Documentazione di riferimento a SQL

AWS Clean Rooms

AWS Clean Rooms: Documentazione di riferimento a SQL

Table of Contents

Panoramica di SQL in AWS Clean Rooms

Benvenuto in AWS Clean RoomsSQL Reference.

AWS Clean Roomsè basato sullo standard di settore Structured Query Language (SQL), un linguaggio di interrogazione costituito da comandi e funzioni utilizzati per lavorare con database e oggetti di database. SQL impone inoltre regole relative all'uso di tipi di dati, espressioni e valori letterali.

Negli argomenti seguenti vengono fornite informazioni generali sulle convenzioni e le regole di denominazione utilizzate in questo riferimento SQL.

Argomenti

- Convenzioni del riferimento SQL
- Regole di denominazione SQL
- Supporto dei tipi di dati tramite il motore SQL

Le sezioni seguenti forniscono informazioni sui valori letterali, i tipi di dati, i comandi SQL, i tipi di funzioni SQL e le condizioni SQL in cui è possibile utilizzare. AWS Clean Rooms

- AWS Clean Rooms SQL Spark

Per ulteriori informazioni in meritoAWS Clean Rooms, consulta la Guida per l'AWS Clean Roomsutente e l'AWS Clean RoomsAPI Reference.

Convenzioni del riferimento SQL

Questa sezione spiega le convenzioni utilizzate per scrivere la sintassi per le espressioni, i comandi e le funzioni SQL.

Carattere	Descrizione
CAPS	Le parole in lettere maiuscole sono parole chiave.
[ ]	Le parentesi indicano argomenti opzionali. Più argomenti tra parentesi indicano che è possibile scegliere qualsiasi numero degli argomenti. Inoltre, gli

Carattere	Descrizione
	argomenti tra parentesi su righe separate indicano che il parser prevede che gli argomenti siano nell'ordine in cui sono elencati nella sintassi.
{ }	Le parentesi graffe indicano che è necessario scegliere uno degli argomenti racchiusi nelle stesse.
|	Le pipe indicano che è possibile scegliere tra gli argomenti.
corsivo	Le parole in corsivo indicano dei segnaposto. Devi inserire il valore appropriato al posto della parola in corsivo.
...	I puntini di sospensione indicano che è possibile ripetere l'elemento precedente.
'	Le parole tra virgolette singole indicano che è necessario digitare le virgolette.

Regole di denominazione SQL

Le seguenti sezioni spiegano le regole di denominazione SQL in. AWS Clean Rooms

Argomenti

Nomi e colonne delle associazioni di tabelle configurate

I membri che possono eseguire query utilizzano i nomi di associazione di tabelle configurati come nomi di tabella nelle query. I nomi di associazione di tabelle configurati e le colonne di tabella configurate possono essere alias nelle query.

Le seguenti regole di denominazione si applicano ai nomi di associazione di tabelle configurati, ai nomi di colonne delle tabelle configurate e agli alias:

- Devono utilizzare solo caratteri alfanumerici, caratteri di sottolineatura (_) o trattino (-), ma non possono iniziare o terminare con un trattino.
  - (Solo regole di analisi personalizzate) Possono utilizzare il simbolo del dollaro ($) ma non uno schema che segue una costante di stringa tra virgolette in dollari.

    Una costante di stringa quotata in dollari è composta da:
    - il simbolo del dollaro ($)
    - un «tag» opzionale di zero o più caratteri
    - un altro simbolo del dollaro
    - sequenza arbitraria di caratteri che costituisce il contenuto della stringa
    - il simbolo del dollaro ($)
    - lo stesso tag con cui è iniziata la quotazione in dollari
    - il simbolo del dollaro

    Ad esempio: `$$invalid$$`
- Non possono contenere trattini consecutivi (-).
- Non possono iniziare con nessuno dei seguenti prefissi:

  `padb_`, `pg_`, `stcs_`, `stl_`, `stll_`, `stv_`, `svcs_`, `svl_`, `svv_`, `sys_`, `systable_`
- Non possono contenere barre rovesciate (\), virgolette (') o spazi che non siano tra virgolette.
- Se iniziano con un carattere non alfabetico, devono essere racchiusi tra virgolette (» «).
- Se contengono un trattino (-), devono essere racchiusi tra virgolette doppie (» «).
- Devono avere una lunghezza compresa tra 1 e 127 caratteri.
- Le parole riservate devono essere racchiuse tra virgolette doppie (» «).
- I seguenti nomi di colonna sono riservati e non possono essere utilizzati in AWS Clean Rooms (anche con virgolette):
  - oid
  - tableoid
  - xmin
  - cmin
  - xmax
  - cmax
  - ctid

Parole riservate

Di seguito è riportato un elenco di parole riservate in 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

Supporto dei tipi di dati tramite il motore SQL

AWS Clean Rooms supporta più motori e dialetti SQL. La comprensione dei sistemi di tipi di dati in queste implementazioni è fondamentale per una collaborazione e un'analisi dei dati di successo.

Le tabelle seguenti mostrano i tipi di dati equivalenti tra AWS Clean Rooms SQL, Snowflake SQL e Spark SQL.

Tipi di dati numerici

I tipi numerici rappresentano vari tipi di numeri, da numeri interi precisi a valori approssimativi in virgola mobile. La scelta del tipo numerico influisce sia sui requisiti di archiviazione che sulla precisione computazionale. I tipi di numeri interi variano in base alla dimensione del byte, mentre i tipi decimali e a virgola mobile offrono diverse opzioni di precisione e scala.

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
Numero intero a 8 byte	BIGINT	Non supportata	BIGINT, LUNGO	Numeri interi con segno da -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.
Numero intero a 4 byte	INT	Non supportata	INT, INTEGER	Numeri interi con segno da -2.147.483.648 a 2.147.483.647
Numero intero a 2 byte	SMALLINT	Non supportata	SMALLINT, BREVE	Numeri interi con segno da -32.768 a 32.767
Numero intero a 1 byte	Non supportata	Non supportata	TINYINT, BYTE	Numeri interi con segno da -128 a 127

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
Flottante a doppia precisione	DOPPIA, DOPPIA PRECISIONE	FLOAT,, FLOAT4 FLOAT8, DOPPIA, DOPPIA PRECISIONE, REALE	DOUBLE	Numeri in virgola mobile a doppia precisione da 8 byte
Flottante a precisione singola	REALE, GALLEGGIANTE	Non supportata	FLOAT	numeri in virgola mobile a precision e singola a 4 byte

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
Decimale (precisione fissa)	DECIMAL	DECIMALE, NUMERICO, NUMERO **Note** Snowflake assegna automaticamente l'alias dei tipi numerici esatti di larghezza inferiore (INT, BIGINT, SMALLINT, ecc.) a NUMBER.	DECIMALE, NUMERICO,	numeri decimali con segno a precisione arbitraria
Decimale (con precisione)	DECIMALE (p)	NUMERO DECIMALE (p), NUMERO (p)	DECIMALE (p)	Numeri decimali a precisione fissa
Decimale (con scala)	DECIMAL(p,s)	NUMERO DECIMALE (p, s), NUMERO (p, s)	DECIMAL(p,s)	Numeri decimali a precisione fissa con scala

Tipi di dati booleani

I tipi booleani rappresentano valori logici semplici. true/false Questi tipi sono coerenti tra i motori SQL e vengono comunemente utilizzati per flag, condizioni e operazioni logiche.

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
Booleano	BOOLEAN	BOOLEAN	BOOLEAN	Rappresenta i valori true/false

Tipi di dati data e ora

I tipi di data e ora gestiscono i dati temporali, con diversi livelli di precisione e consapevolezza del fuso orario. Questi tipi supportano diversi formati per la memorizzazione di date, ore e timestamp, con opzioni per includere o escludere le informazioni sul fuso orario.

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
Data	DATE	DATE	DATE	Valori di data (anno, mese, giorno) senza fuso orario
Orario	TIME	Non supportata	Non supportata	Ora del giorno in UTC, senza fuso orario
Ora con TZ	TIMETZ	Non supportata	Non supportata	Ora del giorno in UTC, con fuso orario
Time stamp	TIMESTAMP	TIMESTAMP, TIMESTAMP_NTZ	TIMESTAMP_NTZ	Timestamp senza fuso orario

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
				Note NTZ indica «Nessun fuso orario»
Timestamp con TZ	TIMESTAMPTZ	TIMESTAMP_LTZ	TIMESTAMP, TIMESTAMP_LTZ	Timestamp con fuso orario locale Note LTZ indica «Fuso orario locale»

Tipi di dati dei caratteri

I tipi di caratteri memorizzano dati testuali, offrendo opzioni a lunghezza fissa e a lunghezza variabile. Questi tipi gestiscono stringhe di testo e dati binari, con specifiche di lunghezza opzionali per controllare l'allocazione dello storage.

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
Carattere a lunghezza fissa	CHAR	CHAR, CHARACTER	CHAR, CHARACTER	Stringa di caratteri a

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
				lunghezza fissa
Carattere a lunghezza fissa con lunghezza	CHAR(n)	CHAR(n), CHARACTER (n)	CHAR(n), CHARACTER (n)	Stringa di caratteri a lunghezza fissa con lunghezza specificata
Carattere a lunghezza variabile	VARCHAR	VARCHAR, STRINGA, TESTO	VARCHAR, STRINGA	Stringa di caratteri a lunghezza variabile
Carattere a lunghezza variabile con lunghezza	VARCHAR(n)	VARCHAR (n), STRINGA (n), TESTO (n)	VARCHAR(n)	Stringa di caratteri a lunghezza variabile con limite di lunghezza
Binario	VARBYTE	BINARY, VARBINARY	BINARY	Sequenza binaria di byte
Binario con lunghezza	VARBYTE(n)	Non supportata	Non supportata	Sequenza binaria di byte con limite di lunghezza

Tipi di dati strutturati

I tipi strutturati consentono un'organizzazione complessa dei dati combinando più valori in singoli campi. Questi includono array per raccolte ordinate, mappe per coppie chiave-valore e strutture per la creazione di strutture di dati personalizzate con campi denominati.

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
Array	MATRICE <type>	ARRAY (tipo)	ARRAY <type>	Sequenza ordinata di elementi dello stesso tipo **Note** I tipi di array devono contenere elementi dello stesso tipo
Eseguire la mappatura	MAPPA<key, value>	MAP (chiave, valore)	MAPPA<key, value>	Raccolta di coppie chiave-valore **Note** I tipi di mappa devono contenere elementi dello stesso tipo

Tipo di dati	AWS Clean Rooms SQL	SQL a forma di fiocco di neve	Spark SQL	Description
Struct	STRUTTURA< field1: type1, field2: type2>	OGGETTO (campo1 tipo1, campo2 tipo2)	STRUTTURA < field1: type1, field2: type2 >	Struttura con campi denominati di tipi specificati **Note** La sintassi dei tipi struttura ti può variare leggermente tra le implementazioni
Fantastico	SUPER	Non supportata	Non supportata	Tipo flessibile che supporta tutti i tipi di dati, compresi i tipi complessi

AWS Clean Rooms SQL Spark

AWS Clean Rooms Spark SQL applica regole riguardanti l'uso di tipi di dati, espressioni e valori letterali.

Per ulteriori informazioni su AWS Clean Rooms Spark SQL, consulta la Guida per l'AWS Clean Rooms utente e l'API Reference.AWS Clean Rooms

I seguenti argomenti forniscono informazioni sui valori letterali, i tipi di dati, i comandi, le funzioni e le condizioni supportati in AWS Clean Rooms Spark SQL.

Argomenti

Valori letterali

Un valore letterale o una costante è un valore di dati fisso, composto da una sequenza di caratteri o da una costante numerica.

AWS Clean Rooms Spark SQL supporta diversi tipi di valori letterali, tra cui:

- Valori letterali per numeri interi, decimali e in virgola mobile.
- I caratteri letterali, noti anche come stringhe, stringhe di caratteri o costanti di caratteri, vengono utilizzati per specificare il valore di una stringa di caratteri.
- Valori letterali di data, ora e timestamp, utilizzati con i tipi di dati datetime. Per ulteriori informazioni, consulta Valori letterali di data, ora e timestamp.
- Valori letterali a intervalli. Per ulteriori informazioni, consulta Valori letterali di intervallo.
- Letterali booleani. Per ulteriori informazioni, consulta Letterali booleani.
- Valori letterali nulli, utilizzati per specificare un valore nullo.
- Solo TAB, CARRIAGE RETURN (CR) e LINE FEED (LF) Sono supportati i caratteri di controllo Unicode della categoria generale Unicode (Cc).

AWS Clean Rooms Spark SQL non supporta riferimenti diretti alle stringhe letterali nella clausola SELECT, ma possono essere utilizzati all'interno di funzioni come CAST.

Operatore + (concatenamento)

Concatena valori letterali numerici, stringhe letterali e/o valori letterali datetime e intervallari. Si trovano su entrambi i lati del simbolo + e restituiscono tipi diversi in base agli input su entrambi i lati del simbolo +.

Sintassi

```
numeric + string
```

```
date + time
```

```
date + timetz
```

L'ordine degli argomenti può essere invertito.

Argomenti

numeric literals

I valori letterali o le costanti che rappresentano numeri possono essere interi o in virgola mobile.

string literals

Stringhe, stringhe di caratteri o costanti di caratteri

date

A DATE colonna o un'espressione che si converte implicitamente in DATE.

time

A TIME colonna o un'espressione che si converte implicitamente in un TIME.

timetz

A TIMETZ colonna o un'espressione che si converte implicitamente in un TIMETZ.

Esempio

La seguente tabella di esempio TIME_TEST ha una colonna TIME_VAL (tipo TIME) con tre valori inseriti.

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

Tipi di dati

Ogni valore archiviato o recuperato da AWS Clean Rooms Spark SQL ha un tipo di dati con un set fisso di proprietà associate. I tipi di dati vengono dichiarati al momento della creazione delle tabelle. Un tipo di dati limita il set di valori che un argomento o una colonna può contenere.

La tabella seguente elenca i tipi di dati che puoi usare in AWS Clean Rooms Spark SQL.

Nome del tipo di dati	Tipo di dati	Alias	Description
ARRAY	the section called “Tipo annidato”	Non applicabile	Tipo di dati annidato nell'array
BIGINT	the section called “Tipi numerici”	Non applicabile	Intero a otto byte firmato
BINARY	the section called “Tipo binario”	Non applicabile	Valori della sequenza di byte
BOOLEAN	the section called “Tipo booleano”	BOOL	Booleano logico (true/false)
BYTE	the section called “Tipi numerici”	Non applicabile	Numeri interi con segno a 1 byte, da -128 a 127
CHAR	the section called “Tipi di carattere”	CHARACTER	Stringa di caratteri a lunghezza fissa
DATE	the section called “Tipi datetime”	Non applicabile	Data di calendario (anno, mese, giorno)

Nome del tipo di dati	Tipo di dati	Alias	Description
DECIMAL	the section called “Tipi numerici”	NUMERIC	Numerico esatto di precisione seleziona bile
FLOAT	the section called “Tipi numerici”	FLOAT8, DOPPIA PRECISIONE	Numero in virgola mobile a precisione doppia
INTEGER	the section called “Tipi numerici”	INT	Intero a quattro byte firmato
INTERVAL	the section called “Tipi datetime”	Non applicabile	Durata temporale in ordine giornaliero o in ordine annuo
LONG	the section called “Tipi numerici”	Non applicabile	numeri interi con segno a 8 byte
MAP	the section called “Tipo annidato”	Non applicabile	Tipo di dati annidato sulla mappa
REAL	the section called “Tipi numerici”	FLOAT4	Numero in virgola mobile a precisione singola
SHORT	the section called “Tipi numerici”	Non applicabile	Numeri interi con segno a 2 byte.
SMALLINT	the section called “Tipi numerici”	Non applicabile	Intero a due byte firmato
STRUCT	the section called “Tipo annidato”	Non applicabile	Struct: tipo di dati annidato
TIMESTAMP_LTZ	the section called “Tipi datetime”	Non applicabile	Ora del giorno con fuso orario locale

Nome del tipo di dati	Tipo di dati	Alias	Description
TIMESTAMP_NTZ	the section called “Tipi datetime”	Non applicabile	Ora del giorno senza fuso orario
TINYINT	the section called “Tipi numerici”	Non applicabile	numeri interi con segno a 1 byte, da -128 a 127
VARCHAR	the section called “Tipi di carattere”	CARATTERE VARIABILE	Stringa di caratteri a lunghezza variabile con un limite definito dall'utente

> **Note**
>
> I tipi di dati annidati ARRAY, STRUCT e MAP sono attualmente abilitati solo per la regola di analisi personalizzata. Per ulteriori informazioni, consulta Tipo annidato.

Caratteri multibyte

Il tipo di dati VARCHAR supporta caratteri multibyte UTF-8 fino a un massimo di quattro byte. I caratteri a cinque byte o più non sono supportati. Per calcolare la dimensione di una colonna VARCHAR che contiene caratteri multibyte, moltiplica il numero di caratteri per il numero di byte per carattere. Ad esempio, se una stringa ha quattro caratteri cinesi e ciascun carattere è lungo tre byte, allora avrai bisogno di una colonna VARCHAR(12) per memorizzare la stringa.

Il tipo di dati VARCHAR non supporta i punti di codice UTF-8 non validi seguenti:

`0xD800 – 0xDFFF` (Sequenze di byte: `ED A0 80` – `ED BF BF`)

Il tipo di dati CHAR non supporta caratteri multibyte.

Tipi numerici

I tipi di dati numerici comprendono numeri interi, decimali e in virgola mobile.

Argomenti

- Tipi Integer
- Tipo DECIMAL o NUMERIC
- Tipi in virgola mobile
- Calcoli con valori numerici

Tipi Integer

Usa i seguenti tipi di dati per memorizzare numeri interi di vari intervalli. Non è possibile memorizzare valori al di fuori dell'intervallo consentito per ogni tipo.

Name	Archiviazione	Intervallo
SMALLINT	2 byte	Da -32768 a +32767
SHORT	2 byte	Da -32768 a +32767
INTEGER o INT	4 byte	Da -2147483648 a +2147483647
BIGINT	8 byte	Da -92233720 36854775808 a 922337203 6854775807
LONG	8 byte	Da -92233720 36854775808 a 922337203 6854775807

Tipo DECIMAL o NUMERIC

Usare il tipo di dati DECIMAL o NUMERIC per memorizzare i valori con una precisione definita dall'utente. Le parole chiave DECIMAL e NUMERIC sono interscambiabili. In questo documento, decimale è il termine preferito per questo tipo di dati. Il termine numerici è usato solitamente per riferirsi a tipi di dati interi, decimali e in virgola mobile.

Archiviazione	Intervallo
Variabile, fino a 128 bit per i tipi DECIMAL non compressi.	Interi firmati da 128 bit con fino a 38 cifre di precisione.

Definisci una colonna DECIMAL in una tabella specificando un e: *precision scale*

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

precision

Il numero totale di cifre significative nell'intero valore: il numero di cifre su entrambi i lati del punto decimale. Ad esempio, il numero `48.2891` ha una precisione di 6 e una scala di 4. La precisione predefinita, se non specificata, è 18. La precisione massima è 38.

Se il numero di cifre a sinistra del punto decimale in un valore di input supera la precisione della colonna meno la scala, il valore non può essere copiato nella colonna (o inserito o aggiornato). Questa regola si applica a qualsiasi valore che non rientra nell'intervallo della definizione della colonna. Ad esempio, gli intervalli di valori ammessi per una colonna `numeric(5,2)` è da `-999.99` a `999.99`.

scale

Il numero di cifre decimale nella parte frazionaria del valore, alla destra del punto decimale. Gli interi hanno una scala di zero. Nella specifica di una colonna, è necessario che il valore della scala sia inferiore o uguale al valore della precisione. La scala predefinita, se non specificata, è 0. La scala massima è 37.

Se la scala di un valore input caricato in una tabella è maggiore della scala della colonna, il valore viene arrotondato alla scala specificata. Ad esempio, la colonna PRICEPAID nella tabella SALES è una colonna DECIMAL(8,2). Se un valore DECIMAL(8,4) viene inserito nella colonna PRICEPAID, il valore viene arrotondato alla scala di 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)
```

Tuttavia, i risultati di espliciti cast di valori selezionati dalle tabelle non sono arrotondati.

> **Note**
>
> Il valore positivo massimo che è possibile inserire in una colonna DECIMAL(19,0) è `9223372036854775807` (2^{63} -1). Il valore negativo massimo è `-9223372036854775807`. Ad esempio, il tentativo di inserire il valore `9999999999999999999` (19 nove) causerà un errore dell'overflow. Indipendentemente dalla posizione del punto decimale, la stringa più grande che AWS Clean Rooms può rappresentare come numero DECIMAL è `9223372036854775807`. Ad esempio, il valore più grande che è possibile caricare in una colonna DECIMAL(19,18) è `9.223372036854775807`.
> Queste regole sono dovute a quanto segue:
>
> - I valori DECIMAL con 19 o meno cifre significative di precisione vengono memorizzati internamente come numeri interi da 8 byte.
> - I valori DECIMAL con una precisione compresa tra 20 e 38 cifre significative vengono memorizzati come numeri interi da 16 byte.

Note sull'utilizzo di colonne NUMERIC o DECIMAL a 128 bit

Non assegnare in modo arbitrario la precisione massima alle colonne DECIMAL a meno che non sia certo che l'applicazione richieda tale precisione. I valori a 128 bit usano il doppio dello spazio su disco rispetto ai valori a 64 bit e possono quindi rallentare il tempo di esecuzione delle query.

Tipi in virgola mobile

Usa i tipi di dati REAL e DOUBLE PRECISION per memorizzare valori numerici con precisione variabile. Questi tipi sono inesatti, il che significa che alcuni valori vengono memorizzati come approssimazioni, così che la memorizzazione e la restituzione di un valore specifico possono risultare in lievi discrepanze. Se hai bisogno di calcoli e storage precisi (come per importi monetari), usa il tipo di dati DECIMAL.

REAL rappresenta il formato a virgola mobile a precisione singola, secondo lo standard IEEE 754 per l'aritmetica in virgola mobile. Ha una precisione di circa 6 cifre e un intervallo compreso tra 1E-37 e 1E+37. È inoltre possibile FLOAT4 specificare questo tipo di dati come.

DOUBLE PRECISION rappresenta il formato a virgola mobile a precisione doppia, secondo lo standard IEEE 754 per l'aritmetica binaria a virgola mobile. Ha una precisione di circa 15 cifre e un intervallo compreso tra 1E-307 e 1E+308. È inoltre possibile specificare questo tipo di dati come FLOAT o FLOAT8.

Calcoli con valori numerici

Nel AWS Clean Rooms, il calcolo si riferisce a operazioni matematiche binarie: addizione, sottrazione, moltiplicazione e divisione. Questa sezione descrive i tipi restituiti previsti per queste operazioni, nonché la formula specifica che viene applicata per determinare la precisione e la scala quando sono coinvolti tipi di dati DECIMAL.

Quando valori numerici vengono calcolati durante l'elaborazione di query, potresti affrontare casi in cui il calcolo è impossibile e la query restituisce un errore dell'overflow numerico. Potresti anche riscontrare casi in cui la scala di valori calcolati varia o è imprevista. Per alcune operazioni, è possibile usare il casting esplicito (promozione tipo) o i parametri di configurazione AWS Clean Rooms per risolvere questi problemi.

Per informazioni sui risultati di calcoli simili con funzioni SQL, consultare AWS Clean Rooms Funzioni Spark SQL.

Tipi restituiti per i calcoli

Dato il set di tipi di dati numerici supportati in AWS Clean Rooms, la tabella seguente mostra i tipi di rendimento previsti per le operazioni di addizione, sottrazione, moltiplicazione e divisione. La prima colonna sul lato sinistro della tabella rappresenta il primo operando nel calcolo e la riga in alto rappresenta il secondo operando.

Operando 1	Operando 2	Tipo restituito
SMALLINT o SHORT	SMALLINT o SHORT	SMALLINT o SHORT
SMALLINT o SHORT	INTEGER	INTEGER
SMALLINT o SHORT	BIGINT	BIGINT

Operando 1	Operando 2	Tipo restituito
SMALLINT o SHORT	DECIMAL	DECIMAL
SMALLINT o SHORT	FLOAT4	FLOAT8
SMALLINT o SHORT	FLOAT8	FLOAT8
INTEGER	INTEGER	INTEGER
INTEGER	BIGINT o LONG	BIGINT o LONG
INTEGER	DECIMAL	DECIMAL
INTEGER	FLOAT4	FLOAT8
INTEGER	FLOAT8	FLOAT8
BIGINT o LONG	BIGINT o LONG	BIGINT o LONG
BIGINT o LONG	DECIMAL	DECIMAL
BIGINT o LONG	FLOAT4	FLOAT8
BIGINT o LONG	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL
DECIMAL	FLOAT4	FLOAT8
DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8
FLOAT8	FLOAT8	FLOAT8

Precisione e scala di risultati DECIMAL calcolati

La tabella seguente riassume le regole per la scala e la precisione risultanti dal calcolo quando operazioni matematiche restituiscono risultati DECIMAL. In questa tabella, `p1` `s1` rappresentano la precisione e la scala del primo operando in un calcolo. `p2`e `s2` rappresentano la precisione e la scala

del secondo operando. (Indipendentemente da questi calcoli, la precisione del risultato massima è 38 e la scala del risultato massima è 38.)

Operation	Scala e precisione del risultato
+ oppure -	Dimensionare = `max(s1,s2)` Precisione = `max(p1-s1,p2-s2)+1+scale`
*	Dimensionare = `s1+s2` Precisione = `p1+p2+1`
/	Dimensionare = `max(4,s1+p2-s2+1)` Precisione = `p1-s1+ s2+scale`

Ad esempio, le colonne PRICEPAID e COMMISSION nella tabella SALES sono entrambe colonne DECIMAL(8,2). Se dividi PRICEPAID per COMMISSION (o viceversa), la formula è applicata come segue:

```
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)
```

Il calcolo seguente è la regola generale per il calcolo della scala e della precisione risultanti per le operazioni eseguite su valori DECIMAL con operatori impostati come UNION, INTERSECT ed EXCEPT o funzioni come COALESCE e DECODE:

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

Ad esempio, una DEC1 tabella con una colonna DECIMAL (7,2) viene unita a una DEC2 tabella con una colonna DECIMAL (15,3) per creare una tabella. DEC3 Lo schema di DEC3 mostra che la colonna diventa una colonna NUMERIC (15,3).

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

Nell'esempio sopra, la formula è applicata come segue:

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

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

Note sulle operazioni di divisione

Per le operazioni di divisione, le divide-by-zero condizioni restituiscono errori.

Il limite di scala di 100 è applicato dopo aver calcolato la precisione e la scala. Se la scala del risultato calcolata è maggiore di 100, i risultati della divisione vengono scalati come segue:

- Precisione = `precision - (scale - max_scale)`
- Dimensionare = `max_scale`

Se la precisione calcolata supera la precisione massima (38), la precisione viene ridotta a 38 e la scala diventa il risultato di: `max(38 + scale - precision), min(4, 100))`

Condizioni di overflow

L'overflow viene controllato per tutti i calcoli numerici. I dati DECIMAL con una precisione di 19 o inferiore vengono memorizzati come interi a 64 bit. I dati DECIMAL con una precisione superiore a 19 vengono memorizzati come interi a 128 bit. La precisione massima per tutti i valori DECIMAL è 38 e la scala massima è 37. Gli errori dell'overflow si verificano quando un valore supera questi limiti, che si applicano agli insiemi dei risultati finali e intermedi:

- Il casting esplicito genera errori di overflow in fase di esecuzione quando valori di dati specifici non corrispondono alla precisione o alla scala richieste specificate dalla funzione cast. Ad esempio, non è possibile eseguire il cast di tutti i valori della colonna PRICEPAID nella tabella SALES (una colonna DECIMAL (8,2)) e restituire un risultato DECIMAL (7,3):

  ```
  select pricepaid::decimal(7,3) from sales;
  ```

```
ERROR:  Numeric data overflow (result precision)
```

Questo errore si verifica perché alcuni dei valori più grandi nella colonna PRICEPAID non possono essere espressi.

- Le operazioni di moltiplicazione producono risultati in cui la scala del risultato è la somma della scala di ciascun operando. Se entrambi gli operando hanno una scala di 4, ad esempio, la scala del risultato è 8, lasciando solo 10 cifre per il lato sinistro del punto decimale. Pertanto, è relativamente facile incorrere in condizioni di overflow quando si moltiplicano due grandi numeri che possiedono entrambi una scala significativa.

Calcoli numerici con tipi INTEGER e DECIMAL

Quando uno degli operandi di un calcolo ha un tipo di dati INTEGER e l'altro operando è DECIMAL, l'operando INTEGER viene implicitamente espresso come DECIMAL.

- SMALLINT o SHORT viene espresso come DECIMAL (5,0)
- INTEGER viene espresso come DECIMAL (10,0)
- BIGINT o LONG viene espresso come DECIMAL (19,0)

Ad esempio, se moltiplichi SALES.COMMISSION, una colonna DECIMAL(8,2), e SALES.QTYSOLD, una colonna SMALLINT, per questo calcolo viene eseguito il cast come segue:

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

Tipi di carattere

I tipi di dati carattere comprendono CHAR (carattere) e VARCHAR (carattere variabile).

Argomenti

- CHAR o CHARACTER
- VARCHAR o CHARACTER VARYING
- Significato degli spazi finali

CHAR o CHARACTER

Usa una colonna CHAR o CHARACTER per memorizzare stringhe di lunghezza fissa. A queste stringhe vengono aggiunti spazi, quindi una colonna CHAR(10) occupa 10 byte di storage.

