usada para produzir o valor criptografado e ter 16, 24 ou 32 bytes.

modo

Especifica qual modo de criptografia de bloco deve ser usado para descriptografar mensagens.

Modos válidos: ECB, GCM, CBC.

acolchoamento

Especifica como preencher mensagens cujo tamanho não seja múltiplo do tamanho do bloco.

Valores válidos: PKCS, NONE, DEFAULT.

O preenchimento DEFAULT significa PKCS para ECB, NONE para GCM e PKCS para CBC.

anúncio

Dados autenticados adicionais (AAD) opcionais. Compatível apenas com o modo GCM. Isso pode ser qualquer entrada de formato livre e deve ser fornecido tanto para criptografia quanto para decodificação.

Tipo de retorno

Retorna um valor descriptografado de expr usando AES no modo com preenchimento.

Exemplos

O exemplo a seguir demonstra como usar a função AES_ENCRYPT do Spark SQL para criptografar com segurança uma sequência de dados (nesse caso, a palavra “Spark”) usando uma chave de criptografia especificada. O texto cifrado resultante é então codificado em Base64 para facilitar o armazenamento ou a transmissão.

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

O exemplo a seguir demonstra como usar a função AES_DECRYPT do Spark SQL para descriptografar dados que foram previamente criptografados e codificados em Base64. O processo de decodificação requer a chave e os parâmetros de criptografia corretos (modo de criptografia e modo de preenchimento) para recuperar com êxito os dados de texto sem formatação originais.

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

Funções de hash

Uma função de hash é uma função matemática que converte um valor de entrada numérico em outro valor.

AWS Clean Rooms O Spark SQL é compatível com as seguintes funções de hash:

Tópicos

- MD5 função
- Função SHA

- SHA1 função
- SHA2 função
- HASH64 função xx

MD5 função

Usa a função hash MD5 criptográfica para converter uma string de comprimento variável em uma string de 32 caracteres que é uma representação em texto do valor hexadecimal de uma soma de verificação de 128 bits.

Sintaxe

```
MD5(string)
```

Argumentos

string

Uma string de comprimento variável.

Tipo de retorno

A MD5 função retorna uma cadeia de caracteres de 32 caracteres que é uma representação em texto do valor hexadecimal de uma soma de verificação de 128 bits.

Exemplos

O seguinte exemplo mostra o valor de 128 bits para a string "AWS Clean Rooms":

```
select md5('AWS Clean Rooms');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

Função SHA

Sinônimo de SHA1 função.

Consulte SHA1 função.

SHA1 função

A SHA1 função usa a função hash SHA1 criptográfica para converter uma string de comprimento variável em uma string de 40 caracteres que é uma representação em texto do valor hexadecimal de uma soma de verificação de 160 bits.

Sintaxe

SHA1 é sinônimo de. Função SHA

```
SHA1(string)
```

Argumentos

string

Uma string de comprimento variável.

Tipo de retorno

A SHA1 função retorna uma string de 40 caracteres que é uma representação em texto do valor hexadecimal de uma soma de verificação de 160 bits.

Exemplo

O seguinte exemplo retorna o valor de 160 bits para a palavra "AWS Clean Rooms":

```
select sha1('AWS Clean Rooms');
```

SHA2 função

A SHA2 função usa a função hash SHA2 criptográfica para converter uma cadeia de caracteres de comprimento variável em uma cadeia de caracteres. A string de caracteres é uma representação de texto do valor hexadecimal da soma de verificação com o número especificado de bits.

Sintaxe

```
SHA2(string, bits)
```

Argumentos

string

Uma string de comprimento variável.

inteiro

O número de bits nas funções de hash. Os valores válidos são 0 (igual a 256), 224, 256, 384 e 512.

Tipo de retorno

A SHA2 função retorna uma cadeia de caracteres que é uma representação em texto do valor hexadecimal da soma de verificação ou uma cadeia vazia se o número de bits for inválido.

Exemplo

O exemplo a seguir retorna o valor de 256 bits para a palavra “AWS Clean Rooms”:

```
select sha2('AWS Clean Rooms', 256);
```

HASH64 função xx

A função xxhash64 retorna um valor de hash de 64 bits dos argumentos.

A função xxhash64 () é uma função hash não criptográfica projetada para ser rápida e eficiente. É frequentemente usado em aplicativos de processamento e armazenamento de dados, em que é necessário um identificador exclusivo para uma parte dos dados, mas o conteúdo exato dos dados não precisa ser mantido em segredo.

No contexto de uma consulta SQL, a função xxhash64 () pode ser usada para várias finalidades, como:

- Gerando um identificador exclusivo para uma linha em uma tabela
- Particionamento de dados com base em um valor de hash
- Implementando estratégias personalizadas de indexação ou distribuição de dados

O caso de uso específico dependeria dos requisitos do aplicativo e dos dados que estão sendo processados.

Sintaxe

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

Argumentos

expr 1

Uma expressão de qualquer tipo.

expr 2

Uma expressão de qualquer tipo.

Retornos

Retorna um valor de hash de 64 bits dos argumentos (BIGINT). A semente de haxixe é 42.

Exemplo

O exemplo a seguir gera um valor de hash de 64 bits (5602566077635097486) com base na entrada fornecida. O primeiro argumento é um valor de string, nesse caso, a palavra “Spark”. O segundo argumento é uma matriz contendo o valor inteiro único 123. O terceiro argumento é um valor inteiro que representa a semente da função hash.

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

Funções do Hyperloglog

As funções HyperLogLog (HLL) no SQL fornecem uma maneira de estimar com eficiência o número de elementos exclusivos (cardinalidade) em um grande conjunto de dados, mesmo quando o conjunto real de elementos exclusivos não está armazenado.

Os principais benefícios do uso das funções HLL são:

- Eficiência de memória: os esboços HLL exigem muito menos memória do que armazenar o conjunto completo de elementos exclusivos, tornando-os adequados para grandes conjuntos de dados.
- Computação distribuída: os esboços do HLL podem ser combinados em várias fontes de dados ou nós de processamento, permitindo uma estimativa de contagem exclusiva distribuída e eficiente.

- Resultados aproximados: o HLL fornece uma estimativa de contagem única aproximada, com uma compensação ajustável entre precisão e uso de memória (por meio do parâmetro de precisão).

Essas funções são particularmente úteis em cenários em que você precisa estimar o número de itens exclusivos, como em aplicativos de análise, armazenamento de dados e processamento de fluxo em tempo real.

AWS Clean Rooms suporta as seguintes funções de HLL.

Tópicos

função HLL_SKETCH_AGG

A função agregada HLL_SKETCH_AGG cria um esboço HLL a partir dos valores na coluna especificada. Ele retorna um tipo de dados HLLSKETCH que encapsula os valores da expressão de entrada.

A função agregada HLL_SKETCH_AGG funciona com qualquer tipo de dados e ignora valores NULL.

Quando não há linhas em uma tabela ou todas as linhas são NULL, o esboço resultante não tem pares de valor de índice, como `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}`.

Sintaxe

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

Argumento

aggregate_expression

Qualquer expressão do tipo INT, BIGINT, STRING ou BINARY em relação à qual ocorrerá uma contagem exclusiva. Todos NULL os valores são ignorados.

LG ConfigK

Uma constante INT opcional entre 4 e 21, inclusive com o padrão 12. A base logarítmica 2 de K, onde K é o número de compartimentos ou slots para o esboço.

Tipo de retorno

A função HLL_SKETCH_AGG retorna um buffer BINÁRIO não NULL contendo o HyperLogLog esboço calculado devido ao consumo e agregação de todos os valores de entrada no grupo de agregação.

Exemplos

Os exemplos a seguir usam o algoritmo HyperLogLog (HLL) para estimar a contagem distinta de valores na `col` coluna. A `hll_sketch_agg(col, 12)` função agrega os valores na coluna col, criando um esboço HLL usando uma precisão de 12. A `hll_sketch_estimate()` função é então usada para estimar a contagem distinta de valores com base no esboço HLL gerado. O resultado final da consulta é 3, que representa a contagem distinta estimada de valores na `col` coluna. Nesse caso, os valores distintos são 1, 2 e 3.

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

O exemplo a seguir também usa o algoritmo HLL para estimar a contagem distinta de valores na `col` coluna, mas não especifica um valor de precisão para o esboço do HLL. Nesse caso, ele usa a precisão padrão de 14. A `hll_sketch_agg(col)` função pega os valores na `col` coluna e cria um esboço HyperLogLog (HLL), que é uma estrutura de dados compacta que pode ser usada para estimar a contagem distinta de elementos. A `hll_sketch_estimate(hll_sketch_agg(col))` função pega o esboço HLL criado na etapa anterior e calcula uma estimativa da contagem distinta de valores na coluna. `col` O resultado final da consulta é 3, que representa a contagem distinta estimada de valores na `col` coluna. Nesse caso, os valores distintos são 1, 2 e 3.

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

função HLL_SKETCH_ESTIMATE

A função HLL_SKETCH_ESTIMATE usa um esboço HLL e estima o número de elementos exclusivos representados pelo esboço. Ele usa o algoritmo HyperLogLog (HLL) para contar uma aproximação probabilística do número de valores exclusivos em uma determinada coluna, consumindo uma representação binária conhecida como buffer de esboço gerada anteriormente pela função HLL_SKETCH_AGG e retornando o resultado como um número inteiro grande.

O algoritmo de esboço HLL fornece uma maneira eficiente de estimar o número de elementos exclusivos, mesmo para grandes conjuntos de dados, sem precisar armazenar o conjunto completo de valores exclusivos.

As `hll_union_agg` funções `hll_union` e também podem combinar esboços consumindo e mesclando esses buffers como entradas.

Sintaxe

```
HLL_SKETCH_ESTIMATE (hllsketch_expression)
```

Argumento

hllsketch_expression

Uma `BINARY` expressão contendo um esboço gerado por HLL_SKETCH_AGG

Tipo de retorno

A função HLL_SKETCH_ESTIMATE retorna um valor BIGINT que é a contagem distinta aproximada representada pelo esboço de entrada.

Exemplos

Os exemplos a seguir usam o algoritmo de esboço HyperLogLog (HLL) para estimar a cardinalidade (contagem exclusiva) dos valores na coluna. `col` A `hll_sketch_agg(col, 12)` função pega a `col` coluna e cria um esboço HLL usando uma precisão de 12 bits. O esboço do HLL é uma estrutura de dados aproximada que pode estimar com eficiência o número de elementos exclusivos em um conjunto. A `hll_sketch_estimate()` função pega o esboço HLL criado por `hll_sketch_agg` e estima a cardinalidade (contagem única) dos valores representados pelo esboço. O `FROM VALUES (1), (1), (2), (2), (3) tab(col);` gera um conjunto de dados

de teste com 5 linhas, em que a `col` coluna contém os valores 1, 1, 2, 2 e 3. O resultado dessa consulta é a contagem exclusiva estimada dos valores na `col` coluna, que é 3.

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

A diferença entre o exemplo a seguir e o anterior é que o parâmetro de precisão (12 bits) não está especificado na chamada da `hll_sketch_agg` função. Nesse caso, a precisão padrão de 14 bits é usada, o que pode fornecer uma estimativa mais precisa para a contagem exclusiva em comparação com o exemplo anterior que usou 12 bits de precisão.

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

função HLL_UNION

A função HLL_UNION combina dois esboços HLL em um único esboço unificado. Ele usa o algoritmo HyperLogLog (HLL) para combinar dois esboços em um único esboço. As consultas podem usar os buffers resultantes para calcular contagens exclusivas aproximadas como números inteiros longos com a função. `hll_sketch_estimate`

Sintaxe

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

Argumento

Exprn

Uma `BINARY` expressão contendo um esboço gerado por HLL_SKETCH_AGG.

allowDifferentLgConfigK

Uma expressão BOOLEAN opcional que controla se é permitido mesclar dois esboços com valores LGConfigK diferentes. O valor padrão é `false`.

Tipo de retorno

A função HLL_UNION retorna um buffer BINARY contendo o HyperLogLog esboço calculado como resultado da combinação das expressões de entrada. Quando o `allowDifferentLgConfigK` parâmetro é`true`, o esboço resultante usa o menor dos dois `lgConfigK` valores fornecidos.

Exemplos

Os exemplos a seguir usam o algoritmo de esboço HyperLogLog (HLL) para estimar a contagem exclusiva de valores em duas colunas `col1` e em um `col2` conjunto de dados.

A `hll_sketch_agg(col1)` função cria um esboço HLL para os valores exclusivos na `col1` coluna.

A `hll_sketch_agg(col2)` função cria um esboço HLL para os valores exclusivos na coluna col2.

A `hll_union(...)` função combina os dois esboços de HLL criados nas etapas 1 e 2 em um único esboço de HLL unificado.

A `hll_sketch_estimate(...)` função pega o esboço combinado do HLL e estima a contagem exclusiva de valores em ambos e. `col1` `col2`

A `FROM VALUES` cláusula gera um conjunto de dados de teste com 5 linhas, onde `col1` contém os valores 1, 1, 2, 2 e 3 e `col2` contém os valores 4, 4, 5, 5 e 6.

O resultado dessa consulta é a contagem exclusiva estimada de valores em ambos `col1` e`col2`, que é 6. O algoritmo de esboço HLL fornece uma maneira eficiente de estimar o número de elementos exclusivos, mesmo para grandes conjuntos de dados, sem precisar armazenar o conjunto completo de valores exclusivos. Neste exemplo, a `hll_union` função é usada para combinar os esboços do HLL das duas colunas, o que permite que a contagem exclusiva seja estimada em todo o conjunto de dados, em vez de apenas para cada coluna individualmente.

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

```
  6
```

A diferença entre o exemplo a seguir e o anterior é que o parâmetro de precisão (12 bits) não está especificado na chamada da `hll_sketch_agg` função. Nesse caso, a precisão padrão de 14 bits é usada, o que pode fornecer uma estimativa mais precisa para a contagem exclusiva em comparação com o exemplo anterior que usou 12 bits de precisão.

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

função HLL_UNION_AGG

A função HLL_UNION_AGG combina vários esboços HLL em um único esboço unificado. Ele usa o algoritmo HyperLogLog (HLL) para combinar um grupo de esboços em um único. As consultas podem usar os buffers resultantes para calcular contagens exclusivas aproximadas com a função. `hll_sketch_estimate`

Sintaxe

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

Argumento

expr

Uma `BINARY` expressão contendo um esboço gerado por HLL_SKETCH_AGG.

allowDifferentLgConfigK

Uma expressão BOOLEAN opcional que controla se é permitido mesclar dois esboços com valores LGConfigK diferentes. O valor padrão é `false`.

Tipo de retorno

A função HLL_UNION_AGG retorna um buffer BINARY contendo o HyperLogLog esboço calculado como resultado da combinação das expressões de entrada do mesmo grupo. Quando o `allowDifferentLgConfigK` parâmetro é`true`, o esboço resultante usa o menor dos dois `lgConfigK` valores fornecidos.

Exemplos

Os exemplos a seguir usam o algoritmo de esboço HyperLogLog (HLL) para estimar a contagem exclusiva de valores em vários esboços de HLL.

O primeiro exemplo estima a contagem exclusiva de valores em um conjunto de dados.

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

A consulta interna cria dois esboços de HLL:

- A primeira instrução SELECT cria um esboço a partir de um único valor de 1.
- A segunda instrução SELECT cria um esboço a partir de outro valor único de 1, mas com uma precisão de 20.

A consulta externa usa a função HLL_UNION_AGG para combinar os dois esboços em um único esboço. Em seguida, ele aplica a função HLL_SKETCH_ESTIMATE a esse esboço combinado para estimar a contagem exclusiva de valores.

O resultado dessa consulta é a contagem exclusiva estimada dos valores na `col` coluna, que é`1`. Isso significa que os dois valores de entrada de 1 são considerados exclusivos, mesmo que tenham o mesmo valor.

O segundo exemplo inclui um parâmetro de precisão diferente para a função HLL_UNION_AGG. Nesse caso, os dois esboços do HLL são criados com uma precisão de 14 bits, o que permite que sejam combinados com sucesso usando o `hll_union_agg` `true` parâmetro.

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

O resultado final da consulta é a contagem exclusiva estimada, que nesse caso também é1. Isso significa que os dois valores de entrada de 1 são considerados exclusivos, mesmo que tenham o mesmo valor.

Funções JSON

Quando você precisa armazenar um conjunto relativamente pequeno de pares de valores de chave, você pode economizar espaço armazenando os dados em formato JSON. Como strings JSON podem ser armazenados em uma única coluna, o uso de JSON pode ser mais eficiente que armazenar seus dados em formato tabular.

Example

Por exemplo, suponha que você tenha uma tabela esparsa, onde você precisa ter muitas colunas para representar totalmente todos os atributos possíveis. No entanto, a maioria dos valores da coluna é NULL para qualquer linha ou coluna. Ao usar JSON para armazenamento, você poderá armazenar os dados de uma linha em pares de valores-chave em uma única string JSON e eliminar as colunas da tabela pouco preenchidas.

Além disso, você pode facilmente modificar strings JSON para armazenar pares de valor:chave sem a necessidade de adicionar colunas à uma tabela.

Recomendamos usar JSON frugalmente. JSON não é uma boa opção para armazenar conjuntos de dados maiores porque, ao armazenar dados díspares em uma única coluna, o JSON não usa a arquitetura de armazenamento de colunas do AWS Clean Rooms .

JSON usa strings de texto codificadas por UTF-8, portanto strings JSON podem ser armazenadas como tipos de dados CHAR ou VARCHAR. Use VARCHAR se as strings incluírem caracteres multibyte.

As strings JSON devem ser adequadamente formatadas como JSON de acordo com as seguintes regras:

- O nível JSON raiz pode ser um objeto JSON ou uma matriz JSON. Um objeto JSON é um conjunto desordenado de pares de valor:chave separados por vírgula cercado por chaves.

  Por exemplo, `{"one":1, "two":2}` .
- Uma matriz JSON é um conjunto ordenado de valores separados por vírgula cercado por parênteses.

  Um exemplo é o seguinte: `["first", {"one":1}, "second", 3, null]`
- Matrizes JSON usam um índice baseado em zero; o primeiro elemento em uma matriz fica na posição 0. Em um par de valores-chave JSON, a chave é uma string entre aspas duplas.
- Um valor JSON pode ser qualquer um dos seguintes:
  - Objeto JSON
  - matriz JSON
  - String entre aspas duplas
  - Número (inteiro e flutuante)
  - Booleano
  - Null
- Objetos vazios e matrizes vazias são valores JSON válidos.
- Os campos JSON diferenciam maiúsculas e minúsculas.
- O espaço em branco entre elementos estruturais JSON (tal como `{ }, [ ]`) é ignorado.

Tópicos

função GET_JSON_OBJECT

A função GET_JSON_OBJECT extrai um objeto json de. `path`

Sintaxe

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

Argumentos

json_txt

Uma expressão STRING contendo JSON bem formado.

path

Um literal STRING com uma expressão de caminho JSON bem formada.

Retornos

Retorna uma STRING.

Um NULL é retornado se o objeto não puder ser encontrado.

Exemplo

O exemplo a seguir extrai um valor de um objeto JSON. O primeiro argumento é uma string JSON que representa um objeto simples com um único par de valores-chave. O segundo argumento é uma expressão de caminho JSON. O $ símbolo representa a raiz do objeto JSON e a `.`a parte especifica que queremos extrair o valor associado à chave a "". A saída da função é 'b', que é o valor associado à chave "a" no objeto JSON de entrada.

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

função TO_JSON

A função TO_JSON converte uma expressão de entrada em uma representação de string JSON. A função manipula a conversão de diferentes tipos de dados (como números, cadeias de caracteres e booleanos) em suas representações JSON correspondentes.

A função TO_JSON é útil quando você precisa converter dados estruturados (como linhas de banco de dados ou objetos JSON) em um formato mais portátil e autodescritivo, como JSON. Isso pode ser particularmente útil quando você precisa interagir com outros sistemas ou serviços que esperam dados formatados em JSON.

Sintaxe

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

Argumentos

expr

A expressão de entrada que você deseja converter em uma string JSON. Pode ser um valor, uma coluna ou qualquer outra expressão SQL válida.

options

Um conjunto opcional de opções de configuração que pode ser usado para personalizar o processo de conversão de JSON. Essas opções podem incluir coisas como o tratamento de valores nulos, a representação de valores numéricos e o tratamento de caracteres especiais.

Retornos

Retorna uma string JSON com um determinado valor de estrutura

Exemplos

O exemplo a seguir converte uma estrutura nomeada (um tipo de dados estruturados) em uma string JSON. O primeiro argumento `(named_struct('a', 1, 'b', 2)` () é a expressão de entrada que é passada para a `to_json()` função. Ele cria uma estrutura nomeada com dois campos: “a” com um valor de 1 e “b” com um valor de 2. A função to_json () usa a estrutura nomeada como argumento e a converte em uma representação de string JSON. A saída é`{"a":1,"b":2}`, que é uma string JSON válida que representa a estrutura nomeada.

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

O exemplo a seguir converte uma estrutura nomeada que contém um valor de timestamp em uma string JSON, com um formato de timestamp personalizado. O primeiro argumento (`named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd'))`) cria uma estrutura nomeada com um único campo 'time' que contém o valor do timestamp. O segundo argumento (`map('timestampFormat', 'dd/MM/yyyy')`) cria um mapa (dicionário de valores-chave) com um único par de valores-chave, em que a chave é 'timestampFormat' e o valor é ". dd/MM/yyyy'. This map is used to specify the desired format for the timestamp value when converting it to JSON. The to_json() function converts the named struct into a JSON string. The second argument, the map, is used to customize the timestamp format to 'dd/MM/yyyy A saída é`{"time":"26/08/2015"}`, que é uma string JSON com um único campo 'time' que contém o valor do timestamp no formato " desejado. dd/MM/yyyy

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

Funções matemáticas

Esta seção descreve os operadores e funções matemáticas compatíveis com o AWS Clean Rooms Spark SQL.

Tópicos

- Função ROUND
- Função SIGN
- Função SIN
- Função SQRT
- Função TRUNC

Símbolos de operadores matemáticos

A tabela a seguir lista os operadores matemáticos compatíveis.

Operadores compatíveis

Operador	Descrição	Exemplo	Resultado
+	adição	2 + 3	5
-	subtração	2 - 3	-1
*	multiplicação	2 x 3	6
/	divisão	4 / 2	2
%	módulo	5 % 4	1
^	exponenciação	2,0 ^ 3,0	8

Exemplos

Calcule a comissão paga mais uma taxa de manuseio de US$ 2,00 para uma determinada transação:

