Referensi SQL

AWS Clean Rooms

AWS Clean Rooms: Referensi SQL

Table of Contents

Ikhtisar SQL di AWS Clean Rooms

Selamat datang di Referensi AWS Clean Rooms SQL.

AWS Clean Roomsdibangun di sekitar standar industri Structured Query Language (SQL), bahasa query yang terdiri dari perintah dan fungsi yang Anda gunakan untuk bekerja dengan database dan objek database. SQL juga memberlakukan aturan mengenai penggunaan tipe data, ekspresi, dan literal.

Topik berikut memberikan informasi umum tentang konvensi dan aturan penamaan yang digunakan dalam Referensi SQL ini.

Topik

Bagian berikut memberikan informasi tentang literal, tipe data, perintah SQL, jenis fungsi SQL, dan kondisi SQL yang dapat Anda gunakan. AWS Clean Rooms

Untuk informasi selengkapnyaAWS Clean Rooms, lihat Panduan AWS Clean Rooms Pengguna dan Referensi AWS Clean Rooms API.

Konvensi referensi SQL

Bagian ini menjelaskan konvensi yang digunakan untuk menulis sintaks untuk ekspresi, perintah, dan fungsi SQL.

Karakter	Deskripsi
PENUTUP	Kata-kata dalam huruf kapital adalah kata kunci.
[ ]	Tanda kurung menunjukkan argumen opsional. Beberapa argumen dalam tanda kurung menunjukkan

Karakter	Deskripsi
	bahwa Anda dapat memilih sejumlah argumen. Selain itu, argumen dalam tanda kurung pada baris terpisah menunjukkan bahwa parser mengharapkan argumen berada dalam urutan yang tercantum dalam sintaks.
{ }	Tanda kurung menunjukkan bahwa Anda diminta untuk memilih salah satu penjelasan di dalam kurung kurawal.
|	Pipa menunjukkan bahwa Anda dapat memilih di antara argumen.
huruf miring	Kata-kata yang dicetak miring menunjukkan placehold er. Anda harus memasukkan nilai yang sesuai di tempat kata yang dicetak miring.
...	Elipsis menunjukkan bahwa Anda dapat mengulangi elemen sebelumnya.
'	Kata-kata dalam tanda kutip tunggal menunjukkan bahwa Anda harus mengetik tanda kutip.

Aturan penamaan SQL

Bagian berikut menjelaskan aturan penamaan SQL di AWS Clean Rooms.

Topik

- Nama dan kolom asosiasi tabel yang dikonfigurasi
- Kata yang dicadangkan

Nama dan kolom asosiasi tabel yang dikonfigurasi

Anggota yang dapat melakukan kueri menggunakan nama asosiasi tabel yang dikonfigurasi sebagai nama tabel dalam kueri. Nama asosiasi tabel yang dikonfigurasi dan kolom tabel yang dikonfigurasi dapat dialias dalam kueri.

Aturan penamaan berikut berlaku untuk nama asosiasi tabel yang dikonfigurasi, nama kolom tabel yang dikonfigurasi, dan alias:

- Mereka harus menggunakan karakter alfanumerik, garis bawah (_), atau tanda hubung (-) tetapi tidak dapat memulai atau mengakhiri dengan tanda hubung.
  - (Hanya aturan analisis khusus) Mereka dapat menggunakan tanda dolar ($) tetapi tidak dapat menggunakan pola yang mengikuti konstanta string yang dikutip dolar.

    Konstanta string yang dikutip dolar terdiri dari:
    - tanda dolar ($)
    - “tag” opsional dari nol atau lebih karakter
    - tanda dolar lainnya
    - urutan karakter sewenang-wenang yang membentuk konten string
    - tanda dolar ($)
    - tag yang sama yang memulai kutipan dolar
    - tanda dolar

    Misalnya: `$$invalid$$`
- Mereka tidak dapat berisi karakter tanda hubung (-) berturut-turut.
- Mereka tidak dapat memulai dengan salah satu awalan berikut:

  `padb_`, `pg_`, `stcs_`, `stl_`, `stll_`, `stv_`, `svcs_`, `svl_`, `svv_`, `sys_`, `systable_`
- Mereka tidak dapat berisi karakter garis miring terbalik (\), tanda kutip ('), atau spasi yang tidak dikutip ganda.
- Jika mereka mulai dengan karakter non-abjad, mereka harus berada dalam tanda kutip ganda (” “).
- Jika mengandung karakter tanda hubung (-), mereka harus berada dalam tanda kutip ganda (” “).
- Panjangnya harus antara 1 dan 127 karakter.
- Kata-kata yang dicadangkan harus dalam tanda kutip ganda (“”).
- Nama kolom berikut dicadangkan tidak dapat digunakan di AWS Clean Rooms (bahkan dengan tanda kutip):
  - oid
  - tableoid
  - xmin
  - cmin

- xmax
- cmax
- ctid

Kata yang dicadangkan

Berikut ini adalah daftar kata-kata yang dicadangkan di 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

Dukungan tipe data oleh mesin SQL

AWS Clean Rooms mendukung beberapa mesin SQL dan dialek. Memahami sistem tipe data di seluruh implementasi ini sangat penting untuk kolaborasi dan analisis data yang sukses. Tabel berikut menunjukkan tipe data yang setara di AWS Clean Rooms SQL, Snowflake SQL, dan Spark SQL.

Jenis data numerik

Tipe numerik mewakili berbagai jenis angka, dari bilangan bulat yang tepat hingga perkiraan nilai floating-point. Pilihan tipe numerik mempengaruhi persyaratan penyimpanan dan presisi komputasi. Jenis integer bervariasi menurut ukuran byte, sedangkan tipe desimal dan floating-point menawarkan opsi presisi dan skala yang berbeda.

Jenis data	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Deskripsi
Bilangan bulat 8-byte	BIGINT	Tidak didukung	BIGINT, PANJANG	Bilangan bulat yang ditandatangani dari -9.223.372.036.854.775.808 menjadi 9.223.372.036,854.775.807.
Bilangan bulat 4-byte	INT	Tidak didukung	INT, BILANGAN BULAT	Bilangan bulat yang ditandatangani dari -2.147.483.648 menjadi 2.147.483.647
Bilangan bulat 2-byte	SMALLINT	Tidak didukung	SMALLINT, PENDEK	Bilangan bulat yang ditandatangani dari

Jenis data	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Deskripsi
				-32.768 hingga 32.767
Bilangan bulat 1 byte	Tidak didukung	Tidak didukung	TINYINT, BYTE	Bilangan bulat yang ditandatangani dari -128 hingga 127
Pelampung Presisi Ganda	GANDA, PRESISI GANDA	MENGAPUNG ,, FLOAT4 FLOAT8, GANDA, PRESISI GANDA, NYATA	DOUBLE	Nomor titik mengambang presisi ganda 8-byte
Pelampung Presisi Tunggal	NYATA, MENGAPUNG	Tidak didukung	FLOAT	Nomor titik mengambang presisi tunggal 4-byte

Jenis data	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Deskripsi
Desimal (presisi tetap)	DECIMAL	DESIMAL, NUMERIK, ANGKA **Note** Snowflake secara otomatis mengalias tipe numerik persis dengan lebar yang lebih kecil (INT, BIGINT, SMALLINT, dll.) ke NUMBER.	DESIMAL, NUMERIK,	Nomor desimal yang ditandatangani dengan presisi arbitrer
Desimal (dengan presisi)	DESIMAL (p)	DESIMAL (p), ANGKA (p)	DESIMAL (p)	Angka desimal presisi tetap
Desimal (dengan skala)	DECIMAL(p,s)	DESIMAL (p, s), BILANGAN (p, s)	DECIMAL(p,s)	Angka desimal presisi tetap dengan skala

Tipe data Boolean

Jenis Boolean mewakili nilai true/false logis sederhana. Jenis ini konsisten di seluruh mesin SQL dan biasanya digunakan untuk flag, kondisi, dan operasi logis.

Jenis data	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Deskripsi
Boolean	BOOLEAN	BOOLEAN	BOOLEAN	Merupakan true/false nilai

Jenis data tanggal dan waktu

Jenis tanggal dan waktu menangani data temporal, dengan berbagai tingkat presisi dan kesadaran zona waktu. Jenis ini mendukung format yang berbeda untuk menyimpan tanggal, waktu, dan stempel waktu, dengan opsi untuk memasukkan atau mengecualikan informasi zona waktu.

Jenis data	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Deskripsi
Date	DATE	DATE	DATE	Nilai tanggal (tahun, bulan, hari) tanpa zona waktu
Waktu	TIME	Tidak didukung	Tidak didukung	Waktu hari di UTC, tanpa zona waktu
Waktu dengan TZ	JADWAL	Tidak didukung	Tidak didukung	Waktu hari di UTC, dengan zona waktu
Stempel waktu	TIMESTAMP	STEMPEL WAKTU, STEMPEL WAKTU	TIMESTAMP_NTZ	Stempel waktu tanpa zona waktu

Jenis data	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Deskripsi
				Note NTZ menunjukkan “Tidak Ada Zona Waktu”
Stempel waktu dengan TZ	TIMESTAMPTZ	STAMP_LTZ	STEMPEL WAKTU, STEMPEL WAKTU	Stempel waktu dengan zona waktu lokal Note LTZ menunjukkan “Zona Waktu Lokal”

Tipe data karakter

Jenis karakter menyimpan data tekstual, menawarkan opsi panjang tetap dan panjang variabel. Jenis ini menangani string teks dan data biner, dengan spesifikasi panjang opsional untuk mengontrol alokasi penyimpanan.

Jenis data	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Deskripsi
Karakter panjang tetap	CHAR	CHAR, KARAKTER	CHAR, KARAKTER	String karakter dengan panjang tetap
Karakter panjang tetap dengan Panjang	CHAR(n)	CHAR (n), KARAKTER (n)	CHAR (n), KARAKTER (n)	String karakter dengan panjang tetap dengan panjang tertentu
Karakter panjang variabel	VARCHAR	VARCHAR, STRING, TEKS	VARCHAR, STRING	String karakter panjang variabel
Karakter panjang variabel dengan Panjang	VARCHAR(n)	VARCHAR (n), STRING (n), TEKS (n)	VARCHAR(n)	String karakter panjang variabel dengan batas panjang
Biner	VARBYTE	BINARY, VARBINARY	BINARY	Urutan byte biner
Biner dengan Panjang	VARBYTE(n)	Tidak didukung	Tidak didukung	Urutan byte biner dengan batas panjang

Tipe data terstruktur

Tipe terstruktur memungkinkan organisasi data yang kompleks dengan menggabungkan beberapa nilai ke dalam bidang tunggal. Ini termasuk array untuk koleksi terurut, peta untuk pasangan kunci-nilai, dan struct untuk membuat struktur data kustom dengan bidang bernama.

Jenis data	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Deskripsi
Array	ARRAY <type>	ARRAY (tipe)	ARRAY <type>	Urutan elemen yang diurutkan dari jenis yang sama **Note** Jenis array harus berisi elemen dari tipe yang sama
Peta	PETA<key, value>	MAP (kunci, nilai)	PETA<key, value>	Koleksi pasangan kunci-nilai **Note** Jenis peta harus berisi elemen dari jenis yang sama

Jenis data	AWS Clean Rooms SQL	Snowflake SQL	Spark SQL	Deskripsi
Struct	STRUCT< field1: type1, field2: type2>	OBJEK (field1 type1, field2 type2)	STRUCT< field1: type1, field2: type2 >	Struktur dengan bidang bernama dari jenis tertentu **Note** Sintaks tipe terstruktur mungkin sedikit berbeda antara implementasi
Super	SUPER	Tidak didukung	Tidak didukung	Tipe fleksibel yang mendukung semua tipe data termasuk tipe kompleks

AWS Clean Rooms Spark SQL

AWS Clean Rooms Spark SQL memberlakukan aturan mengenai penggunaan tipe data, ekspresi, dan literal.

Untuk informasi selengkapnya tentang AWS Clean Rooms Spark SQL, lihat Panduan AWS Clean Rooms Pengguna dan Referensi AWS Clean Rooms API.

Topik berikut memberikan informasi tentang literal, tipe data, perintah, fungsi, dan kondisi yang didukung di AWS Clean Rooms Spark SQL.

Topik

Literal

Literal atau konstanta adalah nilai data tetap, terdiri dari urutan karakter atau konstanta numerik.

AWS Clean Rooms Spark SQL mendukung beberapa jenis literal, termasuk:

- Literal numerik untuk bilangan bulat, desimal, dan floating-point.
- Literal karakter, juga disebut sebagai string, string karakter, atau konstanta karakter, digunakan untuk menentukan nilai string karakter.
- Literal tanggal, waktu, dan stempel waktu, digunakan dengan tipe data datetime. Untuk informasi selengkapnya, lihat Tanggal, waktu, dan literal stempel waktu.
- Literal interval. Untuk informasi selengkapnya, lihat Literal interval.
- Literal Boolean. Untuk informasi selengkapnya, lihat Literal Boolean.
- Null literal, digunakan untuk menentukan nilai null.
- Hanya TAB, CARRIAGE RETURN (CR), dan LINE FEED (LF) Karakter kontrol Unicode dari kategori umum Unicode (Cc) didukung.

AWS Clean Rooms Spark SQL tidak mendukung referensi langsung ke literal string dalam klausa SELECT, tetapi mereka dapat digunakan dalam fungsi seperti CAST.

+ Operator (Penggabungan)

Menggabungan literal numerik, literal string, dan/atau literal datetime dan interval. Mereka berada di kedua sisi simbol + dan mengembalikan jenis yang berbeda berdasarkan input di kedua sisi simbol +.

Sintaks

```
numeric + string
```

```
date + time
```

```
date + timetz
```

Urutan argumen dapat dibalik.

Pendapat

numeric literals

Literal atau konstanta yang mewakili angka dapat berupa integer atau floating-point.

string literals

String, string karakter, atau konstanta karakter

date

A DATE kolom atau ekspresi yang secara implisit mengkonversi ke DATE.

time

A TIME kolom atau ekspresi yang secara implisit mengkonversi ke TIME.

timetz

A TIMETZ kolom atau ekspresi yang secara implisit mengkonversi ke TIMETZ.

Contoh

Contoh tabel berikut TIME_TEST memiliki kolom TIME_VAL (jenis TIME) dengan tiga nilai dimasukkan.

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

Jenis Data

Setiap nilai yang disimpan atau diambil oleh AWS Clean Rooms Spark SQL memiliki tipe data dengan sekumpulan properti terkait yang tetap. Tipe data dideklarasikan saat tabel dibuat. Tipe data membatasi kumpulan nilai yang dapat berisi kolom atau argumen.

Tabel berikut mencantumkan tipe data yang dapat Anda gunakan di AWS Clean Rooms Spark SQL.

Nama tipe data	Jenis data	Alias	Deskripsi
ARRAY	the section called “Jenis bersarang”	Tidak berlaku	Tipe data bersarang array
BIGINT	the section called “Jenis numerik”	Tidak berlaku	Bilangan bulat delapan byte bertanda
BINARY	the section called “Tipe biner”	Tidak berlaku	Nilai urutan byte
BOOLEAN	the section called “Jenis Boolean”	BOOL	Logis Boolean (benar/salah)
BYTE	the section called “Jenis numerik”	Tidak berlaku	Nomor integer bertanda 1-byte, dari -128 hingga 127
CHAR	the section called “Jenis karakter”	KARAKTER	String karakter dengan panjang tetap
DATE	the section called “Jenis Datetime”	Tidak berlaku	Tanggal kalender (tahun, bulan, hari)

Nama tipe data	Jenis data	Alias	Deskripsi
DECIMAL	the section called “Jenis numerik”	NUMERIC	Numerik persis dari presisi yang dapat dipilih
FLOAT	the section called “Jenis numerik”	FLOAT8, PRESISI GANDA	Angka floating-point presisi ganda
INTEGER	the section called “Jenis numerik”	INT	Bilangan bulat empat byte bertanda
INTERVAL	the section called “Jenis Datetime”	Tidak berlaku	Durasi waktu dalam pesanan hari ke waktu atau pesanan tahun ke bulan
LONG	the section called “Jenis numerik”	Tidak berlaku	Nomor bilangan bulat bertanda 8-byte
PETA	the section called “Jenis bersarang”	Tidak berlaku	Memetakan tipe data bersarang
REAL	the section called “Jenis numerik”	FLOAT4	Angka floating-point presisi tunggal
SHORT	the section called “Jenis numerik”	Tidak berlaku	Nomor integer bertanda 2-byte.
SMALLINT	the section called “Jenis numerik”	Tidak berlaku	Bilangan bulat dua byte bertanda
STRUCT	the section called “Jenis bersarang”	Tidak berlaku	Struct tipe data bersarang
STAMP_LTZ	the section called “Jenis Datetime”	Tidak berlaku	Waktu hari dengan zona waktu lokal

Nama tipe data	Jenis data	Alias	Deskripsi
TIMESTAMP_NTZ	the section called “Jenis Datetime”	Tidak berlaku	Waktu hari tanpa zona waktu
TINYINT	the section called “Jenis numerik”	Tidak berlaku	Nomor integer bertanda 1-byte, dari -128 hingga 127
VARCHAR	the section called “Jenis karakter”	KARAKTER BERVARIASI	String karakter panjang variabel dengan batas yang ditentukan pengguna

> **Note**
>
> Tipe data bersarang ARRAY, STRUCT, dan MAP saat ini hanya diaktifkan untuk aturan analisis kustom. Untuk informasi selengkapnya, lihat Jenis bersarang.

Karakter multibyte

Tipe data VARCHAR mendukung karakter multibyte UTF-8 hingga maksimal empat byte. Karakter lima byte atau lebih lama tidak didukung. Untuk menghitung ukuran kolom VARCHAR yang berisi karakter multibyte, kalikan jumlah karakter dengan jumlah byte per karakter. Misalnya, jika string memiliki empat karakter Mandarin, dan setiap karakter panjangnya tiga byte, maka Anda memerlukan kolom VARCHAR (12) untuk menyimpan string.

Tipe data VARCHAR tidak mendukung titik kode UTF-8 yang tidak valid berikut ini:

`0xD800 – 0xDFFF`(Urutan byte: `ED A0 80` —) `ED BF BF`

Tipe data CHAR tidak mendukung karakter multibyte.

Jenis numerik

Tipe data numerik termasuk bilangan bulat, desimal, dan angka floating-point.

Topik

Jenis bilangan bulat

Gunakan tipe data berikut untuk menyimpan seluruh nomor dari berbagai rentang. Anda tidak dapat menyimpan nilai di luar rentang yang diizinkan untuk setiap jenis.

Nama	Penyimpanan	Kisaran
SMALLINT	2 byte	-32768 ke +32767
SHORT	2 byte	-32768 ke +32767
INTEGER atau INT	4 byte	-2147483648 ke +2147483647
BIGINT	8 byte	-9223372036854775808 ke 9223372036854775807
LONG	8 byte	-9223372036854775808 ke 9223372036854775807

Jenis DESIMAL atau NUMERIK

Gunakan tipe data DECIMAL atau NUMERIK untuk menyimpan nilai dengan presisi yang ditentukan pengguna. Kata kunci DECIMAL dan NUMERIK dapat dipertukarkan. Dalam dokumen ini, desimal adalah istilah yang disukai untuk tipe data ini. Istilah numerik digunakan secara umum untuk merujuk pada tipe data integer, desimal, dan floating-point.

Penyimpanan	Kisaran
Variabel, hingga 128 bit untuk tipe DECIMAL yang tidak terkompresi.	Bilangan bulat bertanda 128-bit dengan presisi hingga 38 digit.

Tentukan kolom DECIMAL dalam tabel dengan menentukan dan: *precision* *scale*

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

precision

Jumlah total digit signifikan dalam seluruh nilai: jumlah digit di kedua sisi titik desimal. Misalnya, angka tersebut `48.2891` memiliki presisi 6 dan skala 4. Presisi default, jika tidak ditentukan, adalah 18. Presisi maksimum adalah 38.

Jika jumlah digit di sebelah kiri titik desimal dalam nilai input melebihi presisi kolom dikurangi skalanya, nilai tidak dapat disalin ke kolom (atau dimasukkan atau diperbarui). Aturan ini berlaku untuk setiap nilai yang berada di luar rentang definisi kolom. Misalnya, rentang nilai yang diizinkan untuk `numeric(5,2)` kolom adalah `-999.99` untuk`999.99`.

scale

Jumlah digit desimal di bagian pecahan nilai, di sebelah kanan titik desimal. Bilangan bulat memiliki skala nol. Dalam spesifikasi kolom, nilai skala harus kurang dari atau sama dengan nilai presisi. Skala default, jika tidak ditentukan, adalah 0. Skala maksimum adalah 37.

Jika skala nilai input yang dimuat ke dalam tabel lebih besar dari skala kolom, nilainya dibulatkan ke skala yang ditentukan. Misalnya, kolom PRICEPAID dalam tabel PENJUALAN adalah kolom DECIMAL (8,2). Jika nilai DECIMAL (8,4) dimasukkan ke dalam kolom PRICEPAID, nilainya dibulatkan ke skala 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)
```

Namun, hasil pemeran eksplisit nilai yang dipilih dari tabel tidak dibulatkan.

> **Note**
>
> Nilai positif maksimum yang dapat Anda masukkan ke dalam kolom DECIMAL (19,0) adalah `9223372036854775807` ($2^{63}-1$). Nilai negatif maksimum adalah `-9223372036854775807`. Misalnya, upaya untuk memasukkan nilai `9999999999999999999` (19 nines) akan menyebabkan kesalahan overflow. Terlepas dari penempatan titik desimal, string terbesar yang AWS Clean Rooms dapat mewakili sebagai angka DESIMAL adalah. `9223372036854775807` Misalnya, nilai terbesar yang dapat Anda muat ke kolom DECIMAL (19,18) adalah. `9.223372036854775807`
> Aturan-aturan ini adalah karena hal-hal berikut:
>
> - Nilai DESIMAL dengan 19 atau kurang digit presisi signifikan disimpan secara internal sebagai bilangan bulat 8-byte.
> - Nilai DESIMAL dengan 20 hingga 38 digit presisi signifikan disimpan sebagai bilangan bulat 16-byte.

Catatan tentang menggunakan kolom DESIMAL atau NUMERIK 128-bit

Jangan sewenang-wenang menetapkan presisi maksimum ke kolom DECIMAL kecuali Anda yakin bahwa aplikasi Anda memerlukan presisi itu. Nilai 128-bit menggunakan ruang disk dua kali lebih banyak daripada nilai 64-bit dan dapat memperlambat waktu eksekusi kueri.

Jenis Floating-point

Gunakan tipe data REAL dan DOUBLE PRECISION untuk menyimpan nilai numerik dengan presisi variabel. Jenis ini adalah tipe yang tidak tepat, yang berarti bahwa beberapa nilai disimpan sebagai perkiraan, sehingga menyimpan dan mengembalikan nilai tertentu dapat mengakibatkan sedikit perbedaan. Jika Anda memerlukan penyimpanan dan perhitungan yang tepat (seperti untuk jumlah uang), gunakan tipe data DECIMAL.

REAL mewakili format floating point presisi tunggal, menurut IEEE Standard 754 untuk Floating-Point Arithmetic. Ini memiliki presisi sekitar 6 digit, dan kisaran sekitar 1E-37 hingga 1E+37. Anda juga dapat menentukan tipe data ini sebagai FLOAT4.

DOUBLE PRECISION mewakili format floating point presisi ganda, menurut IEEE Standard 754 untuk Binary Floating-Point Arithmetic. Ini memiliki presisi sekitar 15 digit, dan kisaran sekitar 1E-307 hingga 1E+308. Anda juga dapat menentukan tipe data ini sebagai FLOAT atau FLOAT8.

Perhitungan dengan nilai numerik

Dalam AWS Clean Rooms, komputasi mengacu pada operasi matematika biner: penjumlahan, pengurangan, perkalian, dan pembagian. Bagian ini menjelaskan jenis pengembalian yang diharapkan untuk operasi ini, serta rumus spesifik yang diterapkan untuk menentukan presisi dan skala saat tipe data DECIMAL terlibat.

Ketika nilai numerik dihitung selama pemrosesan kueri, Anda mungkin mengalami kasus di mana perhitungan tidak mungkin dan kueri mengembalikan kesalahan luapan numerik. Anda mungkin juga mengalami kasus di mana skala nilai yang dihitung bervariasi atau tidak terduga. Untuk beberapa operasi, Anda dapat menggunakan casting eksplisit (jenis promosi) atau parameter AWS Clean Rooms konfigurasi untuk mengatasi masalah ini.

Untuk informasi tentang hasil perhitungan serupa dengan fungsi SQL, lihat. AWS Clean Rooms Fungsi Spark SQL

Jenis pengembalian untuk perhitungan

Mengingat kumpulan tipe data numerik yang didukung AWS Clean Rooms, tabel berikut menunjukkan jenis pengembalian yang diharapkan untuk operasi penambahan, pengurangan, perkalian, dan pembagian. Kolom pertama di sisi kiri tabel mewakili operan pertama dalam perhitungan, dan baris atas mewakili operan kedua.

Operan 1	Operan 2	Jenis pengembalian
SMALLINT atau PENDEK	SMALLINT atau PENDEK	SMALLINT atau PENDEK
SMALLINT atau PENDEK	INTEGER	INTEGER
SMALLINT atau PENDEK	BIGINT	BIGINT
SMALLINT atau PENDEK	DECIMAL	DECIMAL
SMALLINT atau PENDEK	FLOAT4	FLOAT8
SMALLINT atau PENDEK	FLOAT8	FLOAT8

Operan 1	Operan 2	Jenis pengembalian
INTEGER	INTEGER	INTEGER
INTEGER	BIGINT atau LONG	BIGINT atau LONG
INTEGER	DECIMAL	DECIMAL
INTEGER	FLOAT4	FLOAT8
INTEGER	FLOAT8	FLOAT8
BIGINT atau LONG	BIGINT atau LONG	BIGINT atau LONG
BIGINT atau LONG	DECIMAL	DECIMAL
BIGINT atau LONG	FLOAT4	FLOAT8
BIGINT atau LONG	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL
DECIMAL	FLOAT4	FLOAT8
DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8
FLOAT8	FLOAT8	FLOAT8

Presisi dan skala hasil DECIMAL yang dihitung

Tabel berikut merangkum aturan untuk menghitung presisi dan skala yang dihasilkan ketika operasi matematika mengembalikan hasil DECIMAL. Dalam tabel ini, `p1` dan `s1` mewakili presisi dan skala operan pertama dalam perhitungan. `p2`dan `s2` mewakili presisi dan skala operan kedua. (Terlepas dari perhitungan ini, presisi hasil maksimum adalah 38, dan skala hasil maksimum adalah 38.)

Operasi	Ketepatan dan skala hasil
+ atau -	Skala = `max(s1,s2)`

Operasi	Ketepatan dan skala hasil
	presisi = `max(p1-s1,p2-s2)+1+scale`
*	Skala = `s1+s2` presisi = `p1+p2+1`
/	Skala = `max(4,s1+p2-s2+1)` presisi = `p1-s1+ s2+scale`

Misalnya, kolom PRICEPAID dan KOMISI dalam tabel PENJUALAN keduanya adalah kolom DECIMAL (8,2). Jika Anda membagi PRICEPAID dengan KOMISI (atau sebaliknya), rumusnya diterapkan sebagai berikut:

```
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)
```

Perhitungan berikut adalah aturan umum untuk menghitung presisi dan skala yang dihasilkan untuk operasi yang dilakukan pada nilai DECIMAL dengan operator yang ditetapkan seperti UNION, INTERSECT, dan EXCEPT atau fungsi seperti COALESCE dan DECODE:

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

Misalnya, DEC1 tabel dengan satu kolom DECIMAL (7,2) digabungkan dengan DEC2 tabel dengan satu kolom DECIMAL (15,3) untuk membuat tabel. DEC3 Skema DEC3 menunjukkan bahwa kolom menjadi kolom NUMERIK (15,3).

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

Pada contoh di atas, rumus diterapkan sebagai berikut:

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

```
= 12 + 3 = 15

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

Catatan tentang operasi divisi

Untuk operasi divisi, divide-by-zero kondisi mengembalikan kesalahan.

Batas skala 100 diterapkan setelah presisi dan skala dihitung. Jika skala hasil yang dihitung lebih besar dari 100, hasil pembagian diskalakan sebagai berikut:

- presisi = `precision - (scale - max_scale)`
- Skala = `max_scale`

Jika presisi yang dihitung lebih besar dari presisi maksimum (38), presisi dikurangi menjadi 38, dan skala menjadi hasil dari: `max(38 + scale - precision), min(4, 100))`

Kondisi luapan

Overflow diperiksa untuk semua perhitungan numerik. Data DESIMAL dengan presisi 19 atau kurang disimpan sebagai bilangan bulat 64-bit. Data DESIMAL dengan presisi yang lebih besar dari 19 disimpan sebagai bilangan bulat 128-bit. Ketepatan maksimum untuk semua nilai DECIMAL adalah 38, dan skala maksimum adalah 37. Kesalahan overflow terjadi ketika nilai melebihi batas ini, yang berlaku untuk set hasil menengah dan akhir:

- Casting eksplisit menghasilkan kesalahan runtime overflow saat nilai data tertentu tidak sesuai dengan presisi atau skala yang diminta yang ditentukan oleh fungsi cast. Misalnya, Anda tidak dapat mentransmisikan semua nilai dari kolom PRICEPAID di tabel PENJUALAN (kolom DECIMAL (8,2)) dan mengembalikan hasil DECIMAL (7,3):

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

  Kesalahan ini terjadi karena beberapa nilai yang lebih besar di kolom PRICEPAID tidak dapat dilemparkan.
- Operasi perkalian menghasilkan hasil di mana skala hasil adalah jumlah dari skala masing-masing operan. Jika kedua operan memiliki skala 4, misalnya, skala hasilnya adalah 8, hanya menyisakan

10 digit untuk sisi kiri titik desimal. Oleh karena itu, relatif mudah untuk mengalami kondisi luapan ketika mengalikan dua angka besar yang keduanya memiliki skala signifikan.

Perhitungan numerik dengan tipe INTEGER dan DECIMAL

Ketika salah satu operan dalam perhitungan memiliki tipe data INTEGER dan operan lainnya adalah DECIMAL, operan INTEGER secara implisit dilemparkan sebagai DECIMAL.

- SMALLINT atau SHORT dilemparkan sebagai DECIMAL (5,0)
- INTEGER dilemparkan sebagai DECIMAL (10,0)
- BIGINT atau LONG dilemparkan sebagai DECIMAL (19,0)

Misalnya, jika Anda mengalikan SALES.COMMISSION, kolom DECIMAL (8,2), dan SALES.QTYSOLD, kolom SMALLINT, perhitungan ini dilemparkan sebagai:

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

Jenis karakter

Tipe data karakter termasuk CHAR (karakter) dan VARCHAR (karakter bervariasi).

Topik

- CHAR atau KARAKTER
- VARCHAR atau KARAKTER BERVARIASI
- Signifikansi trailing blank

CHAR atau KARAKTER

Gunakan kolom CHAR atau KARAKTER untuk menyimpan string dengan panjang tetap. String ini dilapisi dengan blanko, sehingga kolom CHAR (10) selalu menempati 10 byte penyimpanan.

```
char(10)
```

Kolom CHAR tanpa spesifikasi panjang menghasilkan kolom CHAR (1).

Tipe data CHAR dan VARCHAR didefinisikan dalam hal byte, bukan karakter. Kolom CHAR hanya dapat berisi karakter single-byte, sehingga kolom CHAR (10) dapat berisi string dengan panjang maksimum 10 byte.

Nama	Penyimpanan	Rentang (lebar kolom)
CHAR atau KARAKTER	Panjang string, termasuk trailing blank (jika ada)	4096 bita

VARCHAR atau KARAKTER BERVARIASI

Gunakan kolom VARCHAR atau CHARACTER VARY untuk menyimpan string panjang variabel dengan batas tetap. String ini tidak dilapisi dengan blanko, sehingga kolom VARCHAR (120) terdiri dari maksimum 120 karakter single-byte, 60 karakter dua-byte, 40 karakter tiga byte, atau 30 karakter empat byte.

```
varchar(120)
```

Jenis data VARCHAR didefinisikan dalam hal byte, bukan karakter. VARCHAR dapat berisi karakter multibyte, hingga maksimal empat byte per karakter. Misalnya, kolom VARCHAR (12) dapat berisi 12 karakter single-byte, 6 karakter dua-byte, 4 karakter tiga byte, atau 3 karakter empat byte.

Nama	Penyimpanan	Rentang (lebar kolom)
VARCHAR atau KARAKTER BERVARIASI	4 byte+total byte untuk karakter, di mana setiap karakter dapat 1 sampai 4 byte.	65535 byte (64K -1)

Signifikansi trailing blank

Baik tipe data CHAR dan VARCHAR menyimpan string hingga n byte panjangnya. Upaya untuk menyimpan string yang lebih panjang ke dalam kolom jenis ini menghasilkan kesalahan. Namun, jika karakter tambahan adalah semua spasi (kosong), string terpotong hingga panjang maksimum.

Jika string lebih pendek dari panjang maksimum, nilai CHAR dilapisi dengan kosong, tetapi nilai VARCHAR menyimpan string tanpa kosong.

Trailing blank dalam nilai CHAR selalu tidak signifikan secara semantik. Mereka diabaikan ketika Anda membandingkan dua nilai CHAR, tidak termasuk dalam perhitungan PANJANG, dan dihapus saat Anda mengonversi nilai CHAR ke tipe string lain.

Spasi trailing dalam nilai VARCHAR dan CHAR diperlakukan sebagai tidak signifikan secara semantik ketika nilai dibandingkan.

Perhitungan panjang mengembalikan panjang string karakter VARCHAR dengan spasi tambahan yang termasuk dalam panjangnya. Trailing blank tidak dihitung panjangnya untuk string karakter dengan panjang tetap.

Jenis Datetime

Tipe data datetime termasuk DATE, TIME, TIMESTAMP_LTZ, dan TIMESTAMP_NTZ.

Topik

DATE

Gunakan tipe data DATE untuk menyimpan tanggal kalender sederhana tanpa cap waktu.

Nama	Penyimpanan	Kisaran	Resolusi
DATE	4 byte	4713 SM hingga 294276 M	1 hari

STAMP_LTZ

Gunakan tipe data TIMESTAMP_LTZ untuk menyimpan nilai stempel waktu lengkap yang mencakup tanggal, waktu hari, dan zona waktu lokal.

TIMESTAMP mewakili nilai yang terdiri dari nilai bidang`year`,,,`month`,`day`,`hour`, dan `minutesecond`, dengan zona waktu lokal sesi. `timestamp`Nilai mewakili titik waktu absolut.

TIMESTAMP di Spark adalah alias yang ditentukan pengguna yang terkait dengan salah satu variasi TIMESTAMP_LTZ dan TIMESTAMP_NTZ. Anda dapat mengatur tipe stempel waktu default sebagai TIMESTAMP_LTZ (nilai default) atau TIMESTAMP_NTZ melalui konfigurasi. `spark.sql.timestampType`

TIMESTAMP_NTZ

Gunakan tipe data TIMESTAMP_NTZ untuk menyimpan nilai stempel waktu lengkap yang menyertakan tanggal, waktu hari, tanpa zona waktu lokal.

TIMESTAMP mewakili nilai yang terdiri dari nilai bidang`year`,,,`month`, `dayhour`, `minute` dan. `second` Semua operasi dilakukan tanpa memperhitungkan zona waktu.

TIMESTAMP di Spark adalah alias yang ditentukan pengguna yang terkait dengan salah satu variasi TIMESTAMP_LTZ dan TIMESTAMP_NTZ. Anda dapat mengatur tipe stempel waktu default sebagai TIMESTAMP_LTZ (nilai default) atau TIMESTAMP_NTZ melalui konfigurasi. `spark.sql.timestampType`

Contoh dengan tipe datetime

Contoh berikut menunjukkan cara bekerja dengan tipe datetime yang didukung oleh. AWS Clean Rooms

Contoh tanggal

Contoh berikut menyisipkan tanggal yang memiliki format berbeda dan menampilkan output.

```
select * from datetable order by 1;

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

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

Jika Anda memasukkan nilai stempel waktu ke kolom DATE, bagian waktu diabaikan dan hanya tanggal yang dimuat.

Contoh waktu

Contoh berikut menyisipkan nilai TIME dan TIMETZ yang memiliki format berbeda dan menampilkan output.

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

Tanggal, waktu, dan literal stempel waktu

Berikut ini adalah aturan untuk bekerja dengan literal tanggal, waktu, dan stempel waktu yang didukung oleh AWS Clean Rooms Spark SQL.

Tanggal

Tabel berikut menunjukkan tanggal masukan yang merupakan contoh valid dari nilai tanggal literal yang dapat Anda muat ke dalam AWS Clean Rooms tabel. `MDY DateStyle`Mode default diasumsikan berlaku. Mode ini berarti bahwa nilai bulan mendahului nilai hari dalam string seperti `1999-01-08` dan. `01/02/00`

> **Note**
>
> Tanggal atau stempel waktu literal harus dilampirkan dalam tanda kutip saat Anda memuatnya ke dalam tabel.

Tanggal masukan	Tanggal penuh
8 Januari 1999	8 Januari 1999
1999-01-08	8 Januari 1999

Tanggal masukan	Tanggal penuh
1/8/1999	8 Januari 1999
01/02/00	2 Januari 2000
2000-Jan-31	31 Januari 2000
Jan-31-2000	31 Januari 2000
31-Jan-2000	31 Januari 2000
20080215	15 Februari 2008
080215	15 Februari 2008
2008.366	31 Desember 2008 (bagian tiga digit tanggal harus antara 001 dan 366)

Kali

Tabel berikut menunjukkan waktu masukan yang merupakan contoh valid dari nilai waktu literal yang dapat Anda muat ke dalam AWS Clean Rooms tabel.

Waktu masukan	Deskripsi (bagian waktu)
04:05:06.789	4:05 AM dan 6.789 detik
04:05:06	4:05 AM dan 6 detik
04:05	4:05 AM tepatnya
040506	4:05 AM dan 6 detik
04:05AM	4:05 AM tepatnya; AM adalah opsional
04:05 SORE	4:05 PM tepatnya; nilai jam harus kurang dari 12
16:05	16:05 PM tepatnya

Nilai datetime khusus

Tabel berikut menunjukkan nilai-nilai khusus yang dapat digunakan sebagai literal datetime dan sebagai argumen untuk fungsi tanggal. Mereka membutuhkan tanda kutip tunggal dan dikonversi ke nilai stempel waktu biasa selama pemrosesan kueri.

Nilai khusus	Deskripsi
now	Mengevaluasi waktu mulai transaksi saat ini dan mengembalikan stempel waktu dengan presisi mikrodetik.
today	Mengevaluasi ke tanggal yang sesuai dan mengembalikan stempel waktu dengan nol untuk bagian waktu.
tomorrow	Mengevaluasi ke tanggal yang sesuai dan mengembalikan stempel waktu dengan nol untuk bagian waktu.
yesterday	Mengevaluasi ke tanggal yang sesuai dan mengembalikan stempel waktu dengan nol untuk bagian waktu.

Contoh berikut menunjukkan bagaimana now dan today bekerja dengan fungsi 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)
```

Literal interval

Berikut ini adalah aturan untuk bekerja dengan literal interval yang didukung oleh AWS Clean Rooms Spark SQL.

Gunakan interval literal untuk mengidentifikasi periode waktu tertentu, seperti `12 hours` atau`6 weeks`. Anda dapat menggunakan literal interval ini dalam kondisi dan perhitungan yang melibatkan ekspresi datetime.

> **Note**
>
> Anda tidak dapat menggunakan tipe data INTERVAL untuk kolom dalam AWS Clean Rooms tabel.

Interval dinyatakan sebagai kombinasi kata kunci INTERVAL dengan kuantitas numerik dan bagian tanggal yang didukung, misalnya `INTERVAL '7 days'` atau`INTERVAL '59 minutes'`. Anda dapat menghubungkan beberapa kuantitas dan unit untuk membentuk interval yang lebih tepat, misalnya:`INTERVAL '7 days, 3 hours, 59 minutes'`. Singkatan dan bentuk jamak dari setiap unit juga didukung; misalnya:`5 s`,`5 second`, dan `5 seconds` merupakan interval yang setara.

Jika Anda tidak menentukan bagian tanggal, nilai interval mewakili detik. Anda dapat menentukan nilai kuantitas sebagai pecahan (misalnya:`0.5 days`).

Contoh

Contoh berikut menunjukkan serangkaian perhitungan dengan nilai interval yang berbeda.

Contoh berikut menambahkan 1 detik ke tanggal yang ditentukan.

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

Contoh berikut menambahkan 1 menit ke tanggal yang ditentukan.

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

Contoh berikut menambahkan 3 jam dan 35 menit ke tanggal yang ditentukan.

```
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)
```

Contoh berikut menambahkan 52 minggu ke tanggal yang ditentukan.

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

Contoh berikut menambahkan 1 minggu, 1 jam, 1 menit, dan 1 detik ke tanggal yang ditentukan.

```
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)
```

Contoh berikut menambahkan 12 jam (setengah hari) ke tanggal yang ditentukan.

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

```
(1 row)
```

Contoh berikut mengurangi 4 bulan dari 31 Maret 2023 dan hasilnya adalah 30 November 2022. Perhitungan mempertimbangkan jumlah hari dalam sebulan.

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

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

Tipe data interval dan literal

Anda dapat menggunakan tipe data interval untuk menyimpan durasi waktu dalam unit seperti,,,`seconds`, `minuteshours`, `daysmonths`, dan`years`. Tipe data interval dan literal dapat digunakan dalam perhitungan datetime, seperti, menambahkan interval ke tanggal dan stempel waktu, menjumlahkan interval, dan mengurangi interval dari tanggal atau stempel waktu. Literal interval dapat digunakan sebagai nilai masukan untuk kolom tipe data interval dalam tabel.

Sintaks tipe data interval

Untuk menentukan tipe data interval untuk menyimpan durasi waktu dalam tahun dan bulan:

```
INTERVAL year_to_month_qualifier
```

Untuk menentukan tipe data interval untuk menyimpan durasi dalam hari, jam, menit, dan detik:

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

Sintaks interval literal

Untuk menentukan interval literal untuk menentukan durasi waktu dalam tahun dan bulan:

```
INTERVAL quoted-string year_to_month_qualifier
```

Untuk menentukan interval literal untuk menentukan durasi dalam hari, jam, menit, dan detik:

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

Pendapat

string yang dikutip

Menentukan nilai numerik positif atau negatif menentukan kuantitas dan unit datetime sebagai string input. Jika string yang dikutip hanya berisi angka, maka AWS Clean Rooms tentukan unit dari year_to_month_qualifier atau day_to_second_qualifier. Misalnya, `'23' MONTH` mewakili`1 year 11 months`, `'-2' DAY` mewakili`-2 days 0 hours 0 minutes 0.0 seconds`, `'1-2' MONTH` mewakili`1 year 2 months`, dan `'13 day 1 hour 1 minute 1.123 seconds' SECOND` mewakili`13 days 1 hour 1 minute 1.123 seconds`. Untuk informasi selengkapnya tentang format keluaran suatu interval, lihatGaya interval.

year_to_month_qualifier

Menentukan rentang interval. Jika Anda menggunakan kualifikasi dan membuat interval dengan satuan waktu yang lebih kecil dari qualifier, AWS Clean Rooms potong dan buang bagian interval yang lebih kecil. Nilai yang valid untuk year_to_month_qualifier adalah:

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

day_to_second_qualifier

Menentukan rentang interval. Jika Anda menggunakan kualifikasi dan membuat interval dengan satuan waktu yang lebih kecil dari qualifier, AWS Clean Rooms potong dan buang bagian interval yang lebih kecil. Nilai yang valid untuk day_to_second_qualifier adalah:

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

Output dari literal INTERVAL terpotong ke komponen INTERVAL terkecil yang ditentukan. Misalnya, saat menggunakan kualifikasi MINUTE, AWS Clean Rooms buang satuan waktu yang lebih kecil dari MINUTE.

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

Nilai yang dihasilkan terpotong menjadi. `'1 day 01:01:00'`

fractional_precision

Parameter opsional yang menentukan jumlah digit fraksional yang diizinkan dalam interval. Argumen fractional_precision seharusnya hanya ditentukan jika interval Anda berisi SECOND. Misalnya, `SECOND(3)` buat interval yang memungkinkan hanya tiga digit pecahan, seperti 1,234 detik. Jumlah maksimum digit fraksional adalah enam.

Konfigurasi sesi `interval_forbid_composite_literals` menentukan apakah kesalahan dikembalikan ketika interval ditentukan dengan bagian YEAR TO MONTH dan DAY TO SECOND.

Aritmatika interval

Anda dapat menggunakan nilai interval dengan nilai datetime lainnya untuk melakukan operasi aritmatika. Tabel berikut menjelaskan operasi yang tersedia dan jenis data apa yang dihasilkan dari setiap operasi.

> **Note**
>
> Operasi yang dapat menghasilkan keduanya `date` dan `timestamp` hasil melakukannya berdasarkan satuan waktu terkecil yang terlibat dalam persamaan. Misalnya, ketika Anda menambahkan `interval` ke hasilnya adalah `date` jika itu adalah interval TAHUN KE BULAN, dan stempel waktu jika itu adalah interval HARI KE KEDUA. `date`

Operasi di mana operan pertama adalah `interval` menghasilkan hasil sebagai berikut untuk operan kedua yang diberikan:

Operator	Date	Stempel waktu	Interval	Numerik
-	N/A	N/A	Interval	N/A

Operator	Date	Stempel waktu	Interval	Numerik
+	Date	Tanggal/Stempel Waktu	Interval	N/A
*	N/A	N/A	N/A	Interval
/	N/A	N/A	N/A	Interval

Operasi di mana operan pertama adalah `date` menghasilkan hasil sebagai berikut untuk operan kedua yang diberikan:

Operator	Date	Stempel waktu	Interval	Numerik
-	Numerik	Interval	Tanggal/Stempel Waktu	Date
+	N/A	N/A	N/A	N/A

Operasi di mana operan pertama adalah `timestamp` menghasilkan hasil sebagai berikut untuk operan kedua yang diberikan:

Operator	Date	Stempel waktu	Interval	Numerik
-	Numerik	Interval	Stempel waktu	Stempel waktu
+	N/A	N/A	N/A	N/A

Gaya interval

- `postgres`— mengikuti gaya PostgreSQL. Ini adalah opsi default.
- `postgres_verbose`— mengikuti gaya verbose PostgreSQL.
- `sql_standard`— mengikuti gaya literal interval standar SQL.

Perintah berikut menetapkan gaya interval ke`sql_standard`.

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

format keluaran postgres

Berikut ini adalah format output untuk gaya `postgres` interval. Setiap nilai numerik bisa negatif.

```
'<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
```

format keluaran postgres_verbose

sintaks postgres_verbose mirip dengan postgres, tetapi output postgres_verbose juga berisi satuan waktu.

```
'[@] <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
```

format keluaran sql_standard

Nilai interval tahun ke bulan diformat sebagai berikut. Menentukan tanda negatif sebelum interval menunjukkan interval adalah nilai negatif dan berlaku untuk seluruh interval.

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

Interval hari ke nilai kedua diformat sebagai berikut.

```
'[-]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
```

Contoh tipe data interval

Contoh berikut menunjukkan bagaimana menggunakan tipe data INTERVAL dengan tabel.

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

Contoh literal interval

Contoh berikut dijalankan dengan gaya interval diatur ke`postgres`.

Contoh berikut menunjukkan cara membuat INTERVAL literal 1 tahun.

```
select INTERVAL '1' YEAR

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

Jika Anda menentukan string yang dikutip yang melebihi kualifikasi, satuan waktu yang tersisa dipotong dari interval. Dalam contoh berikut, interval 13 bulan menjadi 1 tahun dan 1 bulan, tetapi sisanya 1 bulan ditinggalkan karena kualifikasi YEAR.

```
select INTERVAL '13 months' YEAR

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

Jika Anda menggunakan qualifier yang lebih rendah dari string interval Anda, unit sisa disertakan.

```
select INTERVAL '13 months' MONTH

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

Menentukan presisi dalam interval Anda memotong jumlah digit pecahan ke presisi yang ditentukan.

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

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

Jika Anda tidak menentukan presisi, AWS Clean Rooms gunakan presisi maksimum 6.

```
select INTERVAL '1.23456789' SECOND

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

Contoh berikut menunjukkan cara membuat interval berkisar.

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

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

Kualifikasi mendikte unit yang Anda tentukan. Misalnya, meskipun contoh berikut menggunakan string kutipan yang sama dari '2:2' seperti contoh sebelumnya, AWS Clean Rooms mengakui bahwa ia menggunakan satuan waktu yang berbeda karena qualifier.

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

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

Singkatan dan bentuk jamak dari masing-masing unit juga didukung. Misalnya,, `5s` `5 second`, dan `5 seconds` merupakan interval yang setara. Unit yang didukung adalah tahun, bulan, jam, menit, dan detik.

```
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
```

Contoh literal interval tanpa sintaks qualifier

> **Note**
>
> Contoh berikut menunjukkan menggunakan interval literal tanpa `YEAR TO MONTH` atau `DAY TO SECOND` kualifikasi. Untuk informasi tentang penggunaan literal interval yang direkomendasikan dengan kualifikasi, lihatTipe data interval dan literal.

Gunakan interval literal untuk mengidentifikasi periode waktu tertentu, seperti `12 hours` atau6 `months`. Anda dapat menggunakan literal interval ini dalam kondisi dan perhitungan yang melibatkan ekspresi datetime.

Interval literal dinyatakan sebagai kombinasi kata kunci INTERVAL dengan kuantitas numerik dan bagian tanggal yang didukung, misalnya `INTERVAL '7 days'` atau`INTERVAL '59 minutes'`. Anda dapat menghubungkan beberapa kuantitas dan unit untuk membentuk interval yang lebih tepat, misalnya:`INTERVAL '7 days, 3 hours, 59 minutes'`. Singkatan dan bentuk jamak dari setiap unit juga didukung; misalnya:`5 s`,`5 second`, dan `5 seconds` merupakan interval yang setara.

Jika Anda tidak menentukan bagian tanggal, nilai interval mewakili detik. Anda dapat menentukan nilai kuantitas sebagai pecahan (misalnya:`0.5 days`).

Contoh berikut menunjukkan serangkaian perhitungan dengan nilai interval yang berbeda.

Berikut ini menambahkan 1 detik ke tanggal yang ditentukan.

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

Berikut ini menambahkan 1 menit ke tanggal yang ditentukan.

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

Berikut ini menambahkan 3 jam dan 35 menit ke tanggal yang ditentukan.

```
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)
```

Berikut ini menambahkan 52 minggu ke tanggal yang ditentukan.

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

Berikut ini menambahkan 1 minggu, 1 jam, 1 menit, dan 1 detik ke tanggal yang ditentukan.

```
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)
```

Berikut ini menambahkan 12 jam (setengah hari) ke tanggal yang ditentukan.

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

Berikut ini mengurangi 4 bulan dari 15 Februari 2023 dan hasilnya adalah 15 Oktober 2022.

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

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

Berikut ini mengurangi 4 bulan dari 31 Maret 2023 dan hasilnya adalah 30 November 2022. Perhitungan mempertimbangkan jumlah hari dalam sebulan.

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

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

Jenis Boolean

Gunakan tipe data BOOLEAN untuk menyimpan nilai true dan false dalam kolom single-byte. Tabel berikut menjelaskan tiga kemungkinan status untuk nilai Boolean dan nilai literal yang menghasilkan keadaan itu. Terlepas dari string input, kolom Boolean menyimpan dan mengeluarkan “t” untuk true dan “f” untuk false.

Status	Nilai literal yang valid	Penyimpanan
True	`TRUE 't' 'true' 'y' 'yes' '1'`	1 byte
False	`FALSE 'f' 'false' 'n' 'no' '0'`	1 byte
Tidak Diketahui	`NULL`	1 byte

Anda dapat menggunakan perbandingan IS untuk memeriksa nilai Boolean hanya sebagai predikat dalam klausa WHERE. Anda tidak dapat menggunakan perbandingan IS dengan nilai Boolean dalam daftar SELECT.