```
char(10)
```

Una colonna CHAR senza una specificazione di lunghezza risulta in una colonna CHAR(1).

I tipi di dati CHAR e VARCHAR sono definiti in termini di byte, non caratteri. Una colonna CHAR può contenere solo caratteri a byte singolo, quindi una colonna CHAR(10) può contenere una stringa con una lunghezza massima di 10 byte.

Name	Archiviazione	Intervallo (larghezza della colonna)
CHAR o CHARACTER	Lunghezza della stringa, compresi spazi finali (se presenti)	4096 byte

VARCHAR o CHARACTER VARYING

Usa una colonna VARCHAR o CHARACTER VARYING per memorizzare stringhe di lunghezza variabile con un limite fisso. A queste stringhe non vengono aggiunti spazi vuoti, quindi una colonna VARCHAR(120) consiste di un massimo di 120 caratteri a byte singolo, 60 caratteri a due byte, 40 caratteri a tre byte o 30 caratteri a quattro byte.

```
varchar(120)
```

I tipi di dati VARCHAR sono definiti in termini di byte, non di caratteri. Una VARCHAR può contenere caratteri multibyte fino a un massimo di quattro byte per carattere. Ad esempio, una colonna VARCHAR(12) può contenere 12 caratteri a byte singolo, 6 caratteri a due byte, 4 caratteri a tre byte o 3 caratteri a quattro byte.

Name	Archiviazione	Intervallo (larghezza della colonna)
VARCHAR o CHARACTER VARYING	4 byte + byte totali per	65.535 bytes (64K -1)

Name	Archiviazione	Intervallo (larghezza della colonna)
	carattere, dove ogni carattere può essere da 1 a 4 byte.	

Significato degli spazi finali

Entrambi i tipi di dati CHAR e VARCHAR memorizzano stringhe fino a n byte di lunghezza. Un tentativo di memorizzare una stringa più lunga in una colonna di questi tipi genera un errore. Tuttavia, se i caratteri aggiuntivi sono tutti spazi (spazi vuoti), la stringa viene troncata alla lunghezza massima. Se la stringa è più corta della lunghezza massima, ai valori CHAR vengono aggiunti spazi, ma i valori VARCHAR memorizzano la stringa senza spazi.

Gli spazi inziali nei valori CHAR sono sempre privi di significato dal punto di vista semantico. Vengono ignorati quando confronti due valori CHAR, non compresi nei calcoli LENGTH, e rimossi quando converti un valore CHAR in un altro tipo di stringa.

Gli spazi finali nei valori VARCHAR e CHAR vengono trattati come insignificanti dal punto di vista semantico quando i valori vengono confrontati.

I calcoli della lunghezza restituiscono la lunghezza delle stringhe di caratteri VARCHAR con spazi finali compresi nella lunghezza. Gli spazi finali non vengono contati nella lunghezza per le stringhe di caratteri a lunghezza fissa.

Tipi datetime

I tipi di dati Datetime includono DATE, TIME, TIMESTAMP_LTZ e TIMESTAMP_NTZ.

Argomenti

- Tipi di dati e valori letterali relativi agli intervalli

DATE

Utilizzare il tipo di dati DATE per memorizzare semplici date di calendario senza timestamp.

Name	Archiviazione	Intervallo	Risoluzione
DATE	4 byte	Da 4.713 BC a 294.276 AD	1 giorno

TIMESTAMP_LTZ

Usa il tipo di dati TIMESTAMP_LTZ per memorizzare valori di timestamp completi che includono la data, l'ora del giorno e il fuso orario locale.

TIMESTAMP rappresenta i valori che comprendono i valori dei campi`year`,,,, `month`, `day` con il fuso orario locale della sessione. `hour` `minute` `second` Il `timestamp` valore rappresenta un punto temporale assoluto.

TIMESTAMP in Spark è un alias specificato dall'utente associato a una delle varianti TIMESTAMP_LTZ e TIMESTAMP_NTZ. Puoi impostare il tipo di timestamp predefinito come TIMESTAMP_LTZ (valore predefinito) o TIMESTAMP_NTZ tramite la configurazione. `spark.sql.timestampType`

TIMESTAMP_NTZ

Usa il tipo di dati TIMESTAMP_NTZ per memorizzare valori di timestamp completi che includono la data e l'ora del giorno, senza il fuso orario locale.

TIMESTAMP rappresenta i valori che comprendono i valori dei campi,,,, e. `year` `month` `day` `hour` `minute` `second` Tutte le operazioni vengono eseguite senza tenere conto del fuso orario.

TIMESTAMP in Spark è un alias specificato dall'utente associato a una delle varianti TIMESTAMP_LTZ e TIMESTAMP_NTZ. Puoi impostare il tipo di timestamp predefinito come TIMESTAMP_LTZ (valore predefinito) o TIMESTAMP_NTZ tramite la configurazione. `spark.sql.timestampType`

Esempi con tipi datetime

Gli esempi seguenti mostrano come lavorare con i tipi di datetime supportati da. AWS Clean Rooms

Esempi di data

Nei seguenti esempi vengono inserite date con formati diversi e viene visualizzato il risultato.

```
select * from datetable order by 1;

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

Se si inserisce un valore timestamp in una colonna DATE, la parte dell'ora viene ignorata e viene caricata solo la data.

Esempi di orari

Negli esempi seguenti vengono inseriti valori TIME e TIMETZ con formati diversi e viene visualizzato il risultato.

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

Valori letterali di data, ora e timestamp

Di seguito sono riportate le regole per lavorare con i valori letterali di data, ora e timestamp supportati da Spark SQL. AWS Clean Rooms

Date:

La tabella seguente mostra le date di input che sono esempi validi di valori di data letterali che è possibile caricare nelle tabelle. AWS Clean Rooms Si assume che la modalità `MDY DateStyle` di default sia in vigore. Questa modalità indica che il valore del mese precede il valore del giorno in stringhe come `1999-01-08` e `01/02/00`.

> **Note**
>
> È necessario che un valore letterale data o timestamp quando viene caricato in una tabella sia racchiuso tra virgolette.

Data di input	Data completa
8 gennaio 1999	8 gennaio 1999
1999-01-08	8 gennaio 1999
1/8/1999	8 gennaio 1999
01/02/00	2 gennaio 2000
2000-Jan-31	31 gennaio 2000
Jan-31-2000	31 gennaio 2000
31-Jan-2000	31 gennaio 2000
20080215	15 febbraio 2008
080215	15 febbraio 2008
2008.366	31 dicembre 2008 (è necessario che la parte a 3 cifre della data sia compresa tra 001 e 366)

Volte

La tabella seguente mostra gli orari di input che sono esempi validi di valori temporali letterali che è possibile caricare nelle AWS Clean Rooms tabelle.

Input di orari	Descrizione (della parte dell'ora)
04:05:06.789	4:05 AM e 6.789 secondi
04:05:06	4:05 AM e 6 secondi

Input di orari	Descrizione (della parte dell'ora)
04:05	4:05 AM preciso
040506	4:05 AM e 6 secondi
04:05 AM	4:05 AM preciso; AM è facoltativo
04:05 PM	4:05 PM precise; è necessario che il valore dell'ora sia < 12.
16:05	4:05 PM preciso

Valori datetime speciali

La tabella seguente mostra valori speciali che possono essere usati come valori letterali di data/ora e come argomenti per le funzioni di data. Richiedono virgolette singole e vengono convertiti in valori timestamp regolari durante l'elaborazione delle query.

Valore speciale	Description
`now`	Valuta all'ora di inizio della transazione attuale e restituisce un timestamp con precisione di microsecondi.
`today`	Valuta alla data appropriata e restituisce un timestamp con più zeri al posto dell'ora.
`tomorrow`	Valuta alla data appropriata e restituisce un timestamp con più zeri al posto dell'ora.
`yesterday`	Valuta alla data appropriata e restituisce un timestamp con più zeri al posto dell'ora.

Gli esempi seguenti mostrano come `now` e come utilizzare la funzione `today` 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)
```

Valori letterali di intervallo

Di seguito sono riportate le regole per lavorare con i valori letterali a intervalli supportati da Spark SQL. AWS Clean Rooms

Usa un valore letterale di intervallo per identificare periodi di tempo specifici, come `12 hours` o `6 weeks`. È possibile usare questi valori letterali di intervallo in condizioni e calcoli che comprendono espressioni datetime.

> **Note**
>
> Non è possibile utilizzare il tipo di dati INTERVAL per le colonne nelle tabelle. AWS Clean Rooms

Un intervallo viene espresso come una combinazione della parola chiave INTERVAL con una quantità numerica e una parte di data supportata, ad esempio `INTERVAL '7 days'` o `INTERVAL '59 minutes'`. È possibile collegare diverse quantità e unità per formare un intervallo più preciso; ad esempio `INTERVAL '7 days, 3 hours, 59 minutes'`. Anche abbreviazioni e plurali di ciascuna unità sono supportati; ad esempio: `5 s`, `5 second` e `5 seconds` sono intervalli equivalenti.

Se non si specifica una parte data, il valore di intervallo rappresenterà i secondi. È possibile specificare il valore di quantità come una frazione (ad esempio: `0.5 days`).

Esempi

Gli esempi seguenti mostrano una serie di calcoli con valori di intervallo diversi.

L'esempio seguente aggiunge 1 secondo alla data specificata.

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

L'esempio seguente aggiunge 1 minuto alla data specificata.

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

L'esempio seguente aggiunge 3 ore e 35 minuti alla data specificata.

```
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)
```

L'esempio seguente aggiunge 52 settimane alla data specificata.

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

L'esempio seguente aggiunge 1 settimana, 1 ora, 1 minuto e 1 secondo alla data specificata.

```
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)
```

L'esempio seguente aggiunge 12 ore (mezza giornata) alla data specificata.

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

L'esempio seguente sottrae 4 mesi dal 31 marzo 2023 e il risultato è il 30 novembre 2022. Il calcolo considera il numero di giorni in un mese.

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

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

Tipi di dati e valori letterali relativi agli intervalli

Puoi utilizzare un tipo di dati di intervallo per memorizzare le durate di tempo in unità quali `seconds`, `minutes`, `hours`, `days`, `months` e `years`. I tipi di dati e i valori letterali relativi agli intervalli possono essere utilizzati nei calcoli data/ora, ad esempio aggiungendo intervalli a date e timestamp, sommando gli intervalli e sottraendo un intervallo da una data o un timestamp. I valori letterali relativi agli intervalli possono essere utilizzati come valori per intervallare le colonne dei tipi di dati in una tabella.

Sintassi del tipo di dati di intervallo

Come specificare un tipo di dati di intervallo per memorizzare una durata di tempo in anni e mesi:

```
INTERVAL year_to_month_qualifier
```

Come specificare un tipo di dati di intervallo per memorizzare una durata in giorni, ore, minuti e secondi:

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

Sintassi dell'intervallo letterale

Come specificare un intervallo letterale per definire una durata di tempo in anni e mesi:

```
INTERVAL quoted-string year_to_month_qualifier
```

Come specificare un intervallo letterale per definire una durata in giorni, ore, minuti e secondi:

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

Arguments (Argomenti)

quoted-string

Specifica un valore numerico positivo o negativo che indica una quantità e l'unità data/ora come stringa di input. Se la stringa tra virgolette contiene solo un numero, AWS Clean Rooms determina le unità da year_to_month_qualifier o day_to_second_qualifier. Ad esempio, `'23' MONTH` rappresenta `1 year 11 months`, `'-2' DAY` rappresenta `-2 days 0 hours 0 minutes 0.0 seconds`, `'1-2' MONTH` rappresenta `1 year 2 months` e `'13 day 1 hour 1 minute 1.123 seconds' SECOND` rappresenta `13 days 1 hour 1 minute 1.123 seconds`. Per ulteriori informazioni sui formati di output di un intervallo, consulta Stili di intervallo.

year_to_month_qualifier

Specifica l'intervallo di valori. Se utilizzate un qualificatore e create un intervallo con unità di tempo più piccole del qualificatore, tronca e scarta le parti più piccole dell'intervallo. AWS Clean Rooms I valori validi per year_to_month_qualifier sono:

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

day_to_second_qualifier

Specifica l'intervallo di valori. Se utilizzate un qualificatore e create un intervallo con unità di tempo più piccole del qualificatore, tronca e scarta le parti più piccole dell'intervallo. AWS Clean Rooms I valori validi per day_to_second_qualifier sono:

- `DAY`
- `HOUR`
- `MINUTE`

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

L'output del valore letterale INTERVAL viene troncato in base al componente INTERVAL più piccolo specificato. Ad esempio, quando si utilizza un qualificatore MINUTE, scarta le unità di tempo più piccole di MINUTE. AWS Clean Rooms

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

Il valore risultante viene troncato in `'1 day 01:01:00'`.

fractional_precision

Parametro facoltativo che specifica il numero di cifre frazionarie consentite nell'intervallo. L'argomento fractional_precision deve essere specificato solo se l'intervallo contiene SECOND. Ad esempio, `SECOND(3)` crea un intervallo che prevede solo tre cifre frazionarie, ad esempio 1,234 secondi. Il numero massimo di cifre frazionarie è sei.

La configurazione della sessione `interval_forbid_composite_literals` determina se viene restituito un errore quando viene specificato un intervallo con le parti YEAR TO MONTH e DAY TO SECOND.

Aritmetica dell'intervallo

Puoi utilizzare i valori di intervallo con altri valori di data e ora per eseguire operazioni aritmetiche. Le tabelle seguenti descrivono le operazioni disponibili e il tipo di dati risultante da ciascuna operazione.

> **Note**
>
> Le operazioni che possono produrre entrambi i risultati `date` e `timestamp` lo fanno in base alla più piccola unità di tempo coinvolta nell'equazione. Ad esempio, quando aggiungi un valore `interval` a un valore `date`, il risultato è un valore `date`, se si tratta di un intervallo YEAR TO MONTH e un timestamp se si tratta di un intervallo DAY TO SECOND.

Le operazioni in cui il primo operando è un valore `interval` producono i seguenti risultati per il secondo operando specificato:

Operatore	Data	Time stamp	Interval	Numerico
-	N/D	N/D	Interval	N/D
+	Data	Data/Timestamp	Interval	N/D
*	N/D	N/D	N/D	Interval
/	N/D	N/D	N/D	Interval

Le operazioni in cui il primo operando è un `date` producono i seguenti risultati per il secondo operando specificato:

Operatore	Data	Time stamp	Interval	Numerico
-	Numerico	Interval	Data/Timestamp	Data
+	N/D	N/D	N/D	N/D

Le operazioni in cui il primo operando è un `timestamp` producono i seguenti risultati per il secondo operando specificato:

Operatore	Data	Time stamp	Interval	Numerico
-	Numerico	Interval	Time stamp	Time stamp
+	N/D	N/D	N/D	N/D

Stili di intervallo

- `postgres`: segue lo stile PostgreSQL. Questa è l'impostazione predefinita.
- `postgres_verbose`: segue lo stile verboso PostgreSQL.
- `sql_standard`: segue lo stile dei valori letterali dell'intervallo standard SQL.

Il comando seguente imposta lo stile dell'intervallo su `sql_standard`.

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

formato di output postgres

Di seguito è riportato il formato di output per lo stile di intervallo `postgres`. Ogni valore numerico può essere negativo.

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

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

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

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

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

formato di output postgres_verbose

La sintassi di postgres_verbose è simile a quella di postgres, ma gli output di postgres_verbose contengono anche l'unità di tempo.

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

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

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

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

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

formato di output sql_standard

I valori dell'intervallo da anno a mese sono formattati come segue. Se si specifica un segno negativo prima dell'intervallo, si indica che l'intervallo è un valore negativo e si applica all'intero intervallo.

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

I valori dell'intervallo da giorno a secondo sono formattati come segue.

```
'[-]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
```

Esempi di tipi di dati relativi agli intervalli

Negli esempi seguenti viene illustrato come utilizzare i tipi di dati INTERVAL con le tabelle.

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

Esempi di valori letterali relativi agli intervalli

Gli esempi seguenti vengono eseguiti con lo stile di intervallo impostato su `postgres`.

L'esempio seguente mostra come creare un valore letterale di INTERVAL di 1 anno.

```
select INTERVAL '1' YEAR

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

Se si specifica un argomento quoted-string che supera il qualificatore, le unità di tempo rimanenti vengono troncate dall'intervallo. Nell'esempio seguente, un intervallo di 13 mesi diventa 1 anno e 1 mese, ma il mese restante viene escluso a causa del qualificatore YEAR.

```
select INTERVAL '13 months' YEAR

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

Se utilizzi un qualificatore inferiore alla stringa di intervallo, vengono incluse le unità rimanenti.

```
select INTERVAL '13 months' MONTH

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

Se si specifica una precisione nell'intervallo, il numero di cifre frazionarie viene troncato in base alla precisione specificata.

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

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

Se non si specifica una precisione, AWS Clean Rooms utilizza la precisione massima di 6.

```
select INTERVAL '1.23456789' SECOND

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

L'esempio seguente dimostra come creare un intervallo di valori.

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

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

I qualificatori determinano le unità che specificate. Ad esempio, anche se l'esempio seguente utilizza la stessa stringa tra virgolette '2:2' dell'esempio precedente, AWS Clean Rooms riconosce che utilizza unità di tempo diverse a causa del qualificatore.

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

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

Sono supportati anche le abbreviazioni e i plurali di ciascuna unità. Ad esempio, `5s`, `5 second` e `5 seconds` sono intervalli equivalenti. Le unità supportate sono anni, mesi, ore, minuti e secondi.

```
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
```

Esempi di valori letterali relativi agli intervalli senza sintassi dei qualificatori

> **Note**
>
> Negli esempi seguenti viene illustrato l'utilizzo di un valore letterale relativo agli intervalli senza un qualificatore `YEAR TO MONTH` o `DAY TO SECOND`. Per informazioni sull'utilizzo del valore letterale relativo agli intervalli consigliato con un qualificatore, consulta Tipi di dati e valori letterali relativi agli intervalli.

Usa un valore letterale di intervallo per identificare periodi di tempo specifici, come `12 hours` o `6 months`. È possibile usare questi valori letterali di intervallo in condizioni e calcoli che comprendono espressioni datetime.

Un valore letterale relativo agli intervalli viene espresso come una combinazione della parola chiave INTERVAL con una quantità numerica e una parte di data supportata, ad esempio `INTERVAL '7 days'` o `INTERVAL '59 minutes'`. È possibile collegare diverse quantità e unità per formare un intervallo più preciso; ad esempio `INTERVAL '7 days, 3 hours, 59 minutes'`. Anche

abbreviazioni e plurali di ciascuna unità sono supportati; ad esempio: `5 s`, `5 second` e `5 seconds` sono intervalli equivalenti.

Se non si specifica una parte data, il valore di intervallo rappresenterà i secondi. È possibile specificare il valore di quantità come una frazione (ad esempio: `0.5 days`).

Gli esempi seguenti mostrano una serie di calcoli con valori di intervallo diversi.

Quanto segue aggiunge 1 secondo alla data specificata.

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

Quanto segue aggiunge 1 minuto alla data specificata.

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

Vengono aggiunte 3 ore e 35 minuti alla data specificata.

```
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)
```

Quanto segue aggiunge 52 settimane alla data specificata.

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

```
(1 row)
```

Vengono aggiunti 1 settimana, 1 ora, 1 minuto e 1 secondo alla data specificata.

```
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)
```

Di seguito vengono aggiunte 12 ore (mezza giornata) alla data specificata.

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

Vengono sottratti 4 mesi dal 15 febbraio 2023 e il risultato è il 15 ottobre 2022.

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

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

Vengono sottratti 4 mesi dal 31 marzo 2023 e il risultato è il 30 novembre 2022. Il calcolo considera il numero di giorni in un mese.

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

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

Tipo booleano

Usa valori di dati BOOLEAN per memorizzare valori true e false in una colonna a byte singolo. La tabella seguente descrive i tre possibili stati per un valore booleano e i valori letterali che risultano in

quello stato. Indipendentemente dalla stringa di input, una colonna booleana memorizza e restituisce "t" per true e "f" per false.

Stato	Valori letterali validi	Archiviazione
True	`TRUE 't' 'true' 'y' 'yes' '1'`	1 byte
False	`FALSE 'f' 'false' 'n' 'no' '0'`	1 byte
Sconosciuto	`NULL`	1 byte

È possibile utilizzare un confronto IS per controllare il valore Boolean solo come predicato nella clausola WHERE. Non è possibile utilizzare un confronto IS con un valore Boolean nell'elenco SELECT.

Esempi

È possibile utilizzare una colonna BOOLEAN per memorizzare uno stato «Attivo/Inattivo» per ogni cliente in una tabella CUSTOMER.

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

In questo esempio, la seguente query seleziona gli utenti della tabella USERS che amano lo sport ma non amano il teatro:

```
select firstname, lastname, likesports, liketheatre
from users
where likesports is true and liketheatre is false
order by userid limit 10;

firstname |  lastname  | likesports | liketheatre
```

```
----------+------------+------------+-------------
Alejandro | Rosalez    | t          | f
Akua      | Mansa      | t          | f
Arnav     | Desai      | t          | f
Carlos    | Salazar    | t          | f
Diego     | Ramirez    | t          | f
Efua      | Owusu      | t          | f
John      | Stiles     | t          | f
Jorge     | Souza      | t          | f
Kwaku     | Mensah     | t          | f
Kwesi     | Manu       | t          | f
(10 rows)
```

Il seguente esempio seleziona dalla tabella USERS gli utenti per i quali non si sa se gradiscono la musica 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)
```

L'esempio seguente restituisce un errore perché utilizza un confronto IS nell'elenco SELECT.

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

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

L'esempio seguente riesce perché utilizza un confronto uguale (=) nell'elenco SELECT anziché il IS confronto.

```
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    |
```

Letterali booleani

Le seguenti regole servono per lavorare con i letterali booleani supportati da Spark SQL. AWS Clean Rooms

Usa un valore letterale booleano per specificare un valore booleano, ad esempio o. `TRUE` `FALSE`

Sintassi

```
TRUE | FALSE
```

Esempio

L'esempio seguente mostra una colonna con un valore specificato di. `TRUE`

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

Tipo binario

Usa il tipo di dati BINARY per archiviare e gestire dati binari a lunghezza fissa e non interpretati, fornendo funzionalità di archiviazione e confronto efficienti per casi d'uso specifici.

Il tipo di dati BINARY memorizza un numero fisso di byte, indipendentemente dalla lunghezza effettiva dei dati archiviati. La lunghezza massima è in genere di 255 byte.

BINARY viene utilizzato per archiviare dati binari non elaborati e non interpretati, come immagini, documenti o altri tipi di file. I dati vengono memorizzati esattamente come vengono forniti, senza alcuna codifica o interpretazione dei caratteri. I dati binari memorizzati nelle colonne BINARY vengono confrontati e ordinati byte-by-byte in base ai valori binari effettivi, anziché a qualsiasi regola di codifica o collazione dei caratteri.

La seguente query di esempio mostra la rappresentazione binaria della stringa. `"abc"` Ogni carattere della stringa è rappresentato dal relativo codice ASCII in formato esadecimale: «a» è 0x61, «b» è 0x62 e «c» è 0x63. Quando combinati, questi valori esadecimali formano la rappresentazione binaria. `"616263"`

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

Tipo annidato

AWS Clean Rooms supporta le query che coinvolgono dati con tipi di dati annidati, in particolare i tipi di colonne AWS Glue STRUCT, ARRAY e MAP. Solo la regola di analisi personalizzata supporta i tipi di dati annidati.

In particolare, i tipi di dati annidati non sono conformi alla struttura rigida e tabulare del modello di dati relazionali dei database SQL.

I tipi di dati annidati contengono tag che fanno riferimento a entità distinte all'interno dei dati. Possono contenere valori complessi quali array, strutture nidificate e altre strutture complesse associate ai formati di serializzazione, ad esempio JSON. I tipi di dati nidificati supportano fino a 1 MB di dati per un singolo campo o oggetto del tipo di dati nidificati.

Argomenti

- Tipo ARRAY

Tipo ARRAY

Usa il tipo ARRAY per rappresentare valori costituiti da una sequenza di elementi con il tipo di. `elementType`

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

Utilizzato `containsNull` per indicare se gli elementi di un tipo ARRAY possono avere `null` valori.

Tipo MAP

Usa il tipo MAP per rappresentare i valori che comprendono un insieme di coppie chiave-valore.

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

`keyType`: il tipo di dati delle chiavi

`valueType`: il tipo di dati dei valori

Le chiavi non possono avere `null` valori. `valueContainsNull`Da utilizzare per indicare se i valori di un valore di tipo MAP possono avere `null` valori.

Tipo STRUCT

Usa il tipo STRUCT per rappresentare i valori con la struttura descritta da una sequenza di StructFields (campi).

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

StructField(name, dataType, nullable): rappresenta un campo in un. StructType

`dataType`: il tipo di dati di un campo

`name`: il nome di un campo

Viene utilizzato `nullable` per indicare se i valori di questi campi possono avere `null` valori.

Esempi di tipi di dati annidati

Per il `struct<given:varchar, family:varchar>` tipo, ci sono due nomi di attributo:`given`, `efamily`, ciascuno corrispondente a un `varchar` valore.

Per il `array<varchar>` tipo, l'array viene specificato come elenco di`varchar`.

Il `array<struct<shipdate:timestamp, price:double>>` tipo si riferisce a un elenco di elementi con `struct<shipdate:timestamp, price:double>` tipo.

Il tipo di `map` dati si comporta come un `array` di`structs`, in cui il nome dell'attributo per ogni elemento dell'array è indicato da `key` e viene mappato a. `value`

Example

Ad esempio, il `map<varchar(20), varchar(20)>` tipo viene trattato come`array<struct<key:varchar(20), value:varchar(20)>>`, dove `key` e `value` fa riferimento agli attributi della mappa nei dati sottostanti.

Per informazioni su come AWS Clean Rooms abilitare la navigazione in matrici e strutture, vedereNavigazione.

Per informazioni su come AWS Clean Rooms abilitare l'iterazione sugli array navigando nell'array utilizzando la clausola FROM di una query, vedere. Annullamento di query

Conversione e compatibilità dei tipi

I seguenti argomenti descrivono come funzionano le regole di conversione dei tipi e la compatibilità dei tipi di dati in AWS Clean Rooms Spark SQL.