```
select commission, (commission + 2.00) as comm
from sales where salesid=10000;

commission | comm
-----------+-------
```

```
28.05      | 30.05
(1 row)
```

Calcule 20 por cento do preço de venda para determinada transação:

```
select pricepaid, (pricepaid * .20) as twentypct
from sales where salesid=10000;

pricepaid | twentypct
----------+-----------
187.00    |    37.400
(1 row)
```

Faça a previsão das vendas de ingressos com base em um padrão de crescimento contínuo. Neste exemplo, a subconsulta retorna o número de ingressos vendidos em 2008. Esse resultado é multiplicado exponencialmente por uma taxa de crescimento contínuo de 5% ao longo de 10 anos.

```
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)
```

Encontre o preço total pago e a comissão pelas vendas com ID de data maior ou igual a 2.000. Então, subtraia a comissão total do preço total pago.

```
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)
```

Função ABS

ABS calcula o valor absoluto de um número, que pode ser um literal ou uma expressão que avalie para um número.

Sintaxe

```
ABS (number)
```

Arguments (Argumentos)

número

Número ou expressão que avalia para um número. Pode ser do tipo SMALLINT, INTEGER, BIGINT, DECIMAL ou. FLOAT4 FLOAT8

Tipo de retorno

ABS retorna o mesmo tipo de dados que seu argumento.

Exemplos

Calcule o valor absoluto de -38:

```
select abs (-38);
abs
-------
38
(1 row)
```

Calcule o valor absoluto de (14-76):

```
select abs (14-76);
abs
-------
62
```

```
(1 row)
```

Função ACOS

ACOS é uma função trigonométrica que retorna o arco cosseno de um número. O valor de retorno é em radianos, entre `0` e `PI`.

Sintaxe

```
ACOS(number)
```

Arguments (Argumentos)

número

O parâmetro de entrada é um número `DOUBLE PRECISION`.

Tipo de retorno

`DOUBLE PRECISION`

Exemplos

Para retornar o arco cosseno de `-1`, use o exemplo a seguir.

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Função ASIN

ASIN é uma função trigonométrica que retorna o arco seno de um número. O valor de retorno é em radianos, entre `PI/2` e `-PI/2`.

Sintaxe

```
ASIN(number)
```

Arguments (Argumentos)

número

O parâmetro de entrada é um número `DOUBLE PRECISION`.

Tipo de retorno

`DOUBLE PRECISION`

Exemplos

Para retornar o arco seno de 1, use o exemplo a seguir.

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

Função ATAN

ATAN é uma função trigonométrica que retorna a arco tangente de um número. O valor de retorno é em radianos, entre `-PI` e `PI`.

Sintaxe

```
ATAN(number)
```

Arguments (Argumentos)

número

O parâmetro de entrada é um número `DOUBLE PRECISION`.

Tipo de retorno

`DOUBLE PRECISION`

Exemplos

Para retornar a arco tangente de 1 e a multiplica por 4, use o exemplo a seguir.

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

ATAN2 função

ATAN2 é uma função trigonométrica que retorna o arco tangente de um número dividido por outro número. O valor de retorno é em radianos, entre `PI/2` e `-PI/2`.

Sintaxe

```
ATAN2(number1, number2)
```

Arguments (Argumentos)

number1

Um número `DOUBLE PRECISION`.

number2

Um número `DOUBLE PRECISION`.

Tipo de retorno

`DOUBLE PRECISION`

Exemplos

Para retornar a arco tangente de `2/2` e a multiplica por `4`, use o exemplo a seguir.

```
SELECT ATAN2(2,2) * 4 AS PI;
```

```
+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Função CBRT

A função CBRT é uma função matemática que calcula a raiz cúbica de um número.

Sintaxe

```
CBRT (number)
```

Argumento

CBRT assume um número de DOUBLE PRECISION como um argumento.

Tipo de retorno

CBRT retorna um número de DOUBLE PRECISION.

Exemplos

Calcule a raiz cúbica da comissão paga para determinada transação:

```
select cbrt(commission) from sales where salesid=10000;

cbrt
------------------
3.03839539048843
(1 row)
```

Função CEILING (ou CEIL)

A função CEILING ou CEIL é usada para arredondar um número para o número inteiro seguinte. (A função Função FLOOR arredonda um número para o número inteiro anterior.)

Sintaxe

```
CEIL | CEILING(number)
```

Arguments (Argumentos)

número

O número ou expressão avaliada como um número. Pode ser do tipo SMALLINT, INTEGER, BIGINT, DECIMAL ou. FLOAT4 FLOAT8

Tipo de retorno

CEILING e CEIL retornam o mesmo tipo de dados que seu argumento.

Exemplo

Calcule o teto da comissão paga para determinada transação de vendas:

```
select ceiling(commission) from sales
where salesid=10000;

ceiling
---------
29
(1 row)
```

Função COS

COS é uma função trigonométrica que retorna o cosseno de um número. O valor de retorno é em radianos, entre -1 e 1, incluindo ambos.

Sintaxe

```
COS(double_precision)
```

Argumento

número

O parâmetro de entrada é um número de precisão dupla.

Tipo de retorno

A função COS retorna um número de precisão dupla.

Exemplos

O seguinte exemplo retorna o cosseno de 0:

```
select cos(0);
cos
-----
1
(1 row)
```

O seguinte exemplo retorna o cosseno de PI:

```
select cos(pi());
cos
-----
-1
(1 row)
```

Função COT

COT é uma função trigonométrica que retorna a cotangente de um número. O parâmetro de entrada deve ser diferente de zero.

Sintaxe

```
COT(number)
```

Argumento

número

O parâmetro de entrada é um número `DOUBLE PRECISION`.

Tipo de retorno

`DOUBLE PRECISION`

Exemplos

Para retornar a cotangente de 1, use o exemplo a seguir.

```
SELECT COT(1);
```

```
+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

Função DEGREES

Converte um ângulo em radianos para seu equivalente em graus.

Sintaxe

```
DEGREES(number)
```

Argumento

número

O parâmetro de entrada é um número `DOUBLE PRECISION`.

Tipo de retorno

`DOUBLE PRECISION`

Exemplo

Para retornar o grau equivalente de 0,5 radiano, use o exemplo a seguir.

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
+-------------------+
```

Para converter PI radianos em graus, use o exemplo a seguir.

```
SELECT DEGREES(pi());

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

```
| degrees |
+---------+
|     180 |
+---------+
```

Função DIV

O operador DIV retorna a parte integral da divisão do dividendo por divisor.

Sintaxe

```
dividend div divisor
```

Arguments (Argumentos)

dividendo

Uma expressão que é avaliada como numérica ou intervalo.

divisor

Um tipo de intervalo correspondente se `dividend` for um intervalo, caso contrário, um numérico.

Tipo de retorno

`BIGINT`

Exemplos

O exemplo a seguir seleciona duas colunas da tabela de esquilos: a `id` coluna, que contém o identificador exclusivo para cada esquilo, e uma `calculated` coluna,`age div 2`, que representa a divisão inteira da coluna de idade por 2. O `age div 2` cálculo realiza a divisão de números inteiros na `age` coluna, arredondando efetivamente a idade para o número inteiro par mais próximo. Por exemplo, se a `age` coluna contiver valores como 3, 5, 7 e 10, a `age div 2` coluna conterá os valores 1, 2, 3 e 5, respectivamente.

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

Essa consulta pode ser útil em cenários em que você precisa agrupar ou analisar dados com base em faixas etárias e deseja simplificar os valores de idade arredondando-os para o número inteiro par mais próximo. O resultado resultante forneceria a idade `id` e a divisão por 2 para cada esquilo na `squirrels` tabela.

Função EXP

A função EXP implementa a função exponencial de uma expressão numérica, ou a base de um logaritmo natural, e, elevada à potência da expressão. A função EXP é o inverso de Função LN.

Sintaxe

```
EXP (expression)
```

Argumento

expressão

A expressão deve ser um tipo de dados INTEGER, DECIMAL ou DOUBLE PRECISION.

Tipo de retorno

EXP retorna um número de DOUBLE PRECISION.

Exemplo

Use a função EXP para prever as vendas de ingressos com base em um padrão de crescimento contínuo. Neste exemplo, a subconsulta retorna o número de ingressos vendidos em 2008. Esse resultado é multiplicado pelo resultado da função EXP, que especifica uma taxa de crescimento contínuo de 7% por 10 anos.

```
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)
```

Função FLOOR

A função FLOOR arredonda um número para o número inteiro anterior.

Sintaxe

```
FLOOR (number)
```

Argumento

número

O número ou expressão avaliada como um número. Pode ser do tipo SMALLINT, INTEGER, BIGINT, DECIMAL ou. FLOAT4 FLOAT8

Tipo de retorno

FLOOR retorna o mesmo tipo de dados que seu argumento.

Exemplo

O exemplo mostra o valor da comissão paga por determinada transação de vendas antes e depois de usar a função 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)
```

Função LN

A função LN retorna o logaritmo natural do parâmetro de entrada.

Sintaxe

```
LN(expression)
```

Argumento

expressão

A coluna ou expressão de destino na qual a função opera.

> **Note**
>
> Essa função retornará um erro para alguns tipos de dados se a expressão fizer referência a uma tabela AWS Clean Rooms criada pelo usuário ou a uma tabela do AWS Clean Rooms sistema STL ou STV.

As expressões com os seguintes tipos de dados produzem um erro se fizerem referência a uma tabela criada por usuário ou uma tabela de sistema.

- BOOLEAN
- CHAR
- DATE
- DECIMAL ou NUMERIC
- TIMESTAMP
- VARCHAR

Expressões com os seguintes tipos de dados executam com êxito em tabelas criadas por usuário ou tabelas de sistema STL ou STV:

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

Tipo de retorno

A função LN retorna o mesmo tipo que a expressão.

Exemplo

O seguinte exemplo retorna o logaritmo natural, ou logaritmo de base e, do número 2,718281828:

```
select ln(2.718281828);
ln
--------------------
0.9999999998311267
(1 row)
```

Observe que a resposta é quase igual a 1.

Este exemplo retorna o logaritmo natural dos valores na coluna USERID da tabela 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)
```

Função LOG

Retorna o logaritmo de `expr` com. `base`

Sintaxe

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

Argumento

expr

A expressão deve ter um tipo de dados de número inteiro, decimal ou ponto flutuante.

base

A base para o cálculo do logaritmo. Deve ser um número positivo (não igual a 1) do tipo de dados de precisão dupla.

Tipo de retorno

A função LOG retorna um número de precisão dupla.

Exemplo

O seguinte exemplo retorna o logaritmo de base 10 do número 100:

```
select log(10, 100);
--------
2
(1 row)
```

Função MOD

Retorna o resto de dois números, também chamado de operação modulo. Para calcular o resultado, o primeiro parâmetro é dividido pelo segundo.

Sintaxe

```
MOD(number1, number2)
```

Arguments (Argumentos)

number1

O primeiro parâmetro de entrada é um número INTEGER, SMALLINT, BIGINT ou DECIMAL. Se um dos parâmetros for um tipo DECIMAL, os outros parâmetro também devem ser um tipo DECIMAL. Se um dos parâmetros for um INTEGER, os outros parâmetro podem ser INTEGER, SMALLINT ou BIGINT. Ambos os parâmetros também podem ser SMALLINT ou BIGINT, mas um parâmetro não pode ser SMALLINT se o outro for BIGINT.

number2

O segundo parâmetro é um número INTEGER, SMALLINT, BIGINT ou DECIMAL. As mesmas regras de tipo de dados são válidas para number2 e number1.

Tipo de retorno

Os tipos de retorno válidos são DECIMAL, INT, SMALLINT e BIGINT. O tipo de retorno da função MOD é o mesmo tipo numérico que os parâmetros de entrada se ambos os parâmetros de entrada forem do mesmo tipo. Se um dos parâmetro de entrada for INTEGER, porém, o tipo de retorno também será INTEGER.

Observações de uso

Você pode usar % como um operador de modulo.

Exemplos

O exemplo a seguir retorna o resto da divisão de um número por outro:

```
SELECT MOD(10, 4);

 mod
------
 2
```

O exemplo a seguir retorna um resultado decimal:

```
SELECT MOD(10.5, 4);

 mod
------
 2.5
```

Você pode converter valores de parâmetros:

```
SELECT MOD(CAST(16.4 as integer), 5);

 mod
------
 1
```

Verifique se o primeiro parâmetro é par dividindo-o por 2:

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

 is_even
```

```
--------
 false
```

Você pode usar % como um operador de modulo:

```
SELECT 11 % 4 as remainder;

 remainder
-----------
 3
```

O seguinte exemplo retorna informações para as categorias com números ímpares da tabela 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)
```

Função PI

A função PI retorna o valor de pi para 14 casas decimais.

Sintaxe

```
PI()
```

Tipo de retorno

`DOUBLE PRECISION`

Exemplos

Para retornar o valor de pi, use o exemplo a seguir.

```
SELECT PI();

+------------------+
|        pi        |
+------------------+
| 3.141592653589793 |
+------------------+
```

Função POWER

A função POWER é uma função exponencial que eleva uma expressão numérica para a potência de uma segunda expressão numérica. Por exemplo, 2 elevado à terceira potência é calculado como `POWER(2,3)`, com um resultado de 8.

Sintaxe

```
{POWER(expression1, expression2)
```

Arguments (Argumentos)

expression1

Expressão numérica a ser elevada. Deve ser um tipo de dados `INTEGER`, `DECIMAL` ou `FLOAT`.

expression2

Potência a elevar a expression1. Deve ser um tipo de dados `INTEGER`, `DECIMAL` ou `FLOAT`.

Tipo de retorno

`DOUBLE PRECISION`

Exemplo

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

Função RADIANS

A função RADIANS converte um ângulo em graus em seu equivalente em radianos.

Sintaxe

```
RADIANS(number)
```

Argumento

número

O parâmetro de entrada é um número `DOUBLE PRECISION`.

Tipo de retorno

`DOUBLE PRECISION`

Exemplo

Para retornar o radiano equivalente de 180 graus, use o exemplo a seguir.

```
SELECT RADIANS(180);

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

Função RAND

A função RAND gera um número aleatório de ponto flutuante entre 0 e 1. A função RAND gera um novo número aleatório cada vez que é chamada.

Sintaxe

```
RAND()
```

Tipo de retorno

RANDOM retorna um DOUBLE.

Exemplo

O exemplo a seguir gera uma coluna de números aleatórios de ponto flutuante entre 0 e 1 para cada linha na tabela. `squirrels` A saída resultante seria uma única coluna contendo uma lista de valores decimais aleatórios, com um valor para cada linha na tabela de esquilos.

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

Esse tipo de consulta é útil quando você precisa gerar números aleatórios, por exemplo, para simular eventos aleatórios ou para introduzir aleatoriedade em sua análise de dados. No contexto da `squirrels` tabela, ela pode ser usada para atribuir valores aleatórios a cada esquilo, que podem então ser usados para processamento ou análise posterior.

Função RANDOM

A função RANDOM gera um valor aleatório entre 0,0 (inclusive) e 1,0 (exclusivo).

Sintaxe

```
RANDOM()
```

Tipo de retorno

RANDOM retorna um número de DOUBLE PRECISION.

Exemplos

1. Compute um valor aleatório entre 0 e 99. Se o número aleatório é de 0 a 1, essa consulta produz um número aleatório de 0 a 100:

```
select cast (random() * 100 as int);

INTEGER
```

```
------
24
(1 row)
```

2. Recupere uma amostra aleatória uniforme de 10 itens:

```
select *
from sales
order by random()
limit 10;
```

   Agora recupere uma amostra aleatória de 10 itens, mas escolha os itens em proporção aos preços. Por exemplo, um item cujo preço é o dobro de outro tem duas vezes mais probabilidade de aparecer nos resultados de consulta:

```
select *
from sales
order by log(1 - random()) / pricepaid
limit 10;
```

3. Este exemplo usa o comando SET para definir um valor SEED para que RANDOM gere uma sequência previsível de números.

   Primeiro, retorne três inteiros RANDOM sem definir o valor de SEED primeiro:

```
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)
```

Agora, defina o valor de SEED como `.25` e retorne mais três números RANDOM:

```
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)
```

Por fim, redefina o valor de SEED como `.25` e verifique se RANDOM retorna os mesmos resultados que as três chamadas 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)
```

Função ROUND

A função ROUND arredonda números para o inteiro ou decimal mais próximo.

A função ROUND pode incluir opcionalmente um segundo argumento como um inteiro para indicar o número de casas decimais para arredondamento, em qualquer direção. Quando você não fornece o segundo argumento, a função arredonda para o número inteiro mais próximo. Quando o segundo argumento >n for especificado, a função arredonda para o número mais próximo com n casas decimais de precisão.

Sintaxe

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

Argumento

número

Um número ou expressão avaliada como um número. Pode ser o DECIMAL ou o FLOAT8 tipo. AWS Clean Rooms pode converter outros tipos de dados de acordo com as regras de conversão implícitas.

inteiro (opcional)

Um número inteiro que indica o número de casas decimais para arredondamento em ambas as direções.

Tipo de retorno

ROUND retorna o mesmo tipo de dados numéricos que o(s) argumento(s) de entrada.

Exemplos

Arredonde a comissão paga para determinada transação para o número inteiro mais próximo.