Contoh

Anda dapat menggunakan kolom BOOLEAN untuk menyimpan status “Aktif/Tidak Aktif” untuk setiap pelanggan dalam tabel PELANGGAN.

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

Dalam contoh ini, kueri berikut memilih pengguna dari tabel USERS yang menyukai olahraga tetapi tidak menyukai teater:

```
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)
```

Contoh berikut memilih pengguna dari tabel USERS yang tidak diketahui apakah mereka menyukai musik 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)
```

Contoh berikut mengembalikan kesalahan karena menggunakan perbandingan IS dalam daftar SELECT.

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

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

Contoh berikut berhasil karena menggunakan perbandingan yang sama (=) dalam daftar SELECT alih-alih IS perbandingan.

```
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    |
```

Literal Boolean

Aturan berikut adalah untuk bekerja dengan literal Boolean yang didukung oleh AWS Clean Rooms Spark SQL.

Gunakan literal Boolean untuk menentukan nilai Boolean, seperti `TRUE` atau. `FALSE`

Sintaksis

```
TRUE | FALSE
```

Contoh

Contoh berikut menunjukkan kolom dengan nilai tertentu dari`TRUE`.

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

Tipe biner

Gunakan tipe data BINARY untuk menyimpan dan mengelola data biner dengan panjang tetap dan tidak ditafsirkan, menyediakan kemampuan penyimpanan dan perbandingan yang efisien untuk kasus penggunaan tertentu.

Tipe data BINARY menyimpan sejumlah byte tetap, terlepas dari panjang sebenarnya dari data yang disimpan. Panjang maksimum biasanya 255 byte.

BINARY digunakan untuk menyimpan data biner mentah yang tidak ditafsirkan, seperti gambar, dokumen, atau jenis file lainnya. Data disimpan persis seperti yang disediakan, tanpa pengkodean atau interpretasi karakter apa pun. Data biner yang disimpan dalam kolom BINARY dibandingkan dan diurutkan byte-by-byte, berdasarkan nilai biner aktual, daripada aturan pengkodean atau pemeriksaan karakter apa pun.

Contoh query berikut menunjukkan representasi biner dari string`"abc"`. Setiap karakter dalam string diwakili oleh kode ASCII dalam format heksadesimal: “a” adalah 0x61, “b” adalah 0x62, dan “c” adalah 0x63. Ketika digabungkan, nilai heksadesimal ini membentuk representasi biner. `"616263"`

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

Jenis bersarang

AWS Clean Rooms mendukung kueri yang melibatkan data dengan tipe data bersarang, khususnya tipe kolom AWS Glue STRUCT, ARRAY, dan MAP. Hanya aturan analisis kustom yang mendukung tipe data bersarang.

Khususnya, tipe data bersarang tidak sesuai dengan struktur tabel yang kaku dari model data relasional database SQL.

Tipe data bersarang berisi tag yang mereferensikan entitas berbeda dalam data. Mereka dapat berisi nilai-nilai kompleks seperti array, struktur bersarang, dan struktur kompleks lainnya yang terkait dengan format serialisasi, seperti JSON. Tipe data bersarang mendukung hingga 1 MB data untuk bidang atau objek tipe data bersarang individu.

Topik

- Jenis ARRAY

Jenis ARRAY

Gunakan tipe ARRAY untuk mewakili nilai-nilai yang terdiri dari urutan elemen dengan tipe. `elementType`

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

Gunakan `containsNull` untuk menunjukkan apakah elemen dalam tipe ARRAY dapat memiliki `null` nilai.

Jenis MAP

Gunakan tipe MAP untuk mewakili nilai yang terdiri dari satu set pasangan kunci-nilai.

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

`keyType`: tipe data kunci

`valueType`: tipe data nilai

Kunci tidak diizinkan memiliki `null` nilai. Gunakan `valueContainsNull` untuk menunjukkan apakah nilai dari nilai tipe MAP dapat memiliki `null` nilai.

Jenis STRUCT

Gunakan tipe STRUCT untuk mewakili nilai dengan struktur yang dijelaskan oleh urutan StructFields (bidang).

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

StructField(nama, DataType, nullable): Merupakan bidang di. StructType

`dataType`: tipe data bidang

`name`: nama bidang

Gunakan `nullable` untuk menunjukkan apakah nilai bidang ini dapat memiliki `null` nilai.

Contoh tipe data bersarang

Untuk `struct<given:varchar, family:varchar>` tipe, ada dua nama atribut:`given`, dan`family`, masing-masing sesuai dengan `varchar` nilai.

Untuk `array<varchar>` tipe, array ditentukan sebagai daftar`varchar`.

`array<struct<shipdate:timestamp, price:double>>`Tipe mengacu pada daftar elemen dengan `struct<shipdate:timestamp, price:double>` tipe.

Tipe `map` data berperilaku seperti `array` dari`structs`, di mana nama atribut untuk setiap elemen dalam array dilambangkan dengan `key` dan dipetakan ke a. `value`

Example

Misalnya, `map<varchar(20), varchar(20)>` tipe diperlakukan sebagai`array<struct<key:varchar(20), value:varchar(20)>>`, di mana `key` dan `value` merujuk ke atribut peta dalam data yang mendasarinya.

Untuk informasi tentang cara AWS Clean Rooms mengaktifkan navigasi ke dalam array dan struktur, lihatNavigasi.

Untuk informasi tentang cara AWS Clean Rooms mengaktifkan iterasi melalui array dengan menavigasi array menggunakan klausa FROM dari kueri, lihat. Kueri yang tidak bersarang

Ketik kompatibilitas dan konversi

Topik berikut menjelaskan cara kerja aturan konversi tipe dan kompatibilitas tipe data di AWS Clean Rooms Spark SQL.

Topik

- Kompatibilitas
- Kompatibilitas umum dan aturan konversi
- Jenis konversi implisit

Kompatibilitas

Pencocokan tipe data dan pencocokan nilai literal dan konstanta dengan tipe data terjadi selama berbagai operasi database, termasuk yang berikut ini:

- Bahasa manipulasi data (DML/bahasa manipulasi data) operasi pada tabel
- UNION, INTERSECT, dan EXCEPT query
- Ekspresi CASE
- Evaluasi predikat, seperti LIKE dan IN
- Evaluasi fungsi SQL yang melakukan perbandingan atau ekstraksi data
- Perbandingan dengan operator matematika

Hasil operasi ini bergantung pada aturan konversi tipe dan kompatibilitas tipe data. Kompatibilitas menyiratkan bahwa one-to-one pencocokan nilai tertentu dan tipe data tertentu tidak selalu diperlukan. Karena beberapa tipe data kompatibel, konversi implisit, atau paksaan, dimungkinkan. Untuk informasi selengkapnya, lihat Jenis konversi implisit. Ketika tipe data tidak kompatibel, terkadang Anda dapat mengonversi nilai dari satu tipe data ke tipe data lainnya dengan menggunakan fungsi konversi eksplisit.

Kompatibilitas umum dan aturan konversi

Perhatikan aturan kompatibilitas dan konversi berikut:

- Secara umum, tipe data yang termasuk dalam kategori tipe yang sama (seperti tipe data numerik yang berbeda) kompatibel dan dapat dikonversi secara implisit.

  Misalnya, dengan konversi implisit Anda dapat menyisipkan nilai desimal ke dalam kolom integer. Desimal dibulatkan untuk menghasilkan bilangan bulat. Atau Anda dapat mengekstrak nilai numerik, seperti`2008`, dari tanggal dan memasukkan nilai itu ke dalam kolom integer.
- Tipe data numerik memberlakukan kondisi overflow yang terjadi saat Anda mencoba menyisipkan nilai. out-of-range Misalnya, nilai desimal dengan presisi 5 tidak cocok dengan kolom desimal yang didefinisikan dengan presisi 4. Sebuah integer atau seluruh bagian dari desimal tidak pernah terpotong. Namun, bagian pecahan desimal dapat dibulatkan ke atas atau ke bawah, sebagaimana mestinya. Namun, hasil pemeran eksplisit nilai yang dipilih dari tabel tidak dibulatkan.
- Berbagai jenis string karakter kompatibel. String kolom VARCHAR yang berisi data byte tunggal dan string kolom CHAR sebanding dan dapat dikonversi secara implisit. String VARCHAR yang berisi data multibyte tidak sebanding. Selain itu, Anda dapat mengonversi string karakter ke tanggal, waktu, stempel waktu, atau nilai numerik jika string adalah nilai literal yang sesuai. Setiap spasi utama atau belakang diabaikan. Sebaliknya, Anda dapat mengonversi tanggal, waktu, stempel waktu, atau nilai numerik menjadi string karakter dengan panjang tetap atau panjang variabel.

> **Note**
>
> String karakter yang ingin Anda transmisikan ke tipe numerik harus berisi representasi karakter angka. Misalnya, Anda dapat mentransmisikan string `'1.0'` atau `'5.9'` ke nilai desimal, tetapi Anda tidak dapat mentransmisikan string `'ABC'` ke jenis numerik apa pun.

- Jika Anda membandingkan nilai DECIMAL dengan string karakter, AWS Clean Rooms mencoba untuk mengubah string karakter ke nilai DECIMAL. Saat membandingkan semua nilai numerik lainnya dengan string karakter, nilai numerik dikonversi ke string karakter. Untuk menegakkan konversi yang berlawanan (misalnya, mengubah string karakter menjadi bilangan bulat, atau mengubah nilai DECIMAL menjadi string karakter), gunakan fungsi eksplisit, seperti. Fungsi CAST
- Untuk mengonversi nilai DECIMAL atau NUMERIK 64-bit ke presisi yang lebih tinggi, Anda harus menggunakan fungsi konversi eksplisit seperti fungsi CAST atau CONVERT.

Jenis konversi implisit

Ada dua jenis konversi implisit:

- Konversi implisit dalam tugas, seperti menyetel nilai dalam perintah INSERT atau UPDATE
- Konversi implisit dalam ekspresi, seperti melakukan perbandingan dalam klausa WHERE

Tabel berikut mencantumkan tipe data yang dapat dikonversi secara implisit dalam tugas atau ekspresi. Anda juga dapat menggunakan fungsi konversi eksplisit untuk melakukan konversi ini.

Dari tipe	Untuk mengetik
BIGINT	BOOLEAN
	CHAR
	DESIMAL (NUMERIK)
	PRESISI GANDA (FLOAT8)
	INTEGER
	NYATA (FLOAT4)

Dari tipe	Untuk mengetik
	SMALLINT atau SHORT
	VARCHAR
CHAR	VARCHAR
DATE	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DESIMAL (NUMERIK)	BIGINT atau LONG
	CHAR
	PRESISI GANDA (FLOAT8)
	INTEGER INT)
	NYATA (FLOAT4)
	SMALLINT atau SHORT
	VARCHAR
PRESISI GANDA (FLOAT8)	BIGINT atau LONG
	CHAR
	DESIMAL (NUMERIK)
	BILANGAN BULAT (INT)
	NYATA (FLOAT4)
	SMALLINT atau SHORT

Dari tipe	Untuk mengetik
	VARCHAR
BILANGAN BULAT (INT)	BIGINT atau LONG
	BOOLEAN
	CHAR
	DESIMAL (NUMERIK)
	PRESISI GANDA (FLOAT8)
	NYATA (FLOAT4)
	SMALLINT atau SHORT
	VARCHAR
NYATA (FLOAT4)	BIGINT atau LONG
	CHAR
	DESIMAL (NUMERIK)
	BILANGAN BULAT (INT)
	SMALLINT atau SHORT
	VARCHAR
SMALLINT	BIGINT atau LONG
	BOOLEAN
	CHAR
	DESIMAL (NUMERIK)
	PRESISI GANDA (FLOAT8)

Dari tipe	Untuk mengetik
	BILANGAN BULAT (INT)
	NYATA (FLOAT4)
	VARCHAR
TIME	VARCHAR
	JADWAL

> **Note**
>
> Konversi implisit antara DATE, TIME, TIMESTAMP_LTZ, TIMESTAMP_NTZ, atau string karakter menggunakan zona waktu sesi saat ini.
> Tipe data VARBYTE tidak dapat secara implisit dikonversi ke tipe data lainnya. Lihat informasi yang lebih lengkap di Fungsi CAST.

AWS Clean Rooms Perintah Spark SQL

Perintah SQL berikut didukung di AWS Clean Rooms Spark SQL:

Topik

- TABEL CACHE
- Petunjuk
- SELECT

TABEL CACHE

Perintah CACHE TABLE menyimpan data tabel yang ada atau membuat dan menyimpan tabel baru yang berisi hasil kueri.

> **Note**
>
> Data yang di-cache tetap ada untuk seluruh kueri.

Sintaks, argumen, dan beberapa contoh berasal dari Apache Spark SQL Reference.

Sintaksis

Perintah CACHE TABLE mendukung tiga pola sintaks:

> **Kendala kueri kolom keluaran yang tidak diizinkan dan TABEL CACHE**
>
> Kendala kolom keluaran yang tidak diizinkan dalam aturan analisis kustom diberlakukan pada tabel yang di-cache. Tabel yang di-cache tidak dapat mereferensikan kolom keluaran yang tidak diizinkan dalam klausa SELECT. Untuk menggunakan kolom dengan batasan kolom keluaran yang tidak diizinkan di bagian berikutnya dari kueri Anda, ubah tabel yang di-cache menjadi ekspresi tabel umum (CTE).

Dengan AS (tanpa tanda kurung): Membuat dan menyimpan tabel baru berdasarkan hasil kueri.

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

Dengan AS dan tanda kurung: Fungsi mirip dengan sintaks pertama tetapi menggunakan tanda kurung untuk mengelompokkan kueri secara eksplisit.

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

Tanpa AS: Cache tabel yang ada, menggunakan pernyataan SELECT untuk memfilter baris mana yang akan di-cache.

```
CACHE TABLE cache_table_identifier query;
```

Di mana:

- Semua pernyataan harus diakhiri dengan titik koma (;)
- `query`biasanya pernyataan SELECT

- Tanda kurung di sekitar kueri bersifat opsional dengan AS
- Kata kunci AS adalah opsional

Parameter

cache_table_identifier

Nama untuk tabel cache. Dapat menyertakan kualifikasi nama database opsional.

SEBAGAI

Kata kunci yang digunakan saat membuat dan menyimpan tabel baru dari hasil kueri.

query

Pernyataan SELECT atau kueri lain yang mendefinisikan data yang akan di-cache.

Contoh

Dalam contoh berikut, tabel cache tetap ada untuk seluruh kueri. Setelah caching, kueri berikutnya yang referensi *cache_table_identifier* akan dibaca dari versi cache daripada menghitung ulang atau membaca dari. *sourceTable* Ini dapat meningkatkan kinerja kueri untuk data yang sering diakses.

Buat dan cache tabel yang disaring dari hasil kueri

Contoh pertama menunjukkan cara membuat dan cache tabel baru dari hasil query. Perintah ini menggunakan AS kata kunci tanpa tanda kurung di sekitar pernyataan. SELECT Ini menciptakan tabel baru bernama 'cache_table_identifier' yang hanya berisi baris dari 'sourceTable' di mana statusnya 'active'. Ini menjalankan kueri, menyimpan hasil di tabel baru, dan menyimpan isi tabel baru. Asli 'sourceTable' tetap tidak berubah, dan kueri berikutnya harus referensi 'cache_table_identifier' untuk menggunakan data cache.

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

Hasil kueri cache dengan pernyataan SELECT bertanda kurung

Contoh kedua menunjukkan bagaimana untuk cache hasil query sebagai tabel baru dengan nama tertentu (cache_table_identifier), menggunakan tanda kurung di sekitar pernyataan. SELECT

Perintah ini membuat tabel baru bernama 'cache_table_identifier' yang hanya berisi baris dari 'sourceTable' di mana statusnya 'active'. Ini menjalankan kueri, menyimpan hasil di tabel baru, dan menyimpan isi tabel baru. Asli sourceTable 'tetap tidak berubah. Query selanjutnya harus referensi 'cache_table_identifier' untuk menggunakan data cache.

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

Cache tabel yang ada dengan kondisi filter

Contoh ketiga menunjukkan bagaimana untuk cache tabel yang ada menggunakan sintaks yang berbeda. Sintaks ini, yang menghilangkan kata kunci dan tanda kurung AS '', biasanya menyimpan baris yang ditentukan dari tabel yang ada bernama 'cache_table_identifier' daripada membuat tabel baru. SELECTPernyataan bertindak sebagai filter untuk menentukan baris mana yang akan di-cache.

> **Note**
>
> Perilaku yang tepat dari sintaks ini bervariasi di seluruh sistem database. Selalu verifikasi sintaks yang benar untuk AWS layanan spesifik Anda.

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

Petunjuk

Petunjuk untuk analisis SQL memberikan arahan pengoptimalan yang memandu strategi eksekusi kueri AWS Clean Rooms, memungkinkan Anda meningkatkan kinerja kueri dan mengurangi biaya komputasi. Petunjuk menunjukkan bagaimana mesin analitik Spark harus menghasilkan rencana pelaksanaannya.

Sintaksis

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

Petunjuk disematkan dalam kueri SQL menggunakan sintaks gaya komentar dan harus ditempatkan langsung setelah kata kunci SELECT.

Jenis petunjuk yang didukung

AWS Clean Rooms mendukung dua kategori petunjuk: Gabung petunjuk dan petunjuk Partisi.

Topik

- Bergabunglah dengan petunjuk
- Petunjuk partisi

Bergabunglah dengan petunjuk

Petunjuk bergabung menyarankan strategi bergabung untuk eksekusi kueri. Sintaks, argumen, dan beberapa contoh berasal dari Apache Spark SQL Reference untuk informasi lebih lanjut

MENYIARKAN

Menyarankan bahwa AWS Clean Rooms gunakan broadcast join. Sisi gabungan dengan petunjuk akan disiarkan terlepas dari autoBroadcastJoin Threshold. Jika kedua sisi gabungan memiliki petunjuk siaran, yang dengan ukuran lebih kecil (berdasarkan statistik) akan disiarkan.

Alias: BROADCASTJOIN, MAPJOIN

Parameter: Pengidentifikasi tabel (opsional)

Contoh:

```
-- 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

Menyarankan bahwa AWS Clean Rooms gunakan shuffle sort merge join.

Alias: SHUFFLE_MERGE, MERGEJOIN

Parameter: Pengidentifikasi tabel (opsional)

Contoh:

```
-- 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

Menyarankan bahwa AWS Clean Rooms gunakan shuffle hash join. Jika kedua sisi memiliki petunjuk hash acak, pengoptimal kueri memilih sisi yang lebih kecil (berdasarkan statistik) sebagai sisi build.

Parameter: Pengidentifikasi tabel (opsional)

Contoh:

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

SHUFFLE_REPLICATE_NL

Menyarankan bahwa AWS Clean Rooms gunakan gabungan loop shuffle-and-replicate bersarang.

Parameter: Pengidentifikasi tabel (opsional)

Contoh:

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

Petunjuk Pemecahan Masalah di Spark SQL

Tabel berikut menunjukkan skenario umum di mana petunjuk tidak diterapkan di SparkSQL. Untuk informasi tambahan, lihat the section called “Pertimbangan dan batasan”.

Kasus Penggunaan	Contoh Kueri
Referensi tabel tidak ditemukan	`SELECT /*+ BROADCAST(fake_table) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Tabel tidak berpartisipasi dalam operasi bergabung	`SELECT /*+ BROADCAST(s) */ * FROM students s WHERE s.age > 25;`
Referensi tabel dalam subquery bersarang	`SELECT /*+ BROADCAST(s) */ * FROM employees e INNER JOIN (SELECT * FROM students s WHERE s.age > 20) sub ON e.eid = sub.sid;`
Nama kolom bukan referensi tabel	`SELECT /*+ BROADCAST(e.eid) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Petunjuk tanpa parameter yang diperlukan	`SELECT /*+ BROADCAST */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Nama tabel dasar bukan alias tabel	`SELECT /*+ BROADCAST(employees) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`

Petunjuk partisi

Petunjuk partisi mengontrol distribusi data di seluruh node pelaksana. Ketika beberapa petunjuk partisi ditentukan, beberapa node dimasukkan ke dalam rencana logis, tetapi petunjuk paling kiri dipilih oleh pengoptimal.

BERSATU

Mengurangi jumlah partisi ke jumlah partisi yang ditentukan.

Parameter: Nilai numerik (wajib) - harus berupa bilangan bulat positif antara 1 dan 2147483647

Contoh:

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

PARTISI ULANG

Repartisi data ke jumlah partisi yang ditentukan menggunakan ekspresi partisi yang ditentukan. Menggunakan distribusi round-robin.

Parameter:

- Nilai numerik (opsional) - jumlah partisi; Harus berupa bilangan bulat positif antara 1 dan 2147483647
- Pengidentifikasi kolom (opsional) - kolom untuk dipartisi oleh; Kolom ini harus ada dalam skema input.
- Jika keduanya ditentukan, nilai numerik harus didahulukan

Contoh:

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

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

-- Repartition to 8 partitions by department
```

```
SELECT /*+ REPARTITION(8, department) */ *
FROM employees;

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

REPARTITION_BY_RANGE

Repartisi data ke jumlah partisi yang ditentukan menggunakan partisi rentang pada kolom yang ditentukan.

Parameter:

- Nilai numerik (opsional) - jumlah partisi; Harus berupa bilangan bulat positif antara 1 dan 2147483647
- Pengidentifikasi kolom (opsional) - kolom untuk dipartisi oleh; Kolom ini harus ada dalam skema input.
- Jika keduanya ditentukan, nilai numerik harus didahulukan

Contoh:

```
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;
```

MENYEIMBANGKAN KEMBALI

Menyeimbangkan kembali partisi keluaran hasil kueri sehingga setiap partisi berukuran wajar (tidak terlalu kecil dan tidak terlalu besar). Ini adalah operasi upaya terbaik: jika ada kemiringan, AWS

Clean Rooms akan membagi partisi miring untuk membuatnya tidak terlalu besar. Petunjuk ini berguna ketika Anda perlu menulis hasil kueri ke tabel untuk menghindari file yang terlalu kecil atau terlalu besar.

Parameter:

- Nilai numerik (opsional) - jumlah partisi; Harus berupa bilangan bulat positif antara 1 dan 2147483647
- Pengidentifikasi kolom (opsional) - kolom harus muncul di daftar keluaran SELECT
- Jika keduanya ditentukan, nilai numerik harus didahulukan

Contoh:

```
-- 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;
```

Menggabungkan beberapa petunjuk

Anda dapat menentukan beberapa petunjuk dalam satu kueri dengan memisahkannya dengan koma:

```
-- 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;
```

Pertimbangan dan batasan

- Petunjuk adalah saran pengoptimalan, bukan perintah. Pengoptimal kueri dapat mengabaikan petunjuk berdasarkan batasan sumber daya atau kondisi eksekusi.
- Petunjuk disematkan langsung dalam string kueri SQL untuk keduanya dan. CreateAnalysisTemplate StartProtectedQuery APIs
- Petunjuk harus ditempatkan langsung setelah kata kunci SELECT.
- Parameter bernama tidak didukung dengan petunjuk dan akan memberikan pengecualian.
- Nama kolom di REPARTITION amd REPARTITION_BY_RANGE petunjuk harus ada dalam skema input.
- Nama kolom dalam petunjuk REBALANCE harus muncul di daftar keluaran SELECT.
- Parameter numerik harus bilangan bulat positif antara 1 dan 2147483647. Notasi ilmiah seperti 1e1 tidak didukung
- Petunjuk tidak didukung dalam kueri SQL Privasi Diferensial.
- Petunjuk untuk kueri SQL tidak didukung dalam pekerjaan. PySpark Untuk memberikan arahan untuk rencana eksekusi dalam PySpark pekerjaan, gunakan API bingkai data. Lihat Apache Spark DataFrame API Docs untuk informasi selengkapnya.

SELECT

Perintah SELECT mengembalikan baris dari tabel dan fungsi yang ditentukan pengguna.

Perintah SELECT SQL, klausa, dan operator set berikut didukung di AWS Clean Rooms Spark SQL:

Topik

- Klausa HAVING
- Tetapkan operator
- Klausa ORDER BY
- Contoh subquery
- Subquery berkorelasi

Sintaks, argumen, dan beberapa contoh berasal dari Apache Spark SQL Reference.

SELECT list

SELECT listNama-nama kolom, fungsi, dan ekspresi yang Anda ingin kueri untuk kembali. Daftar ini mewakili output kueri.

Sintaks

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

Parameter

DISTINCT

Opsi yang menghilangkan baris duplikat dari set hasil, berdasarkan nilai yang cocok dalam satu atau beberapa kolom.

expression

Ekspresi yang terbentuk dari satu atau lebih kolom yang ada di tabel yang direferensikan oleh kueri. Ekspresi dapat berisi fungsi SQL. Contoh:

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

AS column_alias

Nama sementara untuk kolom yang digunakan dalam set hasil akhir. ASKata kunci adalah opsional. Contoh:

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

Jika Anda tidak menentukan alias untuk ekspresi yang bukan nama kolom sederhana, set hasil akan menerapkan nama default ke kolom tersebut.

> **Note**
>
> Alias dikenali tepat setelah didefinisikan dalam daftar target. Anda tidak dapat menggunakan alias dalam ekspresi lain yang ditentukan setelahnya dalam daftar target yang sama.

DENGAN klausa

Klausa WITH adalah klausa opsional yang mendahului daftar SELECT dalam kueri. Klausa WITH mendefinisikan satu atau lebih common_table_expressions. Setiap ekspresi tabel umum (CTE) mendefinisikan tabel sementara, yang mirip dengan definisi tampilan. Anda dapat mereferensikan tabel sementara ini di klausa FROM. Mereka hanya digunakan saat kueri milik mereka berjalan. Setiap CTE dalam klausa WITH menentukan nama tabel, daftar opsional nama kolom, dan ekspresi kueri yang mengevaluasi tabel (pernyataan SELECT).

Dengan subquery klausa adalah cara yang efisien untuk mendefinisikan tabel yang dapat digunakan selama eksekusi query tunggal. Dalam semua kasus, hasil yang sama dapat dicapai dengan menggunakan subquery di bagian utama pernyataan SELECT, tetapi dengan subquery klausa mungkin lebih mudah untuk ditulis dan dibaca. Jika memungkinkan, subkueri klausa WITH yang direferensikan beberapa kali dioptimalkan sebagai subexpressions umum; yaitu, dimungkinkan untuk mengevaluasi subquery WITH sekali dan menggunakan kembali hasilnya. (Perhatikan bahwa subexpressions umum tidak terbatas pada yang didefinisikan dalam klausa WITH.)

Sintaksis

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

dimana common_table_expression bisa non-rekursif. Berikut ini adalah bentuk non-rekursif:

```
CTE_table_name AS ( query )
```

Parameter

common_table_expression

Mendefinisikan tabel sementara yang dapat Anda referensikan di Klausa FROM dan hanya digunakan selama eksekusi kueri yang dimilikinya.

CTE_TABLE_NAME

Nama unik untuk tabel sementara yang mendefinisikan hasil subquery klausa WITH. Anda tidak dapat menggunakan nama duplikat dalam satu klausa WITH. Setiap subquery harus diberi nama tabel yang dapat direferensikan di. Klausa FROM

query

Setiap kueri SELECT yang AWS Clean Rooms mendukung. Lihat SELECT.

Catatan penggunaan

Anda dapat menggunakan klausa WITH dalam pernyataan SQL berikut:

- PILIH, DENGAN, UNION, UNION ALL, INTERSECT, INTERSECT ALL, KECUALI, atau KECUALI SEMUA

Jika klausa FROM dari kueri yang berisi klausa WITH tidak mereferensikan salah satu tabel yang ditentukan oleh klausa WITH, klausa WITH diabaikan dan kueri berjalan seperti biasa.

Sebuah tabel yang didefinisikan oleh subquery klausa WITH dapat direferensikan hanya dalam lingkup kueri SELECT bahwa klausa WITH dimulai. Misalnya, Anda dapat mereferensikan tabel tersebut dalam klausa FROM dari subquery dalam daftar SELECT, klausa WHERE, atau HAVING. Anda tidak dapat menggunakan klausa WITH dalam subquery dan mereferensikan tabelnya di klausa FROM dari kueri utama atau subquery lainnya. Pola kueri ini menghasilkan pesan kesalahan formulir `relation table_name doesn't exist` untuk tabel klausa WITH.

Anda tidak dapat menentukan klausa WITH lain di dalam subquery klausa WITH.

Anda tidak dapat meneruskan referensi ke tabel yang ditentukan oleh subkueri klausa WITH. Misalnya, query berikut mengembalikan kesalahan karena referensi forward ke tabel W2 dalam definisi tabel W1:

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

Contoh

Contoh berikut menunjukkan kasus yang paling sederhana dari query yang berisi klausa WITH. Query WITH bernama VENUECOPY memilih semua baris dari tabel VENUE. Kueri utama pada

gilirannya memilih semua baris dari VENUECOPY. Tabel VENUECOPY hanya ada selama durasi kueri ini.

```
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)
```

Contoh berikut menunjukkan klausa WITH yang menghasilkan dua tabel, bernama VENUE_SALES dan TOP_VENUES. Tabel WITH query kedua memilih dari yang pertama. Pada gilirannya, klausa WHERE dari blok kueri utama berisi subquery yang membatasi tabel 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)
```

Dua contoh berikut menunjukkan aturan untuk ruang lingkup referensi tabel berdasarkan subquery klausa WITH. Kueri pertama berjalan, tetapi yang kedua gagal dengan kesalahan yang diharapkan. Kueri pertama memiliki subquery klausa WITH di dalam daftar SELECT dari kueri utama. Tabel yang ditentukan oleh klausa WITH (HOLIDAYS) direferensikan dalam klausa FROM subquery dalam daftar 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)
```

Kueri kedua gagal karena mencoba mereferensikan tabel HOLIDAYS di kueri utama serta di subquery daftar SELECT. Referensi kueri utama berada di luar cakupan.

```
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
```

Klausa FROM

Klausa FROM dalam kueri mencantumkan referensi tabel (tabel, tampilan, dan subkueri) tempat data dipilih. Jika beberapa referensi tabel terdaftar, tabel harus digabungkan, menggunakan sintaks yang sesuai baik dalam klausa FROM atau klausa WHERE. Jika tidak ada kriteria gabungan yang ditentukan, sistem memproses kueri sebagai cross-join (produk Cartesian).

Topik

- Sintaksis
- Parameter
- Catatan penggunaan

Sintaksis

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

di mana table_reference adalah salah satu dari berikut ini:

```
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
```

Parameter

dengan_subquery_table_name

Sebuah tabel didefinisikan oleh subquery di. DENGAN klausa

table_name

Nama tabel atau tampilan.

alias

Nama alternatif sementara untuk tabel atau tampilan. Alias harus disediakan untuk tabel yang berasal dari subquery. Dalam referensi tabel lainnya, alias bersifat opsional. Kata AS kunci selalu opsional. Alias tabel menyediakan pintasan yang nyaman untuk mengidentifikasi tabel di bagian lain dari kueri, seperti klausa WHERE.

Contoh:

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

Jika Anda mendefinisikan alias tabel didefinisikan, maka alias harus digunakan untuk referensi tabel dalam query.

Misalnya, jika kueri adalah`SELECT "tbl"."col" FROM "tbl" AS "t"`, kueri akan gagal karena nama tabel pada dasarnya diganti sekarang. Kueri yang valid dalam kasus ini adalah`SELECT "t"."col" FROM "tbl" AS "t"`.

column_alias

Nama alternatif sementara untuk kolom dalam tabel atau tampilan.

subkueri

Ekspresi kueri yang mengevaluasi ke tabel. Tabel hanya ada selama durasi kueri dan biasanya diberi nama atau alias. Namun, alias tidak diperlukan. Anda juga dapat menentukan nama kolom untuk tabel yang berasal dari subquery. Penamaan alias kolom penting saat Anda ingin menggabungkan hasil subkueri ke tabel lain dan saat Anda ingin memilih atau membatasi kolom tersebut di tempat lain dalam kueri.

Subquery mungkin berisi klausa ORDER BY, tetapi klausa ini mungkin tidak berpengaruh jika klausa LIMIT atau OFFSET tidak juga ditentukan.

ALAMI

Mendefinisikan gabungan yang secara otomatis menggunakan semua pasangan kolom bernama identik dalam dua tabel sebagai kolom bergabung. Tidak diperlukan kondisi gabungan eksplisit. Misalnya, jika tabel CATEGORY dan EVENT keduanya memiliki kolom bernama CATID, gabungan alami dari tabel tersebut adalah gabungan di atas kolom CATID mereka.

> **Note**
>
> Jika gabungan NATURAL ditentukan tetapi tidak ada pasangan kolom bernama identik yang ada di tabel yang akan digabungkan, kueri default ke cross-join.

join_type

Tentukan salah satu jenis join berikut:

- [BATIN] BERGABUNG
- KIRI [LUAR] BERGABUNG
- KANAN [LUAR] BERGABUNG
- PENUH [LUAR] BERGABUNG
- CROSS JOIN

Cross-join adalah gabungan yang tidak memenuhi syarat; mereka mengembalikan produk Cartesian dari dua tabel.

Gabungan dalam dan luar adalah gabungan yang memenuhi syarat. Mereka memenuhi syarat baik secara implisit (dalam gabungan alami); dengan sintaks ON atau USING dalam klausa FROM; atau dengan kondisi klausa WHERE.

Gabungan bagian dalam mengembalikan baris yang cocok saja, berdasarkan kondisi gabungan atau daftar kolom yang bergabung. Gabungan luar mengembalikan semua baris yang akan dikembalikan oleh gabungan dalam yang setara ditambah baris yang tidak cocok dari tabel “kiri”, tabel “kanan”, atau kedua tabel. Tabel kiri adalah tabel yang terdaftar pertama, dan tabel kanan adalah tabel kedua yang terdaftar. Baris yang tidak cocok berisi nilai NULL untuk mengisi celah di kolom output.

PADA join_condition

Jenis spesifikasi gabungan di mana kolom bergabung dinyatakan sebagai kondisi yang mengikuti kata kunci ON. Contoh:

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

MENGGUNAKAN (join_column [,...])

Jenis spesifikasi gabungan di mana kolom bergabung tercantum dalam tanda kurung. Jika beberapa kolom bergabung ditentukan, mereka dibatasi oleh koma. Kata kunci USING harus mendahului daftar. Contoh:

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

Catatan penggunaan

Kolom yang bergabung harus memiliki tipe data yang sebanding.

Gabungan ALAMI atau MENGGUNAKAN hanya mempertahankan satu dari setiap pasangan kolom penggabungan dalam kumpulan hasil perantara.

Gabungan dengan sintaks ON mempertahankan kedua kolom yang bergabung dalam set hasil perantara.

Lihat juga DENGAN klausa.

Klausa JOIN

Klausa SQL JOIN digunakan untuk menggabungkan data dari dua atau lebih tabel berdasarkan bidang umum. Hasilnya mungkin atau mungkin tidak berubah tergantung pada metode gabungan yang ditentukan. Gabungan luar kiri dan kanan mempertahankan nilai dari salah satu tabel yang digabungkan ketika tidak ada kecocokan yang ditemukan di tabel lainnya.

Kombinasi tipe JOIN dan kondisi gabungan menentukan baris mana yang termasuk dalam set hasil akhir. Klausa SELECT dan WHERE kemudian mengontrol kolom mana yang dikembalikan dan bagaimana baris disaring. Memahami berbagai jenis JOIN dan cara menggunakannya secara efektif adalah keterampilan penting dalam SQL, karena memungkinkan Anda untuk menggabungkan data dari beberapa tabel dengan cara yang fleksibel dan kuat.

Sintaks

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

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

Parameter

PILIH kolom1, kolom2,..., kolomN

Kolom yang ingin Anda sertakan dalam set hasil. Anda dapat memilih kolom dari salah satu atau kedua tabel yang terlibat dalam JOIN.

DARI tabel1

Tabel pertama (kiri) dalam operasi JOIN.

[BERGABUNG | BERGABUNG DALAM | KIRI [LUAR] BERGABUNG | KANAN [LUAR] BERGABUNG | LENGKAP [LUAR] BERGABUNG] tabel2:

Jenis JOIN yang akan dilakukan. JOIN atau INNER JOIN hanya mengembalikan baris dengan nilai yang cocok di kedua tabel.

LEFT [OUTER] JOIN mengembalikan semua baris dari tabel kiri, dengan baris yang cocok dari tabel kanan.

RIGHT [OUTER] JOIN mengembalikan semua baris dari tabel kanan, dengan baris yang cocok dari tabel kiri.

FULL [OUTER] JOIN mengembalikan semua baris dari kedua tabel, terlepas dari apakah ada kecocokan atau tidak.

CROSS JOIN menciptakan produk Cartesian dari baris dari dua tabel.

ON table1.column = table2.column

Kondisi bergabung, yang menentukan bagaimana baris dalam dua tabel dicocokkan. Kondisi bergabung dapat didasarkan pada satu atau lebih kolom.

Kondisi DIMANA:

Klausa opsional yang dapat digunakan untuk memfilter hasil yang ditetapkan lebih lanjut, berdasarkan kondisi tertentu.

Contoh

Contoh berikut adalah gabungan antara dua tabel dengan klausa USING. Dalam hal ini, kolom listid dan eventid digunakan sebagai kolom gabungan. Hasilnya dibatasi hingga lima baris.

```
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
```

Bergabunglah dengan tipe

BATIN

Ini adalah tipe gabungan default. Mengembalikan baris yang memiliki nilai yang cocok di kedua referensi tabel.

INNER JOIN adalah jenis gabungan yang paling umum digunakan dalam SQL. Ini adalah cara yang ampuh untuk menggabungkan data dari beberapa tabel berdasarkan kolom umum atau kumpulan kolom.

Sintaksis:

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

Kueri berikut akan mengembalikan semua baris di mana ada nilai customer_id yang cocok antara pelanggan dan tabel pesanan. Set hasil akan berisi kolom customer_id, name, order_id, dan 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;
```

Kueri berikut adalah gabungan dalam (tanpa kata kunci JOIN) antara tabel LISTING dan tabel PENJUALAN, di mana LISTID dari tabel LISTING adalah antara 1 dan 5. Kueri ini cocok dengan nilai kolom LISTID dalam tabel LISTING (tabel kiri) dan tabel PENJUALAN (tabel kanan). Hasilnya menunjukkan bahwa LISTID 1, 4, dan 5 sesuai dengan kriteria.

```
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
```

Contoh berikut adalah gabungan batin dengan klausa ON. Dalam hal ini, baris NULL tidak dikembalikan.

```
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
```

Kueri berikut adalah gabungan batin dari dua subquery dalam klausa FROM. Kueri menemukan jumlah tiket yang terjual dan tidak terjual untuk berbagai kategori acara (konser dan pertunjukan). Subquery klausa FROM adalah subquery tabel; mereka dapat mengembalikan beberapa kolom dan baris.

```
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
```

KIRI [LUAR]

Mengembalikan semua nilai dari referensi tabel kiri dan nilai cocok dari referensi tabel kanan, atau menambahkan NULL jika tidak ada kecocokan. Ini juga disebut sebagai gabungan luar kiri.

Ia mengembalikan semua baris dari tabel kiri (pertama), dan baris yang cocok dari tabel kanan (kedua). Jika tidak ada kecocokan di tabel kanan, set hasil akan berisi nilai NULL untuk kolom dari tabel kanan. Kata kunci OUTER dapat dihilangkan, dan gabungan dapat ditulis hanya sebagai LEFT JOIN. Kebalikan dari LEFT OUTER JOIN adalah RIGHT OUTER JOIN, yang mengembalikan semua baris dari tabel kanan dan baris yang cocok dari tabel kiri.

Sintaksis:

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

Kueri berikut akan mengembalikan semua baris dari tabel pelanggan, bersama dengan baris yang cocok dari tabel pesanan. Jika pelanggan tidak memiliki pesanan, set hasil akan tetap menyertakan informasi pelanggan tersebut, dengan nilai NULL untuk kolom order_id dan 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;
```

Kueri berikut adalah gabungan luar kiri. Gabungan luar kiri dan kanan mempertahankan nilai dari salah satu tabel yang digabungkan ketika tidak ada kecocokan yang ditemukan di tabel lainnya. Tabel kiri dan kanan adalah tabel pertama dan kedua yang tercantum dalam sintaks. Nilai NULL digunakan untuk mengisi “celah” di set hasil. Kueri ini cocok dengan nilai kolom LISTID dalam tabel LISTING (tabel kiri) dan tabel PENJUALAN (tabel kanan). Hasilnya menunjukkan bahwa LISTIDs 2 dan 3 tidak menghasilkan penjualan apa pun.

```
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
```

KANAN [LUAR]

Mengembalikan semua nilai dari referensi tabel kanan dan nilai cocok dari referensi tabel kiri, atau menambahkan NULL jika tidak ada kecocokan. Ini juga disebut sebagai gabungan luar kanan.

Ia mengembalikan semua baris dari tabel kanan (kedua), dan baris yang cocok dari tabel kiri (pertama). Jika tidak ada kecocokan di tabel kiri, set hasil akan berisi nilai NULL untuk kolom dari tabel kiri. Kata kunci OUTER dapat dihilangkan, dan gabungan dapat ditulis hanya sebagai RIGHT JOIN. Kebalikan dari RIGHT OUTER JOIN adalah LEFT OUTER JOIN, yang mengembalikan semua baris dari tabel kiri dan baris yang cocok dari tabel kanan.

Sintaksis:

```
SELECT column1, column2, ..., columnn
FROM table1
RIGHT [OUTER] JOIN table2
```

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

Kueri berikut akan mengembalikan semua baris dari tabel pelanggan, bersama dengan baris yang cocok dari tabel pesanan. Jika pelanggan tidak memiliki pesanan, set hasil akan tetap menyertakan informasi pelanggan tersebut, dengan nilai NULL untuk kolom order_id dan 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;
```

Kueri berikut adalah gabungan luar kanan. Kueri ini cocok dengan nilai kolom LISTID dalam tabel LISTING (tabel kiri) dan tabel PENJUALAN (tabel kanan). Hasilnya menunjukkan bahwa LISTIDs 1, 4, dan 5 sesuai dengan kriteria.

```
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
```

PENUH [LUAR]

Mengembalikan semua nilai dari kedua hubungan, menambahkan nilai NULL di sisi yang tidak memiliki kecocokan. Ini juga disebut sebagai gabungan luar penuh.

Ia mengembalikan semua baris dari kedua tabel kiri dan kanan, terlepas dari apakah ada kecocokan atau tidak. Jika tidak ada kecocokan, set hasil akan berisi nilai NULL untuk kolom dari tabel yang tidak memiliki baris yang cocok. Kata kunci OUTER dapat dihilangkan, dan gabungan dapat ditulis hanya sebagai FULL JOIN. FULL OUTER JOIN lebih jarang digunakan daripada LEFT OUTER JOIN atau RIGHT OUTER JOIN, tetapi dapat berguna dalam skenario tertentu di mana Anda perlu melihat semua data dari kedua tabel, bahkan jika tidak ada kecocokan.

Sintaksis:

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

Kueri berikut akan mengembalikan semua baris dari tabel pelanggan dan pesanan. Jika pelanggan tidak memiliki pesanan, set hasil akan tetap menyertakan informasi pelanggan tersebut, dengan nilai NULL untuk kolom order_id dan order_date. Jika pesanan tidak memiliki pelanggan terkait, kumpulan hasil akan menyertakan urutan itu, dengan nilai NULL untuk customer_id dan kolom nama.

```
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;
```

Kueri berikut adalah gabungan penuh. Gabungan penuh mempertahankan nilai dari tabel yang digabungkan ketika tidak ada kecocokan yang ditemukan di tabel lainnya. Tabel kiri dan kanan adalah tabel pertama dan kedua yang tercantum dalam sintaks. Nilai NULL digunakan untuk mengisi “celah” di set hasil. Kueri ini cocok dengan nilai kolom LISTID dalam tabel LISTING (tabel kiri) dan tabel PENJUALAN (tabel kanan). Hasilnya menunjukkan bahwa LISTIDs 2 dan 3 tidak menghasilkan penjualan apa pun.

```
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
```

Kueri berikut adalah gabungan penuh. Kueri ini cocok dengan nilai kolom LISTID dalam tabel LISTING (tabel kiri) dan tabel PENJUALAN (tabel kanan). Hanya baris yang tidak menghasilkan penjualan apa pun (LISTIDs 2 dan 3) yang ada di hasil.

```
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
```

[KIRI] SEMI

Mengembalikan nilai dari sisi kiri referensi tabel yang memiliki kecocokan dengan kanan. Ini juga disebut sebagai semi join kiri.

Ia mengembalikan hanya baris dari tabel kiri (pertama) yang memiliki baris yang cocok di kanan (kedua) tabel. Itu tidak mengembalikan kolom apa pun dari tabel kanan - hanya kolom dari tabel kiri. LEFT SEMI JOIN berguna ketika Anda ingin menemukan baris dalam satu tabel yang memiliki kecocokan di tabel lain, tanpa perlu mengembalikan data apa pun dari tabel kedua. LEFT SEMI JOIN adalah alternatif yang lebih efisien untuk menggunakan subquery dengan klausa IN atau EXISTS.

Sintaksis:

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

Kueri berikut akan mengembalikan hanya customer_id dan kolom nama dari tabel pelanggan, untuk pelanggan yang memiliki setidaknya satu pesanan dalam tabel pesanan. Set hasil tidak akan menyertakan kolom apa pun dari tabel pesanan.

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

CROSS JOIN

Mengembalikan produk Cartesian dari dua hubungan. Ini berarti bahwa set hasil akan berisi semua kemungkinan kombinasi baris dari dua tabel, tanpa kondisi atau filter apa pun yang diterapkan.

CROSS JOIN berguna ketika Anda perlu menghasilkan semua kemungkinan kombinasi data dari dua tabel, seperti dalam kasus membuat laporan yang menampilkan semua kemungkinan kombinasi informasi pelanggan dan produk. CROSS JOIN berbeda dengan tipe join lainnya (INNER JOIN, LEFT JOIN, dll.) Karena tidak memiliki kondisi gabungan dalam klausa ON. Kondisi bergabung tidak diperlukan untuk CROSS JOIN.

Sintaksis:

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

Kueri berikut akan mengembalikan kumpulan hasil yang berisi semua kemungkinan kombinasi customer_id, customer_name, product_id, dan product_name dari tabel pelanggan dan produk. Jika tabel pelanggan memiliki 10 baris dan tabel produk memiliki 20 baris, set hasil CROSS JOIN akan berisi 10 x 20 = 200 baris.

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

Kueri berikut adalah gabungan silang atau gabungan Cartesian dari tabel LISTING dan tabel PENJUALAN dengan predikat untuk membatasi hasil. Kueri ini cocok dengan nilai kolom LISTID dalam tabel PENJUALAN dan tabel LISTING untuk LISTIDs 1, 2, 3, 4, dan 5 di kedua tabel. Hasilnya menunjukkan bahwa 20 baris cocok dengan kriteria.

```
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 BERGABUNG

Mengembalikan nilai-nilai dari referensi tabel kiri yang tidak cocok dengan referensi tabel kanan. Ini juga disebut sebagai anti join kiri.

ANTI JOIN adalah operasi yang berguna ketika Anda ingin menemukan baris dalam satu tabel yang tidak memiliki kecocokan di tabel lain.

Sintaksis:

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

Kueri berikut akan mengembalikan semua pelanggan yang belum melakukan pemesanan.

```
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;
```

ALAMI

Menentukan bahwa baris dari dua hubungan secara implisit akan dicocokkan pada kesetaraan untuk semua kolom dengan nama yang cocok.

Secara otomatis mencocokkan kolom dengan nama dan tipe data yang sama antara dua tabel. Itu tidak mengharuskan Anda untuk secara eksplisit menentukan kondisi gabungan di klausa ON. Ini menggabungkan semua kolom yang cocok antara dua tabel ke dalam set hasil.

NATURAL JOIN adalah singkatan yang nyaman ketika tabel yang Anda gabungkan memiliki kolom dengan nama dan tipe data yang sama. Namun, umumnya disarankan untuk menggunakan INNER JOIN yang lebih eksplisit... Sintaks ON untuk membuat kondisi gabungan lebih eksplisit dan lebih mudah dipahami.

Sintaksis:

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

Contoh berikut adalah gabungan alami antara dua tabel, `employees` dan`departments`, dengan kolom berikut:

- `employees`tabel:`employee_id`,`first_name`,`last_name`, `department_id`
- `departments`meja:`department_id`, `department_name`

Kueri berikut akan mengembalikan kumpulan hasil yang mencakup nama depan, nama belakang, dan nama departemen untuk semua baris yang cocok antara dua tabel, berdasarkan `department_id` kolom.

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

Contoh berikut adalah gabungan alami antara dua tabel. Dalam hal ini, kolom listid, sellerid, eventid, dan dateid memiliki nama dan tipe data yang identik di kedua tabel dan digunakan sebagai kolom gabungan. Hasilnya dibatasi hingga lima baris.

```
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
```

Klausa WHERE

Klausa WHERE berisi kondisi yang menggabungkan tabel atau menerapkan predikat ke kolom dalam tabel. Tabel dapat bergabung dalam dengan menggunakan sintaks yang sesuai baik dalam klausa WHERE atau klausa FROM. Kriteria gabungan luar harus ditentukan dalam klausa FROM.

Sintaksis

```
[ WHERE condition ]
```

ketentuan

Setiap kondisi pencarian dengan hasil Boolean, seperti kondisi gabungan atau predikat pada kolom tabel. Contoh berikut adalah ketentuan gabungan yang valid:

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

Contoh berikut adalah kondisi yang valid pada kolom dalam tabel:

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

Kondisi bisa sederhana atau kompleks; untuk kondisi kompleks, Anda dapat menggunakan tanda kurung untuk mengisolasi unit logis. Dalam contoh berikut, kondisi bergabung diapit oleh tanda kurung.

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

Catatan penggunaan

Anda dapat menggunakan alias dalam klausa WHERE untuk referensi ekspresi daftar pilih.

Anda tidak dapat membatasi hasil fungsi agregat dalam klausa WHERE; gunakan klausa HAVING untuk tujuan ini.

Kolom yang dibatasi dalam klausa WHERE harus berasal dari referensi tabel dalam klausa FROM.

Contoh

Kueri berikut menggunakan kombinasi batasan klausa WHERE yang berbeda, termasuk kondisi gabungan untuk tabel PENJUALAN dan EVENT, predikat pada kolom EVENTNAME, dan dua predikat pada kolom 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)
```

Klausa VALUES

Klausa VALUES digunakan untuk menyediakan satu set nilai baris langsung dalam query, tanpa perlu referensi tabel.

Klausa VALUES dapat digunakan dalam skenario berikut:

- Anda dapat menggunakan klausa VALUES dalam pernyataan INSERT INTO untuk menentukan nilai untuk baris baru yang dimasukkan ke dalam tabel.
- Anda dapat menggunakan klausa VALUES sendiri untuk membuat kumpulan hasil sementara, atau tabel sebaris, tanpa perlu mereferensikan tabel.
- Anda dapat menggabungkan klausa VALUES dengan klausa SQL lainnya, seperti WHERE, ORDER BY, atau LIMIT, untuk memfilter, mengurutkan, atau membatasi baris dalam kumpulan hasil.

Klausul ini sangat berguna ketika Anda perlu menyisipkan, menanyakan, atau memanipulasi sekumpulan kecil data secara langsung dalam pernyataan SQL Anda, tanpa perlu membuat atau mereferensikan tabel permanen. Ini memungkinkan Anda untuk menentukan nama kolom dan nilai yang sesuai untuk setiap baris, memberi Anda fleksibilitas untuk membuat set hasil sementara atau menyisipkan data dengan cepat, tanpa overhead mengelola tabel terpisah.

Sintaksis

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

Parameter

ekspresi

Ekspresi yang menentukan kombinasi dari satu atau lebih nilai, operator dan fungsi SQL yang menghasilkan nilai.

table_alias

Alias yang menentukan nama sementara dengan daftar nama kolom opsional.

Contoh

Contoh berikut membuat tabel sebaris, hasil seperti tabel sementara yang diatur dengan dua kolom, dan. `col1` `col2` Baris tunggal dalam set hasil berisi nilai-nilai `"one"` dan`1`, masing-masing. `SELECT`

* `FROM`Bagian dari query hanya mengambil semua kolom dan baris dari set hasil sementara ini. Nama kolom (`col1`dan`col2`) secara otomatis dihasilkan oleh sistem database, karena klausa VALUES tidak secara eksplisit menentukan nama kolom.