Argomenti

- Compatibilità
- Regole generali di conversione e compatibilità
- Tipi di conversione implicita

Compatibilità

La corrispondenza dei tipi di dati e la corrispondenza di valori letterali e costanti con tipi di dati avviene durante diverse operazioni di database, comprese le seguenti:

- Operazioni DML (Data Manipulation Language) sulle tabelle

- Query UNION, INTERSECT ed EXCEPT
- Espressioni CASE
- Valutazione di predicati, come LIKE e IN
- Valutazione di funzioni SQL che effettuano confronti o estrazioni di dati
- Confronti con operatori matematici

I risultati di queste operazioni dipendono dalle regole di conversione dei tipi e dalla compatibilità dei tipi di dati. La compatibilità implica che non è sempre richiesta la one-to-one corrispondenza tra un determinato valore e un determinato tipo di dati. Poiché alcuni tipi di dati sono compatibili, è possibile una conversione implicita o una coercizione. Per ulteriori informazioni, consulta Tipi di conversione implicita. Quando i tipi di dati non sono compatibili, a volte è possibile convertire un valore da un tipo di dati a un altro usando una funzione di conversione esplicita.

Regole generali di conversione e compatibilità

Osserva le seguenti regole di conversione e compatibilità:

- In generale, i tipi di dati che rientrano nella stessa categoria (come diversi tipi di dati numerici) sono compatibili ed è possibile convertirli in modo implicito.

  Ad esempio, con la conversione implicita è possibile inserire un valore decimale in una colonna intera. Il decimale viene arrotondato per produrre un numero intero. Altrimenti, è possibile estrarre un valore numerico, come `2008`, da una data e inserirlo nella colonna intera.
- I tipi di dati numerici applicano le condizioni di overflow che si verificano quando si tenta di inserire valori. out-of-range Ad esempio, un valore decimale con una precisione di 5 non rientra in una colonna decimale che è stata definita con una precisione di 4. Un numero intero o l'intera parte di un decimale non viene mai troncato. Tuttavia, la parte frazionaria di un decimale può essere arrotondata per eccesso o per difetto, a seconda dei casi. Tuttavia, i risultati di espliciti cast di valori selezionati dalle tabelle non sono arrotondati.
- Sono compatibili diversi tipi di stringhe di caratteri. Le stringhe di colonna VARCHAR contenenti dati a byte singolo e le stringhe di colonna CHAR sono comparabili e convertibili implicitamente. Le stringhe VARCHAR che contengono dati multibyte non sono confrontabili. Inoltre, è possibile convertire una stringa di caratteri in una data, ora, timestamp o valore numerico se la stringa è un valore letterale appropriato. Tutti gli spazi iniziali o finali vengono ignorati. Per contro, è possibile convertire un valore numero, timestamp o data in una stringa di caratteri a lunghezza variabile o fissa.

> **Note**
>
> È necessario che una stringa di caratteri per la quale si desidera eseguire il cast a un tipo numerico contenga una rappresentazione in caratteri di un numero. Ad esempio, è possibile eseguire il cast delle stringhe `'1.0'` o `'5.9'` dei valori decimali, ma non è possibile eseguire il cast della stringa `'ABC'` in alcun tipo numerico.

- Se si confrontano i valori DECIMAL con le stringhe di caratteri, AWS Clean Rooms tenta di convertire la stringa di caratteri in un valore DECIMAL. Quando si confrontano tutti gli altri valori numerici con stringhe di caratteri, i valori numerici vengono convertiti in stringhe di caratteri. Per applicare la conversione opposta (ad esempio, convertire le stringhe di caratteri in numeri interi o convertire i valori DECIMAL in stringhe di caratteri), usa una funzione esplicita, ad esempio, Funzione CAST.
- Per convertire valori DECIMAL o NUMERIC a 64 bit in una precisione più elevata, è necessario usare una funzione di conversione specifica come le funzioni CAST o CONVERT.

Tipi di conversione implicita

Ci sono due tipi di conversione implicita:

- Conversioni implicite nelle assegnazioni, come l'impostazione di valori nei comandi INSERT o UPDATE
- Conversioni implicite nelle espressioni, ad esempio l'esecuzione di confronti nella clausola WHERE

La tabella seguente elenca i tipi di dati che possono essere convertiti implicitamente in assegnazioni o espressioni. È anche possibile usare una funzione di conversione esplicita per eseguire queste conversioni.

Dal tipo	Al tipo
BIGINT	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)

Dal tipo	Al tipo
	DOPPIA PRECISIONE () FLOAT8
	INTEGER
	REALE (FLOAT4)
	SMALLINT o SHORT
	VARCHAR
CHAR	VARCHAR
DATE	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT o LONG
	CHAR
	DOPPIA PRECISIONE () FLOAT8
	INTERO (INT)
	REALE () FLOAT4
	SMALLINT o SHORT
	VARCHAR
DOPPIA PRECISIONE () FLOAT8	BIGINT o LONG
	CHAR
	DECIMAL (NUMERIC)

Dal tipo	Al tipo
	INTERO (INT)
	REALE () FLOAT4
	SMALLINT o SHORT
	VARCHAR
INTERO (INT)	BIGINT o LONG
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOPPIA PRECISIONE () FLOAT8
	REALE (FLOAT4)
	SMALLINT o SHORT
	VARCHAR
REALE () FLOAT4	BIGINT o LONG
	CHAR
	DECIMAL (NUMERIC)
	INTERO (INT)
	SMALLINT o SHORT
	VARCHAR
SMALLINT	BIGINT o LONG
	BOOLEAN

Dal tipo	Al tipo
	CHAR
	DECIMAL (NUMERIC)
	DOPPIA PRECISIONE () FLOAT8
	INTERO (INT)
	REALE () FLOAT4
	VARCHAR
TIME	VARCHAR
	TIMETZ

> **Note**
>
> Le conversioni implicite tra DATE, TIME, TIMESTAMP_LTZ, TIMESTAMP_NTZ o stringhe di caratteri utilizzano il fuso orario della sessione corrente.
> Il tipo di dati VARBYTE non può essere convertito implicitamente in nessun altro tipo di dati. Per ulteriori informazioni, consulta Funzione CAST.

AWS Clean Rooms Comandi SQL Spark

I seguenti comandi SQL sono supportati in AWS Clean Rooms Spark SQL:

Argomenti

- TABELLA CACHE
- Suggerimenti
- SELECT

TABELLA CACHE

Il comando CACHE TABLE memorizza nella cache i dati di una tabella esistente o crea e memorizza nella cache una nuova tabella contenente i risultati delle query.

> **Note**
>
> I dati memorizzati nella cache persistono per l'intera query.

La sintassi, gli argomenti e alcuni esempi provengono da Apache Spark SQL Reference.

Sintassi

Il comando CACHE TABLE supporta tre modelli di sintassi:

> **Colonne di output non consentite (query constraint e CACHE TABLE).**
>
> Il vincolo delle colonne di output non consentite nella regola di analisi personalizzata viene applicato alle tabelle memorizzate nella cache. Una tabella memorizzata nella cache non può fare riferimento a una colonna di output non consentita nella clausola SELECT. Per utilizzare una colonna con un vincolo di colonna di output non consentito in una parte successiva della query, converti la tabella memorizzata nella cache in un'espressione di tabella comune (CTE).

Con AS (senza parentesi): crea e memorizza nella cache una nuova tabella in base ai risultati della query.

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

Con AS e parentesi: funziona in modo simile alla prima sintassi ma utilizza le parentesi per raggruppare in modo esplicito la query.

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

Senza AS: memorizza nella cache una tabella esistente, utilizzando l'istruzione SELECT per filtrare le righe da memorizzare nella cache.

```
CACHE TABLE cache_table_identifier query;
```

Dove:

- Tutte le istruzioni devono terminare con un punto e virgola (;)
- `query`è in genere un'istruzione SELECT
- Le parentesi attorno alla query sono opzionali con AS
- La parola chiave AS è facoltativa

Parameters

cache_table_identifier

Il nome della tabella memorizzata nella cache. Può includere un qualificatore del nome del database opzionale.

AS

Una parola chiave utilizzata per creare e memorizzare nella cache una nuova tabella dai risultati delle query.

query

Un'istruzione SELECT o un'altra query che definisce i dati da memorizzare nella cache.

Esempi

Negli esempi seguenti, la tabella memorizzata nella cache persiste per l'intera query.
Dopo la memorizzazione nella cache, le query successive a cui fanno riferimento *`cache_table_identifier`* verranno lette a partire dalla versione memorizzata nella cache anziché essere ricalcolate o lette. *`sourceTable`* Ciò può migliorare le prestazioni delle query per i dati a cui si accede di frequente.

Crea e memorizza nella cache una tabella filtrata dai risultati delle query

Il primo esempio dimostra come creare e memorizzare nella cache una nuova tabella dai risultati delle query. Questo comando utilizza la `AS` parola chiave senza parentesi attorno all'istruzione. `SELECT` Crea una nuova tabella denominata '`cache_table_identifier`' contenente solo le righe da '' dove lo stato è `sourceTable` '. `active'` Esegue la query, memorizza i risultati nella nuova tabella e memorizza nella cache il contenuto della nuova tabella. Il '`sourceTable`' originale rimane

invariato e le query successive devono fare riferimento a 'cache_table_identifier' per utilizzare i dati memorizzati nella cache.

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

Memorizza i risultati delle query con istruzioni SELECT tra parentesi

Il secondo esempio dimostra come memorizzare nella cache i risultati di una query come nuova tabella con un nome specificato (cache_table_identifier), utilizzando parentesi attorno all'istruzione. SELECT Questo comando crea una nuova tabella denominata 'cache_table_identifier' contenente solo le righe da '' dove lo stato è sourceTable '. active' Esegue la query, archivia i risultati nella nuova tabella e memorizza nella cache il contenuto della nuova tabella. Il 'sourceTable' originale rimane invariato. Le query successive devono fare riferimento a 'cache_table_identifier' per utilizzare i dati memorizzati nella cache.

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

Memorizza nella cache una tabella esistente con condizioni di filtro

Il terzo esempio dimostra come inserire nella cache una tabella esistente utilizzando una sintassi diversa. Questa sintassi, che omette la parola chiave 'AS' e le parentesi, in genere memorizza nella cache le righe specificate da una tabella esistente denominata '' cache_table_identifier anziché creare una nuova tabella. L'SELECTistruzione funge da filtro per determinare quali righe memorizzare nella cache.

> **Note**
>
> Il comportamento esatto di questa sintassi varia tra i sistemi di database. Verifica sempre la sintassi corretta per il tuo servizio specifico AWS .

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

Suggerimenti

I suggerimenti per le analisi SQL forniscono direttive di ottimizzazione che guidano le strategie di esecuzione delle query AWS Clean Rooms, consentendoti di migliorare le prestazioni delle query e ridurre i costi di elaborazione. I suggerimenti suggeriscono come il motore di analisi Spark dovrebbe generare il suo piano di esecuzione.

Sintassi

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

I suggerimenti sono incorporati nelle query SQL utilizzando una sintassi in stile commento e devono essere inseriti direttamente dopo la parola chiave SELECT.

Tipi di suggerimenti supportati

AWS Clean Rooms supporta due categorie di suggerimenti: suggerimenti per il join e suggerimenti per il partizionamento.

Argomenti

- Join (suggerimenti)
- Suggerimenti per il partizionamento

Join (suggerimenti)

I suggerimenti di unione suggeriscono strategie di unione per l'esecuzione delle query. La sintassi, gli argomenti e alcuni esempi provengono da Apache Spark SQL Reference per ulteriori informazioni

TRASMISSIONE

Suggerisce di AWS Clean Rooms utilizzare broadcast join. La pagina di accesso con il suggerimento verrà trasmessa indipendentemente da autoBroadcastJoin Threshold. Se entrambe le parti del join hanno i suggerimenti per la trasmissione, verrà trasmessa quella con le dimensioni inferiori (in base alle statistiche).

Alias: BROADCASTJOIN, MAPJOIN

Parametri: identificatori di tabella (opzionali)

Esempi:

```
-- 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

Suggerisce l' AWS Clean Rooms uso di shuffle sort merge join.

Alias: SHUFFLE_MERGE, MERGEJOIN

Parametri: identificatori di tabella (opzionali)

Esempi:

```
-- 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

Suggerisce di utilizzare shuffle hash AWS Clean Rooms join. Se entrambe le parti hanno gli hash hint shuffle, l'ottimizzatore di query sceglie il lato più piccolo (in base alle statistiche) come lato di compilazione.

Parametri: identificatori di tabella (opzionali)

Esempi:

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

SHUFFLE_REPLICATE_NL

Suggerisce di utilizzare il nested loop join. AWS Clean Rooms shuffle-and-replicate

Parametri: identificatori di tabella (opzionali)

Esempi:

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

Suggerimenti per la risoluzione dei problemi in Spark SQL

La tabella seguente mostra scenari comuni in cui i suggerimenti non vengono applicati in SparkSQL. Per ulteriori informazioni, consulta the section called “Considerazioni e limitazioni”.

Caso d'uso	Esempio di query
Riferimento alla tabella non trovato	`SELECT /*+ BROADCAST(fake_table) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Tabella che non partecipa all'operazione di unione	`SELECT /*+ BROADCAST(s) */ * FROM students s WHERE s.age > 25;`
Riferimento alla tabella nella sottoquery annidata	`SELECT /*+ BROADCAST(s) */ * FROM employees e INNER JOIN (SELECT * FROM students s WHERE s.age > 20) sub ON e.eid = sub.sid;`
Nome della colonna anziché riferimento alla tabella	`SELECT /*+ BROADCAST(e.eid) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Suggerimento senza parametri obbligatori	`SELECT /*+ BROADCAST */ *`

Caso d'uso	Esempio di query
	`FROM employees e` `INNER JOIN students s ON e.eid = s.sid;`

Nome della tabella di base anziché alias della tabella	`SELECT /*+ BROADCAST(employees) */ *`
	`FROM employees e`
	`INNER JOIN students s ON e.eid = s.sid;`

Suggerimenti per il partizionamento

I suggerimenti per il partizionamento controllano la distribuzione dei dati tra i nodi esecutori. Quando vengono specificati più suggerimenti di partizionamento, più nodi vengono inseriti nel piano logico, ma il suggerimento più a sinistra viene selezionato dall'ottimizzatore.

COALESCE

Riduce il numero di partizioni al numero di partizioni specificato.

Parametri: Valore numerico (obbligatorio), deve essere un numero intero positivo compreso tra 1 e 2147483647

Esempi:

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

RIPARTIZIONAMENTO

Ripartiziona i dati nel numero specificato di partizioni utilizzando le espressioni di partizionamento specificate. Utilizza la distribuzione round-robin.

Parametri:

- Valore numerico (opzionale): numero di partizioni; deve essere un numero intero positivo compreso tra 1 e 2147483647
- Identificatori di colonna (facoltativi): colonne in base alle quali partizionare; queste colonne devono esistere nello schema di input.
- Se vengono specificati entrambi, il valore numerico deve essere al primo posto

Esempi:

```
-- 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;
```

RIPARTIZIONE_PER_INTERVALLO

Ripartiziona i dati nel numero specificato di partizioni utilizzando il partizionamento a intervalli sulle colonne specificate.

Parametri:

- Valore numerico (opzionale): numero di partizioni; deve essere un numero intero positivo compreso tra 1 e 2147483647
- Identificatori di colonna (facoltativi): colonne in base alle quali partizionare; queste colonne devono esistere nello schema di input.
- Se vengono specificati entrambi, il valore numerico deve essere al primo posto

Esempi:

```
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;
```

RIEQUILIBRARE

Riequilibra le partizioni di output dei risultati della query in modo che ogni partizione sia di dimensioni ragionevoli (né troppo piccola né troppo grande). Si tratta di un'operazione che richiede il massimo sforzo: se ci sono degli scostamenti, AWS Clean Rooms dividerà le partizioni inclinate per renderle non troppo grandi. Questo suggerimento è utile quando è necessario scrivere il risultato di una query su una tabella per evitare file troppo piccoli o troppo grandi.

Parametri:

- Valore numerico (opzionale): numero di partizioni; deve essere un numero intero positivo compreso tra 1 e 2147483647
- Identificatori di colonna (facoltativi): le colonne devono apparire nell'elenco di output SELECT
- Se vengono specificati entrambi, il valore numerico deve essere al primo posto

Esempi:

```
-- 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;
```

Combinazione di più suggerimenti

È possibile specificare più suggerimenti in una singola query separandoli con virgole:

```
-- 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;
```

Considerazioni e limitazioni

- I suggerimenti sono suggerimenti di ottimizzazione, non comandi. L'ottimizzatore delle query può ignorare i suggerimenti basati su vincoli di risorse o condizioni di esecuzione.
- I suggerimenti sono incorporati direttamente nelle stringhe di query SQL per entrambi e. CreateAnalysisTemplate StartProtectedQuery APIs
- I suggerimenti devono essere inseriti direttamente dopo la parola chiave SELECT.
- I parametri denominati non sono supportati con i suggerimenti e genereranno un'eccezione.
- I nomi delle colonne nei suggerimenti REPARTITION e REPARTITION_BY_RANGE devono esistere nello schema di input.
- I nomi delle colonne nei suggerimenti REBALANCE devono apparire nell'elenco di output SELECT.
- I parametri numerici devono essere numeri interi positivi compresi tra 1 e 2147483647. Le notazioni scientifiche come 1e1 non sono supportate
- I suggerimenti non sono supportati nelle query SQL sulla privacy differenziale.
- I suggerimenti per le query SQL non sono supportati nei job. PySpark Per fornire direttive per i piani di esecuzione in un PySpark job, utilizza l'API Data Frame. Per ulteriori informazioni, consulta la documentazione sull' DataFrame API Apache Spark.

SELECT

Il comando SELECT restituisce righe da tabelle e funzioni definite dall'utente.

I seguenti comandi, clausole e operatori di set SELECT SQL sono supportati in AWS Clean Rooms Spark SQL:

Argomenti

La sintassi, gli argomenti e alcuni esempi provengono da Apache Spark SQL Reference.

SELECT list

I SELECT list nomi delle colonne, delle funzioni e delle espressioni che vuoi che la query restituisca. L'elenco rappresenta l'output della query.

Sintassi

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

Parameters

DISTINCT

Opzione che elimina le righe duplicate dal set di risultati, in base ai valori corrispondenti in una o più colonne.

expression

Espressione formata da una o più colonne presenti nelle tabelle a cui fa riferimento la query. Un'espressione può contenere funzioni SQL. Esempio:

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

AS column_alias

Nome temporaneo per la colonna che viene utilizzata nel set di risultati finale. La AS parola chiave è facoltativa. Esempio:

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

Se non specifichi un alias per un'espressione che non è un semplice nome di colonna, il set di risultati applica un nome predefinito a quella colonna.

> **Note**
>
> L'alias viene riconosciuto subito dopo essere stato definito nell'elenco di destinazione. Non è possibile utilizzare un alias in altre espressioni definite successivamente nello stesso elenco di destinazione.

Clausola WITH

Una clausola WITH è una clausola facoltativa che precede l'elenco SELECT in una query. La clausola WITH definisce uno o più common_table_expression. Ogni espressione comune di tabella (CTE) definisce una tabella temporanea, che è simile a una definizione di vista. È possibile fare riferimento a queste tabelle temporanee nella clausola FROM. Vengono utilizzati solo durante l'esecuzione della query a cui appartengono. Ogni CTE nella clausola WITH specifica un nome di tabella, un elenco facoltativo di nomi di colonna e un'espressione di query che restituisce una tabella (un'istruzione SELECT).

Le sottoquery della clausola WITH sono un modo efficace per definire le tabelle che possono essere utilizzate durante l'esecuzione di una singola query. In ogni caso, è possibile ottenere gli stessi risultati utilizzando le sottoquery nel corpo principale dell'istruzione SELECT, ma le sottoquery della clausola WITH potrebbero essere più semplici da scrivere e leggere. Ove possibile, le sottoquery della clausola WITH che sono referenziate più volte sono ottimizzate come sottoespressioni comuni; vale a dire, potrebbe essere possibile valutare una sottoquery WITH una volta e riutilizzarne i risultati. Tieni presente che le sottoespressioni comuni non sono limitate a quelle definite nella clausola WITH.

Sintassi

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

dove common_table_expression può essere non ricorsivo. Di seguito è riportata la forma non ricorsiva:

```
CTE_table_name AS ( query )
```

Parameters

common_table_expression

Definisce una tabella temporanea a cui è possibile fare riferimento nella Clausola FROM e viene utilizzato solo durante l'esecuzione della query a cui appartiene.

CTE_table_name

Nome univoco per una tabella temporanea che definisce i risultati di una sottoquery della clausola WITH. Non puoi utilizzare nomi duplicati in una singola clausola WITH. A ogni sottoquery deve essere assegnato un nome di tabella a cui è possibile fare riferimento nella Clausola FROM.

query

Qualsiasi query SELECT AWS Clean Rooms che supporti. Per informazioni, consulta SELECT.

Note per l'utilizzo

È possibile utilizzare una clausola WITH nella seguente istruzione SQL:

- SELECT, WITH, UNION, UNION ALL, INTERSECT, INTERSECT ALL, EXCEPT o EXCEPT ALL

Se la clausola FROM di una query che contiene una clausola WITH non fa riferimento a nessuna delle tabelle definite dalla clausola WITH, la clausola WITH viene ignorata e la query viene eseguita normalmente.

A una tabella definita da una sottoquery della clausola WITH è possibile fare riferimento solo nell'ambito della query SELECT avviata dalla clausola WITH. Ad esempio, puoi fare riferimento a una tabella di questo tipo nella clausola FROM di una sottoquery nell'elenco SELECT, nella clausola WHERE o nella clausola HAVING. Non puoi utilizzare una clausola WITH in una sottoquery e fare riferimento alla tabella nella clausola FROM della query principale o un'altra sottoquery. Questo

modello di query genera un messaggio di errore nel formato `relation table_name doesn't exist` per la tabella della clausola WITH.

Non puoi specificare un'altra clausola WITH all'interno di una sottoquery della clausola WITH.

Non puoi creare riferimenti alle tabelle definite dalle sottoquery della clausola WITH. Ad esempio, la seguente query restituisce un errore a causa del riferimento in avanti alla tabella W2 nella definizione della tabella W1:

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

Esempi

L'esempio seguente mostra il caso più semplice possibile di una query che contiene una clausola WITH. La query WITH denominata VENUECOPY seleziona tutte le righe dalla tabella VENUE. La query principale a sua volta seleziona tutte le righe da VENUECOPY. La tabella VENUECOPY esiste solo per la durata di questa query.

```
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)
```

L'esempio seguente mostra una clausola WITH che produce due tabelle, denominate VENUE_SALES e TOP_VENUES. La seconda tabella della query WITH seleziona dalla prima. A sua volta, la clausola WHERE del blocco di query principale contiene una sottoquery che vincola la tabella 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)
```

I seguenti due esempi illustrano le regole per l'ambito dei riferimenti di tabella basati sulle sottoquery della clausola WITH. La prima query viene eseguita, ma la seconda non riesce con un errore previsto. La prima query contiene la sottoquery clausola WITH all'interno dell'elenco SELECT della

query principale. Alla tabella definita dalla clausola WITH (HOLIDAYS) si fa riferimento nella clausola FROM della sottoquery nell'elenco SELECT:

```
select caldate, sum(pricepaid) as daysales,
(with holidays as (select * from date where holiday ='t')
select sum(pricepaid)
from sales join holidays on sales.dateid=holidays.dateid
where caldate='2008-12-25') as dec25sales
from sales join date on sales.dateid=date.dateid
where caldate in('2008-12-25','2008-12-31')
group by caldate
order by caldate;

caldate   | daysales | dec25sales
-----------+----------+------------
2008-12-25 | 70402.00 |   70402.00
2008-12-31 | 12678.00 |   70402.00
(2 rows)
```

La seconda query non riesce perché tenta di fare riferimento alla tabella HOLIDAYS nella query principale e nella sottoquery elenco SELECT. I riferimenti della query principale sono fuori ambito.

```
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
```

Clausola FROM

La clausola FROM in una query elenca i riferimenti di tabella (tabelle, viste e sottoquery) da cui vengono selezionati i dati. Se sono elencati più riferimenti tabella, è necessario unire le tabelle, utilizzando la sintassi appropriata nella clausola FROM o nella clausola WHERE. Se non vengono specificati criteri di join, il sistema elabora la query come cross-join (prodotto cartesiano).

Argomenti

Sintassi

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

dove table_reference è una delle opzioni seguenti:

```
with_subquery_table_name | table_name | ( subquery ) [ [ AS ] alias ]
table_reference [ NATURAL ] join_type table_reference [ USING ( join_column [, ...] ) ]
table_reference [ INNER ] join_type table_reference ON expr
```

Parameters

with_subquery_table_name

Tabella definita da una sottoquery nella Clausola WITH.

table_name

Nome di una tabella o vista.

alias

Nome alternativo temporaneo per una tabella o vista. L'alias è obbligatorio per una tabella derivata da una sottoquery. In altri riferimenti di tabella, gli alias sono facoltativi. La AS parola chiave è sempre facoltativa. Gli alias di tabella forniscono una comoda scelta rapida per identificare le tabelle in altre parti di una query, come nella clausola WHERE.

Esempio:

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

Se si definisce un alias di tabella, è necessario utilizzare l'alias per fare riferimento a quella tabella nella query.

Ad esempio, se la query è`SELECT "tbl"."col" FROM "tbl" AS "t"`, la query fallirebbe perché ora il nome della tabella viene sostanzialmente sovrascritto. Una query valida in questo caso sarebbe. `SELECT "t"."col" FROM "tbl" AS "t"`

column_alias

Un'espressione semplice che restituisce un valore.

subquery

Espressione della query che restituisce una tabella. La tabella esiste solo per la durata della query e in genere le viene assegnato un nome o un alias; tuttavia l'alias non è obbligatorio. Puoi anche definire i nomi delle colonne per le tabelle che derivano da sottoquery. L'assegnazione degli alias alle colonne è importante quando vuoi unire i risultati delle sottoquery ad altre tabelle e quando vuoi selezionare o vincolare tali colonne altrove nella query.

Una sottoquery può contenere una clausola ORDER BY, ma questa potrebbe non avere alcun effetto se non viene specificata una clausola LIMIT o OFFSET.

NATURAL

Definisce un join che utilizza automaticamente tutte le coppie di colonne con lo stesso nome nelle due tabelle come colonne di unione. La condizione di join esplicita non è obbligatoria. Ad esempio, se le tabelle CATEGORY ed EVENT hanno entrambe le colonne denominate CATID, un join naturale di tali tabelle è un join sulle rispettive colonne CATID.

> **Note**
>
> Se viene specificato un join NATURAL ma non esistono coppie di colonne con nome identico nelle tabelle da unire, la query viene impostata automaticamente su un cross-join.

join_type

Specifica uno dei seguenti tipi di join:

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

I cross-join sono join non qualificati; restituiscono il prodotto cartesiano delle due tabelle.

I join inner e outer sono join qualificati. Sono qualificati in modo implicito (in join naturali) con la sintassi ON o USING nella clausola FROM o con una condizione della clausola WHERE.

Un inner join restituisce solo le righe corrispondenti, in base alla condizione di join o all'elenco delle colonne di join. Un outer join restituisce tutte le righe che l'inner join equivalente restituirebbe, più le righe non corrispondenti dalla tabella "left", dalla tabella "right" o da entrambe le tabelle. La tabella di sinistra è la tabella elencata per prima e la tabella di destra è la seconda tabella nell'elenco. Le righe non corrispondenti contengono valori NULL per riempire gli spazi vuoti nelle colonne di output.

ON join_condition

Tipo di specifica del join in cui le colonne di unione vengono dichiarate come una condizione che segue la parola chiave ON. Ad esempio:

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

USING (join_column [, ...])

Tipo di specifica del join in cui le colonne di unione vengono elencate tra parentesi. Se vengono specificate più colonne di unione, queste sono delimitate da virgole. La parola chiave USING deve precedere l'elenco. Ad esempio:

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

Note per l'utilizzo

Le colonne di unione devono avere tipi di dati comparabili.

Un join NATURAL o USING conserva solo una di ciascuna coppia di colonne di unione nel set di risultati intermedi.

Un join con la sintassi ON mantiene entrambe le colonne di unione nel set di risultati intermedi.

consultare anche Clausola WITH.

Clausola JOIN

Una clausola SQL JOIN viene utilizzata per combinare i dati di due o più tabelle in base a campi comuni. I risultati potrebbero cambiare o meno a seconda del metodo di join specificato. Gli outer join sinistro e destro mantengono i valori da una delle tabelle unite quando non viene trovata alcuna corrispondenza nell'altra tabella.

La combinazione del tipo JOIN e della condizione di join determina quali righe vengono incluse nel set di risultati finale. Le clausole SELECT e WHERE controllano quindi quali colonne vengono restituite e come vengono filtrate le righe. Comprendere i diversi tipi di JOIN e come utilizzarli in modo efficace è un'abilità fondamentale in SQL, perché consente di combinare i dati di più tabelle in modo flessibile e potente.

Sintassi

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

Parameters

SELEZIONA colonna1, colonna2,..., colonnaN

Le colonne da includere nel set di risultati. È possibile selezionare le colonne da una o entrambe le tabelle coinvolte nel JOIN.

DALLA tabella 1

La prima tabella (a sinistra) dell'operazione JOIN.

[JOIN | INNER JOIN | LEFT [OUTER] JOIN | RIGHT [OUTER] JOIN | [OUTER] JOIN COMPLETO] table2:

Il tipo di JOIN da eseguire. JOIN o INNER JOIN restituisce solo le righe con valori corrispondenti in entrambe le tabelle.

LEFT [OUTER] JOIN restituisce tutte le righe della tabella di sinistra, con le righe corrispondenti della tabella di destra.

RIGHT [OUTER] JOIN restituisce tutte le righe della tabella di destra, con le righe corrispondenti della tabella di sinistra.

FULL [OUTER] JOIN restituisce tutte le righe di entrambe le tabelle, indipendentemente dal fatto che esista una corrispondenza o meno.

CROSS JOIN crea un prodotto cartesiano delle righe delle due tabelle.

ON table1.column = table2.column

La condizione di unione, che specifica come vengono abbinate le righe nelle due tabelle. La condizione di unione può essere basata su una o più colonne.

Condizione WHERE:

Una clausola facoltativa che può essere utilizzata per filtrare ulteriormente il set di risultati, in base a una condizione specificata.

Esempio

Di seguito è riportato un esempio di join tra due tabelle con la clausola USING. In questo caso, le colonne listid e eventid vengono utilizzate come colonne di join. I risultati sono limitati a cinque righe.