```
select commission, round(commission)
from sales where salesid=10000;
```

```
commission | round
-----------+-------
     28.05 |    28
(1 row)
```

Arredonde a comissão paga para determinada transação para a primeira casa decimal.

```
select commission, round(commission, 1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |  28.1
(1 row)
```

Para a mesma consulta, estenda a precisão no sentido oposto.

```
select commission, round(commission, -1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    30
(1 row)
```

Função SIGN

A função SIGN retorna o sinal (positivo ou negativo) de um número. O resultado da função SIGN é `1`, `-1` ou `0` indicando o sinal do argumento.

Sintaxe

```
SIGN (number)
```

Argumento

número

Número ou expressão que avalia para um número. Pode ser do DECIMALor FLOAT8 tipo. AWS Clean Rooms pode converter outros tipos de dados de acordo com as regras de conversão implícitas.

Tipo de retorno

SIGN retorna o mesmo tipo de dados numéricos que os argumentos de entrada. Se a entrada for DECIMAL, a saída será DECIMAL(1,0).

Exemplos

Para determinar o sinal da comissão paga por determinada transação na tabela SALES, use o exemplo a seguir.

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

Função SIN

SIN é uma função trigonométrica que retorna o seno de um número. O valor de retorno está entre `-1` e 1.

Sintaxe

```
SIN(number)
```

Argumento

número

Um número `DOUBLE PRECISION` em radianos.

Tipo de retorno

`DOUBLE PRECISION`

Exemplo

Para retornar o seno de `-PI`, use o exemplo a seguir.

```
SELECT SIN(-PI());

+-------------------------+
|           sin           |
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

Função SQRT

A função SQRT retorna a raiz quadrada de um valor numérico. A raiz quadrada é um número multiplicado por si mesmo para obter o valor fornecido.

Sintaxe

```
SQRT (expression)
```

Argumento

expressão

A expressão deve ter um tipo de dados de número inteiro, decimal ou ponto flutuante. A expressão pode incluir funções. O sistema pode realizar conversões de tipo implícitas.

Tipo de retorno

SQRT retorna um número de DOUBLE PRECISION.

Exemplos

O exemplo a seguir retorna a raiz quadrada de um número.

```
select sqrt(16);

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

O exemplo a seguir realiza uma conversão de tipo implícita.

```
select sqrt('16');
```

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

O exemplo a seguir aninha funções para realizar uma tarefa mais complexa.

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

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

O exemplo a seguir resulta no comprimento do raio quando dada a área de um círculo. Ele calcula o raio em polegadas, por exemplo, quando dada a área em polegadas quadradas. A área na amostra é 20.

```
select sqrt(20/pi());
```

Isso retorna o valor 5.046265044040321.

O seguinte exemplo retorna a raiz quadrada para valores de COMMISSION da tabela SALES. A coluna COMMISSION é uma coluna DECIMAL. Este exemplo mostra como você pode usar a função em uma consulta com uma lógica condicional mais complexa.

```
select sqrt(commission)
from sales where salesid < 10 order by salesid;

sqrt
------------------
10.4498803820905
3.37638860322683
7.24568837309472
5.1234753829798
...
```

A seguinte consulta retorna raiz quadrada arredondada para o mesmo conjunto de valores de 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 obter mais informações sobre dados de amostra em AWS Clean Rooms, consulte Banco de dados de amostra.

Função TRUNC

A função TRUNC trunca números para o inteiro ou decimal anterior.

A função TRUNC pode incluir opcionalmente um segundo argumento como um inteiro para indicar o número de casas decimais para arredondamento, em qualquer direção. Quando você não fornece o segundo argumento, a função arredonda para o número inteiro mais próximo. Quando o segundo argumento >n for especificado, a função arredonda para o número mais próximo com >n casas decimais de precisão. Esta função também trunca um timestamp e retorna uma data.

Sintaxe

```
TRUNC (number [ , integer ] |
timestamp )
```

Arguments (Argumentos)

número

Um número ou expressão avaliada como um número. Pode ser o DECIMAL ou o FLOAT8 tipo. AWS Clean Rooms pode converter outros tipos de dados de acordo com as regras de conversão implícitas.

inteiro (opcional)

Um inteiro que indica o número de casas decimais de precisão, em um dos sentidos. Se nenhum inteiro for fornecido, o número será truncado como um número inteiro; se um inteiro for especificado, o número será truncado para a casa decimal especificada.

timestamp

A função também pode retornar a data de um timestamp. (Para retornar um valor de timestamp com `00:00:00` como a hora, converta o resultado da função para um timestamp.)

Tipo de retorno

TRUNC retorna o mesmo tipo de dados que o primeiro argumento de entrada. Para timestamps, TRUNC retorna uma data.

Exemplos

Trunque a comissão paga para determinada transação de vendas.

```
select commission, trunc(commission)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   111

(1 row)
```

Trunque o mesmo valor de comissão para a primeira casa decimal.

```
select commission, trunc(commission,1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 | 111.1

(1 row)
```

Trunque a comissão com um valor negativo para o segundo argumento; `111.15` é arredondado para `110`.

```
select commission, trunc(commission,-1)
from sales where salesid=784;

commission | trunc
```

```
-----------+-------
    111.15 |   110
(1 row)
```

Retorna a porção de data do resultado da função SYSDATE (que retorna um timestamp):

```
select sysdate;

timestamp
---------------------------
2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc
------------
2011-07-21
(1 row)
```

Aplique a função TRUNC à uma coluna TIMESTAMP. O tipo de retorno é uma data.

```
select trunc(starttime) from event
order by eventid limit 1;

trunc
------------
2008-01-25
(1 row)
```

Funções escalares

Esta seção descreve as funções escalares suportadas no AWS Clean Rooms Spark SQL. Uma função escalar é uma função que usa um ou mais valores como entrada e retorna um único valor como saída. As funções escalares operam em linhas ou elementos individuais e produzem um único resultado para cada entrada.

As funções escalares, como SIZE, são diferentes de outros tipos de funções SQL, como funções agregadas (contagem, soma, média) e funções geradoras de tabela (explodir, nivelar). Esses outros tipos de função operam em várias linhas ou geram várias linhas, enquanto as funções escalares funcionam em linhas ou elementos individuais.

Tópicos

Função SIZE

A função SIZE usa uma matriz, mapa ou string existente como argumento e retorna um único valor representando o tamanho ou o comprimento dessa estrutura de dados. Isso não cria uma nova estrutura de dados. Ele é usado para consultar e analisar as propriedades das estruturas de dados existentes, em vez de criar novas.

Essa função é útil para determinar o número de elementos em uma matriz ou o comprimento de uma string. Isso pode ser particularmente útil ao trabalhar com matrizes e outras estruturas de dados em SQL, pois permite obter informações sobre o tamanho ou a cardinalidade dos dados.

Sintaxe

```
size(expr)
```

Argumentos

expr

Uma expressão ARRAY, MAP ou STRING.

Tipo de retorno

A função SIZE retorna um INTEIRO.

Exemplo

Neste exemplo, a função SIZE é aplicada à matriz `['b', 'd', 'c', 'a']` e retorna o valor4, que é o número de elementos na matriz.

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

Neste exemplo, a função SIZE é aplicada ao mapa `{'a': 1, 'b': 2}` e retorna o valor2, que é o número de pares de valores-chave no mapa.

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

```
 2
```

Neste exemplo, a função SIZE é aplicada à string `'hello world'` e retorna o valor`11`, que é o número de caracteres na string.

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

Funções de string

Funções de string processam e manipulam strings de caracteres ou expressões que avaliam para strings de caracteres. Quando o argumento string nessas funções é um valor literal, ele deve ser envolvido por aspas simples. Os tipos de dados compatíveis incluem CHAR e VARCHAR.

A próxima seção fornece os nomes de função, sintaxe e descrições para as funções compatíveis. Todos os deslocamentos em strings são baseados em um.

Tópicos

- Função REVERSE
- Função RTRIM
- Função SPLIT
- Função SPLIT_PART
- Função SUBSTRING
- Função TRANSLATE
- Função TRIM
- Função UPPER
- Função UUID

Operador || (Concatenação)

Concatena duas expressões em ambos os lados do símbolo || e retorna a expressão concatenada.

O operador de concatenação é semelhante a Função CONCAT.

> **Note**
>
> Tanto para a função CONCAT como para o operador de concatenação, se uma ou ambas as expressões forem nulas, o resultado da concatenação será null.

Sintaxe

```
expression1 || expression2
```

Argumentos

expression1, expression2

Ambos os argumentos podem ser strings de caracteres ou expressões de comprimento fixo ou variável.

Tipo de retorno

O operador || retorna uma string. O tipo de string é igual ao tipo dos argumentos de entrada.

Exemplo

O seguinte exemplo concatena os campos FIRSTNAME e LASTNAME da tabela 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 colunas que possam conter nulos, use a expressão Funções NVL e COALESCE. O seguinte exemplo usa NVL para retornar um 0 sempre que NULL for encontrado.

```
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
```

Função BTRIM

A função BTRIM apara uma string removendo os espaços em branco iniciais e finais ou removendo caracteres iniciais ou finais que correspondem a uma string opcional especificada.

Sintaxe

```
BTRIM(string [, trim_chars ] )
```

Argumentos

string

A string VARCHAR de entrada a ser cortada.

trim_chars

A string VARCHAR que contém os caracteres a serem correspondidos.

Tipo de retorno

A função BTRIM retorna uma string VARCHAR.

Exemplos

O seguinte exemplo apara espaços em branco inicias e finais da string `' abc '`:

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

O exemplo a seguir remove a string `'xyz'` inicial e final da string `'xyzaxyzbxyzcxyz'`. As ocorrências inicial e final de `'xyz'` são removidas, mas as ocorrências internas da string não são removidas.

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;
```

```
     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

O exemplo a seguir remove as partes iniciais e finais da string `'setuphistorycassettes'` que correspondem a qualquer um dos caracteres na lista trim_chars `'tes'`. Qualquer `t`, `e` ou `s` que ocorra antes que outro caractere que não esteja na lista trim_chars no início ou no final da string de entrada seja removido.

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
----------------
 uphistoryca
```

Função CONCAT

A função CONCAT concatena duas expressões e retorna a expressão resultante. Para concatenar mais de duas strings, use funções CONCAT aninhadas. O operador de concatenação (`||`) entre duas expressões produz os mesmos resultados que a função CONCAT.

> **Note**
>
> Tanto para a função CONCAT como para o operador de concatenação, se uma ou ambas as expressões forem nulas, o resultado da concatenação será null.

Sintaxe

```
CONCAT ( expression1, expression2 )
```

Argumentos

expression1, expression2

Os dois argumentos podem ser uma cadeia de caracteres de comprimento fixo, uma cadeia de caracteres de comprimento variável, uma expressão binária ou uma expressão que é avaliada para uma dessas entradas.

Tipo de retorno

CONCAT retorna uma expressão. O tipo de dados da expressão é o mesmo tipo dos argumentos de entrada.

Se as expressões de entrada forem de tipos diferentes, AWS Clean Rooms tentará digitar implicitamente uma das expressões. Se os valores não puderem ser convertidos, será retornado um erro.

Exemplos

O seguinte exemplo concatena dois literais de caracteres:

```
select concat('December 25, ', '2008');

concat
-------------------
December 25, 2008
(1 row)
```

A seguinte consulta, usando o operador `||` em vez de CONCAT, produz o mesmo resultado:

```
select 'December 25, '||'2008';

concat
-------------------
December 25, 2008
(1 row)
```

O seguinte exemplo usa duas funções CONCAT para concatenar três strings de caracteres:

```
select concat('Thursday, ', concat('December 25, ', '2008'));

concat
----------------------------
Thursday, December 25, 2008
(1 row)
```

Para concatenar colunas que possam conter nulos, use a Funções NVL e COALESCE. O seguinte exemplo usa NVL para retornar um 0 sempre que NULL for encontrado.

```
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)
```

A seguinte consulta concatena os valores CITY e STATE da tabela VENUE:

```
select concat(venuecity, venuestate)
from venue
where venueseats > 75000
order by venueseats;

concat
-------------------
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

A seguinte consulta usa funções CONCAT aninhadas. A consulta concatena os valores CITY e STATE da tabela VENUE, mas delimita a string resultante com uma vírgula e um espaço:

```
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)
```

função FORMAT_STRING

A função FORMAT_STRING cria uma string formatada substituindo os espaços reservados em uma string de modelo pelos argumentos fornecidos. Ele retorna uma string formatada de strings de formato no estilo printf.

A função FORMAT_STRING funciona substituindo os espaços reservados na string do modelo pelos valores correspondentes passados como argumentos. Esse tipo de formatação de string pode ser útil quando você precisa construir dinamicamente cadeias de caracteres que incluam uma mistura de texto estático e dados dinâmicos, como ao gerar mensagens de saída, relatórios ou outros tipos de texto informativo. A função FORMAT_STRING fornece uma maneira concisa e legível de criar esses tipos de strings formatadas, facilitando a manutenção e a atualização do código que gera a saída.

Sintaxe

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

Argumentos

strfmt

Uma expressão STRING.

obj

Uma STRING ou expressão numérica.

Tipo de retorno

FORMAT_STRING retorna uma STRING.

Exemplo

O exemplo a seguir contém uma string de modelo que contém dois espaços reservados: `%d` para um valor decimal (inteiro) e `%s` para um valor de string. O `%d` espaço reservado é substituído pelo valor decimal (inteiro) () e o espaço reservado %s é substituído pelo valor da string (`100`). `"days"` A saída é uma string de modelo com os espaços reservados substituídos pelos argumentos fornecidos:`"Hello World 100 days"`.

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

Funções LEFT e RIGHT

Essas funções retornam o número especificado de caracteres mais à esquerda ou mais à direita de uma string de caracteres.

O número é baseado no número de caracteres, e não bytes, de forma que caracteres multibyte são contados como caracteres simples.

Sintaxe

```
LEFT ( string,  integer )

RIGHT ( string,  integer )
```

Argumentos

string

Qualquer string de caracteres ou qualquer expressão que avalie para uma string de caracteres.

inteiro

Um inteiro positivo.

Tipo de retorno

LEFT e RIGHT retornam uma string VARCHAR.

Exemplo

O exemplo a seguir retorna os 5 caracteres mais à esquerda e os 5 caracteres mais à direita de nomes de eventos que têm IDs entre 1000 e 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)
```

Função LENGTH

Função LOWER

Converte uma string em letras minúsculas. LOWER é compatível com caracteres UTF-8 multibyte, até o máximo de quatro bytes por caractere.

Sintaxe

```
LOWER(string)
```

Argumento

string

O parâmetro de entrada é uma string VARCHAR (ou qualquer outro tipo de dados, como CHAR, que pode ser convertido implicitamente para VARCHAR).

Tipo de retorno

A função LOWER retorna uma string de caractere no mesmo tipo de dados que a string de entrada.

Exemplos

O exemplo a seguir converte o campo CATNAME em 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)
```

Funções LPAD e RPAD

Essas funções inserem caracteres no início ou final de uma string com base em um comprimento especificado.

Sintaxe

```
LPAD (string1, length, [ string2 ])
```

```
RPAD (string1, length, [ string2 ])
```

Argumentos

string1

Uma string de caracteres ou uma expressão que avalie para uma string de caracteres, tal como o nome de uma coluna de caracteres.

length

Um inteiro que define o comprimento dos resultados da função. O comprimento de uma string é baseado no número de caracteres, e não bytes, de forma que caracteres multibyte são contados como caracteres simples. Se string1 for mais longa que o comprimento especificado, ela será truncada (à direita). Se length for um número negativo, o resultado da função será uma string vazia.

string2

Um ou mais caracteres inseridos no início ou no fim da string1. Este argumento é opcional; se ele não é especificado, espaços são usados.

Tipo de retorno

Essas funções retornam um tipo de dados VARCHAR.

Exemplos

Trunque um conjunto específico de nomes de eventos para 20 caracteres e insira espaços no início dos nomes mais curtos:

```
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)
```

Trunque o mesmo conjunto de nomes de eventos para 20 caracteres, mas insira no início dos nomes mais curtos `0123456789`.

```
select rpad(eventname,20,'0123456789') from event
where eventid between 1 and 5 order by 1;

 rpad
--------------------
Boris Godunov0123456
Gotterdammerung01234
Il Trovatore01234567
La Cenerentola (Cind
Salome01234567890123
(5 rows)
```

Função LTRIM

Corta caracteres do início de uma string. Remove a string mais longa que contém somente caracteres que estão na lista de caracteres de corte. O corte é concluído quando um caractere de corte não aparece na string de entrada.

Sintaxe

```
LTRIM( string [, trim_chars] )
```

Argumentos

string

Uma coluna, expressão ou literal de string a ser cortado.

trim_chars

Uma coluna, expressão ou literal de string que representa os caracteres a serem cortados do começo da string. Se não for especificado, um espaço será usado como caractere de corte.

Tipo de retorno

A função LTRIM retorna uma string no mesmo tipo de dado que a string de entrada (CHAR ou VARCHAR).

Exemplos

O exemplo a seguir corta o ano da coluna `listime`. Os caracteres de corte no literal de string `'2008-'` indicam os caracteres a serem cortados da esquerda. Se você usar os caracteres de corte `'028-'`, obterá o mesmo 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 remove qualquer um dos caracteres em trim_chars quando eles aparecem no início da string. O seguinte exemplo apara os caracteres “C”, “D” e “G” quando eles aparecem no início de VENUENAME, que é uma coluna 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
```

O exemplo a seguir usa o caractere de corte 2 que é recuperado da coluna `venueid`.

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

O exemplo a seguir não corta nenhum caractere porque 2 é encontrado antes do caractere de corte `'0'`.

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

O exemplo a seguir usa o caractere de corte de espaço padrão e corta os dois espaços do início da string.

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

Função POSITION

Retorna a localização da substring especificada dentro de uma string.

Sintaxe

```
POSITION(substring IN string )
```

Argumentos

substring

A substring a procurar dentro da string.

string

A string ou coluna a ser procurada.

Tipo de retorno

A função POSITION retorna um inteiro correspondente à posição da substring (baseada em 1, não baseada em zero). A posição é baseada no número de caracteres, e não bytes, de forma que caracteres multibyte são contados como caracteres simples.