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

Jika Anda ingin menentukan nama kolom kustom, Anda dapat melakukannya dengan menggunakan klausa AS setelah klausa VALUES, seperti ini:

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

Ini akan membuat set hasil sementara dengan nama kolom `name` dan`id`, bukan default `col1` dan`col2`.

Klausa GROUP BY

Klausa GROUP BY mengidentifikasi kolom pengelompokan untuk kueri. Kolom pengelompokan harus dideklarasikan saat kueri menghitung agregat dengan fungsi standar seperti SUM, AVG, dan COUNT. Jika fungsi agregat hadir dalam ekspresi SELECT, kolom apa pun dalam ekspresi SELECT yang tidak dalam fungsi agregat harus berada dalam klausa GROUP BY.

Untuk informasi selengkapnya, lihat AWS Clean Rooms Fungsi Spark SQL.

Sintaks

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

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

```
        }
```

Parameter

expr

Daftar kolom atau ekspresi harus cocok dengan daftar ekspresi non-agregat dalam daftar pilih kueri. Misalnya, pertimbangkan kueri sederhana berikut.

```
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)
```

Dalam kueri ini, daftar pilih terdiri dari dua ekspresi agregat. Yang pertama menggunakan fungsi SUM dan yang kedua menggunakan fungsi COUNT. Dua kolom yang tersisa, LISTID dan EVENTID, harus dinyatakan sebagai kolom pengelompokan.

Ekspresi dalam klausa GROUP BY juga dapat mereferensikan daftar pilih dengan menggunakan nomor urut. Misalnya, contoh sebelumnya dapat disingkat sebagai berikut.

```
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

Anda dapat menggunakan ekstensi agregasi ROLLUP untuk melakukan pekerjaan beberapa operasi GROUP BY dalam satu pernyataan. Untuk informasi selengkapnya tentang ekstensi agregasi dan fungsi terkait, lihatEkstensi agregasi.

Ekstensi agregasi

AWS Clean Rooms mendukung ekstensi agregasi untuk melakukan pekerjaan beberapa operasi GROUP BY dalam satu pernyataan.

SET PENGELOMPOKAN

Menghitung satu atau lebih kumpulan pengelompokan dalam satu pernyataan. Kumpulan pengelompokan adalah kumpulan klausa GROUP BY tunggal, satu set kolom 0 atau lebih yang dengannya Anda dapat mengelompokkan kumpulan hasil kueri. GROUP BY GROUPING SETS setara dengan menjalankan query UNION ALL pada satu set hasil yang dikelompokkan berdasarkan kolom yang berbeda. Misalnya, GROUP BY GROUPING SETS ((a), (b)) setara dengan GROUP BY a UNION ALL GROUP BY b.

Contoh berikut mengembalikan biaya produk tabel pesanan dikelompokkan sesuai dengan kategori produk dan jenis produk yang dijual.

```
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

Mengasumsikan hierarki di mana kolom sebelumnya dianggap sebagai orang tua dari kolom berikutnya. ROLLUP mengelompokkan data berdasarkan kolom yang disediakan, mengembalikan baris subtotal tambahan yang mewakili total di semua tingkat kolom pengelompokan, selain baris yang dikelompokkan. Misalnya, Anda dapat menggunakan GROUP BY ROLLUP ((a), (b)) untuk mengembalikan kumpulan hasil yang dikelompokkan terlebih dahulu oleh a, kemudian oleh b sambil mengasumsikan bahwa b adalah ayat dari a. ROLLUP juga mengembalikan baris dengan seluruh hasil yang ditetapkan tanpa pengelompokan kolom.

GROUP BY ROLLUP ((a), (b)) setara dengan GROUP BY GROUPING SETS ((a, b), (a), ()).

Contoh berikut mengembalikan biaya produk tabel pesanan dikelompokkan pertama berdasarkan kategori dan kemudian produk, dengan produk sebagai subdivisi kategori.

```
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)
```

KUBUS

Kelompokkan data berdasarkan kolom yang disediakan, mengembalikan baris subtotal tambahan yang mewakili total di semua tingkat kolom pengelompokan, selain baris yang dikelompokkan. CUBE mengembalikan baris yang sama dengan ROLLUP, sambil menambahkan baris subtotal tambahan untuk setiap kombinasi kolom pengelompokan yang tidak dicakup oleh ROLLUP. Misalnya, Anda dapat menggunakan GROUP BY CUBE ((a), (b)) untuk mengembalikan kumpulan hasil yang dikelompokkan terlebih dahulu oleh a, kemudian oleh b sambil mengasumsikan bahwa b adalah subbagian dari a, lalu oleh b saja. CUBE juga mengembalikan baris dengan seluruh hasil yang ditetapkan tanpa pengelompokan kolom.

GROUP BY CUBE ((a), (b)) setara dengan GROUP BY GROUPING SETS ((a, b), (a), (b), ()).

Contoh berikut mengembalikan biaya produk tabel pesanan dikelompokkan pertama berdasarkan kategori dan kemudian produk, dengan produk sebagai subdivisi kategori. Berbeda dengan contoh sebelumnya untuk ROLLUP, pernyataan mengembalikan hasil untuk setiap kombinasi kolom pengelompokan.

```
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)
```

Klausa HAVING

Klausa HAVING menerapkan kondisi untuk kumpulan hasil dikelompokkan menengah yang dikembalikan kueri.

Sintaksis

```
[ HAVING condition ]
```

Misalnya, Anda dapat membatasi hasil fungsi SUM:

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

Kondisi HAVING diterapkan setelah semua kondisi klausa WHERE diterapkan dan operasi GROUP BY selesai.

Kondisi itu sendiri mengambil bentuk yang sama dengan kondisi klausa WHERE.

Catatan penggunaan

- Setiap kolom yang direferensikan dalam kondisi klausa HAVING harus berupa kolom pengelompokan atau kolom yang mengacu pada hasil fungsi agregat.
- Dalam klausa HAVING, Anda tidak dapat menentukan:
  - Nomor urut yang mengacu pada item daftar pilih. Hanya klausa GROUP BY dan ORDER BY yang menerima nomor urut.

Contoh

Kueri berikut menghitung total penjualan tiket untuk semua acara berdasarkan nama, kemudian menghilangkan peristiwa di mana total penjualan kurang dari $800.000. Kondisi HAVING diterapkan pada hasil fungsi agregat dalam daftar pilih:`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)
```

Query berikut menghitung set hasil yang sama. Namun, dalam kasus ini, kondisi HAVING diterapkan ke agregat yang tidak ditentukan dalam daftar pilih:`sum(qtysold)`. Acara yang tidak menjual lebih dari 2.000 tiket dihilangkan dari hasil akhir.

```
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)
```

Tetapkan operator

Operator set digunakan untuk membandingkan dan menggabungkan hasil dari dua ekspresi kueri terpisah.

AWS Clean Rooms Spark SQL mendukung operator set berikut yang tercantum dalam tabel berikut.

Tetapkan operator
BERPOTONGAN
BERPOTONGAN SEMUA
KECUALI
KECUALI SEMUA
UNION
SERIKAT SEMUA

Misalnya, jika Anda ingin mengetahui pengguna situs web mana yang merupakan pembeli dan penjual tetapi nama pengguna mereka disimpan dalam kolom atau tabel terpisah, Anda dapat menemukan persimpangan kedua jenis pengguna ini. Jika Anda ingin tahu pengguna situs web mana yang merupakan pembeli tetapi bukan penjual, Anda dapat menggunakan operator EXCEPT untuk menemukan perbedaan antara dua daftar pengguna. Jika Anda ingin membuat daftar semua pengguna, apa pun perannya, Anda dapat menggunakan operator UNION.

> **Note**
>
> Klausa ORDER BY, LIMIT, SELECT TOP, dan OFFSET tidak dapat digunakan dalam ekspresi kueri yang digabungkan oleh operator set UNION, UNION ALL, INTERSECT, dan EXCEPT.

Topik

Sintaksis

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

Parameter

subkueri1, subkueri2

Ekspresi kueri yang sesuai, dalam bentuk daftar pilihannya, dengan ekspresi kueri kedua yang mengikuti operator UNION, UNION ALL, INTERSECT, INTERSECT ALL, KECUALI, atau KECUALI SEMUA. Kedua ekspresi harus berisi jumlah kolom keluaran yang sama dengan tipe data yang kompatibel; jika tidak, dua set hasil tidak dapat dibandingkan dan digabungkan. Operasi set tidak memungkinkan konversi implisit antara berbagai kategori tipe data. Untuk informasi selengkapnya, lihat Ketik kompatibilitas dan konversi.

Anda dapat membuat kueri yang berisi ekspresi kueri dalam jumlah tak terbatas dan menautkannya dengan operator UNION, INTERSECT, dan EXCEPT dalam kombinasi apa

pun. Misalnya, struktur kueri berikut ini valid, dengan asumsi bahwa tabel T1, T2, dan T3 berisi kumpulan kolom yang kompatibel:

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

SERIKAT [SEMUA | BERBEDA]

Mengatur operasi yang mengembalikan baris dari dua ekspresi query, terlepas dari apakah baris berasal dari satu atau kedua ekspresi.

BERPOTONGAN [SEMUA | BERBEDA]

Mengatur operasi yang mengembalikan baris yang berasal dari dua ekspresi query. Baris yang tidak dikembalikan oleh kedua ekspresi akan dibuang.

KECUALI [SEMUA | BERBEDA]

Mengatur operasi yang mengembalikan baris yang berasal dari salah satu dari dua ekspresi query. Agar memenuhi syarat untuk hasil, baris harus ada di tabel hasil pertama tetapi bukan yang kedua.

KECUALI SEMUA tidak menghapus duplikat dari baris hasil.

MINUS dan EXCEPT adalah sinonim yang tepat.

Urutan evaluasi untuk operator yang ditetapkan

Operator set UNION dan EXCEPLE adalah asosiatif kiri. Jika tanda kurung tidak ditentukan untuk mempengaruhi urutan prioritas, kombinasi dari operator set ini dievaluasi dari kiri ke kanan. Misalnya, dalam kueri berikut, UNION T1 dan T2 dievaluasi terlebih dahulu, kemudian operasi EXCEPT dilakukan pada hasil UNION:

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

Operator INTERSECT lebih diutamakan daripada operator UNION dan EXCEPT ketika kombinasi operator digunakan dalam kueri yang sama. Misalnya, kueri berikut mengevaluasi persimpangan T2 dan T3, lalu menyatukan hasilnya dengan T1:

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

Dengan menambahkan tanda kurung, Anda dapat menerapkan urutan evaluasi yang berbeda. Dalam kasus berikut, hasil penyatuan T1 dan T2 berpotongan dengan T3, dan kueri kemungkinan akan menghasilkan hasil yang berbeda.

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

Catatan penggunaan

- Nama kolom yang dikembalikan dalam hasil kueri operasi set adalah nama kolom (atau alias) dari tabel dalam ekspresi kueri pertama. Karena nama kolom ini berpotensi menyesatkan, karena nilai dalam kolom berasal dari tabel di kedua sisi operator set, Anda mungkin ingin memberikan alias yang berarti untuk kumpulan hasil.
- Ketika kueri operator yang disetel mengembalikan hasil desimal, kolom hasil yang sesuai dipromosikan untuk mengembalikan presisi dan skala yang sama. Misalnya, dalam kueri berikut, di mana T1.REVENUE adalah kolom DECIMAL (10,2) dan T2.REVENUE adalah kolom DECIMAL (8,4), hasil desimal dipromosikan ke DECIMAL (12,4):

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

  Skala ini 4 karena itu adalah skala maksimum dari dua kolom. Ketepatannya adalah 12 karena T1.REVENUE membutuhkan 8 digit di sebelah kiri titik desimal (12 - 4 = 8). Promosi jenis ini memastikan bahwa semua nilai dari kedua sisi UNION sesuai dengan hasilnya. Untuk nilai 64-bit, presisi hasil maksimum adalah 19 dan skala hasil maksimum adalah 18. Untuk nilai 128-bit, presisi hasil maksimum adalah 38 dan skala hasil maksimum adalah 37.

Jika tipe data yang dihasilkan melebihi AWS Clean Rooms presisi dan batas skala, kueri mengembalikan kesalahan.

- Untuk operasi set, dua baris diperlakukan sebagai identik jika, untuk setiap pasangan kolom yang sesuai, dua nilai data sama atau keduanya NULL. Misalnya, jika tabel T1 dan T2 keduanya berisi satu kolom dan satu baris, dan baris itu adalah NULL di kedua tabel, operasi INTERSECT di atas tabel tersebut mengembalikan baris itu.

Contoh kueri UNION

Dalam query UNION berikut, baris dalam tabel PENJUALAN digabungkan dengan baris dalam tabel LISTING. Tiga kolom yang kompatibel dipilih dari setiap tabel; dalam hal ini, kolom yang sesuai memiliki nama dan tipe data yang sama.

```
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
```

Contoh berikut menunjukkan bagaimana Anda dapat menambahkan nilai literal untuk output dari query UNION sehingga Anda dapat melihat ekspresi query yang dihasilkan setiap baris dalam set hasil. Kueri mengidentifikasi baris dari ekspresi kueri pertama sebagai “B” (untuk pembeli) dan baris dari ekspresi kueri kedua sebagai “S” (untuk penjual).

Kueri mengidentifikasi pembeli dan penjual untuk transaksi tiket yang harganya $10.000 atau lebih. Satu-satunya perbedaan antara dua ekspresi kueri di kedua sisi operator UNION adalah kolom bergabung untuk tabel PENJUALAN.

```
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
```

Contoh berikut menggunakan operator UNION ALL karena baris duplikat, jika ditemukan, perlu dipertahankan dalam hasilnya. Untuk rangkaian acara tertentu IDs, kueri mengembalikan 0 atau lebih baris untuk setiap penjualan yang terkait dengan setiap acara, dan 0 atau 1 baris untuk setiap daftar acara tersebut. Acara IDs unik untuk setiap baris dalam tabel LISTING dan EVENT, tetapi mungkin ada beberapa penjualan untuk kombinasi acara dan daftar yang sama IDs di tabel PENJUALAN.

Kolom ketiga dalam set hasil mengidentifikasi sumber baris. Jika berasal dari tabel PENJUALAN, itu ditandai “Ya” di kolom SALESROW. (SALESROW adalah alias untuk SALES.LISTID.) Jika baris berasal dari tabel LISTING, itu ditandai “Tidak” di kolom SALESROW.

Dalam hal ini, set hasil terdiri dari tiga baris penjualan untuk daftar 500, acara 7787. Dengan kata lain, tiga transaksi berbeda terjadi untuk daftar dan kombinasi acara ini. Dua daftar lainnya, 501 dan 502, tidak menghasilkan penjualan apa pun, jadi satu-satunya baris yang dihasilkan kueri untuk daftar ini IDs berasal dari tabel LISTING (SALESROW = 'Tidak').

```
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
```

Jika Anda menjalankan kueri yang sama tanpa kata kunci ALL, hasilnya hanya mempertahankan satu transaksi penjualan.

```
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
```

Contoh UNION ALL query

Contoh berikut menggunakan operator UNION ALL karena baris duplikat, jika ditemukan, perlu dipertahankan dalam hasilnya. Untuk rangkaian acara tertentu IDs, kueri mengembalikan 0 atau lebih baris untuk setiap penjualan yang terkait dengan setiap acara, dan 0 atau 1 baris untuk setiap daftar acara tersebut. Acara IDs unik untuk setiap baris dalam tabel LISTING dan EVENT, tetapi mungkin ada beberapa penjualan untuk kombinasi acara dan daftar yang sama IDs di tabel PENJUALAN.

Kolom ketiga dalam set hasil mengidentifikasi sumber baris. Jika berasal dari tabel PENJUALAN, itu ditandai “Ya” di kolom SALESROW. (SALESROW adalah alias untuk SALES.LISTID.) Jika baris berasal dari tabel LISTING, itu ditandai “Tidak” di kolom SALESROW.

Dalam hal ini, set hasil terdiri dari tiga baris penjualan untuk daftar 500, acara 7787. Dengan kata lain, tiga transaksi berbeda terjadi untuk daftar dan kombinasi acara ini. Dua daftar lainnya, 501 dan 502, tidak menghasilkan penjualan apa pun, jadi satu-satunya baris yang dihasilkan kueri untuk daftar ini IDs berasal dari tabel LISTING (SALESROW = 'Tidak').

```
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
```

Jika Anda menjalankan kueri yang sama tanpa kata kunci ALL, hasilnya hanya mempertahankan satu transaksi penjualan.

```
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
```

Contoh pertanyaan INTERSECT

Bandingkan contoh berikut dengan contoh UNION pertama. Satu-satunya perbedaan antara kedua contoh adalah operator set yang digunakan, tetapi hasilnya sangat berbeda. Hanya satu baris yang sama:

```
235494 |    23875 |    8771
```

Ini adalah satu-satunya baris dalam hasil terbatas dari 5 baris yang ditemukan di kedua tabel.

```
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
```

Pertanyaan berikut menemukan peristiwa (yang tiketnya terjual) yang terjadi di tempat-tempat di New York City dan Los Angeles pada bulan Maret. Perbedaan antara dua ekspresi kueri adalah kendala pada kolom 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
```

Contoh KECUALI kueri

Tabel CATEGORY dalam database berisi 11 baris berikut:

```
 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)
```

Asumsikan bahwa tabel CATEGORY_STAGE (tabel pementasan) berisi satu baris tambahan:

```
 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)
```

Kembalikan perbedaan antara dua tabel. Dengan kata lain, kembalikan baris yang ada di tabel CATEGORY_STAGE tetapi tidak di tabel CATEGORY:

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

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

Kueri setara berikut menggunakan sinonim MINUS.

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

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

Jika Anda membalikkan urutan ekspresi SELECT, kueri tidak mengembalikan baris.

Klausa ORDER BY

Klausa ORDER BY mengurutkan kumpulan hasil kueri.

> **Note**
>
> Ekspresi ORDER BY terluar harus hanya memiliki kolom yang ada di daftar pilih.

Topik

- Sintaksis
- Parameter
- Catatan penggunaan
- Contoh dengan ORDER BY

Sintaksis

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

Parameter

ekspresi

Ekspresi yang mendefinisikan urutan pengurutan hasil query. Ini terdiri dari satu atau lebih kolom dalam daftar pilih. Hasil dikembalikan berdasarkan urutan UTF-8 biner. Anda juga dapat menentukan yang berikut:

- Nomor urut yang mewakili posisi entri daftar pilih (atau posisi kolom dalam tabel jika tidak ada daftar pilih)
- Alias yang menentukan entri daftar pilih

Ketika klausa ORDER BY berisi beberapa ekspresi, kumpulan hasil diurutkan menurut ekspresi pertama, maka ekspresi kedua diterapkan ke baris yang memiliki nilai yang cocok dari ekspresi pertama, dan seterusnya.

ASC | DESC

Opsi yang mendefinisikan urutan pengurutan untuk ekspresi, sebagai berikut:

- ASC: naik (misalnya, rendah ke tinggi untuk nilai numerik dan 'A' ke 'Z' untuk string karakter). Jika tidak ada opsi yang ditentukan, data diurutkan dalam urutan menaik secara default.
- DESC: turun (tinggi ke rendah untuk nilai numerik; 'Z' ke 'A' untuk string).

NULLS PERTAMA | NULLS TERAKHIR

Opsi yang menentukan apakah nilai NULL harus diurutkan terlebih dahulu, sebelum nilai non-null, atau terakhir, setelah nilai non-null. Secara default, nilai NULL diurutkan dan diberi peringkat terakhir dalam urutan ASC, dan diurutkan dan diberi peringkat pertama dalam urutan DESC.

BATASAN nomor | SEMUA

Opsi yang mengontrol jumlah baris yang diurutkan yang dikembalikan kueri. Bilangan LIMIT harus berupa bilangan bulat positif; nilai maksimumnya adalah2147483647.

LIMIT 0 tidak mengembalikan baris. Anda dapat menggunakan sintaks ini untuk tujuan pengujian: untuk memeriksa apakah kueri berjalan (tanpa menampilkan baris apa pun) atau mengembalikan daftar kolom dari tabel. Klausa ORDER BY berlebihan jika Anda menggunakan LIMIT 0 untuk mengembalikan daftar kolom. Defaultnya adalah LIMIT ALL.

OFFSET mulai

Opsi yang menentukan untuk melewati jumlah baris sebelum memulai sebelum mulai mengembalikan baris. Nomor OFFSET harus berupa bilangan bulat positif; nilai maksimumnya adalah. 2147483647 Saat digunakan dengan opsi LIMIT, baris OFFSET dilewati sebelum mulai menghitung baris LIMIT yang dikembalikan. Jika opsi LIMIT tidak digunakan, jumlah baris dalam kumpulan hasil dikurangi dengan jumlah baris yang dilewati. Baris yang dilewati oleh klausa OFFSET masih harus dipindai, jadi mungkin tidak efisien untuk menggunakan nilai OFFSET yang besar.

Catatan penggunaan

Perhatikan perilaku yang diharapkan berikut dengan klausa ORDER BY:

- Nilai NULL dianggap “lebih tinggi” dari semua nilai lainnya. Dengan urutan urutan menaik default, nilai NULL mengurutkan di akhir. Untuk mengubah perilaku ini, gunakan opsi NULLS FIRST.
- Ketika kueri tidak berisi klausa ORDER BY, sistem mengembalikan set hasil tanpa urutan baris yang dapat diprediksi. Kueri yang sama dijalankan dua kali mungkin mengembalikan set hasil dalam urutan yang berbeda.
- Opsi LIMIT dan OFFSET dapat digunakan tanpa klausa ORDER BY; namun, untuk mengembalikan serangkaian baris yang konsisten, gunakan opsi ini bersama dengan ORDER BY.
- Dalam sistem parallel seperti AWS Clean Rooms, ketika ORDER BY tidak menghasilkan urutan yang unik, urutan baris adalah nondeterministik. Artinya, jika ekspresi ORDER BY menghasilkan nilai duplikat, urutan pengembalian baris tersebut mungkin berbeda dari sistem lain atau dari satu proses AWS Clean Rooms ke yang berikutnya.
- AWS Clean Rooms tidak mendukung literal string dalam klausa ORDER BY.

Contoh dengan ORDER BY

Kembalikan semua 11 baris dari tabel KATEGORI, diurutkan berdasarkan kolom kedua, CATGROUP. Untuk hasil yang memiliki nilai CATGROUP yang sama, urutkan nilai kolom CATDESC dengan panjang string karakter. Kemudian urutkan berdasarkan kolom CATID dan 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)
```

Kembalikan kolom yang dipilih dari tabel PENJUALAN, diurutkan berdasarkan nilai QTYSOLD tertinggi. Batasi hasilnya ke 10 baris teratas:

```
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
```

Kembalikan daftar kolom dan tidak ada baris dengan menggunakan sintaks LIMIT 0:

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

Contoh subquery

Contoh berikut menunjukkan cara yang berbeda di mana subquery cocok dengan kueri SELECT. Lihat Contoh contoh lain dari penggunaan subquery.

PILIH daftar subquery

Contoh berikut berisi subquery dalam daftar SELECT. Subquery ini adalah skalar: ia mengembalikan hanya satu kolom dan satu nilai, yang diulang dalam hasil untuk setiap baris yang dikembalikan dari query luar. Kueri membandingkan nilai Q1SALES yang dihitung subquery dengan nilai penjualan untuk dua kuartal lainnya (2 dan 3) pada tahun 2008, seperti yang didefinisikan oleh kueri luar.

```
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)
```

Subquery klausa WHERE

Contoh berikut berisi subquery tabel dalam klausa WHERE. Subquery ini menghasilkan beberapa baris. Dalam hal ini, baris hanya berisi satu kolom, tetapi subquery tabel dapat berisi beberapa kolom dan baris, sama seperti tabel lainnya.

Kueri menemukan 10 penjual teratas dalam hal tiket maksimum yang terjual. Daftar 10 teratas dibatasi oleh subquery, yang menghapus pengguna yang tinggal di kota di mana ada tempat tiket.

Kueri ini dapat ditulis dengan cara yang berbeda; misalnya, subquery dapat ditulis ulang sebagai gabungan dalam kueri utama.

```
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)
```

DENGAN subquery klausa

Lihat DENGAN klausa.

Subquery berkorelasi

Contoh berikut berisi subquery berkorelasi dalam klausa WHERE; subquery semacam ini berisi satu atau lebih korelasi antara kolom dan kolom yang dihasilkan oleh kueri luar. Dalam hal ini, korelasinya adalah`where s.listid=l.listid`. Untuk setiap baris yang dihasilkan kueri luar, subquery dijalankan untuk memenuhi syarat atau mendiskualifikasi baris.

```
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)
```

Pola subquery berkorelasi yang tidak didukung

Perencana kueri menggunakan metode penulisan ulang kueri yang disebut decorrelation subquery untuk mengoptimalkan beberapa pola subquery berkorelasi untuk eksekusi di lingkungan MPP. Beberapa jenis subkueri berkorelasi mengikuti pola yang tidak AWS Clean Rooms dapat mendekorasi dan tidak mendukung. Kueri yang berisi referensi korelasi berikut mengembalikan kesalahan:

- Referensi korelasi yang melewati blok kueri, juga dikenal sebagai “referensi korelasi tingkat lewati.” Misalnya, dalam kueri berikut, blok yang berisi referensi korelasi dan blok yang dilewati dihubungkan oleh predikat NOT EXISTS:

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

  Blok yang dilewati dalam kasus ini adalah subquery terhadap tabel LISTING. Referensi korelasi menghubungkan tabel EVENT dan SALES.
- Referensi korelasi dari subquery yang merupakan bagian dari klausa ON dalam kueri luar:

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

  Klausa ON berisi referensi korelasi dari SALES di subquery ke EVENT di kueri luar.
- Referensi korelasi sensitif nol ke tabel sistem. AWS Clean Rooms Contoh:

  ```
  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);
```

- Referensi korelasi dari dalam subquery yang berisi fungsi jendela.

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

- Referensi dalam kolom GROUP BY ke hasil subquery yang berkorelasi. Contoh:

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

- Referensi korelasi dari subquery dengan fungsi agregat dan klausa GROUP BY, terhubung ke kueri luar oleh predikat IN. (Pembatasan ini tidak berlaku untuk fungsi agregat MIN dan MAX.) Contoh:

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

AWS Clean Rooms Fungsi Spark SQL

AWS Clean Rooms Spark SQL mendukung fungsi SQL berikut:

Topik

- Fungsi agregat
- Fungsi array
- Ekspresi bersyarat
- Fungsi konstruktor
- Fungsi pemformatan tipe data

- Fungsi tanggal dan waktu
- Fungsi enkripsi dan dekripsi
- Fungsi hash
- Fungsi hyperloglog
- Fungsi JSON
- Fungsi matematika
- Fungsi skalar
- Fungsi string
- Fungsi terkait privasi
- Fungsi jendela

Fungsi agregat

Fungsi agregat di AWS Clean Rooms Spark SQL digunakan untuk melakukan perhitungan atau operasi pada sekelompok baris dan mengembalikan nilai tunggal. Mereka penting untuk analisis data dan tugas ringkasan.

AWS Clean Rooms Spark SQL mendukung fungsi agregat berikut:

Topik

- Fungsi ANY_VALUE
- APPROX COUNT_DISTINCT fungsi
- Fungsi PERSENTIL APPROX
- Fungsi AVG
- Fungsi BOOL_AND
- Fungsi BOOL_OR
- Fungsi KARDINALITAS
- Fungsi COLLECT_LIST
- Fungsi COLLECT_SET
- COUNTdan COUNT DISTINCT fungsi
- Fungsi COUNT
- Fungsi MAX

- Fungsi MEDIAN
- Fungsi MIN
- Fungsi PERSENTIL
- Fungsi SKEWNESS
- Fungsi STDDEV_SAMP dan STDDEV_POP
- SUMdan SUM DISTINCT fungsi
- Fungsi VAR_SAMP dan VAR_POP

Fungsi ANY_VALUE

Fungsi ANY_VALUE mengembalikan nilai apapun dari nilai ekspresi masukan nondeterministik. Fungsi ini dapat mengembalikan NULL jika ekspresi input tidak menghasilkan baris apa pun yang dikembalikan.

Sintaksis

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

Argumen

ekspresi

Kolom target atau ekspresi di mana fungsi beroperasi. Ekspresi adalah salah satu tipe data berikut:

isIgnoreNull

Sebuah boolean yang menentukan apakah fungsi harus mengembalikan hanya nilai-nilai non-null.

Pengembalian

Mengembalikan tipe data yang sama sebagai ekspresi.

Catatan penggunaan

Jika pernyataan yang menentukan fungsi ANY_VALUE untuk kolom juga menyertakan referensi kolom kedua, kolom kedua harus muncul dalam klausa GROUP BY atau disertakan dalam fungsi agregat.

Contoh

Contoh berikut mengembalikan sebuah instance dari `dateid` mana pun di mana `eventname` adalah`Eagles`.

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

Berikut adalah hasilnya.

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

Contoh berikut mengembalikan sebuah instance dari `dateid` mana pun di mana `eventname` adalah `Eagles` atau`Cold War Kids`.

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

Berikut adalah hasilnya.

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

APPROX COUNT_DISTINCT fungsi

APPROX COUNT_DISTINCT menyediakan cara yang efisien untuk memperkirakan jumlah nilai unik dalam kolom atau dataset.

Sintaksis

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

Pendapat

expr

Ekspresi atau kolom yang ingin Anda perkirakan jumlah nilai unik.

Ini bisa berupa kolom tunggal, ekspresi kompleks, atau kombinasi kolom.

KerabatD

Parameter opsional yang menentukan standar deviasi relatif yang diinginkan dari estimasi.

Ini adalah nilai antara 0 dan 1, mewakili kesalahan relatif maksimum yang dapat diterima dari estimasi. Nilai RelativeSD yang lebih kecil akan menghasilkan estimasi yang lebih akurat tetapi lebih lambat.

Jika parameter ini tidak disediakan, nilai default (biasanya sekitar 0,05 atau 5%) digunakan.

Pengembalian

Mengembalikan perkiraan kardinalitas oleh HyperLogLog ++. relativeSD mendefinisikan standar deviasi relatif maksimum yang diizinkan.

Contoh

Kueri berikut memperkirakan jumlah nilai unik di `col1` kolom, dengan standar deviasi relatif 1% (0,01).

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

Kueri berikut memperkirakan bahwa ada 3 nilai unik di `col1` kolom (nilai 1, 2, dan 3).

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

Fungsi PERSENTIL APPROX

PERKIRAAN PERSENTIL digunakan untuk memperkirakan nilai persentil dari ekspresi atau kolom tertentu tanpa harus mengurutkan seluruh dataset. Fungsi ini berguna dalam skenario di mana Anda perlu memahami dengan cepat distribusi kumpulan data besar atau melacak metrik berbasis persentil, tanpa overhead komputasi untuk melakukan perhitungan persentil yang tepat. Namun, penting untuk memahami trade-off antara kecepatan dan akurasi, dan untuk memilih toleransi kesalahan yang sesuai berdasarkan persyaratan spesifik kasus penggunaan Anda.

Sintaksis

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

Pendapat

expr

Ekspresi atau kolom yang ingin Anda perkirakan nilai persentil.

Ini bisa berupa kolom tunggal, ekspresi kompleks, atau kombinasi kolom.

persentil

Nilai persentil yang ingin Anda perkirakan, dinyatakan sebagai nilai antara 0 dan 1.

Misalnya, 0,5 akan sesuai dengan persentil ke-50 (median).

akurasi

Parameter opsional yang menentukan akurasi estimasi persentil yang diinginkan. Ini adalah nilai antara 0 dan 1, mewakili kesalahan relatif maksimum yang dapat diterima dari estimasi. `accuracy`Nilai yang lebih kecil akan menghasilkan estimasi yang lebih tepat tetapi lebih lambat. Jika parameter ini tidak disediakan, nilai default (biasanya sekitar 0,05 atau 5%) digunakan.

Pengembalian

Mengembalikan perkiraan persentil kolom interval numerik atau ANSI col yang merupakan nilai terkecil dalam nilai col yang diurutkan (diurutkan dari terkecil ke terbesar) sehingga tidak lebih dari persentase nilai col kurang dari nilai atau sama dengan nilai itu.

Nilai persentase harus antara 0,0 dan 1,0. Parameter akurasi (default: 10000) adalah literal numerik positif yang mengontrol akurasi perkiraan dengan mengorbankan memori.

Nilai akurasi yang lebih tinggi menghasilkan akurasi yang lebih baik, `1.0/accuracy` adalah kesalahan relatif dari perkiraan.

Ketika persentase adalah array, setiap nilai dari array persentase harus antara 0,0 dan 1,0. Dalam hal ini, mengembalikan perkiraan array persentil kolom col pada array persentase yang diberikan.

Contoh

Kueri berikut memperkirakan persentil ke-95 `response_time` kolom, dengan kesalahan relatif maksimum 1% (0,01).

```
SELECT APPROX_PERCENTILE(response_time, 0.95, 0.01) AS p95_response_time
```

```
FROM my_table;
```

Kueri berikut memperkirakan nilai persentil ke-50, ke-40, dan ke-10 dari kolom dalam `col` tabel. `tab`

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

Kueri berikut memperkirakan persentil ke-50 (median) dari nilai di kolom col.

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

Fungsi AVG

AVGFungsi mengembalikan rata-rata (rata-rata aritmatika) dari nilai ekspresi masukan. AVGFungsi ini bekerja dengan nilai numerik dan mengabaikan nilai NULL.

Sintaksis

```
AVG (column)
```

Pendapat

`column`

Kolom target tempat fungsi beroperasi. Kolom adalah salah satu tipe data berikut:

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

Jenis Data

Tipe argumen yang didukung oleh AVG fungsi adalahSMALLINT,INTEGER,BIGINT,DECIMAL, danDOUBLE.

Jenis pengembalian yang didukung oleh AVG fungsi adalah:

- BIGINTuntuk argumen tipe integer apa pun
- DOUBLEuntuk argumen floating point
- Mengembalikan tipe data yang sama sebagai ekspresi untuk jenis argumen lainnya

Presisi default untuk hasil AVG fungsi dengan DECIMAL argumen adalah 38. Skala hasilnya sama dengan skala argumen. Misalnya, DEC(5,2) kolom mengembalikan tipe DEC(38,2) data. AVG

Contoh

Temukan jumlah rata-rata yang terjual per transaksi dari SALES tabel.

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

Fungsi BOOL_AND

Fungsi BOOL_AND beroperasi pada satu kolom atau ekspresi Boolean atau integer. Fungsi ini menerapkan logika yang mirip dengan fungsi BIT_AND dan BIT_OR. Untuk fungsi ini, tipe kembali adalah nilai Boolean (`true`atau`false`).

Jika semua nilai dalam satu set adalah true, fungsi BOOL_AND mengembalikan `true` (). `t` Jika ada nilai palsu, fungsi mengembalikan `false` (`f`).

Sintaksis

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

Argumen

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi. Ekspresi ini harus memiliki tipe data BOOLEAN atau integer. Jenis pengembalian fungsi adalah BOOLEAN.

BERBEDA | SEMUA

Dengan argumen DISTINCT, fungsi menghilangkan semua nilai duplikat untuk ekspresi yang ditentukan sebelum menghitung hasilnya. Dengan argumen ALL, fungsi mempertahankan semua nilai duplikat. ALL adalah default.

Contoh

Anda dapat menggunakan fungsi Boolean terhadap ekspresi Boolean atau ekspresi integer.

Misalnya, query berikut mengembalikan hasil dari tabel USERS standar dalam database TICKIT, yang memiliki beberapa kolom Boolean.

Fungsi BOOL_AND kembali `false` untuk semua lima baris. Tidak semua pengguna di masing-masing negara menyukai olahraga.

```
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)
```

Fungsi BOOL_OR

Fungsi BOOL_OR beroperasi pada satu kolom atau ekspresi Boolean atau integer. Fungsi ini menerapkan logika yang mirip dengan fungsi BIT_AND dan BIT_OR. Untuk fungsi ini, tipe kembali adalah nilai Boolean (`true`,`false`, or`NULL`).

Jika nilai dalam satu set adalah`true`, fungsi BOOL_OR mengembalikan `true` (). `t` Jika nilai dalam satu set adalah`false`, fungsi mengembalikan `false` (`f`). NULL dapat dikembalikan jika nilainya tidak diketahui.

Sintaksis

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

Argumen

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi. Ekspresi ini harus memiliki tipe data BOOLEAN atau integer. Jenis pengembalian fungsi adalah BOOLEAN.

BERBEDA | SEMUA

Dengan argumen DISTINCT, fungsi menghilangkan semua nilai duplikat untuk ekspresi yang ditentukan sebelum menghitung hasilnya. Dengan argumen ALL, fungsi mempertahankan semua nilai duplikat. ALL adalah default.

Contoh

Anda dapat menggunakan fungsi Boolean dengan ekspresi Boolean atau ekspresi integer. Misalnya, query berikut mengembalikan hasil dari tabel USERS standar dalam database TICKIT, yang memiliki beberapa kolom Boolean.

Fungsi BOOL_OR kembali `true` untuk semua lima baris. Setidaknya satu pengguna di masing-masing negara bagian menyukai olahraga.

```
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)
```

Contoh berikut mengembalikan NULL.

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

Fungsi KARDINALITAS

Fungsi CARDINALITY mengembalikan ukuran ekspresi ARRAY atau MAP (expr).

Fungsi ini berguna untuk menemukan ukuran atau panjang array.

Sintaksis

```
cardinality(expr)
```

Pendapat

expr

Ekspresi ARRAY atau MAP.

Pengembalian

Mengembalikan ukuran array atau peta (INTEGER).

Fungsi mengembalikan `NULL` untuk input null jika `sizeOfNull` diatur ke `false` atau `enabled` diatur ke`true`.

Jika tidak, fungsi kembali `-1` untuk input null. Dengan pengaturan default, fungsi kembali `-1` untuk input null.

Contoh

Query berikut menghitung kardinalitas, atau jumlah elemen, dalam array yang diberikan. Array (`'b', 'd', 'c', 'a'`) memiliki 4 elemen, sehingga output dari query ini akan menjadi`4`.

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

Fungsi COLLECT_LIST

Fungsi COLLECT_LIST mengumpulkan dan mengembalikan daftar elemen non-unik.

Jenis fungsi ini berguna ketika Anda ingin mengumpulkan beberapa nilai dari satu set baris ke dalam array tunggal atau struktur data daftar.

> **Note**
>
> Fungsi ini non-deterministik karena urutan hasil yang dikumpulkan tergantung pada urutan baris, yang mungkin non-deterministik setelah operasi shuffle dilakukan.

Sintaksis

```
collect_list(expr)
```

Pendapat

expr

Ekspresi jenis apa pun.

Pengembalian

Mengembalikan ARRAY dari tipe argumen. Urutan elemen dalam array adalah non-deterministik.

Nilai NULL dikecualikan.

Jika DISTINCT ditentukan, fungsi hanya mengumpulkan nilai unik dan merupakan sinonim untuk `collect_set` fungsi agregat.

Contoh

Query berikut mengumpulkan semua nilai dari kolom col ke dalam daftar. `VALUES`Klausa ini digunakan untuk membuat tabel inline dengan tiga baris, di mana setiap baris memiliki satu kolom col dengan nilai 1, 2, dan 1 masing-masing. `collect_list()` Fungsi ini kemudian digunakan untuk menggabungkan semua nilai dari kolom col ke dalam array tunggal. Output dari pernyataan SQL ini akan menjadi array`[1,2,1]`, yang berisi semua nilai dari kolom col dalam urutan mereka muncul dalam data input.

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

Fungsi COLLECT_SET

Fungsi COLLECT_SET mengumpulkan dan mengembalikan satu set elemen unik.

Fungsi ini berguna ketika Anda ingin mengumpulkan semua nilai yang berbeda dari satu set baris ke dalam struktur data tunggal, tanpa menyertakan duplikat apa pun.

> **Note**
>
> Fungsi ini non-deterministik karena urutan hasil yang dikumpulkan tergantung pada urutan baris, yang mungkin non-deterministik setelah operasi shuffle dilakukan.

Sintaksis

```
collect_set(expr)
```

Pendapat

expr

Ekspresi jenis apa pun kecuali MAP.

Pengembalian

Mengembalikan ARRAY dari tipe argumen. Urutan elemen dalam array adalah non-deterministik.

Nilai NULL dikecualikan.

Contoh

Query berikut mengumpulkan semua nilai unik dari kolom col ke dalam satu set. `VALUES`Klausa ini digunakan untuk membuat tabel inline dengan tiga baris, di mana setiap baris memiliki satu kolom col dengan nilai 1, 2, dan 1 masing-masing. `collect_set()`Fungsi ini kemudian digunakan untuk menggabungkan semua nilai unik dari kolom col ke dalam satu set. Output dari pernyataan SQL ini akan menjadi himpunan`[1,2]`, yang berisi nilai-nilai unik dari kolom col. Nilai duplikat 1 hanya disertakan sekali dalam hasil.

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

COUNTdan COUNT DISTINCT fungsi

COUNTFungsi menghitung baris yang ditentukan oleh ekspresi. COUNT DISTINCTFungsi menghitung jumlah nilai non-Null yang berbeda dalam kolom atau ekspresi. Ini menghilangkan semua nilai duplikat dari ekspresi yang ditentukan sebelum melakukan penghitungan.

Sintaksis

```
COUNT (DISTINCT column)
```

Pendapat

column

Kolom target tempat fungsi beroperasi.

Jenis Data

COUNTFungsi dan COUNT DISTINCT fungsi mendukung semua tipe data argumen.

COUNT DISTINCTFungsi kembaliBIGINT.

Contoh

Hitung semua pengguna dari negara bagian Florida.

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

Hitung semua tempat unik IDs dari EVENT meja.

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

Fungsi COUNT

Fungsi COUNT menghitung baris yang ditentukan oleh ekspresi.

Fungsi COUNT memiliki variasi berikut.

- COUNT (*) menghitung semua baris dalam tabel target apakah mereka termasuk nol atau tidak.
- COUNT (ekspresi) menghitung jumlah baris dengan nilai non-Null dalam kolom atau ekspresi tertentu.
- COUNT (ekspresi DISTINCT) menghitung jumlah nilai non-Null yang berbeda dalam kolom atau ekspresi.

Sintaksis

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

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

Argumen

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi. Fungsi COUNT mendukung semua tipe data argumen.

BERBEDA | SEMUA

Dengan argumen DISTINCT, fungsi menghilangkan semua nilai duplikat dari ekspresi yang ditentukan sebelum melakukan penghitungan. Dengan argumen ALL, fungsi mempertahankan semua nilai duplikat dari ekspresi untuk menghitung. ALL adalah default.

Jenis pengembalian

Fungsi COUNT mengembalikan BIGINT.

Contoh

Hitung semua pengguna dari negara bagian Florida:

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

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

Hitung semua nama acara dari tabel EVENT:

```
select count(eventname) from event;

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

Hitung semua nama acara dari tabel EVENT:

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

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

Hitung semua tempat unik IDs dari tabel EVENT:

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

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

Hitung berapa kali setiap penjual mencantumkan batch lebih dari empat tiket untuk dijual. Kelompokkan hasil berdasarkan ID penjual:

```
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
...
```

Fungsi MAX

Fungsi MAX mengembalikan nilai maksimum dalam satu set baris. DISTINCT atau ALL dapat digunakan tetapi tidak mempengaruhi hasilnya.

Sintaksis

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

Argumen

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi. Ekspresi adalah tipe data numerik apa pun.

BERBEDA | SEMUA

Dengan argumen DISTINCT, fungsi menghilangkan semua nilai duplikat dari ekspresi yang ditentukan sebelum menghitung maksimum. Dengan argumen ALL, fungsi mempertahankan semua nilai duplikat dari ekspresi untuk menghitung maksimum. ALL adalah default.

Jenis Data

Mengembalikan tipe data yang sama sebagai ekspresi.

Contoh

Temukan harga tertinggi yang dibayarkan dari semua penjualan:

```
select max(pricepaid) from sales;

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

Temukan harga tertinggi yang dibayarkan per tiket dari semua penjualan:

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

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

Fungsi MEDIAN

Sintaksis

```
MEDIAN ( median_expression )
```

Argumen

median_expression

Kolom target atau ekspresi tempat fungsi beroperasi.

Fungsi MIN

Fungsi MIN mengembalikan nilai minimum dalam satu set baris. DISTINCT atau ALL dapat digunakan tetapi tidak mempengaruhi hasilnya.

Sintaksis

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

Argumen

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi. Ekspresi adalah tipe data numerik apa pun.

BERBEDA | SEMUA

Dengan argumen DISTINCT, fungsi menghilangkan semua nilai duplikat dari ekspresi yang ditentukan sebelum menghitung minimum. Dengan argumen ALL, fungsi mempertahankan semua nilai duplikat dari ekspresi untuk menghitung minimum. ALL adalah default.

Jenis Data

Mengembalikan tipe data yang sama sebagai ekspresi.

Contoh

Temukan harga terendah yang dibayarkan dari semua penjualan:

```
select min(pricepaid) from sales;

min
-------
20.00
```

```
(1 row)
```

Temukan harga terendah yang dibayarkan per tiket dari semua penjualan:

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

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

Fungsi PERSENTIL

Fungsi PERCENTILE digunakan untuk menghitung nilai persentil yang tepat dengan terlebih dahulu menyortir nilai di `col` kolom dan kemudian menemukan nilai pada yang ditentukan. `percentage`

Fungsi PERCENTILE berguna ketika Anda perlu menghitung nilai persentil yang tepat dan biaya komputasi dapat diterima untuk kasus penggunaan Anda. Ini memberikan hasil yang lebih akurat daripada fungsi APPROX_PERCENTILE, tetapi mungkin lebih lambat, terutama untuk kumpulan data besar.

Sebaliknya, fungsi APPROX_PERCENTILE adalah alternatif yang lebih efisien yang dapat memberikan perkiraan nilai persentil dengan toleransi kesalahan yang ditentukan, sehingga lebih cocok untuk skenario di mana kecepatan adalah prioritas yang lebih tinggi daripada presisi absolut.

Sintaksis

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

Pendapat

col

Ekspresi atau kolom yang ingin Anda hitung nilai persentilnya.

persentase

Nilai persentil yang ingin Anda hitung, dinyatakan sebagai nilai antara 0 dan 1.

Misalnya, 0,5 akan sesuai dengan persentil ke-50 (median).

frekuensi

Parameter opsional yang menentukan frekuensi atau berat setiap nilai dalam `col` kolom. Jika disediakan, fungsi akan menghitung persentil berdasarkan frekuensi masing-masing nilai.

Pengembalian

Mengembalikan nilai persentil yang tepat dari kolom interval numerik atau ANSI col pada persentase yang diberikan.

Nilai persentase harus antara 0,0 dan 1,0.

Nilai frekuensi harus integral positif

Contoh

Query berikut menemukan nilai yang lebih besar dari atau sama dengan 30% dari nilai-nilai dalam `col` kolom. Karena nilainya 0 dan 10, persentil ke-30 adalah 3,0, karena nilainya lebih besar dari atau sama dengan 30% data.

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

Fungsi SKEWNESS

Fungsi SKEWNESS mengembalikan nilai skewness dihitung dari nilai-nilai grup.

Skewness adalah ukuran statistik yang menggambarkan asimetri atau kurangnya simetri dalam kumpulan data. Ini memberikan informasi tentang bentuk distribusi data.

Fungsi ini dapat berguna dalam memahami sifat statistik dari kumpulan data dan menginformasikan analisis lebih lanjut atau pengambilan keputusan.

Sintaksis

```
skewness(expr)
```

Pendapat

expr

Ekspresi yang mengevaluasi ke numerik.

Pengembalian

Mengembalikan DOUBLE.

Jika DISTINCT ditentukan, fungsi hanya beroperasi pada satu set unik nilai expr.

Contoh

Kueri berikut menghitung kemiringan nilai dalam kolom. `col` Dalam contoh ini, `VALUES` klausa digunakan untuk membuat tabel sebaris dengan empat baris, di mana setiap baris memiliki satu kolom `col` dengan nilai -10, -20, 100, dan 1000. `skewness()`Fungsi ini kemudian digunakan untuk menghitung kemiringan nilai dalam kolom. `col` Hasilnya, 1.1135657469022011, mewakili derajat dan arah kemiringan dalam data. Nilai kemiringan positif menunjukkan bahwa data miring ke kanan, dengan sebagian besar nilai terkonsentrasi di sisi kiri distribusi. Nilai kemiringan negatif menunjukkan bahwa data miring ke kiri, dengan sebagian besar nilai terkonsentrasi di sisi kanan distribusi.

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

Query berikut menghitung kemiringan nilai dalam kolom col. Mirip dengan contoh sebelumnya, `VALUES` klausa digunakan untuk membuat tabel sebaris dengan empat baris, di mana setiap baris memiliki satu kolom `col` dengan nilai -1000, -100, 10, dan 20. `skewness()`Fungsi ini kemudian digunakan untuk menghitung kemiringan nilai dalam kolom. `col` Hasilnya, -1.1135657469022011, mewakili derajat dan arah kemiringan dalam data. Dalam hal ini, nilai kemiringan negatif menunjukkan bahwa data miring ke kiri, dengan sebagian besar nilai terkonsentrasi di sisi kanan distribusi.

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

Fungsi STDDEV_SAMP dan STDDEV_POP

Fungsi STDDEV_SAMP dan STDDEV_POP mengembalikan sampel dan standar deviasi populasi dari satu set nilai numerik (integer, desimal, atau floating-point). Hasil dari fungsi STDDEV_SAMP setara dengan akar kuadrat dari varians sampel dari kumpulan nilai yang sama.

STDDEV_SAMP dan STDDEV adalah sinonim untuk fungsi yang sama.

Sintaksis

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

Ekspresi harus memiliki tipe data numerik. Terlepas dari tipe data ekspresi, tipe pengembalian fungsi ini adalah angka presisi ganda.

> **Note**
>
> Standar deviasi dihitung menggunakan aritmatika floating point, yang dapat mengakibatkan sedikit ketidaktepatan.

Catatan penggunaan

Ketika standar deviasi sampel (STDDEV atau STDDEV_SAMP) dihitung untuk ekspresi yang terdiri dari satu nilai, hasil fungsinya adalah NULL bukan 0.

Contoh

Kueri berikut mengembalikan rata-rata nilai di kolom VENUESEATS dari tabel VENUE, diikuti oleh standar deviasi sampel dan standar deviasi populasi dari kumpulan nilai yang sama. VENUESEATS adalah kolom INTEGER. Skala hasil dikurangi menjadi 2 digit.

```
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)
```

Kueri berikut mengembalikan standar deviasi sampel untuk kolom KOMISI dalam tabel PENJUALAN. KOMISI adalah kolom DESIMAL. Skala hasilnya dikurangi menjadi 10 digit.

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

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

Kueri berikut menampilkan standar deviasi sampel untuk kolom COMMISSION sebagai integer.

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

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

Kueri berikut mengembalikan standar deviasi sampel dan akar kuadrat dari varians sampel untuk kolom KOMISI. Hasil perhitungan ini sama.

```
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)
```

SUMdan SUM DISTINCT fungsi

SUMFungsi mengembalikan jumlah kolom input atau nilai ekspresi. SUMFungsi ini bekerja dengan nilai numerik dan mengabaikan NULL nilai.

SUM DISTINCTFungsi menghilangkan semua nilai duplikat dari ekspresi yang ditentukan sebelum menghitung jumlah.

Sintaksis

```
SUM (DISTINCT column )
```

Pendapat

column

Kolom target tempat fungsi beroperasi. Kolom adalah semua tipe data numerik.

Contoh

Temukan jumlah semua komisi yang dibayarkan dari SALES tabel.

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

Temukan jumlah semua komisi berbeda yang dibayarkan dari SALES tabel.

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

Fungsi VAR_SAMP dan VAR_POP

Fungsi VAR_SAMP dan VAR_POP mengembalikan sampel dan varians populasi dari sekumpulan nilai numerik (integer, desimal, atau floating-point). Hasil dari fungsi VAR_SAMP setara dengan standar deviasi sampel kuadrat dari kumpulan nilai yang sama.

VAR_SAMP dan VARIANCE adalah sinonim untuk fungsi yang sama.

Sintaksis

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

Ekspresi harus memiliki tipe data integer, desimal, atau floating-point. Terlepas dari tipe data ekspresi, tipe pengembalian fungsi ini adalah angka presisi ganda.

> **Note**
>
> Hasil dari fungsi-fungsi ini mungkin bervariasi di seluruh cluster data warehouse, tergantung pada konfigurasi cluster dalam setiap kasus.

Catatan penggunaan

Ketika varians sampel (VARIANCE atau VAR_SAMP) dihitung untuk ekspresi yang terdiri dari satu nilai, hasil fungsinya adalah NULL bukan 0.

Contoh

Kueri berikut mengembalikan sampel bulat dan varians populasi kolom NUMTICKETS dalam tabel LISTING.

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

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

Kueri berikut menjalankan perhitungan yang sama tetapi melemparkan hasilnya ke nilai desimal.