```
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
```

Tipi di join

INNER

Questo è il tipo di join predefinito. Restituisce le righe con valori corrispondenti in entrambi i riferimenti alla tabella.

L'INNER JOIN è il tipo di join più comune utilizzato in SQL. È un modo efficace per combinare i dati di più tabelle in base a una colonna o un set di colonne comune.

Sintassi:

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

La seguente query restituirà tutte le righe in cui è presente un valore customer_id corrispondente tra le tabelle clienti e ordini. Il set di risultati conterrà le colonne customer_id, name, order_id e order_date.

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

La seguente query è un inner join (senza la parola chiave JOIN) tra la tabella LISTING e la tabella SALES, in cui il LISTID della tabella LISTING è compreso tra 1 e 5. Questa query corrisponde ai valori della colonna LISTID nella tabella LISTING (la tabella di sinistra) e SALES (la tabella di destra). I risultati mostrano che LISTID 1, 4 e 5 corrispondono ai criteri.

```
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
```

Di seguito è riportato un esempio di inner join con la clausola ON. In questo caso, le righe NULL non vengono restituite.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from sales join listing
on sales.listid=listing.listid and sales.eventid=listing.eventid
where listing.listid between 1 and 5
```

```
group by 1
order by 1;

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

La seguente query è un inner join di due sottoquery della clausola FROM. La query trova il numero di biglietti venduti e invenduti per diverse categorie di eventi (concerti e spettacoli). Queste sottoquery della clausola FROM sono sottoquery table e possono restituire più colonne e righe.

```
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
```

SINISTRA [ESTERNO]

Restituisce tutti i valori dal riferimento alla tabella sinistra e i valori corrispondenti dal riferimento alla tabella destra oppure aggiunge NULL se non c'è corrispondenza. Viene anche chiamato left outer join.

Restituisce tutte le righe della tabella sinistra (prima) e le righe corrispondenti della tabella destra (seconda). Se non c'è alcuna corrispondenza nella tabella di destra, il set di risultati conterrà valori

NULL per le colonne della tabella di destra. La parola chiave OUTER può essere omessa e il join può essere scritto semplicemente come LEFT JOIN. L'opposto di un LEFT OUTER JOIN è un RIGHT OUTER JOIN, che restituisce tutte le righe della tabella di destra e le righe corrispondenti della tabella di sinistra.

Sintassi:

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

La seguente query restituirà tutte le righe della tabella clienti, insieme alle righe corrispondenti della tabella ordini. Se un cliente non ha ordini, il set di risultati includerà comunque le informazioni del cliente, con valori NULL per le colonne order_id e order_date.

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

La seguente query è un outer join. Gli outer join sinistro e destro mantengono i valori da una delle tabelle unite quando non viene trovata alcuna corrispondenza nell'altra tabella. Le tabelle sinistra e destra sono la prima e la seconda tabella elencate nella sintassi. I valori NULL vengono utilizzati per riempire gli "spazi vuoti" nel set di risultati. Questa query corrisponde ai valori della colonna LISTID nella tabella LISTING (la tabella di sinistra) e SALES (la tabella di destra). I risultati mostrano che LISTIDs 2 e 3 non hanno portato ad alcuna vendita.

```
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
```

DESTRA [ESTERNO]

Restituisce tutti i valori dal riferimento alla tabella destra e i valori corrispondenti dal riferimento alla tabella sinistra oppure aggiunge NULL se non c'è corrispondenza. Viene anche chiamato right outer join.

Restituisce tutte le righe della tabella destra (seconda) e le righe corrispondenti della tabella sinistra (prima). Se non c'è alcuna corrispondenza nella tabella a sinistra, il set di risultati conterrà valori NULL per le colonne della tabella di sinistra. La parola chiave OUTER può essere omessa e il join può essere scritto semplicemente come RIGHT JOIN. L'opposto di un RIGHT OUTER JOIN è un LEFT OUTER JOIN, che restituisce tutte le righe della tabella di sinistra e le righe corrispondenti della tabella di destra.

Sintassi:

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

La seguente query restituirà tutte le righe della tabella clienti, insieme alle righe corrispondenti della tabella ordini. Se un cliente non ha ordini, il set di risultati includerà comunque le informazioni del cliente, con valori NULL per le colonne order_id e order_date.

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

La seguente query è un outer join. Questa query corrisponde ai valori della colonna LISTID nella tabella LISTING (la tabella di sinistra) e SALES (la tabella di destra). I risultati mostrano che LISTIDs 1, 4 e 5 corrispondono ai criteri.

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

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

COMPLETO [ESTERNO]

Restituisce tutti i valori di entrambe le relazioni, aggiungendo valori NULL sul lato che non corrisponde. Viene anche chiamato join esterno completo.

Restituisce tutte le righe delle tabelle sinistra e destra, indipendentemente dal fatto che esista una corrispondenza o meno. Se non c'è alcuna corrispondenza, il set di risultati conterrà valori NULL per le colonne della tabella che non hanno una riga corrispondente. La parola chiave OUTER può essere omessa e il join può essere scritto semplicemente come FULL JOIN. Il FULL OUTER JOIN è usato meno comunemente rispetto al LEFT OUTER JOIN o al RIGHT OUTER JOIN, ma può essere utile in alcuni scenari in cui è necessario visualizzare tutti i dati di entrambe le tabelle, anche se non ci sono corrispondenze.

Sintassi:

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

La seguente query restituirà tutte le righe delle tabelle clienti e ordini. Se un cliente non ha ordini, il set di risultati includerà comunque le informazioni del cliente, con valori NULL per le colonne order_id e order_date. Se a un ordine non è associato alcun cliente, il set di risultati includerà quell'ordine, con valori NULL per le colonne customer_id e name.

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

La seguente query è un fullr join. I full join mantengono i valori da una delle tabelle unite quando non viene trovata alcuna corrispondenza nell'altra tabella. Le tabelle sinistra e destra sono la prima e la seconda tabella elencate nella sintassi. I valori NULL vengono utilizzati per riempire gli "spazi vuoti"

nel set di risultati. Questa query corrisponde ai valori della colonna LISTID nella tabella LISTING (la tabella di sinistra) e SALES (la tabella di destra). I risultati mostrano che LISTIDs 2 e 3 non hanno portato ad alcuna vendita.

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

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

La seguente query è un full join. Questa query corrisponde ai valori della colonna LISTID nella tabella LISTING (la tabella di sinistra) e SALES (la tabella di destra). Nei risultati vengono visualizzate solo le righe che non generano vendite (LISTIDs 2 e 3).

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

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

[SINISTRA] SEMI

Restituisce i valori dal lato sinistro del riferimento alla tabella che corrisponde a quello destro. Viene anche chiamato left semi join.

Restituisce solo le righe della tabella sinistra (prima) che hanno una riga corrispondente nella tabella destra (seconda). Non restituisce alcuna colonna della tabella di destra, ma solo le colonne della

tabella di sinistra. Il LEFT SEMI JOIN è utile quando si desidera trovare le righe di una tabella che hanno una corrispondenza in un'altra tabella, senza dover restituire alcun dato dalla seconda tabella. Il LEFT SEMI JOIN è un'alternativa più efficiente all'utilizzo di una sottoquery con una clausola IN o EXISTS.

Sintassi:

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

La seguente query restituirà solo le colonne customer_id e name della tabella customers, per i clienti che hanno almeno un ordine nella tabella ordini. Il set di risultati non includerà alcuna colonna della tabella degli ordini.

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

CROSS JOIN

Restituisce il prodotto cartesiano di due relazioni. Ciò significa che il set di risultati conterrà tutte le possibili combinazioni di righe delle due tabelle, senza applicare alcuna condizione o filtro.

Il CROSS JOIN è utile quando è necessario generare tutte le possibili combinazioni di dati da due tabelle, ad esempio nel caso di creazione di un report che mostri tutte le possibili combinazioni di informazioni sui clienti e sui prodotti. Il CROSS JOIN è diverso dagli altri tipi di join (INNER JOIN, LEFT JOIN, ecc.) perché non ha una condizione di join nella clausola ON. La condizione di unione non è richiesta per un CROSS JOIN.

Sintassi:

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

La seguente query restituirà un set di risultati che contiene tutte le possibili combinazioni di customer_id, customer_name, product_id e product_name dalle tabelle clienti e prodotti. Se la tabella

clienti ha 10 righe e la tabella prodotti ha 20 righe, il set di risultati di CROSS JOIN conterrà 10 x 20 = 200 righe.

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

La seguente query è un cross join o un join cartesiano della tabella LISTING e della tabella SALES con un predicato per limitare i risultati. Questa query corrisponde ai valori delle colonne LISTID nella tabella SALES e nella tabella LISTING per LISTIDs 1, 2, 3, 4 e 5 in entrambe le tabelle. I risultati mostrano che 20 righe soddisfano i criteri.

```
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 JOIN

Restituisce i valori del riferimento alla tabella sinistra che non corrispondono al riferimento alla tabella destra. Viene anche chiamato antijoin sinistro.

L'ANTI JOIN è un'operazione utile quando si desidera trovare le righe di una tabella che non hanno una corrispondenza in un'altra tabella.

Sintassi:

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

La seguente query restituirà tutti i clienti che non hanno effettuato ordini.

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

NATURAL

Speciifica che le righe delle due relazioni verranno associate implicitamente in termini di uguaglianza per tutte le colonne con nomi corrispondenti.

Abbina automaticamente le colonne con lo stesso nome e tipo di dati tra le due tabelle. Non richiede di specificare esplicitamente la condizione di unione nella clausola ON. Combina tutte le colonne corrispondenti tra le due tabelle nel set di risultati.

NATURAL JOIN è una comoda abbreviazione quando le tabelle che stai unendo hanno colonne con gli stessi nomi e tipi di dati. Tuttavia, in genere si consiglia di utilizzare il più esplicito INNER JOIN... Sintassi ON per rendere le condizioni di unione più esplicite e facili da capire.

Sintassi:

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

L'esempio seguente è un'unione naturale tra due tabelle `employees` e`departments`, con le seguenti colonne:

- `employees`tabella: `employee_id``first_name`,`last_name`, `department_id`
- `departments`tavolo:`department_id`, `department_name`

La seguente query restituirà un set di risultati che include il nome, il cognome e il nome del reparto per tutte le righe corrispondenti tra le due tabelle, in base alla `department_id` colonna.

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

Di seguito è riportato un esempio di join naturale tra due tabelle. In questo caso, le colonne listid, sellerid, eventid e dateid hanno nomi e tipi di dati identici in entrambe le tabelle e quindi vengono utilizzate come colonne di join. I risultati sono limitati a cinque righe.

```
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
```

Clausola WHERE

La clausola WHERE contiene le condizioni che possono unire tabelle o applicare predicati alle colonne nelle tabelle. Le tabelle possono inner join utilizzando la sintassi appropriata nella clausola WHERE o nella clausola FROM. I criteri degli outer join devono essere specificati nella clausola FROM.

Sintassi

```
[ WHERE condition ]
```

condizione

Qualsiasi condizione di ricerca con un risultato booleano, ad esempio una condizione di join o un predicato su una colonna della tabella. I seguenti esempi sono condizioni di join valide:

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

I seguenti esempi sono condizioni valide sulle colonne delle tabelle:

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

Le condizioni possono essere semplici o complesse; per le condizioni complesse, puoi utilizzare le parentesi per isolare le unità logiche. Nell'esempio seguente, la condizione di join è racchiusa tra parentesi.

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

Note per l'utilizzo

Puoi utilizzare gli alias nella clausola WHERE per fare riferimento alle espressioni di elenco selezionate.

Non puoi limitare i risultati delle funzioni di aggregazione nella clausola WHERE; utilizza la clausola HAVING per questo scopo.

Le colonne che sono limitate nella clausola WHERE devono derivare dai riferimenti di tabella nella clausola FROM.

Esempio

La seguente query utilizza una combinazione di diverse restrizioni della clausola WHERE, inclusa una condizione di join per le tabelle SALES ed EVENT, un predicato sulla colonna EVENTNAME e due predicati sulla colonna 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)
```

clausola VALUES

La clausola VALUES viene utilizzata per fornire un set di valori di riga direttamente nella query, senza la necessità di fare riferimento a una tabella.

La clausola VALUES può essere utilizzata nei seguenti scenari:

- È possibile utilizzare la clausola VALUES in un'istruzione INSERT INTO per specificare i valori per le nuove righe da inserire in una tabella.
- È possibile utilizzare la clausola VALUES da sola per creare un set di risultati temporaneo o una tabella in linea, senza la necessità di fare riferimento a una tabella.
- È possibile combinare la clausola VALUES con altre clausole SQL, ad esempio WHERE, ORDER BY o LIMIT, per filtrare, ordinare o limitare le righe del set di risultati.

Questa clausola è particolarmente utile quando è necessario inserire, interrogare o manipolare un piccolo set di dati direttamente nell'istruzione SQL, senza la necessità di creare o fare riferimento a una tabella permanente. Consente di definire i nomi delle colonne e i valori corrispondenti per ogni riga, offrendovi la flessibilità necessaria per creare set di risultati temporanei o inserire dati all'istante, senza il sovraccarico dovuto alla gestione di una tabella separata.

Sintassi

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

Parameters

espressione

Un'espressione che specifica una combinazione di uno o più valori, operatori e funzioni SQL che restituisce un valore.

table_alias

Un alias che specifica un nome temporaneo con un elenco di nomi di colonna opzionale.

Esempio

L'esempio seguente crea una tabella in linea, un set di risultati temporaneo simile a una tabella con due colonne e. `col1` `col2` La singola riga del set di risultati contiene i valori `"one"` e`1`, rispettivamente. La `SELECT * FROM` parte della query recupera semplicemente tutte le colonne e le righe da questo set di risultati temporaneo. I nomi delle colonne (`col1`and`col2`) vengono generati automaticamente dal sistema di database, poiché la clausola VALUES non specifica esplicitamente i nomi delle colonne.

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

Se desideri definire nomi di colonna personalizzati, puoi farlo utilizzando una clausola AS dopo la clausola VALUES, in questo modo:

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

Ciò creerebbe un set di risultati temporaneo con i nomi delle colonne `name` `eid`, invece del valore predefinito `col1` e. `col2`

Clausola GROUP BY

La clausola GROUP BY identifica le colonne di raggruppamento per la query. Le colonne di raggruppamento devono essere dichiarate quando la query calcola gli aggregati con le funzioni standard, ad esempio SUM, AVG e COUNT. Se nell'espressione SELECT è presente una funzione di aggregazione, qualsiasi colonna dell'espressione SELECT che non si trova in una funzione aggregata deve essere inclusa nella clausola GROUP BY.

Per ulteriori informazioni, consulta AWS Clean Rooms Funzioni Spark SQL.

Sintassi

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

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

Parametri

expr

L'elenco di colonne o espressioni deve corrispondere all'elenco di espressioni non aggregate dell'elenco di selezione della query. A titolo illustrativo, prendi in considerazione la query semplice riportata di seguito.

```
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)
```

In questa query, l'elenco di selezione è composto da due espressioni di aggregazione. La prima utilizza la funzione SUM e la seconda utilizza la funzione COUNT. Le restanti due colonne, LISTID ed EVENTID, devono essere dichiarate come colonne di raggruppamento.

Le espressioni nella clausola GROUP BY possono anche fare riferimento all'elenco di selezione usando numeri ordinali. A titolo illustrativo, l'esempio precedente potrebbe essere abbreviato come segue.

```
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

È possibile utilizzare l'estensione di aggregazione ROLLUP per eseguire il lavoro di più operazioni GROUP BY in un'unica istruzione. Per ulteriori informazioni sulle estensioni di aggregazione e sulle funzioni correlate, consulta Estensioni di aggregazione.

Estensioni di aggregazione

AWS Clean Rooms supporta le estensioni di aggregazione per eseguire il lavoro di più operazioni GROUP BY in un'unica istruzione.

GROUPING SETS

Calcola uno o più set di raggruppamento in una singola istruzione. Un set di raggruppamento è l'insieme di una singola clausola GROUP BY, un set di 0 o più colonne in base al quale è possibile raggruppare il set di risultati di una query. GROUP BY GROUPING SETS equivale all'esecuzione di una query UNION ALL su un set di risultati raggruppato in colonne diverse. Ad esempio, GROUP BY GROUP SETS((a), (b)) è equivalente a GROUP BY a UNION ALL GROUP BY b.

L'esempio seguente restituisce il costo dei prodotti nella tabella degli ordini raggruppati in base alle categorie dei prodotti e al tipo di prodotti venduti.

```
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

Presuppone una gerarchia in cui le colonne precedenti sono considerate le colonne padri delle colonne successive. ROLLUP raggruppa i dati in base alle colonne fornite, restituendo righe di subtotali aggiuntive che rappresentano i totali in tutti i livelli di colonne di raggruppamento, oltre alle righe raggruppate. Ad esempio, puoi utilizzare GROUP BY ROLLUP((a), (b)) per restituire un set di risultati raggruppato prima per a, poi per b supponendo che b sia una sottosezione di a. ROLLUP restituisce anche una riga con l'intero set di risultati senza colonne di raggruppamento.

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

L'esempio seguente restituisce il costo dei prodotti nella tabella degli ordini raggruppati prima per categoria e poi per prodotto, con product come sezione della categoria.

```
SELECT category, product, sum(cost) as total
```

```
FROM orders
GROUP BY ROLLUP(category, product) ORDER BY 1,2;

       category       |       product        | total
----------------------+----------------------+-------
 cellphones           | smartphone           |  1610
 cellphones           |                      |  1610
 computers            | laptop               |  2050
 computers            | mouse                |    50
 computers            |                      |  2100
                      |                      |  3710
(6 rows)
```

CUBE

Raggruppa i dati in base alle colonne fornite, restituendo righe di subtotali aggiuntive che rappresentano i totali in tutti i livelli di colonne di raggruppamento, oltre alle righe raggruppate. CUBE restituisce le stesse righe di ROLLUP, ma aggiunge ulteriori righe di subtotali per ogni combinazione di colonne di raggruppamento non previste da ROLLUP. Ad esempio, è possibile utilizzare GROUP BY CUBE((a), (b)) per restituire un set di risultati raggruppato prima per a, poi per b, supponendo che b sia una sottosezione di a, quindi solo per b. CUBE restituisce anche una riga con l'intero set di risultati senza colonne di raggruppamento.

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

L'esempio seguente restituisce il costo dei prodotti nella tabella degli ordini raggruppati prima per categoria e poi per prodotto, con product come sezione della categoria. A differenza dell'esempio precedente per ROLLUP, l'istruzione restituisce risultati per ogni combinazione di colonne di raggruppamento.

```
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)
```

Clausola HAVING

La clausola HAVING applica una condizione al set di risultati raggruppati intermedi restituiti da una query.

Sintassi

```
[ HAVING condition ]
```

Ad esempio, puoi limitare i risultati di una funzione SUM:

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

La condizione HAVING viene applicata dopo che tutte le condizioni della clausola WHERE sono state applicate e le operazioni GROUP BY sono state completate.

La condizione stessa assume lo stesso formato di qualsiasi condizione della clausola WHERE.

Note per l'utilizzo

- Qualsiasi colonna a cui viene fatto riferimento in una condizione della clausola HAVING deve essere una colonna di raggruppamento o una colonna che fa riferimento al risultato di una funzione di aggregazione.
- In una clausola HAVING, non è possibile specificare:
  - Un numero ordinale che fa riferimento a una voce di elenco selezionata. Solo le clausole GROUP BY e ORDER BY accettano numeri ordinali.

Esempi

La seguente query calcola le vendite totali dei biglietti per tutti gli eventi in base al nome, quindi elimina gli eventi in cui le vendite totali erano inferiori a $800.000. La condizione HAVING viene applicata ai risultati della funzione di aggregazione nell'elenco di selezione: `sum(pricepaid)`.

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
```

```
group by 1
having sum(pricepaid) > 800000
order by 2 desc, 1;

eventname     |    sum
------------------+-----------
Mamma Mia!       | 1135454.00
Spring Awakening |  972855.00
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
(6 rows)
```

La seguente query calcola un set di risultati simile. In questo caso, tuttavia, la condizione HAVING viene applicata a un'aggregazione che non è specificata nell'elenco di selezione: `sum(qtysold)`. Gli eventi che non hanno venduto più di 2.000 biglietti sono stati eliminati dal risultato finale.

```
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)
```

Operatori su set

Gli operatori set vengono utilizzati per confrontare e unire i risultati di due espressioni di query separate.

AWS Clean Rooms Spark SQL supporta i seguenti operatori di set elencati nella tabella seguente.

Imposta operatore
INTERSECT
INTERSECA TUTTO
EXCEPT
TRANNE TUTTI
UNION
UNION ALL

Ad esempio, se vuoi sapere quali utenti di un sito web sono sia acquirenti che venditori ma i loro nomi utente sono memorizzati in colonne o tabelle separate, puoi trovare l'intersezione di questi due tipi di utenti. Se vuoi sapere quali utenti del sito sono acquirenti ma non venditori, puoi utilizzare l'operatore EXCEPT per trovare la difference tra i due elenchi di utenti. Se vuoi creare l'elenco di tutti gli utenti, indipendentemente dal ruolo, puoi utilizzare l'operatore UNION.

> **Note**
>
> Le clausole ORDER BY, LIMIT, SELECT TOP e OFFSET non possono essere utilizzate nelle espressioni di query unite dagli operatori di set UNION, UNION ALL, INTERSECT e EXCEPT.

Argomenti

Sintassi

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

Parameters

subquery1, subquery2

Un'espressione di query che corrisponde, sotto forma di elenco di selezione, a una seconda espressione di query che segue l'operatore UNION, UNION ALL, INTERSECT, INTERSECT ALL, EXCEPT o EXCEPT ALL. Le due espressioni devono contenere lo stesso numero di colonne di output con tipi di dati compatibili; in caso contrario, i due set di risultati non possono essere confrontati e uniti. Le operazioni di set non consentono la conversione implicita tra diverse categorie di tipi di dati. Per ulteriori informazioni, consulta Conversione e compatibilità dei tipi.

Puoi creare query contenenti un numero illimitato di espressioni di query e collegarle agli operatori UNION, INTERSECT ed EXCEPT in qualsiasi combinazione. Ad esempio, la seguente struttura di query è valida, assumendo che le tabelle T1, T2 e T3 contengano set di colonne compatibili:

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

UNIONE [TUTTI | DISTINTI]

Operazione di definizione che restituisce le righe da due espressioni di query, indipendentemente dal fatto che le righe derivino da una o entrambe le espressioni.

INTERSECARE [TUTTI | DISTINTI]

Operazione di definizione che restituisce le righe che derivano da due espressioni di query. Le righe che non vengono restituite da entrambe le espressioni vengono scartate.

TRANNE [TUTTI | DISTINTI]

Operazione di definizione che restituisce le righe che derivano da una delle due espressioni di query. Per qualificarsi per il risultato, le righe devono esistere nella prima tabella dei risultati ma non nella seconda.

EXCEPT ALL non rimuove i duplicati dalle righe dei risultati.

MINUS ed EXCEPT sono sinonimi esatti.

Ordine di valutazione degli operatori di definizione

Gli operatori di definizione UNION ed EXCEPT sono associativi a sinistra. Se non si specificano le parentesi per influenzare l'ordine di precedenza, una combinazione di questi operatori di definizione viene valutata da sinistra a destra. Ad esempio, nella seguente query, l'operazione UNION di T1 e T2 viene valutata per prima, quindi l'operazione EXCEPT viene eseguita sul risultato di UNION:

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

L'operatore INTERSECT ha la precedenza sugli operatori UNION ed EXCEPT quando una combinazione di operatori viene utilizzata nella stessa query. Ad esempio, la seguente query valuta l'intersezione di T2 e T3, quindi l'unione del risultato con T1:

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

Aggiungendo le parentesi, puoi applicare un diverso ordine di valutazione. Nel seguente caso, il risultato dell'unione di T1 e T2 viene intersecato con T3 e la query produce probabilmente un risultato diverso.

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

Note per l'utilizzo

- I nomi di colonna restituiti nel risultato di una query dell'operazione di definizione sono i nomi di colonna (o alias) delle tabelle nella prima espressione di query. Poiché questi nomi di colonne sono potenzialmente fuorvianti, in quanto i valori della colonna derivano da tabelle su entrambi i lati dell'operatore di definizione, puoi fornire alias significativi per il set di risultati.
- Quando le query dell'operatore di definizione restituiscono risultati decimali, le colonne dei risultati corrispondenti vengono elevate per restituire la stessa precisione e scala. Ad esempio, nella seguente query, dove T1.REVENUE è una colonna DECIMAL (10,2) e T2.REVENUE è una colonna DECIMAL (8,4), il risultato decimale viene elevato a DECIMAL (12,4):

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

  La scala è 4 perché è la scala massima delle due colonne. La precisione è 12 perché T1.REVENUE richiede 8 cifre a sinistra del punto decimale (12 - 4 = 8). Questo tipo di elevazione garantisce che tutti i valori di entrambi i lati dell'UNION si adattino al risultato. Per i valori a 64 bit, la precisione massima del risultato è 19 e la scala del risultato massimo è 18. Per i valori a 128 bit, la precisione massima del risultato è 38 e la scala del risultato massimo è 37.

  Se il tipo di dati risultante supera i limiti AWS Clean Rooms di precisione e scala, la query restituisce un errore.
- Per le operazioni di definizione, due righe vengono considerate identiche se, per ciascuna coppia di colonne corrispondente, i due valori di dati sono uguali o entrambi NULL. Ad esempio, se le tabelle T1 e T2 contengono entrambe una colonna e una riga e tale riga è NULL in entrambe le tabelle, un'operazione INTERSECT su quelle tabelle restituisce tale riga.