Observações de uso

POSITION retornará 0 se a substring não for localizada dentro da string:

```
select position('dog' in 'fish');

position
----------
```

```
 0
(1 row)
```

Exemplos

O seguinte exemplo mostra a posição da string `fish` na palavra `dogfish`:

```
select position('fish' in 'dogfish');

position
----------
 4
(1 row)
```

O seguinte exemplo retorna o número de transações de vendas com uma COMMISSION acima de 999,00 da tabela 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)
```

Função REGEXP_COUNT

Pesquisa uma string quanto a um padrão de expressão regular e retorna um inteiro que indica o número de vezes que o padrão ocorre na string. Se nenhuma correspondência for encontrada, a função retornará 0.

Sintaxe

```
REGEXP_COUNT ( source_string, pattern [, position [, parameters ] ] )
```

Argumentos

source_string

Uma expressão de string, tal como um nome de coluna, a ser procurada.

pattern

Um literal de string que representa um padrão de expressão regular.

position

Um inteiro positivo que indica a posição em source_string para começar a pesquisar. A posição é baseada no número de caracteres, e não bytes, de forma que caracteres multibyte são contados como caracteres simples. O padrão é um. Se a posição for menor que 1, a pesquisa começará no primeiro caractere da source_string. Se position for maior que o número de caracteres na source_string, o resultado será 0.

parameters

Uma ou mais literais de sequências que indicam como a função corresponde o padrão. Os valores possíveis são os seguintes:

- c – Executa a correspondência diferenciando maiúsculas e minúsculas. O padrão é usar a correspondência diferenciando maiúsculas e minúsculas.
- i – Executa a correspondência sem diferenciar maiúsculas de minúsculas.
- p — Interpreta o padrão com o dialeto de expressão regular compatível com Perl (PCRE - Perl Compatible Regular Expression).

Tipo de retorno

Inteiro

Exemplo

O seguinte exemplo conta o número de vezes que uma sequência de três letra ocorre.

```
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');

 regexp_count
 --------------
            8
```

O seguinte exemplo conta o número de vezes que nome de domínio de nível superior é `org` ou `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
```

O exemplo a seguir conta as ocorrências da string F0X usando a correspondência sem diferenciar maiúsculas de minúsculas.

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

 regexp_count
 --------------
            1
```

O exemplo a seguir usa um padrão escrito no dialeto PCRE para localizar palavras contendo pelo menos um número e uma letra minúscula. Ele usa o operador ?=, que tem uma conotação específica look-ahead em PCRE. Este exemplo conta o número de ocorrências de tais palavras, com correspondência diferenciando maiúsculas de minúsculas.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

 regexp_count
 --------------
            2
```

O exemplo a seguir usa um padrão escrito no dialeto PCRE para localizar palavras contendo pelo menos um número e uma letra minúscula. Ele usa o operador ?=, que tem uma conotação específica em PCRE. Este exemplo conta o número de ocorrências de tais palavras, mas difere do exemplo anterior na medida em que usa correspondência sem diferenciar maiúsculas dee minúsculas.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');

 regexp_count
 --------------
            3
```

Função REGEXP_INSTR

Pesquisa um padrão de expressão regular em uma sequência e retorna um inteiro que indica a posição inicial ou final da subsequência correspondente. Se nenhuma correspondência for encontrada, a função retornará 0. REGEXP_INSTR é semelhante à função POSITION, mas permite que você pesquise um padrão de expressão regular em uma sequência.

Sintaxe

```
REGEXP_INSTR ( source_string, pattern [, position [, occurrence] [, option
 [, parameters ] ] ] ] )
```

Argumentos

source_string

Uma expressão de string, tal como um nome de coluna, a ser procurada.

pattern

Um literal de string que representa um padrão de expressão regular.

position

Um inteiro positivo que indica a posição em source_string para começar a pesquisar. A posição é baseada no número de caracteres, e não bytes, de forma que caracteres multibyte são contados como caracteres simples. O padrão é um. Se a posição for menor que 1, a pesquisa começará no primeiro caractere da source_string. Se position for maior que o número de caracteres na source_string, o resultado será 0.

occurrence

Um inteiro positivo que indica qual ocorrência do padrão usar. REGEXP_INSTR ignora as primeiras correspondências de occurrence -1. O padrão é um. Se ocurrence for menor que 1 ou maior que o número de caracteres em source_string, a pesquisa será ignorada e o resultado será 0.

option

Um valor que indica se retornar a posição do primeiro caractere da correspondência (`0`) ou a posição do primeiro caractere seguinte ao final da correspondência (1). Um valor diferente de zero é o mesmo que 1. O valor padrão é 0.

parameters

Uma ou mais literais de sequências que indicam como a função corresponde o padrão. Os valores possíveis são os seguintes:

- c – Executa a correspondência diferenciando maiúsculas e minúsculas. O padrão é usar a correspondência diferenciando maiúsculas e minúsculas.
- i – Executa a correspondência sem diferenciar maiúsculas de minúsculas.
- e – Extrai uma subsequência usando uma subexpressão.

  Se o padrão incluir uma subexpressão, REGEXP_INSTR corresponderá uma subsequência usando a primeira subexpressão em padrão. REGEXP_INSTR considera apenas a primeira subexpressão. As subexpressões adicionais são ignoradas. Se o padrão não tiver uma subexpressão, REGEXP_INSTR ignorará o parâmetro 'e'.
- p — Interpreta o padrão com o dialeto de expressão regular compatível com Perl (PCRE - Perl Compatible Regular Expression).

Tipo de retorno

Inteiro

Exemplo

O seguinte exemplo procura pelo caractere @ que inicia o nome de um domínio e retorna a posição inicial da primeira correspondência.

```
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
```

O seguinte exemplo procura por variações da palavra `Center` e retorna a posição inicial da primeira correspondência.

```
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
```

O exemplo a seguir encontra a posição inicial da primeira ocorrência da string FOX usando lógica de correspondência sem diferenciar maiúsculas de minúsculas.

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

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

O exemplo a seguir usa um padrão escrito em dialeto PCRE para localizar palavras contendo pelo menos um número e uma letra minúscula. Ele usa o operador ?=, que tem uma conotação específica look-ahead em PCRE. Este exemplo encontra a posição inicial da segunda palavra.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

 regexp_instr
 --------------
           21
```

O exemplo a seguir usa um padrão escrito em dialeto PCRE para localizar palavras contendo pelo menos um número e uma letra minúscula. Ele usa o operador ?=, que tem uma conotação específica look-ahead em PCRE. Este exemplo localiza a posição inicial da segunda palavra, mas difere do exemplo anterior na medida em que usa correspondência sem diferenciar maiúsculas de minúsculas.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');
```

```
 regexp_instr
 --------------
           15
```

Função REGEXP_REPLACE

Pesquisa uma string quanto a um padrão de expressão regular e substitui cada ocorrência do padrão pela string especificada. REGEXP_REPLACE é semelhante a Função REPLACE, mas permite que você pesquise uma string quanto a um padrão de expressão regular.

REGEXP_REPLACE é semelhante a Função TRANSLATE e Função REPLACE, exceto que TRANSLATE faz várias substituições de caractere único e REPLACE substitui uma string inteira por outra string, enquanto REGEXP_REPLACE permite que você pesquise uma string quanto a um padrão de expressão regular.

Sintaxe

```
REGEXP_REPLACE ( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

Argumentos

source_string

Uma expressão de string, tal como um nome de coluna, a ser procurada.

pattern

Um literal de string que representa um padrão de expressão regular.

replace_string

Uma expressão de string, tal como um nome de coluna, que substituirá cada ocorrência do padrão. O padrão é uma string vazia ("").

position

Um inteiro positivo que indica a posição em source_string para começar a pesquisar. A posição é baseada no número de caracteres, e não bytes, de forma que caracteres multibyte são contados como caracteres simples. O padrão é um. Se a posição for menor que 1, a pesquisa começará no primeiro caractere da source_string. Se position for maior que o número de caracteres na source_string, o resultado será source_string.

parameters

Uma ou mais literais de sequências que indicam como a função corresponde o padrão. Os valores possíveis são os seguintes:

- c – Executa a correspondência diferenciando maiúsculas e minúsculas. O padrão é usar a correspondência diferenciando maiúsculas e minúsculas.
- i – Executa a correspondência sem diferenciar maiúsculas de minúsculas.
- p — Interpreta o padrão com o dialeto de expressão regular compatível com Perl (PCRE - Perl Compatible Regular Expression).

Tipo de retorno

VARCHAR

Se pattern ou replace_string for NULL, o retorno será NULL.

Exemplo

O seguinte exemplo exclui o @ e o nome de domínio dos endereços de e-mail.

```
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
```

O seguinte exemplo substitui os nomes de domínio de endereços de e-mail por esse 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
```

O exemplo a seguir conta as ocorrências da string `FOX` no valor `quick brown fox` usando a correspondência sem diferenciar maiúsculas de minúsculas.

```
SELECT regexp_replace('the fox', 'FOX', 'quick brown fox', 1, 'i');

   regexp_replace
---------------------
 the quick brown fox
```

O exemplo a seguir usa um padrão escrito no dialeto PCRE para localizar palavras contendo pelo menos um número e uma letra minúscula. Ele usa o operador `?=`, que tem uma conotação específica look-ahead em PCRE. Este exemplo substitui cada ocorrência de tal palavra pelo valor `[hidden]`.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

        regexp_replace
------------------------------
 [hidden] plain A1234 [hidden]
```

O exemplo a seguir usa um padrão escrito no dialeto PCRE para localizar palavras contendo pelo menos um número e uma letra minúscula. Ele usa o operador `?=`, que tem uma conotação específica look-ahead em PCRE. Este exemplo substitui cada ocorrência de tal palavra pelo valor`[hidden]`, mas difere do exemplo anterior na medida em que ele usa correspondência sem diferenciar maiúsculas de minúsculas.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

          regexp_replace
---------------------------------
```

```
 [hidden] plain [hidden] [hidden]
```

Função REGEXP_SUBSTR

Retorna os caracteres de uma string ao procurar por um padrão de expressão regular. REGEXP_SUBSTR é semelhante a função Função SUBSTRING, mas permite que você pesquise uma string quanto a um padrão de expressão regular. Se a função não conseguir corresponder a expressão regular com nenhum caractere na string, ela retornará uma string vazia.

Sintaxe

```
REGEXP_SUBSTR ( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

Argumentos

source_string

Uma expressão de string a ser pesquisada.

pattern

Um literal de string que representa um padrão de expressão regular.

position

Um inteiro positivo que indica a posição em source_string para começar a pesquisar. A posição é baseada no número de caracteres, e não bytes, de forma que caracteres multibyte são contados como caracteres simples. O padrão é um. Se a posição for menor que 1, a pesquisa começará no primeiro caractere da source_string. Se position for maior que o número de caracteres na source_string, o resultado será uma string vazia ("").

occurrence

Um inteiro positivo que indica qual ocorrência do padrão usar. REGEXP_SUBSTR ignora as primeiras correspondências de occurrence -1. O padrão é um. Se a ocorrência for menor que 1 ou maior que o número de caracteres em source_string, a pesquisa será ignorada e o resultado será NULL.

parameters

Uma ou mais literais de sequências que indicam como a função corresponde o padrão. Os valores possíveis são os seguintes:

- c – Executa a correspondência diferenciando maiúsculas e minúsculas. O padrão é usar a correspondência diferenciando maiúsculas e minúsculas.
- i – Executa a correspondência sem diferenciar maiúsculas de minúsculas.
- e – Extrai uma subsequência usando uma subexpressão.

  Se o padrão incluir uma subexpressão, REGEXP_SUBSTR corresponderá uma subsequência usando a primeira subexpressão em padrão. Uma subexpressão é uma expressão dentro do padrão que está entre parênteses. Por exemplo, para o padrão `'This is a (\\w+)'` faz correspondência com a primeira expressão com a string `'This is a '` seguida por uma palavra. Em vez de retornar o padrão, REGEXP_SUBSTR com o parâmetro e retorna somente a string dentro da subexpressão.

  REGEXP_SUBSTR considera apenas a primeira subexpressão. As subexpressões adicionais são ignoradas. Se o padrão não tiver uma subexpressão, REGEXP_SUBSTR ignorará o parâmetro 'e'.
- p — Interpreta o padrão com o dialeto de expressão regular compatível com Perl (PCRE - Perl Compatible Regular Expression).

Tipo de retorno

VARCHAR

Exemplo

O seguinte exemplo retorna a porção de um endereço de e-mail entre o caractere @ e a extensão de domínio.

```
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
```

O exemplo a seguir retorna a porção da entrada correspondente à primeira ocorrência da string FOX com a correspondência sem diferenciar maiúsculas de minúsculas.

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

 regexp_substr
---------------
 fox
```

O exemplo a seguir retorna a primeira parte da entrada que começa com letras minúsculas. Isso é funcionalmente idêntico à mesma instrução SELECT sem o parâmetro c.

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');

 regexp_substr
---------------
 abc
```

O exemplo a seguir usa um padrão escrito no dialeto PCRE para localizar palavras contendo pelo menos um número e uma letra minúscula. Ele usa o operador ?=, que tem uma conotação específica look-ahead em PCRE. Este exemplo retorna a parte da entrada correspondente à segunda palavra.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

O exemplo a seguir usa um padrão escrito no dialeto PCRE para localizar palavras contendo pelo menos um número e uma letra minúscula. Ele usa o operador ?=, que tem uma conotação específica look-ahead em PCRE. Este exemplo retorna a parte da entrada correspondente à segunda palavra, mas difere do exemplo anterior na medida em que usa a correspondência sem diferenciar maiúsculas de minúsculas.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
---------------
```

```
 A1234
```

O exemplo a seguir usa uma subexpressão para encontrar a segunda string correspondente ao padrão `'this is a (\\w+)'` usando a correspondência que não diferencia letras maiúsculas de minúsculas. Ele retorna a subexpressão dentro dos parênteses.

```
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
```

Função REPEAT

Repete uma string pelo número especificado de vezes. Se o parâmetro de entrada for numérico, REPEAT o tratará como uma string.

Sintaxe

```
REPEAT(string, integer)
```

Argumentos

string

O primeiro parâmetro de entrada é a string a ser repetida.

inteiro

O segundo parâmetro é um inteiro indicando o número de vezes a repetir a string.

Tipo de retorno

A função REPEAT retorna uma string.

Exemplos

O seguinte exemplo repete o valor da coluna CATID na tabela CATEGORY três vezes:

```
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)
```

Função REPLACE

Substitui todas as ocorrências de um conjunto de caracteres em uma string existente por outros caracteres especificados.

REPLACE é semelhante a Função TRANSLATE e Função REGEXP_REPLACE, exceto que TRANSLATE faz várias substituições de caractere único e REGEXP_REPLACE permite que você pesquise uma string quanto a um padrão de expressão regular, enquanto REPLACE substitui uma string inteira por outra string.

Sintaxe

```
REPLACE(string1, old_chars, new_chars)
```

Argumentos

string

String CHAR ou VARCHAR a ser procurada.

old_chars

String CHAR ou VARCHAR a substituir.

new_chars

Nova string CHAR ou VARCHAR que substitui old_string.

Tipo de retorno

VARCHAR

Se old_chars ou new_chars for NULL, o retorno será NULL.

Exemplos

O seguinte exemplo converte a string `Shows` em `Theatre` no 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)
```

Função REVERSE

A função REVERSE opera em uma string e retorna os caracteres na ordem reversa. Por exemplo, `reverse('abcde')` retorna `edcba`. Essa função funciona em tipos de dados numéricos e de data, assim como nos tipos de dados de caracteres; no entanto, na maioria dos casos ela possui um valor prático para strings de caracteres.

Sintaxe

```
REVERSE ( expression )
```

Argumento

expressão

Uma expressão com um tipo de dados de caractere, data, timestamp ou numérico que representa o destino da reversão de caracteres. Todas as expressões são convertidas implicitamente em strings de comprimento variável. Os espaços em branco finais em strings de caracteres de largura fixa são ignorados.

Tipo de retorno

REVERSE retorna um VARCHAR.

Exemplos

Selecione cinco nomes de cidade distintos e seus nomes invertidos correspondentes da tabela 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)
```

Selecione cinco vendas IDs e seu IDs elenco invertido correspondente como sequências 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)
```

Função RTRIM

A função RTRIM apara um conjunto específico de caracteres do final de uma string. Remove a string mais longa que contém somente caracteres que estão na lista de caracteres de corte. O corte é concluído quando um caractere de corte não aparece na string de entrada.

Sintaxe

```
RTRIM( string, trim_chars )
```

Argumentos

string

Uma coluna, expressão ou literal de string a ser cortado.

trim_chars

Uma coluna, expressão ou literal de string que representa os caracteres a serem cortados do final da string. Se não for especificado, um espaço será usado como caractere de corte.

Tipo de retorno

Uma string no mesmo tipo de dados que o argumento da string.

Exemplo

O seguinte exemplo apara espaços em branco inicias e finais da string `' abc '`:

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

O exemplo a seguir remove a string `'xyz'` final da string `'xyzaxyzbxyzcxyz'`. As ocorrências iniciais de `'xyz'` são removidas, mas as ocorrências internas da string não são removidas.

```
select 'xyzaxyzbxyzcxyz' as untrim,
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

O exemplo a seguir remove as partes finais da string `'setuphistorycassettes'` que correspondem a qualquer um dos caracteres na lista trim_chars `'tes'`. Qualquer `t`, `e` ou `s` que ocorra antes que outro caractere que não esteja na lista trim_chars no final da string de entrada é removido.

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

O seguinte exemplo apara os caracteres "Park" do final de VENUENAME, onde presente:

```
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
```

Observe que RTRIM remove qualquer um dos caracteres `P`, `a`, `r` ou `k` que aparecem no final de um VENUENAME.

Função SPLIT

A função SPLIT permite extrair substrings de uma string maior e trabalhar com elas como uma matriz. A função SPLIT é útil quando você precisa dividir uma string em componentes individuais com base em um delimitador ou padrão específico.

Sintaxe

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

Argumentos

str

Uma expressão de string para dividir.

regex

Uma string representando uma expressão regular. A string regex deve ser uma expressão regular Java.

limit

Uma expressão inteira que controla o número de vezes que o regex é aplicado.

- limite > 0: o comprimento da matriz resultante não será maior que o limite, e a última entrada da matriz resultante conterá todas as entradas além da última regex correspondente.
- limit <= 0: o regex será aplicado quantas vezes for possível, e a matriz resultante pode ser de qualquer tamanho.

Tipo de retorno

A função SPLIT retorna um ARRAY<STRING>.

Se `limit > 0`: O comprimento da matriz resultante não será maior que o limite, e a última entrada da matriz resultante conterá todas as entradas além da última regex correspondente.

If `limit <= 0`: regex será aplicado quantas vezes for possível e a matriz resultante poderá ser de qualquer tamanho.

Exemplo

Neste exemplo, a função SPLIT divide a string de entrada `'oneAtwoBthreeC'` sempre que encontra os caracteres `'A''B'`, ou `'C'` (conforme especificado pelo padrão de expressão regular). `'[ABC]'` A saída resultante é uma matriz de quatro elementos:`"one"`, `"two""three"`, e uma string vazia`""`.

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

Função SPLIT_PART

Divide uma string no delimitador especificado e retorna a parte na posição especificada.

Sintaxe

```
SPLIT_PART(string, delimiter, position)
```

Argumentos

string

Uma coluna, expressão ou literal de string a ser dividido. A string pode ser CHAR ou VARCHAR.

delimitador

A string delimitadora que indica seções da string de entrada.

Se o delimitador for um literal, coloque-o entre aspas simples.

position

Posição da porção da string a retornar (contando de 1). Deve ser um número inteiro maior que 0. Se position for maior que o número de porções de string, SPLIT_PART retornará uma string vazia. Se delimiter não for encontrado em string, o valor retornado conterá o conteúdo da parte especificada, que poderá ser toda a string ou um valor vazio.

Tipo de retorno

Uma string CHAR ou VARCHAR, o mesmo que o parâmetro da string.

Exemplos

O exemplo a seguir divide uma string literal em partes usando o delimitador $ e retorna a segunda parte.