```
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)
```

Fungsi array

Bagian ini menjelaskan fungsi array untuk SQL yang didukung di AWS Clean Rooms.

Topik

- Fungsi ARRAY
- Fungsi ARRAY_CONTAINS
- Fungsi ARRAY_DISTINCT

- Fungsi ARRAY_EXCEPLE
- Fungsi ARRAY_INTERSECT
- Fungsi ARRAY_JOIN
- Fungsi ARRAY_REMOVE
- Fungsi ARRAY_UNION
- Fungsi EXPLODE
- Fungsi FLATTEN

Fungsi ARRAY

Menciptakan array dengan elemen yang diberikan.

Sintaksis

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

Pendapat

expr1, expr2

Ekspresi tipe data apa pun kecuali tipe tanggal dan waktu. Argumen tidak harus dari tipe data yang sama.

Jenis pengembalian

Fungsi array mengembalikan ARRAY dengan unsur-unsur dalam ekspresi.

Contoh

Contoh berikut menunjukkan array nilai numerik dan array tipe data yang berbeda.

```
--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)
```

Fungsi ARRAY_CONTAINS

Fungsi ARRAY_CONTAINS dapat digunakan untuk melakukan pemeriksaan keanggotaan dasar pada struktur data array. Fungsi ARRAY_CONTAINS berguna ketika Anda perlu memeriksa apakah nilai tertentu hadir dalam array.

Sintaksis

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

Pendapat

array

ARRAY yang akan dicari.

nilai

Ekspresi dengan tipe berbagi tipe yang paling tidak umum dengan elemen array.

Jenis pengembalian

Fungsi ARRAY_CONTAINS mengembalikan BOOLEAN.

Jika nilainya NULL, hasilnya adalah NULL.

Jika ada elemen dalam array adalah NULL, hasilnya adalah NULL jika nilai tidak cocok dengan elemen lainnya.

Contoh

Contoh berikut memeriksa apakah array `[1, 2, 3]` berisi nilai`4`. Karena array`[1, 2, 3]` tidak berisi nilai`4`, fungsi array_contains kembali. `false`

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

Contoh berikut memeriksa apakah array `[1, 2, 3]` berisi nilai2. Karena array `[1, 2, 3]` memang berisi nilai2, fungsi array_contains kembali. `true`

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

Fungsi ARRAY_DISTINCT

Fungsi ARRAY_DISTINCT dapat digunakan untuk menghapus nilai duplikat dari array. Fungsi ARRAY_DISTINCT berguna ketika Anda perlu menghapus duplikat dari array dan bekerja dengan hanya elemen unik. Ini dapat membantu dalam skenario di mana Anda ingin melakukan operasi atau analisis pada kumpulan data tanpa gangguan nilai berulang.

Sintaksis

```
array_distinct(array)
```

Pendapat

array

Ekspresi ARRAY.

Jenis pengembalian

Fungsi ARRAY_DISTINCT mengembalikan ARRAY yang hanya berisi elemen unik dari array input.

Contoh

Dalam contoh ini, array input `[1, 2, 3, null, 3]` berisi nilai duplikat dari. `3` `array_distinct`Fungsi menghapus nilai duplikat ini `3` dan mengembalikan array baru dengan elemen unik:`[1, 2, 3, null]`.

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

Dalam contoh ini, array input `[1, 2, 2, 3, 3, 3]` berisi nilai duplikat 2 dan3. `array_distinct`Fungsi menghapus duplikat ini dan mengembalikan array baru dengan elemen unik:`[1, 2, 3]`.

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

Fungsi ARRAY_EXCEPLE

Fungsi ARRAY_EXCEPLE mengambil dua array sebagai argumen dan mengembalikan array baru yang hanya berisi elemen yang hadir dalam array pertama tetapi bukan array kedua.

ARRAY_EXCEPLE berguna ketika Anda perlu menemukan elemen yang unik untuk satu array dibandingkan dengan yang lain. Ini dapat membantu dalam skenario di mana Anda perlu melakukan operasi set-like pada array, seperti menemukan perbedaan antara dua set data.

Sintaksis

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

Pendapat

array1

ARRAY jenis apa pun dengan elemen yang sebanding.

array2

ARRAY elemen berbagi tipe yang paling tidak umum dengan elemen array1.

Jenis pengembalian

Fungsi ARRAY_EXCEPLE mengembalikan ARRAY tipe yang cocok untuk array1 tanpa duplikat.

Contoh

Dalam contoh ini, array pertama `[1, 2, 3]` berisi elemen 1, 2, dan 3. Array kedua `[2, 3, 4]` berisi elemen 2, 3, dan 4. `array_except`Fungsi menghapus elemen 2 dan 3 dari array pertama, karena mereka juga hadir dalam array kedua. Output yang dihasilkan adalah array`[1]`.

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

```
  [1]
```

Dalam contoh ini, array pertama `[1, 2, 3]` berisi elemen 1, 2, dan 3. Array kedua `[1, 3, 5]` berisi elemen 1, 3, dan 5. `array_except`Fungsi menghapus elemen 1 dan 3 dari array pertama, karena mereka juga hadir dalam array kedua. Output yang dihasilkan adalah array`[2]`.

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

Fungsi ARRAY_INTERSECT

Fungsi ARRAY_INTERSECT mengambil dua array sebagai argumen dan mengembalikan array baru yang berisi elemen yang ada di kedua array input. Fungsi ini berguna ketika Anda perlu menemukan elemen umum antara dua array. Ini dapat membantu dalam skenario di mana Anda perlu melakukan operasi set-like pada array, seperti menemukan persimpangan antara dua set data.

Sintaksis

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

Pendapat

array1

ARRAY jenis apa pun dengan elemen yang sebanding.

array2

ARRAY elemen berbagi tipe yang paling tidak umum dengan elemen array1.

Jenis pengembalian

Fungsi ARRAY_INTERSECT mengembalikan ARRAY tipe yang cocok ke array1 tanpa duplikat dan elemen yang terkandung dalam array1 dan array2.

Contoh

Dalam contoh ini, array pertama `[1, 2, 3]` berisi elemen 1, 2, dan 3. Array kedua `[1, 3, 5]` berisi elemen 1, 3, dan 5. Fungsi ARRAY_INTERSECT mengidentifikasi elemen umum antara dua array, yaitu 1 dan 3. Array keluaran yang dihasilkan adalah`[1, 3]`.

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

Fungsi ARRAY_JOIN

Fungsi ARRAY_JOIN mengambil dua argumen: Argumen pertama adalah array input yang akan bergabung. Argumen kedua adalah string pemisah yang akan digunakan untuk menggabungkan elemen array. Fungsi ini berguna ketika Anda perlu mengubah array string (atau tipe data lainnya) menjadi string gabungan tunggal. Ini dapat membantu dalam skenario di mana Anda ingin menyajikan array nilai sebagai string berformat tunggal, seperti untuk tujuan tampilan atau untuk digunakan dalam pemrosesan lebih lanjut.

Sintaksis

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

Pendapat

array

Setiap jenis ARRAY, tetapi elemen-elemennya ditafsirkan sebagai string.

pembatas

Sebuah STRING digunakan untuk memisahkan elemen array gabungan.

NullReplacement

Sebuah STRING digunakan untuk mengekspresikan nilai NULL dalam hasil.

Jenis pengembalian

Fungsi ARRAY_JOIN mengembalikan STRING di mana elemen array dipisahkan oleh pembatas dan elemen null diganti. `nullReplacement` Jika `nullReplacement` dihilangkan, `null` elemen disaring. Jika ada argumen`NULL`, hasilnya adalah`NULL`.

Contoh

Dalam contoh ini, fungsi ARRAY_JOIN mengambil array `['hello', 'world']` dan menggabungkan elemen menggunakan pemisah `' '` (karakter spasi). Output yang dihasilkan adalah string`'hello world'`.

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

Dalam contoh ini, fungsi ARRAY_JOIN mengambil array `['hello', null, 'world']` dan menggabungkan elemen menggunakan pemisah `' '` (karakter spasi). `null`Nilai diganti dengan string pengganti yang disediakan `','` (koma). Output yang dihasilkan adalah string`'hello , world'`.

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

Fungsi ARRAY_REMOVE

Fungsi ARRAY_REMOVE mengambil dua argumen: Argumen pertama adalah array input dari mana elemen akan dihapus. Argumen kedua adalah nilai yang akan dihapus dari array. Fungsi ini berguna ketika Anda perlu menghapus elemen tertentu dari array. Ini dapat membantu dalam skenario di mana Anda perlu melakukan pembersihan data atau preprocessing pada array nilai.

Sintaksis

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

Pendapat

array

Sebuah ARRAY.

elemen

Ekspresi tipe berbagi tipe yang paling tidak umum dengan elemen array.

Jenis pengembalian

Fungsi ARRAY_REMOVE mengembalikan jenis hasil yang cocok dengan tipe array. Jika elemen yang akan dihapus adalah`NULL`, hasilnya adalah`NULL`.

Contoh

Dalam contoh ini, fungsi ARRAY_REMOVE mengambil array `[1, 2, 3, null, 3]` dan menghapus semua kemunculan nilai 3. Output yang dihasilkan adalah array`[1, 2, null]`.

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

Fungsi ARRAY_UNION

Fungsi ARRAY_UNION mengambil dua array sebagai argumen dan mengembalikan array baru yang berisi elemen unik dari kedua array input. Fungsi ini berguna ketika Anda perlu menggabungkan dua array dan menghilangkan elemen duplikat. Ini dapat membantu dalam skenario di mana Anda perlu melakukan operasi set-like pada array, seperti menemukan penyatuan antara dua set data.

Sintaksis

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

Pendapat

array1

Sebuah ARRAY.

array2

ARRAY dari tipe yang sama dengan array1.

Jenis pengembalian

Fungsi ARRAY_UNION mengembalikan ARRAY dari jenis yang sama seperti array.

Contoh

Dalam contoh ini, array pertama `[1, 2, 3]` berisi elemen 1, 2, dan 3. Array kedua `[1, 3, 5]` berisi elemen 1, 3, dan 5. Fungsi ARRAY_UNION menggabungkan elemen unik dari kedua array, menghasilkan array output. `[1, 2, 3, 5]` T

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

Fungsi EXPLODE

Fungsi EXPLODE digunakan untuk mengubah satu baris dengan array atau kolom peta menjadi beberapa baris, di mana setiap baris sesuai dengan satu elemen dari array atau peta.

Sintaksis

```
explode(expr)
```

Pendapat

expr

Ekspresi array atau ekspresi peta.

Jenis pengembalian

Fungsi EXPLODE mengembalikan satu set baris, di mana setiap baris mewakili elemen tunggal dari array input atau peta.

Tipe data dari baris output tergantung pada tipe data elemen dalam array input atau peta.

Contoh

Contoh berikut mengambil array baris tunggal [10, 20] dan mengubahnya menjadi dua baris terpisah, masing-masing berisi salah satu elemen array (10 dan 20).

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

Pada contoh pertama, array input langsung diteruskan sebagai argumen ke`explode()`. Dalam contoh ini, array input ditentukan menggunakan => sintaks, di mana nama kolom (`collection`) secara eksplisit disediakan.

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

Kedua pendekatan tersebut valid dan mencapai hasil yang sama, tetapi sintaks kedua dapat lebih berguna ketika Anda perlu meledakkan kolom dari kumpulan data yang lebih besar, bukan hanya literal array sederhana.

Fungsi FLATTEN

Fungsi FLATTEN digunakan untuk “meratakan” struktur array bersarang menjadi array datar tunggal.

Sintaksis

```
flatten(arrayOfArrays)
```

Pendapat

arrayOfArrays

Sebuah array array.

Jenis pengembalian

Fungsi FLATTEN mengembalikan array.

Contoh

Dalam contoh ini, input adalah array bersarang dengan dua array dalam, dan outputnya adalah array datar tunggal yang berisi semua elemen dari array bagian dalam. Fungsi FLATTEN mengambil array bersarang `[[1, 2], [3, 4]]` dan menggabungkan semua elemen ke dalam satu array. `[1, 2, 3, 4]`

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

Ekspresi bersyarat

Dalam SQL, ekspresi kondisional digunakan untuk membuat keputusan berdasarkan kondisi tertentu. Mereka memungkinkan Anda untuk mengontrol aliran pernyataan SQL Anda dan mengembalikan nilai yang berbeda atau melakukan tindakan yang berbeda berdasarkan evaluasi satu atau lebih kondisi.

AWS Clean Rooms mendukung ekspresi bersyarat berikut:

Topik

- Ekspresi bersyarat CASE
- COALESCEekspresi
- Ekspresi terbesar dan terkecil
- Ekspresi IF
- Ekspresi IS_NULL
- Ekspresi IS_NOT_NULL
- Fungsi NVL dan COALESCE
- NVL2 fungsi

- Fungsi NULLIF

Ekspresi bersyarat CASE

Ekspresi CASE adalah ekspresi bersyarat, mirip dengan if/then/else pernyataan yang ditemukan dalam bahasa lain. CASE digunakan untuk menentukan hasil jika terdapat beberapa kondisi. Gunakan CASE di mana ekspresi SQL valid, seperti dalam perintah SELECT.

Ada dua jenis ekspresi CASE: sederhana dan dicari.

- Dalam ekspresi CASE sederhana, ekspresi dibandingkan dengan nilai. Ketika kecocokan ditemukan, tindakan yang ditentukan dalam klausul THEN diterapkan. Jika tidak ada kecocokan ditemukan, tindakan dalam klausul ELSE diterapkan.
- Dalam ekspresi CASE yang dicari, setiap CASE dievaluasi berdasarkan ekspresi Boolean, dan pernyataan CASE mengembalikan CASE yang cocok pertama. Jika tidak ada kecocokan yang ditemukan di antara klausa WHEN, tindakan dalam klausa ELSE dikembalikan.

Sintaksis

Pernyataan CASE sederhana yang digunakan untuk menyesuaikan kondisi:

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

Pernyataan CASE yang dicari digunakan untuk mengevaluasi setiap kondisi:

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

Argumen

ekspresi

Nama kolom atau ekspresi yang valid.

nilai

Nilai yang dibandingkan dengan ekspresi, seperti konstanta numerik atau string karakter.

hasil

Nilai target atau ekspresi yang dikembalikan ketika ekspresi atau kondisi Boolean dievaluasi. Tipe data dari semua ekspresi hasil harus dikonversi ke tipe output tunggal.

ketentuan

Ekspresi Boolean yang mengevaluasi benar atau salah. Jika kondisi benar, nilai ekspresi CASE adalah hasil yang mengikuti kondisi, dan sisa ekspresi CASE tidak diproses. Jika kondisinya salah, klausa WHEN berikutnya dievaluasi. Jika tidak ada hasil kondisi WHEN yang benar, nilai ekspresi CASE adalah hasil dari klausa ELSE. Jika klausa ELSE dihilangkan dan tidak ada kondisi yang benar, hasilnya adalah nol.

Contoh

Gunakan ekspresi CASE sederhana untuk mengganti `New York City` `Big Apple` dengan query terhadap tabel VENUE. Ganti semua nama kota lainnya dengan `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
...
```

Gunakan ekspresi CASE yang dicari untuk menetapkan nomor grup berdasarkan nilai PRICEPAID untuk penjualan tiket individu:

```
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
...
```

COALESCEekspresi

COALESCEEkspresi mengembalikan nilai ekspresi pertama dalam daftar yang tidak null. Jika semua ekspresi adalah null, hasilnya adalah null. Ketika nilai non-null ditemukan, ekspresi yang tersisa dalam daftar tidak dievaluasi.

Jenis ekspresi ini berguna ketika Anda ingin mengembalikan nilai cadangan untuk sesuatu ketika nilai yang diinginkan hilang atau nol. Misalnya, kueri mungkin mengembalikan salah satu dari tiga nomor telepon (sel, rumah, atau kantor, dalam urutan itu), mana yang ditemukan pertama kali dalam tabel (bukan nol).

Sintaksis

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

Contoh

Terapkan COALESCE ekspresi ke dua kolom.

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

Nama kolom default untuk ekspresi NVL adalah. COALESCE Query berikut mengembalikan hasil yang sama.

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

Ekspresi terbesar dan terkecil

Mengembalikan nilai terbesar atau terkecil dari daftar sejumlah ekspresi.

Sintaksis

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

Parameter

expression_list

Daftar ekspresi yang dipisahkan koma, seperti nama kolom. Ekspresi semua harus dikonversi ke tipe data umum. Nilai NULL dalam daftar diabaikan. Jika semua ekspresi mengevaluasi ke NULL, hasilnya adalah NULL.

Pengembalian

Mengembalikan nilai terbesar (untuk GREATEST) atau least (untuk LEAST) dari daftar ekspresi yang disediakan.

Contoh

Contoh berikut mengembalikan nilai tertinggi menurut abjad untuk `firstname` atau. `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)
```

Ekspresi IF

Fungsi kondisional IF mengembalikan salah satu dari dua nilai berdasarkan kondisi.

Fungsi ini adalah pernyataan aliran kontrol umum yang digunakan dalam SQL untuk membuat keputusan dan mengembalikan nilai yang berbeda berdasarkan evaluasi suatu kondisi. Ini berguna untuk menerapkan logika if-else sederhana dalam kueri.

Sintaksis

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

Pendapat

expr1

Kondisi atau ekspresi yang dievaluasi. Jika ya`true`, fungsi akan mengembalikan nilai expr2. Jika expr1 adalah*`false`*, fungsi akan mengembalikan nilai expr3.

expr2

Ekspresi yang dievaluasi dan dikembalikan jika expr1 adalah. `true`

expr3

Ekspresi yang dievaluasi dan dikembalikan jika expr1 adalah. `false`

Pengembalian

Jika `expr1` mengevaluasi`true`, maka kembali`expr2`; jika tidak kembali`expr3`.

Contoh

Contoh berikut menggunakan `if()` fungsi untuk mengembalikan salah satu dari dua nilai berdasarkan kondisi. Kondisi yang dievaluasi adalah`1 < 2`, yaitu`true`, sehingga nilai pertama `'a'` dikembalikan.

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

```
 a]
```

Ekspresi IS_NULL

Ekspresi IS_NULL kondisional digunakan untuk memeriksa apakah nilai adalah null.

Ungkapan ini adalah sinonim untuk. IS NULL

Sintaksis

```
is_null(expr)
```

Pendapat

expr

Ekspresi jenis apa pun.

Pengembalian

Ekspresi IS_NULL kondisional mengembalikan Boolean. Jika `expr1` NULL, kembali`true`, jika tidak kembali`false`.

Contoh

Contoh berikut memeriksa 1 apakah nilainya nol, dan mengembalikan hasil boolean `true` karena 1 adalah nilai yang valid, non-null.

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

Contoh berikut memilih `id` kolom dari `squirrels` tabel, tetapi hanya untuk baris di mana kolom usia berada`null`.

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

Ekspresi IS_NOT_NULL

Ekspresi IS_NOT_NULL kondisional digunakan untuk memeriksa apakah nilai tidak null.

Ungkapan ini adalah sinonim untuk. IS NOT NULL

Sintaksis

```
is_not_null(expr)
```

Pendapat

expr

Ekspresi jenis apa pun.

Pengembalian

Ekspresi IS_NOT_NULL kondisional mengembalikan Boolean. Jika `expr1` tidak NULL, kembali`true`, jika tidak kembali`false`.

Contoh

Contoh berikut memeriksa 1 apakah nilainya tidak null, dan mengembalikan hasil boolean `true` karena 1 adalah nilai yang valid, non-null.

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

Contoh berikut memilih `id` kolom dari `squirrels` tabel, tetapi hanya untuk baris di mana kolom usia tidak`null`.

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

Fungsi NVL dan COALESCE

Mengembalikan nilai ekspresi pertama yang tidak null dalam serangkaian ekspresi. Ketika nilai non-null ditemukan, ekspresi yang tersisa dalam daftar tidak dievaluasi.

NVL identik dengan COALESCE. Mereka adalah sinonim. Topik ini menjelaskan sintaks dan berisi contoh untuk keduanya.

Sintaksis

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

Sintaks untuk COALESCE adalah sama:

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

Jika semua ekspresi adalah null, hasilnya adalah null.

Fungsi-fungsi ini berguna ketika Anda ingin mengembalikan nilai sekunder ketika nilai primer hilang atau null. Misalnya, kueri mungkin mengembalikan yang pertama dari tiga nomor telepon yang tersedia: ponsel, rumah, atau kantor. Urutan ekspresi dalam fungsi menentukan urutan evaluasi.

Argumen

ekspresi

Ekspresi, seperti nama kolom, yang akan dievaluasi untuk status null.

Jenis pengembalian

AWS Clean Rooms menentukan tipe data dari nilai yang dikembalikan berdasarkan ekspresi masukan. Jika tipe data dari ekspresi input tidak memiliki tipe umum, maka kesalahan dikembalikan.

Contoh

Jika daftar berisi ekspresi integer, fungsi mengembalikan integer.

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

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

Contoh ini, yang sama dengan contoh sebelumnya, kecuali menggunakan NVL, mengembalikan hasil yang sama.

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

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

Contoh berikut mengembalikan tipe string.

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

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

Contoh berikut menghasilkan kesalahan karena tipe data bervariasi dalam daftar ekspresi. Dalam hal ini, ada tipe string dan tipe angka dalam daftar.

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

NVL2 fungsi

Mengembalikan salah satu dari dua nilai berdasarkan apakah ekspresi tertentu mengevaluasi ke NULL atau TIDAK NULL.

Sintaksis

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

Argumen

ekspresi

Ekspresi, seperti nama kolom, yang akan dievaluasi untuk status null.

not_null_return_value

Nilai yang dikembalikan jika ekspresi mengevaluasi ke NOT NULL. Nilai not_null_return_value harus memiliki tipe data yang sama dengan ekspresi atau secara implisit dapat dikonversi ke tipe data tersebut.

null_return_value

Nilai yang dikembalikan jika ekspresi mengevaluasi ke NULL. Nilai null_return_value harus memiliki tipe data yang sama dengan ekspresi atau secara implisit dapat dikonversi ke tipe data tersebut.

Jenis pengembalian

Jenis NVL2 pengembalian ditentukan sebagai berikut:

- Jika not_null_return_value atau null_return_value adalah null, tipe data dari ekspresi not-null dikembalikan.

Jika kedua not_null_return_value dan null_return_value tidak null:

- Jika not_null_return_value dan null_return_value memiliki tipe data yang sama, tipe data tersebut dikembalikan.
- Jika not_null_return_value dan null_return_value memiliki tipe data numerik yang berbeda, tipe data numerik terkecil yang kompatibel dikembalikan.
- Jika not_null_return_value dan null_return_value memiliki tipe data datetime yang berbeda, tipe data stempel waktu dikembalikan.
- Jika not_null_return_value dan null_return_value memiliki tipe data karakter yang berbeda, tipe data not_null_return_value dikembalikan.
- Jika not_null_return_value dan null_return_value memiliki tipe data numerik dan non-numerik campuran, tipe data not_null_return_value dikembalikan.

> **Important**
>
> Dalam dua kasus terakhir di mana tipe data not_null_return_value dikembalikan, null_return_value secara implisit dilemparkan ke tipe data tersebut. Jika tipe data tidak kompatibel, fungsi gagal.

Catatan penggunaan

Untuk NVL2, pengembalian akan memiliki nilai parameter not_null_return_value atau null_return_value, mana saja yang dipilih oleh fungsi, tetapi akan memiliki tipe data not_null_return_value.

Misalnya, dengan asumsi kolom1 adalah NULL, kueri berikut akan mengembalikan nilai yang sama. Namun, tipe data nilai pengembalian DECODE adalah INTEGER dan tipe data nilai yang NVL2 dikembalikan adalah VARCHAR.

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

Contoh

Contoh berikut memodifikasi beberapa data sampel, kemudian mengevaluasi dua bidang untuk memberikan informasi kontak yang sesuai bagi pengguna:

```
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
```

Fungsi NULLIF

Membandingkan dua argumen dan mengembalikan null jika argumen sama. Jika mereka tidak sama, argumen pertama dikembalikan.

Sintaksis

Ekspresi NULLIF membandingkan dua argumen dan mengembalikan null jika argumennya sama. Jika mereka tidak sama, argumen pertama dikembalikan. Ekspresi ini adalah kebalikan dari ekspresi NVL atau COALESCE.

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

Argumen

ekspresi1, ekspresi2

Kolom target atau ekspresi yang dibandingkan. Tipe pengembalian sama dengan tipe ekspresi pertama.

Contoh

Dalam contoh berikut, query mengembalikan string `first` karena argumen tidak sama.

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

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

Dalam contoh berikut, query kembali NULL karena argumen literal string sama.

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

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

Dalam contoh berikut, query kembali 1 karena argumen integer tidak sama.

```
SELECT NULLIF(1, 2);

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

Dalam contoh berikut, query kembali NULL karena argumen integer sama.

```
SELECT NULLIF(1, 1);

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

Dalam contoh berikut, query mengembalikan null ketika nilai LISTID dan SALESID cocok:

```
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)
```

Fungsi konstruktor

Fungsi konstruktor SQL adalah fungsi yang digunakan untuk membuat struktur data baru, seperti array atau peta.

Mereka mengambil beberapa nilai input dan mengembalikan objek struktur data baru. Fungsi konstruktor biasanya dinamai berdasarkan tipe data yang mereka buat, seperti ARRAY atau MAP.

Fungsi konstruktor berbeda dari fungsi skalar atau fungsi agregat, yang beroperasi pada data yang ada dan mengembalikan satu nilai. Fungsi konstruktor digunakan untuk membuat struktur data baru yang kemudian dapat digunakan dalam pemrosesan atau analisis data lebih lanjut.

AWS Clean Rooms mendukung fungsi konstruktor berikut:

Topik

Fungsi konstruktor MAP

Fungsi konstruktor MAP membuat peta dengan pasangan kunci/nilai yang diberikan.

Fungsi konstruktor seperti MAP berguna ketika Anda perlu membuat struktur data baru secara terprogram dalam kueri SQL Anda. Mereka memungkinkan Anda untuk membangun struktur data yang kompleks yang dapat digunakan dalam pemrosesan atau analisis data lebih lanjut.

Sintaks

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

Pendapat

kunci0

Ekspresi dari jenis apa pun yang sebanding. Semua key0 harus berbagi jenis yang paling tidak umum.

nilai0

Ekspresi jenis apa pun. Semua ValueN harus berbagi jenis yang paling tidak umum.

Pengembalian

Fungsi MAP mengembalikan MAP dengan kunci yang diketik sebagai tipe key0 yang paling tidak umum dan nilai yang diketik sebagai tipe value0 yang paling tidak umum.

Contoh

Contoh berikut membuat peta baru dengan dua pasangan kunci-nilai: Kunci `1.0` dikaitkan dengan nilai. `'2'` Kuncinya `3.0` dikaitkan dengan nilai`'4'`. Peta yang dihasilkan kemudian dikembalikan sebagai output dari pernyataan SQL.

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

Fungsi konstruktor NAMED_STRUCT

Fungsi konstruktor NAMED_STRUCT membuat struct dengan nama bidang dan nilai yang diberikan.

Fungsi konstruktor seperti NAMED_STRUCT berguna ketika Anda perlu membuat struktur data baru secara terprogram dalam kueri SQL Anda. Mereka memungkinkan Anda untuk membangun struktur data yang kompleks, seperti struct atau catatan, yang dapat digunakan dalam pemrosesan atau analisis data lebih lanjut.

Sintaks

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

Argumen

nama1

Bidang penamaan literal STRING 1.

val1

Ekspresi dari jenis apa pun yang menentukan nilai untuk bidang 1.

Pengembalian

Fungsi NAMED_STRUCT mengembalikan struct dengan bidang 1 yang cocok dengan jenis val1.

Contoh

Contoh berikut membuat struct baru dengan tiga bidang bernama: `"a"` Bidang diberi nilai1. `"b"`Bidang diberi nilai `2`. `"c"` Bidang diberi nilai3. Struct yang dihasilkan kemudian dikembalikan sebagai output dari pernyataan SQL.

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

Fungsi konstruktor STRUCT

Fungsi konstruktor STRUCT menciptakan struct dengan nilai bidang yang diberikan.

Fungsi konstruktor seperti STRUCT berguna ketika Anda perlu membuat struktur data baru secara terprogram dalam kueri SQL Anda. Mereka memungkinkan Anda untuk membangun struktur data

yang kompleks, seperti struct atau catatan, yang dapat digunakan dalam pemrosesan atau analisis data lebih lanjut.

Sintaks

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

Argumen

col1

Nama kolom atau ekspresi yang valid.

Pengembalian

Fungsi STRUCT mengembalikan struct dengan field1 yang cocok dengan tipe expr1.

Jika argumen diberi nama referensi, nama digunakan untuk memberi nama bidang. Jika tidak, bidang diberi nama ColN, di mana N adalah posisi bidang dalam struct.

Contoh

Contoh berikut membuat struct baru dengan tiga bidang: Bidang pertama diberi nilai 1. Bidang kedua diberi nilai 2. Bidang ketiga diberi nilai 3. Secara default, bidang dalam struct yang dihasilkan diberi nama `col1col2`,`col3`, dan, berdasarkan posisinya dalam daftar argumen. Struct yang dihasilkan kemudian dikembalikan sebagai output dari pernyataan SQL.

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

Fungsi pemformatan tipe data

Menggunakan fungsi pemformatan tipe data, Anda dapat mengonversi nilai dari satu tipe data ke tipe data lainnya. Untuk masing-masing fungsi ini, argumen pertama selalu nilai yang akan diformat dan argumen kedua berisi template untuk format baru.

AWS Clean Rooms Spark SQL mendukung beberapa fungsi pemformatan tipe data.

Topik

- BASE64 fungsi

- Fungsi CAST
- Fungsi DECODE
- Fungsi ENCODE
- Fungsi HEX
- Fungsi STR_TO_MAP
- TO_CHAR
- Fungsi TO_DATE
- TO_NUMBER
- UNBASE64 fungsi
- Fungsi UNHEX
- String format datetime
- String format numerik

BASE64 fungsi

BASE64 Fungsi mengkonversi ekspresi ke string dasar 64 menggunakan pengkodean transfer RFC2045 Base64 untuk MIME.

Sintaksis

```
base64(expr)
```

Argumen

expr

Ekspresi BINER atau STRING yang fungsinya akan ditafsirkan sebagai BINER.

Jenis pengembalian

`STRING`

Contoh

Untuk mengonversi input string yang diberikan menjadi representasi yang dikodekan Base64. gunakan contoh berikut. Hasilnya adalah representasi Base64 yang dikodekan dari string input 'Spark SQL', yaitu 'U3BHCMSGU1fm'.

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

Fungsi CAST

Fungsi CAST mengubah satu tipe data ke tipe data lain yang kompatibel. Misalnya, Anda dapat mengonversi string ke tanggal, atau tipe numerik menjadi string. CAST melakukan konversi runtime, yang berarti konversi tidak mengubah tipe data nilai dalam tabel sumber. Itu hanya berubah dalam konteks kueri.

Tipe data tertentu memerlukan konversi eksplisit ke tipe data lain menggunakan fungsi CAST. Tipe data lain dapat dikonversi secara implisit, sebagai bagian dari perintah lain, tanpa menggunakan CAST. Lihat Ketik kompatibilitas dan konversi.

Sintaksis

Gunakan salah satu dari dua bentuk sintaks yang setara ini untuk mentransmisikan ekspresi dari satu tipe data ke tipe data lainnya.

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

Pendapat

ekspresi

Ekspresi yang mengevaluasi satu atau lebih nilai, seperti nama kolom atau literal. Mengkonversi nilai null mengembalikan nol. Ekspresi tidak dapat berisi string kosong atau kosong.

jenis

Salah satu yang didukungJenis Data, kecuali untuk tipe data BINARY dan BINARY VARIATING.

Jenis pengembalian

CAST mengembalikan tipe data yang ditentukan oleh argumen tipe.

> **Note**
>
> AWS Clean Rooms mengembalikan kesalahan jika Anda mencoba untuk melakukan konversi bermasalah, seperti konversi DECIMAL yang kehilangan presisi, seperti berikut ini:

```
select 123.456::decimal(2,1);
```

atau konversi INTEGER yang menyebabkan overflow:

```
select 12345678::smallint;
```

Contoh

Dua pertanyaan berikut adalah setara. Keduanya memberikan nilai desimal ke bilangan bulat:

```
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)
```

Berikut ini menghasilkan hasil yang serupa. Tidak memerlukan data sampel untuk dijalankan:

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

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

Dalam contoh ini, nilai dalam kolom stempel waktu dilemparkan sebagai tanggal, yang mengakibatkan penghapusan waktu dari setiap hasil:

```
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)
```

Jika Anda tidak menggunakan CAST seperti yang diilustrasikan dalam sampel sebelumnya, hasilnya akan mencakup waktu: 2008-02-18 02:36:48.

Kueri berikut melemparkan data karakter variabel ke tanggal. Tidak memerlukan data sampel untuk dijalankan.

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

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

Dalam contoh ini, nilai dalam kolom tanggal dilemparkan sebagai stempel waktu:

```
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)
```

Dalam kasus seperti sampel sebelumnya, Anda dapat memperoleh kontrol tambahan atas pemformatan output dengan menggunakanTO_CHAR.

Dalam contoh ini, bilangan bulat dilemparkan sebagai string karakter:

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

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

Dalam contoh ini, nilai DECIMAL (6,3) dilemparkan sebagai nilai DECIMAL (4,1):

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

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

Contoh ini menunjukkan ekspresi yang lebih kompleks. Kolom PRICEPAID (kolom DECIMAL (8,2)) dalam tabel PENJUALAN dikonversi ke kolom DECIMAL (38,2) dan nilainya dikalikan dengan 100000000000000000000:

```
select salesid, pricepaid::decimal(38,2)*100000000000000000000
as value from sales where salesid<10 order by salesid;


 salesid |           value
---------+----------------------------
       1 | 72800000000000000000000.00
       2 |  76000000000000000000000.00
       3 | 35000000000000000000000.00
```

```
       4 | 17500000000000000000000.00
       5 | 15400000000000000000000.00
       6 | 39400000000000000000000.00
       7 | 78800000000000000000000.00
       8 | 19700000000000000000000.00
       9 | 59100000000000000000000.00

(9 rows)
```

Fungsi DECODE

Fungsi DECODE adalah mitra dari fungsi ENCODE, yang digunakan untuk mengonversi string ke format biner menggunakan pengkodean karakter tertentu. Fungsi DECODE mengambil data biner dan mengubahnya kembali ke format string yang dapat dibaca menggunakan pengkodean karakter yang ditentukan.

Fungsi ini berguna ketika Anda perlu bekerja dengan data biner yang disimpan dalam database dan perlu menyajikannya dalam format yang dapat dibaca manusia, atau ketika Anda perlu mengonversi data antara pengkodean karakter yang berbeda.

Sintaksis

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

Pendapat

expr

Ekspresi BINER yang dikodekan dalam charset.

charset

Ekspresi STRING.

Pengkodean set karakter yang didukung (case-insensitive):`'US-ASCII'`,,,,`'ISO-8859-1'`, `'UTF-8'` dan. `'UTF-16BE' 'UTF-16LE' 'UTF-16'`

Jenis pengembalian

Fungsi DECODE mengembalikan STRING.

Contoh

Contoh berikut memiliki tabel yang disebut `messages` dengan kolom yang disebut `message_text` yang menyimpan data pesan dalam format biner menggunakan pengkodean karakter UTF-8. Fungsi DECODE mengubah data biner kembali ke format string yang dapat dibaca. Output dari kueri ini adalah teks yang dapat dibaca dari pesan yang disimpan dalam tabel pesan, dengan ID123, dikonversi dari format biner ke string menggunakan `'utf-8'` pengkodean.

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

Fungsi ENCODE

Fungsi ENCODE digunakan untuk mengonversi string ke representasi binernya menggunakan pengkodean karakter tertentu.

Fungsi ini berguna ketika Anda perlu bekerja dengan data biner atau ketika Anda perlu mengonversi antara pengkodean karakter yang berbeda. Misalnya, Anda mungkin menggunakan fungsi ENCODE saat menyimpan data dalam database yang memerlukan penyimpanan biner, atau saat Anda perlu mentransfer data antar sistem yang menggunakan pengkodean karakter yang berbeda.

Sintaksis

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

Pendapat

str

Ekspresi STRING yang akan dikodekan.

charset

Ekspresi STRING yang menentukan pengkodean.

Pengkodean set karakter yang didukung (case-insensitive):`'US-ASCII'`,,,,`'ISO-8859-1'`, `'UTF-8'` dan. `'UTF-16BE' 'UTF-16LE' 'UTF-16'`

Jenis pengembalian

Fungsi ENCODE mengembalikan BINARY.

Contoh

Contoh berikut mengkonversi string `'abc'` ke representasi biner menggunakan `'utf-8'` encoding, yang dalam hal ini menghasilkan string asli dikembalikan. Ini karena `'utf-8'` pengkodean adalah pengkodean karakter lebar variabel yang dapat mewakili seluruh set karakter ASCII (yang mencakup huruf`'a'`,`'b'`, dan`'c'`) menggunakan satu byte per karakter. Oleh karena itu, representasi biner dari `'abc'` penggunaan `'utf-8'` adalah sama dengan string asli.

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

Fungsi HEX

Fungsi HEX mengkonversi nilai numerik (baik integer atau angka floating-point) ke representasi string heksadesimal yang sesuai.

Heksadesimal adalah sistem angka yang menggunakan 16 simbol berbeda (0-9 dan A-F) untuk mewakili nilai numerik. Hal ini umumnya digunakan dalam ilmu komputer dan pemrograman untuk mewakili data biner dalam format yang lebih kompak dan dapat dibaca manusia.

Sintaksis

```
hex(expr)
```

Pendapat

expr

Ekspresi BIGINT, BINER, atau STRING.

Jenis pengembalian

HEX mengembalikan STRING. Fungsi mengembalikan representasi heksadesimal dari argumen.

Contoh

Contoh berikut mengambil nilai integer 17 sebagai input dan menerapkan fungsi HEX () untuk itu. Outputnya adalah11, yang merupakan representasi heksadesimal dari nilai input. 17

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

```
 11
```

Contoh berikut mengkonversi string `'Spark_SQL'` ke representasi heksadesimal nya. Outputnya adalah`537061726B2053514C`, yang merupakan representasi heksadesimal dari string input. `'Spark_SQL'`

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

Dalam contoh ini, string 'Spark_SQL' dikonversi sebagai berikut:

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

Penggabungan nilai heksadesimal ini menghasilkan output akhir ". `537061726B2053514C`"

Fungsi STR_TO_MAP

Fungsi STR_TO_MAP adalah fungsi konversi. string-to-map Ini mengubah representasi string dari peta (atau kamus) menjadi struktur data peta yang sebenarnya.

Fungsi ini berguna ketika Anda perlu bekerja dengan struktur data peta di SQL, tetapi data awalnya disimpan sebagai string. Dengan mengonversi representasi string ke peta yang sebenarnya, Anda kemudian dapat melakukan operasi dan manipulasi pada data peta.

Sintaksis

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

Pendapat

teks

Ekspresi STRING yang mewakili peta.

PairDelim

Sebuah string literal opsional yang menentukan bagaimana untuk memisahkan entri. Ini default ke koma (). `','`

keyValueDelim

Sebuah literal STRING opsional yang menentukan bagaimana memisahkan setiap pasangan kunci-nilai. Ini default ke titik dua (). `':'`

Jenis pengembalian

Fungsi STR_TO_MAP mengembalikan MAP STRING untuk kedua kunci dan nilai. Baik PairDelim dan keyValueDelimdiperlakukan sebagai ekspresi reguler.

Contoh

Contoh berikut mengambil string input dan dua argumen pembatas, dan mengubah representasi string menjadi struktur data peta aktual. Dalam contoh khusus ini, string input `'a:1,b:2,c:3'` mewakili peta dengan pasangan kunci-nilai berikut: `'a'` adalah kuncinya, dan `'1'` nilainya. `'b'`adalah kuncinya, dan `'2'` nilainya. `'c'`adalah kuncinya, dan `'3'` nilainya. `','`Pembatas digunakan untuk memisahkan pasangan kunci-nilai, dan `':'` pembatas digunakan untuk memisahkan kunci dan nilai dalam setiap pasangan. Output dari kueri ini adalah:`{"a":"1","b":"2","c":"3"}`. Ini adalah struktur data peta yang dihasilkan, di mana kuncinya berada `'a''b'`,, dan`'c'`, dan nilai yang sesuai adalah`'1'`,`'2'`, dan`'3'`.

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

Contoh berikut menunjukkan bahwa fungsi STR_TO_MAP mengharapkan string input berada dalam format tertentu, dengan pasangan kunci-nilai dibatasi dengan benar. Jika string input tidak cocok dengan format yang diharapkan, fungsi akan tetap mencoba membuat peta, tetapi nilai yang dihasilkan mungkin tidak seperti yang diharapkan.

```
SELECT str_to_map('a');
```

```
{"a":null}
```

TO_CHAR

TO_CHAR mengkonversi timestamp atau ekspresi numerik ke format data karakter-string.

Sintaksis

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

Argumen

timestamp_expression

Ekspresi yang menghasilkan nilai tipe TIMESTAMP atau TIMESTAMPTZ atau nilai yang secara implisit dapat dipaksa ke stempel waktu.

numeric_expression

Ekspresi yang menghasilkan nilai tipe data numerik atau nilai yang secara implisit dapat dipaksa ke tipe numerik. Untuk informasi selengkapnya, lihat Jenis numerik. TO_CHAR menyisipkan spasi di sebelah kiri string angka.

> **Note**
>
> TO_CHAR tidak mendukung nilai DESIMAL 128-bit.

format

Format untuk nilai baru. Untuk format yang valid, lihat String format datetime danString format numerik.

Jenis pengembalian

VARCHAR

Contoh

Contoh berikut mengubah stempel waktu menjadi nilai dengan tanggal dan waktu dalam format dengan nama bulan empuk menjadi sembilan karakter, nama hari dalam seminggu, dan nomor hari bulan.

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

Contoh berikut mengonversi stempel waktu menjadi nilai dengan nomor hari dalam setahun.

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

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

Contoh berikut mengonversi stempel waktu ke nomor hari ISO dalam seminggu.

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

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

Contoh berikut mengekstrak nama bulan dari tanggal.

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

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

Contoh berikut mengkonversi setiap nilai STARTTIME dalam tabel EVENT ke string yang terdiri dari jam, menit, dan detik.

```
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)
```

Contoh berikut mengkonversi seluruh nilai timestamp ke dalam format yang berbeda.

```
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)
```

Contoh berikut mengkonversi stempel waktu literal ke string karakter.

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

Contoh berikut mengkonversi angka ke string karakter dengan tanda negatif di akhir.

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

Contoh berikut mengkonversi angka ke string karakter dengan simbol mata uang.

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

Contoh berikut mengkonversi angka ke string karakter menggunakan kurung sudut untuk angka negatif.

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

Contoh berikut mengkonversi angka ke string angka Romawi.

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

Contoh berikut menampilkan hari dalam seminggu.

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

Contoh berikut menampilkan akhiran nomor urut untuk angka.

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

Contoh berikut mengurangi komisi dari harga yang dibayarkan dalam tabel penjualan. Perbedaannya kemudian dibulatkan dan diubah menjadi angka romawi, yang ditunjukkan pada `to_char` kolom:

```
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)
```

Contoh berikut menambahkan simbol mata uang ke nilai selisih yang ditunjukkan pada `to_char` kolom:

```
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)
```

Contoh berikut mencantumkan abad di mana setiap penjualan dilakukan.

```
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)
```

Contoh berikut mengkonversi setiap nilai STARTTIME dalam tabel EVENT ke string yang terdiri dari jam, menit, detik, dan zona waktu.

```
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)
```

Contoh berikut menunjukkan pemformatan untuk detik, milidetik, dan mikrodetik.

```
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
```

Fungsi TO_DATE

TO_DATE mengonversi tanggal yang diwakili oleh string karakter ke tipe data DATE.

Sintaksis

```
TO_DATE (date_str)
```

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

Argumen

date_str

Sebuah string tanggal atau tipe data yang dapat dilemparkan ke dalam string tanggal.

format

String literal yang cocok dengan pola datetime Spark. Untuk pola datetime yang valid, lihat Pola Datetime untuk Pemformatan dan Penguraian.

Jenis pengembalian

TO_DATE mengembalikan DATE, tergantung pada nilai format.

Jika konversi ke format gagal, maka kesalahan dikembalikan.

Contoh

Pernyataan SQL berikut mengubah tanggal `02 Oct 2001` menjadi tipe data tanggal.

```
select to_date('02 Oct 2001', 'dd MMM yyyy');

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

Pernyataan SQL berikut mengkonversi string `20010631` ke tanggal.

```
select to_date('20010631', 'yyyyMMdd');
```

Pernyataan SQL berikut mengkonversi string `20010631` ke tanggal:

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

Hasilnya adalah nilai nol karena hanya ada 30 hari di bulan Juni.

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

TO_NUMBER

TO_NUMBER mengkonversi string ke nilai numerik (desimal).

Sintaksis

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

Argumen

tali

String yang akan dikonversi. Formatnya harus berupa nilai literal.

format

Argumen kedua adalah string format yang menunjukkan bagaimana string karakter harus diurai untuk membuat nilai numerik. Misalnya, format `'99D999'` menentukan bahwa string yang akan dikonversi terdiri dari lima digit dengan titik desimal di posisi ketiga. Misalnya, `to_number('12.345','99D999')` kembali `12.345` sebagai nilai numerik. Untuk daftar format yang valid, lihatString format numerik.

Jenis pengembalian

TO_NUMBER mengembalikan nomor DECIMAL.

Jika konversi ke format gagal, maka kesalahan dikembalikan.