Query UNION di esempio

Nella seguente query UNION, le righe nella tabella SALES vengono unite alle righe nella tabella LISTING. Tre colonne compatibili sono selezionate da ciascuna tabella; in questo caso, le colonne corrispondenti hanno gli stessi nomi e tipi di dati.

```
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
```

L'esempio seguente mostra come è possibile aggiungere un valore letterale all'output di una query UNION in modo da poter vedere quale espressione di query ha prodotto ogni riga nel set di risultati. La query identifica le righe dalla prima espressione di query come "B" (per gli acquirenti) e le righe dalla seconda espressione di query come "S" (per i venditori).

La query identifica acquirenti e venditori per transazioni di biglietti che costano almeno $10.000. L'unica differenza tra le due espressioni di query su entrambi i lati dell'operatore UNION è la colonna di collegamento per la tabella SALES.

```
select listid, lastname, firstname, username,
pricepaid as price, 'S' as buyorsell
from sales, users
where sales.sellerid=users.userid
and pricepaid >=10000
union
select listid, lastname, firstname, username, pricepaid,
'B' as buyorsell
from sales, users
where sales.buyerid=users.userid
and pricepaid >=10000

listid | lastname | firstname | username |   price   | buyorsell
--------+----------+-----------+----------+-----------+-----------
209658 | Lamb     | Colette   | VOR15LYI |  10000.00 | B
209658 | West     | Kato      | ELU81XAA |  10000.00 | S
212395 | Greer    | Harlan    | GXO71KOC |  12624.00 | S
212395 | Perry    | Cora      | YWR73YNZ |  12624.00 | B
215156 | Banks    | Patrick   | ZNQ69CLT |  10000.00 | S
215156 | Hayden   | Malachi   | BBG56AKU |  10000.00 | B
```

L'esempio seguente utilizza un operatore UNION ALL perché le righe duplicate, se trovate, devono essere conservate nel risultato. Per una serie specifica di eventi IDs, la query restituisce 0 o più righe per ogni vendita associata a ciascun evento e 0 o 1 riga per ogni elenco di quell'evento. IDs Gli eventi sono univoci per ogni riga delle tabelle LISTING ed EVENT, ma potrebbero esserci più vendite per la stessa combinazione di evento e inserzione IDs nella tabella SALES.

La terza colonna nel set di risultati identifica l'origine della riga. Se proviene dalla tabella SALES, è contrassegnata con "Yes" nella colonna SALESROW. SALESROW è un alias per SALES.LISTID. Se la riga proviene dalla tabella LISTING, è contrassegnata con "No" nella colonna SALESROW.

In questo caso, il set di risultati è costituito da tre righe di vendita per l'elenco 500, evento 7787. In altre parole, sono state eseguite tre diverse transazioni per l'elenco e la combinazione di eventi. Le altre due inserzioni, 501 e 502, non hanno prodotto vendite, quindi l'unica riga generata dalla query per questi elenchi IDs proviene dalla tabella LISTING (SALESROW = 'No').

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
where listid in(500,501,502)

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
```

Se esegui la stessa query senza la parola chiave ALL, il risultato conserva solo una delle transazioni di vendita.

```
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
```

Query UNION ALL di esempio

L'esempio seguente utilizza un operatore UNION ALL perché le righe duplicate, se trovate, devono essere conservate nel risultato. Per una serie specifica di eventi IDs, la query restituisce 0 o più righe per ogni vendita associata a ciascun evento e 0 o 1 riga per ogni elenco di quell'evento. IDs Gli eventi sono univoci per ogni riga delle tabelle LISTING ed EVENT, ma potrebbero esserci più vendite per la stessa combinazione di evento e inserzione IDs nella tabella SALES.

La terza colonna nel set di risultati identifica l'origine della riga. Se proviene dalla tabella SALES, è contrassegnata con "Yes" nella colonna SALESROW. SALESROW è un alias per SALES.LISTID. Se la riga proviene dalla tabella LISTING, è contrassegnata con "No" nella colonna SALESROW.

In questo caso, il set di risultati è costituito da tre righe di vendita per l'elenco 500, evento 7787. In altre parole, sono state eseguite tre diverse transazioni per l'elenco e la combinazione di eventi. Le altre due inserzioni, 501 e 502, non hanno prodotto vendite, quindi l'unica riga generata dalla query per questi elenchi IDs proviene dalla tabella LISTING (SALESROW = 'No').

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
where listid in(500,501,502)

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
```

Se esegui la stessa query senza la parola chiave ALL, il risultato conserva solo una delle transazioni di vendita.

```
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
```

Query INTERSECT di esempio

Confronta il seguente esempio con il primo esempio di UNION. L'unica differenza tra i due esempi è l'operatore di definizione che viene utilizzato, ma i risultati sono molto diversi. Solo una delle righe è uguale:

```
235494 |    23875 |    8771
```

Questa è l'unica riga del risultato limitato di 5 righe trovata in entrambe le tabelle.

```
select listid, sellerid, eventid from listing
intersect
select listid, sellerid, eventid from sales

listid | sellerid | eventid
--------+----------+---------
235494 |    23875 |    8771
235482 |     1067 |    2667
235479 |     1589 |    7303
235476 |    15550 |     793
235475 |    22306 |    7848
```

La seguente query trova gli eventi (per i quali sono stati venduti i biglietti) che si sono verificati nelle sedi di New York City e Los Angeles a marzo. La differenza tra le due espressioni di query è il vincolo sulla colonna 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
```

Query EXCEPT di esempio

La tabella CATEGORY del database contiene le seguenti 11 righe:

```
 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)
```

Supponi che una tabella CATEGORY_STAGE (una tabella di gestione temporanea) contenga una riga aggiuntiva:

```
 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)
```

Restituisce la differenza tra le due tabelle. In altre parole, restituisce le righe che si trovano nella tabella CATEGORY_STAGE ma non nella tabella CATEGORY:

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

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

La seguente query equivalente utilizza il sinonimo MINUS.

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

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

Se inverti l'ordine delle espressioni SELECT, la query non restituisce alcuna riga.

Clausola ORDER BY

La clausola ORDER BY ordina il set di risultati di una query.

> **Note**
>
> L'espressione ORDER BY più esterna deve contenere solo colonne presenti nell'elenco di selezione.

Argomenti

Sintassi

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

Parameters

espressione

Espressione che definisce il tipo di ordinamento del risultato della query. È costituita da una o più colonne nell'elenco di selezione. I risultati vengono restituiti in base all'ordinamento binario UTF-8. È anche possibile specificare:

- Numeri ordinali che rappresentano la posizione delle voci dell'elenco di selezione (o la posizione delle colonne nella tabella se non esiste alcun elenco di selezione)
- Alias che definiscono le voci dell'elenco di selezione

Quando la clausola ORDER BY contiene più espressioni, il set di risultati viene ordinato in base alla prima espressione, quindi la seconda espressione viene applicata alle righe che presentano valori corrispondenti della prima espressione e così via.

ASC | DESC

Opzione che definisce l'ordinamento per l'espressione, come segue:

- ASC: crescente (ad esempio, dal più piccolo al più grande per i valori numerici e da 'A' a 'Z' per le stringhe di caratteri). Se non viene specificata alcuna opzione, i dati vengono ordinati in ordine crescente per impostazione predefinita.
- DESC: decrescente (ad esempio, dal più grande al più piccolo per i valori numerici e da 'Z' ad 'A' per le stringhe).

NULLS FIRST | NULLS LAST

Opzione che specifica se i valori NULL devono essere ordinati per primi, prima dei valori non null, o per ultimi, dopo i valori non null. Per impostazione predefinita, i valori NULL vengono ordinati e classificati per ultimi in ordine ASC e ordinati e classificati per primi in ordine DESC.

LIMIT number | ALL

Opzione che controlla il numero di righe ordinate restituite dalla query. Il numero LIMIT deve essere un integer positivo; il valore massimo è `2147483647`.

LIMIT 0 non restituisce righe. Puoi utilizzare questa sintassi a scopo di test: per verificare che una query venga eseguita (senza visualizzare alcuna riga) o per restituire un elenco di colonne da una tabella. Una clausola ORDER BY è ridondante se si utilizza LIMIT 0 per restituire un elenco di colonne. L'impostazione predefinita è LIMIT ALL.

OFFSET start

Opzione che specifica di ignorare il numero di righe prima di start prima di iniziare a restituire righe. Il numero OFFSET deve essere un integer intero positivo; il valore massimo è `2147483647`. Se utilizzato con l'opzione LIMIT, le righe OFFSET vengono ignorate prima di iniziare a contare le righe LIMIT restituite. Se non si utilizza l'opzione LIMIT, il numero di righe nel set dei risultati viene ridotto del numero di righe che vengono ignorate. Le righe ignorate da una clausola OFFSET devono ancora essere analizzate, quindi potrebbe essere inefficiente utilizzare un valore OFFSET di grandi dimensioni.

Note per l'utilizzo

Nota il seguente comportamento previsto con le clausole ORDER BY:

- I valori NULL sono considerati "superiori" rispetto a tutti gli altri valori. Con l'ordine di ordinamento crescente predefinito, i valori NULL vengono ordinati alla fine. Per modificare questo comportamento, utilizza l'opzione NULLS FIRST.
- Quando una query non contiene una clausola ORDER BY, il sistema restituisce serie di risultati senza ordine prevedibile delle righe. La stessa query eseguita due volte potrebbe restituire il set di risultati in un ordine diverso.
- Le opzioni LIMIT e OFFSET possono essere utilizzate senza una clausola ORDER BY; tuttavia, per restituire un insieme coerente di righe, utilizza queste opzioni insieme a ORDER BY.
- In qualsiasi sistema parallelo come AWS Clean Rooms, quando ORDER BY non produce un ordinamento univoco, l'ordine delle righe non è deterministico. Cioè, se l'espressione ORDER BY produce valori duplicati, l'ordine di restituzione di tali righe potrebbe variare da un sistema all'altro o da un'esecuzione all'altra. AWS Clean Rooms
- AWS Clean Rooms non supporta stringhe letterali nelle clausole ORDER BY.

Esempi di ORDER BY

Restituisce tutte le 11 righe dalla tabella CATEGORY, ordinate in base alla seconda colonna CATGROUP. Per i risultati con lo stesso valore CATGROUP, ordina i valori della colonna CATDESC in base alla lunghezza della stringa di caratteri. Quindi ordina per colonne CATID e CATNAME.

```
select * from category order by 2, 1, 3;

catid | catgroup |  catname  |                 catdesc
-------+----------+-----------+--------------------------------------
10 | Concerts | Jazz      | All jazz singers and bands
9 | Concerts | Pop       | All rock and pop music concerts
11 | Concerts | Classical | All symphony, concerto, and choir conce
6 | Shows    | Musicals  | Musical theatre
7 | Shows    | Plays     | All non-musical theatre
8 | Shows    | Opera     | All opera and light opera
5 | Sports   | MLS       | Major League Soccer
1 | Sports   | MLB       | Major League Baseball
2 | Sports   | NHL       | National Hockey League
3 | Sports   | NFL       | National Football League
4 | Sports   | NBA       | National Basketball Association
```

```
(11 rows)
```

Restituisce le colonne selezionate dalla tabella SALES, ordinate in base ai valori QTYSOLD più alti. Limita il risultato alle prime 10 righe:

```
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
```

Restituisce un elenco di colonne e nessuna riga usando la sintassi LIMIT 0:

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

Esempi di sottoquery

Gli esempi seguenti mostrano diversi modi in cui le sottoquery si adattano alle query SELECT. Per un altro esempio dell'uso delle sottoquery, consultare Esempio.

Sottoquery dell'elenco SELECT

L'esempio seguente contiene una sottoquery nell'elenco SELECT. Questa sottoquery è scalar: restituisce solo una colonna e un valore, che viene ripetuto nel risultato per ogni riga restituita dalla query esterna. La query confronta il valore Q1SALES che la sottoquery calcola con i valori di vendita per due altri trimestri (2 e 3) nel 2008, come definito dalla query esterna.

```
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)
```

Sottoquery della clausola WHERE

L'esempio seguente contiene una sottoquery di tabella nella clausola WHERE. Questa sottoquery produce più righe. In questo caso, le righe contengono solo una colonna, ma le sottoquery di tabella possono contenere più colonne e righe, proprio come qualsiasi altra tabella.

La query trova i primi 10 venditori in termini di numero di biglietti venduti. L'elenco dei primi 10 è limitato dalla sottoquery che rimuove gli utenti che vivono in città dove ci sono le sedi dei biglietti. Questa query può essere scritta in diversi modi; ad esempio, la sottoquery potrebbe essere riscritta come un join all'interno della query principale.

```
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)
```

Sottoquery della clausola WITH

Per informazioni, consulta Clausola WITH.

Sottoquery correlate

L'esempio seguente contiene una sottoquery correlata nella clausola WHERE; questo tipo di sottoquery contiene una o più correlazioni tra le sue colonne e le colonne prodotte dalla query esterna. In questo caso, la correlazione è `where s.listid=l.listid`. Per ogni riga prodotta dalla query esterna, la sottoquery viene eseguita per qualificare o squalificare la riga.

```
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)
```

Modelli di sottoquery correlate non supportate

Il pianificatore di query utilizza un metodo di riscrittura delle query denominato decorrelazione delle sottoquery per ottimizzare diversi modelli di sottoquery correlate per l'esecuzione in un ambiente MPP. Alcuni tipi di sottoquery correlate seguono schemi che non AWS Clean Rooms possono essere decorrelate e che non supportano. Le query che contengono i seguenti riferimenti di correlazione restituiscono errori:

- Riferimenti di correlazione che ignorano un blocco di query, noti anche come "riferimenti di correlazione ignorati". Ad esempio, nella seguente query, il blocco contenente il riferimento di correlazione e il blocco ignorato sono collegati da un predicato NOT EXISTS:

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

Il blocco ignorato in questo caso è la sottoquery rispetto alla tabella LISTING. Il riferimento di correlazione correla le tabelle EVENT e SALES.

- Riferimenti di correlazione di una sottoquery che fa parte di una clausola ON in una query esterna:

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

La clausola ON contiene un riferimento di correlazione da SALES nella sottoquery a EVENT nella query esterna.

- Riferimenti di correlazione sensibili a valori nulli a una tabella di sistema. AWS Clean Rooms Esempio:

```
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);
```

- Riferimenti di correlazione da una sottoquery contenente una funzione finestra.

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

- Riferimenti in una colonna GROUP BY ai risultati di una sottoquery correlata. Ad esempio:

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

- Riferimenti di correlazione da una sottoquery con una funzione di aggregazione e una clausola GROUP BY, connessi alla query esterna da un predicato IN. Questa restrizione non si applica alle funzioni di aggregazione MIN e MAX. Esempio:

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

AWS Clean Rooms Funzioni Spark SQL

AWS Clean Rooms Spark SQL supporta le seguenti funzioni SQL:

Argomenti

Funzioni di aggregazione

Le funzioni aggregate in AWS Clean Rooms Spark SQL vengono utilizzate per eseguire calcoli o operazioni su un gruppo di righe e restituire un singolo valore. Sono essenziali per le attività di analisi e riepilogo dei dati.

AWS Clean Rooms Spark SQL supporta le seguenti funzioni di aggregazione:

Argomenti

Funzione ANY_VALUE

La funzione ANY_VALUE restituisce qualsiasi valore dai valori dell'espressione di input in modo non deterministico. Questa funzione può restituire NULL se l'espressione di input non determina la restituzione di righe.

Sintassi

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

Arguments (Argomenti)

expression

L'espressione o la colonna di destinazione su cui viene eseguita la funzione. L'espressione è uno dei seguenti tipi di dati:

isIgnoreNull

Un valore booleano che determina se la funzione deve restituire solo valori non nulli.

Valori restituiti

Restituisce lo stesso tipo di dati come espressione.

Note per l'utilizzo

Se un'istruzione che specifica la funzione ANY_VALUE per una colonna include anche un riferimento a una secondo colonna, la seconda colonna deve essere visualizzata in una clausola GROUP BY o inclusa in una funzione di aggregazione.

Esempi

L'esempio seguente restituisce un'istanza di any `dateid` where the is. `eventname` `Eagles`

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

Di seguito sono riportati i risultati.

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

L'esempio seguente restituisce un'istanza di any `dateid` where the `eventname` is `Eagles` or`Cold War Kids`.

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

Di seguito sono riportati i risultati.

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

Funzione APPROX COUNT_DISTINCT

APPROX COUNT_DISTINCT fornisce un modo efficiente per stimare il numero di valori univoci in una colonna o in un set di dati.

Sintassi

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

Arguments (Argomenti)

expr

L'espressione o la colonna per cui si desidera stimare il numero di valori univoci.

Può essere una singola colonna, un'espressione complessa o una combinazione di colonne.

Relative D

Un parametro opzionale che specifica la deviazione standard relativa desiderata della stima.

È un valore compreso tra 0 e 1, che rappresenta l'errore relativo massimo accettabile della stima. Un valore RelativeD inferiore darà come risultato una stima più accurata ma più lenta.

Se questo parametro non viene fornito, viene utilizzato un valore predefinito (in genere intorno a 0,05 o 5%).

Valori restituiti

Restituisce la cardinalità stimata in HyperLogLog ++. relativeSD definisce la deviazione standard relativa massima consentita.

Esempio

La seguente query stima il numero di valori univoci nella `col1` colonna, con una deviazione standard relativa dell'1% (0,01).

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

La seguente query stima che nella `col1` colonna siano presenti 3 valori univoci (i valori 1, 2 e 3).

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

Funzione APPROX PERCENTILE

APPROX PERCENTILE viene utilizzato per stimare il valore percentile di una determinata espressione o colonna senza dover ordinare l'intero set di dati. Questa funzione è utile in scenari in cui è necessario comprendere rapidamente la distribuzione di un set di dati di grandi dimensioni o tenere traccia delle metriche basate sui percentili, senza il sovraccarico computazionale dovuto all'esecuzione di un calcolo percentile esatto. Tuttavia, è importante comprendere i compromessi tra velocità e precisione e scegliere la tolleranza di errore appropriata in base ai requisiti specifici del caso d'uso.

Sintassi

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

Arguments (Argomenti)

expr

L'espressione o la colonna per cui si desidera stimare il valore del percentile.

Può essere una singola colonna, un'espressione complessa o una combinazione di colonne.

percentile

Il valore percentile da stimare, espresso come valore compreso tra 0 e 1.

Ad esempio, 0,5 corrisponderebbe al 50° percentile (mediana).

precisione

Un parametro opzionale che specifica la precisione desiderata della stima del percentile. È un valore compreso tra 0 e 1, che rappresenta l'errore relativo massimo accettabile della stima. Un

`accuracy` valore inferiore darà come risultato una stima più precisa ma più lenta. Se questo parametro non viene fornito, viene utilizzato un valore predefinito (in genere intorno allo 0,05 o al 5%).

Valori restituiti

Restituisce il percentile approssimativo della colonna numerica o dell'intervallo ANSI col, che è il valore più piccolo tra i valori col ordinati (ordinati dal minimo al più grande) in modo che non più della percentuale dei valori col sia inferiore o uguale a tale valore.

Il valore della percentuale deve essere compreso tra 0,0 e 1,0. Il parametro di precisione (predefinito: 10000) è un valore letterale numerico positivo che controlla la precisione dell'approssimazione a scapito della memoria.

Un valore di precisione più elevato produce una migliore precisione, `1.0/accuracy` è l'errore relativo dell'approssimazione.

Quando la percentuale è una matrice, ogni valore della matrice percentuale deve essere compreso tra 0,0 e 1,0. In questo caso, restituisce l'array percentile approssimativo della colonna col nella matrice percentuale specificata.

Esempi

La seguente query stima il 95° percentile della `response_time` colonna, con un errore relativo massimo dell'1% (0,01).

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

La seguente query stima i valori del 50°, 40° e 10° percentile della colonna della tabella. `col` `tab`

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

La seguente query stima il 50° percentile (mediano) dei valori nella colonna col.

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

Funzione AVG

La AVG funzione restituisce la media (media aritmetica) dei valori delle espressioni di input. La AVG funzione funziona con valori numerici e ignora i valori NULL.

Sintassi

```
AVG (column)
```

Arguments (Argomenti)

column

La colonna di destinazione su cui opera la funzione. La colonna è uno dei seguenti tipi di dati:

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

Tipi di dati

I tipi di argomento supportati dalla AVG funzione sono SMALLINTINTEGER,BIGINT,DECIMAL, eDOUBLE.

I tipi di ritorno supportati dalla AVG funzione sono:

- BIGINTper qualsiasi argomento di tipo intero
- DOUBLEper un argomento in virgola mobile
- Restituisce lo stesso tipo di dati dell'espressione per qualsiasi altro tipo di argomento

La precisione predefinita per il risultato di una AVG funzione con un DECIMAL argomento è 38. Il ridimensionamento del risultato coincide con il ridimensionamento dell'argomento. Ad esempio, un elemento AVG di una DEC(5,2) colonna restituisce un tipo di DEC(38,2) dati.

Esempio

Trova la quantità media venduta per transazione dalla SALES tabella.

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

Funzione BOOL_AND

La funzione BOOL_AND opera su una singola colonna o espressione booleana o intera. Questa funzione applica una logica simile alle funzioni BIT_AND e BIT_OR. Per questa funzione, il tipo restituito è un valore booleano (`true` o `false`).

Se tutti i valori in un insieme sono veri, viene restituita la funzione BOOL_AND `true` (`t`). Se un valore è falso, la funzione restituisce `false` (`f`).

Sintassi

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

Arguments (Argomenti)

expression

L'espressione o colonna di destinazione su cui viene eseguita la funzione. Questa espressione deve avere un tipo di dati intero o BOOLEAN. Il tipo restituito della funzione è BOOLEAN.

DISTINCT | ALL

Con l'argomento DISTINCT, la funzione elimina tutti i valori duplicati per l'espressione specificata prima di calcolare il risultato. Con l'argomento ALL, la funzione mantiene tutti i valori duplicati. ALL è il valore predefinito.

Esempi

È possibile utilizzare le funzioni booleane rispetto alle espressioni booleane o alle espressioni intere.

Ad esempio, la seguente query restituisce i risultati dalla tabella USERS standard nel database TICKIT, che ha diverse colonne booleane.

La funzione BOOL_AND restituisce `false` per tutte e cinque le righe. Non a tutti gli utenti in ciascuno di questi stati piace lo sport.

```
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)
```

Funzione BOOL_OR

La funzione BOOL_OR opera su una singola colonna o espressione booleana o intera. Questa funzione applica una logica simile alle funzioni BIT_AND e BIT_OR. Per questa funzione, il tipo restituito è un valore booleano (`true`, `false` o NULL).

Se un valore in un insieme è `true`, la funzione BOOL_OR restituisce `true` (`t`). Se un valore in un insieme è `false`, la funzione restituisce `false` (`f`). NULL può essere restituito se il valore è sconosciuto.

Sintassi

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

Arguments (Argomenti)

expression

L'espressione o colonna di destinazione su cui viene eseguita la funzione. Questa espressione deve avere un tipo di dati intero o BOOLEAN. Il tipo restituito della funzione è BOOLEAN.

DISTINCT | ALL

Con l'argomento DISTINCT, la funzione elimina tutti i valori duplicati per l'espressione specificata prima di calcolare il risultato. Con l'argomento ALL, la funzione mantiene tutti i valori duplicati. ALL è il valore predefinito.

Esempi

È possibile utilizzare le funzioni booleane rispetto alle espressioni booleane o alle espressioni intere. Ad esempio, la seguente query restituisce i risultati dalla tabella USERS standard nel database TICKIT, che ha diverse colonne booleane.

La funzione BOOL_OR restituisce `true` per tutte e cinque le righe. Ad almeno un utente in ciascuno di questi stati piace lo sport.

```
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)
```

Il seguente esempio restituisce NULL.

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

Funzione CARDINALITY

La funzione CARDINALITY restituisce la dimensione di un'espressione ARRAY o MAP (expr).

Questa funzione è utile per trovare la dimensione o la lunghezza di un array.

Sintassi

```
cardinality(expr)
```

Arguments (Argomenti)

expr

Un'espressione ARRAY o MAP.

Valori restituiti

Restituisce la dimensione di un array o di una mappa (INTEGER).

La funzione restituisce `NULL` un input nullo se `sizeOfNull` è impostata su `false` o `enabled` è impostata su. `true`

Altrimenti, la funzione restituisce un `-1` input nullo. Con le impostazioni predefinite, la funzione restituisce un `-1` input nullo.

Esempio

La seguente query calcola la cardinalità, o il numero di elementi, nell'array dato. L'array (`'b', 'd', 'c', 'a'`) ha 4 elementi, quindi l'output di questa query sarebbe. `4`

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

Funzione COLLECT_LIST

La funzione COLLECT_LIST raccoglie e restituisce un elenco di elementi non unici.

Questo tipo di funzione è utile quando si desidera raccogliere più valori da un insieme di righe in una singola matrice o struttura di dati di elenco.

> **Note**
>
> La funzione non è deterministica perché l'ordine dei risultati raccolti dipende dall'ordine delle righe, che può essere non deterministico dopo l'esecuzione di un'operazione di shuffle.

Sintassi

```
collect_list(expr)
```

Arguments (Argomenti)

expr

Un'espressione di qualsiasi tipo.

Valori restituiti

Restituisce un ARRAY del tipo di argomento. L'ordine degli elementi nell'array non è deterministico.

I valori NULL sono esclusi.

Se viene specificato DISTINCT, la funzione raccoglie solo valori univoci ed è sinonimo di funzione aggregata. `collect_set`

Esempio

La seguente query raccoglie tutti i valori dalla colonna col in un elenco. La `VALUES` clausola viene utilizzata per creare una tabella in linea con tre righe, in cui ogni riga ha una singola colonna con i valori 1, 2 e 1 rispettivamente. La `collect_list()` funzione viene quindi utilizzata per aggregare tutti i valori della colonna col in un unico array. L'output di questa istruzione SQL sarebbe l'array`[1,2,1]`, che contiene tutti i valori della colonna col nell'ordine in cui sono apparsi nei dati di input.

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

Funzione COLLECT_SET

La funzione COLLECT_SET raccoglie e restituisce un set di elementi unici.

Questa funzione è utile quando si desidera raccogliere tutti i valori distinti da un insieme di righe in un'unica struttura di dati, senza includere duplicati.

> **Note**
>
> La funzione non è deterministica perché l'ordine dei risultati raccolti dipende dall'ordine delle righe, che può essere non deterministico dopo l'esecuzione di un'operazione di shuffle.

Sintassi

```
collect_set(expr)
```

Arguments (Argomenti)

expr

Un'espressione di qualsiasi tipo tranne MAP.

Valori restituiti

Restituisce un ARRAY del tipo di argomento. L'ordine degli elementi nell'array non è deterministico.

I valori NULL sono esclusi.

Esempio

La seguente query raccoglie tutti i valori univoci dalla colonna col in un set. La `VALUES` clausola viene utilizzata per creare una tabella in linea con tre righe, in cui ogni riga ha una singola colonna con i valori 1, 2 e 1 rispettivamente. La `collect_set()` funzione viene quindi utilizzata per aggregare tutti i valori univoci della colonna col in un unico set. L'output di questa istruzione SQL sarebbe il set`[1,2]`, che contiene i valori univoci della colonna col. Il valore duplicato di 1 viene incluso una sola volta nel risultato.

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

COUNTe funzioni COUNT DISTINCT

La COUNT funzione conta le righe definite dall'espressione. La COUNT DISTINCT funzione calcola il numero di valori distinti non NULL in una colonna o in un'espressione. Elimina tutti i valori duplicati dall'espressione specificata prima di eseguire il conteggio.

Sintassi

```
COUNT (DISTINCT column)
```

Arguments (Argomenti)

column

La colonna di destinazione su cui opera la funzione.

Tipi di dati

La COUNT funzione e la COUNT DISTINCT funzione supportano tutti i tipi di dati degli argomenti.

La COUNT DISTINCT funzione restituisceBIGINT.

Esempi

Conta tutti gli utenti dello stato della Florida.

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

Conta tutti i locali unici IDs visti dalla EVENT tabella.

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

Funzione COUNT

La funzione COUNT conta le righe definite dall'espressione.

La funzione COUNT ha le seguenti variazioni.

- COUNT (*) conta tutte le righe nella tabella di destinazione indipendentemente dal fatto che includano valori null o meno.
- COUNT (espressione) calcola il numero di righe con valori non NULL in una colonna o espressione specifica.
- COUNT (DISTINCT espressione) calcola il numero di valori distinti non NULL in una colonna o espressione.

Sintassi

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

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

Arguments (Argomenti)

expression

L'espressione o colonna di destinazione su cui viene eseguita la funzione. La funzione COUNT supporta tutti i tipi di dati degli argomenti.

DISTINCT | ALL

Con l'argomento DISTINCT, la funzione elimina tutti i valori duplicati dall'espressione specificata prima di eseguire il conteggio. Con l'argomento ALL, la funzione mantiene tutti i valori duplicati dall'espressione per il conteggio. ALL è il valore predefinito.

Tipo restituito

La funzione COUNT restituisce BIGINT.

Esempi

Calcolare tutti gli utenti provenienti dallo stato della Florida:

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

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

Calcolare tutti i nomi degli eventi dalla tabella EVENT:

```
select count(eventname) from event;

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

Calcolare tutti i nomi degli eventi dalla tabella EVENT:

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

count
```

```
-------
8798
```

Conta tutti i luoghi unici presenti IDs nella tabella EVENT:

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

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

Contare il numero di volte in cui ciascun venditore ha elencato lotti di oltre quattro biglietti in vendita. Raggruppare i risultati per ID venditore:

```
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
...
```

Funzione MAX

La funzione MAX restituisce il valore massimo in un insieme di righe. È possibile utilizzare DISTINCT oppure ALL ma non influenzano il risultato.

Sintassi

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

Arguments (Argomenti)

expression

L'espressione o colonna di destinazione su cui viene eseguita la funzione. L'espressione è qualsiasi tipo di dato numerico.

DISTINCT | ALL

Con l'argomento DISTINCT, la funzione elimina tutti i valori duplicati dall'espressione specificata prima di calcolare il massimo. Con l'argomento ALL, la funzione mantiene tutti i valori duplicati dall'espressione per calcolare il massimo. ALL è il valore predefinito.

Tipi di dati

Restituisce lo stesso tipo di dati come espressione.

Esempi

Trovare il prezzo più alto pagato da tutte le vendite:

```
select max(pricepaid) from sales;

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

Trovare il prezzo più alto pagato per biglietto da tutte le vendite:

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

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

Funzione MEDIAN

Sintassi

```
MEDIAN ( median_expression )
```

Arguments (Argomenti)

median_expression

L'espressione o colonna di destinazione su cui viene eseguita la funzione.