```
select split_part('abc$def$ghi','$',2)

split_part
----------
def
```

O exemplo a seguir divide uma string literal em partes usando o delimitador $. Ele retorna uma string vazia porque a parte 4 não foi encontrada.

```
select split_part('abc$def$ghi','$',4)

split_part
----------

```

O exemplo a seguir divide uma string literal em partes usando o delimitador #. Ele retorna a string inteira, que é a primeira parte, porque o delimitador não foi encontrado.

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

O exemplo a seguir divide o campo de timestamp LISTTIME em componentes de ano, mês e dia.

```
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
```

O seguinte exemplo seleciona o campo de timestamp LISTTIME e o divide no caractere `'-'` para obter o mês (a segunda parte da string LISTTIME) e, então, conta o número de entradas para cada mês:

```
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
```

Função SUBSTRING

Retorna o subconjunto de uma string com base na posição inicial especificada da string.

Se a entrada for uma cadeia de caracteres, a posição inicial e o número de caracteres extraídos são baseados nos caracteres, e não bytes, de forma que caracteres multibyte são contados como caracteres simples. Se a entrada for uma expressão binária, a posição inicial e a substring extraída são baseadas em bytes. Você não pode especificar um comprimento negativo, mas pode especificar uma posição inicial negativa.

Sintaxe

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

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

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

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

Argumentos

cadeia de caracteres

A string a ser pesquisada. Tipos de dados não caracteres são tratados como uma string.

start_position

A posição dentro da sequência para começar a extração, começando em 1. A start_position é baseada no número de caracteres, e não bytes, de forma que caracteres multibyte são contados como caracteres simples. Esse número pode ser negativo.

caracteres numéricos

O número de caracteres a extrair (o comprimento da substring). Os caracteres numéricos são baseados no número de caracteres, não em bytes, de modo que os caracteres de vários bytes sejam contados como caracteres únicos. Esse número não pode ser negativo.

start_byte

A posição dentro da expressão binária para começar a extração, começando por 1. Esse número pode ser negativo.

número de bytes

O número de bytes a serem extraídos, ou seja, o comprimento da substring. Esse número não pode ser negativo.

Tipo de retorno

VARCHAR

Notas de uso para cadeias de caracteres

O seguinte exemplo retorna uma string de quatro caracteres começando com o sexto caractere.

```
select substring('caterpillar',6,4);
substring
-----------
```

```
pill
(1 row)
```

Se os caracteres start_position + numbecharacters excederem o comprimento da string, SUBSTRING retornará uma substring começando da start_position até o final da string. Por exemplo:

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

Se `start_position` for negativa ou 0, a função SUBSTRING retornará uma substring começando no primeiro caractere da string com um comprimento de `start_position` + `numbecharacters` -1. Por exemplo:

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

Se `start_position` + `numbecharacters` -1 for menor ou igual a zero, a SUBSTRING retornará uma string vazia. Por exemplo:

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

Exemplos

O seguinte exemplo retorna o mês da string LISTTIME na tabela 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)
```

O seguinte exemplo é o mesmo que o exemplo acima, mas usa a opção 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)
```

Não é possível usar a SUBSTRING para extrair previsivelmente o prefixo de uma string que possa conter caracteres multibyte, pois é necessário especificar o comprimento de uma string multibyte com base no número de bytes, e não no número de caracteres. Para extrair o segmento inicial de uma sequência com base no comprimento em bytes, você pode CAST a string como VARCHAR (byte_length) para truncar a string, onde byte_length é o tamanho exigido. O seguinte exemplo extrai os primeiros cinco bytes da string `'Fourscore and seven'`.

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
Fours
```

O exemplo a seguir retorna o nome `Ana` que aparece após o último espaço na string de entrada `Silva, Ana`.

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))

 reverse
-----------
 Ana
```

Função TRANSLATE

Para dada expressão, substitui todas as ocorrências dos caracteres especificados pelos substitutos especificados. Os caracteres existentes são mapeados aos caracteres de substituição pelas suas posições nos argumentos characters_to_replace e characters_to_substitute. Se mais caracteres estiverem especificados no argumento characters_to_replace que no argumento characters_to_substitute, os caracteres adicionais do argumento characters_to_replace serão omitidos do valor de retorno.

TRANSLATE é semelhante a Função REPLACE e Função REGEXP_REPLACE, exceto que REPLACE substitui uma string inteira por outra string e REGEXP_REPLACE permite que você pesquise uma string quanto a um padrão de expressão regular, enquanto TRANSLATE faz várias substituições de caracteres simples.

Se qualquer um dos argumentos for nulo, o retorno será NULL.

Sintaxe

```
TRANSLATE ( expression, characters_to_replace, characters_to_substitute )
```

Argumentos

expressão

A expressão a ser traduzida.

characters_to_replace

Uma string contendo os caracteres a serem substituídos.

characters_to_substitute

Uma string contendo os caracteres a substituir.

Tipo de retorno

VARCHAR

Exemplos

O seguinte exemplo substitui vários caracteres em uma string:

```
select translate('mint tea', 'inea', 'osin');

translate
-----------
most tin
```

O seguinte exemplo substitui o sinal (@) por um ponto final para todos os valores em uma coluna:

```
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
```

O seguinte exemplo substitui espaços por sublinhados e remove pontos finais de todos os valores em uma coluna:

```
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
```

Função TRIM

Apara uma string removendo os espaços em branco iniciais e finais ou removendo caracteres iniciais ou finais que correspondem a uma string opcional especificada.

Sintaxe

```
TRIM( [ BOTH ] [ trim_chars FROM ] string
```

Argumentos

trim_chars

(Opcional) Os caracteres a serem aparados da string. Se este parâmetro for omitido, espaços em branco serão aparados.

string

A string a ser aparada.

Tipo de retorno

A função TRIM retorna uma string VARCHAR ou CHAR. Se você usar a função TRIM com um comando SQL, converte AWS Clean Rooms implicitamente os resultados em VARCHAR. Se você usar a função TRIM na lista SELECT para uma função SQL, AWS Clean Rooms isso não converte implicitamente os resultados e talvez seja necessário realizar uma conversão explícita para evitar um erro de incompatibilidade de tipos de dados. Consulte a Função CAST função para obter informações sobre conversões explícitas.

Exemplo

O seguinte exemplo apara espaços em branco inicias e finais da string `' abc '`:

```
select '     abc    ' as untrim, trim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

O seguinte exemplo remove as aspas duplas que cercam a string `"dog"`:

```
select trim('"' FROM '"dog"');

btrim
-------
dog
```

TRIM remove qualquer um dos caracteres em trim_chars quando eles aparecem no início da string. O seguinte exemplo apara os caracteres “C”, “D” e “G” quando eles aparecem no início de VENUENAME, que é uma coluna 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
```

Função UPPER

Converte uma string em letras maiúsculas. UPPER é compatível com caracteres UTF-8 multibyte, até o máximo de quatro bytes por caractere.

Sintaxe

```
UPPER(string)
```

Argumentos

string

O parâmetro de entrada é uma string VARCHAR (ou qualquer outro tipo de dados, como CHAR, que pode ser convertido implicitamente para VARCHAR).

Tipo de retorno

A função UPPER retorna uma string de caracteres que é o mesmo tipo de dados da string de entrada.

Exemplos

O seguinte exemplo converte o campo CATNAME para maiúsculas:

```
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)
```

Função UUID

A função UUID gera um Identificador Único Universal (UUID).

UUIDs são identificadores globais exclusivos que são comumente usados para fornecer identificadores exclusivos para várias finalidades, como:

- Identificação de registros de banco de dados ou outras entidades de dados.
- Gerando nomes ou chaves exclusivos para arquivos, diretórios ou outros recursos.
- Rastreamento e correlação de dados em sistemas distribuídos.
- Fornecendo identificadores exclusivos para pacotes de rede, componentes de software ou outros ativos digitais.

A função UUID gera um valor de UUID que é exclusivo com uma probabilidade muito alta, mesmo em sistemas distribuídos e por longos períodos de tempo. UUIDs normalmente são gerados usando uma combinação do timestamp atual, do endereço de rede do computador e outros dados aleatórios ou pseudo-aleatórios, garantindo que é altamente improvável que cada UUID gerado entre em conflito com qualquer outro UUID.

No contexto de uma consulta SQL, a função UUID pode ser usada para gerar identificadores exclusivos para novos registros inseridos em um banco de dados ou para fornecer chaves exclusivas

para particionamento de dados, indexação ou outras finalidades em que um identificador exclusivo seja necessário.

> **Note**
>
> A função UUID não é determinística.

Sintaxe

```
uuid()
```

Argumentos

A função UUID não aceita argumentos.

Tipo de retorno

O UUID retorna uma string de identificador exclusivo universal (UUID). O valor é retornado como uma string canônica de 36 caracteres do UUID.

Exemplo

O exemplo a seguir gera um Identificador Único Universal (UUID). A saída é uma string de 36 caracteres representando um identificador universal exclusivo.

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

Funções relacionadas à privacidade

AWS Clean Rooms fornece funções para ajudá-lo a cumprir a conformidade relacionada à privacidade com as seguintes especificações.

- Global Privacy Platform (GPP) — Uma especificação do Interactive Advertising Bureau (IAB) que estabelece uma estrutura global e padronizada para privacidade on-line e uso de dados. Para obter mais informações sobre as especificações técnicas do GPP, consulte a documentação da Global Privacy Platform em GitHub.

- Estrutura de Transparência e Consentimento (TCF) — Um componente essencial do GPP, lançado em 2020, que fornece uma estrutura técnica padronizada para ajudar as empresas a cumprir os regulamentos de privacidade, como o Regulamento Geral de Proteção de Dados (GDPR) da UE. O TCF permite que os clientes concedam ou neguem o consentimento para a coleta e processamento de dados. Para obter mais informações sobre as especificações técnicas do TCF, consulte a documentação do TCF em. GitHub

Tópicos

função consent_gpp_v1_decode

A `consent_gpp_v1_decode` função é usada para decodificar os dados de consentimento da Global Privacy Platform (GPP) v1. Ele usa a string de consentimento codificada como entrada e retorna os dados de consentimento decodificados, que incluem informações sobre as preferências de privacidade e as opções de consentimento do usuário. Essa função é útil ao trabalhar com dados que incluem informações de consentimento do GPP v1, pois permite acessar e analisar os dados de consentimento em um formato estruturado.

Sintaxe

```
consent_gpp_v1_decode(gpp_string)
```

Argumentos

cadeia de caracteres gpp

A string de consentimento codificada do GPP v1.

Retornos

O dicionário retornado inclui os seguintes pares de valores-chave:

- `version`: A versão da especificação GPP usada (atualmente 1).
- `cmpId`: o ID da Plataforma de Gerenciamento de Consentimento (CMP) que codificou a sequência de caracteres de consentimento.

- `cmpVersion`: a versão do CMP que codificou a sequência de caracteres de consentimento.
- `consentScreen`: o ID da tela na interface do usuário do CMP em que o usuário forneceu consentimento.
- `consentLanguage`: O código do idioma das informações de consentimento.
- `vendorListVersion`: a versão da lista de fornecedores usada.
- `publisherCountryCode`: O código do país da editora.
- `purposeConsent`: uma lista de números inteiros representando as finalidades com as quais o usuário consentiu.
- `purposeLegitimateInterest`: Uma lista de propósitos IDs para os quais o interesse legítimo do usuário foi comunicado de forma transparente.
- `specialFeatureOptIns`: uma lista de números inteiros representando os recursos especiais pelos quais o usuário optou.
- `vendorConsent`: uma lista de fornecedores com os IDs quais o usuário consentiu.
- `vendorLegitimateInterest`: uma lista de fornecedores IDs para os quais o interesse legítimo do usuário foi comunicado de forma transparente.

Exemplo

O exemplo a seguir usa um único argumento, que é a string de consentimento codificada. Ele retorna um dicionário contendo os dados de consentimento decodificados, incluindo informações sobre as preferências de privacidade do usuário, as opções de consentimento e outros metadados.

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

A estrutura básica dos dados de consentimento retornados inclui informações sobre a versão da cadeia de consentimento, os detalhes da CMP (Plataforma de Gerenciamento de Consentimento), o consentimento do usuário e as escolhas de interesse legítimo para diferentes finalidades e fornecedores e outros metadados.

```
{
    "version": 1,
    "cmpId": 12,
    "cmpVersion": 34,
    "consentScreen": 5,
    "consentLanguage": "en",
```

```
    "vendorListVersion": 89,
    "publisherCountryCode": "US",
    "purposeConsent": [1],
    "purposeLegitimateInterests": [1],
    "specialFeatureOptins": [1],
    "vendorConsent": [1],
    "vendorLegitimateInterests": [1]}
}
```

função consent_tcf_v2_decode

A `consent_tcf_v2_decode` função é usada para decodificar os dados de consentimento do Transparency and Consent Framework (TCF) v2. Ele usa a string de consentimento codificada como entrada e retorna os dados de consentimento decodificados, que incluem informações sobre as preferências de privacidade e as opções de consentimento do usuário. Essa função é útil ao trabalhar com dados que incluem informações de consentimento do TCF v2, pois permite acessar e analisar os dados de consentimento em um formato estruturado.

Sintaxe

```
consent_tcf_v2_decode(tcf_string)
```

Argumentos

string tcf

A string de consentimento codificada do TCF v2.

Retornos

A `consent_tcf_v2_decode` função retorna um dicionário contendo os dados de consentimento decodificados de uma string de consentimento do Transparency and Consent Framework (TCF) v2.

O dicionário retornado inclui os seguintes pares de valores-chave:

Segmento principal

- `version`: A versão da especificação TCF usada (atualmente 2).
- `created`: a data e a hora em que a sequência de consentimento foi criada.

- `lastUpdated`: a data e a hora em que a sequência de consentimento foi atualizada pela última vez.
- `cmpId`: o ID da Plataforma de Gerenciamento de Consentimento (CMP) que codificou a sequência de caracteres de consentimento.
- `cmpVersion`: a versão do CMP que codificou a sequência de caracteres de consentimento.
- `consentScreen`: o ID da tela na interface do usuário do CMP em que o usuário forneceu consentimento.
- `consentLanguage`: O código do idioma das informações de consentimento.
- `vendorListVersion`: a versão da lista de fornecedores usada.
- `tcfPolicyVersion`: a versão da política do TCF na qual a string de consentimento se baseia.
- `isServiceSpecific`: um valor booleano que indica se o consentimento é específico para um determinado serviço ou se aplica a todos os serviços.
- `useNonStandardStacks`: um valor booleano que indica se pilhas não padrão são usadas.
- `specialFeatureOptIns`: uma lista de números inteiros representando os recursos especiais pelos quais o usuário optou.
- `purposeConsent`: uma lista de números inteiros representando as finalidades com as quais o usuário consentiu.
- `purposesLITransparency`: uma lista de números inteiros representando as finalidades para as quais o usuário deu transparência aos interesses legítimos.
- `purposeOneTreatment`: um valor booleano que indica se o usuário solicitou o “tratamento de propósito único” (ou seja, todos os propósitos são tratados igualmente).
- `publisherCountryCode`: O código do país da editora.
- `vendorConsent`: uma lista de fornecedores com os IDs quais o usuário consentiu.
- `vendorLegitimateInterest`: uma lista de fornecedores IDs para os quais o interesse legítimo do usuário foi comunicado de forma transparente.
- `pubRestrictionEntry`: Uma lista de restrições do editor. Esse campo contém a ID da finalidade, o tipo de restrição e a lista de fornecedores IDs sob essa restrição de finalidade.

Segmento de fornecedores divulgado

- `disclosedVendors`: uma lista de números inteiros representando os fornecedores que foram divulgados ao usuário.

Segmento de propósitos do editor

- `pubPurposesConsent`: uma lista de números inteiros representando as finalidades específicas do editor para as quais o usuário deu consentimento.
- `pubPurposesLITransparency`: uma lista de números inteiros representando as finalidades específicas do editor para as quais o usuário deu transparência aos interesses legítimos.
- `customPurposesConsent`: uma lista de números inteiros representando as finalidades personalizadas para as quais o usuário deu consentimento.
- `customPurposesLITransparency`: uma lista de números inteiros representando as finalidades personalizadas para as quais o usuário deu transparência aos interesses legítimos.

Esses dados de consentimento detalhados podem ser usados para entender e respeitar as preferências de privacidade do usuário ao trabalhar com dados pessoais.

Exemplo

O exemplo a seguir usa um único argumento, que é a string de consentimento codificada. Ele retorna um dicionário contendo os dados de consentimento decodificados, incluindo informações sobre as preferências de privacidade do usuário, as opções de consentimento e outros metadados.

```
from aws_clean_rooms.functions import consent_tcf_v2_decode

consent_string = "CO1234567890abcdef"
consent_data = consent_tcf_v2_decode(consent_string)

print(consent_data)
```

A estrutura básica dos dados de consentimento retornados inclui informações sobre a versão da cadeia de consentimento, os detalhes da CMP (Plataforma de Gerenciamento de Consentimento), o consentimento do usuário e as escolhas de interesse legítimo para diferentes finalidades e fornecedores e outros metadados.

```
    /** core segment **/
    version: 2,
    created: "2023-10-01T12:00:00Z",
    lastUpdated: "2023-10-01T12:00:00Z",
    cmpId: 1234,
    cmpVersion: 5,
```

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

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

    /** publisher purposes  segment **/
    pubPurposesConsent: [1, 2, 3],
    pubPurposesLITransparency: [1, 2, 3],
    customPurposesConsent: [1, 2, 3],
    customPurposesLITransparency: [1, 2, 3],
};
```

Funções de janela

Usando funções da janela, é possível criar consultas analíticas empresariais de forma mais eficiente. Funções de janela operam em uma partição ou “janela” de um conjunto de resultados e retornam um valor para cada linha naquela janela. Por outro lado, funções sem janela executam seus cálculos em relação a cada linha no conjunto de resultados. Diferente de funções de grupo que agregam linhas de resultado, as funções de janela retêm todas as linhas na expressão da tabela.

Os valores retornados são calculados usando valores dos conjuntos de linhas dessa janela. Para cada linha da tabela, a janela define um conjunto de linhas que é usado para computar atributos adicionais. Um janela é definida usando uma especificação de janela (a cláusula OVER) se baseia em três conceitos principais:

- Particionamento da janela, que forma grupos de linhas (cláusula PARTITION)

- Ordenação da janela, que define uma ordem ou sequência de linhas dentro de cada partição (cláusula ORDER BY)
- Quadros da janela, que são definidos em relação a cada linha para restringir ainda mais o conjunto de linhas (especificação de ROWS)

As funções da janela são o último conjunto de operações executadas em uma consulta, exceto pela cláusula ORDER BY final. Todas as junções e todas as cláusulas WHERE, GROUP BY e HAVING são concluídas antes do processamento das funções da janela. Portanto, as funções da janela podem aparecer somente na lista de seleção ou na cláusula ORDER BY. Você pode usar várias funções da janela em uma única consulta com diferentes cláusulas de quadro. Você também pode usar funções da janela em outras expressões escalares, tal como CASE.

Resumo da sintaxe de funções da janela

As funções de janela seguem uma sintaxe padrão, mostrada a seguir.

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

Aqui, function é uma das funções descritas nesta seção.

A expr_list é como indicado a seguir.

```
expression | column_name [, expr_list ]
```

A order_list é como a seguir.