Contoh

Contoh berikut mengkonversi string `12,454.8-` ke nomor:

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

to_number
```

```
-----------
-12454.8
```

Contoh berikut mengkonversi string `$ 12,454.88` ke nomor:

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

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

Contoh berikut mengkonversi string `$ 2,012,454.88` ke nomor:

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

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

UNBASE64 fungsi

UNBASE64 Fungsi mengkonversi argumen dari string dasar 64 ke biner.

Pengkodean Base64 umumnya digunakan untuk mewakili data biner (seperti gambar, file, atau informasi terenkripsi) dalam format tekstual yang aman untuk transmisi melalui berbagai saluran komunikasi (seperti email, parameter URL, atau penyimpanan basis data).

UNBASE64 Fungsi ini memungkinkan Anda untuk membalikkan proses ini dan memulihkan data biner asli. Jenis fungsi ini dapat berguna dalam skenario di mana Anda perlu bekerja dengan data yang telah dikodekan dalam format Base64, seperti saat mengintegrasikan dengan sistem eksternal atau APIs yang menggunakan Base64 sebagai mekanisme transfer data.

Sintaksis

```
unbase64(expr)
```

Pendapat

expr

Ekspresi STRING dalam format base64.

Jenis pengembalian

`BINARY`

Contoh

Dalam contoh berikut, string yang dikodekan `'U3BhcmsgU1FM'` Base64-dikonversi kembali ke string asli. `'Spark SQL'`

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

Fungsi UNHEX

Fungsi UNHEX mengkonversi string heksadesimal kembali ke representasi string aslinya.

Fungsi ini dapat berguna dalam skenario di mana Anda perlu bekerja dengan data yang telah disimpan atau ditransmisikan dalam format heksadesimal, dan Anda perlu mengembalikan representasi string asli untuk diproses atau ditampilkan lebih lanjut.

Fungsi UNHEX adalah mitra dari fungsi HEX.

Sintaksis

```
unhex(expr)
```

Pendapat

expr

Ekspresi STRING dari karakter heksadesimal.

Jenis pengembalian

UNHEX mengembalikan BINARY.

Jika panjang expr ganjil, karakter pertama dibuang dan hasilnya dilapisi dengan byte nol. Jika expr berisi karakter non hex hasilnya adalah NULL.

Contoh

Contoh berikut mengkonversi string heksadesimal kembali ke representasi string aslinya dengan menggunakan UNHEX () dan DECODE () fungsi bersama-sama. Bagian pertama dari kueri,

menggunakan fungsi UNHEX () untuk mengonversi string heksadesimal '537061726B2053514C' menjadi representasi binernya. Bagian kedua dari kueri, menggunakan fungsi DECODE () untuk mengonversi data biner yang diperoleh dari fungsi UNHEX () kembali ke string, menggunakan pengkodean karakter 'UTF-8'. Output dari query, adalah string asli 'Spark_SQL' yang dikonversi ke heksadesimal dan kemudian kembali ke string.

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

String format datetime

Anda dapat menggunakan pola datetime dalam skenario umum berikut:

- Saat bekerja dengan sumber data CSV dan JSON untuk mengurai dan memformat konten datetime
- Saat mengonversi antara tipe string dan tipe tanggal atau stempel waktu menggunakan fungsi seperti:
  - unix_stempel waktu
  - date_format
  - to_unix_timestamp
  - dari_unixtime
  - to_date
  - to_timestamp
  - dari_utc_timestamp
  - to_utc_timestamp

Gunakan huruf pola dalam tabel berikut untuk penguraian dan pemformatan tanggal dan stempel waktu.

Datepart atau timepart	Arti	Contoh
a	AM atau PM hari ini, disajikan sebagai am-pm	PM
D	Hari dalam setahun, disajikan sebagai angka 3 digit	189

Datepart atau timepart	Arti	Contoh
d	Hari dalam sebulan, disajikan sebagai angka 2 digit	28
E	Hari dalam seminggu, disajikan sebagai teks	Sel Selasa
F	Hari selaras dalam seminggu di bulan itu, disajikan sebagai angka 1 digit	3
G	Indikator era, disajikan sebagai teks	AD Anno Domini
-h	Jam-jam AM atau PM, disajikan sebagai angka 2 digit	12
H	Jam sehari, disajikan sebagai angka 2 digit dari 0-23	0
k	Jam-jam sehari, disajikan sebagai angka 2 digit dari 1-24	1
K	Jam AM atau PM, disajikan sebagai angka 2 digit dari 0-11	0
m	Menit jam, disajikan sebagai angka 2 digit	30
M/L	Bulan dalam setahun, disajikan sebagai bulan	7 07 Juli Juli

Datepart atau timepart	Arti	Contoh
O	Offset zona lokal dari UTC	GMT+8 GMT+ 8:00 UTC- 08:00
Q/Q	Kuartal tahun ini, disajikan sebagai angka (1 hingga 4) atau teks	3 03 Q3 Kuartal ke-3
detik	Detik menit, disajikan sebagai angka 2 digit	55
D	Fraksi detik, disajikan sebagai pecahan	978
V	Pengidentifikasi zona waktu, disajikan sebagai id zona	Amerika/Los_Angeles Z 08:30
x	Zona offset dari UTC (Offset-X)	+0000 -08 -0830 - 08:30 -083015 - 08:30:15

Datepart atau timepart	Arti	Contoh
X	Zona offset dari UTC; di mana Z adalah nol	Z -08 -0830 - 08:30 -083015 - 08:30:15
y	Tahun, disajikan sebagai tahun	2020 20
z	Nama zona waktu, disajikan sebagai teks	Waktu Standar Pasifik PST
Z	Zona offset dari UTC (Offset-z)	+0000 -0800 - 08:00
'	Melarikan diri untuk teks, disajikan sebagai pembatas	N/A
''	Kutipan tunggal, disajikan sebagai literal	'
[	Bagian opsional mulai	N/A
]	Bagian akhir opsional	N/A

Jumlah huruf pola menentukan jenis format:

Format Teks

- Gunakan 1-3 huruf untuk formulir yang disingkat (misalnya, “Senin” untuk hari Senin)
- Gunakan tepat 4 huruf untuk formulir lengkap (misalnya, “Senin”)
- Jangan gunakan 5 huruf atau lebih - ini akan menyebabkan kesalahan

Format Angka (n)

- Nilai n mewakili jumlah maksimum huruf yang diizinkan
- Untuk pola huruf tunggal:
  - Output menggunakan digit minimum tanpa padding
- Untuk beberapa pola huruf:
  - Output dilapisi dengan nol agar sesuai dengan lebar hitungan huruf
- Saat mengurai, input harus berisi jumlah digit yang tepat

Format Nomor/Teks

- Untuk 3 huruf atau lebih, ikuti aturan Format Teks
- Untuk huruf yang lebih sedikit, ikuti aturan Format Angka

Format Pecahan

- Gunakan 1-9 karakter 'S' (misalnya, SSSSSS)
- Untuk parsing:
  - Terima pecahan antara 1 dan jumlah karakter S
- Untuk memformat:
  - Pad dengan nol untuk mencocokkan jumlah karakter S
- Mendukung hingga 6 digit untuk presisi mikrodetik
- Dapat mengurai nanodetik tetapi memotong digit ekstra

Format Tahun

- Jumlah huruf menetapkan lebar bidang minimum untuk padding
- Untuk dua huruf:
  - Mencetak dua digit terakhir

  - Mengurai tahun antara 2000-2099
- Untuk kurang dari empat huruf (kecuali dua):
  - Menunjukkan tanda hanya untuk tahun-tahun negatif
- Jangan gunakan 7 huruf atau lebih - ini akan menyebabkan kesalahan

Format Bulan

- Gunakan 'M' untuk bentuk standar atau 'L' untuk bentuk mandiri
- Tunggal 'M' atau 'L':
  - Menunjukkan nomor bulan 1-12 tanpa padding

- 'MM' atau 'LL':
  - Menunjukkan nomor bulan 01-12 dengan padding
- 'MMM':
  - Menampilkan nama bulan yang disingkat dalam bentuk standar
  - Harus menjadi bagian dari pola tanggal lengkap
- 'LLL':
  - Menampilkan nama bulan yang disingkat dalam bentuk mandiri
  - Gunakan untuk pemformatan hanya bulan
- 'MMMM':
  - Menampilkan nama bulan penuh dalam bentuk standar
  - Gunakan untuk tanggal dan stempel waktu
- 'LLLL':
  - Menampilkan nama bulan penuh dalam bentuk mandiri
  - Gunakan untuk pemformatan hanya bulan

Format Zona Waktu

- am-pm: Gunakan 1 huruf saja
- ID Zona (V): Gunakan 2 huruf saja
- Nama zona (z):
  - 1-3 huruf: Menunjukkan nama pendek

- 4 huruf: Menunjukkan nama lengkap
- Jangan gunakan 5 huruf atau lebih

Format Offset

- X dan X:
  - 1 huruf: Menunjukkan jam (+01) atau jam-menit (+0130)
  - 2 huruf: Menunjukkan jam-menit tanpa titik dua (+0130)
  - 3 huruf: Menunjukkan jam-menit dengan titik dua (+ 01:30)
  - 4 huruf: Menunjukkan hour-minute-second tanpa titik dua (+013015)
  - 5 huruf: Menunjukkan hour-minute-second dengan titik dua (+ 01:30:15)
  - X menggunakan 'Z' untuk offset nol
  - x menggunakan '+00', '+0000', atau '+ 00:00 'untuk offset nol
- O:
  - 1 huruf: Menunjukkan formulir pendek (GMT+8)
  - 4 huruf: Menunjukkan formulir lengkap (GMT+08:00)
- Z:
  - 1-3 huruf: Menunjukkan jam-menit tanpa titik dua (+0130)
  - 4 huruf: Menunjukkan formulir lokal lengkap
  - 5 huruf: Menunjukkan hour-minute-second dengan titik dua

Bagian opsional

- Gunakan tanda kurung siku [] untuk menandai konten opsional
- Anda dapat membuat sarang bagian opsional
- Semua data yang valid muncul di output
- Masukan dapat menghilangkan seluruh bagian opsional

> **Note**
>
> Simbol 'E', 'F', 'q', dan 'Q' hanya berfungsi untuk pemformatan datetime (seperti date_format). Jangan gunakan mereka untuk penguraian datetime (seperti to_timestamp).

String format numerik

String format numerik berikut berlaku untuk fungsi seperti TO_NUMBER dan TO_CHAR.

- Untuk contoh memformat string sebagai angka, lihat. TO_NUMBER
- Untuk contoh memformat angka sebagai string, lihat. TO_CHAR

Format	Deskripsi
9	Nilai numerik dengan jumlah digit yang ditentukan.
0	Nilai numerik dengan angka nol di depan.
. (periode), D	Titik desimal.
, (koma)	Ribuan pemisah.
CC	Kode abad. Misalnya, abad ke-21 dimulai pada 2001-01-01 (hanya didukung untuk TO_CHAR).
FM	Mode isi. Menekan padding kosong dan nol.
PR	Nilai negatif dalam kurung sudut.
D	Tanda tangani berlabuh ke nomor.
L	Simbol mata uang di posisi yang ditentukan.
G	Pemisah kelompok.
MI	Tanda minus di posisi yang ditentukan untuk angka yang kurang dari 0.
PL	Ditambah tanda di posisi yang ditentukan untuk angka yang lebih besar dari 0.
SG	Tanda plus atau minus di posisi yang ditentuka n.

Format	Deskripsi
RN	Angka romawi antara 1 dan 3999 (hanya didukung untuk TO_CHAR).
TH atau th	Akhiran nomor urut. Tidak mengubah angka pecahan atau nilai yang kurang dari nol.

Fungsi tanggal dan waktu

Fungsi tanggal dan waktu memungkinkan Anda melakukan berbagai operasi pada data tanggal dan waktu, seperti mengekstrak bagian tanggal, melakukan perhitungan tanggal, memformat tanggal dan waktu, dan bekerja dengan tanggal dan waktu saat ini. Fungsi-fungsi ini penting untuk tugas-tugas seperti analisis data, pelaporan, dan manipulasi data yang melibatkan data temporal.

AWS Clean Rooms mendukung fungsi tanggal dan waktu berikut:

Topik

- Fungsi MONTH
- Fungsi KEDUA
- Fungsi TIMESTAMP
- Fungsi TO_TIMESTAMP
- Fungsi YEAR
- Bagian tanggal untuk fungsi tanggal atau stempel waktu

Fungsi ADD_MONTHS

ADD_MONTHS menambahkan jumlah bulan yang ditentukan ke nilai atau ekspresi tanggal atau stempel waktu. DATE_ADDFungsi ini menyediakan fungsionalitas serupa.

Sintaksis

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

Argumen

tanggal | stempel waktu

Kolom tanggal atau stempel waktu atau ekspresi yang secara implisit mengkonversi ke tanggal atau stempel waktu. Jika tanggal adalah hari terakhir bulan itu, atau jika bulan yang dihasilkan lebih pendek, fungsi mengembalikan hari terakhir bulan dalam hasilnya. Untuk tanggal lain, hasilnya berisi nomor hari yang sama dengan ekspresi tanggal.

bilangan bulat

Sebuah bilangan bulat positif atau negatif. Gunakan angka negatif untuk mengurangi bulan dari tanggal.

Jenis pengembalian

TIMESTAMP

Contoh

Query berikut menggunakan fungsi ADD_MONTHS di dalam fungsi TRUNC. Fungsi TRUNC menghapus waktu hari dari hasil ADD_MONTHS. Fungsi ADD_MONTHS menambahkan 12 bulan ke setiap nilai dari kolom 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)
```

Contoh berikut menunjukkan perilaku ketika fungsi ADD_MONTHS beroperasi pada tanggal dengan bulan yang memiliki jumlah hari yang berbeda.

```
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)
```

Fungsi CONVERT_TIMEZONE

CONVERT_TIMEZONE mengonversi stempel waktu dari satu zona waktu ke zona waktu lainnya. Fungsi ini secara otomatis menyesuaikan waktu musim panas.

Sintaksis

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

Argumen

source_timezone

(Opsional) Zona waktu stempel waktu saat ini. Defaultnya adalah UTC.

target_zona waktu

Zona waktu untuk stempel waktu baru.

stempel waktu

Kolom timestamp atau ekspresi yang secara implisit mengkonversi ke stempel waktu.

Jenis pengembalian

TIMESTAMP

Contoh

Contoh berikut mengkonversi nilai timestamp dari zona waktu UTC default untuk PST.

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

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

Contoh berikut mengkonversi nilai timestamp dalam kolom LISTTIME dari zona waktu UTC default ke PST. Meskipun stempel waktu berada dalam periode waktu siang hari, itu diubah menjadi waktu standar karena zona waktu target ditentukan sebagai singkatan (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
```

Contoh berikut mengonversi timestamp kolom LISTTIME dari zona waktu UTC default ke zona waktu. US/Pacific Zona waktu target menggunakan nama zona waktu, dan stempel waktu berada dalam periode waktu siang hari, sehingga fungsi mengembalikan waktu siang hari.

```
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
```

Contoh berikut mengkonversi string timestamp dari EST ke PST:

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

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

Contoh berikut mengubah stempel waktu ke Waktu Standar Timur AS karena zona waktu target menggunakan nama zona waktu (America/New_York) dan stempel waktu berada dalam periode waktu standar.

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

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

Contoh berikut mengubah stempel waktu menjadi US Eastern Daylight Time karena zona waktu target menggunakan nama zona waktu (America/New_York) dan stempel waktu berada dalam periode waktu siang hari.

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

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

Contoh berikut menunjukkan penggunaan offset.

```
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)
```

Fungsi CURRENT_DATE

CURRENT_DATE mengembalikan tanggal di zona waktu sesi saat ini (UTC secara default) dalam format default:. YYYY-MM-DD

> **Note**
>
> CURRENT_DATE mengembalikan tanggal mulai untuk transaksi saat ini, bukan untuk awal pernyataan saat ini. Pertimbangkan skenario di mana Anda memulai transaksi yang berisi beberapa pernyataan pada 10/01/08 23:59, dan pernyataan yang berisi CURRENT_DATE berjalan pada 10/02/08 00:00. CURRENT_DATE kembali`10/01/08`, tidak. `10/02/08`

Sintaksis

```
CURRENT_DATE
```

Jenis pengembalian

DATE

Contoh

Contoh berikut mengembalikan tanggal saat ini (di Wilayah AWS mana fungsi berjalan).

```
select current_date;

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

Fungsi CURRENT_TIMESTAMP

CURRENT_TIMESTAMP mengembalikan tanggal dan waktu saat ini, termasuk tanggal, waktu, dan (opsional) milidetik atau mikrodetik.

Fungsi ini berguna ketika Anda perlu mendapatkan tanggal dan waktu saat ini, misalnya, untuk merekam stempel waktu suatu peristiwa, untuk melakukan perhitungan berbasis waktu, atau untuk mengisi kolom. date/time

Sintaksis

```
current_timestamp()
```

Jenis pengembalian

Fungsi CURRENT_TIMESTAMP mengembalikan DATE.

Contoh

Contoh berikut mengembalikan tanggal dan waktu saat ini pada saat kueri dijalankan, yaitu 25 April 2020, pukul 15:49:11.914 (3:49:11.914 PM).

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

Contoh berikut mengambil tanggal dan waktu saat ini untuk setiap baris dalam `squirrels` tabel.

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

Fungsi DATE_ADD

Mengembalikan tanggal yang num_days setelah start_date.

Sintaksis

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

Argumen

start_date

Nilai tanggal mulai.

num_days

Jumlah hari untuk menambahkan (integer). Angka positif menambahkan hari, angka negatif mengurangi hari.

Jenis pengembalian

DATE

Contoh

Contoh berikut menambahkan satu hari ke tanggal:

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

Result:
2016-07-31
```

Contoh berikut menambahkan beberapa hari.

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

Result:
2016-08-04
```

Catatan penggunaan

Dokumentasi ini untuk fungsi DATE_ADD Spark SQL, yang menyediakan antarmuka yang lebih sederhana untuk menambahkan hari ke tanggal dibandingkan dengan beberapa varian SQL lainnya. Untuk menambahkan interval lain seperti bulan atau tahun, fungsi yang berbeda mungkin diperlukan.

Fungsi DATE_DIFF

DATE_DIFF mengembalikan perbedaan antara bagian tanggal dari dua ekspresi tanggal atau waktu.

Sintaksis

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

Argumen

EndDate

Ekspresi DATE.

StartDate

Ekspresi DATE.

Jenis pengembalian

BIGINT

Contoh dengan kolom DATE

Contoh berikut menemukan perbedaan, dalam jumlah minggu, antara dua nilai tanggal literal.

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

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

Contoh berikut menemukan perbedaan, dalam jam, antara dua nilai tanggal literal. Ketika Anda tidak memberikan nilai waktu untuk tanggal, defaultnya adalah 00:00:00.

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

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

Contoh berikut menemukan perbedaan, dalam hari, antara dua nilai TIMESTAMETZ literal.

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

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

Contoh berikut menemukan perbedaan, dalam hari, antara dua tanggal dalam baris tabel yang sama.

```
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)
```

Contoh berikut menemukan perbedaan, dalam jumlah kuartal, antara nilai literal di masa lalu dan tanggal hari ini. Contoh ini mengasumsikan bahwa tanggal saat ini adalah 5 Juni 2008. Anda dapat memberi nama bagian tanggal secara lengkap atau menyingkatnya. Nama kolom default untuk fungsi DATE_DIFF adalah DATE_DIFF.

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

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

Contoh berikut bergabung dengan tabel PENJUALAN dan DAFTAR untuk menghitung berapa hari setelah mereka terdaftar, tiket apa pun dijual untuk daftar 1000 hingga 1005. Penantian terpanjang untuk penjualan daftar ini adalah 15 hari, dan yang terpendek kurang dari satu hari (0 hari).

```
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)
```

Contoh ini menghitung jumlah rata-rata jam penjual menunggu semua penjualan tiket.

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

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

Contoh dengan kolom TIME

Berikut contoh tabel TIME_TEST memiliki kolom TIME_VAL (tipe TIME) dengan tiga nilai dimasukkan.

```
select time_val from time_test;

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

Contoh berikut menemukan perbedaan jumlah jam antara kolom TIME_VAL dan literal waktu.

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

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

```
        15
```

Contoh berikut menemukan perbedaan jumlah menit antara dua nilai waktu literal.

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

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

Contoh dengan kolom TIMETZ

Contoh tabel berikut TIMETZ_TEST memiliki kolom TIMETZ_VAL (tipe TIMETZ) dengan tiga nilai dimasukkan.

```
select timetz_val from timetz_test;

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

Contoh berikut menemukan perbedaan jumlah jam, antara literal TIMETZ dan timetz_val.

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

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

Contoh berikut menemukan perbedaan jumlah jam, antara dua nilai TIMETZ literal.

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

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

Fungsi DATE_PART

DATE_PART mengekstrak nilai bagian tanggal dari ekspresi. DATE_PART adalah sinonim dari fungsi PGDATE_PART.

Sintaksis

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

Argumen

lapangan

Bagian mana dari sumber yang harus diekstraksi, dan nilai string yang didukung sama dengan bidang fungsi ekivalen EXTRACT.

sumber

Kolom DATE atau INTERVAL dari mana bidang harus diekstraksi.

Jenis pengembalian

Jika bidang adalah 'KEDUA', DESIMAL (8, 6). Dalam semua kasus lain, INTEGER.

Contoh

Contoh berikut mengekstrak hari tahun (DOY) dari nilai tanggal. Output menunjukkan bahwa hari dalam setahun untuk tanggal “2019-08-12" adalah. 224 Ini berarti bahwa 12 Agustus 2019 adalah hari ke 224 tahun 2019.

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

Fungsi DATE_TRUNC

Fungsi DATE_TRUNC memotong ekspresi stempel waktu atau literal berdasarkan bagian tanggal yang Anda tentukan, seperti jam, hari, atau bulan.

Sintaksis

```
date_trunc(format, datetime)
```

Argumen

format

Format yang mewakili unit yang akan dipotong. Format yang valid adalah sebagai berikut:

- “TAHUN”, “YYYY”, “YY” - potong ke tanggal pertama tahun dimana ts jatuh, bagian waktu akan menjadi nol
- “QUARTER” - potong ke tanggal pertama kuartal tempat ts jatuh, bagian waktu akan menjadi nol
- “BULAN”, “MM”, “MON” - potong ke tanggal pertama bulan dimana ts jatuh, bagian waktu akan menjadi nol
- “MINGGU” - potong ke hari Senin dalam seminggu di mana ts jatuh, bagian waktu akan menjadi nol
- “DAY”, “DD” - nol bagian waktu
- “JAM” - nol menit dan detik dengan bagian fraksi
- “MINUTE” - nol yang kedua dengan bagian fraksi
- “KEDUA” - nol bagian fraksi kedua
- “MILLISECOND” - nol mikrodetik
- “MICROSECOND” - semuanya tetap

ts

Nilai datetime

Jenis pengembalian

Mengembalikan stempel waktu ts terpotong ke unit yang ditentukan oleh model format

Contoh

Contoh berikut memotong nilai tanggal ke awal tahun. Output menunjukkan bahwa tanggal “2015-03-05" telah dipotong menjadi “2015-01-01", yang merupakan awal tahun 2015.

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

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

Fungsi DAY

Fungsi DAY mengembalikan hari bulan tanggal/timestamp.

Fungsi ekstraksi tanggal berguna ketika Anda perlu bekerja dengan komponen tertentu dari tanggal atau stempel waktu, seperti saat melakukan perhitungan berbasis tanggal, memfilter data, atau memformat nilai tanggal.

Sintaksis

```
day(date)
```

Pendapat

tanggal

Ekspresi DATE atau TIMESTAMP.

Pengembalian

Fungsi DAY mengembalikan INTEGER.

Contoh

Contoh berikut mengekstrak hari bulan (`30`) dari tanggal input`'2009-07-30'`.

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

Contoh berikut mengekstrak hari bulan dari `birthday` kolom `squirrels` tabel dan mengembalikan hasil sebagai output dari pernyataan SELECT. Output dari kueri ini akan menjadi daftar nilai hari, satu untuk setiap baris dalam `squirrels` tabel, mewakili hari dalam sebulan untuk ulang tahun setiap tupai.

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

Fungsi DAYOFMONTH

Fungsi DAYOFMONTH mengembalikan hari dari bulan date/timestamp (nilai antara 1 dan 31, tergantung pada bulan dan tahun).

Fungsi DAYOFMONTH mirip dengan fungsi DAY, tetapi mereka memiliki nama yang sedikit berbeda dan perilaku yang sedikit berbeda. Fungsi DAY lebih umum digunakan, tetapi fungsi DAYOFMONTH dapat digunakan sebagai alternatif. Jenis kueri ini dapat berguna ketika Anda perlu melakukan analisis berbasis tanggal atau pemfilteran pada tabel yang berisi data tanggal atau stempel waktu, seperti mengekstrak komponen tertentu dari tanggal untuk diproses atau dilaporkan lebih lanjut.

Sintaksis

```
dayofmonth(date)
```

Pendapat

tanggal

Ekspresi DATE atau TIMESTAMP.

Pengembalian

Fungsi DAYOFMONTH mengembalikan INTEGER.

Contoh

Contoh berikut mengekstrak hari bulan (`30`) dari tanggal input `'2009-07-30'`.

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

Contoh berikut menerapkan fungsi DAYOFMONTH ke `birthday` kolom tabel. `squirrels` Untuk setiap baris dalam `squirrels` tabel, hari bulan dari `birthday` kolom akan diekstraksi dan dikembalikan sebagai output dari pernyataan SELECT. Output dari kueri ini akan menjadi daftar nilai hari, satu untuk setiap baris dalam `squirrels` tabel, mewakili hari dalam sebulan untuk ulang tahun setiap tupai.

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

Fungsi DAYOFWEEK

Fungsi DAYOFWEEK mengambil tanggal atau stempel waktu sebagai input dan mengembalikan hari dalam seminggu sebagai angka (1 untuk Minggu, 2 untuk Senin,..., 7 untuk Sabtu).

Fungsi ekstraksi tanggal ini berguna ketika Anda perlu bekerja dengan komponen tertentu dari tanggal atau stempel waktu, seperti saat melakukan perhitungan berbasis tanggal, memfilter data, atau memformat nilai tanggal.

Sintaksis

```
dayofweek(date)
```

Pendapat

tanggal

Ekspresi DATE atau TIMESTAMP.

Pengembalian

Fungsi DAYOFWEEK mengembalikan INTEGER dimana

1 = Minggu

2 = Senin

3 = Selasa

4 = Rabu

5 = Kamis

6 = Jumat

7 = Sabtu

Contoh

Contoh berikut mengekstrak hari dalam seminggu dari tanggal ini, yaitu 5 (mewakili Kamis).

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

Contoh berikut mengekstrak hari dalam seminggu dari `birthday` kolom `squirrels` tabel dan mengembalikan hasil sebagai output dari pernyataan SELECT. Output dari kueri ini akan menjadi

daftar nilai hari dalam seminggu, satu untuk setiap baris dalam `squirrels` tabel, mewakili hari dalam seminggu untuk ulang tahun setiap tupai.

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

Fungsi DAYOFYEAR

Fungsi DAYOFYEAR adalah fungsi ekstraksi tanggal yang mengambil tanggal atau stempel waktu sebagai input dan mengembalikan hari dalam setahun (nilai antara 1 dan 366, tergantung pada tahun dan apakah itu tahun kabisat).

Fungsi ini berguna ketika Anda perlu bekerja dengan komponen tertentu dari tanggal atau stempel waktu, seperti saat melakukan perhitungan berbasis tanggal, memfilter data, atau memformat nilai tanggal.

Sintaksis

```
dayofyear(date)
```

Pendapat

tanggal

Ekspresi DATE atau TIMESTAMP.

Pengembalian

Fungsi DAYOFYEAR mengembalikan INTEGER (antara 1 dan 366, tergantung pada tahun dan apakah itu tahun kabisat).

Contoh

Contoh berikut mengekstrak hari tahun (`100`) dari tanggal input`'2016-04-09'`.

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

Contoh berikut mengekstrak hari dalam setahun dari `birthday` kolom `squirrels` tabel dan mengembalikan hasil sebagai output dari pernyataan SELECT.

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

Fungsi EKSTRAK

Fungsi EXTRACT mengembalikan bagian tanggal atau waktu dari nilai TIMESTAMP, TIMESTAMPTZ, TIME, atau TIMETZ. Contohnya termasuk hari, bulan, tahun, jam, menit, detik, milidetik, atau mikrodetik dari stempel waktu.

Sintaksis

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

Argumen

datepart

Subbidang tanggal atau waktu untuk mengekstrak, seperti hari, bulan, tahun, jam, menit, detik, milidetik, atau mikrodetik. Untuk nilai yang mungkin, lihatBagian tanggal untuk fungsi tanggal atau stempel waktu.

sumber

Kolom atau ekspresi yang mengevaluasi tipe data TIMESTAMP, TIMESTAMPTZ, TIME, atau TIMETZ.

Jenis pengembalian

INTEGER jika nilai sumber mengevaluasi tipe data TIMESTAMP, TIME, atau TIMETZ.

PRESISI GANDA jika nilai sumber mengevaluasi tipe data TIMESTAMPTZ.

Contoh dengan waktu

Berikut contoh tabel TIME_TEST memiliki kolom TIME_VAL (tipe TIME) dengan tiga nilai dimasukkan.

```
select time_val from time_test;

time_val
---------------------
20:00:00
```

```
00:00:00.5550
00:58:00
```

Contoh berikut mengekstrak menit dari setiap time_val.

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

Contoh berikut mengekstrak jam dari setiap time_val.

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

hours
-----------
         20
         0
         0
```

Fungsi FROM_UTC_TIMESTAMP

Fungsi FROM_UTC_TIMESTAMP mengubah tanggal input dari UTC (Coordinated Universal Time) ke zona waktu yang ditentukan.

Fungsi ini berguna ketika Anda perlu mengonversi nilai tanggal dan waktu dari UTC ke zona waktu tertentu. Ini bisa menjadi penting ketika bekerja dengan data yang berasal dari berbagai belahan dunia dan perlu disajikan dalam waktu setempat yang sesuai.

Sintaksis

```
from_utc_timestamp(timestamp, timezone
```

Pendapat

stempel waktu

Ekspresi TIMESTAMP dengan stempel waktu UTC.

zona waktu

Ekspresi STRING yang merupakan zona waktu yang valid dimana tanggal input atau stempel waktu harus dikonversi.

Pengembalian

Fungsi FROM_UTC_TIMESTAMP mengembalikan TIMESTAMP.

Contoh

Contoh berikut mengubah tanggal input dari UTC ke zona waktu yang ditentukan (`'Asia/Seoul'`), yang dalam hal ini adalah 9 jam lebih awal dari UTC. Output yang dihasilkan adalah tanggal dan waktu di zona waktu Seoul, yaitu`2016-08-31 09:00:00`.

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

Fungsi HOUR

Fungsi HOUR adalah fungsi ekstraksi waktu yang membutuhkan waktu atau stempel waktu sebagai input dan mengembalikan komponen jam (nilai antara 0 dan 23).

Fungsi ekstraksi waktu ini berguna ketika Anda perlu bekerja dengan komponen waktu atau stempel waktu tertentu, seperti saat melakukan perhitungan berbasis waktu, memfilter data, atau memformat nilai waktu.

Sintaksis

```
hour(timestamp)
```

Pendapat

stempel waktu

Ekspresi TIMESTAMP.

Pengembalian

Fungsi HOUR mengembalikan INTEGER.

Contoh

Contoh berikut mengekstrak komponen jam (12) dari stempel waktu `'2009-07-30 12:58:59'` masukan.

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

Fungsi MINUTE

Fungsi MINUTE adalah fungsi ekstraksi waktu yang membutuhkan waktu atau stempel waktu sebagai input dan mengembalikan komponen menit (nilai antara 0 dan 60).

Sintaksis

```
minute(timestamp)
```

Pendapat

stempel waktu

Ekspresi TIMESTAMP atau STRING dari format stempel waktu yang valid.

Pengembalian

Fungsi MINUTE mengembalikan INTEGER.

Contoh

Contoh berikut mengekstrak komponen menit (58) dari stempel waktu `'2009-07-30 12:58:59'` masukan.

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

Fungsi MONTH

Fungsi MONTH adalah fungsi ekstraksi waktu yang membutuhkan waktu atau stempel waktu sebagai input dan mengembalikan komponen bulan (nilai antara 0 dan 12).

Sintaksis

```
month(date)
```

Pendapat

tanggal

Ekspresi TIMESTAMP atau STRING dari format stempel waktu yang valid.

Pengembalian

Fungsi MONTH mengembalikan INTEGER.

Contoh

Contoh berikut mengekstrak komponen bulan (7) dari stempel waktu `'2016-07-30'` masukan.

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

Fungsi KEDUA

Fungsi KEDUA adalah fungsi ekstraksi waktu yang membutuhkan waktu atau stempel waktu sebagai input dan mengembalikan komponen kedua (nilai antara 0 dan 60).

Sintaksis

```
second(timestamp)
```

Pendapat

stempel waktu

Ekspresi TIMESTAMP.

Pengembalian

Fungsi KEDUA mengembalikan INTEGER.

Contoh

Contoh berikut mengekstrak komponen kedua (59) dari stempel waktu `'2009-07-30 12:58:59'` masukan.

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

Fungsi TIMESTAMP

Fungsi TIMESTAMP mengambil nilai (biasanya angka) dan mengubahnya menjadi tipe data stempel waktu.

Fungsi ini berguna ketika Anda perlu mengonversi nilai numerik yang mewakili waktu atau tanggal ke tipe data stempel waktu. Ini dapat membantu ketika Anda bekerja dengan data yang disimpan dalam format numerik, seperti stempel waktu Unix atau waktu epoch.

Sintaksis

```
timestamp(expr)
```

Pendapat

expr

Ekspresi apa pun yang dapat dilemparkan ke TIMESTAMP.

Pengembalian

Fungsi TIMESTAMP mengembalikan TIMESTAMP.

Contoh

Contoh berikut mengonversi stempel waktu Unix numerik (1632416400) ke tipe data stempel waktu yang sesuai: 22 September 2021 pukul 12:00:00 UTC.

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

Fungsi TO_TIMESTAMP

TO_TIMESTAMP mengonversi string TIMESTAMP ke TIMESTAMPTZ.

Sintaksis

```
to_timestamp (timestamp)
```

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

Argumen

stempel waktu

String timestamp atau tipe data yang dapat dilemparkan ke string timestamp.

format

String literal yang cocok dengan pola datetime Spark. Untuk pola datetime yang valid, lihat Pola Datetime untuk Pemformatan dan Penguraian.

Jenis pengembalian

TIMESTAMP

Contoh

Contoh berikut menunjukkan penggunaan fungsi TO_TIMESTAMP untuk mengonversi string TIMESTAMP ke 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
```

Dimungkinkan untuk melewatkan TO_TIMESTAMP bagian dari tanggal. Bagian tanggal yang tersisa diatur ke nilai default. Waktu termasuk dalam output:

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp
--------------------------
2017-01-01 00:00:00+00
```

Pernyataan SQL berikut mengonversi string '2011-12-18 24:38:15 'menjadi TIMESTAMP. Hasilnya adalah TIMESTAMP yang jatuh pada hari berikutnya karena jumlah jam lebih dari 24 jam:

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

to_timestamp
---------------------
2011-12-19 00:38:15+00
```

Fungsi YEAR

Fungsi YEAR adalah fungsi ekstraksi tanggal yang mengambil tanggal atau stempel waktu sebagai input dan mengembalikan komponen tahun (angka empat digit).

Sintaksis

```
year(date)
```

Pendapat

tanggal

Ekspresi DATE atau TIMESTAMP.

Pengembalian

Fungsi YEAR mengembalikan INTEGER.

Contoh

Contoh berikut mengekstrak komponen tahun (2016) dari tanggal input'2016-07-30'.

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

```
2016
```

Contoh berikut mengekstrak komponen tahun dari `birthday` kolom `squirrels` tabel dan mengembalikan hasil sebagai output dari pernyataan SELECT. Output dari kueri ini akan menjadi daftar nilai tahun, satu untuk setiap baris dalam `squirrels` tabel, mewakili tahun ulang tahun setiap tupai.

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

Bagian tanggal untuk fungsi tanggal atau stempel waktu

Tabel berikut mengidentifikasi nama bagian tanggal dan waktu bagian dan singkatan yang diterima sebagai argumen untuk fungsi berikut:

- DATE_ADD
- DATE_DIFF
- DATE_PART
- EKSTRAK

Tanggal paruh waktu atau paruh waktu	Singkatan
milenium, milenium	mil, mil
abad, berabad-abad	c, sen, sen
dekade, dekade	Desember, decs
jangka waktu	epoch (didukung oleh) EKSTRAK
tahun, tahun	y, thn, thn
seperempat, kuartal	qtr, qtrs
bulan, bulan	mon, mons
minggu, minggu	w

Tanggal paruh waktu atau paruh waktu	Singkatan
hari dalam seminggu	dayofweek, dow, dw, hari kerja (didukung oleh dan) DATE_PART Fungsi EKSTRAK Mengembalikan integer dari 0-6, dimulai dengan hari Minggu. **Note** Bagian tanggal DOW berperilaku berbeda dari bagian tanggal hari minggu (D) yang digunakan untuk string format datetime. D didasarkan pada bilangan bulat 1-7, di mana hari Minggu adalah 1. Untuk informasi selengkap nya, lihat String format datetime.
hari dalam setahun	dayofyear, doy, dy, yearday (didukung oleh) EKSTRAK
hari, hari	d
jam, jam	h, jam, jam
menit, menit	m, min, menit
kedua, detik	s, detik, detik
milidetik, milidetik	ms, msec, msec, mdetik, mdetik, milidetik, milidetik, milidetik, milidetik
mikrodetik, mikrodetik	mikrosec, mikrodetik, mikrodetik, usecond, useconds, us, usec, usec
zona waktu, zona waktu_jam , zona waktu_menit	Didukung oleh EKSTRAK untuk timestamp dengan zona waktu (TIMESTAMPTZ) saja.

Variasi hasil dengan detik, milidetik, dan mikrodetik

Perbedaan kecil dalam hasil kueri terjadi ketika fungsi tanggal yang berbeda menentukan detik, milidetik, atau mikrodetik sebagai bagian tanggal:

- Fungsi EXTRACT mengembalikan bilangan bulat untuk bagian tanggal yang ditentukan saja, mengabaikan bagian tanggal tingkat yang lebih tinggi dan lebih rendah. Jika bagian tanggal yang ditentukan adalah detik, milidetik dan mikrodetik tidak termasuk dalam hasil. Jika bagian tanggal yang ditentukan adalah milidetik, detik dan mikrodetik tidak termasuk. Jika bagian tanggal yang ditentukan adalah mikrodetik, detik dan milidetik tidak termasuk.
- Fungsi DATE_PART mengembalikan bagian detik lengkap dari stempel waktu, terlepas dari bagian tanggal yang ditentukan, mengembalikan nilai desimal atau bilangan bulat sesuai kebutuhan.

Catatan CENTURY, EPOCH, DECADE, dan MIL

CENTURY atau CENTURY

AWS Clean Rooms menafsirkan CENTURY untuk memulai dengan tahun ## #1 dan diakhiri dengan tahun: ###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

AWS Clean Rooms Implementasi EPOCH relatif terhadap 1970-01-01 00:00:00.000 000 terlepas dari zona waktu di mana cluster berada. Anda mungkin perlu mengimbangi hasil dengan perbedaan jam tergantung pada zona waktu di mana cluster berada.

DEKADE atau DEKADE

AWS Clean Rooms menafsirkan DECADE atau DECADECADES DATEPART berdasarkan kalender umum. Misalnya, karena kalender umum dimulai dari tahun 1, dekade pertama (dekade 1) adalah 0001-01-01 hingga 0009-12-31, dan dekade kedua (dekade 2) adalah 0010-01-01 hingga 0019-12-31. Misalnya, dekade 201 membentang dari 2000-01-01 hingga 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 atau MILS

AWS Clean Rooms menafsirkan MIL untuk memulai dengan hari pertama tahun #001 dan diakhiri dengan hari terakhir tahun#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)
```

Fungsi enkripsi dan dekripsi

Fungsi enkripsi dan dekripsi membantu pengembang SQL melindungi data sensitif dari akses atau penyalahgunaan yang tidak sah dengan mengubahnya antara formulir teks biasa yang dapat dibaca dan formulir ciphertext yang tidak dapat dibaca.

AWS Clean Rooms Spark SQL mendukung fungsi enkripsi dan dekripsi berikut:

Topik

Fungsi AES_ENCRYPT

Fungsi AES_ENCRYPT digunakan untuk mengenkripsi data menggunakan algoritma Advanced Encryption Standard (AES).

Sintaks

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

Argumen

expr

Nilai biner untuk mengenkripsi.

kunci

Passphrase yang digunakan untuk mengenkripsi data.

Panjang kunci 16, 24 dan 32 bit didukung.

modus

Menentukan modus blok cipher yang harus digunakan untuk mengenkripsi pesan.

Mode yang valid: ECB (Elektronik CodeBook), GCM (Mode Galois/Penghitung), CBC (Rantai Blok Sandi).

bantalan

Menentukan cara pad pesan yang panjangnya bukan kelipatan dari ukuran blok.

Nilai yang valid: PKCS, NONE, DEFAULT.

Padding DEFAULT berarti PKCS (Standar Kriptografi Kunci Publik) untuk ECB, NONE untuk GCM dan PKCS untuk CBC.

Kombinasi yang didukung dari (mode, padding) adalah ('ECB', 'PKCS'), ('GCM', 'NONE') dan ('CBC', 'PKCS').

iv

Vektor inisialisasi opsional (IV). Hanya didukung untuk mode CBC dan GCM.

Nilai yang valid: panjang 12-byte untuk GCM dan 16 byte untuk CBC.

aad

Data otentikasi tambahan opsional (AAD). Hanya didukung untuk mode GCM. Ini dapat berupa input bentuk bebas dan harus disediakan untuk enkripsi dan dekripsi.

Jenis pengembalian

Fungsi AES_ENCRYPT mengembalikan nilai expr terenkripsi menggunakan AES dalam mode yang diberikan dengan padding yang ditentukan.

Contoh

Contoh berikut menunjukkan cara menggunakan fungsi Spark SQL AES_ENCRYPT untuk mengenkripsi string data dengan aman (dalam hal ini, kata “Spark”) menggunakan kunci enkripsi tertentu. Ciphertext yang dihasilkan kemudian dikodekan Base64 untuk membuatnya lebih mudah untuk menyimpan atau mengirimkan.

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

Contoh berikut menunjukkan cara menggunakan fungsi Spark SQL AES_ENCRYPT untuk mengenkripsi string data dengan aman (dalam hal ini, kata “Spark”) menggunakan kunci enkripsi tertentu. Ciphertext yang dihasilkan kemudian direpresentasikan dalam format heksadesimal, yang dapat berguna untuk tugas-tugas seperti penyimpanan data, transmisi, atau debugging.

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

Contoh berikut menunjukkan cara menggunakan fungsi Spark SQL AES_ENCRYPT untuk mengenkripsi string data dengan aman (dalam hal ini, “Spark SQL”) menggunakan kunci enkripsi tertentu, mode enkripsi, dan mode padding. Ciphertext yang dihasilkan kemudian dikodekan Base64 untuk membuatnya lebih mudah untuk menyimpan atau mengirimkan.

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

Fungsi AES_DECRYPT

Fungsi AES_DECRYPT digunakan untuk mendekripsi data menggunakan algoritma Advanced Encryption Standard (AES).

Sintaks

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

Argumen

expr

Nilai biner untuk mendekripsi.

kunci

Passphrase yang digunakan untuk mendekripsi data.

Frasa sandi harus sesuai dengan kunci yang awalnya digunakan untuk menghasilkan nilai terenkripsi dan panjangnya 16, 24, atau 32 byte.

modus

Menentukan modus blok cipher yang harus digunakan untuk mendekripsi pesan.

Mode yang valid: ECB, GCM, CBC.

bantalan

Menentukan cara pad pesan yang panjangnya bukan kelipatan dari ukuran blok.

Nilai yang valid: PKCS, NONE, DEFAULT.

Padding DEFAULT berarti PKCS untuk ECB, NONE untuk GCM dan PKCS untuk CBC.

aad

Data otentikasi tambahan opsional (AAD). Hanya didukung untuk mode GCM. Ini dapat berupa input bentuk bebas dan harus disediakan untuk enkripsi dan dekripsi.

Jenis pengembalian

Mengembalikan nilai didekripsi expr menggunakan AES dalam mode dengan padding.

Contoh

Contoh berikut menunjukkan cara menggunakan fungsi Spark SQL AES_ENCRYPT untuk mengenkripsi string data dengan aman (dalam hal ini, kata “Spark”) menggunakan kunci enkripsi tertentu. Ciphertext yang dihasilkan kemudian dikodekan Base64 untuk membuatnya lebih mudah untuk menyimpan atau mengirimkan.

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

Contoh berikut menunjukkan bagaimana menggunakan fungsi Spark SQL AES_DECRYPT untuk mendekripsi data yang sebelumnya telah dienkripsi dan Base64-dikodekan. Proses dekripsi memerlukan kunci enkripsi dan parameter yang benar (mode enkripsi dan mode padding) untuk berhasil memulihkan data plaintext asli.

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

Fungsi hash

Fungsi hash adalah fungsi matematika yang mengubah nilai input numerik menjadi nilai lain.

AWS Clean Rooms Spark SQL mendukung fungsi hash berikut:

Topik

- MD5 fungsi
- Fungsi SHA
- SHA1 fungsi
- SHA2 fungsi
- HASH64 fungsi xx

MD5 fungsi

Menggunakan fungsi hash MD5 kriptografi untuk mengubah string panjang variabel menjadi string 32-karakter yang merupakan representasi teks dari nilai heksadesimal dari checksum 128-bit.

Sintaksis

```
MD5(string)
```

Argumen

tali

String panjang variabel.

Jenis pengembalian

MD5 Fungsi mengembalikan string 32-karakter yang merupakan representasi teks dari nilai heksadesimal dari checksum 128-bit.

Contoh

Contoh berikut menunjukkan nilai 128-bit untuk string "AWS Clean Rooms:

```
select md5('AWS Clean Rooms');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

Fungsi SHA

Sinonim dari fungsi. SHA1

Lihat SHA1 fungsi.

SHA1 fungsi

SHA1 Fungsi ini menggunakan fungsi hash SHA1 kriptografi untuk mengubah string panjang variabel menjadi string 40 karakter yang merupakan representasi teks dari nilai heksadesimal dari checksum 160-bit.

Sintaksis

SHA1 adalah sinonim dari. Fungsi SHA

```
SHA1(string)
```

Argumen

tali

String panjang variabel.

Jenis pengembalian

SHA1 Fungsi mengembalikan string 40 karakter yang merupakan representasi teks dari nilai heksadesimal dari checksum 160-bit.

Contoh

Contoh berikut mengembalikan nilai 160-bit untuk kata "AWS Clean Rooms:

```
select sha1('AWS Clean Rooms');
```

SHA2 fungsi

SHA2 Fungsi ini menggunakan fungsi hash SHA2 kriptografi untuk mengubah string panjang variabel menjadi string karakter. String karakter adalah representasi teks dari nilai heksadesimal checksum dengan jumlah bit yang ditentukan.

Sintaksis

```
SHA2(string, bits)
```

Argumen

tali

String panjang variabel.

bilangan bulat

Jumlah bit dalam fungsi hash. Nilai yang valid adalah 0 (sama dengan 256), 224, 256, 384, dan 512.

Jenis pengembalian

SHA2 Fungsi mengembalikan string karakter yang merupakan representasi teks dari nilai heksadesimal checksum atau string kosong jika jumlah bit tidak valid.

Contoh

Contoh berikut mengembalikan nilai 256-bit untuk kata "AWS Clean Rooms:

```
select sha2('AWS Clean Rooms', 256);
```

HASH64 fungsi xx

Fungsi xxhash64 mengembalikan nilai hash 64-bit dari argumen.

Fungsi xxhash64 () adalah fungsi hash non-kriptografi yang dirancang agar cepat dan efisien. Ini sering digunakan dalam pemrosesan data dan aplikasi penyimpanan, di mana pengidentifikasi unik untuk sepotong data diperlukan, tetapi konten yang tepat dari data tidak perlu dirahasiakan.

Dalam konteks kueri SQL, fungsi xxhash64 () dapat digunakan untuk berbagai tujuan, seperti:

- Menghasilkan pengenal unik untuk baris dalam tabel
- Mempartisi data berdasarkan nilai hash
- Menerapkan strategi pengindeksan atau distribusi data khusus

Kasus penggunaan spesifik akan tergantung pada persyaratan aplikasi dan data yang sedang diproses.

Sintaksis

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

Pendapat

expr1

Ekspresi jenis apa pun.

expr2

Ekspresi jenis apa pun.

Pengembalian

Mengembalikan nilai hash 64-bit dari argumen (BIGINT). Benih hash adalah 42.

Contoh

Contoh berikut menghasilkan nilai hash 64-bit (5602566077635097486) berdasarkan input yang disediakan. Argumen pertama adalah nilai string, dalam hal ini, kata “Spark”. Argumen kedua adalah array yang berisi nilai integer tunggal 123. Argumen ketiga adalah nilai integer yang mewakili seed untuk fungsi hash.

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

Fungsi hyperloglog

Fungsi HyperLogLog (HLL) di SQL menyediakan cara untuk memperkirakan secara efisien jumlah elemen unik (kardinalitas) dalam kumpulan data besar, bahkan ketika kumpulan elemen unik yang sebenarnya tidak disimpan.

Manfaat utama menggunakan fungsi HLL adalah:

- Efisiensi memori: Sketsa HLL membutuhkan memori jauh lebih sedikit daripada menyimpan set lengkap elemen unik, membuatnya cocok untuk kumpulan data besar.
- Komputasi terdistribusi: Sketsa HLL dapat digabungkan di beberapa sumber data atau node pemrosesan, memungkinkan estimasi hitungan unik terdistribusi yang efisien.
- Hasil perkiraan: HLL memberikan perkiraan estimasi hitungan unik, dengan trade-off yang dapat disetel antara akurasi dan penggunaan memori (melalui parameter presisi).

Fungsi-fungsi ini sangat berguna dalam skenario di mana Anda perlu memperkirakan jumlah item unik, seperti dalam analitik, pergudangan data, dan aplikasi pemrosesan aliran waktu nyata.

AWS Clean Rooms mendukung fungsi HLL berikut.

Topik

- Fungsi HLL_SKETCH_AGG
- Fungsi HLL_SKETCH_ESTIMATE
- Fungsi HLL_UNION

- Fungsi HLL_UNION_AGG

Fungsi HLL_SKETCH_AGG

Fungsi agregat HLL_SKETCH_AGG membuat sketsa HLL dari nilai-nilai di kolom yang ditentukan. Ia mengembalikan tipe data HLLSKETCH yang merangkum nilai ekspresi input.

Fungsi agregat HLL_SKETCH_AGG bekerja dengan tipe data apa pun dan mengabaikan nilai NULL.

Ketika tidak ada baris dalam tabel atau semua baris adalah NULL, sketsa yang dihasilkan tidak memiliki pasangan nilai indeks seperti. `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}`

Sintaks

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

Pendapat

aggregate_expression

Setiap ekspresi tipe INT, BIGINT, STRING, atau BINARY yang dengannya penghitungan unik akan terjadi. `NULL`Nilai apa pun diabaikan.

LGConfigk

Konstanta INT opsional antara 4 dan 21 inklusif dengan default 12. Log-base-2 dari K, di mana K adalah jumlah ember atau slot untuk sketsa.

Jenis pengembalian

Fungsi HLL_SKETCH_AGG mengembalikan buffer BINARY non-NULL yang berisi HyperLogLog sketsa yang dihitung karena mengkonsumsi dan menggabungkan semua nilai input dalam grup agregasi.

Contoh

Contoh berikut menggunakan algoritma HyperLogLog (HLL) untuk memperkirakan jumlah nilai yang berbeda dalam `col` kolom. `hll_sketch_agg(col, 12)`Fungsi agregat nilai-nilai dalam kolom col, membuat sketsa HLL menggunakan presisi 12. `hll_sketch_estimate()`Fungsi ini kemudian

digunakan untuk memperkirakan jumlah nilai yang berbeda berdasarkan sketsa HLL yang dihasilkan. Hasil akhir dari kueri adalah 3, yang mewakili perkiraan jumlah nilai yang berbeda di `col` kolom. Dalam hal ini, nilai yang berbeda adalah 1, 2, dan 3.

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

Contoh berikut juga menggunakan algoritma HLL untuk memperkirakan jumlah nilai yang berbeda di `col` kolom, tetapi tidak menentukan nilai presisi untuk sketsa HLL. Dalam hal ini, ia menggunakan presisi default 14. `hll_sketch_agg(col)`Fungsi mengambil nilai-nilai dalam `col` kolom dan membuat sketsa HyperLogLog (HLL), yang merupakan struktur data kompak yang dapat digunakan untuk memperkirakan jumlah elemen yang berbeda. `hll_sketch_estimate(hll_sketch_agg(col))`Fungsi ini mengambil sketsa HLL yang dibuat pada langkah sebelumnya dan menghitung perkiraan jumlah nilai yang berbeda di kolom. `col` Hasil akhir dari kueri adalah 3, yang mewakili perkiraan jumlah nilai yang berbeda di `col` kolom. Dalam hal ini, nilai yang berbeda adalah 1, 2, dan 3.

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

Fungsi HLL_SKETCH_ESTIMATE

Fungsi HLL_SKETCH_ESTIMATE mengambil sketsa HLL dan memperkirakan jumlah elemen unik yang diwakili oleh sketsa. Ini menggunakan algoritma HyperLogLog (HLL) untuk menghitung perkiraan probabilistik dari jumlah nilai unik dalam kolom tertentu, menggunakan representasi biner yang dikenal sebagai buffer sketsa yang sebelumnya dihasilkan oleh fungsi HLL_SKETCH_AGG dan mengembalikan hasilnya sebagai bilangan bulat besar.

Algoritma sketsa HLL menyediakan cara yang efisien untuk memperkirakan jumlah elemen unik, bahkan untuk kumpulan data besar, tanpa harus menyimpan set lengkap nilai unik.

`hll_union_agg`Fungsi `hll_union` dan juga dapat menggabungkan sketsa bersama-sama dengan mengkonsumsi dan menggabungkan buffer ini sebagai input.

Sintaks

```
HLL_SKETCH_ESTIMATE (hllsketch_expression)
```

Pendapat

hllsketch_expression

`BINARY`Ekspresi yang memegang sketsa yang dihasilkan oleh HLL_SKETCH_AGG

Jenis pengembalian

Fungsi HLL_SKETCH_ESTIMATE mengembalikan nilai BIGINT yang merupakan perkiraan jumlah berbeda yang diwakili oleh sketsa masukan.

Contoh

Contoh berikut menggunakan algoritma sketsa HyperLogLog (HLL) untuk memperkirakan kardinalitas (jumlah unik) nilai dalam kolom. `col` `hll_sketch_agg(col, 12)`Fungsi mengambil `col` kolom dan membuat sketsa HLL menggunakan presisi 12 bit. Sketsa HLL adalah struktur data perkiraan yang dapat secara efisien memperkirakan jumlah elemen unik dalam satu set. `hll_sketch_estimate()`Fungsi mengambil sketsa HLL yang dibuat oleh `hll_sketch_agg` dan memperkirakan kardinalitas (jumlah unik) dari nilai yang diwakili oleh sketsa. `FROM VALUES (1), (1), (2), (2), (3) tab(col);`Menghasilkan dataset uji dengan 5 baris, di mana `col` kolom berisi nilai 1, 1, 2, 2, dan 3. Hasil dari kueri ini adalah perkiraan jumlah unik dari nilai-nilai di `col` kolom, yaitu 3.

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

Perbedaan antara contoh berikut dan yang sebelumnya adalah bahwa parameter presisi (12 bit) tidak ditentukan dalam panggilan `hll_sketch_agg` fungsi. Dalam hal ini, presisi default 14 bit digunakan, yang dapat memberikan perkiraan yang lebih akurat untuk hitungan unik dibandingkan dengan contoh sebelumnya yang menggunakan 12 bit presisi.

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

Fungsi HLL_UNION

Fungsi HLL_UNION menggabungkan dua sketsa HLL menjadi satu sketsa terpadu. Ini menggunakan algoritma HyperLogLog (HLL) untuk menggabungkan dua sketsa menjadi satu sketsa. Kueri dapat

menggunakan buffer yang dihasilkan untuk menghitung perkiraan jumlah unik sebagai bilangan bulat panjang dengan fungsi tersebut. `hll_sketch_estimate`

Sintaks

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

Pendapat

ExPRN

`BINARY`Ekspresi yang memegang sketsa yang dihasilkan oleh HLL_SKETCH_AGG.

allowDifferentLgConfigK

Ekspresi BOOLEAN opsional yang mengontrol apakah akan mengizinkan penggabungan dua sketsa dengan nilai LGConfigk yang berbeda. Nilai default-nya adalah `false`.

Jenis pengembalian

Fungsi HLL_UNION mengembalikan buffer BINARY yang berisi HyperLogLog sketsa dihitung sebagai hasil dari menggabungkan ekspresi input. Ketika `allowDifferentLgConfigK` parameternya`true`, sketsa hasil menggunakan yang lebih kecil dari dua `lgConfigK` nilai yang disediakan.

Contoh

Contoh berikut menggunakan algoritma sketsa HyperLogLog (HLL) untuk memperkirakan jumlah nilai unik di dua kolom, `col1` dan`col2`, dalam kumpulan data.

`hll_sketch_agg(col1)`Fungsi ini membuat sketsa HLL untuk nilai unik di kolom. `col1`

`hll_sketch_agg(col2)`Fungsi ini membuat sketsa HLL untuk nilai unik di kolom col2.

`hll_union(...)`Fungsi ini menggabungkan dua sketsa HLL yang dibuat dalam langkah 1 dan 2 menjadi satu sketsa HLL terpadu.