Funzione MIN

La funzione MIN restituisce il valore minimo in un insieme di righe. È possibile utilizzare DISTINCT oppure ALL ma non influenzano il risultato.

Sintassi

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

Arguments (Argomenti)

expression

L'espressione o colonna di destinazione su cui viene eseguita la funzione. L'espressione è un tipo di dati numerico qualsiasi.

DISTINCT | ALL

Con l'argomento DISTINCT, la funzione elimina tutti i valori duplicati dall'espressione specificata prima di calcolare il minimo. Con l'argomento ALL, la funzione mantiene tutti i valori duplicati dall'espressione per calcolare il minimo. ALL è il valore predefinito.

Tipi di dati

Restituisce lo stesso tipo di dati come espressione.

Esempi

Trovare il prezzo più basso pagato da tutte le vendite:

```
select min(pricepaid) from sales;

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

Trovare il prezzo più basso pagato per biglietto da tutte le vendite:

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

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

Funzione PERCENTILE

La funzione PERCENTILE viene utilizzata per calcolare il valore percentile esatto ordinando prima i valori nella `col` colonna e poi trovando il valore nel valore specificato. `percentage`

La funzione PERCENTILE è utile quando è necessario calcolare il valore percentile esatto e il costo computazionale è accettabile per il caso d'uso. Fornisce risultati più accurati rispetto alla funzione APPROX_PERCENTILE, ma può essere più lenta, specialmente per set di dati di grandi dimensioni.

Al contrario, la funzione APPROX_PERCENTILE è un'alternativa più efficiente in grado di fornire una stima del valore del percentile con una tolleranza di errore specificata, il che la rende più adatta per scenari in cui la velocità ha una priorità maggiore rispetto alla precisione assoluta.

Sintassi

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

Arguments (Argomenti)

col

L'espressione o la colonna per cui si desidera calcolare il valore del percentile.

percentuale

Il valore percentile da calcolare, espresso come valore compreso tra 0 e 1.

Ad esempio, 0,5 corrisponderebbe al 50° percentile (mediana).

frequenza

Un parametro opzionale che specifica la frequenza o il peso di ogni valore nella `col` colonna. Se fornito, la funzione calcolerà il percentile in base alla frequenza di ciascun valore.

Valori restituiti

Restituisce il valore percentile esatto della colonna numerica o dell'intervallo ANSI col alla percentuale specificata.

Il valore della percentuale deve essere compreso tra 0,0 e 1,0.

Il valore della frequenza deve essere un integrale positivo

Esempio

La seguente query trova il valore maggiore o uguale al 30% dei valori nella `col` colonna. Poiché i valori sono 0 e 10, il 30° percentile è 3,0, poiché è il valore maggiore o uguale al 30% dei dati.

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

Funzione SKEWNESS

La funzione SKEWNESS restituisce il valore di asimmetria calcolato in base ai valori di un gruppo.

L'asimmetria è una misura statistica che descrive l'asimmetria o la mancanza di simmetria in un set di dati. Fornisce informazioni sulla forma della distribuzione dei dati.

Questa funzione può essere utile per comprendere le proprietà statistiche di un set di dati e fornire informazioni per ulteriori analisi o processi decisionali.

Sintassi

```
skewness(expr)
```

Arguments (Argomenti)

expr

Un'espressione che restituisce un valore numerico.

Valori restituiti

Restituisce DOUBLE.

Se viene specificato DISTINCT, la funzione opera solo su un set univoco di valori expr.

Esempi

La seguente query calcola l'asimmetria dei valori nella colonna. `col` In questo esempio, la `VALUES` clausola viene utilizzata per creare una tabella in linea con quattro righe, in cui ogni riga ha una

singola colonna `col` con i valori -10, -20, 100 e 1000. La `skewness()` funzione viene quindi utilizzata per calcolare l'asimmetria dei valori nella colonna. `col` Il risultato, 1.1135657469022011, rappresenta il grado e la direzione dell'asimmetria nei dati. Un valore di asimmetria positivo indica che i dati sono inclinati verso destra, con la maggior parte dei valori concentrati sul lato sinistro della distribuzione. Un valore di asimmetria negativo indica che i dati sono inclinati verso sinistra, con la maggior parte dei valori concentrati sul lato destro della distribuzione.

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

La seguente query calcola l'asimmetria dei valori nella colonna col. Analogamente all'esempio precedente, la `VALUES` clausola viene utilizzata per creare una tabella in linea con quattro righe, in cui ogni riga ha una singola colonna `col` con i valori -1000, -100, 10 e 20. La `skewness()` funzione viene quindi utilizzata per calcolare l'asimmetria dei valori nella colonna. `col` Il risultato, -1.1135657469022011, rappresenta il grado e la direzione dell'asimmetria nei dati. In questo caso, il valore di asimmetria negativo indica che i dati sono inclinati verso sinistra, con la maggior parte dei valori concentrati sul lato destro della distribuzione.

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

Funzioni STDDEV_SAMP e STDDEV_POP

Le funzioni STDDEV_SAMP e STDDEV_POP restituiscono la deviazione standard del campione e della popolazione di un insieme di valori numerici (integer, numero decimale, numero in virgola mobile). Il risultato della funzione STDDEV_SAMP è equivalente alla radice quadrata della varianza campione dello stesso insieme di valori.

STDDEV_SAMP e STDDEV sono sinonimi della stessa funzione.

Sintassi

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

L'espressione deve avere un tipo di dati numerico. Indipendentemente dal tipo di dati dell'espressione, il tipo di restituzione di questa funzione è un numero a precisione doppia.

> **Note**
>
> La deviazione standard viene calcolata utilizzando l'aritmetica del numero in virgola mobile, che potrebbe causare una leggera imprecisione.

Note per l'utilizzo

Quando la deviazione standard del campione (STDDEV o STDDEV_SAMP) viene calcolata per un'espressione che consiste in un singolo valore, il risultato della funzione è NULL non 0.

Esempi

La seguente query restituisce la media dei valori nella colonna VENUESEATS della tabella VENUE, seguita dalla deviazione standard del campione e dalla deviazione standard della popolazione dello stesso insieme di valori. VENUESEATS è una colonna INTEGER. Il ridimensionamento del risultato è ridotto a 2 cifre.

```
select avg(venueseats),
cast(stddev_samp(venueseats) as dec(14,2)) stddevsamp,
cast(stddev_pop(venueseats) as dec(14,2)) stddevpop
from venue;

avg  | stddevsamp | stddevpop
-------+------------+-----------
17503 |   27847.76 |  27773.20
(1 row)
```

La seguente query restituisce la deviazione standard del campione per la colonna COMMISSIONE nella tabella SALES. COMMISSION è una colonna DECIMAL. Il ridimensionamento del risultato è ridotto a 10 cifre.

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

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

La seguente query assegna la deviazione standard del campione per la colonna COMMISSIONE come un integer.

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

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

La seguente query restituisce sia la deviazione standard del campione sia la radice quadrata della varianza campionaria per la colonna COMMISSIONE. I risultati di questi calcoli sono gli stessi.

```
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)
```

SUMe funzioni SUM DISTINCT

La SUM funzione restituisce la somma dei valori della colonna di input o dell'espressione. La SUM funzione funziona con valori numerici e ignora i valori. NULL

La SUM DISTINCT funzione elimina tutti i valori duplicati dall'espressione specificata prima di calcolare la somma.

Sintassi

```
SUM (DISTINCT column )
```

Arguments (Argomenti)

column

La colonna di destinazione su cui opera la funzione. La colonna contiene qualsiasi tipo di dato numerico.

Esempi

Trova la somma di tutte le commissioni pagate dalla SALES tabella.

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

Trova la somma di tutte le commissioni distinte pagate dalla SALES tabella.

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

Funzioni VAR_SAMP e VAR_POP

Le funzioni VAR_SAMP e VAR_POP restituiscono la varianza del campione e della popolazione di un insieme di valori numerici (integer, numero decimale, numero in virgola mobile). Il risultato della funzione VAR_SAMP è equivalente alla deviazione standard del campione quadrato dello stesso insieme di valori.

VAR_SAMP e VARIANCE sono sinonimi della stessa funzione.

Sintassi

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

L'espressione deve avere un tipo di dati integer, numero decimale o numero in virgola mobile. Indipendentemente dal tipo di dati dell'espressione, il tipo di restituzione di questa funzione è un numero a precisione doppia.

> **Note**
>
> I risultati di queste funzioni potrebbero variare tra i cluster di data warehouse, a seconda della configurazione del cluster in ciascun caso.

Note per l'utilizzo

Quando la varianza del campione (VARIANCE o VAR_SAMP) viene calcolata per un'espressione che consiste in un singolo valore, il risultato della funzione è NULL non 0.

Esempi

La seguente query restituisce l'esempio arrotondato e la varianza di popolazione della colonna NUMTICKETS nella tabella LISTING.

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

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

La seguente query esegue gli stessi calcoli ma assegna i risultati ai valori decimali.

```
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)
```

Funzioni di array

Questa sezione descrive le funzioni di array per SQL supportate in. AWS Clean Rooms

Argomenti

- Funzione ARRAY
- Funzione ARRAY_CONTAINS
- Funzione ARRAY_DISTINCT
- Funzione ARRAY_EXCEPT
- Funzione ARRAY_INTERSECT
- Funzione ARRAY_JOIN

- Funzione ARRAY_REMOVE
- Funzione ARRAY_UNION
- Funzione EXPLODE
- Funzione FLATTEN

Funzione ARRAY

Crea un array con gli elementi dati.

Sintassi

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

Argomento

expr1, expr2

Espressioni di qualsiasi tipo di dati ad eccezione dei tipi di data e ora. Non è necessario che gli argomenti siano dello stesso tipo di dati.

Tipo restituito

La funzione array restituisce un ARRAY con gli elementi dell'espressione.

Esempio

L'esempio seguente mostra una matrice di valori numerici e una matrice di diversi tipi di dati.

```
--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)
```

Funzione ARRAY_CONTAINS

La funzione ARRAY_CONTAINS può essere utilizzata per eseguire controlli di appartenenza di base sulle strutture di dati degli array. La funzione ARRAY_CONTAINS è utile quando è necessario verificare se un valore specifico è presente all'interno di un array.

Sintassi

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

Argomenti

matrice

Un ARRAY da cercare.

value

Un'espressione con un tipo che condivide un tipo meno comune con gli elementi dell'array.

Tipo restituito

La funzione ARRAY_CONTAINS restituisce un valore BOOLEAN.

Se il valore è NULL, il risultato è NULL.

Se un elemento dell'array è NULL, il risultato è NULL se il valore non corrisponde a nessun altro elemento.

Esempi

L'esempio seguente verifica se l'array `[1, 2, 3]` contiene il valore. `4` Poiché l'array`[1, 2, 3]` non contiene il valore`4`, viene restituita la funzione array_contains. `false`

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

L'esempio seguente verifica se l'array `[1, 2, 3]` contiene il valore. `2` Poiché l'array `[1, 2, 3]` contiene il valore`2`, viene restituita la funzione array_contains. `true`

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

Funzione ARRAY_DISTINCT

La funzione ARRAY_DISTINCT può essere utilizzata per rimuovere valori duplicati da un array. La funzione ARRAY_DISTINCT è utile quando è necessario rimuovere i duplicati da un array e lavorare solo con gli elementi unici. Ciò può essere utile in scenari in cui si desidera eseguire operazioni o analisi su un set di dati senza l'interferenza di valori ripetuti.

Sintassi

```
array_distinct(array)
```

Argomenti

matrice

Un'espressione ARRAY.

Tipo restituito

La funzione ARRAY_DISTINCT restituisce un ARRAY che contiene solo gli elementi unici dell'array di input.

Esempi

In questo esempio, l'array di input `[1, 2, 3, null, 3]` contiene un valore duplicato di. `3` La `array_distinct` funzione rimuove questo valore duplicato `3` e restituisce un nuovo array con gli elementi unici:. `[1, 2, 3, null]`

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

In questo esempio, la matrice di input `[1, 2, 2, 3, 3, 3]` contiene valori duplicati di `2` e. `3` La `array_distinct` funzione rimuove questi duplicati e restituisce un nuovo array con gli elementi unici:. `[1, 2, 3]`

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

```
  [1,2,3]
```

Funzione ARRAY_EXCEPT

La funzione ARRAY_EXCEPT accetta due array come argomenti e restituisce un nuovo array che contiene solo gli elementi presenti nel primo array ma non nel secondo array.

ARRAY_EXCEPT è utile quando è necessario trovare gli elementi che sono unici per un array rispetto a un altro. Questo può essere utile in scenari in cui è necessario eseguire operazioni simili a set sugli array, come trovare la differenza tra due set di dati.

Sintassi

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

Argomenti

matrice 1

Un ARRAY di qualsiasi tipo con elementi comparabili.

matrice 2

Un ARRAY di elementi che condividono un tipo meno comune con gli elementi di array1.

Tipo restituito

La funzione ARRAY_EXCEPT restituisce un ARRAY di tipo corrispondente a array1 senza duplicati.

Esempi

In questo esempio, il primo array `[1, 2, 3]` contiene gli elementi 1, 2 e 3. Il secondo array `[2, 3, 4]` contiene gli elementi 2, 3 e 4. La `array_except` funzione rimuove gli elementi 2 e 3 dal primo array, poiché sono presenti anche nel secondo array. L'output risultante è l'array`[1]`.

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

In questo esempio, il primo array `[1, 2, 3]` contiene gli elementi 1, 2 e 3. Il secondo array `[1, 3, 5]` contiene gli elementi 1, 3 e 5. La `array_except` funzione rimuove gli elementi 1 e 3 dal primo array, poiché sono presenti anche nel secondo array. L'output risultante è l'array`[2]`.

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

Funzione ARRAY_INTERSECT

La funzione ARRAY_INTERSECT accetta due array come argomenti e restituisce un nuovo array che contiene gli elementi presenti in entrambi gli array di input. Questa funzione è utile quando è necessario trovare gli elementi comuni tra due array. Ciò può essere utile in scenari in cui è necessario eseguire operazioni simili a set sugli array, ad esempio trovare l'intersezione tra due set di dati.

Sintassi

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

Argomenti

matrice 1

Un ARRAY di qualsiasi tipo con elementi comparabili.

matrice 2

Un ARRAY di elementi che condividono un tipo meno comune con gli elementi di array1.

Tipo restituito

La funzione ARRAY_INTERSECT restituisce un ARRAY di tipo corrispondente a array1 senza duplicati ed elementi contenuti sia in array1 che in array2.

Esempi

In questo esempio, il primo array contiene gli elementi 1, 2 e 3. `[1, 2, 3]` Il secondo array `[1, 3, 5]` contiene gli elementi 1, 3 e 5. La funzione ARRAY_INTERSECT identifica gli elementi comuni tra i due array, che sono 1 e 3. L'array `[1, 3]` di output risultante è.

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

Funzione ARRAY_JOIN

La funzione ARRAY_JOIN accetta due argomenti: il primo argomento è l'array di input che verrà unito. Il secondo argomento è la stringa separatrice che verrà utilizzata per concatenare gli elementi dell'array. Questa funzione è utile quando è necessario convertire una matrice di stringhe (o qualsiasi altro tipo di dati) in un'unica stringa concatenata. Ciò può essere utile negli scenari in cui si desidera presentare una matrice di valori come una singola stringa formattata, ad esempio per scopi di visualizzazione o per l'utilizzo in ulteriori elaborazioni.

Sintassi

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

Argomenti

matrice

Qualsiasi tipo di ARRAY, ma i relativi elementi vengono interpretati come stringhe.

delimiter

Una STRINGA utilizzata per separare gli elementi dell'array concatenati.

Sostituzione nulla

Una STRINGA usata per esprimere un valore NULL nel risultato.

Tipo restituito

La funzione ARRAY_JOIN restituisce una STRING in cui gli elementi dell'array sono separati da un delimitatore e vengono sostituiti gli elementi nulli. `nullReplacement` Se `nullReplacement` viene omesso, gli elementi vengono filtrati. `null` Se un argomento è`NULL`, il risultato è. `NULL`

Esempi

In questo esempio, la funzione ARRAY_JOIN prende l'array `['hello', 'world']` e unisce gli elementi utilizzando il separatore `' '` (un carattere di spazio). L'output risultante è la stringa. `'hello world'`

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

In questo esempio, la funzione ARRAY_JOIN prende l'array `['hello', null, 'world']` e unisce gli elementi utilizzando il separatore `' '` (un carattere di spazio). Il `null` valore viene sostituito con la stringa `','` sostitutiva fornita (una virgola). L'output risultante è la stringa`'hello , world'`.

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

Funzione ARRAY_REMOVE

La funzione ARRAY_REMOVE accetta due argomenti: il primo argomento è l'array di input da cui verranno rimossi gli elementi. Il secondo argomento è il valore che verrà rimosso dall'array. Questa funzione è utile quando è necessario rimuovere elementi specifici da un array. Ciò può essere utile in scenari in cui è necessario eseguire la pulizia o la preelaborazione dei dati su una matrice di valori.

Sintassi

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

Argomenti

matrice

Un ARRAY.

elemento

Un'espressione di un tipo che condivide un tipo meno comune con gli elementi dell'array.

Tipo restituito

La funzione ARRAY_REMOVE restituisce il tipo di risultato corrispondente al tipo dell'array. Se l'elemento da rimuovere è`NULL`, il risultato è. `NULL`

Esempi

In questo esempio, la funzione ARRAY_REMOVE prende l'array `[1, 2, 3, null, 3]` e rimuove tutte le occorrenze del valore 3. L'output risultante è l'array. `[1, 2, null]`

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

```
 [1,2,null]
```

Funzione ARRAY_UNION

La funzione ARRAY_UNION accetta due array come argomenti e restituisce un nuovo array che contiene gli elementi unici di entrambi gli array di input. Questa funzione è utile quando è necessario combinare due array ed eliminare eventuali elementi duplicati. Ciò può essere utile in scenari in cui è necessario eseguire operazioni simili a set sugli array, ad esempio trovare l'unione tra due set di dati.

Sintassi

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

Argomenti

matrice 1

Un ARRAY.

matrice 2

Un ARRAY dello stesso tipo di array1.

Tipo restituito

La funzione ARRAY_UNION restituisce un ARRAY dello stesso tipo di array.

Esempio

In questo esempio, il primo array `[1, 2, 3]` contiene gli elementi 1, 2 e 3. Il secondo array `[1, 3, 5]` contiene gli elementi 1, 3 e 5. La funzione ARRAY_UNION combina gli elementi unici di entrambi gli array, dando origine all'array di output. `[1, 2, 3, 5]` T

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

Funzione EXPLODE

La funzione EXPLODE viene utilizzata per trasformare una singola riga con una matrice o una colonna di mappa in più righe, in cui ogni riga corrisponde a un singolo elemento dell'array o della mappa.

Sintassi

```
explode(expr)
```

Argomenti

expr

Un'espressione di matrice o un'espressione cartografica.

Tipo restituito

La funzione EXPLODE restituisce un set di righe, in cui ogni riga rappresenta un singolo elemento della matrice o della mappa di input.

Il tipo di dati delle righe di output dipende dal tipo di dati degli elementi nella matrice o nella mappa di input.

Esempi

L'esempio seguente prende l'array a riga singola [10, 20] e lo trasforma in due righe separate, ciascuna contenente uno degli elementi dell'array (10 e 20).

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

Nel primo esempio, l'array di input è stato passato direttamente come argomento a. `explode()` In questo esempio, l'array di input viene specificato utilizzando la => sintassi, in cui viene fornito esplicitamente il nome della colonna (`collection`).

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

Entrambi gli approcci sono validi e consentono di ottenere lo stesso risultato, ma la seconda sintassi può essere più utile quando è necessario esplodere una colonna da un set di dati più grande, anziché un semplice array letterale.

Funzione FLATTEN

La funzione FLATTEN viene utilizzata per «appiattire» una struttura a matrice annidata in una singola matrice piatta.

Sintassi

```
flatten(arrayOfArrays)
```

Argomenti

arrayOfArrays

Una serie di array.

Tipo restituito

La funzione FLATTEN restituisce un array.

Esempio

In questo esempio, l'input è un array annidato con due array interni e l'output è un singolo array flat contenente tutti gli elementi degli array interni. La funzione FLATTEN prende l'array annidato `[[1, 2], [3, 4]]` e combina tutti gli elementi in un unico array. `[1, 2, 3, 4]`

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

Espressioni condizionali

In SQL, le espressioni condizionali vengono utilizzate per prendere decisioni in base a determinate condizioni. Consentono di controllare il flusso delle istruzioni SQL e restituire valori diversi o eseguire azioni diverse in base alla valutazione di una o più condizioni.

AWS Clean Rooms supporta le seguenti espressioni condizionali:

Argomenti

- Espressione condizionale CASE
- COALESCEespressione
- espressione MASSIMA e MINIMA
- Espressione IF
- espressione IS_NULL

- espressione IS_NOT_NULL
- Funzioni NVL e COALESCE
- NVL2 funzione
- Funzione NULLIF

Espressione condizionale CASE

L'espressione CASE è un'espressione condizionale, simile alle if/then/else istruzioni presenti in altre lingue. CASE è utilizzata per specificare un risultato quando ci sono condizioni multiple. Usa CASE quando un'espressione SQL è valida, ad esempio in un comando SELECT.

Esistono due tipi di espressioni CASE: semplici e ricercate.

- Nelle espressioni CASE semplici, un'espressione viene confrontata con un valore. Quando viene trovata una corrispondenza, viene applicata l'azione specificata nella clausola THEN. Se non viene trovata una corrispondenza, viene applicata l'azione nella clausola ELSE.
- Nelle espressioni CASE cercate, ogni CASE viene valutata in base a un'espressione booleana e l'istruzione CASE restituisce la prima CASE corrispondente. Se non vengono trovate corrispondenze tra le clausole WHEN, viene restituita l'operazione nella clausola ELSE.

Sintassi

Semplice istruzione CASE usata per abbinare le condizioni:

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

Istruzione CASE ricercata usata per valutare ogni condizione:

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

Argomenti

espressione

Un nome di colonna o qualsiasi espressione valida.

value

Valore con cui viene confrontata l'espressione, ad esempio una costante numerica o una stringa di caratteri.

result

Il valore o espressione di destinazione che viene restituito quando viene valutata un'espressione o una condizione booleana. I tipi di dati di tutte le espressioni dei risultati devono essere convertibili in un singolo tipo di output.

condizione

Un'espressione booleana che restituisce true o false. Se la condizione è true, il valore dell'espressione CASE è il risultato che segue la condizione e il resto dell'espressione CASE non viene elaborato. Se la condizione è false, vengono valutate tutte le clausole WHEN successive. Se nessun risultato della condizione WHEN è true, il valore dell'espressione CASE è il risultato della clausola ELSE. Se la clausola ELSE viene omessa e nessuna condizione è true, il risultato è null.

Esempi

Utilizzare una semplice espressione CASE per sostituire `New York City` con `Big Apple` in una query sulla tabella VENUE. Sostituire tutti gli altri nomi di città con `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
...
```

Utilizzare un'espressione CASE ricercata per assegnare numeri di gruppo in base al valore PRICEPAID per le vendite di biglietti singoli:

```
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
...
```

COALESCEespressione

Un'COALESCEespressione restituisce il valore della prima espressione dell'elenco che non è nulla. Se tutte le espressioni sono null, il risultato è null. Quando viene trovato un valore non null, le espressioni rimanenti nell'elenco non vengono valutate.

Questo tipo di espressione è utile quando si desidera restituire un valore di backup per qualcosa quando il valore preferito è mancante o null. Ad esempio, una query può restituire uno dei tre numeri di telefono (cellulare, casa o lavoro, in tale ordine), a seconda di quale si trova prima nella tabella (non null).

Sintassi

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

Esempi

Applica COALESCE l'espressione a due colonne.

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

Il nome di colonna predefinito per un'espressione NVL è. COALESCE La seguente query restituisce gli stessi risultati.

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

espressione MASSIMA e MINIMA

Restituisce il valore più grande o più piccolo da un elenco numeri di espressioni.

Sintassi

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

Parametri

expression_list

Un elenco di espressioni separate da virgole, come ad esempio i nomi di colonne. Le espressioni devono essere tutte convertibili in un tipo di dati comune. I valori NULL nell'elenco vengono ignorati. Se tutte le espressioni vengono valutate su NULL, il risultato è NULL.

Valori restituiti

Restituisce il valore massimo (per GREATEST) o minimo (per LEAST) dell'elenco di espressioni fornito.