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

O frame_clause é como a seguir.

```
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

função

A função de janela. Para obter detalhes, consulte as descrições individuais da função.

OVER

A cláusula que define a especificação da janela. A cláusula OVER é obrigatória para funções da janela e diferencia funções da janela de outras funções SQL.

PARTITION BY expr_list

(Opcional) A cláusula PARTITION BY subdivide o conjunto de resultados em partições, bem como a cláusula GROUP BY. Se uma cláusula de partição estiver presente, a função será calculada para as linhas em cada partição. Se nenhuma cláusula de partição estiver especificada, uma única partição contém a tabela inteira e a função é computada para esta tabela completa.

As funções de classificação DENSE_RANK, NTILE, RANK e ROW_NUMBER exigem uma comparação global de todas as linhas no conjunto de resultados. Quando uma cláusula PARTITION BY é utilizada, o otimizador de consulta pode executar cada agregação em paralelo, distribuindo a workload em várias fatias de acordo com as partições. Se a cláusula PARTITION BY não estiver presente, a etapa de agregação deverá ser executada em série em uma única fatia, o que poderá ter um impacto negativo considerável na performance, sobretudo para grandes clusters.

AWS Clean Roomsnão oferece suporte a literais de string nas cláusulas PARTITION BY.

ORDER BY order_list

(Opcional) A função da janela é aplicada às linhas dentro de cada partição classificada de acordo com a especificação do pedido em ORDER BY. Esta cláusula ORDER BY é diferente e totalmente não relacionada a uma cláusula ORDER BY na frame_clause. A cláusula ORDER BY pode ser usada sem a cláusula PARTITION BY.

Para as funções de classificação, a cláusula ORDER BY identifica as medidas para os valores de classificação. Para funções de agregação, as linhas particionadas devem ser ordenadas antes que a função agregada seja computada para cada quadro. Para obter mais informações sobre os tipos de função da janela, consulte Funções de janela.

Os identificadores de coluna ou expressões que avaliam os identificadores de coluna são obrigatórios na lista de ordenação. Nem constantes ou expressões constantes podem ser usadas como substitutos para nomes de coluna.

Valores NULL são tratados como seu próprio grupo, ordenados e classificados de acordo com a opção NULLS FIRST ou NULLS LAST. Por padrão, os valores NULL são ordenados e classificados por último na ordem ASC e são ordenados e classificados primeiro na ordem DESC.

AWS Clean Roomsnão oferece suporte a literais de string nas cláusulas ORDER BY.

Se a cláusula ORDER BY for omitida, a ordem das linhas não será determinística.

> **Note**
>
> Em qualquer sistema paraleloAWS Clean Rooms, como quando uma cláusula ORDER BY não produz uma ordenação exclusiva e total dos dados, a ordem das linhas não é determinística. Ou seja, se a expressão ORDER BY produzir valores duplicados (uma ordenação parcial), a ordem de retorno dessas linhas poderá variar de uma sequência AWS Clean Rooms para outra. Por sua vez, funções da janela podem retornar resultados inesperados ou inconsistentes. Para obter mais informações, consulte Ordenação exclusiva de dados para funções da janela.

column_name

Nome de uma coluna a ser particionada por ou ordenada por.

ASC | DESC

Opção que define a ordem de classificação para a expressão, da seguinte forma:

- ASC: ascendente (por exemplo, de valores numéricos menores para maiores e de "A" a "Z" para strings de caracteres). Se nenhuma opção é especificada, os dados são classificados na ordem ascendente por padrão.
- DESC: descendente (de valores numéricos maiores para menores; de "Z" a "A" para strings).

NULLS FIRST | NULLS LAST

Opção que especifica se NULLS devem ser ordenados primeiro, antes de valores não nulos, ou por último, após valores não nulos. Por padrão, NULLs são ordenados e classificados por último na ordem ASC e ordenados e classificados primeiro na ordem DESC.

frame_clause

Para funções agregadas, a cláusula do quadro refina ainda mais o conjunto de linhas na janela de uma função ao usar ORDER BY. Ele permite que você inclua ou exclua conjuntos de linhas no resultado ordenado. A cláusula de quadro consiste na palavra-chave ROWS e nos especificadores associados.

A cláusula frame não se aplica a funções de classificação. Além disso, a cláusula frame não é necessária quando nenhuma cláusula ORDER BY é usada na cláusula OVER para uma função agregada. Se uma cláusula ORDER BY é usada para uma função agregada, uma cláusula de quadro explícita é necessária.

Quando nenhuma cláusula ORDER BY é especificada, o quadro implícito é ilimitado, equivalente a ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

ROWS

Especificando um deslocamento físico da linha atual especificando um deslocamento físico da linha atual.

Essa cláusula especifica as linhas na janela ou particionamento atual ao qual o valor da linha atual dever ser combinado. Ela usa os argumentos que especificam a posição da linha, que pode ser antes ou depois da linha atual. O ponto de referência para todos os quadros de janela é a linha atual. Cada linha se torna a linha atual, por sua vez, à medida que o quadro de janela avança pela partição.

O quadro pode ser um conjunto simples de linhas até e incluindo a linha atual.

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

Ou pode ser um conjunto de linhas entre dois limites.

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING indica que a janela começa na primeira linha da partição; deslocamento PRECEDING indica que a janela começa um número de linhas equivalentes ao valor do deslocamento antes da linha atual. UNBOUNDED PRECEDING é o padrão.

CURRENT ROW indica que a janela começa ou termina na linha atual.

UNBOUNDED FOLLOWING indica que a janela termina na última linha da partição; deslocamento FOLLOWING indica que a janela termina um número de linhas equivalentes ao valor do deslocamento depois da linha atual.

O offset identifica um número físico de linhas antes ou depois da linha atual. Nesse caso, o deslocamento deve ser uma constante que retorna um valor numérico positivo. Por exemplo, 5 FOLLOWING termina o quadro 5 linhas após a linha atual.

Onde BETWEEN não é especificado, o quadro é limitado implicitamente pela linha atual. Por exemplo, `ROWS 5 PRECEDING` é igual a `ROWS BETWEEN 5 PRECEDING AND CURRENT ROW`. Além disso, `ROWS UNBOUNDED FOLLOWING` é igual a `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`.

> **Note**
>
> Você não pode especificar um quadro em que o limite inicial seja maior do que o limite final. Por exemplo, você não pode especificar nenhum dos quadros a seguir.
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

Ordenação exclusiva de dados para funções da janela

Se uma cláusula ORDER BY para uma função da janela não produz uma ordem única e total dos dados, a ordem das linhas não é determinística. Se a expressão ORDER BY produzir valores duplicados (uma ordenação parcial), a ordem de retorno dessas linhas pode variar em várias execuções. Nesse caso, as funções da janela também podem retornar resultados inesperados ou inconsistentes.

Por exemplo, a consulta a seguir retorna resultados diferentes ao longo de várias execuções. Esses resultados diferentes ocorrem porque `order by dateid` não produz uma ordenação exclusiva dos dados para a função da janela 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
...
```

Nesse caso, adicionar uma segunda coluna ORDER BY à função da janela pode resolver o 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
...
```

Funções compatíveis

AWS Clean RoomsO Spark SQL suporta dois tipos de funções de janela: agregação e classificação.

Veja a seguir as funções agregadas compatíveis:

- Função de janela CUME_DIST
- Função de janela DENSE_RANK
- Função FIRST window
- Função de janela FIRST_VALUE
- Função de janela LAG
- Função LAST window
- Função de janela LAST_VALUE
- Função de janela LEAD

Veja a seguir as funções de classificação compatíveis:

- Função de janela DENSE_RANK
- Função de janela PERCENT_RANK
- Função de janela RANK
- Função de janela ROW_NUMBER

Amostra de tabela para exemplos de funções de janela

É possível encontrar exemplos de função de janela específicos com cada descrição de função. Alguns dos exemplos usam uma tabela chamada WINSALES, que contém 11 linhas, conforme mostrado na tabela a seguir.

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
30001	8/2/2003	3	B	10	10
10001	12/24/2003	1	C	10	10
10005	12/24/2003	1	A	30	
40001	1/9/2004	4	A	40	
10006	1/18/2004	1	C	10	

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
20001	2/12/2004	2	B	20	20
40005	2/12/2004	4	A	10	10
20002	2/16/2004	2	C	20	20
30003	4/18/2004	3	B	15	
30004	4/18/2004	3	B	20	
30007	9/7/2004	3	C	30	

Função de janela CUME_DIST

Calcula a distribuição cumulativa de um valor em uma janela ou partição. Assumindo uma ordem ascendente, a distribuição cumulativa é determinada usando esta fórmula:

```
count of rows with values <= x / count of rows in the window or partition
```

onde x é igual ao valor na linha atual da coluna especificada na cláusula ORDER BY. O seguinte conjunto de dados ilustra O uso desta 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
```

O intervalo de valor de retorno é >0 a 1, inclusive.

Sintaxe

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
```

```
)
```

Argumentos

OVER

Uma cláusula que especifica o particionamento da janela. A cláusula OVER não pode conter uma especificação de quadro da janela.

PARTITION BY partition_expression

Opcional. Uma expressão que define o intervalo de registros para cada grupo na cláusula OVER.

ORDER BY order_list

A expressão na qual calcular a distribuição cumulativa. A expressão deve ter um tipo de dados numérico ou ser implicitamente conversível para um. Se ORDER BY for omitida, o valor de retorno será 1 para todas as linhas.

Se ORDER BY não produzir uma ordem única, a ordem das linhas não é determinística. Para obter mais informações, consulte Ordenação exclusiva de dados para funções da janela.

Tipo de retorno

FLOAT8

Exemplos

O seguinte exemplo calcula a distribuição cumulativa da quantidade 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 uma descrição da tabela WINSALES, consulte Amostra de tabela para exemplos de funções de janela.

Função de janela DENSE_RANK

A função de janela DENSE_RANK determina a classificação de um valor em um grupo de valores com base na expressão ORDER BY da cláusula OVER. Se a cláusula opcional PARTITION BY estiver presente, as classificações são redefinidas para cada grupo de linhas. Linhas com valores iguais para os critérios de classificação recebem a mesma classificação. A função DENSE_RANK difere de RANK em um aspecto: se duas ou mais linhas empatarem, não há uma lacuna na sequência de valores classificados. Por exemplo, se duas linhas são classificadas como 1, a classificação seguinte é 2.

Você pode ter funções de classificação com diferentes cláusulas PARTITION BY e ORDER BY na mesma consulta.

Sintaxe

```
DENSE_RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumentos

( )

A função não aceita argumentos, mas os parênteses vazios são necessários.

OVER

As cláusulas de janela para a função DENSE_RANK.

PARTITION BY expr_list

Opcional. Uma ou várias expressões que definem a janela.

ORDER BY order_list

Opcional. A expressão na qual os valores de classificação se baseiam. Se nenhuma PARTITION BY for especificada, ORDER BY usa a tabela completa. Se ORDER BY for omitida, o valor de retorno será 1 para todas as linhas.

Se ORDER BY não produzir uma ordem única, a ordem das linhas não é determinística. Para obter mais informações, consulte Ordenação exclusiva de dados para funções da janela.

Tipo de retorno

INTEGER

Exemplos

O exemplo a seguir ordena a mesa pela quantidade vendida (em ordem decrescente) e atribui uma classificação densa e uma classificação regular a cada linha. Os resultados são classificados após a aplicação dos resultados da função de janela.

```
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)
```

Observe a diferença nas classificações atribuídas ao mesmo conjunto de linhas quando as funções DENSE_RANK e RANK são usadas lado a lado na mesma consulta. Para uma descrição da tabela WINSALES, consulte Amostra de tabela para exemplos de funções de janela.

O exemplo a seguir particiona a tabela por SELLERID e ordena cada partição pela quantidade (em ordem decrescente) e atribui uma classificação densa a cada linha. Os resultados são classificados após a aplicação dos resultados da função de janela.

```
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 uma descrição da tabela WINSALES, consulte Amostra de tabela para exemplos de funções de janela.

Função FIRST window

Dado um conjunto ordenado de linhas, FIRST retorna o valor da expressão especificada em relação à primeira linha na moldura da janela.

Para obter informações sobre como selecionar a última linha no quadro, consulte Função LAST window.

Sintaxe

```
FIRST( expression )[ IGNORE NULLS | RESPECT NULLS ]
```

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

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera.

IGNORE NULLS

Quando essa opção é usada com FIRST, a função retorna o primeiro valor no quadro que não é NULL (ou NULL se todos os valores forem NULL).

RESPECT NULLS

Indica que AWS Clean Rooms deve incluir valores nulos na determinação de qual linha usar. RESPECT NULLS é compatível por padrão se você não especificar IGNORE NULLS.

OVER

Introduz as cláusulas de janela para a função.

PARTITION BY expr_list

Define a janela para a função em termos de uma ou mais expressões.

ORDER BY order_list

Classifica as linhas dentro de cada partição. Se nenhuma cláusula PARTITION BY for especificada, ORDER BY classifica a tabela inteira. Se você especificar uma cláusula ORDER BY, você também deve especificar uma frame_clause.

Os resultados da função FIRST dependem da ordem dos dados. Os resultados são não determinísticos nos seguintes casos:

- Quando uma cláusula ORDER BY é especificada e uma partição contém dois valores diferentes para uma expressão
- Quando uma expressão avalia para valores diferentes que correspondem ao mesmo valor na lista ORDER BY.

frame_clause

Se uma cláusula ORDER BY é usada para uma função agregada, uma cláusula de quadro explícita é necessária. A cláusula de quadro refina o conjunto de linhas na janela de uma função,

incluindo ou excluindo conjuntos de linhas no resultado ordenado. A cláusula de quadro consiste na palavra-chave ROWS e nos especificadores associados. Consulte Resumo da sintaxe de funções da janela.

Tipo de retorno

Essas funções oferecem suporte a expressões que usam tipos de AWS Clean Rooms dados primitivos. O tipo de retorno é igual ao tipo de dados da expressão.

Exemplos

O seguinte exemplo retorna a capacidade de acomodação para cada local de evento da tabela VENUE com os resultados ordenados por capacidade (alta a baixa). A função FIRST é usada para selecionar o nome do local que corresponde à primeira fila no quadro: nesse caso, a fila com o maior número de assentos. Os resultados são particionados por estado, portanto quando o valor VENUESTATE muda, um novo primeiro valor é selecionado. O quadro da janela não é vinculado, portanto o mesmo primeiro valor é selecionado para cada linha em cada partição.

Para a Califórnia, `Qualcomm Stadium` tem mais alto número de assentos (`70561`), portanto esse nome é o primeiro valor para todas as linhas da partição `CA`.

```
select venuestate, venueseats, venuename,
first(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |          venuename            |         first
-----------+------------+-------------------------------
+-----------------------------
CA         |      70561 | Qualcomm Stadium              | Qualcomm Stadium
CA         |      69843 | Monster Park                  | Qualcomm Stadium
CA         |      63026 | McAfee Coliseum               | Qualcomm Stadium
CA         |      56000 | Dodger Stadium                | Qualcomm Stadium
CA         |      45050 | Angel Stadium of Anaheim      | Qualcomm Stadium
CA         |      42445 | PETCO Park                    | Qualcomm Stadium
CA         |      41503 | AT&T Park                     | Qualcomm Stadium
CA         |      22000 | Shoreline Amphitheatre        | Qualcomm Stadium
CO         |      76125 | INVESCO Field                 | INVESCO Field
CO         |      50445 | Coors Field                   | INVESCO Field
```

```
DC         |      41888 | Nationals Park                | Nationals Park
FL         |      74916 | Dolphin Stadium               | Dolphin Stadium
FL         |      73800 | Jacksonville Municipal Stadium | Dolphin Stadium
FL         |      65647 | Raymond James Stadium         | Dolphin Stadium
FL         |      36048 | Tropicana Field               | Dolphin Stadium
...
```

Função de janela FIRST_VALUE

Considerando um conjunto de linhas ordenado, FIRST_VALUE retorna o valor da expressão especificada em relação à primeira linha no quadro de janela.

Para obter informações sobre como selecionar a última linha no quadro, consulte Função de janela LAST_VALUE.

Sintaxe

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera.

IGNORE NULLS

Quando essa opção é usada com FIRST_VALUE, a função retorna o primeiro valor no quadro que não seja NULL (ou NULL se todos os valores forem NULL).

RESPECT NULLS

Indica que AWS Clean Rooms deve incluir valores nulos na determinação de qual linha usar. RESPECT NULLS é compatível por padrão se você não especificar IGNORE NULLS.

OVER

Introduz as cláusulas de janela para a função.

PARTITION BY expr_list

Define a janela para a função em termos de uma ou mais expressões.

ORDER BY order_list

Classifica as linhas dentro de cada partição. Se nenhuma cláusula PARTITION BY for especificada, ORDER BY classifica a tabela inteira. Se você especificar uma cláusula ORDER BY, você também deve especificar uma frame_clause.

Os resultados da função FIRST_VALUE dependem da ordem dos dados. Os resultados são não determinísticos nos seguintes casos:

- Quando uma cláusula ORDER BY é especificada e uma partição contém dois valores diferentes para uma expressão
- Quando uma expressão avalia para valores diferentes que correspondem ao mesmo valor na lista ORDER BY.

frame_clause

Se uma cláusula ORDER BY é usada para uma função agregada, uma cláusula de quadro explícita é necessária. A cláusula de quadro refina o conjunto de linhas na janela de uma função, incluindo ou excluindo conjuntos de linhas no resultado ordenado. A cláusula de quadro consiste na palavra-chave ROWS e nos especificadores associados. Consulte Resumo da sintaxe de funções da janela.

Tipo de retorno

Essas funções oferecem suporte a expressões que usam tipos de AWS Clean Rooms dados primitivos. O tipo de retorno é igual ao tipo de dados da expressão.

Exemplos

O seguinte exemplo retorna a capacidade de acomodação para cada local de evento da tabela VENUE com os resultados ordenados por capacidade (alta a baixa). A função FIRST_VALUE é usada para selecionar o local de evento que corresponde à primeira linha no quadro: nesse caso, a linha com o mais alto número de assentos. Os resultados são particionados por estado, portanto quando o valor VENUESTATE muda, um novo primeiro valor é selecionado. O quadro da janela não é vinculado, portanto o mesmo primeiro valor é selecionado para cada linha em cada partição.

Para a Califórnia, `Qualcomm Stadium` tem mais alto número de assentos (`70561`), portanto esse nome é o primeiro valor para todas as linhas da partição `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
...
```

Função de janela LAG

A função de janela LAG retorna os valores para uma linha em determinado deslocamento acima (antes) da linha atual na partição.

Sintaxe

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Argumentos

value_expr

A coluna ou expressão de destino na qual a função opera.

deslocamento

Um parâmetro opcional que especifica o número de linhas antes da linha atual para as quais retornar valores. Este deslocamento pode ser um inteiro constante ou uma expressão que avalia para um inteiro. Se você não especificar um deslocamento, AWS Clean Rooms usa 1 como valor padrão. Um deslocamento de `0` indica a linha atual.

IGNORE NULLS

Uma especificação opcional que indica que os valores nulos AWS Clean Rooms devem ser ignorados na determinação de qual linha usar. Valores nulos são incluídos se IGNORE NULLS não for listada.

> **Note**
>
> Você pode usar uma expressão NVL ou COALESCE para substituir os valores nulos por outro valor.

RESPECT NULLS

Indica que AWS Clean Rooms deve incluir valores nulos na determinação de qual linha usar. RESPECT NULLS é compatível por padrão se você não especificar IGNORE NULLS.

OVER

Especifica o particionamento e ordem da janela. A cláusula OVER não pode conter uma especificação de quadro da janela.