`hll_sketch_estimate(...)`Fungsi ini mengambil sketsa HLL gabungan dan memperkirakan jumlah nilai unik di keduanya `col1` dan. `col2`

`FROM VALUES`Klausa menghasilkan kumpulan data uji dengan 5 baris, yang `col1` berisi nilai 1, 1, 2, 2, dan 3, dan `col2` berisi nilai 4, 4, 5, 5, dan 6.

Hasil dari kueri ini adalah perkiraan jumlah nilai unik di keduanya `col1` dan`col2`, yaitu 6. Algoritma sketsa HLL menyediakan cara yang efisien untuk memperkirakan jumlah elemen unik, bahkan untuk kumpulan data besar, tanpa harus menyimpan set lengkap nilai unik. Dalam contoh ini, `hll_union` fungsi ini digunakan untuk menggabungkan sketsa HLL dari dua kolom, yang memungkinkan hitungan unik diperkirakan di seluruh kumpulan data, bukan hanya untuk setiap kolom satu per satu.

```
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
```

Perbedaan antara contoh berikut dan yang sebelumnya adalah bahwa parameter presisi (12 bit) tidak ditentukan dalam panggilan `hll_sketch_agg` fungsi. Dalam hal ini, presisi default 14 bit digunakan, yang dapat memberikan perkiraan yang lebih akurat untuk hitungan unik dibandingkan dengan contoh sebelumnya yang menggunakan 12 bit presisi.

```
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);
```

Fungsi HLL_UNION_AGG

Fungsi HLL_UNION_AGG menggabungkan beberapa sketsa HLL menjadi satu sketsa terpadu. Ini menggunakan algoritma HyperLogLog (HLL) untuk menggabungkan sekelompok sketsa menjadi satu. Kueri dapat menggunakan buffer yang dihasilkan untuk menghitung perkiraan jumlah unik dengan fungsi tersebut. `hll_sketch_estimate`

Sintaks

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

Pendapat

expr

`BINARY`Ekspresi yang memegang sketsa yang dihasilkan oleh HLL_SKETCH_AGG.

allowDifferentLgConfigK

Ekspresi BOOLEAN opsional yang mengontrol apakah akan mengizinkan penggabungan dua sketsa dengan nilai LGConfigk yang berbeda. Nilai default-nya adalah `false`.

Jenis pengembalian

Fungsi HLL_UNION_AGG mengembalikan buffer BINARY yang berisi HyperLogLog sketsa dihitung sebagai hasil dari menggabungkan ekspresi masukan dari grup yang sama. Ketika `allowDifferentLgConfigK` parameternya`true`, sketsa hasil menggunakan yang lebih kecil dari dua `lgConfigK` nilai yang disediakan.

Contoh

Contoh berikut menggunakan algoritma sketsa HyperLogLog (HLL) untuk memperkirakan jumlah nilai unik di beberapa sketsa HLL.

Contoh pertama memperkirakan jumlah unik nilai dalam kumpulan data.

```
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
```

Kueri bagian dalam membuat dua sketsa HLL:

- Pernyataan SELECT pertama membuat sketsa dari satu nilai 1.

- Pernyataan SELECT kedua membuat sketsa dari nilai tunggal lain dari 1, tetapi dengan presisi 20.

Kueri luar menggunakan fungsi HLL_UNION_AGG untuk menggabungkan dua sketsa menjadi sketsa tunggal. Kemudian menerapkan fungsi HLL_SKETCH_ESTIMATE ke sketsa gabungan ini untuk memperkirakan jumlah nilai yang unik.

Hasil dari kueri ini adalah perkiraan jumlah unik dari nilai-nilai di `col` kolom, yaitu`1`. Ini berarti bahwa dua nilai input dari 1 dianggap unik, meskipun mereka memiliki nilai yang sama.

Contoh kedua mencakup parameter presisi yang berbeda untuk fungsi HLL_UNION_AGG. Dalam hal ini, kedua sketsa HLL dibuat dengan presisi 14 bit, yang memungkinkan mereka untuk berhasil `hll_union_agg` digabungkan menggunakan parameter. `true`

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

Hasil akhir dari kueri adalah perkiraan jumlah unik, yang dalam hal ini juga`1`. Ini berarti bahwa dua nilai input dari 1 dianggap unik, meskipun mereka memiliki nilai yang sama.

Fungsi JSON

Ketika Anda perlu menyimpan kumpulan pasangan kunci-nilai yang relatif kecil, Anda dapat menghemat ruang dengan menyimpan data dalam format JSON. Karena string JSON dapat disimpan dalam satu kolom, menggunakan JSON mungkin lebih efisien daripada menyimpan data Anda dalam format tabel.

Example

Misalnya, Anda memiliki tabel jarang, di mana Anda harus memiliki banyak kolom untuk sepenuhnya mewakili semua atribut yang mungkin. Namun, sebagian besar nilai kolom adalah NULL untuk setiap baris tertentu atau kolom tertentu. Dengan menggunakan JSON untuk penyimpanan, Anda mungkin dapat menyimpan data untuk baris dalam pasangan kunci-nilai dalam string JSON tunggal dan menghilangkan kolom tabel yang jarang diisi.

Selain itu, Anda dapat dengan mudah memodifikasi string JSON untuk menyimpan pasangan kunci: nilai tambahan tanpa perlu menambahkan kolom ke tabel.

Kami merekomendasikan menggunakan JSON dengan hemat. JSON bukanlah pilihan yang baik untuk menyimpan kumpulan data yang lebih besar karena, dengan menyimpan data yang berbeda dalam satu kolom, JSON tidak menggunakan arsitektur penyimpanan kolom. AWS Clean Rooms

JSON menggunakan string teks yang dikodekan UTF-8, sehingga string JSON dapat disimpan sebagai tipe data CHAR atau VARCHAR. Gunakan VARCHAR jika string menyertakan karakter multi-byte.

String JSON harus benar diformat JSON, sesuai dengan aturan berikut:

- Tingkat akar JSON dapat berupa objek JSON atau array JSON. Objek JSON adalah kumpulan pasangan kunci:nilai yang dipisahkan koma yang tidak berurutan yang diapit oleh kurawal kurawal.

  Sebagai contoh, `{"one":1, "two":2}` .
- Array JSON adalah sekumpulan nilai yang dipisahkan koma yang diurutkan yang diapit oleh tanda kurung.

  Contohnya adalah sebagai berikut: `["first", {"one":1}, "second", 3, null]`
- Array JSON menggunakan indeks berbasis nol; elemen pertama dalam array berada pada posisi 0. Dalam pasangan kunci JSON: nilai, kuncinya adalah string dalam tanda kutip ganda.
- Nilai JSON dapat berupa salah satu dari berikut ini:
  - Objek JSON
  - Array JSON
  - String dalam tanda kutip ganda
  - Angka (integer dan float)
  - Boolean
  - Nol
- Objek kosong dan array kosong adalah nilai JSON yang valid.
- Bidang JSON peka huruf besar/kecil.
- Ruang putih antara elemen struktural JSON (seperti`{ }, [ ]`) diabaikan.

Topik

- Fungsi GET_JSON_OBJECT

- Fungsi TO_JSON

Fungsi GET_JSON_OBJECT

Fungsi GET_JSON_OBJECT mengekstrak objek json dari. `path`

Sintaksis

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

Pendapat

json_txt

Ekspresi STRING yang berisi JSON yang terbentuk dengan baik.

path

Sebuah literal STRING dengan ekspresi jalur JSON yang terbentuk dengan baik.

Pengembalian

Mengembalikan STRING.

NULL dikembalikan jika objek tidak dapat ditemukan.

Contoh

Contoh berikut mengekstrak nilai dari objek JSON.. Argumen pertama adalah string JSON yang mewakili objek sederhana dengan pasangan kunci-nilai tunggal. Argumen kedua adalah ekspresi jalur JSON. $Simbol mewakili akar objek JSON, dan `.a` bagian menentukan bahwa kita ingin mengekstrak nilai yang terkait dengan kunci a "". Output dari fungsi ini adalah 'b', yang merupakan nilai yang terkait dengan tombol a "" di objek input JSON.

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

Fungsi TO_JSON

Fungsi TO_JSON mengubah ekspresi input menjadi representasi string JSON. Fungsi ini menangani konversi tipe data yang berbeda (seperti angka, string, dan boolean) ke dalam representasi JSON yang sesuai.

Fungsi TO_JSON berguna ketika Anda perlu mengonversi data terstruktur (seperti baris database atau objek JSON) menjadi format yang lebih portabel dan menggambarkan diri seperti JSON.
Ini dapat sangat membantu ketika Anda perlu berinteraksi dengan sistem atau layanan lain yang mengharapkan data berformat JSON.

Sintaksis

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

Pendapat

expr

Ekspresi masukan yang ingin Anda konversi ke string JSON. Ini bisa berupa nilai, kolom, atau ekspresi SQL valid lainnya.

options

Kumpulan opsi konfigurasi opsional yang dapat digunakan untuk menyesuaikan proses konversi JSON. Opsi ini dapat mencakup hal-hal seperti penanganan nilai nol, representasi nilai numerik, dan perlakuan karakter khusus..

Pengembalian

Mengembalikan string JSON dengan nilai struct yang diberikan

Contoh

Contoh berikut mengkonversi struct bernama (jenis data terstruktur) menjadi string JSON. Argumen pertama(`named_struct('a', 1, 'b', 2)`) adalah ekspresi input yang diteruskan ke `to_json()` fungsi. Ini menciptakan struct bernama dengan dua bidang: “a” dengan nilai 1, dan “b” dengan nilai 2. Fungsi to_json () mengambil struct bernama sebagai argumennya dan mengubahnya menjadi representasi string JSON. Outputnya adalah`{"a":1,"b":2}`, yang merupakan string JSON valid yang mewakili struct bernama.

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

Contoh berikut mengonversi struct bernama yang berisi nilai timestamp menjadi string JSON, dengan format stempel waktu yang disesuaikan. Argumen pertama (`named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd'))`) membuat struct bernama dengan satu bidang 'waktu' yang berisi nilai stempel waktu. Argumen kedua (`map('timestampFormat', 'dd/MM/yyyy')`) membuat peta (key-value dictionary) dengan pasangan kunci-nilai tunggal, di mana kuncinya adalah 'TimestampFormat' dan nilainya adalah ". 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 Outputnya adalah`{"time":"26/08/2015"}`, yaitu string JSON dengan satu bidang 'waktu' yang berisi nilai stempel waktu dalam format " dd/MM/yyyy yang diinginkan.

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

Fungsi matematika

Bagian ini menjelaskan operator matematika dan fungsi yang didukung dalam AWS Clean Rooms Spark SQL.

Topik

- Fungsi DERAJAT
- Fungsi DIV
- Fungsi EXP
- Fungsi FLOOR
- Fungsi LN
- Fungsi LOG
- Fungsi MOD
- Fungsi PI
- Fungsi POWER
- Fungsi RADIANS
- Fungsi RAND
- fungsi RANDOM
- Fungsi ROUND
- Fungsi SIGN
- Fungsi SIN
- Fungsi SQRT
- Fungsi TRUNC

Simbol operator matematika

Tabel berikut mencantumkan operator matematika yang didukung.

Operator yang didukung

Operator	Deskripsi	Contoh	Hasil
+	tambahan	2 + 3	5
-	pengurangan	2 - 3	-1
*	perkalian	2 * 3	6

Operator	Deskripsi	Contoh	Hasil
/	pembagian	4/2	2
%	modulo	5% 4	1
^	eksponensial	2.0 ^ 3.0	8

Contoh

Hitung komisi yang dibayarkan ditambah biaya penanganan $2,00 untuk transaksi tertentu:

```
select commission, (commission + 2.00) as comm
from sales where salesid=10000;

commission | comm
-----------+-------
28.05      | 30.05
(1 row)
```

Hitung 20 persen dari harga jual untuk transaksi tertentu:

```
select pricepaid, (pricepaid * .20) as twentypct
from sales where salesid=10000;

pricepaid | twentypct
----------+-----------
187.00    |    37.400
(1 row)
```

Forecast penjualan tiket berdasarkan pola pertumbuhan berkelanjutan. Dalam contoh ini, subquery mengembalikan jumlah tiket yang terjual pada tahun 2008. Hasil itu dikalikan secara eksponensial dengan tingkat pertumbuhan berkelanjutan sebesar 5 persen selama 10 tahun.

```
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)
```

Temukan total harga yang dibayarkan dan komisi untuk penjualan dengan ID tanggal yang lebih besar dari atau sama dengan 2.000. Kemudian kurangi total komisi dari total harga yang dibayarkan.

```
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)
```

Fungsi ABS

ABS menghitung nilai absolut dari suatu angka, di mana angka itu dapat berupa literal atau ekspresi yang mengevaluasi angka.

Sintaksis

```
ABS (number)
```

Argumen

jumlah

Angka atau ekspresi yang mengevaluasi angka. Ini bisa berupa SMALLINT, INTEGER, BIGINT, DECIMAL,, atau type. FLOAT4 FLOAT8

Jenis pengembalian

ABS mengembalikan tipe data yang sama dengan argumennya.

Contoh

Hitung nilai absolut -38:

```
select abs (-38);
abs
-------
38
(1 row)
```

Hitung nilai absolut (14-76):

```
select abs (14-76);
abs
-------
62
(1 row)
```

Fungsi ACOS

ACOS adalah fungsi trigonometri yang mengembalikan kosinus busur suatu angka. Nilai kembali dalam radian dan berada di antara `0` dan`PI`.

Sintaksis

```
ACOS(number)
```

Argumen

jumlah

Parameter input adalah `DOUBLE PRECISION` angka.

Jenis pengembalian

`DOUBLE PRECISION`

Contoh

Untuk mengembalikan arc cosinus dari-1, gunakan contoh berikut.

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Fungsi ASIN

ASIN adalah fungsi trigonometri yang mengembalikan sinus busur dari suatu angka. Nilai kembali dalam radian dan berada di antara PI/2 dan-PI/2.

Sintaksis

```
ASIN(number)
```

Argumen

jumlah

Parameter input adalah DOUBLE PRECISION angka.

Jenis pengembalian

DOUBLE PRECISION

Contoh

Untuk mengembalikan sinus busur1, gunakan contoh berikut.

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
```

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

Fungsi ATAN

ATAN adalah fungsi trigonometri yang mengembalikan garis singgung busur dari suatu bilangan. Nilai kembali dalam radian dan berada di antara `-PI` dan`PI`.

Sintaksis

```
ATAN(number)
```

Argumen

jumlah

Parameter input adalah `DOUBLE PRECISION` angka.

Jenis pengembalian

`DOUBLE PRECISION`

Contoh

Untuk mengembalikan garis singgung busur 1 dan kalikan dengan 4, gunakan contoh berikut.

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

ATAN2 fungsi

ATAN2 adalah fungsi trigonometri yang mengembalikan tangen busur dari satu angka dibagi dengan angka lain. Nilai kembali dalam radian dan berada di antara `PI/2` dan`-PI/2`.

Sintaksis

```
ATAN2(number1, number2)
```

Argumen

nomor1

Sebuah DOUBLE PRECISION angka.

nomor2

Sebuah DOUBLE PRECISION angka.

Jenis pengembalian

DOUBLE PRECISION

Contoh

Untuk mengembalikan garis singgung busur 2/2 dan kalikan dengan 4, gunakan contoh berikut.

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Fungsi CBRT

Fungsi CBRT adalah fungsi matematika yang menghitung akar kubus dari suatu angka.

Sintaksis

```
CBRT (number)
```

Pendapat

CBRT mengambil nomor PRESISI GANDA sebagai argumen.

Jenis pengembalian

CBRT mengembalikan nomor PRESISI GANDA.

Contoh

Hitung akar kubus dari komisi yang dibayarkan untuk transaksi tertentu:

```
select cbrt(commission) from sales where salesid=10000;

cbrt
------------------
3.03839539048843
(1 row)
```

Fungsi CEILING (atau CEIL)

Fungsi CEILING atau CEIL digunakan untuk membulatkan angka ke bilangan bulat berikutnya. (Fungsi FLOORMembulatkan angka ke bawah ke bilangan bulat berikutnya.)

Sintaksis

```
CEIL | CEILING(number)
```

Argumen

jumlah

Angka atau ekspresi yang mengevaluasi ke angka. Ini bisa berupa SMALLINT, INTEGER, BIGINT, DECIMAL,, atau type. FLOAT4 FLOAT8

Jenis pengembalian

CEILING dan CEIL mengembalikan tipe data yang sama dengan argumennya.

Contoh

Hitung plafon komisi yang dibayarkan untuk transaksi penjualan tertentu:

```
select ceiling(commission) from sales
where salesid=10000;

ceiling
---------
29
(1 row)
```

Fungsi COS

COS adalah fungsi trigonometri yang mengembalikan kosinus suatu bilangan. Nilai kembalinya dalam radian dan berada di antara -1 dan1, inklusif.

Sintaksis

```
COS(double_precision)
```

Pendapat

jumlah

Parameter input adalah angka presisi ganda.

Jenis pengembalian

Fungsi COS mengembalikan angka presisi ganda.

Contoh

Contoh berikut mengembalikan cosinus dari 0:

```
select cos(0);
cos
-----
1
(1 row)
```

Contoh berikut mengembalikan kosinus PI:

```
select cos(pi());
cos
-----
-1
(1 row)
```

Fungsi COT

COT adalah fungsi trigonometri yang mengembalikan kotangen angka. Parameter input harus bukan nol.

Sintaksis

```
COT(number)
```

Pendapat

jumlah

Parameter input adalah DOUBLE PRECISION angka.

Jenis pengembalian

DOUBLE PRECISION

Contoh

Untuk mengembalikan kotangen 1, gunakan contoh berikut.

```
SELECT COT(1);

+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

Fungsi DERAJAT

Mengubah sudut dalam radian menjadi setara dalam derajat.

Sintaksis

```
DEGREES(number)
```

Pendapat

jumlah

Parameter input adalah DOUBLE PRECISION angka.

Jenis pengembalian

`DOUBLE PRECISION`

Contoh

Untuk mengembalikan derajat setara dengan 0,5 radian, gunakan contoh berikut.

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
+-------------------+
```

Untuk mengkonversi PI radian ke derajat, gunakan contoh berikut.

```
SELECT DEGREES(pi());

+---------+
| degrees |
+---------+
|     180 |
+---------+
```

Fungsi DIV

Operator DIV mengembalikan bagian integral dari pembagian dividen oleh pembagi.

Sintaksis

```
dividend div divisor
```

Argumen

dividen

Ekspresi yang mengevaluasi numerik atau interval.

pembagi

Tipe interval yang cocok jika `dividend` adalah interval, numerik sebaliknya.

Jenis pengembalian

`BIGINT`

Contoh

Contoh berikut memilih dua kolom dari tabel tupai: `id` kolom, yang berisi pengidentifikasi unik untuk setiap tupai, dan kolom,`age div 2`, yang mewakili pembagian bilangan bulat dari `calculated` kolom usia dengan 2. `age div 2`Perhitungan melakukan pembagian bilangan bulat pada `age` kolom, secara efektif membulatkan usia ke bilangan bulat genap terdekat. Misalnya, jika `age` kolom berisi nilai-nilai seperti 3, 5, 7, dan 10, `age div 2` kolom akan berisi nilai 1, 2, 3, dan 5, masing-masing.

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

Kueri ini dapat berguna dalam skenario di mana Anda perlu mengelompokkan atau menganalisis data berdasarkan rentang usia, dan Anda ingin menyederhanakan nilai usia dengan membulatkannya ke bilangan bulat genap terdekat. Output yang dihasilkan akan memberikan `id` dan usia dibagi 2 untuk setiap tupai dalam tabel`squirrels`.

Fungsi EXP

Fungsi EXP mengimplementasikan fungsi eksponensial untuk ekspresi numerik, atau dasar logaritma natural,, e dinaikkan ke kekuatan ekspresi. Fungsi EXP adalah kebalikan dari. Fungsi LN

Sintaksis

```
EXP (expression)
```

Pendapat

ekspresi

Ekspresi harus berupa tipe data INTEGER, DECIMAL, atau DOUBLE PRECISION.

Jenis pengembalian

EXP mengembalikan nomor PRESISI GANDA.

Contoh

Gunakan fungsi EXP untuk memperkirakan penjualan tiket berdasarkan pola pertumbuhan berkelanjutan. Dalam contoh ini, subquery mengembalikan jumlah tiket yang terjual pada tahun 2008. Hasil itu dikalikan dengan hasil fungsi EXP, yang menentukan tingkat pertumbuhan berkelanjutan 7% selama 10 tahun.

```
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)
```

Fungsi FLOOR

Fungsi FLOOR membulatkan angka ke bawah ke bilangan bulat berikutnya.

Sintaksis

```
FLOOR (number)
```

Pendapat

jumlah

Angka atau ekspresi yang mengevaluasi ke angka. Ini bisa berupa SMALLINT, INTEGER, BIGINT, DECIMAL,, atau type. FLOAT4 FLOAT8

Jenis pengembalian

FLOOR mengembalikan tipe data yang sama sebagai argumennya.

Contoh

Contoh menunjukkan nilai komisi yang dibayarkan untuk transaksi penjualan tertentu sebelum dan sesudah menggunakan fungsi 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)
```

Fungsi LN

Fungsi LN mengembalikan logaritma natural dari parameter input.

Sintaksis

```
LN(expression)
```

Pendapat

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi.

> **Note**
>
> Fungsi ini mengembalikan kesalahan untuk beberapa tipe data jika ekspresi referensi tabel yang AWS Clean Rooms dibuat pengguna atau tabel sistem AWS Clean Rooms STL atau STV.

Ekspresi dengan tipe data berikut menghasilkan kesalahan jika mereka mereferensikan tabel yang dibuat pengguna atau sistem.

- BOOLEAN
- CHAR

- DATE
- DESIMAL atau NUMERIK
- TIMESTAMP
- VARCHAR

Ekspresi dengan tipe data berikut berjalan dengan sukses pada tabel yang dibuat pengguna dan tabel sistem STL atau STV:

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

Jenis pengembalian

Fungsi LN mengembalikan tipe yang sama dengan ekspresi.

Contoh

Contoh berikut mengembalikan logaritma natural, atau logaritma basis e, dari angka 2.718281828:

```
select ln(2.718281828);
ln
--------------------
0.9999999998311267
(1 row)
```

Perhatikan bahwa jawabannya hampir sama dengan 1.

Contoh ini mengembalikan logaritma natural dari nilai-nilai dalam kolom USERID dalam tabel USERID:

```
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)
```

Fungsi LOG

Mengembalikan logaritma dengan. `expr` `base`

Sintaksis

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

Pendapat

expr

Ekspresi harus memiliki tipe data integer, desimal, atau floating-point.

dasar

Dasar untuk perhitungan logaritma. Harus berupa angka positif (tidak sama dengan 1) tipe data presisi ganda.

Jenis pengembalian

Fungsi LOG mengembalikan nomor presisi ganda.

Contoh

Contoh berikut mengembalikan basis 10 logaritma dari angka 100:

```
select log(10, 100);
--------
2
(1 row)
```

Fungsi MOD

Mengembalikan sisa dari dua angka, atau dikenal sebagai operasi modulo. Untuk menghitung hasilnya, parameter pertama dibagi dengan yang kedua.

Sintaksis

```
MOD(number1, number2)
```

Argumen

nomor1

Parameter input pertama adalah bilangan INTEGER, SMALLINT, BIGINT, atau DECIMAL. Jika salah satu parameter adalah tipe DECIMAL, parameter lainnya juga harus tipe DECIMAL. Jika salah satu parameter adalah INTEGER, parameter lainnya dapat berupa INTEGER, SMALLINT, atau BIGINT. Kedua parameter juga dapat berupa SMALLINT atau BIGINT, tetapi satu parameter tidak dapat menjadi SMALLINT jika yang lain adalah BIGINT.

nomor2

Parameter kedua adalah bilangan INTEGER, SMALLINT, BIGINT, atau DECIMAL. Aturan tipe data yang sama berlaku untuk number2 untuk number1.

Jenis pengembalian

Jenis pengembalian yang valid adalah DECIMAL, INT, SMALLINT, dan BIGINT. Jenis pengembalian fungsi MOD adalah tipe numerik yang sama dengan parameter input, jika kedua parameter input adalah tipe yang sama. Jika salah satu parameter input adalah INTEGER, bagaimanapun, tipe kembali juga akan menjadi INTEGER.

Catatan penggunaan

Anda dapat menggunakan% sebagai operator modulo.

Contoh

Contoh berikut mengembalikan sisanya ketika angka dibagi dengan yang lain:

```
SELECT MOD(10, 4);
```

```
 mod
------
 2
```

Contoh berikut mengembalikan hasil desimal:

```
SELECT MOD(10.5, 4);

 mod
------
 2.5
```

Anda dapat mentransmisikan nilai parameter:

```
SELECT MOD(CAST(16.4 as integer), 5);

 mod
------
 1
```

Periksa apakah parameter pertama genap dengan membaginya dengan 2:

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

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

Anda dapat menggunakan% sebagai operator modulo:

```
SELECT 11 % 4 as remainder;

 remainder
-----------
 3
```

Contoh berikut mengembalikan informasi untuk kategori bernomor ganjil dalam tabel 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)
```

Fungsi PI

Fungsi PI mengembalikan nilai pi ke 14 tempat desimal.

Sintaksis

```
PI()
```

Jenis pengembalian

`DOUBLE PRECISION`

Contoh

Untuk mengembalikan nilai pi, gunakan contoh berikut.

```
SELECT PI();

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Fungsi POWER

Fungsi POWER adalah fungsi eksponensial yang meningkatkan ekspresi numerik ke kekuatan ekspresi numerik kedua. Misalnya, 2 hingga daya ketiga dihitung sebagai`POWER(2,3)`, dengan hasil dari8.

Sintaksis

```
{POWER(expression1, expression2)
```

Argumen

ekspresi1

Ekspresi numerik yang akan dinaikkan. Harus berupa`INTEGER`,`DECIMAL`, atau tipe `FLOAT` data.

ekspresi2

Kekuatan untuk meningkatkan ekspresi1. Harus berupa`INTEGER`,`DECIMAL`, atau tipe `FLOAT` data.

Jenis pengembalian

`DOUBLE PRECISION`

Contoh

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

Fungsi RADIANS

Fungsi RADIANS mengubah sudut dalam derajat ke ekuivalennya dalam radian.

Sintaksis

```
RADIANS(number)
```

Pendapat

jumlah

Parameter input adalah `DOUBLE PRECISION` angka.

Jenis pengembalian

`DOUBLE PRECISION`

Contoh

Untuk mengembalikan radian setara 180 derajat, gunakan contoh berikut.

```
SELECT RADIANS(180);

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

Fungsi RAND

Fungsi RAND menghasilkan angka floating-point acak antara 0 dan 1. Fungsi RAND menghasilkan nomor acak baru setiap kali dipanggil.

Sintaksis

```
RAND()
```

Jenis pengembalian

RANDOM mengembalikan DOUBLE.

Contoh

Contoh berikut menghasilkan kolom angka floating-point acak antara 0 dan 1 untuk setiap baris dalam tabel. `squirrels` Output yang dihasilkan akan menjadi kolom tunggal yang berisi daftar nilai desimal acak, dengan satu nilai untuk setiap baris dalam tabel tupai.

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

Jenis kueri ini berguna ketika Anda perlu menghasilkan angka acak, misalnya, untuk mensimulasikan peristiwa acak atau untuk memperkenalkan keacakan ke dalam analisis data Anda. Dalam konteks

`squirrels` tabel, mungkin digunakan untuk menetapkan nilai acak untuk setiap tupai, yang kemudian dapat digunakan untuk pemrosesan atau analisis lebih lanjut.

fungsi RANDOM

Fungsi RANDOM menghasilkan nilai acak antara 0,0 (inklusif) dan 1,0 (eksklusif).

Sintaksis

```
RANDOM()
```

Jenis pengembalian

RANDOM mengembalikan nomor PRESISI GANDA.

Contoh

1. Hitung nilai acak antara 0 dan 99. Jika angka acak adalah 0 hingga 1, kueri ini menghasilkan angka acak dari 0 hingga 100:

   ```
   select cast (random() * 100 as int);

   INTEGER
   ------
   24
   (1 row)
   ```

2. Ambil sampel acak seragam dari 10 item:

   ```
   select *
   from sales
   order by random()
   limit 10;
   ```

   Sekarang ambil sampel acak 10 item, tetapi pilih item secara proporsional dengan harganya. Misalnya, item yang dua kali harga yang lain akan dua kali lebih mungkin muncul dalam hasil kueri:

   ```
   select *
   from sales
   order by log(1 - random()) / pricepaid
   ```

```
limit 10;
```

3. Contoh ini menggunakan perintah SET untuk menetapkan nilai SEED sehingga RANDOM menghasilkan urutan angka yang dapat diprediksi.

   Pertama, kembalikan tiga bilangan bulat RANDOM tanpa mengatur nilai SEED terlebih dahulu:

```
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)
```

   Sekarang, atur nilai SEED ke `.25`, dan kembalikan tiga angka RANDOM lagi:

```
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)
```

Terakhir, setel ulang nilai SEED ke.25, dan verifikasi bahwa RANDOM mengembalikan hasil yang sama dengan tiga panggilan sebelumnya:

```
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)
```

Fungsi ROUND

Fungsi ROUND membulatkan angka ke bilangan bulat atau desimal terdekat.

Fungsi ROUND secara opsional dapat menyertakan argumen kedua sebagai bilangan bulat untuk menunjukkan jumlah tempat desimal untuk pembulatan, di kedua arah. Ketika Anda tidak memberikan argumen kedua, fungsi dibulatkan ke bilangan bulat terdekat. Ketika argumen kedua >n ditentukan, fungsi dibulatkan ke angka terdekat dengan n tempat desimal presisi.

Sintaksis

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

Pendapat

jumlah

Angka atau ekspresi yang mengevaluasi angka. Ini bisa berupa DESIMAL atau FLOAT8 tipe. AWS Clean Rooms dapat mengonversi tipe data lain sesuai aturan konversi implisit.

bilangan bulat (opsional)

Bilangan bulat yang menunjukkan jumlah tempat desimal untuk pembulatan di kedua arah.

Jenis pengembalian

ROUND mengembalikan tipe data numerik yang sama dengan argumen masukan.

Contoh

Bulatkan komisi yang dibayarkan untuk transaksi tertentu ke seluruh nomor terdekat.

```
select commission, round(commission)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    28
(1 row)
```

Bulatkan komisi yang dibayarkan untuk transaksi tertentu ke tempat desimal pertama.

```
select commission, round(commission, 1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |  28.1
(1 row)
```

Untuk kueri yang sama, perluas presisi ke arah yang berlawanan.

```
select commission, round(commission, -1)
from sales where salesid=10000;
```

```
commission | round
-----------+-------
     28.05 |    30
(1 row)
```

Fungsi SIGN

Fungsi SIGN mengembalikan tanda (positif atau negatif) dari suatu angka. Hasil dari fungsi SIGN adalah `1`-`1`,, atau `0` menunjukkan tanda argumen.

Sintaksis

```
SIGN (number)
```

Pendapat

jumlah

Angka atau ekspresi yang mengevaluasi angka. Itu bisa menjadi DECIMALor FLOAT8 tipenya. AWS Clean Rooms dapat mengonversi tipe data lain sesuai aturan konversi implisit.

Jenis pengembalian

SIGN mengembalikan tipe data numerik yang sama dengan argumen masukan. Jika input adalah DESIMAL, outputnya adalah DESIMAL (1,0).

Contoh

Untuk menentukan tanda komisi yang dibayarkan untuk transaksi tertentu dari tabel PENJUALAN, gunakan contoh berikut.

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

Fungsi SIN

SIN adalah fungsi trigonometri yang mengembalikan sinus suatu angka. Nilai yang dikembalikan adalah antara `-1` dan`1`.

Sintaksis

```
SIN(number)
```

Pendapat

jumlah

`DOUBLE PRECISION`Angka dalam radian.

Jenis pengembalian

`DOUBLE PRECISION`

Contoh

Untuk mengembalikan sinus`-PI`, gunakan contoh berikut.

```
SELECT SIN(-PI());

+-------------------------+
|           sin           |
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

Fungsi SQRT

Fungsi SQRT mengembalikan akar kuadrat dari nilai numerik. Akar kuadrat adalah angka yang dikalikan dengan sendirinya untuk mendapatkan nilai yang diberikan.

Sintaksis

```
SQRT (expression)
```

Pendapat

ekspresi

Ekspresi harus memiliki tipe data integer, desimal, atau floating-point. Ekspresi dapat mencakup fungsi. Sistem mungkin melakukan konversi tipe implisit.

Jenis pengembalian

SQRT mengembalikan nomor PRESISI GANDA.

Contoh

Contoh berikut mengembalikan akar kuadrat dari angka.

```
select sqrt(16);

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

Contoh berikut melakukan konversi tipe implisit.

```
select sqrt('16');

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

Contoh sarang berikut berfungsi untuk melakukan tugas yang lebih kompleks.

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

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

Contoh berikut menghasilkan panjang jari-jari ketika diberi luas lingkaran. Ini menghitung radius dalam inci, misalnya, ketika diberi luas dalam inci persegi. Area dalam sampel adalah 20.

```
select sqrt(20/pi());
```

Ini mengembalikan nilai 5.046265044040321.

Contoh berikut mengembalikan akar kuadrat untuk nilai KOMISI dari tabel PENJUALAN. Kolom KOMISI adalah kolom DESIMAL. Contoh ini menunjukkan bagaimana Anda dapat menggunakan fungsi dalam kueri dengan logika kondisional yang lebih kompleks.

```
select sqrt(commission)
from sales where salesid < 10 order by salesid;

sqrt
------------------
10.4498803820905
3.37638860322683
7.24568837309472
5.1234753829798
...
```

Kueri berikut mengembalikan akar kuadrat bulat untuk set nilai KOMISI yang sama.

```
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
...
```

Untuk informasi selengkapnya tentang data sampel di AWS Clean Rooms, lihat Database sampel.

Fungsi TRUNC

Fungsi TRUNC memotong angka ke bilangan bulat atau desimal sebelumnya.

Fungsi TRUNC secara opsional dapat menyertakan argumen kedua sebagai bilangan bulat untuk menunjukkan jumlah tempat desimal untuk pembulatan, di kedua arah. Ketika Anda tidak memberikan argumen kedua, fungsi dibulatkan ke bilangan bulat terdekat. Ketika argumen kedua >n ditentukan, fungsi dibulatkan ke angka terdekat dengan>n tempat desimal presisi. Fungsi ini juga memotong stempel waktu dan mengembalikan tanggal.

Sintaksis

```
TRUNC (number [ , integer ] |
timestamp )
```

Argumen

jumlah

Angka atau ekspresi yang mengevaluasi angka. Ini bisa berupa DESIMAL atau FLOAT8 tipe. AWS Clean Rooms dapat mengonversi tipe data lain sesuai aturan konversi implisit.

bilangan bulat (opsional)

Bilangan bulat yang menunjukkan jumlah tempat desimal presisi, di kedua arah. Jika tidak ada bilangan bulat yang disediakan, angka tersebut terpotong sebagai bilangan bulat; jika bilangan bulat ditentukan, angka tersebut dipotong ke tempat desimal yang ditentukan.

stempel waktu

Fungsi ini juga dapat mengembalikan tanggal dari stempel waktu. (Untuk mengembalikan nilai stempel waktu dengan `00:00:00` waktu, lemparkan hasil fungsi ke stempel waktu.)

Jenis pengembalian

TRUNC mengembalikan tipe data yang sama dengan argumen masukan pertama. Untuk stempel waktu, TRUNC mengembalikan tanggal.

Contoh

Memangkas komisi yang dibayarkan untuk transaksi penjualan tertentu.

```
select commission, trunc(commission)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   111

(1 row)
```

Memangkas nilai komisi yang sama ke tempat desimal pertama.

```
select commission, trunc(commission,1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 | 111.1

(1 row)
```

Potong komisi dengan nilai negatif untuk argumen kedua; `111.15` dibulatkan ke bawah menjadi. `110`

```
select commission, trunc(commission,-1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   110
(1 row)
```

Kembalikan bagian tanggal dari hasil fungsi SYSDATE (yang mengembalikan stempel waktu):

```
select sysdate;

timestamp
----------------------------
2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc
------------
2011-07-21
(1 row)
```

Terapkan fungsi TRUNC ke kolom TIMESTAMP. Jenis pengembalian adalah tanggal.

```
select trunc(starttime) from event
order by eventid limit 1;

trunc
```

```
------------
2008-01-25
(1 row)
```

Fungsi skalar

Bagian ini menjelaskan fungsi skalar yang didukung di AWS Clean Rooms Spark SQL. Fungsi skalar adalah fungsi yang mengambil satu atau lebih nilai sebagai input dan mengembalikan nilai tunggal sebagai output. Fungsi skalar beroperasi pada baris atau elemen individu dan menghasilkan hasil tunggal untuk setiap input.

Fungsi skalar, seperti SIZE, berbeda dari jenis fungsi SQL lainnya, seperti fungsi agregat (hitung, jumlah, rata-rata) dan fungsi penghasil tabel (meledak, meratakan). Jenis fungsi lain ini beroperasi pada beberapa baris atau menghasilkan beberapa baris, sedangkan fungsi skalar bekerja pada baris atau elemen individu.

Topik

- Fungsi UKURAN

Fungsi UKURAN

Fungsi SIZE mengambil array, peta, atau string yang ada sebagai argumen dan mengembalikan nilai tunggal yang mewakili ukuran atau panjang struktur data tersebut. Itu tidak membuat struktur data baru. Ini digunakan untuk menanyakan dan menganalisis properti struktur data yang ada, bukan untuk membuat yang baru.

Fungsi ini berguna untuk menentukan jumlah elemen dalam array atau panjang string. Ini dapat sangat membantu ketika bekerja dengan array dan struktur data lainnya di SQL, karena memungkinkan Anda untuk mendapatkan informasi tentang ukuran atau kardinalitas data.

Sintaks

```
size(expr)
```

Pendapat

expr

Ekspresi ARRAY, MAP, atau STRING.

Jenis pengembalian

Fungsi SIZE mengembalikan INTEGER.

Contoh

Dalam contoh ini, fungsi SIZE diterapkan ke array`['b', 'd', 'c', 'a']`, dan mengembalikan nilai`4`, yang merupakan jumlah elemen dalam array.

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

Dalam contoh ini, fungsi SIZE diterapkan ke peta`{'a': 1, 'b': 2}`, dan mengembalikan nilai`2`, yang merupakan jumlah pasangan kunci-nilai di peta.

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

Dalam contoh ini, fungsi SIZE diterapkan ke string`'hello world'`, dan mengembalikan nilai`11`, yang merupakan jumlah karakter dalam string.

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

Fungsi string

Fungsi string memproses dan memanipulasi string karakter atau ekspresi yang mengevaluasi string karakter. Ketika argumen string dalam fungsi ini adalah nilai literal, itu harus diapit dalam tanda kutip tunggal. Tipe data yang didukung termasuk CHAR dan VARCHAR.

Bagian berikut menyediakan nama fungsi, sintaks, dan deskripsi untuk fungsi yang didukung. Semua offset menjadi string berbasis satu.

Topik

- || Operator (Penggabungan)
- Fungsi BTRIM
- Fungsi CONCAT
- Fungsi FORMAT_STRING

- Fungsi KIRI dan KANAN
- Fungsi PANJANG
- Fungsi LOWER
- Fungsi LPAD dan RPAD
- Fungsi LTRIM
- Fungsi POSISI
- Fungsi REGEXP_COUNT
- Fungsi REGEXP_INSTR
- Fungsi REGEXP_REPLACE
- Fungsi REGEXP_SUBSTR
- Fungsi REPEAT
- GANTI fungsi
- Fungsi REVERSE
- Fungsi RTRIM
- Fungsi SPLIT
- Fungsi SPLIT_PART
- Fungsi SUBSTRING
- FUNGSI TRANSLATE
- Fungsi TRIM
- Fungsi UPPER
- Fungsi UUID

|| Operator (Penggabungan)

Menggabungkan dua ekspresi di kedua sisi simbol || dan mengembalikan ekspresi gabungan.

Operator penggabungan mirip dengan. Fungsi CONCAT

> **Note**
>
> Untuk fungsi CONCAT dan operator penggabungan, jika salah satu atau kedua ekspresi adalah nol, hasil penggabungan adalah nol.

Sintaksis

```
expression1 || expression2
```

Argumen

ekspresi1, ekspresi2

Kedua argumen dapat berupa string atau ekspresi karakter fixed-length atau variable-length.

Jenis pengembalian

Operator || mengembalikan string. Jenis string sama dengan argumen masukan.

Contoh

Contoh berikut menggabungkan bidang FIRSTNAME dan LASTNAME dari tabel 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)
```

Untuk menggabungkan kolom yang mungkin berisi nol, gunakan ekspresi. Fungsi NVL dan COALESCE Contoh berikut menggunakan NVL untuk mengembalikan 0 setiap kali NULL ditemui.

```
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
```

Fungsi BTRIM

Fungsi BTRIM memangkas string dengan menghapus bagian depan dan belakang kosong atau dengan menghapus karakter utama dan belakang yang cocok dengan string tertentu opsional.

Sintaksis

```
BTRIM(string [, trim_chars ] )
```

Argumen

tali

String input VARCHAR yang akan dipangkas.

trim_chars

String VARCHAR yang berisi karakter yang akan dicocokkan.

Jenis pengembalian

Fungsi BTRIM mengembalikan string VARCHAR.

Contoh

Contoh berikut memangkas bagian depan dan belakang kosong dari string: `' abc '`

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

Contoh berikut menghapus string depan dan trailing dari `'xyz'` string. `'xyzaxyzbxyzcxyz'` Kejadian leading dan trailing `'xyz'` dihapus, tetapi kejadian yang internal di dalam string tidak dihapus.

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

Contoh berikut menghapus bagian depan dan belakang dari string `'setuphistorycassettes'` yang cocok dengan salah satu karakter dalam daftar trim_chars. `'tes'` Apa pun `te`,, atau `s` yang terjadi sebelum karakter lain yang tidak ada dalam daftar trim_chars di awal atau akhir string input dihapus.

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
-----------------
 uphistoryca
```

Fungsi CONCAT

Fungsi CONCAT menggabungkan dua ekspresi dan mengembalikan ekspresi yang dihasilkan. Untuk menggabungkan lebih dari dua ekspresi, gunakan fungsi CONCAT bersarang. Operator penggabungan (| |) antara dua ekspresi menghasilkan hasil yang sama dengan fungsi CONCAT.

> **Note**
>
> Untuk fungsi CONCAT dan operator penggabungan, jika salah satu atau kedua ekspresi adalah nol, hasil penggabungan adalah nol.

Sintaksis

```
CONCAT ( expression1, expression2 )
```

Argumen

ekspresi1, ekspresi2

Kedua argumen dapat berupa string karakter fixed-length, string karakter panjang variabel, ekspresi biner, atau ekspresi yang mengevaluasi salah satu input ini.

Jenis pengembalian

CONCAT mengembalikan ekspresi. Tipe data ekspresi adalah tipe yang sama dengan argumen masukan.

Jika ekspresi input dari jenis yang berbeda, AWS Clean Rooms mencoba untuk secara implisit mengetik cast salah satu ekspresi. Jika nilai tidak dapat dilemparkan, kesalahan dikembalikan.

Contoh

Contoh berikut menggabungkan dua literal karakter:

```
select concat('December 25, ', '2008');

concat
-------------------
December 25, 2008
(1 row)
```

Kueri berikut, menggunakan || operator bukan CONCAT, menghasilkan hasil yang sama:

```
select 'December 25, '||'2008';

concat
-------------------
December 25, 2008
(1 row)
```

Contoh berikut menggunakan dua fungsi CONCAT untuk menggabungkan tiga string karakter:

```
select concat('Thursday, ', concat('December 25, ', '2008'));

concat
----------------------------
Thursday, December 25, 2008
(1 row)
```

Untuk menggabungkan kolom yang mungkin berisi nol, gunakan kolom. Fungsi NVL dan COALESCE Contoh berikut menggunakan NVL untuk mengembalikan 0 setiap kali NULL ditemui.

```
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)
```

Kueri berikut menggabungkan nilai CITY dan STATE dari tabel VENUE:

```
select concat(venuecity, venuestate)
from venue
where venueseats > 75000
order by venueseats;

concat
-------------------
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

Kueri berikut menggunakan fungsi CONCAT bersarang. Kueri menggabungkan nilai CITY dan STATE dari tabel VENUE tetapi membatasi string yang dihasilkan dengan koma dan spasi:

```
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)
```

Fungsi FORMAT_STRING

Fungsi FORMAT_STRING membuat string yang diformat dengan mengganti placeholder dalam string template dengan argumen yang disediakan. Ia mengembalikan string diformat dari string format printf-style.

Fungsi FORMAT_STRING bekerja dengan mengganti placeholder dalam string template dengan nilai yang sesuai diteruskan sebagai argumen. Jenis pemformatan string ini dapat berguna ketika Anda perlu membangun string secara dinamis yang mencakup campuran teks statis dan data dinamis, seperti saat menghasilkan pesan keluaran, laporan, atau jenis teks informatif lainnya. Fungsi FORMAT_STRING menyediakan cara ringkas dan mudah dibaca untuk membuat jenis string yang diformat ini, membuatnya lebih mudah untuk memelihara dan memperbarui kode yang menghasilkan output.

Sintaksis

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

Pendapat

strfmt

Ekspresi STRING.

obj

Ekspresi STRING atau numerik.

Jenis pengembalian

FORMAT_STRING mengembalikan STRING.

Contoh

Contoh berikut berisi string template yang berisi dua placeholder: `%d` untuk nilai desimal (integer), dan `%s` untuk nilai string. `%d`Placeholder diganti dengan nilai desimal (integer) (`100`), dan placeholder %s diganti dengan nilai string (). `"days"` Outputnya adalah string template dengan placeholder diganti dengan argumen yang disediakan:. `"Hello World 100 days"`

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

Fungsi KIRI dan KANAN

Fungsi-fungsi ini mengembalikan jumlah karakter paling kiri atau paling kanan yang ditentukan dari string karakter.

Jumlahnya didasarkan pada jumlah karakter, bukan byte, sehingga karakter multibyte dihitung sebagai karakter tunggal.

Sintaksis

```
LEFT ( string,  integer )

RIGHT ( string,  integer )
```

Argumen

tali

String karakter apa pun atau ekspresi apa pun yang mengevaluasi string karakter.

bilangan bulat

Integer positif.

Jenis pengembalian

KIRI dan KANAN mengembalikan string VARCHAR.

Contoh

Contoh berikut mengembalikan 5 karakter paling kiri dan paling kanan 5 dari nama acara yang memiliki IDs antara 1000 dan 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)
```

Fungsi PANJANG

Fungsi LOWER

Mengkonversi string ke huruf kecil. LOWER mendukung karakter multibyte UTF-8, hingga maksimal empat byte per karakter.

Sintaksis

```
LOWER(string)
```

Pendapat

tali

Parameter input adalah string VARCHAR (atau tipe data lainnya, seperti CHAR, yang dapat secara implisit dikonversi ke VARCHAR).

Jenis pengembalian

Fungsi LOWER mengembalikan string karakter yang merupakan tipe data yang sama dengan string input.

Contoh

Contoh berikut mengonversi bidang CATNAME menjadi huruf kecil:

```
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)
```

Fungsi LPAD dan RPAD

Fungsi-fungsi ini menambahkan atau menambahkan karakter ke string, berdasarkan panjang tertentu.

Sintaksis

```
LPAD (string1, length, [ string2 ])
```

```
RPAD (string1, length, [ string2 ])
```

Argumen

senar1

String karakter atau ekspresi yang mengevaluasi string karakter, seperti nama kolom karakter.

panjang

Sebuah integer yang mendefinisikan panjang hasil dari fungsi. Panjang string didasarkan pada jumlah karakter, bukan byte, sehingga karakter multi-byte dihitung sebagai karakter tunggal. Jika string1 lebih panjang dari panjang yang ditentukan, itu terpotong (di sebelah kanan). Jika panjang adalah angka negatif, hasil dari fungsi adalah string kosong.

senar2

Satu atau lebih karakter yang ditambahkan atau ditambahkan ke string1. Argumen ini opsional; jika tidak ditentukan, spasi digunakan.

Jenis pengembalian

Fungsi-fungsi ini mengembalikan tipe data VARCHAR.

Contoh

Potong satu set nama acara tertentu menjadi 20 karakter dan tambahkan nama yang lebih pendek dengan spasi:

```
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)
```

Potong set nama acara yang sama menjadi 20 karakter tetapi tambahkan nama yang lebih pendek dengan. `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)
```

Fungsi LTRIM

Memangkas karakter dari awal string. Menghapus string terpanjang yang hanya berisi karakter dalam daftar karakter trim. Pemangkasan selesai ketika karakter trim tidak muncul di string input.

Sintaksis

```
LTRIM( string [, trim_chars] )
```

Argumen

tali

Sebuah kolom string, ekspresi, atau string literal yang akan dipangkas.

trim_chars

Sebuah kolom string, ekspresi, atau string literal yang mewakili karakter yang akan dipangkas dari awal string. Jika tidak ditentukan, spasi digunakan sebagai karakter trim.

Jenis pengembalian

Fungsi LTRIM mengembalikan string karakter yang merupakan tipe data yang sama dengan string input (CHAR atau VARCHAR).

Contoh

Contoh berikut memangkas tahun dari `listime` kolom. Karakter trim dalam string literal `'2008-'` menunjukkan karakter yang akan dipangkas dari kiri. Jika Anda menggunakan karakter trim`'028-'`, Anda mencapai hasil yang sama.

```
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 menghapus salah satu karakter di trim_chars ketika mereka muncul di awal string. Contoh berikut memangkas karakter 'C', 'D', dan 'G' ketika mereka muncul di awal VENUENAME, yang merupakan kolom 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
```

Contoh berikut menggunakan karakter trim 2 yang diambil dari `venueid` kolom.

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

Contoh berikut tidak memangkas karakter apa pun 2 karena a ditemukan sebelum karakter `'0'` trim.

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

Contoh berikut menggunakan karakter trim spasi default dan memangkas dua spasi dari awal string.

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

Fungsi POSISI

Mengembalikan lokasi substring tertentu dalam string.

Sintaksis

```
POSITION(substring IN string )
```

Argumen

substring

Substring untuk mencari di dalam string.

tali

String atau kolom yang akan dicari.

Jenis pengembalian

Fungsi POSITION mengembalikan bilangan bulat yang sesuai dengan posisi substring (berbasis satu, bukan berbasis nol). Posisi didasarkan pada jumlah karakter, bukan byte, sehingga karakter multi-byte dihitung sebagai karakter tunggal.

Catatan penggunaan

POSITION mengembalikan 0 jika substring tidak ditemukan dalam string:

```
select position('dog' in 'fish');

position
----------
 0
(1 row)
```

Contoh

Contoh berikut menunjukkan posisi string `fish` dalam kata`dogfish`:

```
select position('fish' in 'dogfish');

position
----------
 4
(1 row)
```

Contoh berikut mengembalikan jumlah transaksi penjualan dengan KOMISI lebih dari 999.00 dari tabel PENJUALAN:

```
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)
```

Fungsi REGEXP_COUNT

Mencari string untuk pola ekspresi reguler dan mengembalikan integer yang menunjukkan berapa kali pola terjadi dalam string. Jika tidak ada kecocokan yang ditemukan, maka fungsi mengembalikan 0.

Sintaksis

```
REGEXP_COUNT ( source_string, pattern [, position [, parameters ] ] )
```

Argumen

source_string

Ekspresi string, seperti nama kolom, yang akan dicari.

pola

Sebuah string literal yang mewakili pola ekspresi reguler.

posisi

Sebuah integer positif yang menunjukkan posisi dalam source_string untuk mulai mencari. Posisi didasarkan pada jumlah karakter, bukan byte, sehingga karakter multibyte dihitung sebagai karakter tunggal. Default-nya adalah 1. Jika posisi kurang dari 1, pencarian dimulai pada karakter pertama source_string. Jika posisi lebih besar dari jumlah karakter di source_string, hasilnya adalah 0.

parameter

Satu atau lebih string literal yang menunjukkan bagaimana fungsi cocok dengan pola. Nilai yang mungkin adalah sebagai berikut:

- c — Lakukan pencocokan peka huruf besar/kecil. Defaultnya adalah menggunakan pencocokan peka huruf besar/kecil.
- i — Lakukan pencocokan case-insensitive.
- p — Menafsirkan pola dengan dialek Perl Compatible Regular Expression (PCRE).

Jenis pengembalian

Bilangan Bulat

Contoh

Contoh berikut menghitung berapa kali urutan tiga huruf terjadi.