Esempio

Nell'esempio seguente viene restituito il valore più alto in ordine alfabetico per `firstname` oppure `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)
```

Espressione IF

La funzione condizionale IF restituisce uno dei due valori in base a una condizione.

Questa funzione è un'istruzione di flusso di controllo comune utilizzata in SQL per prendere decisioni e restituire valori diversi in base alla valutazione di una condizione. È utile per implementare una semplice logica if-else all'interno di una query.

Sintassi

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

Argomenti

espr (1)

La condizione o l'espressione che viene valutata. In caso affermativo`true`, la funzione restituirà il valore di expr2. Se expr1 è*`false`*, la funzione restituirà il valore di expr3.

espr (2)

L'espressione che viene valutata e restituita se expr1 è. `true`

expr 3

L'espressione che viene valutata e restituita se expr1 è. `false`

Valori restituiti

Se `expr1` restituisce a`true`, restituisce; in caso contrario restituisce`expr2`. `expr3`

Esempio

L'esempio seguente utilizza la `if()` funzione per restituire uno dei due valori in base a una condizione. La condizione da valutare è`1 < 2`, ovvero`true`, quindi `'a'` viene restituito il primo valore.

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

espressione IS_NULL

L'espressione IS_NULL condizionale viene utilizzata per verificare se un valore è nullo.

Questa espressione è sinonimo di. IS NULL

Sintassi

```
is_null(expr)
```

Argomenti

expr

Un'espressione di qualsiasi tipo.

Valori restituiti

L'espressione IS_NULL condizionale restituisce un valore booleano. Se `expr1` è NULL, restituisce, altrimenti restituisce`true`. `false`

Esempi

L'esempio seguente verifica se il valore 1 è null e restituisce il risultato booleano `true` perché 1 è un valore valido e non nullo.

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

L'esempio seguente seleziona la `id` colonna dalla `squirrels` tabella, ma solo per le righe in cui si trova la colonna dell'età. `null`

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

espressione IS_NOT_NULL

L'espressione IS_NOT_NULL condizionale viene utilizzata per verificare se un valore non è nullo.

Questa espressione è sinonimo di. IS NOT NULL

Sintassi

```
is_not_null(expr)
```

Argomenti

expr

Un'espressione di qualsiasi tipo.

Valori restituiti

L'espressione IS_NOT_NULL condizionale restituisce un valore booleano. Se non `expr1` è NULL, restituisce, altrimenti restituisce`true`. `false`

Esempi

L'esempio seguente verifica se il valore non 1 è nullo e restituisce il risultato booleano `true` perché 1 è un valore valido e non nullo.

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

L'esempio seguente seleziona la `id` colonna dalla `squirrels` tabella, ma solo per le righe in cui non è presente la colonna dell'età. `null`

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

Funzioni NVL e COALESCE

Restituisce il valore della prima espressione che non è null in una serie di espressioni. Quando viene trovato un valore non null, le restanti espressioni nell'elenco non vengono valutate.

NVL è identica a COALESCE. Sono sinonimi. Questo argomento illustra la sintassi e contiene esempi per entrambe.

Sintassi

```
NVL( expression, expression, ... )
```

La sintassi di COALESCE è la stessa:

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

Se tutte le espressioni sono null, il risultato è null.

Queste funzioni sono utili quando si desidera restituire un valore secondario quando manca un valore primario o è null. Ad esempio, una query potrebbe restituire il primo dei tre numeri di telefono disponibili: cellulare, casa o ufficio. L'ordine delle espressioni nella funzione determina l'ordine di valutazione.

Argomenti

espressione

Un'espressione, come ad esempio un nome di colonna, da valutare per lo stato null.

Tipo restituito

AWS Clean Rooms determina il tipo di dati del valore restituito in base alle espressioni di input. Se i tipi di dati delle espressioni di input non hanno un tipo comune, viene restituito un errore.

Esempi

Se l'elenco contiene espressioni intere, la funzione restituisce un numero intero.

```
SELECT COALESCE(NULL, 12, NULL);

coalesce
--------------
12
```

Questo esempio, che è uguale all'esempio precedente tranne per il fatto che utilizza NVL, restituisce lo stesso risultato.

```
SELECT NVL(NULL, 12, NULL);

coalesce
--------------
12
```

Nell'esempio seguente viene restituito un tipo di stringa.

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', NULL);

coalesce
--------------
AWS Clean Rooms
```

L'esempio seguente genera un errore perché i tipi di dati variano nell'elenco delle espressioni. In questo caso, nell'elenco sono presenti sia un tipo di stringa che un tipo numerico.

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', 12);
ERROR: invalid input syntax for integer: "AWS Clean Rooms"
```

NVL2 funzione

Restituisce uno dei due valori in base al fatto che un'espressione specificata valuti in NULL o NOT NULL.

Sintassi

```
NVL2 ( expression, not_null_return_value, null_return_value )
```

Argomenti

espressione

Un'espressione, come ad esempio un nome di colonna, da valutare per lo stato null.

not_null_return_value

Il valore restituito se l' espressione valuta in NOT NULL. Il valore not_null_return_value deve avere lo stesso tipo di dati dell' espressione o essere implicitamente convertibile in quel tipo di dati.

null_return_value

Il valore restituito se l' espressione valuta in NULL. Il valore null_return_value deve avere lo stesso tipo di dati dell' espressione o essere implicitamente convertibile in quel tipo di dati.

Tipo restituito

Il tipo di NVL2 restituzione è determinato come segue:

- Se not_null_return_value o null_return_value è null, viene restituito il tipo di dati dell'espressione not null.

Se sia not_null_return_value sia null_return_value sono non null:

- Se sia not_null_return_value sia null_return_value hanno lo stesso tipo di dati, viene restituito quel tipo di dati.
- Se not_null_return_value e null_return_value hanno tipi di dati numerici diversi, viene restituito il tipo di dati numerico compatibile più piccolo.
- Se not_null_return_value e null_return_value hanno tipi di dati di datetime diversi, viene restituito un tipo di dati numerici di timestamp.
- Se not_null_return_value e null_return_value hanno diversi tipi di dati di carattere, viene restituito il tipo di dati di not_null_return_value.
- Se not_null_return_value e null_return_value hanno tipi di dati numerici e non numerici misti viene restituito il tipo di dati di not_null_return_value.

> **Important**
>
> Negli ultimi due casi in cui viene restituito il tipo di dati di not_null_return_value, null_return_value viene assegnato implicitamente a quel tipo di dati. Se i tipi di dati non sono compatibili, la funzione ha esito negativo.

Note per l'utilizzo

Infatti NVL2, il valore restituito avrà il valore del parametro not_null_return_value o null_return_value, a seconda di quale sia selezionato dalla funzione, ma avrà il tipo di dati not_null_return_value.

Ad esempio, supponendo che la colonna 1 sia NULL, le seguenti query restituiranno lo stesso valore. Tuttavia, il tipo di dati del valore restituito NVL2 DECODE sarà INTEGER e il tipo di dati del valore restituito sarà VARCHAR.

```
select decode(column1, null, 1234, '2345');
select nvl2(column1, '2345', 1234);
```

Esempio

L'esempio seguente modifica alcuni dati di esempio, quindi valuta due campi per fornire informazioni di contatto appropriate per gli utenti:

```
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
```

Funzione NULLIF

Confronta due argomenti e restituisce null se gli argomenti sono uguali. Se non sono uguali, viene restituito il primo argomento.

Sintassi

L'espressione NULLIF confronta due argomenti e restituisce null se gli argomenti sono uguali. Se non sono uguali, viene restituito il primo argomento. Questa espressione è l'inverso dell'espressione NVL o COALESCE.

```
NULLIF ( expression1, expression2 )
```

Argomenti

expression1, expression2

Le colonne o le espressioni di destinazione che vengono confrontate. Il tipo di restituzione è uguale al tipo della prima espressione.

Esempi

Nell'esempio seguente, la query restituisce la stringa `first` perché gli argomenti non sono uguali.

```
SELECT NULLIF('first', 'second');

case
-------
first
```

Nell'esempio seguente, la query restituisce `NULL` perché gli argomenti della stringa letterale non sono uguali.

```
SELECT NULLIF('first', 'first');

case
-------
NULL
```

Nell'esempio seguente, la query restituisce 1 perché gli argomenti del numero intero non sono uguali.

```
SELECT NULLIF(1, 2);

case
-------
1
```

Nell'esempio seguente, la query restituisce NULL perché gli argomenti del numero intero sono uguali.

```
SELECT NULLIF(1, 1);

case
-------
NULL
```

Nell'esempio seguente, la query restituisce null quando i valori LISTID e SALESID corrispondono:

```
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)
```

Funzioni costruttore

Una funzione di costruzione SQL è una funzione utilizzata per creare nuove strutture di dati, come matrici o mappe.

Prendono alcuni valori di input e restituiscono un nuovo oggetto della struttura dati. Le funzioni di costruzione in genere prendono il nome dal tipo di dati che creano, ad esempio ARRAY o MAP.

Le funzioni di costruzione sono diverse dalle funzioni scalari o dalle funzioni aggregate, che operano su dati esistenti e restituiscono un singolo valore. Le funzioni di costruzione vengono utilizzate per creare nuove strutture di dati che possono quindi essere utilizzate per ulteriori elaborazioni o analisi dei dati.

AWS Clean Rooms supporta le seguenti funzioni di costruzione:

Argomenti

funzione di costruzione MAP

La funzione di costruzione MAP crea una mappa con le coppie chiave/valore specificate.

Le funzioni di costruzione come MAP sono utili quando è necessario creare nuove strutture di dati a livello di codice all'interno delle query SQL. Consentono di creare strutture di dati complesse che possono essere utilizzate per ulteriori elaborazioni o analisi dei dati.

Sintassi

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

Argomenti

chiave 0

Un'espressione di qualsiasi tipo comparabile. Tutte le key0 devono condividere un tipo meno comune.

valore 0

Un'espressione di qualsiasi tipo. Tutti i ValueN devono condividere un tipo meno comune.

Valori restituiti

La funzione MAP restituisce un MAP con tasti digitati come tipo meno comune di key0 e valori digitati come tipo meno comune di value0.

Esempi

L'esempio seguente crea una nuova mappa con due coppie chiave-valore: La chiave è associata al valore. `1.0 '2'` La chiave `3.0` è associata al valore. `'4'` La mappa risultante viene quindi restituita come output dell'istruzione SQL.

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

funzione di costruzione NAMED_STRUCT

La funzione di costruzione NAMED_STRUCT crea una struttura con i nomi e i valori dei campi specificati.

Le funzioni di costruzione come NAMED_STRUCT sono utili quando è necessario creare nuove strutture di dati a livello di codice all'interno delle query SQL. Consentono di creare strutture di dati complesse, come strutture o record, che possono essere utilizzate per ulteriori elaborazioni o analisi dei dati.

Sintassi

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

Argomenti

nome1

Un campo di denominazione letterale STRING 1.

val1

Un'espressione di qualsiasi tipo che specifica il valore per il campo 1.

Valori restituiti

La funzione NAMED_STRUCT restituisce una struttura con il campo 1 corrispondente al tipo di val1.

Esempi

L'esempio seguente crea una nuova struttura con tre campi denominati: Al campo `"a"` viene assegnato il valore. `1` Al campo `"b"` viene assegnato il valore `2.` Al campo `"c"` viene assegnato il valore`3`. La struttura risultante viene quindi restituita come output dell'istruzione SQL.

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

funzione di costruzione STRUCT

La funzione di costruzione STRUCT crea una struttura con i valori di campo specificati.

Le funzioni di costruzione come STRUCT sono utili quando è necessario creare nuove strutture di dati a livello di codice all'interno delle query SQL. Consentono di creare strutture di dati complesse, come strutture o record, che possono essere utilizzate per ulteriori elaborazioni o analisi dei dati.

Sintassi

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

Argomenti

col1

Un nome di colonna o qualsiasi espressione valida.

Valori restituiti

La funzione STRUCT restituisce una struttura con field1 corrispondente al tipo di expr1.

Se gli argomenti sono denominati riferimenti, i nomi vengono utilizzati per denominare il campo. Altrimenti, i campi sono denominati colN, dove N è la posizione del campo nella struttura.

Esempi

L'esempio seguente crea una nuova struttura con tre campi: Al primo campo viene assegnato il valore 1. Al secondo campo viene assegnato il valore 2. Al terzo campo viene assegnato il valore 3. Per impostazione predefinita, i campi nella struttura risultante sono `col1` denominati e `col3` in base alla loro posizione nell'elenco degli argomenti. `col2` La struttura risultante viene quindi restituita come output dell'istruzione SQL.

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

Funzioni di formattazione del tipo di dati

Utilizzando una funzione di formattazione dei tipi di dati, è possibile convertire i valori da un tipo di dati a un altro. Per ognuna di queste funzioni, il primo argomento è sempre il valore da formattare e il secondo argomento contiene il modello per il nuovo formato.

AWS Clean Rooms Spark SQL supporta diverse funzioni di formattazione dei tipi di dati.

Argomenti

BASE64 funzione

La BASE64 funzione converte un'espressione in una stringa base 64 utilizzando la codifica di trasferimento RFC2045 Base64 per MIME.

Sintassi

```
base64(expr)
```

Arguments (Argomenti)

expr

Un'espressione BINARY o una STRINGA che la funzione interpreterà come BINARY.

Tipo restituito

`STRING`

Esempio

Per convertire l'input di stringa specificato nella sua rappresentazione codificata Base64. usa il seguente esempio. Il risultato è la rappresentazione codificata in Base64 della stringa di input 'Spark SQL', che è 'U3BHCMsGU1FM'.

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

Funzione CAST

La funzione CAST converte un tipo di dati in un altro tipo di dati compatibile. Ad esempio, puoi convertire una stringa in una data o un tipo numerico in una stringa. CAST esegue una conversione in fase di runtime, il che significa che la conversione non modifica il tipo di dati di un valore in una tabella di origine. Viene modificato solo nel contesto della query.

Alcuni tipi di dati richiedono una conversione esplicita in altri tipi di dati utilizzando la funzione CAST. Altri tipi di dati possono essere convertiti implicitamente, come parte di un altro comando, senza utilizzare CAST. Per informazioni, consulta Conversione e compatibilità dei tipi.

Sintassi

Utilizza i seguenti due formati di sintassi equivalenti per convertire espressioni da un tipo di dati a un altro:

```
CAST ( expression AS type )
```

Arguments (Argomenti)

espressione

Un'espressione che valuta uno o più valori, ad esempio un nome di colonna o un letterale. La conversione di valori null restituisce null. L'espressione non può contenere stringhe vuote o vuote.

tipo

Una delle opzioni supportateTipi di dati, ad eccezione dei tipi di dati BINARY e BINARY VARIYING.

Tipo restituito

CAST restituisce il tipo di dati specificato dall'argomento tipo.

> **Note**
>
> AWS Clean Rooms restituisce un errore se si tenta di eseguire una conversione problematica, ad esempio una conversione DECIMAL che perde precisione, come la seguente:
>
> ```
> select 123.456::decimal(2,1);
> ```
>
> o una conversione INTEGER che provoca un eccesso:
>
> ```
> select 12345678::smallint;
> ```

Esempi

Le seguenti due query sono equivalenti. Entrambi assegnano un valore decimale a un integer:

```
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)
```

La seguente query produce un risultato simile. Non richiede dati di esempio per l'esecuzione:

```
select cast(162.00 as integer) as pricepaid;

pricepaid
-----------
162
(1 row)
```

In questo esempio, i valori in una colonna timestamp vengono convertiti come date, il che comporta la rimozione dell'ora da ogni risultato:

```
select cast(saletime as date), salesid
from sales order by salesid limit 10;

 saletime  | salesid
-----------+---------
2008-02-18 |       1
2008-06-06 |       2
2008-06-06 |       3
2008-06-09 |       4
2008-08-31 |       5
2008-07-16 |       6
2008-06-26 |       7
2008-07-10 |       8
2008-07-22 |       9
2008-08-06 |      10

(10 rows)
```

Se non hai usato CAST come illustrato nell'esempio precedente, i risultati includono l'ora: 18-02-2008 02:36:48.

La seguente query converte i dati di caratteri variabili in una data. Non richiede dati di esempio per l'esecuzione.

```
select cast('2008-02-18 02:36:48' as date) as mysaletime;

mysaletime
--------------------
2008-02-18
(1 row)
```

In questo esempio, il casting dei valori in una colonna di data viene eseguito come timestamps:

```
select cast(caldate as timestamp), dateid
from date order by dateid limit 10;

      caldate       | dateid
--------------------+--------
2008-01-01 00:00:00 |   1827
2008-01-02 00:00:00 |   1828
2008-01-03 00:00:00 |   1829
2008-01-04 00:00:00 |   1830
2008-01-05 00:00:00 |   1831
2008-01-06 00:00:00 |   1832
2008-01-07 00:00:00 |   1833
2008-01-08 00:00:00 |   1834
2008-01-09 00:00:00 |   1835
2008-01-10 00:00:00 |   1836

(10 rows)
```

In un caso come nell'esempio precedente, è possibile ottenere un ulteriore controllo sulla formattazione dell'output utilizzando. TO_CHAR

In questo esempio, un integer è assegnato come una stringa di caratteri:

```
select cast(2008 as char(4));

bpchar
--------
2008
```

In questo esempio, un valore DECIMAL(6,3) viene assegnato come valore DECIMAL(4,1):

```
select cast(109.652 as decimal(4,1));

numeric
---------
109.7
```

Questo esempio mostra un'espressione più complessa. La colonna PRICEPAID (una colonna DECIMAL(8,2)) nella tabella SALES viene convertita in una colonna DECIMAL(38,2) e i valori vengono moltiplicati per 100000000000000000000:

```
select salesid, pricepaid::decimal(38,2)*100000000000000000000
as value from sales where salesid<10 order by salesid;


 salesid |           value
---------+----------------------------
       1 | 728000000000000000000000.00
       2 |  760000000000000000000000.00
       3 | 350000000000000000000000.00
       4 | 175000000000000000000000.00
       5 | 154000000000000000000000.00
       6 | 394000000000000000000000.00
       7 | 788000000000000000000000.00
       8 | 197000000000000000000000.00
       9 | 591000000000000000000000.00

(9 rows)
```

Funzione DECODE

La funzione DECODE è la controparte della funzione ENCODE, utilizzata per convertire una stringa in un formato binario utilizzando una codifica di caratteri specifica. La funzione DECODE prende i dati binari e li riconverte in un formato di stringa leggibile utilizzando la codifica dei caratteri specificata.

Questa funzione è utile quando è necessario lavorare con dati binari archiviati in un database e presentarli in un formato leggibile dall'uomo o quando è necessario convertire dati tra diverse codifiche di caratteri.

Sintassi

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

Arguments (Argomenti)

expr

Un'espressione BINARY codificata in un set di caratteri.

set di caratteri

Un'espressione STRING.

Codifiche dei set di caratteri supportate (senza distinzione tra maiuscole e minuscole): `'US-ASCII'`,,, `'ISO-8859-1''UTF-8'`, `'UTF-16BE'` e. `'UTF-16LE' 'UTF-16'`

Tipo restituito

La funzione DECODE restituisce una STRING.

Esempio

L'esempio seguente presenta una tabella chiamata `messages` con una colonna chiamata `message_text` che memorizza i dati dei messaggi in un formato binario utilizzando la codifica dei caratteri UTF-8. La funzione DECODE riconverte i dati binari in un formato di stringa leggibile. L'output di questa query è il testo leggibile del messaggio memorizzato nella tabella dei messaggi, con l'ID`123`, convertito dal formato binario in una stringa utilizzando la codifica. `'utf-8'`

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

Funzione ENCODE

La funzione ENCODE viene utilizzata per convertire una stringa nella sua rappresentazione binaria utilizzando una codifica di caratteri specificata.

Questa funzione è utile quando è necessario lavorare con dati binari o quando è necessario eseguire conversioni tra diverse codifiche di caratteri. Ad esempio, è possibile utilizzare la funzione ENCODE per archiviare dati in un database che richiede la memorizzazione binaria o quando è necessario trasferire dati tra sistemi che utilizzano codifiche di caratteri diverse.

Sintassi

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

Arguments (Argomenti)

str

Un'espressione STRING da codificare.

set di caratteri

Un'espressione STRING che specifica la codifica.

Codifiche dei set di caratteri supportate (senza distinzione tra maiuscole e minuscole): `'US-ASCII'`,,,, `e'ISO-8859-1'`. `'UTF-8' 'UTF-16BE' 'UTF-16LE' 'UTF-16'`

Tipo restituito

La funzione ENCODE restituisce un valore BINARY.

Esempio

L'esempio seguente converte la stringa `'abc'` nella sua rappresentazione binaria utilizzando la `'utf-8'` codifica, che in questo caso restituisce la stringa originale. Questo perché la `'utf-8'` codifica è una codifica di caratteri a larghezza variabile che può rappresentare l'intero set di caratteri ASCII (che include le lettere `'a' e'c'`) utilizzando un solo byte per carattere. `'b'` Pertanto, la rappresentazione binaria dell'`'abc'`utilizzo `'utf-8'` è la stessa della stringa originale.

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

Funzione HEX

La funzione HEX converte un valore numerico (un intero o un numero a virgola mobile) nella rappresentazione di stringa esadecimale corrispondente.

L'esadecimale è un sistema numerico che utilizza 16 simboli distinti (0-9 e A-F) per rappresentare valori numerici. È comunemente usato nell'informatica e nella programmazione per rappresentare dati binari in un formato più compatto e leggibile dall'uomo.

Sintassi

```
hex(expr)
```

Arguments (Argomenti)

expr

Un'espressione BIGINT, BINARY o STRING.

Tipo restituito

HEX restituisce una STRING. La funzione restituisce la rappresentazione esadecimale dell'argomento.

Esempio

L'esempio seguente prende come input il valore intero 17 e vi applica la funzione HEX (). L'output è11, che è la rappresentazione esadecimale del valore di input. 17

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

L'esempio seguente converte la stringa `'Spark_SQL'` nella sua rappresentazione esadecimale. L'output è`537061726B2053514C`, che è la rappresentazione esadecimale della stringa di input. `'Spark_SQL'`

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

In questo esempio, la stringa 'Spark_SQL' viene convertita come segue:

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

La concatenazione di questi valori esadecimali produce l'output finale ". 537061726B2053514C"

Funzione STR_TO_MAP

La funzione STR_TO_MAP è una funzione di conversione. string-to-map Converte una rappresentazione in formato stringa di una mappa (o dizionario) in una vera struttura di dati cartografici.

Questa funzione è utile quando è necessario lavorare con strutture di dati cartografiche in SQL, ma i dati vengono inizialmente memorizzati come stringa. Convertendo la rappresentazione della stringa in una mappa effettiva, è quindi possibile eseguire operazioni e manipolazioni sui dati della mappa.

Sintassi

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

Arguments (Argomenti)

testo

Un'espressione STRING che rappresenta la mappa.

PairDelim

Un valore letterale STRING opzionale che specifica come separare le voci. Il valore predefinito è una virgola (). `','`

keyValueDelim

Un valore letterale STRING opzionale che specifica come separare ogni coppia chiave-valore. Il valore predefinito è i due punti (). `':'`

Tipo restituito

La funzione STR_TO_MAP restituisce una MAP di STRING sia per le chiavi che per i valori. Sia PairDelim che vengono trattati come espressioni regolari keyValueDelim.

Esempio

L'esempio seguente prende la stringa di input e i due argomenti di delimitazione e converte la rappresentazione della stringa in una vera struttura di dati della mappa. In questo esempio specifico, la stringa di input `'a:1,b:2,c:3'` rappresenta una mappa con le seguenti coppie chiave-valore: `'a'` è la chiave e è il valore. `'1' 'b'`è la chiave ed `'2'` è il valore. `'c'`è la chiave ed `'3'` è il valore.

Il `','` delimitatore viene utilizzato per separare le coppie chiave-valore e il `':'` delimitatore viene utilizzato per separare la chiave e il valore all'interno di ciascuna coppia. L'output di questa query è:. `{"a":"1","b":"2","c":"3"}` Questa è la struttura dei dati della mappa risultante, dove le chiavi sono `'a' 'b''c'`, e, e i valori corrispondenti sono `'1''2'`, e`'3'`.

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

L'esempio seguente dimostra che la funzione STR_TO_MAP prevede che la stringa di input sia in un formato specifico, con le coppie chiave-valore delimitate correttamente. Se la stringa di input non corrisponde al formato previsto, la funzione tenterà comunque di creare una mappa, ma i valori risultanti potrebbero non essere quelli previsti.

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

TO_CHAR

TO_CHAR converte un timestamp o un'espressione numerica in un formato di dati carattere-stringa.

Sintassi

```
TO_CHAR (timestamp_expression | numeric_expression , 'format')
```

Arguments (Argomenti)

timestamp_expression

Un'espressione che restituisce un valore di tipo TIMESTAMP o TIMESTAMPTZ o un valore che può essere implicitamente costretto in un timestamp.

numeric_expression

Un'espressione che restituisce un valore di tipo di dati numerici un valore che può essere implicitamente costretto in un tipo numerico. Per ulteriori informazioni, consulta Tipi numerici. TO_CHAR inserisce uno spazio a sinistra della stringa numerica.

> **Note**
>
> TO_CHAR non supporta valori DECIMALI a 128 bit.

format

Il formato per il nuovo valore. Per i formati validi, consultare Stringhe di formato datetime e Stringhe di formato numerico.

Tipo restituito

VARCHAR

Esempi

Nell'esempio seguente un timestamp viene convertito in un valore con la data e l'ora in un formato con il nome del mese a nove caratteri, il nome del giorno della settimana e il numero del giorno del mese.

```
select to_char(timestamp '2009-12-31 23:15:59', 'MONTH-DY-DD-YYYY HH12:MIPM');
to_char
-------------------------
DECEMBER -THU-31-2009 11:15PM
```

Nell'esempio seguente un timestamp viene convertito in un valore con il numero di giorno dell'anno.

```
select to_char(timestamp '2009-12-31 23:15:59', 'DDD');

to_char
-------------------------
365
```

Nell'esempio seguente un timestamp viene convertito in un numero di giorno ISO della settimana.

```
select to_char(timestamp '2022-05-16 23:15:59', 'ID');

to_char
-------------------------
1
```

L'esempio seguente estrae il nome del mese da un valore di data.

```
select to_char(date '2009-12-31', 'MONTH');
```

```
to_char
------------------------
DECEMBER
```

Nell'esempio seguente viene convertito ogni valore STARTTIME nella tabella EVENT in una stringa composta da ore, minuti e secondi.

```
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)
```

L'esempio seguente converte un intero valore timestamp in un formato diverso.

```
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)
```

L'esempio seguente converte un letterale di timestamp in una stringa di caratteri.

```
select to_char(timestamp '2009-12-31 23:15:59','HH24:MI:SS');
to_char
----------
23:15:59
(1 row)
```

L'esempio seguente converte un numero in una stringa di caratteri con il segno negativo alla fine.

```
select to_char(-125.8, '999D99S');
```

```
to_char
---------
125.80-
(1 row)
```

L'esempio seguente converte un numero in una stringa di caratteri con il simbolo di valuta.

```
select to_char(-125.88, '$S999D99');
to_char
---------
$-125.88
(1 row)
```

L'esempio seguente converte un numero in una stringa di caratteri con parentesi angolari per i numeri negativi.

```
select to_char(-125.88, '$999D99PR');
to_char
---------
$<125.88>
(1 row)
```

L'esempio seguente converte un numero in una stringa di numeri romani.

```
select to_char(125, 'RN');
to_char
---------
CXXV
(1 row)
```

L'esempio seguente mostra il giorno della settimana.

```
SELECT to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS');
               to_char
-----------------------
Wednesday, 31 09:34:26
```

L'esempio seguente visualizza il suffisso numerico ordinale per un numero.

```
SELECT to_char(482, '999th');
               to_char
```

```
-----------------------
 482nd
```

L'esempio seguente sottrae la commissione dal prezzo pagato nella tabella delle vendite. La differenza viene quindi arrotondata per eccesso e convertita in un numero romano, mostrato nella colonna: `to_char`

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

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

L'esempio seguente aggiunge il simbolo della valuta ai valori di differenza mostrati nella `to_char` colonna:

```
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)
```

L'esempio seguente elenca il secolo in cui è stata effettuata ciascuna vendita.

```
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)
```

Nell'esempio seguente viene convertito ogni valore STARTTIME nella tabella EVENT in una stringa composta da ore, minuti, secondi e fuso orario.

```
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)
```

L'esempio seguente mostra la formattazione per secondi, millisecondi e microsecondi.

```
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
```

Funzione TO_DATE

TO_DATE converte una data rappresentata con una stringa di caratteri in un tipo di dati DATE.

Sintassi

```
TO_DATE (date_str)
```

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

Arguments (Argomenti)

data_str

Una stringa di data o un tipo di dati che può essere inserito in una stringa di date.

format

Una stringa letterale che corrisponde ai modelli datetime di Spark. Per modelli datetime validi, vedi Datetime Patterns for Formatting and Parsing.

Tipo restituito

TO_DATE restituisce una DATA, in base al valore formato.

Se la conversione in formato non riesce, viene restituito un errore.

Esempi

L'istruzione SQL seguente converte la data `02 Oct 2001` in un tipo di dati data.