PARTITION BY window_partition

Um argumento ideal que define o intervalo de registros para cada grupo na cláusula OVER.

ORDER BY window_ordering

Classifica as linhas dentro de cada partição.

A função de janela LAG suporta expressões que usam qualquer um dos tipos de AWS Clean Rooms dados. O tipo de retorno é igual ao tipo de value_expr.

Exemplos

O seguinte exemplo mostra a quantidade de ingressos vendidos para o comprador com ID de comprador 3 e a hora que o comprador 3 adquiriu os ingressos. Para comparar cada venda com

venda anterior para o comprador 3, a consulta retorna a quantidade anterior vendida em cada venda. Já que não há compras antes de 1/16/2008, o primeiro valor de quantidade vendida anteriormente é 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)
```

Função LAST window

Dado um conjunto ordenado de linhas, a função LAST retorna o valor da expressão em relação à última linha no quadro.

Para obter informações sobre como selecionar a primeira linha no quadro, consulte Função FIRST window.

Sintaxe

```
LAST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera.

IGNORE NULLS

A função retorna o último valor no quadro que não seja NULL (ou NULL se todos os valores forem NULL).

RESPECT NULLS

Indica que AWS Clean Rooms deve incluir valores nulos na determinação de qual linha usar. RESPECT NULLS é compatível por padrão se você não especificar IGNORE NULLS.

OVER

Introduz as cláusulas de janela para a função.

PARTITION BY expr_list

Define a janela para a função em termos de uma ou mais expressões.

ORDER BY order_list

Classifica as linhas dentro de cada partição. Se nenhuma cláusula PARTITION BY for especificada, ORDER BY classifica a tabela inteira. Se você especificar uma cláusula ORDER BY, você também deve especificar uma frame_clause.

Os resultados dependem da ordem dos dados. Os resultados são não determinísticos nos seguintes casos:

- Quando uma cláusula ORDER BY é especificada e uma partição contém dois valores diferentes para uma expressão
- Quando uma expressão avalia para valores diferentes que correspondem ao mesmo valor na lista ORDER BY.

frame_clause

Se uma cláusula ORDER BY é usada para uma função agregada, uma cláusula de quadro explícita é necessária. A cláusula de quadro refina o conjunto de linhas na janela de uma função, incluindo ou excluindo conjuntos de linhas no resultado ordenado. A cláusula de quadro consiste na palavra-chave ROWS e nos especificadores associados. Consulte Resumo da sintaxe de funções da janela.

Tipo de retorno

Essas funções oferecem suporte a expressões que usam tipos de AWS Clean Rooms dados primitivos. O tipo de retorno é igual ao tipo de dados da expressão.

Exemplos

O seguinte exemplo retorna a capacidade de acomodação para cada local de evento da tabela VENUE com os resultados ordenados por capacidade (alta a baixa). A função LAST é usada para selecionar o nome do local que corresponde à última linha no quadro: nesse caso, a linha com o menor número de assentos. Os resultados são particionados por estado, portanto quando o valor VENUESTATE muda, um novo último valor é selecionado. O quadro da janela não é vinculado, portanto o mesmo último valor é selecionado para cada linha em cada partição.

Para a Califórnia, `Shoreline Amphitheatre` será retornado para cada linha na partição, pois tem o menor número de assentos (`22000`).

```
select venuestate, venueseats, venuename,
last(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |           venuename            |          last
-----------+------------+--------------------------------
+------------------------------
CA         |      70561 | Qualcomm Stadium               | Shoreline Amphitheatre
CA         |      69843 | Monster Park                   | Shoreline Amphitheatre
CA         |      63026 | McAfee Coliseum                | Shoreline Amphitheatre
CA         |      56000 | Dodger Stadium                 | Shoreline Amphitheatre
CA         |      45050 | Angel Stadium of Anaheim       | Shoreline Amphitheatre
CA         |      42445 | PETCO Park                     | Shoreline Amphitheatre
CA         |      41503 | AT&T Park                      | Shoreline Amphitheatre
CA         |      22000 | Shoreline Amphitheatre         | Shoreline Amphitheatre
CO         |      76125 | INVESCO Field                  | Coors Field
CO         |      50445 | Coors Field                    | Coors Field
DC         |      41888 | Nationals Park                 | Nationals Park
FL         |      74916 | Dolphin Stadium                | Tropicana Field
FL         |      73800 | Jacksonville Municipal Stadium | Tropicana Field
FL         |      65647 | Raymond James Stadium          | Tropicana Field
```

```
FL         |      36048 | Tropicana Field                | Tropicana Field
...
```

Função de janela LAST_VALUE

Considerando um conjunto de linhas ordenadas, a função LAST_VALUE retorna o valor da expressão em relação à última linha no quadro.

Para obter informações sobre como selecionar a primeira linha no quadro, consulte Função de janela FIRST_VALUE.

Sintaxe

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumentos

expressão

A coluna ou expressão de destino na qual a função opera.

IGNORE NULLS

A função retorna o último valor no quadro que não seja NULL (ou NULL se todos os valores forem NULL).

RESPECT NULLS

Indica que AWS Clean Rooms deve incluir valores nulos na determinação de qual linha usar. RESPECT NULLS é compatível por padrão se você não especificar IGNORE NULLS.

OVER

Introduz as cláusulas de janela para a função.

PARTITION BY expr_list

Define a janela para a função em termos de uma ou mais expressões.

ORDER BY order_list

Classifica as linhas dentro de cada partição. Se nenhuma cláusula PARTITION BY for especificada, ORDER BY classifica a tabela inteira. Se você especificar uma cláusula ORDER BY, você também deve especificar uma frame_clause.

Os resultados dependem da ordem dos dados. Os resultados são não determinísticos nos seguintes casos:

- Quando uma cláusula ORDER BY é especificada e uma partição contém dois valores diferentes para uma expressão
- Quando uma expressão avalia para valores diferentes que correspondem ao mesmo valor na lista ORDER BY.

frame_clause

Se uma cláusula ORDER BY é usada para uma função agregada, uma cláusula de quadro explícita é necessária. A cláusula de quadro refina o conjunto de linhas na janela de uma função, incluindo ou excluindo conjuntos de linhas no resultado ordenado. A cláusula de quadro consiste na palavra-chave ROWS e nos especificadores associados. Consulte Resumo da sintaxe de funções da janela.

Tipo de retorno

Essas funções oferecem suporte a expressões que usam tipos de AWS Clean Rooms dados primitivos. O tipo de retorno é igual ao tipo de dados da expressão.

Exemplos

O seguinte exemplo retorna a capacidade de acomodação para cada local de evento da tabela VENUE com os resultados ordenados por capacidade (alta a baixa). A função LAST_VALUE é usada para selecionar o local de evento que corresponde à última linha no quadro: nesse caso, a linha com o mais baixo número de assentos. Os resultados são particionados por estado, portanto quando o valor VENUESTATE muda, um novo último valor é selecionado. O quadro da janela não é vinculado, portanto o mesmo último valor é selecionado para cada linha em cada partição.

Para a Califórnia, `Shoreline Amphitheatre` será retornado para cada linha na partição, pois tem o menor número de assentos (`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
...
```

Função de janela LEAD

A função de janela LEAD retorna os valores para uma linha em determinado deslocamento abaixo (depois) da linha atual na partição.

Sintaxe

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Argumentos

value_expr

A coluna ou expressão de destino na qual a função opera.

deslocamento

Um parâmetro opcional que especifica o número de linhas abaixo da linha atual para as quais retornar valores. Este deslocamento pode ser um inteiro constante ou uma expressão que avalia para um inteiro. Se você não especificar um deslocamento, AWS Clean Rooms usa 1 como valor padrão. Um deslocamento de `0` indica a linha atual.

IGNORE NULLS

Uma especificação opcional que indica que os valores nulos AWS Clean Rooms devem ser ignorados na determinação de qual linha usar. Valores nulos são incluídos se IGNORE NULLS não for listada.

> **Note**
>
> Você pode usar uma expressão NVL ou COALESCE para substituir os valores nulos por outro valor.

RESPECT NULLS

Indica que AWS Clean Rooms deve incluir valores nulos na determinação de qual linha usar. RESPECT NULLS é compatível por padrão se você não especificar IGNORE NULLS.

OVER

Especifica o particionamento e ordem da janela. A cláusula OVER não pode conter uma especificação de quadro da janela.

PARTITION BY window_partition

Um argumento ideal que define o intervalo de registros para cada grupo na cláusula OVER.

ORDER BY window_ordering

Classifica as linhas dentro de cada partição.

A função de janela LEAD oferece suporte a expressões que usam qualquer um dos tipos de AWS Clean Rooms dados. O tipo de retorno é igual ao tipo de value_expr.

Exemplos

O seguinte exemplo fornece a comissão para eventos na tabela SALES para os quais ingressos foram vendidos em 1º de janeiro de 2008 e 2 de janeiro de 2008, assim como a comissão paga por vendas de ingressos para a venda subsequente.

```
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)
```

Função de janela PERCENT_RANK

Calcula a classificação percentual de dada linha. A classificação percentual é determinada usando esta fórmula:

```
(x - 1) / (the number of rows in the window or partition - 1)
```

onde x é a classificação da linha atual. O seguinte conjunto de dados ilustra O uso desta 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
```

O intervalo de valor de retorno é 0 a 1, inclusive. A primeira linha em qualquer conjunto tem um PERCENT_RANK de 0.

Sintaxe

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Argumentos

( )

A função não aceita argumentos, mas os parênteses vazios são necessários.

OVER

Uma cláusula que especifica o particionamento da janela. A cláusula OVER não pode conter uma especificação de quadro da janela.

PARTITION BY partition_expression

Opcional. Uma expressão que define o intervalo de registros para cada grupo na cláusula OVER.

ORDER BY order_list

Opcional. A expressão na qual calcular a classificação percentual. A expressão deve ter um tipo de dados numérico ou ser implicitamente conversível para um. Se ORDER BY for omitida, o valor de retorno será 0 para todas as linhas.

Se ORDER BY não produzir uma ordenação exclusiva, a ordem das linhas será não determinística. Para obter mais informações, consulte Ordenação exclusiva de dados para funções da janela.

Tipo de retorno

FLOAT8

Exemplos

O seguinte exemplo calcula a classificação percentual das quantidades de vendas 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 uma descrição da tabela WINSALES, consulte Amostra de tabela para exemplos de funções de janela.

Função de janela RANK

A função de janela RANK determina a classificação de um valor em um grupo de valores com base na expressão ORDER BY da cláusula OVER. Se a cláusula opcional PARTITION BY estiver presente, as classificações são redefinidas para cada grupo de linhas. As linhas com valores iguais para os critérios de classificação recebem a mesma classificação. AWS Clean Roomsadiciona o número de linhas empatadas à classificação empatada para calcular a próxima classificação e, portanto, as classificações podem não ser números consecutivos. Por exemplo, se duas linhas são classificadas como 1, a classificação seguinte é 3.

RANK difere de Função de janela DENSE_RANK em um aspecto: para DENSE_RANK, se duas ou mais linhas empatarem, não há uma lacuna na sequência de valores classificados. Por exemplo, se duas linhas são classificadas como 1, a classificação seguinte é 2.

Você pode ter funções de classificação com diferentes cláusulas PARTITION BY e ORDER BY na mesma consulta.

Sintaxe

```
RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumentos

( )

A função não aceita argumentos, mas os parênteses vazios são necessários.

OVER

As cláusulas de janela para a função RANK.

PARTITION BY expr_list

Opcional. Uma ou várias expressões que definem a janela.

ORDER BY order_list

Opcional. Define as colunas nas quais os valores de classificação se baseiam. Se nenhuma PARTITION BY for especificada, ORDER BY usa a tabela completa. Se ORDER BY for omitida, o valor de retorno será 1 para todas as linhas.

Se ORDER BY não produzir uma ordenação exclusiva, a ordem das linhas será não determinística. Para obter mais informações, consulte Ordenação exclusiva de dados para funções da janela.

Tipo de retorno

INTEGER

Exemplos

O exemplo a seguir ordena a tabela pela quantidade vendida (padrão crescente) e atribui uma classificação a cada linha. O valor de classificação de 1 é o valor de classificação mais alto. Os resultados são classificados após a aplicação dos resultados da função de janela:

```
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)
```

Observe que a cláusula ORDER BY externa neste exemplo inclui as colunas 2 e 1 para garantir que AWS Clean Rooms retorne resultados classificados de forma consistente sempre que essa consulta for executada. Por exemplo, linhas com vendas IDs 10001 e 10006 têm valores idênticos de QTY e RNK. Ordenar o conjunto de resultados final pela coluna 1 garante que a linha 10.001 sempre caia antes de 10.006. Para uma descrição da tabela WINSALES, consulte Amostra de tabela para exemplos de funções de janela.

No exemplo a seguir, a ordenação é revertida para a função da janela (`order by qty desc`). Agora, o valor de classificação mais alto se aplica ao valor de QTY mais 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 uma descrição da tabela WINSALES, consulte Amostra de tabela para exemplos de funções de janela.

O exemplo a seguir particiona a tabela por SELLERID e ordena cada partição pela quantidade (em ordem decrescente) e atribui uma classificação a cada linha. Os resultados são classificados após a aplicação dos resultados da função de janela.

```
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)
```

Função de janela ROW_NUMBER

Determina o número ordinal da linha atual em um grupo de linhas, começando com 1, com base na expressão ORDER BY da cláusula OVER. Se a cláusula opcional PARTITION BY estiver presente, os números ordinais são redefinidos para cada grupo de linhas. As linhas com valores iguais para as expressões ORDER BY recebem os diferentes números de linha de forma não determinística.

Sintaxe

```
ROW_NUMBER () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumentos

( )

A função não aceita argumentos, mas os parênteses vazios são necessários.

OVER

As cláusulas de janela para a função ROW_NUMBER.

PARTITION BY expr_list

Opcional. Uma ou mais expressões que definem a função ROW_NUMBER.

ORDER BY order_list

Opcional. A expressão que define as colunas nas quais os números de linha se baseiam. Se nenhuma PARTITION BY for especificada, ORDER BY usa a tabela completa.

Se ORDER BY não produzir uma ordenação exclusiva ou for omitido, a ordem das linhas será não determinística. Para obter mais informações, consulte Ordenação exclusiva de dados para funções da janela.

Tipo de retorno

BIGINT

Exemplos

O seguinte exemplo particiona a tabela por SELLERID e ordena cada partição por QTY (na ordem ascendente) e, então, atribui um número de linha para cada linha. Os resultados são classificados após a aplicação dos resultados da função de janela.

```
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 uma descrição da tabela WINSALES, consulte Amostra de tabela para exemplos de funções de janela.

AWS Clean Rooms Condições do Spark SQL

Condições são declarações de uma ou mais expressões e operadores lógicos avaliados como verdadeiros, falsos ou desconhecidos. As condições também são ocasionalmente chamadas de predicados.

Sintaxe

```
comparison_condition
| logical_condition
| range_condition
```

```
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

> **Note**
>
> Todas as comparações de strings e correspondências de padrão LIKE diferenciam entre letras maiúsculas e minúsculas. Por exemplo, “A” e “a” não são correspondentes. No entanto, você pode fazer uma correspondência de padrão que não diferencia maiúsculas e minúsculas usando o predicado ILIKE.

As seguintes condições SQL são compatíveis com o AWS Clean Rooms Spark SQL.

Tópicos

Operadores de comparação

A comparação de condições indica as relações lógicas entre dois valores. Todas as condições de comparação são operadores binários com um tipo de retorno booleano.

AWS Clean Rooms O Spark SQL é compatível com os operadores de comparação descritos na tabela a seguir.

Operador	Sintaxe	Descrição
!	`!expression`	O NOT operador lógico. Usado para negar uma expressão

Operador	Sintaxe	Descrição
		booleana, o que significa que ela retorna o oposto do valor da expressão. O! O operador também pode ser combinado com outros operadores lógicos, como AND e OR, para criar expressões booleanas mais complexas.
<	`a < b`	O menos do que um operador de comparação. Usado para comparar dois valores e determinar se o valor à esquerda é menor que o valor à direita.
>	`a > b`	O operador maior que o de comparação. Usado para comparar dois valores e determinar se o valor à esquerda é maior que o valor à direita.
<=	`a <= b`	O operador de comparaçã o menor ou igual. Usado para comparar dois valores e retorna `true` se o valor à esquerda for menor ou igual ao valor à direita e de `false` outra forma.

Operador	Sintaxe	Descrição
>=	`a >= b`	O operador de comparaçã o maior ou igual. Usado para comparar dois valores e determinar se o valor à esquerda é maior ou igual ao valor à direita.
=	`a = b`	O operador de comparação de igualdade, que compara dois valores e retorna `true` se eles são iguais ou `false` não.
<> ou **!=**	`a <> b` ou `a != b`	O operador de comparação `false` diferente, que compara dois valores e retorna `true` se eles não forem iguais, etc.

Operador	Sintaxe	Descrição
==	`a == b`	O operador de comparação de igualdade padrão, que compara dois valores e retorna `true` se eles são iguais ou `false` não. **Note** O operador == diferencia maiúsculas de minúsculas ao comparar valores de string. Se precisar realizar uma comparação sem distinção entre maiúsculas e minúsculas, você pode usar funções como UPPER () ou LOWER () para converter os valores no mesmo caso antes da comparação.

Exemplos

Veja alguns exemplos simples de condições de comparação:

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

A consulta a seguir retorna os valores de identificação de todos os esquilos que não estão se alimentando no momento.

```
SELECT id FROM squirrels
WHERE !is_foraging
```

A consulta a seguir retorna locais com mais de 10.000 lugares da tabela 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 exemplo seleciona os usuários (USERID) da tabela USERS que gostam de rock:

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
13
16
(5 rows)
```

Este exemplo seleciona os usuários (USERID) da tabela USERS onde não se sabe se eles gostam de 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
```

Exemplos com uma coluna TIME

A tabela de exemplo a seguir TIME_TEST tem uma coluna TIME_VAL (tipo TIME) com três valores inseridos.

```
select time_val from time_test;

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

O exemplo a seguir extrai as 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
```

O exemplo a seguir compara dois literais de tempo.

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
 t
```

Exemplos com uma coluna TIMETZ

A tabela de exemplo a seguir TIMETZ_TEST tem uma coluna TIMETZ_VAL (tipo TIMETZ) com três valores inseridos.

```
select timetz_val from timetz_test;

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

O exemplo a seguir seleciona apenas os valores TIMETZ menores que `3:00:00 UTC`. A comparação é feita depois de converter o valor para UTC.

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

O seguinte exemplo compara dois literais TIMETZ. O fuso horário é ignorado para a comparação.

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

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

Condições lógicas

As condições lógicas combinam o resultado de duas condições para produzir um único resultado. Todas as condições lógicas são operadores binários com um tipo de retorno booleano.

Sintaxe

```
expression
{ AND | OR }
expression
NOT expression
```

As condições lógicas usam uma lógica booleana de três valores onde o valor nulo representa uma relação desconhecida. A tabela a seguir descreve os resultados para condições lógicas, onde E1 e E2 representam expressões:

E1	E2	E1 E E2	E1 OU E2	NÃO E2
VERDADEIRO	VERDADEIRO	VERDADEIRO	VERDADEIRO	FALSE
VERDADEIRO	FALSE	FALSE	VERDADEIRO	VERDADEIRO
VERDADEIRO	UNKNOWN	UNKNOWN	VERDADEIRO	UNKNOWN
FALSE	VERDADEIRO	FALSE	VERDADEIRO	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	VERDADEIRO	UNKNOWN	VERDADEIRO	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

O operador NOT é avaliado antes de AND e o operador AND é avaliado antes do operador OR. O uso de qualquer parênteses pode cancelar esta ordem de avaliação padrão.