```
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');

 regexp_count
 --------------
            8
```

Contoh berikut menghitung berapa kali nama domain tingkat atas adalah salah satu atauorg. 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
```

Contoh berikut menghitung kemunculan string, menggunakan pencocokan FOX case-insensitive.

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

 regexp_count
 --------------
            1
```

Contoh berikut menggunakan pola yang ditulis dalam dialek PCRE untuk menemukan kata-kata yang mengandung setidaknya satu angka dan satu huruf kecil. Ini menggunakan ?= operator, yang memiliki konotasi pandangan ke depan tertentu di PCRE. Contoh ini menghitung jumlah kemunculan kata-kata tersebut, dengan pencocokan peka huruf besar/kecil.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

 regexp_count
 --------------
            2
```

Contoh berikut menggunakan pola yang ditulis dalam dialek PCRE untuk menemukan kata-kata yang mengandung setidaknya satu angka dan satu huruf kecil. Ini menggunakan ?= operator, yang memiliki konotasi khusus di PCRE. Contoh ini menghitung jumlah kemunculan kata-kata tersebut, tetapi berbeda dari contoh sebelumnya karena menggunakan pencocokan case-insensitive.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');

 regexp_count
 --------------
```

```
            3
```

Fungsi REGEXP_INSTR

Mencari string untuk pola ekspresi reguler dan mengembalikan integer yang menunjukkan posisi awal atau posisi akhir dari substring yang cocok. Jika tidak ada kecocokan yang ditemukan, maka fungsi mengembalikan 0. REGEXP_INSTR mirip dengan fungsi POSITION, tetapi memungkinkan Anda mencari string untuk pola ekspresi reguler.

Sintaksis

```
REGEXP_INSTR ( source_string, pattern [, position [, occurrence] [, option
 [, parameters ] ] ] ] )
```

Argumen

source_string

Ekspresi string, seperti nama kolom, yang akan dicari.

pola

Sebuah string literal yang mewakili pola ekspresi reguler.

posisi

Sebuah integer positif yang menunjukkan posisi dalam source_string untuk mulai mencari. Posisi didasarkan pada jumlah karakter, bukan byte, sehingga karakter multibyte dihitung sebagai karakter tunggal. Default-nya adalah 1. Jika posisi kurang dari 1, pencarian dimulai pada karakter pertama source_string. Jika posisi lebih besar dari jumlah karakter di source_string, hasilnya adalah 0.

kejadian

Sebuah bilangan bulat positif yang menunjukkan kemunculan pola yang akan digunakan. REGEXP_INSTR melewatkan kejadian pertama -1 pertandingan. Default-nya adalah 1. Jika kejadian kurang dari 1 atau lebih besar dari jumlah karakter di source_string, pencarian diabaikan dan hasilnya adalah 0.

pilihan

Nilai yang menunjukkan apakah akan mengembalikan posisi karakter pertama pertandingan (0) atau posisi karakter pertama setelah akhir pertandingan (1). Nilai bukan nol sama dengan 1. Nilai default-nya adalah 0.

parameter

Satu atau lebih string literal yang menunjukkan bagaimana fungsi cocok dengan pola. Nilai yang mungkin adalah sebagai berikut:

- c — Lakukan pencocokan peka huruf besar/kecil. Defaultnya adalah menggunakan pencocokan peka huruf besar/kecil.
- i — Lakukan pencocokan case-insensitive.
- e — Ekstrak substring menggunakan subexpression.

  Jika pola menyertakan subexpression, REGEXP_INSTR cocok dengan substring menggunakan subexpression pertama dalam pola. REGEXP_INSTR hanya mempertimbangkan subexpression pertama; subexpressions tambahan diabaikan. Jika pola tidak memiliki subexpression, REGEXP_INSTR mengabaikan parameter 'e'.
- p — Menafsirkan pola dengan dialek Perl Compatible Regular Expression (PCRE).

Jenis pengembalian

Bilangan Bulat

Contoh

Contoh berikut mencari @ karakter yang memulai nama domain dan mengembalikan posisi awal kecocokan pertama.

```
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
```

Contoh berikut mencari varian kata `Center` dan mengembalikan posisi awal kecocokan pertama.

```
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
```

Contoh berikut menemukan posisi awal dari kemunculan pertama stringFOX, menggunakan logika pencocokan case-insensitive.

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

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

Contoh berikut menggunakan pola yang ditulis dalam dialek PCRE untuk menemukan kata-kata yang mengandung setidaknya satu angka dan satu huruf kecil. Ini menggunakan ?= operator, yang memiliki konotasi pandangan ke depan tertentu di PCRE. Contoh ini menemukan posisi awal dari kata kedua tersebut.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

 regexp_instr
 --------------
           21
```

Contoh berikut menggunakan pola yang ditulis dalam dialek PCRE untuk menemukan kata-kata yang mengandung setidaknya satu angka dan satu huruf kecil. Ini menggunakan ?= operator, yang memiliki konotasi pandangan ke depan tertentu di PCRE. Contoh ini menemukan posisi awal dari kata kedua tersebut, tetapi berbeda dari contoh sebelumnya karena menggunakan pencocokan case-insensitive.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');
```

```
 regexp_instr
 --------------
           15
```

Fungsi REGEXP_REPLACE

Mencari string untuk pola ekspresi reguler dan menggantikan setiap kemunculan pola dengan string yang ditentukan. REGEXP_REPLACE mirip dengan GANTI fungsi, tetapi memungkinkan Anda mencari string untuk pola ekspresi reguler.

REGEXP_REPLACE mirip dengan FUNGSI TRANSLATE dan GANTI fungsi, kecuali bahwa TRANSLATE membuat beberapa substitusi karakter tunggal dan REPLACE menggantikan satu seluruh string dengan string lain, sementara REGEXP_REPLACE memungkinkan Anda mencari string untuk pola ekspresi reguler.

Sintaksis

```
REGEXP_REPLACE ( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

Argumen

source_string

Ekspresi string, seperti nama kolom, yang akan dicari.

pola

Sebuah string literal yang mewakili pola ekspresi reguler.

replace_string

Ekspresi string, seperti nama kolom, yang akan menggantikan setiap kemunculan pola. Defaultnya adalah string kosong (“”).

posisi

Sebuah integer positif yang menunjukkan posisi dalam source_string untuk mulai mencari. Posisi didasarkan pada jumlah karakter, bukan byte, sehingga karakter multibyte dihitung sebagai karakter tunggal. Default-nya adalah 1. Jika posisi kurang dari 1, pencarian dimulai pada karakter pertama source_string. Jika posisi lebih besar dari jumlah karakter di source_string, hasilnya adalah source_string.

parameter

Satu atau lebih string literal yang menunjukkan bagaimana fungsi cocok dengan pola. Nilai yang mungkin adalah sebagai berikut:

- c — Lakukan pencocokan peka huruf besar/kecil. Defaultnya adalah menggunakan pencocokan peka huruf besar/kecil.
- i — Lakukan pencocokan case-insensitive.
- p — Menafsirkan pola dengan dialek Perl Compatible Regular Expression (PCRE).

Jenis pengembalian

VARCHAR

Jika salah satu pola atau replace_string adalah NULL, pengembaliannya adalah NULL.

Contoh

Contoh berikut menghapus @ dan nama domain dari alamat email.

```
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
```

Contoh berikut menggantikan nama domain alamat email dengan nilai ini:`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
```

Contoh berikut menggantikan semua kemunculan string `FOX` dalam nilai, menggunakan pencocokan `quick brown fox` case-insensitive.

```
SELECT regexp_replace('the fox', 'FOX', 'quick brown fox', 1, 'i');

   regexp_replace
---------------------
 the quick brown fox
```

Contoh berikut menggunakan pola yang ditulis dalam dialek PCRE untuk menemukan kata-kata yang mengandung setidaknya satu angka dan satu huruf kecil. Ini menggunakan `?=` operator, yang memiliki konotasi pandangan ke depan tertentu di PCRE. Contoh ini menggantikan setiap kemunculan kata seperti itu dengan nilainya`[hidden]`.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

        regexp_replace
------------------------------
 [hidden] plain A1234 [hidden]
```

Contoh berikut menggunakan pola yang ditulis dalam dialek PCRE untuk menemukan kata-kata yang mengandung setidaknya satu angka dan satu huruf kecil. Ini menggunakan `?=` operator, yang memiliki konotasi pandangan ke depan tertentu di PCRE. Contoh ini menggantikan setiap kemunculan kata seperti itu dengan nilai`[hidden]`, tetapi berbeda dari contoh sebelumnya karena menggunakan pencocokan case-insensitive.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

          regexp_replace
---------------------------------
 [hidden] plain [hidden] [hidden]
```

Fungsi REGEXP_SUBSTR

Mengembalikan karakter dari string dengan mencarinya untuk pola ekspresi reguler. REGEXP_SUBSTR mirip dengan Fungsi SUBSTRING fungsinya, tetapi memungkinkan Anda mencari string untuk pola ekspresi reguler. Jika fungsi tidak dapat mencocokkan ekspresi reguler dengan karakter apa pun dalam string, ia mengembalikan string kosong.

Sintaksis

```
REGEXP_SUBSTR ( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

Argumen

source_string

Ekspresi string yang akan dicari.

pola

Sebuah string literal yang mewakili pola ekspresi reguler.

posisi

Sebuah integer positif yang menunjukkan posisi dalam source_string untuk mulai mencari. Posisi didasarkan pada jumlah karakter, bukan byte, sehingga karakter multi-byte dihitung sebagai karakter tunggal. Default-nya adalah 1. Jika posisi kurang dari 1, pencarian dimulai pada karakter pertama source_string. Jika posisi lebih besar dari jumlah karakter di source_string, hasilnya adalah string kosong (“”).

kejadian

Sebuah bilangan bulat positif yang menunjukkan kemunculan pola yang akan digunakan. REGEXP_SUBSTR melewatkan kejadian pertama -1 pertandingan. Default-nya adalah 1. Jika kejadian kurang dari 1 atau lebih besar dari jumlah karakter di source_string, pencarian diabaikan dan hasilnya adalah NULL.

parameter

Satu atau lebih string literal yang menunjukkan bagaimana fungsi cocok dengan pola. Nilai yang mungkin adalah sebagai berikut:

- c — Lakukan pencocokan peka huruf besar/kecil. Defaultnya adalah menggunakan pencocokan peka huruf besar/kecil.
- i — Lakukan pencocokan case-insensitive.

- e — Ekstrak substring menggunakan subexpression.

  Jika pola menyertakan subexpression, REGEXP_SUBSTR cocok dengan substring menggunakan subexpression pertama dalam pola. Sebuah subexpression adalah ekspresi dalam pola yang dikurung dengan tanda kurung. Misalnya, untuk pola `'This is a (\\w+)'` cocok ekspresi pertama dengan string `'This is a '` diikuti oleh sebuah kata. Alih-alih mengembalikan pola, REGEXP_SUBSTR dengan e parameter hanya mengembalikan string di dalam subexpression.

  REGEXP_SUBSTR hanya mempertimbangkan subexpression pertama; subexpressions tambahan diabaikan. Jika pola tidak memiliki subexpression, REGEXP_SUBSTR mengabaikan parameter 'e'.
- p — Menafsirkan pola dengan dialek Perl Compatible Regular Expression (PCRE).

Jenis pengembalian

VARCHAR

Contoh

Contoh berikut mengembalikan bagian dari alamat email antara karakter @ dan ekstensi domain.

```
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
```

Contoh berikut mengembalikan bagian dari input yang sesuai dengan kejadian pertama stringF0X, menggunakan pencocokan case-insensitive.

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

 regexp_substr
---------------
```

```
 fox
```

Contoh berikut mengembalikan bagian pertama dari input yang dimulai dengan huruf kecil. Ini secara fungsional identik dengan pernyataan SELECT yang sama tanpa `c` parameter.

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');

 regexp_substr
---------------
 abc
```

Contoh berikut menggunakan pola yang ditulis dalam dialek PCRE untuk menemukan kata-kata yang mengandung setidaknya satu angka dan satu huruf kecil. Ini menggunakan `?=` operator, yang memiliki konotasi pandangan ke depan tertentu di PCRE. Contoh ini mengembalikan bagian dari input yang sesuai dengan kata kedua tersebut.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

Contoh berikut menggunakan pola yang ditulis dalam dialek PCRE untuk menemukan kata-kata yang mengandung setidaknya satu angka dan satu huruf kecil. Ini menggunakan `?=` operator, yang memiliki konotasi pandangan ke depan tertentu di PCRE. Contoh ini mengembalikan bagian input yang sesuai dengan kata kedua tersebut, tetapi berbeda dari contoh sebelumnya karena menggunakan pencocokan case-insensitive.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
---------------
 A1234
```

Contoh berikut menggunakan subexpression untuk menemukan string kedua yang cocok dengan pola `'this is a (\\w+)'` menggunakan pencocokan case-insensitive. Ia mengembalikan subexpression di dalam tanda kurung.

```
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
```

Fungsi REPEAT

Mengulangi string jumlah yang ditentukan kali. Jika parameter input numerik, REPEAT memperlakukannya sebagai string.

Sintaksis

```
REPEAT(string, integer)
```

Argumen

tali

Parameter input pertama adalah string yang akan diulang.

bilangan bulat

Parameter kedua adalah bilangan bulat yang menunjukkan berapa kali untuk mengulangi string.

Jenis pengembalian

Fungsi REPEAT mengembalikan string.

Contoh

Contoh berikut mengulangi nilai kolom CATID dalam tabel CATEGORY tiga kali:

```
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)
```

GANTI fungsi

Menggantikan semua kemunculan satu set karakter dalam string yang ada dengan karakter tertentu lainnya.

REPLACE mirip dengan FUNGSI TRANSLATE danFungsi REGEXP_REPLACE, kecuali bahwa TRANSLATE membuat beberapa substitusi karakter tunggal dan REGEXP_REPLACE memungkinkan Anda mencari string untuk pola ekspresi reguler, sementara REPLACE mengganti satu seluruh string dengan string lain.

Sintaksis

```
REPLACE(string1, old_chars, new_chars)
```

Argumen

tali

CHAR atau VARCHAR string yang akan dicari pencarian

old_chars

String CHAR atau VARCHAR untuk diganti.

new_chars

String CHAR atau VARCHAR baru menggantikan old_string.

Jenis pengembalian

VARCHAR

Jika old_chars atau new_chars adalah NULL, pengembaliannya adalah NULL.

Contoh

Contoh berikut mengkonversi string `Shows` ke `Theatre` dalam bidang 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)
```

Fungsi REVERSE

Fungsi REVERSE beroperasi pada string dan mengembalikan karakter dalam urutan terbalik. Misalnya, `reverse('abcde')` mengembalikan `edcba`. Fungsi ini bekerja pada tipe data numerik dan tanggal serta tipe data karakter; Namun, dalam banyak kasus memiliki nilai praktis untuk string karakter.

Sintaksis

```
REVERSE ( expression )
```

Pendapat

ekspresi

Ekspresi dengan karakter, tanggal, stempel waktu, atau tipe data numerik yang mewakili target pembalikan karakter. Semua ekspresi secara implisit dikonversi ke string karakter panjang variabel. Bagian belakang kosong dalam string karakter dengan lebar tetap diabaikan.

Jenis pengembalian

REVERSE mengembalikan VARCHAR.

Contoh

Pilih lima nama kota yang berbeda dan nama terbalik yang sesuai dari tabel 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)
```

Pilih lima penjualan IDs dan IDs pemeran terbalik yang sesuai sebagai string karakter:

```
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)
```

Fungsi RTRIM

Fungsi RTRIM memangkas satu set karakter tertentu dari akhir string. Menghapus string terpanjang yang hanya berisi karakter dalam daftar karakter trim. Pemangkasan selesai ketika karakter trim tidak muncul di string input.

Sintaksis

```
RTRIM( string, trim_chars )
```

Argumen

tali

Sebuah kolom string, ekspresi, atau string literal yang akan dipangkas.

trim_chars

Sebuah kolom string, ekspresi, atau string literal yang mewakili karakter yang akan dipangkas dari akhir string. Jika tidak ditentukan, spasi digunakan sebagai karakter trim.

Jenis pengembalian

String yang merupakan tipe data yang sama dengan argumen string.

Contoh

Contoh berikut memangkas bagian depan dan belakang kosong dari string: `' abc '`

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

Contoh berikut menghapus string trailing dari `'xyz'` string. `'xyzaxyzbxyzcxyz'` Kejadian tertinggal `'xyz'` dihapus, tetapi kejadian yang internal di dalam string tidak dihapus.

```
select 'xyzaxyzbxyzcxyz' as untrim,
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

Contoh berikut menghapus bagian trailing dari string `'setuphistorycassettes'` yang cocok dengan salah satu karakter dalam daftar trim_chars. `'tes'` Apa pun `te`,, atau `s` yang terjadi sebelum karakter lain yang tidak ada dalam daftar trim_chars di akhir string input dihapus.

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

Contoh berikut memangkas karakter 'Park' dari akhir VENUENAME di mana ada:

```
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
```

Perhatikan bahwa RTRIM menghapus salah satu karakterP,a,r, atau k ketika mereka muncul di akhir VENUENAME.

Fungsi SPLIT

Fungsi SPLIT memungkinkan Anda untuk mengekstrak substring dari string yang lebih besar dan bekerja dengan mereka sebagai array. Fungsi SPLIT berguna ketika Anda perlu memecah string menjadi komponen individual berdasarkan pembatas atau pola tertentu.

Sintaksis

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

Pendapat

str

Ekspresi string untuk dibagi.

regex

Sebuah string yang mewakili ekspresi reguler. String regex harus berupa ekspresi reguler Java.

batasi

Ekspresi integer yang mengontrol berapa kali regex diterapkan.

- limit > 0: Panjang array yang dihasilkan tidak akan lebih dari batas, dan entri terakhir array yang dihasilkan akan berisi semua input di luar regex terakhir yang cocok.
- limit <= 0: regex akan diterapkan sebanyak mungkin, dan array yang dihasilkan dapat berukuran berapa pun.

Jenis pengembalian

Fungsi SPLIT mengembalikan ARRAY<STRING>.

Jika`limit > 0`: Panjang array yang dihasilkan tidak akan lebih dari batas, dan entri terakhir array yang dihasilkan akan berisi semua input di luar regex terakhir yang cocok.

Jika`limit <= 0`: regex akan diterapkan sebanyak mungkin, dan array yang dihasilkan dapat berukuran berapa pun.

Contoh

Dalam contoh ini, fungsi SPLIT membagi string input di `'oneAtwoBthreeC'` mana pun ia bertemu karakter`'A'`,`'B'`, atau `'C'` (seperti yang ditentukan oleh pola `'[ABC]'` ekspresi reguler). Output yang dihasilkan adalah array dari empat elemen: `"one""two"`,`"three"`,, dan string kosong`""`.

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

Fungsi SPLIT_PART

Membagi string pada pembatas yang ditentukan dan mengembalikan bagian pada posisi yang ditentukan.

Sintaksis

```
SPLIT_PART(string, delimiter, position)
```

Argumen

tali

Sebuah kolom string, ekspresi, atau string literal yang akan dibagi. String dapat berupa CHAR atau VARCHAR.

pembatas

String pembatas menunjukkan bagian dari string input.

Jika pembatas adalah literal, lampirkan dalam tanda kutip tunggal.

posisi

Posisi bagian string untuk kembali (menghitung dari 1). Harus bilangan bulat lebih besar dari 0. Jika posisi lebih besar dari jumlah bagian string, SPLIT_PART mengembalikan string kosong. Jika pembatas tidak ditemukan dalam string, maka nilai yang dikembalikan berisi isi dari bagian yang ditentukan, yang mungkin seluruh string atau nilai kosong.

Jenis pengembalian

Sebuah string CHAR atau VARCHAR, sama dengan parameter string.

Contoh

Contoh berikut membagi string literal menjadi beberapa bagian menggunakan $ pembatas dan mengembalikan bagian kedua.

```
select split_part('abc$def$ghi','$',2)

split_part
----------
def
```

Contoh berikut membagi string literal menjadi beberapa bagian menggunakan $ pembatas. Ia mengembalikan string kosong karena bagian 4 tidak ditemukan.

```
select split_part('abc$def$ghi','$',4)
```

```
split_part
----------
```

Contoh berikut membagi string literal menjadi beberapa bagian menggunakan # pembatas. Ia mengembalikan seluruh string, yang merupakan bagian pertama, karena pembatas tidak ditemukan.

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

Contoh berikut membagi bidang timestamp LISTTIME menjadi komponen tahun, bulan, dan hari.

```
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
```

Contoh berikut memilih bidang timestamp LISTTIME dan membaginya pada `'-'` karakter untuk mendapatkan bulan (bagian kedua dari string LISTTIME), lalu menghitung jumlah entri untuk setiap bulan:

```
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
```

Fungsi SUBSTRING

Mengembalikan subset dari string berdasarkan posisi awal yang ditentukan.

Jika input adalah string karakter, posisi awal dan jumlah karakter yang diekstraksi didasarkan pada karakter, bukan byte, sehingga karakter multi-byte dihitung sebagai karakter tunggal. Jika input adalah ekspresi biner, posisi awal dan substring yang diekstraksi didasarkan pada byte. Anda tidak dapat menentukan panjang negatif, tetapi Anda dapat menentukan posisi awal negatif.

Sintaksis

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

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

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

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

Argumen

Charactertestring

String yang akan dicari. Tipe data non-karakter diperlakukan seperti string.

start_position

Posisi dalam string untuk memulai ekstraksi, mulai dari 1. Start_position didasarkan pada jumlah karakter, bukan byte, sehingga karakter multi-byte dihitung sebagai karakter tunggal. Angka ini bisa negatif.

numbecharacters

Jumlah karakter yang akan diekstrak (panjang substring). Numbecharacters didasarkan pada jumlah karakter, bukan byte, sehingga karakter multi-byte dihitung sebagai karakter tunggal. Angka ini tidak bisa negatif.

start_byte

Posisi dalam ekspresi biner untuk memulai ekstraksi, mulai dari 1. Angka ini bisa negatif.

numbebyte

Jumlah byte untuk mengekstrak, yaitu, panjang substring. Angka ini tidak bisa negatif.

Jenis pengembalian

VARCHAR

Catatan penggunaan untuk string karakter

Contoh berikut mengembalikan string empat karakter yang dimulai dengan karakter keenam.

```
select substring('caterpillar',6,4);
substring
-----------
pill
(1 row)
```

Jika numbecharacters start_position + melebihi panjang string, SUBSTRING mengembalikan substring mulai dari start_position hingga akhir string. Misalnya:

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

Jika negatif atau 0, fungsi SUBSTRING mengembalikan substring yang dimulai pada karakter pertama string dengan panjang `start_position` numbecharacters +-1. `start_position` Misalnya:

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

```
substring
-----------
cat
(1 row)
```

Jika `start_position` + `numbecharacters` -1 kurang dari atau sama dengan nol, SUBSTRING mengembalikan string kosong. Misalnya:

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

Contoh

Contoh berikut mengembalikan bulan dari string LISTTIME dalam tabel 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)
```

Contoh berikut sama seperti di atas, tetapi menggunakan opsi 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)
```

Anda tidak dapat menggunakan SUBSTRING untuk mengekstrak awalan string yang mungkin berisi karakter multi-byte karena Anda perlu menentukan panjang string multi-byte berdasarkan jumlah byte, bukan jumlah karakter. Untuk mengekstrak segmen awal string berdasarkan panjang dalam byte, Anda dapat CAST string sebagai VARCHAR (byte_length) untuk memotong string, di mana byte_length adalah panjang yang diperlukan. Contoh berikut mengekstrak 5 byte pertama dari string`'Fourscore and seven'`.

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
Fours
```

Contoh berikut mengembalikan nama depan `Ana` yang muncul setelah spasi terakhir dalam string input`Silva, Ana`.

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))

 reverse
-----------
 Ana
```

FUNGSI TRANSLATE

Untuk ekspresi tertentu, menggantikan semua kemunculan karakter tertentu dengan pengganti tertentu. Karakter yang ada dipetakan ke karakter pengganti berdasarkan posisinya dalam argumen characters_to_replace dan characters_to_substitusi. Jika lebih banyak karakter ditentukan dalam argumen characters_to_replace daripada dalam argumen characters_to_substitusi, karakter tambahan dari argumen characters_to_replace dihilangkan dalam nilai pengembalian.

TRANSLATE mirip dengan GANTI fungsi danFungsi REGEXP_REPLACE, kecuali bahwa REPLACE mengganti satu seluruh string dengan string lain dan REGEXP_REPLACE memungkinkan Anda mencari string untuk pola ekspresi reguler, sementara TRANSLATE membuat beberapa substitusi karakter tunggal.

Jika ada argumen nol, pengembaliannya adalah NULL.

Sintaksis

```
TRANSLATE ( expression, characters_to_replace, characters_to_substitute )
```

Argumen

ekspresi

Ekspresi yang akan diterjemahkan.

characters_to_replace

Sebuah string yang berisi karakter yang akan diganti.

characters_to_substitusi

Sebuah string yang berisi karakter untuk menggantikan.

Jenis pengembalian

VARCHAR

Contoh

Contoh berikut menggantikan beberapa karakter dalam string:

```
select translate('mint tea', 'inea', 'osin');

translate
-----------
most tin
```

Contoh berikut menggantikan tanda at (@) dengan periode untuk semua nilai dalam kolom:

```
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
```

Contoh berikut menggantikan spasi dengan garis bawah dan menghapus periode untuk semua nilai dalam kolom:

```
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
```

Fungsi TRIM

Memangkas string dengan menghapus bagian depan dan belakang kosong atau dengan menghapus karakter utama dan belakang yang cocok dengan string tertentu opsional.

Sintaksis

```
TRIM( [ BOTH ] [ trim_chars FROM ] string
```

Argumen

trim_chars

(Opsional) Karakter yang akan dipangkas dari string. Jika parameter ini dihilangkan, blanko dipangkas.

tali

Tali yang akan dipangkas.

Jenis pengembalian

Fungsi TRIM mengembalikan string VARCHAR atau CHAR. Jika Anda menggunakan fungsi TRIM dengan perintah SQL, secara AWS Clean Rooms implisit mengubah hasilnya ke VARCHAR. Jika Anda menggunakan fungsi TRIM dalam daftar SELECT untuk fungsi SQL, AWS Clean Rooms tidak secara implisit mengonversi hasilnya, dan Anda mungkin perlu melakukan konversi eksplisit untuk

menghindari kesalahan ketidakcocokan tipe data. Lihat Fungsi CAST fungsi untuk informasi tentang konversi eksplisit.

Contoh

Contoh berikut memangkas bagian depan dan belakang kosong dari string: `'  abc  '`

```
select '     abc    ' as untrim, trim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

Contoh berikut menghapus tanda kutip ganda yang mengelilingi string: `"dog"`

```
select trim('"' FROM '"dog"');

btrim
-------
dog
```

TRIM menghapus salah satu karakter di trim_chars ketika mereka muncul di awal string. Contoh berikut memangkas karakter 'C', 'D', dan 'G' ketika mereka muncul di awal VENUENAME, yang merupakan kolom 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
```

Fungsi UPPER

Mengkonversi string ke huruf besar. UPPER mendukung karakter multibyte UTF-8, hingga maksimal empat byte per karakter.

Sintaksis

```
UPPER(string)
```

Argumen

tali

Parameter input adalah string VARCHAR (atau tipe data lainnya, seperti CHAR, yang dapat secara implisit dikonversi ke VARCHAR).

Jenis pengembalian

Fungsi UPPER mengembalikan string karakter yang merupakan tipe data yang sama dengan string input.

Contoh

Contoh berikut mengkonversi bidang CATNAME ke huruf besar:

```
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)
```

Fungsi UUID

Fungsi UUID menghasilkan Universe Unique Identifier (UUID).

UUIDs adalah pengidentifikasi unik global yang biasanya digunakan untuk menyediakan pengidentifikasi unik untuk berbagai tujuan, seperti:

- Mengidentifikasi catatan basis data atau entitas data lainnya.
- Menghasilkan nama atau kunci unik untuk file, direktori, atau sumber daya lainnya.
- Melacak dan menghubungkan data di seluruh sistem terdistribusi.
- Menyediakan pengidentifikasi unik untuk paket jaringan, komponen perangkat lunak, atau aset digital lainnya.

Fungsi UUID menghasilkan nilai UUID yang unik dengan probabilitas yang sangat tinggi, bahkan di seluruh sistem terdistribusi dan dalam jangka waktu yang lama. UUIDs biasanya dihasilkan menggunakan kombinasi stempel waktu saat ini, alamat jaringan komputer, dan data acak atau pseudo-acak lainnya, memastikan bahwa setiap UUID yang dihasilkan sangat tidak mungkin bertentangan dengan UUID lainnya.

Dalam konteks kueri SQL, fungsi UUID dapat digunakan untuk menghasilkan pengidentifikasi unik untuk catatan baru yang dimasukkan ke dalam database, atau untuk menyediakan kunci unik untuk partisi data, pengindeksan, atau tujuan lain di mana pengidentifikasi unik diperlukan.

> **Note**
>
> Fungsi UUID bersifat non-deterministik.

Sintaksis

```
uuid()
```

Pendapat

Fungsi UUID tidak membutuhkan argumen.

Jenis pengembalian

UUID mengembalikan string pengenal unik universal (UUID). Nilai dikembalikan sebagai string 36 karakter UUID kanonik.

Contoh

Contoh berikut menghasilkan Universally Unique Identifier (UUID). Outputnya adalah string 36 karakter yang mewakili Universally Unique Identifier.

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

Fungsi terkait privasi

AWS Clean Rooms menyediakan fungsi untuk membantu Anda mematuhi kepatuhan terkait privasi untuk spesifikasi berikut.

- Global Privacy Platform (GPP) — Spesifikasi dari Interactive Advertising Bureau (IAB) yang menetapkan kerangka kerja standar global untuk privasi online dan penggunaan data. Untuk informasi selengkapnya tentang spesifikasi teknis GPP, lihat dokumentasi Platform Privasi Global di GitHub.
- Transparency and Consent Framework (TCF) — Komponen kunci dari GPP, diluncurkan pada tahun 2020, yang menyediakan kerangka teknis standar untuk membantu perusahaan mematuhi peraturan privasi seperti Peraturan Perlindungan Data Umum Uni Eropa (GDPR). TCF memungkinkan pelanggan untuk memberikan atau menahan persetujuan untuk pengumpulan dan pemrosesan data. Untuk informasi selengkapnya tentang spesifikasi teknis TCF, lihat dokumentasi TCF di. GitHub

Topik

- fungsi consent_gpp_v1_decode
- fungsi consent_tcf_v2_decode

fungsi consent_gpp_v1_decode

`consent_gpp_v1_decode`Fungsi ini digunakan untuk memecahkan kode data persetujuan Global Privacy Platform (GPP) v1. Dibutuhkan string persetujuan yang dikodekan sebagai masukan dan

mengembalikan data persetujuan yang diterjemahkan, yang mencakup informasi tentang preferensi privasi pengguna dan pilihan persetujuan. Fungsi ini berguna saat bekerja dengan data yang mencakup informasi persetujuan GPP v1, karena memungkinkan Anda untuk mengakses dan menganalisis data persetujuan dalam format terstruktur.

Sintaks

```
consent_gpp_v1_decode(gpp_string)
```

Argumen

gpp_string

String persetujuan GPP v1 yang dikodekan.

Pengembalian

Kamus yang dikembalikan mencakup pasangan kunci-nilai berikut:

- `version`: Versi spesifikasi GPP yang digunakan (saat ini 1).
- `cmpId`: ID Platform Manajemen Persetujuan (CMP) yang menyandikan string persetujuan.
- `cmpVersion`: Versi CMP yang mengkodekan string persetujuan.
- `consentScreen`: ID layar di UI CMP tempat pengguna memberikan persetujuan.
- `consentLanguage`: Kode bahasa dari informasi persetujuan.
- `vendorListVersion`: Versi daftar vendor yang digunakan.
- `publisherCountryCode`: Kode negara penerbit.
- `purposeConsent`: Daftar bilangan bulat yang mewakili tujuan yang telah disetujui pengguna.
- `purposeLegitimateInterest`: Daftar IDs tujuan yang kepentingan sah pengguna telah dikomunikasikan secara transparan.
- `specialFeatureOptIns`: Daftar bilangan bulat yang mewakili fitur khusus yang telah dipilih pengguna.
- `vendorConsent`: Daftar vendor IDs yang telah disetujui pengguna.
- `vendorLegitimateInterest`: Daftar vendor IDs yang kepentingan sah penggunanya telah dikomunikasikan secara transparan.

Contoh

Contoh berikut mengambil argumen tunggal, yang merupakan string persetujuan yang dikodekan. Ini mengembalikan kamus yang berisi data persetujuan yang diterjemahkan, termasuk informasi tentang preferensi privasi pengguna, pilihan persetujuan, dan metadata lainnya.

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

Struktur dasar dari data persetujuan yang dikembalikan mencakup informasi tentang versi string persetujuan, detail CMP (Platform Manajemen Persetujuan), persetujuan pengguna dan pilihan kepentingan yang sah untuk tujuan dan vendor yang berbeda, dan metadata lainnya.

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

fungsi consent_tcf_v2_decode

`consent_tcf_v2_decode`Fungsi ini digunakan untuk memecahkan kode data persetujuan Transparency and Consent Framework (TCF) v2. Dibutuhkan string persetujuan yang dikodekan sebagai masukan dan mengembalikan data persetujuan yang diterjemahkan, yang mencakup informasi tentang preferensi privasi pengguna dan pilihan persetujuan. Fungsi ini berguna saat bekerja dengan data yang mencakup informasi persetujuan TCF v2, karena memungkinkan Anda mengakses dan menganalisis data persetujuan dalam format terstruktur.

Sintaks

```
consent_tcf_v2_decode(tcf_string)
```

Argumen

tcf_string

String persetujuan TCF v2 yang dikodekan.

Pengembalian

`consent_tcf_v2_decode`Fungsi mengembalikan kamus yang berisi data persetujuan yang diterjemahkan dari string persetujuan Transparency and Consent Framework (TCF) v2.

Kamus yang dikembalikan mencakup pasangan kunci-nilai berikut:

Segmen inti

- `version`: Versi spesifikasi TCF yang digunakan (saat ini 2).
- `created`: Tanggal dan waktu ketika string persetujuan dibuat.
- `lastUpdated`: Tanggal dan waktu ketika string persetujuan terakhir diperbarui.
- `cmpId`: ID Platform Manajemen Persetujuan (CMP) yang menyandikan string persetujuan.
- `cmpVersion`: Versi CMP yang mengkodekan string persetujuan.
- `consentScreen`: ID layar di UI CMP tempat pengguna memberikan persetujuan.
- `consentLanguage`: Kode bahasa dari informasi persetujuan.
- `vendorListVersion`: Versi daftar vendor yang digunakan.
- `tcfPolicyVersion`: Versi kebijakan TCF yang menjadi dasar string persetujuan.
- `isServiceSpecific`Nilai Boolean yang menunjukkan apakah persetujuan khusus untuk layanan tertentu atau berlaku untuk semua layanan.
- `useNonStandardStacks`: Nilai Boolean yang menunjukkan apakah tumpukan non-standar digunakan.
- `specialFeatureOptIns`: Daftar bilangan bulat yang mewakili fitur khusus yang telah dipilih pengguna.
- `purposeConsent`: Daftar bilangan bulat yang mewakili tujuan yang telah disetujui pengguna.
- `purposesLITransparency`: Daftar bilangan bulat yang mewakili tujuan yang pengguna telah memberikan transparansi kepentingan yang sah.
- `purposeOneTreatment`: Nilai Boolean yang menunjukkan apakah pengguna telah meminta “tujuan satu perawatan” (yaitu, semua tujuan diperlakukan sama).

- `publisherCountryCode`: Kode negara penerbit.
- `vendorConsent`: Daftar vendor IDs yang telah disetujui pengguna.
- `vendorLegitimateInterest`: Daftar vendor IDs yang kepentingan sah penggunanya telah dikomunikasikan secara transparan.
- `pubRestrictionEntry`: Daftar pembatasan penerbit. Bidang ini berisi ID Tujuan, Jenis Pembatasan, dan Daftar Penjual IDs di bawah batasan Tujuan tersebut.

Segmen vendor yang diungkapkan

- `disclosedVendors`: Daftar bilangan bulat yang mewakili vendor yang telah diungkapkan kepada pengguna.

Segmen tujuan penerbit

- `pubPurposesConsent`: Daftar bilangan bulat yang mewakili tujuan khusus penerbit yang telah disetujui oleh pengguna.
- `pubPurposesLITransparency`: Daftar bilangan bulat yang mewakili tujuan khusus penerbit yang pengguna telah memberikan transparansi kepentingan yang sah.
- `customPurposesConsent`: Daftar bilangan bulat yang mewakili tujuan khusus yang telah disetujui oleh pengguna.
- `customPurposesLITransparency`: Daftar bilangan bulat yang mewakili tujuan khusus yang pengguna telah memberikan transparansi kepentingan yang sah.

Data persetujuan terperinci ini dapat digunakan untuk memahami dan menghormati preferensi privasi pengguna saat bekerja dengan data pribadi.

Contoh

Contoh berikut mengambil argumen tunggal, yang merupakan string persetujuan yang dikodekan. Ini mengembalikan kamus yang berisi data persetujuan yang diterjemahkan, termasuk informasi tentang preferensi privasi pengguna, pilihan persetujuan, dan metadata lainnya.

```
from aws_clean_rooms.functions import consent_tcf_v2_decode

consent_string = "CO1234567890abcdef"
consent_data = consent_tcf_v2_decode(consent_string)
```

```
print(consent_data)
```

Struktur dasar dari data persetujuan yang dikembalikan mencakup informasi tentang versi string persetujuan, detail CMP (Platform Manajemen Persetujuan), persetujuan pengguna dan pilihan kepentingan yang sah untuk tujuan dan vendor yang berbeda, dan metadata lainnya.

```
    /** 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],
};
```

Fungsi jendela

Dengan menggunakan fungsi jendela, Anda dapat membuat kueri bisnis analitik dengan lebih efisien. Fungsi jendela beroperasi pada partisi atau “jendela” dari kumpulan hasil, dan mengembalikan nilai untuk setiap baris di jendela itu. Sebaliknya, fungsi non-windowed melakukan perhitungan mereka sehubungan dengan setiap baris dalam set hasil. Tidak seperti fungsi grup yang menggabungkan baris hasil, fungsi jendela mempertahankan semua baris dalam ekspresi tabel.

Nilai yang dikembalikan dihitung dengan menggunakan nilai dari kumpulan baris di jendela itu. Untuk setiap baris dalam tabel, jendela mendefinisikan satu set baris yang digunakan untuk menghitung atribut tambahan. Sebuah jendela didefinisikan menggunakan spesifikasi jendela (klausa OVER), dan didasarkan pada tiga konsep utama:

- Partisi jendela, yang membentuk kelompok baris (klausa PARTISI)
- Pengurutan jendela, yang mendefinisikan urutan atau urutan baris dalam setiap partisi (klausa ORDER BY)
- Bingkai jendela, yang didefinisikan relatif terhadap setiap baris untuk lebih membatasi set baris (spesifikasi ROWS)

Fungsi jendela adalah rangkaian operasi terakhir yang dilakukan dalam kueri kecuali klausa ORDER BY akhir. Semua bergabung dan semua klausa WHERE, GROUP BY, dan HAVING selesai sebelum fungsi jendela diproses. Oleh karena itu, fungsi jendela hanya dapat muncul di daftar pilih atau klausa ORDER BY. Anda dapat menggunakan beberapa fungsi jendela dalam satu kueri dengan klausa bingkai yang berbeda. Anda juga dapat menggunakan fungsi jendela dalam ekspresi skalar lainnya, seperti CASE.

Ringkasan sintaks fungsi jendela

Fungsi jendela mengikuti sintaks standar, yaitu sebagai berikut.

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

Di sini, fungsi adalah salah satu fungsi yang dijelaskan dalam bagian ini.

Expr_list adalah sebagai berikut.

```
expression | column_name [, expr_list ]
```

Order_list adalah sebagai berikut.

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

Frame_clause adalah sebagai berikut.

```
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 }}
```

Pendapat

fungsi

Fungsi jendela. Untuk detailnya, lihat deskripsi fungsi individual.

DI ATAS

Klausul yang mendefinisikan spesifikasi jendela. Klausa OVER wajib untuk fungsi jendela, dan membedakan fungsi jendela dari fungsi SQL lainnya.

PARTISI OLEH expr_list

(Opsional) Klausa PARTITION BY membagi hasil yang ditetapkan menjadi partisi, seperti klausa GROUP BY. Jika klausa partisi hadir, fungsi dihitung untuk baris di setiap partisi. Jika tidak ada klausa partisi yang ditentukan, partisi tunggal berisi seluruh tabel, dan fungsi dihitung untuk tabel lengkap itu.

Fungsi peringkat DENSE_RANK, NTILE, RANK, dan ROW_NUMBER memerlukan perbandingan global dari semua baris dalam kumpulan hasil. Ketika klausa PARTITION BY digunakan, pengoptimal kueri dapat menjalankan setiap agregasi secara paralel dengan menyebarkan beban kerja di beberapa irisan sesuai dengan partisi. Jika klausa PARTITION BY tidak ada, langkah agregasi harus dijalankan secara serial pada satu irisan, yang dapat memiliki dampak negatif yang signifikan pada kinerja, terutama untuk cluster besar.

AWS Clean Roomstidak mendukung literal string dalam klausa PARTITION BY.

PESANAN BERDASARKAN order_list

(Opsional) Fungsi jendela diterapkan ke baris dalam setiap partisi yang diurutkan sesuai dengan spesifikasi pesanan di ORDER BY. Klausa ORDER BY ini berbeda dari dan sama sekali tidak terkait dengan klausa ORDER BY di frame_clause. Klausa ORDER BY dapat digunakan tanpa klausa PARTITION BY.

Untuk fungsi peringkat, klausa ORDER BY mengidentifikasi ukuran untuk nilai peringkat. Untuk fungsi agregasi, baris yang dipartisi harus diurutkan sebelum fungsi agregat dihitung untuk setiap frame. Untuk selengkapnya tentang jenis fungsi jendela, lihatFungsi jendela.

Pengidentifikasi kolom atau ekspresi yang mengevaluasi ke pengidentifikasi kolom diperlukan dalam daftar urutan. Baik konstanta maupun ekspresi konstan tidak dapat digunakan sebagai pengganti nama kolom.

Nilai NULLS diperlakukan sebagai grup mereka sendiri, diurutkan dan diberi peringkat sesuai dengan opsi NULLS FIRST atau NULLS LAST. Secara default, nilai NULL diurutkan dan diberi peringkat terakhir dalam urutan ASC, dan diurutkan dan diberi peringkat pertama dalam urutan DESC.

AWS Clean Roomstidak mendukung literal string dalam klausa ORDER BY.

Jika klausa ORDER BY dihilangkan, urutan baris adalah nondeterministik.

> **Note**
>
> Dalam sistem paralel apa pun sepertiAWS Clean Rooms, ketika klausa ORDER BY tidak menghasilkan urutan data yang unik dan total, urutan baris adalah nondeterministik. Artinya, jika ekspresi ORDER BY menghasilkan nilai duplikat (urutan sebagian), urutan pengembalian baris tersebut dapat bervariasi dari satu proses AWS Clean Rooms ke yang berikutnya. Pada gilirannya, fungsi jendela mungkin mengembalikan hasil yang tidak terduga atau tidak konsisten. Untuk informasi selengkapnya, lihat Urutan data yang unik untuk fungsi jendela.

column_name

Nama kolom yang akan dipartisi oleh atau diurutkan oleh.

ASC | DESC

Opsi yang mendefinisikan urutan pengurutan untuk ekspresi, sebagai berikut:

- ASC: naik (misalnya, rendah ke tinggi untuk nilai numerik dan 'A' ke 'Z' untuk string karakter). Jika tidak ada opsi yang ditentukan, data diurutkan dalam urutan menaik secara default.
- DESC: turun (tinggi ke rendah untuk nilai numerik; 'Z' ke 'A' untuk string).

NULLS PERTAMA | NULLS TERAKHIR

Opsi yang menentukan apakah NULLS harus diurutkan terlebih dahulu, sebelum nilai non-null, atau terakhir, setelah nilai non-null. Secara default, NULLS diurutkan dan diberi peringkat terakhir dalam urutan ASC, dan diurutkan dan diberi peringkat pertama dalam urutan DESC.

frame_clause

Untuk fungsi agregat, klausa bingkai lebih lanjut menyempurnakan kumpulan baris di jendela fungsi saat menggunakan ORDER BY. Ini memungkinkan Anda untuk memasukkan atau mengecualikan set baris dalam hasil yang diurutkan. Klausa bingkai terdiri dari kata kunci ROWS dan penentu terkait.

Klausa bingkai tidak berlaku untuk fungsi peringkat. Selain itu, klausa bingkai tidak diperlukan ketika tidak ada klausa ORDER BY yang digunakan dalam klausa OVER untuk fungsi agregat. Jika klausa ORDER BY digunakan untuk fungsi agregat, klausa bingkai eksplisit diperlukan.

Ketika tidak ada klausa ORDER BY yang ditentukan, bingkai tersirat tidak dibatasi, setara dengan BARIS ANTARA TIDAK TERBATAS SEBELUMNYA DAN TIDAK TERBATAS BERIKUT.

BARIS

Klausa ini mendefinisikan bingkai jendela dengan menentukan offset fisik dari baris saat ini.

Klausa ini menentukan baris di jendela atau partisi saat ini yang akan digabungkan dengan nilai dalam baris saat ini. Ini menggunakan argumen yang menentukan posisi baris, yang bisa sebelum atau sesudah baris saat ini. Titik referensi untuk semua bingkai jendela adalah baris saat ini. Setiap baris menjadi baris saat ini secara bergantian saat bingkai jendela meluncur ke depan di partisi.

Bingkai dapat berupa serangkaian baris sederhana hingga dan termasuk baris saat ini.

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

Atau bisa juga satu set baris antara dua batas.

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING menunjukkan bahwa jendela dimulai pada baris pertama partisi; offset PRECEDING menunjukkan bahwa jendela memulai sejumlah baris yang setara dengan nilai offset sebelum baris saat ini. UNBOUNDED PRECEDING adalah default.

ROW SAAT INI menunjukkan jendela dimulai atau berakhir pada baris saat ini.

BERIKUT TIDAK TERBATAS menunjukkan bahwa jendela berakhir pada baris terakhir partisi; offset BERIKUT menunjukkan bahwa jendela mengakhiri sejumlah baris yang setara dengan nilai offset setelah baris saat ini.

offset mengidentifikasi jumlah fisik baris sebelum atau sesudah baris saat ini. Dalam hal ini, offset harus berupa konstanta yang mengevaluasi nilai numerik positif. Misalnya, 5 BERIKUT mengakhiri bingkai lima baris setelah baris saat ini.

Dimana BETWEEN tidak ditentukan, frame secara implisit dibatasi oleh baris saat ini. Misalnya, `ROWS 5 PRECEDING` sama dengan`ROWS BETWEEN 5 PRECEDING AND CURRENT ROW`. Juga, `ROWS UNBOUNDED FOLLOWING` sama dengan`ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`.

> **Note**
>
> Anda tidak dapat menentukan bingkai di mana batas awal lebih besar dari batas akhir. Misalnya, Anda tidak dapat menentukan salah satu frame berikut.
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

Urutan data yang unik untuk fungsi jendela

Jika klausa ORDER BY untuk fungsi jendela tidak menghasilkan urutan data yang unik dan total, urutan baris adalah nondeterministik. Jika ekspresi ORDER BY menghasilkan nilai duplikat (urutan

sebagian), urutan pengembalian baris tersebut dapat bervariasi dalam beberapa kali proses. Dalam hal ini, fungsi jendela juga dapat mengembalikan hasil yang tidak terduga atau tidak konsisten.

Misalnya, kueri berikut mengembalikan hasil yang berbeda selama beberapa proses. Hasil yang berbeda ini terjadi karena `order by dateid` tidak menghasilkan urutan data yang unik untuk fungsi jendela 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
...
```

Dalam hal ini, menambahkan kolom ORDER BY kedua ke fungsi jendela dapat menyelesaikan masalah.

```
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
...
```

Fungsi yang didukung

AWS Clean RoomsSpark SQL mendukung dua jenis fungsi jendela: agregat dan peringkat.

Berikut ini adalah fungsi agregat yang didukung:

- Fungsi jendela CUME_DIST
- Fungsi jendela DENSE_RANK
- Fungsi jendela PERTAMA
- Fungsi jendela FIRST_VALUE
- Fungsi jendela LAG
- Fungsi jendela TERAKHIR
- Fungsi jendela LAST_VALUE
- Fungsi jendela LEAD

Berikut ini adalah fungsi peringkat yang didukung:

- Fungsi jendela DENSE_RANK
- Fungsi jendela PERCENT_RANK
- Fungsi jendela RANK
- Fungsi jendela ROW_NUMBER

Contoh tabel untuk contoh fungsi jendela

Anda dapat menemukan contoh fungsi jendela tertentu dengan setiap deskripsi fungsi. Beberapa contoh menggunakan tabel bernama WINSALES, yang berisi 11 baris, seperti yang ditunjukkan pada tabel berikut.

SALESID	DATEID	SELLERID	PEMBELI	QTY	QTY_DIKIRIM
30001	8/2/2003	3	B	10	10

SALESID	DATEID	SELLERID	PEMBELI	QTY	QTY_DIKIRIM
10001	12/24/2003	1	C	10	10
10005	12/24/2003	1	A	30	
40001	1/9/2004	4	A	40	
10006	1/18/2004	1	C	10	
20001	2/12/2004	2	B	20	20
40005	2/12/2004	4	A	10	10
20002	2/16/2004	2	C	20	20
30003	4/18/2004	3	B	15	
30004	4/18/2004	3	B	20	
30007	9/7/2004	3	C	30	

Fungsi jendela CUME_DIST

Menghitung distribusi kumulatif nilai dalam jendela atau partisi. Dengan asumsi urutan naik, distribusi kumulatif ditentukan dengan menggunakan rumus ini:

```
count of rows with values <= x / count of rows in the window or partition
```

di mana x sama dengan nilai di baris kolom saat ini yang ditentukan dalam klausa ORDER BY. Dataset berikut menggambarkan penggunaan rumus ini:

```
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
```

Rentang nilai pengembalian adalah> 0 hingga 1, inklusif.

Sintaksis

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Argumen

DI ATAS

Sebuah klausa yang menentukan partisi jendela. Klausa OVER tidak dapat berisi spesifikasi bingkai jendela.

PARTISI OLEH partition_expression

Tidak wajib. Ekspresi yang menetapkan rentang catatan untuk setiap grup dalam klausa OVER.

PESANAN BERDASARKAN order_list

Ekspresi untuk menghitung distribusi kumulatif. Ekspresi harus memiliki tipe data numerik atau secara implisit dapat dikonversi menjadi satu. Jika ORDER BY dihilangkan, nilai kembalinya adalah 1 untuk semua baris.

Jika ORDER BY tidak menghasilkan urutan unik, urutan baris adalah nondeterministik. Untuk informasi selengkapnya, lihat Urutan data yang unik untuk fungsi jendela.

Jenis pengembalian

FLOAT8

Contoh

Contoh berikut menghitung distribusi kumulatif kuantitas untuk setiap penjual:

```
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
```

Untuk deskripsi tabel WINSALES, lihatContoh tabel untuk contoh fungsi jendela.

Fungsi jendela DENSE_RANK

Fungsi jendela DENSE_RANK menentukan peringkat nilai dalam sekelompok nilai, berdasarkan ekspresi ORDER BY dalam klausa OVER. Jika klausa PARTITION BY opsional ada, peringkat diatur ulang untuk setiap kelompok baris. Baris dengan nilai yang sama untuk kriteria peringkat menerima peringkat yang sama. Fungsi DENSE_RANK berbeda dari RANK dalam satu hal: Jika dua atau lebih baris terikat, tidak ada celah dalam urutan nilai peringkat. Misalnya, jika dua baris diberi peringkat 1, peringkat berikutnya adalah 2.

Anda dapat memiliki fungsi peringkat dengan klausa PARTITION BY dan ORDER BY yang berbeda dalam kueri yang sama.

Sintaksis

```
DENSE_RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumen

( )

Fungsi ini tidak mengambil argumen, tetapi tanda kurung kosong diperlukan.

DI ATAS

Klausa jendela untuk fungsi DENSE_RANK.

PARTISI OLEH expr_list

Tidak wajib. Satu atau lebih ekspresi yang menentukan jendela.

PESANAN BERDASARKAN order_list

Tidak wajib. Ekspresi yang menjadi dasar nilai peringkat. Jika tidak ada PARTITION BY yang ditentukan, ORDER BY menggunakan seluruh tabel. Jika ORDER BY dihilangkan, nilai kembalinya adalah 1 untuk semua baris.

Jika ORDER BY tidak menghasilkan urutan unik, urutan baris adalah nondeterministik. Untuk informasi selengkapnya, lihat Urutan data yang unik untuk fungsi jendela.

Jenis pengembalian

INTEGER

Contoh

Contoh berikut memesan tabel berdasarkan kuantitas yang terjual (dalam urutan menurun), dan menetapkan peringkat padat dan peringkat reguler untuk setiap baris. Hasilnya diurutkan setelah hasil fungsi jendela diterapkan.

```
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)
```

Perhatikan perbedaan peringkat yang ditetapkan ke kumpulan baris yang sama saat fungsi DENSE_RANK dan RANK digunakan berdampingan dalam kueri yang sama. Untuk deskripsi tabel WINSALES, lihatContoh tabel untuk contoh fungsi jendela.

Contoh berikut mempartisi tabel oleh SELLERID dan memerintahkan setiap partisi dengan kuantitas (dalam urutan menurun) dan menetapkan peringkat padat untuk setiap baris. Hasilnya diurutkan setelah hasil fungsi jendela diterapkan.

```
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)
```

Untuk deskripsi tabel WINSALES, lihatContoh tabel untuk contoh fungsi jendela.

Fungsi jendela PERTAMA

Diberikan serangkaian baris yang diurutkan, FIRST mengembalikan nilai ekspresi yang ditentukan sehubungan dengan baris pertama di bingkai jendela.

Untuk informasi tentang memilih baris terakhir dalam bingkai, lihatFungsi jendela TERAKHIR.

Sintaksis

```
FIRST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
```

```
[ ORDER BY order_list frame_clause ]
)
```

Argumen

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi.

ABAIKAN NULLS

Ketika opsi ini digunakan dengan FIRST, fungsi mengembalikan nilai pertama dalam frame yang bukan NULL (atau NULL jika semua nilai NULL).

RESPECT NULLS

Menunjukkan bahwa AWS Clean Rooms harus menyertakan nilai nol dalam penentuan baris mana yang akan digunakan. RESPECT NULLS didukung secara default jika Anda tidak menentukan IGNORE NULLS.

DI ATAS

Memperkenalkan klausa jendela untuk fungsi tersebut.

PARTISI OLEH expr_list

Mendefinisikan jendela untuk fungsi dalam hal satu atau lebih ekspresi.

PESANAN BERDASARKAN order_list

Mengurutkan baris dalam setiap partisi. Jika tidak ada klausa PARTITION BY yang ditentukan, ORDER BY mengurutkan seluruh tabel. Jika Anda menentukan klausa ORDER BY, Anda juga harus menentukan frame_clause.

Hasil fungsi PERTAMA tergantung pada urutan data. Hasilnya nondeterministik dalam kasus-kasus berikut:

- Ketika tidak ada klausa ORDER BY ditentukan dan partisi berisi dua nilai yang berbeda untuk ekspresi
- Ketika ekspresi mengevaluasi nilai yang berbeda yang sesuai dengan nilai yang sama dalam daftar ORDER BY.

frame_clause

Jika klausa ORDER BY digunakan untuk fungsi agregat, klausa bingkai eksplisit diperlukan. Klausa bingkai menyempurnakan kumpulan baris di jendela fungsi, termasuk atau mengecualikan

kumpulan baris dalam hasil yang diurutkan. Klausa bingkai terdiri dari kata kunci ROWS dan penentu terkait. Lihat Ringkasan sintaks fungsi jendela.

Jenis pengembalian

Fungsi-fungsi ini mendukung ekspresi yang menggunakan tipe AWS Clean Rooms data primitif. Tipe pengembalian sama dengan tipe data ekspresi.

Contoh

Contoh berikut mengembalikan kapasitas tempat duduk untuk setiap tempat di meja VENUE, dengan hasil yang diurutkan berdasarkan kapasitas (tinggi ke rendah). Fungsi PERTAMA digunakan untuk memilih nama tempat yang sesuai dengan baris pertama dalam bingkai: dalam hal ini, baris dengan jumlah kursi tertinggi. Hasilnya dipartisi berdasarkan status, jadi ketika nilai VENUESTATE berubah, nilai pertama yang baru dipilih. Bingkai jendela tidak terbatas sehingga nilai pertama yang sama dipilih untuk setiap baris di setiap partisi.

Untuk California, `Qualcomm Stadium` memiliki jumlah kursi (`70561`) tertinggi, jadi nama ini adalah nilai pertama untuk semua baris di `CA` partisi.

```
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
...
```

Fungsi jendela FIRST_VALUE

Diberikan kumpulan baris yang diurutkan, FIRST_VALUE mengembalikan nilai ekspresi yang ditentukan sehubungan dengan baris pertama di bingkai jendela.

Untuk informasi tentang memilih baris terakhir dalam bingkai, lihatFungsi jendela LAST_VALUE.

Sintaksis

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumen

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi.

ABAIKAN NULLS

Ketika opsi ini digunakan dengan FIRST_VALUE, fungsi mengembalikan nilai pertama dalam frame yang tidak NULL (atau NULL jika semua nilai NULL).

RESPECT NULLS

Menunjukkan bahwa AWS Clean Rooms harus menyertakan nilai nol dalam penentuan baris mana yang akan digunakan. RESPECT NULLS didukung secara default jika Anda tidak menentukan IGNORE NULLS.

DI ATAS

Memperkenalkan klausa jendela untuk fungsi tersebut.

PARTISI OLEH expr_list

Mendefinisikan jendela untuk fungsi dalam hal satu atau lebih ekspresi.

PESANAN BERDASARKAN order_list

Mengurutkan baris dalam setiap partisi. Jika tidak ada klausa PARTITION BY yang ditentukan, ORDER BY mengurutkan seluruh tabel. Jika Anda menentukan klausa ORDER BY, Anda juga harus menentukan frame_clause.

Hasil fungsi FIRST_VALUE tergantung pada urutan data. Hasilnya nondeterministik dalam kasus-kasus berikut:

- Ketika tidak ada klausa ORDER BY ditentukan dan partisi berisi dua nilai yang berbeda untuk ekspresi
- Ketika ekspresi mengevaluasi nilai yang berbeda yang sesuai dengan nilai yang sama dalam daftar ORDER BY.

frame_clause

Jika klausa ORDER BY digunakan untuk fungsi agregat, klausa bingkai eksplisit diperlukan. Klausa bingkai menyempurnakan kumpulan baris di jendela fungsi, termasuk atau mengecualikan kumpulan baris dalam hasil yang diurutkan. Klausa bingkai terdiri dari kata kunci ROWS dan penentu terkait. Lihat Ringkasan sintaks fungsi jendela.

Jenis pengembalian

Fungsi-fungsi ini mendukung ekspresi yang menggunakan tipe AWS Clean Rooms data primitif. Tipe pengembalian sama dengan tipe data ekspresi.

Contoh

Contoh berikut mengembalikan kapasitas tempat duduk untuk setiap tempat di meja VENUE, dengan hasil yang diurutkan berdasarkan kapasitas (tinggi ke rendah). Fungsi FIRST_VALUE digunakan untuk memilih nama tempat yang sesuai dengan baris pertama dalam bingkai: dalam hal ini, baris dengan jumlah kursi tertinggi. Hasilnya dipartisi berdasarkan status, jadi ketika nilai VENUESTATE berubah, nilai pertama yang baru dipilih. Bingkai jendela tidak terbatas sehingga nilai pertama yang sama dipilih untuk setiap baris di setiap partisi.

Untuk California, `Qualcomm Stadium` memiliki jumlah kursi (`70561`) tertinggi, jadi nama ini adalah nilai pertama untuk semua baris di `CA` partisi.

```
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
...
```

Fungsi jendela LAG

Fungsi jendela LAG mengembalikan nilai untuk baris pada offset tertentu di atas (sebelum) baris saat ini di partisi.

Sintaksis

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Argumen

value_expr

Kolom target atau ekspresi tempat fungsi beroperasi.

mengimbangi

Parameter opsional yang menentukan jumlah baris sebelum baris saat ini untuk mengembalikan nilai untuk. Offset dapat berupa bilangan bulat konstan atau ekspresi yang mengevaluasi ke bilangan bulat. Jika Anda tidak menentukan offset, AWS Clean Rooms gunakan 1 sebagai nilai default. Offset `0` menunjukkan baris saat ini.

ABAIKAN NULLS

Spesifikasi opsional yang menunjukkan bahwa AWS Clean Rooms harus melewati nilai nol dalam penentuan baris mana yang akan digunakan. Nilai nol disertakan jika IGNORE NULLS tidak terdaftar.

> **Note**
>
> Anda dapat menggunakan ekspresi NVL atau COALESCE untuk mengganti nilai null dengan nilai lain.

RESPECT NULLS

Menunjukkan bahwa AWS Clean Rooms harus menyertakan nilai nol dalam penentuan baris mana yang akan digunakan. RESPECT NULLS didukung secara default jika Anda tidak menentukan IGNORE NULLS.

DI ATAS

Menentukan jendela partisi dan pemesanan. Klausa OVER tidak dapat berisi spesifikasi bingkai jendela.

PARTISI OLEH window_partition

Argumen opsional yang menetapkan rentang catatan untuk setiap grup dalam klausa OVER.

PESANAN DENGAN window_ordering

Mengurutkan baris dalam setiap partisi.

Fungsi jendela LAG mendukung ekspresi yang menggunakan salah satu tipe AWS Clean Rooms data. Jenis pengembalian sama dengan tipe value_expr.

Contoh

Contoh berikut menunjukkan jumlah tiket yang dijual kepada pembeli dengan ID pembeli 3 dan waktu pembeli 3 membeli tiket. Untuk membandingkan setiap penjualan dengan penjualan sebelumnya untuk pembeli 3, kueri mengembalikan jumlah sebelumnya yang dijual untuk setiap penjualan. Karena tidak ada pembelian sebelum 1/16/2008, nilai jual kuantitas pertama sebelumnya adalah nol:

```
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)
```

Fungsi jendela TERAKHIR

Diberikan kumpulan baris yang diurutkan, fungsi LAST mengembalikan nilai ekspresi sehubungan dengan baris terakhir dalam bingkai.

Untuk informasi tentang memilih baris pertama dalam bingkai, lihatFungsi jendela PERTAMA.

Sintaksis

```
LAST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumen

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi.

ABAIKAN NULLS

Fungsi mengembalikan nilai terakhir dalam frame yang tidak NULL (atau NULL jika semua nilai NULL).

RESPECT NULLS

Menunjukkan bahwa AWS Clean Rooms harus menyertakan nilai nol dalam penentuan baris mana yang akan digunakan. RESPECT NULLS didukung secara default jika Anda tidak menentukan IGNORE NULLS.

DI ATAS

Memperkenalkan klausa jendela untuk fungsi tersebut.

PARTISI OLEH expr_list

Mendefinisikan jendela untuk fungsi dalam hal satu atau lebih ekspresi.

PESANAN BERDASARKAN order_list

Mengurutkan baris dalam setiap partisi. Jika tidak ada klausa PARTITION BY yang ditentukan, ORDER BY mengurutkan seluruh tabel. Jika Anda menentukan klausa ORDER BY, Anda juga harus menentukan frame_clause.

Hasilnya tergantung pada urutan data. Hasilnya nondeterministik dalam kasus-kasus berikut:

- Ketika tidak ada klausa ORDER BY ditentukan dan partisi berisi dua nilai yang berbeda untuk ekspresi
- Ketika ekspresi mengevaluasi nilai yang berbeda yang sesuai dengan nilai yang sama dalam daftar ORDER BY.

frame_clause

Jika klausa ORDER BY digunakan untuk fungsi agregat, klausa bingkai eksplisit diperlukan. Klausa bingkai menyempurnakan kumpulan baris di jendela fungsi, termasuk atau mengecualikan kumpulan baris dalam hasil yang diurutkan. Klausa bingkai terdiri dari kata kunci ROWS dan penentu terkait. Lihat Ringkasan sintaks fungsi jendela.

Jenis pengembalian

Fungsi-fungsi ini mendukung ekspresi yang menggunakan tipe AWS Clean Rooms data primitif. Tipe pengembalian sama dengan tipe data ekspresi.

Contoh

Contoh berikut mengembalikan kapasitas tempat duduk untuk setiap tempat di meja VENUE, dengan hasil yang diurutkan berdasarkan kapasitas (tinggi ke rendah). Fungsi TERAKHIR digunakan untuk memilih nama tempat yang sesuai dengan baris terakhir dalam bingkai: dalam hal ini, baris dengan jumlah kursi paling sedikit. Hasilnya dipartisi berdasarkan status, jadi ketika nilai VENUESTATE berubah, nilai terakhir yang baru dipilih. Bingkai jendela tidak terbatas sehingga nilai terakhir yang sama dipilih untuk setiap baris di setiap partisi.

Untuk California, `Shoreline Amphitheatre` dikembalikan untuk setiap baris di partisi karena memiliki jumlah kursi terendah (`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
...
```

Fungsi jendela LAST_VALUE

Diberikan kumpulan baris yang diurutkan, fungsi LAST_VALUE mengembalikan nilai ekspresi sehubungan dengan baris terakhir dalam bingkai.

Untuk informasi tentang memilih baris pertama dalam bingkai, lihatFungsi jendela FIRST_VALUE.

Sintaksis

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumen

ekspresi

Kolom target atau ekspresi tempat fungsi beroperasi.

ABAIKAN NULLS

Fungsi mengembalikan nilai terakhir dalam frame yang tidak NULL (atau NULL jika semua nilai NULL).

RESPECT NULLS

Menunjukkan bahwa AWS Clean Rooms harus menyertakan nilai nol dalam penentuan baris mana yang akan digunakan. RESPECT NULLS didukung secara default jika Anda tidak menentukan IGNORE NULLS.

DI ATAS

Memperkenalkan klausa jendela untuk fungsi tersebut.

PARTISI OLEH expr_list

Mendefinisikan jendela untuk fungsi dalam hal satu atau lebih ekspresi.

PESANAN BERDASARKAN order_list

Mengurutkan baris dalam setiap partisi. Jika tidak ada klausa PARTITION BY yang ditentukan, ORDER BY mengurutkan seluruh tabel. Jika Anda menentukan klausa ORDER BY, Anda juga harus menentukan frame_clause.

Hasilnya tergantung pada urutan data. Hasilnya nondeterministik dalam kasus-kasus berikut:

- Ketika tidak ada klausa ORDER BY ditentukan dan partisi berisi dua nilai yang berbeda untuk ekspresi
- Ketika ekspresi mengevaluasi nilai yang berbeda yang sesuai dengan nilai yang sama dalam daftar ORDER BY.

frame_clause

Jika klausa ORDER BY digunakan untuk fungsi agregat, klausa bingkai eksplisit diperlukan. Klausa bingkai menyempurnakan kumpulan baris di jendela fungsi, termasuk atau mengecualikan kumpulan baris dalam hasil yang diurutkan. Klausa bingkai terdiri dari kata kunci ROWS dan penentu terkait. Lihat Ringkasan sintaks fungsi jendela.

Jenis pengembalian

Fungsi-fungsi ini mendukung ekspresi yang menggunakan tipe AWS Clean Rooms data primitif. Tipe pengembalian sama dengan tipe data ekspresi.

Contoh

Contoh berikut mengembalikan kapasitas tempat duduk untuk setiap tempat di meja VENUE, dengan hasil yang diurutkan berdasarkan kapasitas (tinggi ke rendah). Fungsi LAST_VALUE digunakan untuk memilih nama tempat yang sesuai dengan baris terakhir dalam bingkai: dalam hal ini, baris dengan jumlah kursi paling sedikit. Hasilnya dipartisi berdasarkan status, jadi ketika nilai VENUESTATE berubah, nilai terakhir yang baru dipilih. Bingkai jendela tidak terbatas sehingga nilai terakhir yang sama dipilih untuk setiap baris di setiap partisi.

Untuk California, `Shoreline Amphitheatre` dikembalikan untuk setiap baris di partisi karena memiliki jumlah kursi terendah (`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
...
```

Fungsi jendela LEAD

Fungsi jendela LEAD mengembalikan nilai untuk baris pada offset tertentu di bawah (setelah) baris saat ini di partisi.

Sintaksis

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Argumen

value_expr

Kolom target atau ekspresi tempat fungsi beroperasi.

mengimbangi

Parameter opsional yang menentukan jumlah baris di bawah baris saat ini untuk mengembalikan nilai untuk. Offset dapat berupa bilangan bulat konstan atau ekspresi yang mengevaluasi ke

bilangan bulat. Jika Anda tidak menentukan offset, AWS Clean Rooms gunakan 1 sebagai nilai default. Offset `0` menunjukkan baris saat ini.

ABAIKAN NULLS

Spesifikasi opsional yang menunjukkan bahwa AWS Clean Rooms harus melewati nilai nol dalam penentuan baris mana yang akan digunakan. Nilai nol disertakan jika IGNORE NULLS tidak terdaftar.

> **Note**
>
> Anda dapat menggunakan ekspresi NVL atau COALESCE untuk mengganti nilai null dengan nilai lain.

RESPECT NULLS

Menunjukkan bahwa AWS Clean Rooms harus menyertakan nilai nol dalam penentuan baris mana yang akan digunakan. RESPECT NULLS didukung secara default jika Anda tidak menentukan IGNORE NULLS.

DI ATAS

Menentukan jendela partisi dan pemesanan. Klausa OVER tidak dapat berisi spesifikasi bingkai jendela.

PARTISI OLEH window_partition

Argumen opsional yang menetapkan rentang catatan untuk setiap grup dalam klausa OVER.

PESANAN DENGAN window_ordering

Mengurutkan baris dalam setiap partisi.

Fungsi jendela LEAD mendukung ekspresi yang menggunakan salah satu tipe AWS Clean Rooms data. Jenis pengembalian sama dengan tipe value_expr.

Contoh

Contoh berikut memberikan komisi untuk acara-acara di tabel PENJUALAN di mana tiket dijual pada 1 Januari 2008 dan 2 Januari 2008 dan komisi yang dibayarkan untuk penjualan tiket untuk penjualan berikutnya.

```
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)
```

Fungsi jendela PERCENT_RANK

Menghitung peringkat persen dari baris yang diberikan. Peringkat persen ditentukan dengan menggunakan rumus ini:

`(x - 1) / (the number of rows in the window or partition - 1)`

dimana x adalah pangkat dari baris saat ini. Dataset berikut menggambarkan penggunaan rumus ini:

```
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
```

Rentang nilai pengembalian adalah 0 hingga 1, inklusif. Baris pertama dalam set apa pun memiliki PERCENT_RANK 0.

Sintaksis

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Argumen

( )

Fungsi ini tidak mengambil argumen, tetapi tanda kurung kosong diperlukan.

DI ATAS

Sebuah klausa yang menentukan partisi jendela. Klausa OVER tidak dapat berisi spesifikasi bingkai jendela.

PARTISI OLEH partition_expression

Tidak wajib. Ekspresi yang menetapkan rentang catatan untuk setiap grup dalam klausa OVER.

PESANAN BERDASARKAN order_list

Tidak wajib. Ekspresi untuk menghitung peringkat persen. Ekspresi harus memiliki tipe data numerik atau secara implisit dapat dikonversi menjadi satu. Jika ORDER BY dihilangkan, nilai kembalinya adalah 0 untuk semua baris.

Jika ORDER BY tidak menghasilkan urutan unik, urutan baris adalah nondeterministik. Untuk informasi selengkapnya, lihat Urutan data yang unik untuk fungsi jendela.

Jenis pengembalian

FLOAT8

Contoh

Contoh berikut menghitung peringkat persen dari jumlah penjualan untuk setiap penjual:

```
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
```

Untuk deskripsi tabel WINSALES, lihatContoh tabel untuk contoh fungsi jendela.

Fungsi jendela RANK

Fungsi jendela RANK menentukan peringkat nilai dalam sekelompok nilai, berdasarkan ekspresi ORDER BY dalam klausa OVER. Jika klausa PARTITION BY opsional ada, peringkat diatur ulang untuk setiap kelompok baris. Baris dengan nilai yang sama untuk kriteria peringkat menerima peringkat yang sama. AWS Clean Roomsmenambahkan jumlah baris terikat ke peringkat terikat untuk menghitung peringkat berikutnya dan dengan demikian peringkat mungkin bukan angka berurutan. Misalnya, jika dua baris diberi peringkat 1, peringkat berikutnya adalah 3.

RANK berbeda dari Fungsi jendela DENSE_RANK dalam satu hal: Untuk DENSE_RANK, jika dua atau lebih baris mengikat, tidak ada celah dalam urutan nilai peringkat. Misalnya, jika dua baris diberi peringkat 1, peringkat berikutnya adalah 2.

Anda dapat memiliki fungsi peringkat dengan klausa PARTITION BY dan ORDER BY yang berbeda dalam kueri yang sama.

Sintaksis

```
RANK () OVER
(
[ PARTITION BY expr_list ]
```

```
[ ORDER BY order_list ]
)
```

Argumen

( )

Fungsi ini tidak mengambil argumen, tetapi tanda kurung kosong diperlukan.

DI ATAS

Jendela klausa untuk fungsi RANK.

PARTISI OLEH expr_list

Tidak wajib. Satu atau lebih ekspresi yang menentukan jendela.

PESANAN BERDASARKAN order_list

Tidak wajib. Mendefinisikan kolom yang menjadi dasar nilai peringkat. Jika tidak ada PARTITION BY yang ditentukan, ORDER BY menggunakan seluruh tabel. Jika ORDER BY dihilangkan, nilai kembalinya adalah 1 untuk semua baris.

Jika ORDER BY tidak menghasilkan urutan unik, urutan baris adalah nondeterministik. Untuk informasi selengkapnya, lihat Urutan data yang unik untuk fungsi jendela.

Jenis pengembalian

INTEGER

Contoh

Contoh berikut memesan tabel berdasarkan kuantitas yang dijual (naik default), dan menetapkan peringkat untuk setiap baris. Nilai peringkat 1 adalah nilai peringkat tertinggi. Hasilnya diurutkan setelah hasil fungsi jendela diterapkan:

```
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)
```

Perhatikan bahwa klausa ORDER BY luar dalam contoh ini menyertakan kolom 2 dan 1 untuk memastikan bahwa AWS Clean Rooms mengembalikan hasil yang diurutkan secara konsisten setiap kali kueri ini dijalankan. Misalnya, baris dengan penjualan IDs 10001 dan 10006 memiliki nilai QTY dan RNK yang identik. Memesan hasil akhir yang ditetapkan oleh kolom 1 memastikan bahwa baris 10001 selalu jatuh sebelum 10006. Untuk deskripsi tabel WINSALES, lihatContoh tabel untuk contoh fungsi jendela.

Dalam contoh berikut, urutan dibalik untuk fungsi window (`order by qty desc`). Sekarang nilai peringkat tertinggi berlaku untuk nilai QTY terbesar.

```
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)
```

Untuk deskripsi tabel WINSALES, lihatContoh tabel untuk contoh fungsi jendela.

Contoh berikut mempartisi tabel oleh SELLERID dan mengurutkan setiap partisi dengan kuantitas (dalam urutan menurun) dan menetapkan peringkat untuk setiap baris. Hasilnya diurutkan setelah hasil fungsi jendela diterapkan.

```
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)
```

Fungsi jendela ROW_NUMBER

Menentukan nomor urut dari baris saat ini dalam sekelompok baris, dihitung dari 1, berdasarkan ekspresi ORDER BY dalam klausa OVER. Jika klausa PARTITION BY opsional ada, nomor urut diatur ulang untuk setiap kelompok baris. Baris dengan nilai yang sama untuk ekspresi ORDER BY menerima nomor baris yang berbeda secara nondeterministik.

Sintaksis

```
ROW_NUMBER () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumen

( )

Fungsi ini tidak mengambil argumen, tetapi tanda kurung kosong diperlukan.

DI ATAS

Klausa jendela untuk fungsi ROW_NUMBER.

PARTISI OLEH expr_list

Tidak wajib. Satu atau lebih ekspresi yang mendefinisikan fungsi ROW_NUMBER.

PESANAN BERDASARKAN order_list

Tidak wajib. Ekspresi yang mendefinisikan kolom yang menjadi dasar nomor baris. Jika tidak ada PARTITION BY yang ditentukan, ORDER BY menggunakan seluruh tabel.

Jika ORDER BY tidak menghasilkan urutan unik atau dihilangkan, urutan baris adalah nondeterministik. Untuk informasi selengkapnya, lihat Urutan data yang unik untuk fungsi jendela.

Jenis pengembalian

BIGINT

Contoh

Contoh berikut mempartisi tabel oleh SELLERID dan memerintahkan setiap partisi dengan QTY (dalam urutan menaik), kemudian menetapkan nomor baris untuk setiap baris. Hasilnya diurutkan setelah hasil fungsi jendela diterapkan.

```
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)
```

Untuk deskripsi tabel WINSALES, lihatContoh tabel untuk contoh fungsi jendela.

AWS Clean Rooms Kondisi Spark SQL

Kondisi adalah pernyataan dari satu atau lebih ekspresi dan operator logis yang mengevaluasi menjadi benar, salah, atau tidak diketahui. Kondisi juga kadang-kadang disebut sebagai predikat.

Sintaksis

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

> **Note**
>
> Semua perbandingan string dan kecocokan pola LIKE peka huruf besar/kecil. Misalnya, 'A' dan 'a' tidak cocok. Namun, Anda dapat melakukan kecocokan pola case-insensitive dengan menggunakan predikat ILIKE.

Kondisi SQL berikut didukung di AWS Clean Rooms Spark SQL.

Topik

- Operator perbandingan
- Kondisi logis
- Kondisi pencocokan pola

- ANTARA kondisi rentang
- Kondisi nol
- Kondisi ADA
- Dalam kondisi

Operator perbandingan

Kondisi perbandingan menyatakan hubungan logis antara dua nilai. Semua kondisi perbandingan adalah operator biner dengan tipe pengembalian Boolean.

AWS Clean Rooms Spark SQL mendukung operator perbandingan yang dijelaskan dalam tabel berikut.

Operator	Sintaksis	Deskripsi
!	`!expression`	`NOT`Operator logis. Digunakan untuk meniadakan ekspresi boolean, artinya mengembal ikan kebalikan dari nilai ekspresi. Itu! operator juga dapat dikombinasikan dengan operator logika lainnya, seperti AND dan OR, untuk membuat ekspresi boolean yang lebih kompleks.
<	`a < b`	Operator yang kurang dari perbandingan. Digunakan untuk membandingkan dua nilai dan menentukan apakah nilai di sebelah kiri kurang dari nilai di sebelah kanan.
>	`a > b`	Lebih besar dari operator perbandingan. Digunakan

Operator	Sintaksis	Deskripsi
		untuk membandingkan dua nilai dan menentukan apakah nilai di sebelah kiri lebih besar dari nilai di sebelah kanan.
<=	`a <= b`	Kurang dari atau sama dengan operator perbandin gan. Digunakan untuk membandingkan dua nilai dan mengembalikan `true` jika nilai di sebelah kiri kurang dari atau sama dengan nilai di sebelah kanan, dan `false` sebaliknya.
>=	`a >= b`	Lebih besar dari atau sama dengan operator perbandin gan. Digunakan untuk membandingkan dua nilai dan menentukan apakah nilai di sebelah kiri lebih besar dari atau sama dengan nilai di sebelah kanan.
=	`a = b`	Operator perbandingan kesetaraan, yang membandin gkan dua nilai dan mengembal ikan `true` jika mereka sama, dan `false` sebaliknya.
<> atau **! =**	`a <> b` atau `a != b`	Tidak sama dengan operator perbandingan, yang membandingkan dua nilai dan kembali `true` jika mereka tidak sama, dan `false` sebaliknya.

Operator	Sintaksis	Deskripsi
==	a == b	Operator perbandingan kesetaraan standar, yang membandingkan dua nilai dan mengembalikan `true` jika mereka sama, dan `false` sebaliknya. **Note** Operator == peka huruf besar/kecil saat membandingkan nilai string. Jika Anda perlu melakukan perbandingan case-insensitive, Anda dapat menggunakan fungsi seperti UPPER () atau LOWER () untuk mengonversi nilai ke kasus yang sama sebelum perbandingan.

Contoh

Berikut adalah beberapa contoh sederhana dari kondisi perbandingan:

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

Kueri berikut mengembalikan nilai id untuk semua tupai yang saat ini tidak mencari makan.

```
SELECT id FROM squirrels
WHERE !is_foraging
```

Kueri berikut mengembalikan tempat dengan lebih dari 10.000 kursi dari tabel 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)
```

Contoh ini memilih user (USERID) dari tabel USERS yang menyukai musik rock:

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
13
16
(5 rows)
```

Contoh ini memilih user (USERID) dari tabel USERS yang tidak diketahui apakah mereka menyukai musik 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
```

Contoh dengan kolom TIME

Berikut contoh tabel TIME_TEST memiliki kolom TIME_VAL (tipe TIME) dengan tiga nilai dimasukkan.

```
select time_val from time_test;

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

Contoh berikut mengekstrak jam dari setiap timetz_val.

```
select time_val from time_test where time_val < '3:00';
   time_val
---------------
 00:00:00.5550
 00:58:00
```

Contoh berikut membandingkan dua literal waktu.

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
```

```
----------
 t
```

Contoh dengan kolom TIMETZ

Contoh tabel berikut TIMETZ_TEST memiliki kolom TIMETZ_VAL (tipe TIMETZ) dengan tiga nilai dimasukkan.

```
select timetz_val from timetz_test;

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

Contoh berikut hanya memilih nilai TIMETZ kurang dari. `3:00:00 UTC` Perbandingan dilakukan setelah mengkonversi nilai ke UTC.

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

Contoh berikut membandingkan dua literal TIMETZ. Zona waktu diabaikan untuk perbandingan.

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

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

Kondisi logis

Kondisi logis menggabungkan hasil dari dua kondisi untuk menghasilkan satu hasil. Semua kondisi logis adalah operator biner dengan tipe pengembalian Boolean.

Sintaks

```
expression
```

```
{ AND | OR }
expression
NOT expression
```

Kondisi logis menggunakan logika Boolean tiga nilai di mana nilai nol mewakili hubungan yang tidak diketahui. Tabel berikut menjelaskan hasil untuk kondisi logis, di mana `E1` dan `E2` mewakili ekspresi:

E1	E2	E1 DAN E2	E1 ATAU E2	BUKAN E2
BETUL	BETUL	BETUL	BETUL	SALAH
BETUL	SALAH	SALAH	BETUL	BETUL
BETUL	TIDAK DIKETAHUI	TIDAK DIKETAHUI	BETUL	TIDAK DIKETAHUI
SALAH	BETUL	SALAH	BETUL	
SALAH	SALAH	SALAH	SALAH	
SALAH	TIDAK DIKETAHUI	SALAH	TIDAK DIKETAHUI	
TIDAK DIKETAHUI	BETUL	TIDAK DIKETAHUI	BETUL	
TIDAK DIKETAHUI	SALAH	SALAH	TIDAK DIKETAHUI	
TIDAK DIKETAHUI	TIDAK DIKETAHUI	TIDAK DIKETAHUI	TIDAK DIKETAHUI	

Operator NOT dievaluasi sebelum AND, dan operator AND dievaluasi sebelum operator OR. Tanda kurung apa pun yang digunakan dapat mengesampingkan urutan evaluasi default ini.

Contoh

Contoh berikut mengembalikan USERID dan USERNAME dari tabel USERS tempat pengguna menyukai Las Vegas dan olahraga:

```
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)
```

Contoh berikutnya mengembalikan USERID dan USERNAME dari tabel USERS di mana pengguna menyukai Las Vegas, atau olahraga, atau keduanya. Kueri ini mengembalikan semua output dari contoh sebelumnya ditambah pengguna yang hanya menyukai Las Vegas atau olahraga.

```
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)
```

Kueri berikut menggunakan tanda kurung di sekitar OR kondisi untuk menemukan tempat di New York atau California tempat Macbeth dilakukan:

```
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
...
```

Menghapus tanda kurung dalam contoh ini mengubah logika dan hasil kueri.

Contoh berikut menggunakan NOT operator:

```
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
...
```

Contoh berikut menggunakan NOT kondisi yang diikuti oleh suatu AND kondisi:

```
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)
```

Kondisi pencocokan pola

Operator pencocokan pola mencari string untuk pola yang ditentukan dalam ekspresi kondisional dan mengembalikan true atau false tergantung pada apakah ia menemukan kecocokan. AWS Clean Rooms Spark SQL menggunakan metode berikut untuk pencocokan pola:

- Seperti ekspresi

  Operator LIKE membandingkan ekspresi string, seperti nama kolom, dengan pola yang menggunakan karakter wildcard % (persen) dan _ (garis bawah). Pencocokan pola LIKE selalu mencakup seluruh string. LIKE melakukan kecocokan peka huruf besar/kecil.

Topik

- SUKA
- RLIKE

SUKA

Operator LIKE membandingkan ekspresi string, seperti nama kolom, dengan pola yang menggunakan karakter wildcard% (percent) dan _ (underscore). Pencocokan pola LIKE selalu mencakup seluruh string. Untuk mencocokkan urutan di mana saja dalam string, pola harus dimulai dan diakhiri dengan tanda persen.

LIKE adalah case-sensitive.

Sintaksis

```
expression [ NOT ] LIKE | pattern [ ESCAPE 'escape_char' ]
```

Argumen

ekspresi

Ekspresi karakter UTF-8 yang valid, seperti nama kolom.

SUKA

LIKE melakukan kecocokan pola peka huruf besar/kecil. Untuk melakukan kecocokan pola case-insensitive untuk karakter multibyte, gunakan fungsi LOWER pada ekspresi dan pola dengan kondisi LIKE.

Berbeda dengan predikat perbandingan, seperti = dan <>, predikat LIKE tidak secara implisit mengabaikan spasi tambahan. Untuk mengabaikan spasi tambahan, gunakan RTRIM atau secara eksplisit melemparkan kolom CHAR ke VARCHAR.

~~Operator setara dengan LIKE. !~~Operator juga setara dengan NOT LIKE.

pola

Ekspresi karakter UTF-8 yang valid dengan pola yang akan dicocokkan.

escape_char

Ekspresi karakter yang akan lolos dari karakter metakarakter dalam pola. Defaultnya adalah dua garis miring terbalik ('\\').

Jika pola tidak mengandung metakarakter, maka pola hanya mewakili string itu sendiri; dalam hal ini LIKE bertindak sama dengan operator sama dengan.

Salah satu ekspresi karakter dapat berupa tipe data CHAR atau VARCHAR. Jika mereka berbeda, AWS Clean Rooms mengkonversi pola ke tipe data ekspresi.

LIKE mendukung metakarakter pencocokan pola berikut:

Operator	Deskripsi
%	Cocokkan dengan urutan nol atau lebih karakter.

Operator	Deskripsi
_	Cocokkan karakter tunggal apa pun.

Contoh

Tabel berikut menunjukkan contoh pencocokan pola menggunakan LIKE:

Ekspresi	Pengembalian
`'abc' LIKE 'abc'`	True
`'abc' LIKE 'a%'`	Benar
`'abc' LIKE '_B_'`	Salah
`'abc' LIKE 'c%'`	False

Contoh berikut menemukan semua kota yang namanya dimulai dengan “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
...
```

Contoh berikut menemukan pengguna yang nama belakangnya berisi “sepuluh”:

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
```

```
-------------
Christensen
Wooten
...
```

Contoh berikut menemukan kota yang karakter ketiga dan keempatnya adalah “ea” . :

```
select distinct city from users where city like '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

Contoh berikut menggunakan string escape default (\\) untuk mencari string yang menyertakan “start_” (teks `start` diikuti oleh garis bawah): _

```
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)
```

Contoh berikut menentukan '^' sebagai karakter escape, kemudian menggunakan karakter escape untuk mencari string yang menyertakan “start_” (teks `start` diikuti dengan garis bawah): _

```
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

Operator RLIKE memungkinkan Anda untuk memeriksa apakah string cocok dengan pola ekspresi reguler tertentu.

Mengembalikan `true` jika str cocok`regexp`, atau `false` sebaliknya.

Sintaksis

```
rlike(str, regexp)
```

Pendapat

str

Ekspresi string

regexp

Ekspresi string. String regex harus berupa ekspresi reguler Java.

Literal string (termasuk pola regex) tidak dapat diloloskan di parser SQL kami. Misalnya, untuk mencocokkan “\ abc”, ekspresi reguler untuk regexp dapat berupa “^\ abc$”.

Contoh

Contoh berikut menetapkan nilai parameter `spark.sql.parser.escapedStringLiterals` konfigurasi untuk`true`. Parameter ini khusus untuk mesin Spark SQL. `spark.sql.parser.escapedStringLiterals`Parameter di Spark SQL mengontrol bagaimana parser SQL menangani literal string yang lolos. Ketika diatur k`etrue`, parser akan menafsirkan

karakter garis miring terbalik (\) dalam literal string sebagai karakter escape, memungkinkan Anda untuk memasukkan karakter khusus seperti baris baru, tab, dan tanda kutip dalam nilai string Anda.

```
SET spark.sql.parser.escapedStringLiterals=true;
spark.sql.parser.escapedStringLiterals  true
```

Misalnya, dengan`spark.sql.parser.escapedStringLiterals=true`, Anda dapat menggunakan string literal berikut dalam kueri SQL Anda:

```
SELECT 'Hello, world!\n'
```

Karakter baris baru `\n` akan ditafsirkan sebagai karakter baris baru literal dalam output.

Contoh berikut melakukan pencocokan pola ekspresi reguler. Argumen pertama diteruskan ke operator RLIKE. Ini adalah string yang mewakili jalur file, di mana nama pengguna sebenarnya diganti dengan pola '****'. Argumen kedua adalah pola ekspresi reguler yang digunakan untuk pencocokan. Output (`true`) menunjukkan bahwa string pertama (`'%SystemDrive%\Users\****'`) cocok dengan pola ekspresi reguler (`'%SystemDrive%\\Users.*'`).

```
SELECT rlike('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

ANTARA kondisi rentang

Sebuah `BETWEEN` kondisi menguji ekspresi untuk dimasukkan dalam berbagai nilai, menggunakan kata kunci `BETWEEN` dan`AND`.

Sintaks

```
expression [ NOT ] BETWEEN expression AND expression
```

Ekspresi dapat berupa tipe data numerik, karakter, atau datetime, tetapi harus kompatibel. Kisarannya inklusif.

Contoh

Contoh pertama menghitung berapa banyak transaksi terdaftar penjualan baik 2, 3, atau 4 tiket:

```
select count(*) from sales
```

```
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

Kondisi rentang mencakup nilai awal dan akhir.

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

Ekspresi pertama dalam kondisi rentang harus nilai yang lebih rendah dan ekspresi kedua nilai yang lebih besar. Contoh berikut akan selalu mengembalikan nol baris karena nilai-nilai ekspresi:

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

Namun, menerapkan pengubah NOT akan membalikkan logika dan menghasilkan hitungan semua baris:

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
172456
(1 row)
```

Kueri berikut mengembalikan daftar tempat dengan 20000 hingga 50000 kursi:

```
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)
```

Contoh berikut menunjukkan menggunakan BETWEEN untuk nilai tanggal:

```
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
```

Perhatikan bahwa meskipun rentang BETWEEN inklusif, tanggal default memiliki nilai waktu 00:00:00. Satu-satunya baris 3 Januari yang valid untuk kueri sampel adalah baris dengan waktu penjualan. `1/3/2008 00:00:00`

Kondisi nol

Bagian NULL tes kondisi untuk nol, ketika nilai hilang atau tidak diketahui.

Sintaks

```
expression IS [ NOT ] NULL
```

Argumen

ekspresi

Ekspresi apa pun seperti kolom.

IS NULL

Benar ketika nilai ekspresi adalah null dan false ketika memiliki nilai.

IS NOT NULL

Adalah false ketika nilai ekspresi adalah null dan true ketika memiliki nilai.

Contoh

Contoh ini menunjukkan berapa kali tabel PENJUALAN berisi null di bidang QTYSOLD:

```
select count(*) from sales
where qtysold is null;
count
-------
0
(1 row)
```

Kondisi ADA

EXISTS kondisi menguji keberadaan baris dalam subquery, dan mengembalikan true jika subquery mengembalikan setidaknya satu baris. Jika NOT ditentukan, kondisi mengembalikan true jika subquery mengembalikan tidak ada baris.

Sintaks

```
[ NOT ] EXISTS (table_subquery)
```

Argumen

ADA

Benar ketika table_subquery mengembalikan setidaknya satu baris.

TIDAK ADA

Benar ketika table_subquery tidak mengembalikan baris.

table_subquery

Subquery yang mengevaluasi tabel dengan satu atau lebih kolom dan satu atau lebih baris.

Contoh

Contoh ini mengembalikan semua pengidentifikasi tanggal, masing-masing satu kali, untuk setiap tanggal yang memiliki penjualan dalam bentuk apa pun:

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
1827
1828
1829
...
```

Dalam kondisi

INKondisi menguji nilai untuk keanggotaan dalam satu set nilai atau dalam subquery.

Sintaksis

```
expression [ NOT ] IN (expr_list | table_subquery)
```

Pendapat

ekspresi

Ekspresi numerik, karakter, atau datetime yang dievaluasi terhadap expr_list atau table_subquery dan harus kompatibel dengan tipe data daftar atau subquery tersebut.

expr_list

Satu atau lebih ekspresi yang dibatasi koma, atau satu atau lebih kumpulan ekspresi yang dibatasi koma yang dibatasi oleh tanda kurung.

table_subquery

Subquery yang mengevaluasi tabel dengan satu atau lebih baris, tetapi terbatas hanya satu kolom dalam daftar pilihannya.

DI | TIDAK DI

IN mengembalikan true jika ekspresi adalah anggota dari daftar ekspresi atau query. NOT IN mengembalikan true jika ekspresi bukan anggota. IN dan NOT IN mengembalikan NULL dan tidak ada baris yang dikembalikan dalam kasus berikut: Jika ekspresi menghasilkan null; atau jika tidak ada nilai expr_list atau table_subquery yang cocok dan setidaknya satu dari baris perbandingan ini menghasilkan null.

Contoh

Kondisi berikut benar hanya untuk nilai-nilai yang tercantum:

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

Optimalisasi untuk Daftar IN Besar

Untuk mengoptimalkan kinerja kueri, daftar IN yang mencakup lebih dari 10 nilai dievaluasi secara internal sebagai array skalar. Daftar IN dengan nilai kurang dari 10 dievaluasi sebagai serangkaian predikat OR. Optimalisasi ini didukung untuk tipe data SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, dan TIMESTAMPTZ.

Lihatlah output EXPLOW untuk kueri untuk melihat efek dari pengoptimalan ini. Contoh:

```
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)
```

Menanyakan data bersarang

AWS Clean Rooms menawarkan akses yang kompatibel dengan SQL ke data relasional dan bersarang.

AWS Clean Rooms menggunakan notasi putus-putus dan subskrip array untuk navigasi jalur saat mengakses data bersarang. Hal ini juga memungkinkan FROM item klausa untuk mengulangi array dan digunakan untuk operasi unnest. Topik berikut memberikan deskripsi pola kueri berbeda yang menggabungkan penggunaan tipe array/struct/map data dengan navigasi jalur dan array, unnesting, dan gabungan.

Topik

Navigasi

AWS Clean Rooms memungkinkan navigasi ke array dan struktur menggunakan `[...]` braket dan notasi titik masing-masing. Selanjutnya, Anda dapat mencampur navigasi ke dalam struktur menggunakan notasi titik dan array menggunakan notasi braket.

Example

Misalnya, contoh query berikut mengasumsikan bahwa kolom data `c_orders` array adalah array dengan struktur dan atribut bernamao_`orderkey`.

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

Anda dapat menggunakan notasi titik dan braket di semua jenis kueri, seperti pemfilteran, gabungan, dan agregasi. Anda dapat menggunakan notasi ini dalam kueri di mana biasanya ada referensi kolom.

Example

Contoh berikut menggunakan pernyataan SELECT yang memfilter hasil.

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0].o_orderkey IS NOT NULL;
```

Example

Contoh berikut menggunakan braket dan navigasi titik di kedua klausa GROUP BY dan 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;
```

Kueri yang tidak bersarang

Untuk kueri unnest, AWS Clean Rooms aktifkan iterasi melalui array. Hal ini dilakukan dengan menavigasi array menggunakan klausa FROM dari query.

Example

Menggunakan contoh sebelumnya, contoh berikut iterasi atas nilai atribut untuk`c_orders`.

```
SELECT o FROM customer_orders_lineitem c, c.c_orders o;
```

Sintaks unnesting adalah perpanjangan dari klausa FROM. Dalam SQL standar, klausa FROM x (AS) y berarti bahwa y iterasi atas setiap tupel dalam hubungannya. x Dalam hal ini, x mengacu pada relasi dan y mengacu pada alias untuk relasi`x`. Demikian pula, sintaks unnesting menggunakan item klausa FROM x (AS) y berarti bahwa y iterasi atas setiap nilai dalam ekspresi array. x Dalam hal ini, x adalah ekspresi array dan y merupakan alias untuk`x`.

Operan kiri juga dapat menggunakan notasi titik dan braket untuk navigasi reguler.

Example

Pada contoh sebelumnya:

- `customer_orders_lineitem c`adalah iterasi di atas tabel `customer_order_lineitem` dasar

- `c.c_orders` oadalah iterasi atas `c.c_orders array`

Untuk mengulangi `o_lineitems` atribut, yang merupakan array dalam array, Anda menambahkan beberapa klausa.

```
SELECT o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

AWS Clean Rooms juga mendukung indeks array saat mengulangi array menggunakan AT kata kunci. Klausa `x AS y AT z` iterasi atas array `x` dan menghasilkan bidang`z`, yang merupakan indeks array.

Example

Contoh berikut menunjukkan bagaimana indeks array bekerja.

```
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

Contoh berikut iterasi atas array skalar.

```
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

Contoh berikut iterasi atas array dari beberapa level. Contoh menggunakan beberapa klausa unnest untuk beralih ke array terdalam. Sebuah `f.multi_level_array` AS array iterasi. `multi_level_array` Array AS elemen adalah iterasi atas array di dalamnya. `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)
```

Semantik longgar

Secara default, operasi navigasi pada nilai data bersarang mengembalikan null alih-alih mengembalikan kesalahan saat navigasi tidak valid. Navigasi objek tidak valid jika nilai data bersarang bukan objek atau jika nilai data bersarang adalah objek tetapi tidak berisi nama atribut yang digunakan dalam kueri.

Example

Misalnya, kueri berikut mengakses nama atribut yang tidak valid di kolom data bersarang: `c_orders`

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

Navigasi array mengembalikan null jika nilai data bersarang bukan array atau indeks array di luar batas.

Example

Query berikut mengembalikan null `c_orders[1][1]` karena di luar batas.

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

Jenis introspeksi

Kolom tipe data bersarang mendukung fungsi inspeksi yang mengembalikan tipe dan informasi tipe lainnya tentang nilai. AWS Clean Rooms mendukung fungsi boolean berikut untuk kolom data bersarang:

- DESIMAL_PRESISI
- SKALA DESIMAL
- IS_ARRAY
- IS_BIGINT
- IS_CHAR
- ADALAH_DESIMAL
- IS_MENGAPUNG
- IS_INTEGER
- IS_OBJEK
- IS_SKALAR
- IS_SMALLINT
- IS_VARCHAR
- JSON_TYPEOF

Semua fungsi ini mengembalikan false jika nilai input adalah null. IS_SCALAR, IS_OBJECT, dan IS_ARRAY saling eksklusif dan mencakup semua nilai yang mungkin kecuali untuk null. Untuk menyimpulkan tipe yang sesuai dengan data, AWS Clean Rooms gunakan fungsi JSON_TYPEOF yang mengembalikan tipe (tingkat atas) nilai data bersarang seperti yang ditunjukkan pada contoh berikut:

```
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
```

Riwayat dokumen untuk AWS Clean Rooms Referensi SQL

Tabel berikut menjelaskan rilis dokumentasi untuk Referensi AWS Clean Rooms SQL.

Untuk notifikasi tentang pembaruan-pembaruan dokumentasi ini, Anda dapat berlangganan ke sebuah umpan RSS. Untuk berlangganan pembaruan RSS, Anda harus mengaktifkan plug-in RSS untuk browser yang Anda gunakan.

Perubahan	Deskripsi	Tanggal
Spark SQL mendukung Petunjuk	AWS Clean Rooms Spark SQL mendukung petunjuk kueri untuk mengoptimalkan kinerja kueri dan mengurangi biaya komputasi.	Januari 20, 2026
Spark SQL mendukung TABEL CACHE	AWS Clean Rooms Spark SQL mendukung perintah CACHE TABLE, yang memungkinkan pelanggan untuk menyimpan tabel yang ada atau membuat dan menyimpan tabel baru dari hasil kueri untuk meningkatkan kinerja kueri.	Oktober 22, 2025
Spark SQL mendukung fungsi Jendela PERTAMA dan TERAKHIR	AWS Clean Rooms Spark SQL mendukung fungsi Window berikut: PERTAMA dan TERAKHIR.	12 Juni 2025
Pembaruan dokumentasi fungsi Spark SQL	Pembaruan khusus dokumentasi untuk secara akurat mencerminkan fungsi Spark SQL yang didukung. Dokumentasi yang dihapus untuk 25 fungsi yang	Mei 20, 2025

	tidak didukung, termasuk <=> operator, MILAR TO, LISTAGG, dan ARRAY_INS ERT. Nama fungsi yang dikoreksi dari DATEADD ke DATE_ADD, DATEDIFF ke DATE_DIFF, ISNULL ke IS_NULL, dan ISNOTNULL ke IS_NOT_NULL. Memperbaiki kesalahan ketik dalam contoh DATE_PART.	
AWS Clean Rooms Spark SQL	Pelanggan sekarang dapat menjalankan kueri menggunak an beberapa kondisi SQL, fungsi, perintah, dan konvensi yang didukung dengan mesin analisis Spark SQL.	Oktober 29, 2024
Perintah SQL dan fungsi SQL - pembaruan	Contoh telah ditambahk an untuk klausa JOIN, KECUALI set operator, ekspresi bersyarat CASE, dan fungsi berikut: ANY_VALUE , NVL dan COALESCE, NULLIF, CAST, CONVERT, CONVERT_TIMEZONE, EXTRACT, MOD, SIGN, CONCAT, FIRST_VALUE, dan LAST_VALUE.	Februari 28, 2024

Fungsi SQL - pembaruan	AWS Clean Rooms sekarang mendukung fungsi SQL berikut: Array, SUPER, dan VARBYTE. Fungsi matematik a berikut sekarang didukung: ACOS, ASIN, ATAN, COT ATAN2, DEXP, PI, POW, RADIANS, dan SIN. Fungsi JSON berikut sekarang didukung: CAN_JSON_ PARSE, JSON_PARSE, dan JSON_SERIALIZE.	Oktober 6, 2023
Dukungan tipe data bersarang	AWS Clean Rooms sekarang mendukung tipe data bersarang.	Agustus 30, 2023
Aturan penamaan SQL - perbarui	Perubahan hanya dokumenta si untuk memperjelas nama kolom yang dicadangkan.	16 Agustus 2023
Ketersediaan umum	Referensi AWS Clean Rooms SQL sekarang tersedia secara umum.	31 Juli 2023

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.