Exemplos

O seguinte exemplo retorna o USERID e USERNAME da tabela USERS onde o usuário gosta de Las Vegas e de esportes:

```
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)
```

O próximo exemplo retorna o USERID e USERNAME da tabela USERS onde o usuário gosta de Las Vegas, de esportes ou de ambos. Essa consulta retorna todas as saídas do exemplo anterior, mais os usuários que gostam somente de Las Vegas ou de esportes.

```
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)
```

A seguinte consulta usa parênteses em torno da condição OR para encontrar locais em Nova Iorque ou na Califórnia onde Macbeth foi apresentada:

```
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
...
```

A remoção dos parênteses neste exemplo altera a lógica e os resultados da consulta.

O seguinte exemplo usa o 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
...
```

O seguinte exemplo usa uma condição NOT seguida de uma condição 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)
```

Condições de correspondência de padrões

Um operador de correspondência de padrões pesquisa em uma string por um padrão especificado na expressão condicional e retorna verdadeiro ou falso, dependendo de encontrar uma correspondência. AWS Clean Rooms O Spark SQL usa os seguintes métodos para correspondência de padrões:

- Expressões LIKE

  O operador LIKE compara uma expressão de string, tal como um nome de coluna, a um padrão que usa os caracteres curinga % (por cento) e _ (sublinhado). A correspondência de padrão LIKE sempre abrange toda a string. O LIKE realiza uma correspondência com distinção entre maiúsculas e minúsculas.

Tópicos

- LIKE
- RLIKE

LIKE

O operador LIKE compara uma expressão de string, tal como um nome de coluna, a um padrão que usa os caracteres curinga % (por cento) e _ (sublinhado). A correspondência de padrão LIKE sempre abrange toda a string. Para corresponder uma sequência em qualquer lugar de uma string, o padrão deve começar e terminar com um sinal de por cento.

LIKE diferencia maiúsculas de minúsculas.

Sintaxe

```
expression [ NOT ] LIKE | pattern [ ESCAPE 'escape_char' ]
```

Argumentos

expressão

Uma expressão de caractere UTF-8 válida, tal como um nome de coluna.

LIKE

LIKE executa uma correspondência de padrão com diferenciação entre maiúsculas e minúsculas. Para realizar uma correspondência de padrões sem distinção entre maiúsculas e minúsculas para caracteres de vários bytes, use a função LOWER em expressão e padrão com uma condição LIKE.

Em contraste com os predicados de comparação, como = e <>, os predicados LIKE não ignoram implicitamente os espaços à direita. Para ignorar espaços à direita, use RTRIM ou converta explicitamente uma coluna CHAR em VARCHAR.

O ~~ operador é equivalente a LIKE. Além disso, o !~~ operador é equivalente a NOT LIKE.

pattern

Uma expressão de caractere UTF-8 válida com o padrão a ser correspondido.

escape_char

Uma expressão de caractere que irá escapar caracteres de metacaracteres no padrão. O padrão é duas barras invertidas ('\\').

Se o padrão não contém metacaracteres, o padrão representa somente a própria string; nesse caso, LIKE age da mesma forma que o operador de igualdade.

Qualquer uma das expressões de caracteres pode ser de tipos de dados CHAR ou VARCHAR. Se eles forem diferentes, o AWS Clean Rooms converterá o padrão no tipo de dados da expressão.

LIKE é compatível com os seguintes metacaracteres de correspondência de padrão:

Operador	Descrição
%	Corresponde a qualquer sequência de zero ou mais caracteres.

Operador	Descrição
_	Corresponde a qualquer caractere único.

Exemplos

A seguinte tabela mostra exemplos de correspondência de padrão usando LIKE:

Expressão	Retornos
`'abc' LIKE 'abc'`	Verdadeiro
`'abc' LIKE 'a%'`	Verdadeiro
`'abc' LIKE '_B_'`	Falso
`'abc' LIKE 'c%'`	Falso

O seguinte exemplo localiza todas as cidades cujos nomes começam com “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
...
```

O seguinte exemplo localiza usuários cujos sobrenomes contêm “ten”:

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
```

```
-------------
Christensen
Wooten
...
```

O exemplo a seguir encontra cidades cujos terceiro e quarto caracteres são “ea”. . :

```
select distinct city from users where city like '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

O seguinte exemplo usa a string de escape padrão (\\) para pesquisar strings que incluem “start_” (o texto `start` seguido por um sublinhado _):

```
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)
```

O seguinte exemplo especifica “^” como caractere de escape e o utiliza para pesquisar strings que incluem “start_” (o texto `start` seguido por um sublinhado _):

```
select tablename, "column" from my_table_def

where "column" like '%start^_%' escape '^'
```

```
limit 5;

     tablename     |    column
-------------------+---------------
 my_s3client       | start_time
 my_tr_conflict    | xact_start_ts
 my_undone         | undo_start_ts
 my_unload_log     | start_time
 my_vacuum_detail | start_row
(5 rows)
```

RLIKE

O operador RLIKE permite verificar se uma string corresponde a um padrão de expressão regular especificado.

Retorna `true` se str corresponder `regexp` `false` ou não.

Sintaxe

```
rlike(str, regexp)
```

Argumentos

str

Uma expressão de string

regexp

Uma expressão em cadeia de caracteres. A string regex deve ser uma expressão regular Java.

Literais de string (incluindo padrões de regex) não escapam em nosso analisador SQL. Por exemplo, para corresponder a “\ abc”, uma expressão regular para regexp pode ser “^\ abc$”.

Exemplos

O exemplo a seguir define o valor do parâmetro de `spark.sql.parser.escapedStringLiterals` configuração como`true`. Esse parâmetro é específico para o mecanismo Spark SQL. O `spark.sql.parser.escapedStringLiterals` parâmetro no Spark SQL controla como o analisador SQL manipula literais de string escapados. Quando definido como`true`, o analisador interpretará caracteres de barra invertida (\) em literais

de string como caracteres de escape, permitindo que você inclua caracteres especiais como novas linhas, tabulações e aspas nos valores da string.

```
SET spark.sql.parser.escapedStringLiterals=true;
spark.sql.parser.escapedStringLiterals  true
```

Por exemplo, com`spark.sql.parser.escapedStringLiterals=true`, você pode usar a seguinte string literal em sua consulta SQL:

```
SELECT 'Hello, world!\n'
```

O caractere de nova linha \n seria interpretado como um caractere literal de nova linha na saída.

O exemplo a seguir executa uma correspondência de padrão de expressão regular. O primeiro argumento é passado para o operador RLIKE. É uma string que representa um caminho de arquivo, em que o nome de usuário real é substituído pelo padrão '****'. O segundo argumento é o padrão de expressão regular usado para a correspondência. A saída (`true`) indica que a primeira string (`'%SystemDrive%\Users\****'`) corresponde ao padrão de expressão regular (`'%SystemDrive%\\Users.*'`).

```
SELECT rlike('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

Condição de intervalo BETWEEN

Uma condição `BETWEEN` testa expressões para a inclusão em um intervalo de valores, usando as palavras chave `BETWEEN` e `AND`.

Sintaxe

```
expression [ NOT ] BETWEEN expression AND expression
```

As expressões podem ser numéricas, caracteres ou tipos de dados de data e hora, mas elas devem ser compatíveis. O intervalo é inclusivo.

Exemplos

O primeiro exemplo conta quantas transações registraram vendas de 2, 3 ou 4 ingressos:

```
select count(*) from sales
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

A condição de intervalo inclui os valores de começo e de término.

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

A primeira expressão em uma condição de intervalo deve ser o menor valor e a segunda expressão o maior valor. O seguinte exemplo retornará sempre zero linhas devido aos valores das expressões:

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

Contudo, a aplicação do modificador NOT inverterá a lógica e produzirá uma contagem de todas as linhas:

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
172456
(1 row)
```

A seguinte consulta retorna uma lista de locais com 20.000 a 50.000 assentos:

```
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)
```

O exemplo a seguir demonstra o uso de BETWEEN para valores de data:

```
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
```

Observe que, embora o intervalo de BETWEEN seja inclusivo, as datas têm como padrão um valor de hora de 00:00:00. A única linha válida de 3 de janeiro para a consulta de amostra seria uma linha com hora de venda de `1/3/2008 00:00:00`.

Condição null

A ferramenta NULL testes de condição para valores nulos, quando um valor está ausente ou é desconhecido.

Sintaxe

```
expression IS [ NOT ] NULL
```

Argumentos

expressão

Qualquer expressão, tal como uma coluna.

IS NULL

É verdadeiro quando o valor de expressão é nulo e falso quando ele tem um valor.

IS NOT NULL

É falso quando o valor de expressão é nulo e verdadeiro quando ele tem um valor.

Exemplo

Este exemplo indica quantas vezes a tabela SALES contém null no campo QTYSOLD:

```
select count(*) from sales
where qtysold is null;
count
-------
0
(1 row)
```

Condição EXISTS

As condições EXISTS testam a existência de linhas em uma subconsulta e retornam verdadeiro se uma subconsulta retornar pelo menos uma linha. Se NOT estiver especificado, a condição retorna verdadeiro se uma subconsulta não retornar qualquer linha.

Sintaxe

```
[ NOT ] EXISTS (table_subquery)
```

Argumentos

EXISTS

É verdadeiro quando table_subquery retorna pelo menos uma linha.

NOT EXISTS

É verdadeiro quando table_subquery não retorna qualquer linha.

table_subquery

Uma subconsulta que avalia em uma tabela com uma ou mais colunas e uma ou mais linhas.

Exemplo

Este exemplo retorna todos os identificadores de data, um de cada vez, para cada data teve uma venda de qualquer tipo:

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
1827
1828
1829
...
```

Condição IN

Uma IN condição testa um valor para associação em um conjunto de valores ou em uma subconsulta.

Sintaxe

```
expression [ NOT ] IN (expr_list | table_subquery)
```

Argumentos

expressão

Uma expressão numérica, de caractere ou de data e hora que é avaliada em relação a expr_list ou table_subquery e deve ser compatível com o tipo de dados daquela lista ou subconsulta.

expr_list

Uma ou várias expressões delimitadas por vírgula ou um ou mais conjuntos de expressões delimitadas por vírgula entre parênteses.

table_subquery

Uma subconsulta que avalia em uma tabela com uma ou mais linhas, mas é limitada a somente uma coluna em sua lista de seleção.

IN | NOT IN

IN retorna verdadeiro se a expressão é um membro da lista de expressão ou consulta. NOT IN retorna verdadeiro se a expressão não é um membro. IN e NOT IN retornam null e nenhuma linha é retornada nos seguintes casos: Se a expressão resulta em nulo; ou se não há valores expr_list ou table_subquery correspondentes e pelo menos uma dessas linhas de comparação resulta em null.

Exemplos

As seguintes condições são verdadeiras somente para os valores listados:

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

Otimização para grandes listas IN

Para otimizar a performance da consulta, uma lista IN que inclua mais do que 10 valores é internamente avaliada como uma matriz escalar. Listas IN com menos do que 10 valores são avaliadas como uma série de predicados OR. Essa otimização é compatível com os tipos de dados SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP e TIMESTAMPTZ.

Observe a saída EXPLAIN para a consulta para visualizar o efeito desta otimização. Por exemplo:

```
explain select * from sales
QUERY PLAN
------------------------------------------------------------------------
XN Seq Scan on sales  (cost=0.00..6035.96 rows=86228 width=53)
Filter: (salesid = ANY ('{1,2,3,4,5,6,7,8,9,10,11}'::integer[]))
(2 rows)
```

Consultar dados aninhados

AWS Clean Rooms oferece acesso compatível com SQL a dados relacionais e aninhados.

AWS Clean Rooms usa notação pontilhada e matriz subscrita para navegação de caminhos ao acessar dados aninhados. Também permite o FROM itens de cláusula para iterar em matrizes e usar para operações não aninhadas. Os tópicos a seguir fornecem descrições dos diferentes padrões de consulta que combinam o uso do tipo de array/struct/map dados com navegação por caminhos e matrizes, desaninhamento e uniões.

Tópicos

Navegação

AWS Clean Rooms permite a navegação em matrizes e estruturas usando a notação de `[...]` colchetes e pontos, respectivamente. Além disso, você pode misturar navegação em estruturas usando a notação de pontos e matrizes usando a notação de colchetes.

Example

Por exemplo, o exemplo de consulta a seguir pressupõe que a coluna de dados da matriz `c_orders` é uma matriz com uma estrutura e um atributo denominado `o_orderkey`.

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

Você pode usar as notações de ponto e colchetes em todos os tipos de consultas, como filtragem, junção e agregação. Você pode usar essas notações em uma consulta na qual normalmente há referências de coluna.

Example

O exemplo a seguir usa uma instrução SELECT que filtra resultados.

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0].o_orderkey IS NOT NULL;
```

Example

O exemplo a seguir usa a navegação entre colchetes e pontos nas cláusulas GROUP BY e 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;
```

Desaninhar consultas

Para desaninhar consultas, AWS Clean Rooms habilita a iteração em matrizes. Ele faz isso navegando pela matriz usando a cláusula FROM de uma consulta.

Example

Com o exemplo anterior, o exemplo a seguir itera sobre os valores do atributo para `c_orders`.

```
SELECT o FROM customer_orders_lineitem c, c.c_orders o;
```

A sintaxe de desaninhamento é uma extensão da cláusula FROM. No SQL padrão, a cláusula FROM `x (AS) y` significa que `y` itera sobre cada tupla em relação a `x`. Nesse caso, `x` refere-se a uma relação, e `y` refere-se a um alias para a relação `x`. Da mesma forma, a sintaxe de desaninhamento usando o item da cláusula FROM `x (AS) y` significa que `y` itera sobre cada valor na expressão da matriz `x`. Nesse caso, `x` é uma expressão de matriz e `y` é um alias para `x`.

O operando esquerdo também pode usar a notação de pontos e colchetes para navegação regular.

Example

No exemplo anterior:

- `customer_orders_lineitem c` é a iteração sobre a tabela base `customer_order_lineitem`
- `c.c_orders o` é a iteração sobre o `c.c_orders array`

Para iterar sobre atributo o`_lineitems`, que é uma matriz dentro de uma matriz, é necessário adicionar várias cláusulas.

```
SELECT o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

AWS Clean Rooms também suporta um índice de matriz ao iterar sobre a matriz usando o AT palavra-chave. A cláusula `x AS y AT z` itera sobre a matriz `x` e gera o campo `z`, que é o índice de matriz.

Example

O exemplo a seguir mostra como o índice da matriz funciona.

```
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

O exemplo a seguir itera sobre uma 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

O exemplo a seguir itera sobre uma matriz de vários níveis. O exemplo usa várias cláusulas unnest para iterar nas matrizes mais internas. A `f.multi_level_array` AS a matriz repete. `multi_level_array` A matriz AS elemento é a iteração sobre as matrizes internas. `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)
```

Semântica lax

Por padrão, as operações de navegação em valores de dados aninhados retornam nulo em vez de retornar um erro quando a navegação é inválida. A navegação de objetos será inválida se o valor dos dados aninhados não for um objeto ou se o valor dos dados aninhados for um objeto, mas não contiver o nome do atributo usado na consulta.

Example

Por exemplo, a consulta a seguir acessa um nome de atributo inválido na coluna de dados aninhados `c_orders`:

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

A navegação da matriz retornará nulo se o valor dos dados aninhados não for uma matriz ou se o índice da matriz estiver fora dos limites.

Example

A consulta a seguir retorna nulo porque `c_orders[1][1]` está fora dos limites.

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

Tipos de introspecção

Colunas de tipo de dados aninhadas suportam funções de inspeção que retornam o tipo e outras informações de tipo sobre o valor. O AWS Clean Rooms oferece suporte às seguintes funções booleanas para colunas de dados aninhados:

- 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 essas funções retornam false se o valor de entrada for nulo. IS_SCALAR, IS_OBJECT e IS_ARRAY são mutuamente exclusivos e cobrem todos os valores possíveis, exceto nulo. Para inferir os tipos correspondentes aos dados, AWS Clean Rooms usa a função JSON_TYPEOF que retorna o tipo (o nível superior) do valor dos dados aninhados, conforme mostrado no exemplo a seguir:

```
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
```

Histórico do documento para a Referência AWS Clean Rooms SQL

A tabela a seguir descreve as versões da documentação da Referência AWS Clean Rooms SQL.

Para receber notificações sobre atualizações dessa documentação, você pode se inscrever em o feed RSS. Para assinar as atualizações de RSS, você deve ter um plug-in de RSS habilitado para o navegador que está usando.

Alteração	Descrição	Data
O Spark SQL é compatível com dicas	AWS Clean Rooms O Spark SQL oferece suporte a dicas de consulta para otimizar o desempenho da consulta e reduzir os custos de computação.	20 de janeiro de 2026
O Spark SQL é compatível com CACHE TABLE	AWS Clean Rooms O Spark SQL é compatível com o comando CACHE TABLE, que permite que os clientes armazenem tabelas existente s em cache ou criem e armazenem novas tabelas a partir dos resultados da consulta para melhorar o desempenho da consulta.	22 de outubro de 2025
O Spark SQL suporta as funções FIRST e LAST Window	AWS Clean Rooms O Spark SQL é compatível com as seguintes funções de janela: PRIMEIRA e ÚLTIMA.	12 de junho de 2025
Atualizações da documenta ção da função Spark SQL	Atualização somente com documentação para refletir com precisão as funções	20 de maio de 2025

	compatíveis do Spark SQL. A documentação de 25 funções não suportadas foi removida, incluindo **<=>** operator, SIMILAR TO, LISTAGG e ARRAY_INSERT. Nomes de função corrigidos de DATEADD a DATE_ADD, DATEDIFF a DATE_DIFF , ISNULL a IS_NULL e ISNOTNULL a IS_NOT_NULL. Corrigido um erro de digitação nos exemplos de DATE_PART .	
AWS Clean Rooms Spark SQL	Agora, os clientes podem executar consultas usando algumas condições, funções, comandos e convenções SQL compatíveis com o mecanismo de análise Spark SQL.	29 de outubro de 2024
Comandos SQL e funções SQL — atualização	Foram adicionados exemplos para a cláusula JOIN, o operador de conjunto EXCEPT, a expressão condicional CASE e as seguintes funções: ANY_VALUE, NVL e COALESCE, NULLIF, CAST, CONVERT, CONVERT_T IMEZONE, EXTRACT, MOD, SIGN, CONCAT, FIRST_VAL UE e LAST_VALUE.	28 de fevereiro de 2024

Funções SQL - atualização	AWS Clean Rooms agora oferece suporte às seguintes funções SQL: Array, SUPER e VARBYTE. As seguintes funções matemáticas agora são suportadas: ACOS, ASIN, ATAN, COT ATAN2, DEXP, PI, POW, RADIANS e SIN. As seguintes funções JSON agora são suportadas: CAN_JSON_PARSE, JSON_PARSE e JSON_SERIALIZE.	6 de outubro de 2023
Suporte a tipos de dados aninhados	AWS Clean Rooms agora oferece suporte a tipos de dados aninhados.	30 de agosto de 2023
Regras de nomenclatura SQL - atualização	Alteração somente na documentação para esclarecer os nomes das colunas reservadas.	16 de agosto de 2023
Disponibilidade geral	A Referência AWS Clean Rooms SQL agora está disponível ao público em geral.	31 de julho de 2023

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.