```
select to_date('02 Oct 2001', 'dd MMM yyyy');

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

L'istruzione SQL seguente converte la stringa 20010631 in una data.

```
select to_date('20010631', 'yyyyMMdd');
```

L'istruzione SQL seguente converte la stringa 20010631 in una data:

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

Il risultato è un valore nullo perché ci sono solo 30 giorni a giugno.

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

TO_NUMBER

TO_NUMBER converte una stringa in un valore numerico (decimale).

Sintassi

```
to_number(string, format)
```

Arguments (Argomenti)

stringa

Stringa da convertire. Il formato deve essere un valore letterale.

format

Il secondo argomento è una stringa di formato che indica come deve essere analizzata la stringa di caratteri per creare il valore numerico. Ad esempio, il formato '99D999' specifica che la stringa da convertire è composta da cinque cifre con il punto decimale nella terza posizione. Ad

esempio, `to_number('12.345','99D999')` restituisce `12.345` come valore numerico. Per un elenco di formati validi, consultare Stringhe di formato numerico.

Tipo restituito

TO_NUMBER restituisce un numero DECIMAL.

Se la conversione in formato non riesce, viene restituito un errore.

Esempi

L'esempio seguente converte la stringa `12,454.8-` in un numero:

```
select to_number('12,454.8-', '99G999D9S');

to_number
-----------
-12454.8
```

L'esempio seguente converte la stringa `$ 12,454.88` in un numero:

```
select to_number('$ 12,454.88', 'L 99G999D99');

to_number
-----------
12454.88
```

L'esempio seguente converte la stringa `$ 2,012,454.88` in un numero:

```
select to_number('$ 2,012,454.88', 'L 9,999,999.99');

to_number
-----------
2012454.88
```

UNBASE64 funzione

La UNBASE64 funzione converte un argomento da una stringa in base 64 a un file binario.

La codifica Base64 viene comunemente utilizzata per rappresentare dati binari (come immagini, file o informazioni crittografate) in un formato testuale sicuro per la trasmissione su vari canali di comunicazione (come e-mail, parametri URL o archiviazione di database).

La UNBASE64 funzione consente di invertire questo processo e ripristinare i dati binari originali. Questo tipo di funzionalità può essere utile in scenari in cui è necessario lavorare con dati codificati in formato Base64, ad esempio durante l'integrazione con sistemi esterni o APIs che utilizzano Base64 come meccanismo di trasferimento dei dati.

Sintassi

```
unbase64(expr)
```

Arguments (Argomenti)

expr

Un'espressione STRING in formato base64.

Tipo restituito

`BINARY`

Esempio

Nell'esempio seguente, la stringa con codifica Base64 `'U3BhcmsgU1FM'` viene riconvertita nella stringa originale. `'Spark SQL'`

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

Funzione UNHEX

La funzione UNHEX riconverte una stringa esadecimale nella sua rappresentazione di stringa originale.

Questa funzione può essere utile in scenari in cui è necessario lavorare con dati archiviati o trasmessi in formato esadecimale e ripristinare la rappresentazione di stringa originale per un'ulteriore elaborazione o visualizzazione.

La funzione UNHEX è la controparte della funzione HEX.

Sintassi

```
unhex(expr)
```

Arguments (Argomenti)

expr

Un'espressione STRING di caratteri esadecimali.

Tipo restituito

UNHEX restituisce un valore BINARIO.

Se la lunghezza di expr è dispari, il primo carattere viene scartato e al risultato viene aggiunto un byte nullo. Se expr contiene caratteri non esadecimali, il risultato è NULL.

Esempio

L'esempio seguente converte una stringa esadecimale nella sua rappresentazione di stringa originale utilizzando insieme le funzioni UNHEX () e DECODE (). La prima parte della query utilizza la funzione UNHEX () per convertire la stringa esadecimale '537061726B2053514C' nella sua rappresentazione binaria. La seconda parte della query utilizza la funzione DECODE () per convertire i dati binari ottenuti dalla funzione UNHEX () in una stringa, utilizzando la codifica dei caratteri 'UTF-8'. L'output della query è la stringa originale 'Spark_SQL' che è stata convertita in formato esadecimale e poi nuovamente in una stringa.

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

Stringhe di formato datetime

È possibile utilizzare i modelli datetime nei seguenti scenari comuni:

- Quando si lavora con fonti di dati CSV e JSON per analizzare e formattare il contenuto datetime
- Durante la conversione tra tipi di stringhe e tipi di data o timestamp utilizzando funzioni come:
  - unix_timestamp
  - date_format
  - to_unix_timestamp
  - da_unixtime
  - to_date
  - to_timestamp

- da_utc_timestamp
- to_utc_timestamp

Utilizza lo schema di lettere nella tabella seguente per l'analisi e la formattazione di data e ora.

Parte di data o parte di ora	Significato	Esempi
a	AM o PM del giorno, presentat a come am-pm	PM
D	Giorno dell'anno, presentato come numero a 3 cifre	189
d	Giorno del mese, presentato come numero a 2 cifre	28
E	Giorno della settimana, presentato come testo	Il Martedì
F	Giorno della settimana allineato nel mese, presentato come numero a 1 cifra	3
G	Indicatore dell'era, presentato come testo	AD Anno Domini
h	Ora del mattino o del pomeriggio, presentata come numero a 2 cifre	12
H	Ora del giorno, presentat a come numero a due cifre compreso tra 0 e 23	0
k	Ora del giorno, presentat a come numero a due cifre compreso tra 1 e 24	1

Parte di data o parte di ora	Significato	Esempi
K	Ora del mattino o del pomeriggio, presentata come numero a due cifre compreso tra 0 e 11	0
m	Minuto dell'ora, presentato come numero a due cifre	30
M/L	Mese dell'anno, presentato come mese	7 07 lug luglio
O	Scostamento di zona localizzato rispetto all'UTC	GMT+8 GMT+ 8:00 UTC- 08:00
Q/q	Trimestre dell'anno, presentato come numero (da 1 a 4) o testo	3 03 D3 3° trimestre
s	Secondo del minuto, presentato come un numero a due cifre	55
S	Frazione di secondo, presentata come frazione	978

Parte di data o parte di ora	Significato	Esempi
V	Identificatore del fuso orario, presentato come ID di fuso orario	America/Los_Angeles Z 08:30
x	Scostamento di zona rispetto a UTC (offset-X)	+0000 -08 -0830 - 08:30 -083015 - 08:30:15
X	scostamento di zona rispetto all'UTC; dove Z sta per zero	Z -08 -0830 - 08:30 -083015 - 08:30:15
y	Anno, presentato come anno	2020 20
z	Nome del fuso orario, presentato come testo	Orario standard Pacifico PST

Parte di data o parte di ora	Significato	Esempi
Z	Scostamento di zona rispetto a UTC (offset-Z)	+0000 -0800 - 08:00
'	Escape per il testo, presentato come delimitatore	N/D
''	Citazione singola, presentata come un valore letterale	'
[	Inizio della sezione opzionale	N/D
]	Fine della sezione opzionale	N/D

Il numero di lettere dello schema determina il tipo di formato:

Formato del testo

- Usa 1-3 lettere per la forma abbreviata (ad esempio, «lun» per lunedì)
- Usa esattamente 4 lettere per il modulo completo (ad esempio, «lunedì»)
- Non utilizzare 5 o più lettere: ciò causerà un errore

Formato numerico (n)

- Il valore n rappresenta il numero massimo di lettere consentito
- Per modelli a lettera singola:
  - L'output utilizza un numero minimo di cifre senza imbottitura
- Per modelli con più lettere:
  - L'output è riempito con zeri in base alla larghezza del conteggio delle lettere
- Durante l'analisi, l'input deve contenere il numero esatto di cifre

Formato numerico/testo

- Per 3 o più lettere, segui le regole del formato del testo
- Per un numero inferiore di lettere, segui le regole del formato numerico

Formato delle frazioni

- Usa 1-9 caratteri «S» (ad esempio, SSSSSS)
- Per l'analisi:
  - Accetta frazioni comprese tra 1 e il numero di caratteri S
- Per la formattazione:
  - Tasto con zeri corrispondente al numero di caratteri S
- Supporta fino a 6 cifre per una precisione al microsecondo
- Può analizzare i nanosecondi ma tronca le cifre aggiuntive

Formato dell'anno

- Il conteggio delle lettere imposta la larghezza minima del campo per il riempimento
- Per due lettere:
  - Stampa le ultime due cifre
  - Analizza gli anni compresi tra il 2000 e il 2099
- Per meno di quattro lettere (tranne due):
  - Mostra il segno solo per gli anni negativi
- Non utilizzare 7 o più lettere: ciò causerà un errore

Formato mensile

- Usa «M» per il modulo standard o «L» per il modulo autonomo
- Singola «M» o «L»:
  - Mostra i numeri dei mesi da 1 a 12 senza imbottitura
- 'MM' o 'LL':
  - Mostra i numeri dei mesi 01-12 con imbottitura
- 'MMM':
  - Mostra il nome abbreviato del mese in formato standard

    - Deve far parte di uno schema di data completo
- 'LLL':
    - Mostra il nome abbreviato del mese in forma autonoma
    - Utilizzare per la formattazione solo per un mese
- 'MMMM':
    - Mostra il nome completo del mese in formato standard
    - Utilizza per date e timestamp
- 'LLLL':
    - Mostra il nome completo del mese in forma autonoma
    - Utilizza per la formattazione solo per un mese

Formati di fuso orario

- am-pm: usa solo 1 lettera
- ID zona (V): utilizza solo 2 lettere
- Nomi delle zone (z):
    - 1-3 lettere: mostra il nome breve
    - 4 lettere: mostra il nome completo
    - Non utilizzare 5 o più lettere

Formati offset

- X e x:
    - 1 lettera: indica l'ora (+01) o l'ora-minuto (+0130)
    - 2 lettere: mostra le ore e i minuti senza i due punti (+0130)
    - 3 lettere: mostra l'ora e i minuti con i due punti (+ 01:30)
    - 4 lettere: mostra hour-minute-second senza due punti (+013015)
    - 5 lettere: mostra hour-minute-second con i due punti (+ 01:30:15)
    - X usa 'Z' per l'offset zero
    - x usa '+00', '+0000' o '+ 00:00 'per l'offset zero
- O:
    - 1 lettera: mostra il formato breve (GMT+8)

  - 4 lettere: mostra il modulo completo (GMT+ 08:00)
- Z:
  - 1-3 lettere: mostra l'ora e il minuto senza i due punti (+0130)
  - 4 lettere: mostra il modulo localizzato completo
  - 5 lettere: mostra hour-minute-second con i due punti

Sezioni opzionali

- Usa le parentesi quadre [] per contrassegnare il contenuto opzionale
- È possibile annidare sezioni opzionali
- Tutti i dati validi vengono visualizzati nell'output
- L'input può omettere intere sezioni opzionali

> **Note**
>
> I simboli 'E', 'F', 'q' e 'Q' funzionano solo per la formattazione della data e dell'ora (come date_format). Non usarli per l'analisi della data e dell'ora (come to_timestamp).

Stringhe di formato numerico

Le seguenti stringhe di formato numerico si applicano a funzioni come TO_NUMBER e TO_CHAR.

- Per esempi di formattazione di stringhe come numeri, consulta TO_NUMBER.
- Per esempi di formattazione di numeri come stringhe, consulta TO_CHAR.

Formato	Description
9	Valore numerico con il numero di cifre specifica to.
0	Valore numerico con zeri iniziali.
. (periodo), D	Punto decimale.

Formato	Description
, (virgola)	Separatore di migliaia.
CC	Codice del secolo. Ad esempio, il 21° secolo è iniziato il 01-01-2001 (supportato solo per TO_CHAR).
FM	Modalità di riempimento. Eliminare spazi vuoti e zeri.
PR	I valore negativo tra parentesi angolari.
S	Segno ancorato a un numero.
L	Simbolo di valuta nella posizione specificata.
G	Separatore di gruppo.
MI	Segno meno nella posizione specificata per numeri inferiori a 0.
PL	Segno più nella posizione specificata per numeri superiori a 0.
SG	Segno più o meno nella posizione specificata.
RN	Numero romano compreso tra 1 e 3999 (supportato solo per TO_CHAR).
TH o th	Suffisso del numero ordinale. Non converte numeri frazionari o valori inferiori a zero.

Funzioni di data e ora

Le funzioni di data e ora consentono di eseguire un'ampia gamma di operazioni sui dati di data e ora, ad esempio estrarre parti di una data, eseguire calcoli di date, formattare date e ore e lavorare con la data e l'ora correnti. Queste funzioni sono essenziali per attività quali l'analisi dei dati, la creazione di report e la manipolazione dei dati che coinvolgono dati temporali.

AWS Clean Rooms supporta le seguenti funzioni di data e ora:

Argomenti

Funzione ADD_MONTHS

ADD_MONTHS aggiunge il numero di mesi specificato a un valore o espressione di data o timestamp. La funzione DATE_ADD fornisce una funzionalità simile.

Sintassi

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

Arguments (Argomenti)

date | timestamp

Un'espressione o una colonna data o timestamp che viene implicitamente convertita in una data o un timestamp. Se la data è l'ultimo giorno del mese o se il mese risultante è più corto, la funzione restituisce l'ultimo giorno del mese nel risultato. Per le altre date, il risultato contiene lo stesso numero di giorni dell'espressione di data.

integer

Un integer positivo o negativo. Utilizza un numero negativo per sottrarre mesi dalle date.

Tipo restituito

TIMESTAMP

Esempio

La query seguente utilizza la funzione ADD_MONTHS in una funzione TRUNC. La funzione TRUNC rimuove l'ora del giorno dal risultato di ADD_MONTHS. La funzione ADD_MONTHS aggiunge 12 mesi a ogni valore della colonna 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)
```

Gli esempi seguenti illustrano il comportamento quando la funzione ADD_MONTHS opera su date con mesi che hanno un numero di giorni differente.

```
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)
```

Funzione CONVERT_TIMEZONE

CONVERT_TIMEZONE converte un timestamp da un fuso orario a un altro. La funzione si regola automaticamente in base all'ora legale.

Sintassi

```
CONVERT_TIMEZONE ( ['source_timezone',] 'target_timezone', 'timestamp')
```

Arguments (Argomenti)

source_timezone

(Facoltativo) Il fuso orario del timestamp corrente. Il valore predefinito è UTC.

target_timezone

Il fuso orario del nuovo timestamp.

timestamp

Una colonna timestamp o un'espressione che viene implicitamente convertita in un timestamp.

Tipo restituito

TIMESTAMP

Esempi

L'esempio seguente converte il valore di timestamp dal fuso orario UTC predefinito in PST.

```
select convert_timezone('PST', '2008-08-21 07:23:54');

 convert_timezone
------------------------
2008-08-20 23:23:54
```

L'esempio seguente converte il valore di timestamp nella colonna LISTTIME dal fuso orario UTC predefinito in PST. Anche se il timestamp rientra nel periodo dell'ora legale, viene convertito nell'ora standard in quanto il fuso orario di destinazione è specificato come abbreviazione (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
```

L'esempio seguente converte una colonna LISTTIME con timestamp dal fuso orario UTC predefinito al fuso orario. US/Pacific Il fuso orario di destinazione utilizza un nome di fuso orario e il timestamp è nel periodo dell'ora legale, di conseguenza la funzione restituisce l'ora legale.

```
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
```

L'esempio seguente converte una stringa di timestamp da EST a PST:

```
select convert_timezone('EST', 'PST', '20080305 12:25:29');

 convert_timezone
-------------------
2008-03-05 09:25:29
```

L'esempio seguente converte un timestamp all'ora standard degli Stati Uniti orientali in quanto il fuso orario di destinazione utilizza un nome di fuso orario (America/New_York) e il timestamp si trova nel periodo dell'ora standard.

```
select convert_timezone('America/New_York', '2013-02-01 08:00:00');

 convert_timezone
---------------------
2013-02-01 03:00:00
(1 row)
```

L'esempio seguente converte il timestamp nell'ora legale dell'est degli Stati Uniti in quanto il fuso orario target utilizza un nome di fuso orario (America/New_York) e il timestamp si trova nel periodo dell'ora legale.

```
select convert_timezone('America/New_York', '2013-06-01 08:00:00');

 convert_timezone
---------------------
2013-06-01 04:00:00
(1 row)
```

L'esempio seguente illustra l'utilizzo di 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)
```

Funzione CURRENT_DATE

CURRENT_DATE restituisce una data nel fuso orario della sessione corrente (UTC per impostazione predefinita) nel formato predefinito:. YYYY-MM-DD

> **Note**
>
> CURRENT_DATE restituisce la data di inizio della transazione corrente e non dell'istruzione corrente. Considera lo scenario quando avvii una transazione contenente più istruzioni alle 23:59 del giorno 01/10/08 e l'istruzione contenente CURRENT_DATE viene eseguita alle 00:00 del 02/10/08. CURRENT_DATE restituisce `10/01/08`, non `10/02/08`.

Sintassi

```
CURRENT_DATE
```

Tipo restituito

DATE

Esempio

L'esempio seguente restituisce la data corrente (nel punto in Regione AWS cui viene eseguita la funzione).

```
select current_date;

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

Funzione CURRENT_TIMESTAMP

CURRENT_TIMESTAMP restituisce la data e l'ora correnti, incluse la data, l'ora e (facoltativamente) i millisecondi o i microsecondi.

Questa funzione è utile quando è necessario ottenere la data e l'ora correnti, ad esempio per registrare il timestamp di un evento, eseguire calcoli basati sul tempo o popolare le colonne. date/time

Sintassi

```
current_timestamp()
```

Tipo restituito

La funzione CURRENT_TIMESTAMP restituisce un valore DATE.

Esempio

L'esempio seguente restituisce la data e l'ora correnti nel momento in cui viene eseguita la query, ovvero il 25 aprile 2020 alle 15:49:11.914 (3:49:11.914 PM).

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

L'esempio seguente recupera la data e l'ora correnti per ogni riga della tabella. `squirrels`

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

Funzione DATE_ADD

Restituisce la data corrispondente a num_days dopo start_date.

Sintassi

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

Arguments (Argomenti)

data_iniziale

Il valore della data di inizio.

num_days

Il numero di giorni da aggiungere (numero intero). Un numero positivo aggiunge giorni, un numero negativo sottrae giorni.

Tipo restituito

DATE

Esempi

L'esempio seguente aggiunge un giorno a una data:

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

Result:
2016-07-31
```

L'esempio seguente aggiunge più giorni.

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

Result:
2016-08-04
```

Note per l'utilizzo

Questa documentazione riguarda la funzione DATE_ADD di Spark SQL, che fornisce un'interfaccia più semplice per aggiungere giorni alle date rispetto ad altre varianti SQL. Per aggiungere altri intervalli come mesi o anni, potrebbero essere necessarie funzioni diverse.

Funzione DATE_DIFF

DATE_DIFF restituisce la differenza tra le parti relative alla data di due espressioni di data o ora.

Sintassi

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

Arguments (Argomenti)

endDate

Un'espressione DATE.

startDate

Un'espressione DATE.

Tipo restituito

BIGINT

Esempi con una colonna DATE

Nell'esempio seguente viene rilevata la differenza in numero di settimane tra due valori di data letterali.

```
select date_diff(week,'2009-01-01','2009-12-31') as numweeks;

numweeks
----------
52
(1 row)
```

Nell'esempio seguente viene rilevata la differenza in ore tra due valori di data letterali. Quando non si fornisce il valore dell'ora per una data, il valore predefinito è 00:00:00.

```
select date_diff(hour, '2023-01-01', '2023-01-03 05:04:03');

date_diff
----------
53
(1 row)
```

Nell'esempio seguente viene rilevata la differenza in giorni tra due valori TIMESTAMETZ letterali.

```
Select date_diff(days, 'Jun 1,2008  09:59:59 EST', 'Jul 4,2008  09:59:59 EST')

date_diff
----------
33
```

Nell'esempio seguente viene rilevata la differenza, in giorni, tra due date nella stessa riga di una tabella.

```
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)
```

Nel seguente esempio viene trovata la differenza, in numero di trimestri, tra un valore letterale nel passato e la data odierna. Questo esempio presuppone che la data corrente è il 5 giugno 2008. È possibile assegnare un nome completo o abbreviato alle parti di data. Il nome di colonna predefinito per la funzione DATE_DIFF è DATE_DIFF.

```
select date_diff(qtr, '1998-07-01', current_date);

date_diff
-----------
40
(1 row)
```

In questo esempio viene eseguito il join delle tabelle SALES e LISTING per calcolare quanti giorni dopo la pubblicazione sono stati venduti i biglietti per i risultati da 1000 a 1005. L'attesa più lunga per la vendita di questi elenchi è di 15 giorni e quella più breve è inferiore a 1 giorno (0 giorni).

```
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)
```

Questo esempio calcola il numero medio di ore di attesa dei venditori per tutte le vendite di biglietti.

```
select avg(date_diff(hours, listtime, saletime)) as avgwait
from sales, listing
where sales.listid = listing.listid;

avgwait
---------
465
(1 row)
```

Esempi con una colonna TIME

La tabella di esempio seguente TIME_TEST contiene una colonna TIME_VAL (tipo TIME) con tre valori inseriti.

```
select time_val from time_test;

time_val
--------------------
20:00:00
00:00:00.5550
00:58:00
```

Nell'esempio seguente viene rilevata la differenza di numero di ore tra la colonna TIME_VAL e un valore letterale temporale.

```
select date_diff(hour, time_val, time '15:24:45') from time_test;

 date_diff
-----------
        -5
        15
        15
```

Nell'esempio seguente viene rilevata la differenza in numero di minuti tra due valori letterali temporali.

```
select date_diff(minute, time '20:00:00', time '21:00:00') as nummins;

nummins
----------
60
```

Esempi con una colonna TIMETZ

La tabella di esempio seguente TIMETZ_TEST contiene una colonna TIMETZ_VAL (tipo TIMETZ) con tre valori inseriti.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

Nell'esempio seguente vengono individuate le differenze nel numero di ore, tra un letterale TIMETZ e timetz_val.

```
select date_diff(hours, timetz '20:00:00 PST', timetz_val) as numhours from
 timetz_test;

numhours
----------
0
-4
1
```

Nell'esempio seguente viene rilevata la differenza in numero di ore tra due valori TIMETZ letterali.

```
select date_diff(hours, timetz '20:00:00 PST', timetz '00:58:00 EST') as numhours;

numhours
----------
1
```

Funzione DATE_PART

DATE_PART estrae valori di parte di data da un'espressione. DATE_PART è un sinonimo della funzione PGDATE_PART.

Sintassi

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

Arguments (Argomenti)

campo

Quale parte del sorgente deve essere estratta e i valori di stringa supportati sono gli stessi dei campi della funzione equivalente EXTRACT.

source (origine)

Una colonna DATE o INTERVAL da cui estrarre il campo.

Tipo restituito

Se il campo è 'SECOND', un DECIMAL (8, 6). In tutti gli altri casi, un numero INTERO.

Esempio

L'esempio seguente estrae il giorno dell'anno (DOY) da un valore di data. L'output mostra che il giorno dell'anno per la data «2019-08-12" è. 224 Ciò significa che il 12 agosto 2019 è il 224° giorno dell'anno 2019.

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

Funzione DATE_TRUNC

La funzione DATE_TRUNC tronca un'espressione di timestamp o letterale in base alla parte di data specificata, ad esempio ora, settimana o mese.

Sintassi

```
date_trunc(format, datetime)
```

Arguments (Argomenti)

format

Il formato che rappresenta l'unità in cui troncare. I formati validi sono:

- «YEAR», «YYYY»: tronca alla prima data dell'anno in cui cade la ts, la parte temporale sarà azzerata

- «QUARTER»: viene troncato alla prima data del trimestre in cui rientra la ts, la parte temporale verrà azzerata
- «MONTH», «MM», «MON»: tronca alla prima data del mese in cui cade ts, la parte temporale verrà azzerata
- «SETTIMANA»: viene troncata al lunedì della settimana in cui cade la ts, la parte temporale verrà azzerata
- «DAY», «DD»: azzera la parte temporale
- «HOUR»: azzera il minuto e il secondo con la parte frazionaria
- «MINUTO»: azzera il secondo con la parte frazionaria
- «SECONDO»: azzera la seconda parte della frazione
- «MILLISECOND»: azzera i microsecondi
- «MICROSECOND»: tutto rimane

ts

Un valore datetime

Tipo restituito

Restituisce il timestamp ts troncato all'unità specificata dal modello di formato

Esempi

L'esempio seguente tronca un valore di data all'inizio dell'anno. L'output mostra che la data «2015-03-05" è stata troncata a «2015-01-01", che è l'inizio dell'anno 2015.

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

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

Funzione DAY

La funzione DAY restituisce il giorno del mese della data/ora.

Le funzioni di estrazione della data sono utili quando è necessario lavorare con componenti specifici di una data o di un timestamp, ad esempio quando si eseguono calcoli basati sulla data, si filtrano i dati o si formattano i valori delle date.

Sintassi

```
day(date)
```

Arguments (Argomenti)

data

Un'espressione DATE o TIMESTAMP.

Valori restituiti

La funzione DAY restituisce un valore INTEGER.

Esempi

L'esempio seguente estrae il giorno del mese (`30`) dalla data di input. `'2009-07-30'`

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

L'esempio seguente estrae il giorno del mese dalla `birthday` colonna della `squirrels` tabella e restituisce i risultati come output dell'istruzione SELECT. L'output di questa query sarà un elenco di valori giornalieri, uno per ogni riga della `squirrels` tabella, che rappresentano il giorno del mese per il compleanno di ogni scoiattolo.

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

Funzione DAYOFMONTH

La funzione DAYOFMONTH restituisce il giorno del mese di date/timestamp (un valore compreso tra 1 e 31, a seconda del mese e dell'anno).

La funzione DAYOFMONTH è simile alla funzione DAY, ma hanno nomi e comportamenti leggermente diversi. La funzione DAY è più comunemente utilizzata, ma la funzione DAYOFMONTH può essere utilizzata come alternativa. Questo tipo di query può essere utile quando è necessario eseguire analisi o filtri basati sulla data su una tabella che contiene dati relativi a data o ora, ad esempio per estrarre componenti specifici di una data per ulteriori elaborazioni o report.

Sintassi

```
dayofmonth(date)
```

Arguments (Argomenti)

data

Un'espressione DATE o TIMESTAMP.

Valori restituiti

La funzione DAYOFMONTH restituisce un valore INTEGER.

Esempio

L'esempio seguente estrae il giorno del mese (`30`) dalla data di input. `'2009-07-30'`

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

L'esempio seguente applica la funzione DAYOFMONTH alla `birthday` colonna della tabella. `squirrels` Per ogni riga della `squirrels` tabella, il giorno del mese dalla `birthday` colonna verrà estratto e restituito come output dell'istruzione SELECT. L'output di questa query sarà un elenco di valori giornalieri, uno per ogni riga della `squirrels` tabella, che rappresentano il giorno del mese per il compleanno di ogni scoiattolo.

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

Funzione DAYOFWEEK

La funzione DAYOFWEEK accetta una data o un timestamp come input e restituisce il giorno della settimana come numero (1 per domenica, 2 per lunedì,..., 7 per sabato).

Questa funzione di estrazione della data è utile quando è necessario utilizzare componenti specifici di una data o di un timestamp, ad esempio quando si eseguono calcoli basati sulla data, si filtrano i dati o si formattano i valori delle date.

Sintassi

```
dayofweek(date)
```

Arguments (Argomenti)

data

Un'espressione DATE o TIMESTAMP.

Valori restituiti

La funzione DAYOFWEEK restituisce un numero INTERO dove

1 = domenica

2 = lunedì

3 = martedì

4 = mercoledì

5 = giovedì

6 = venerdì

7 = sabato

Esempi

L'esempio seguente estrae il giorno della settimana da questa data, che è 5 (che rappresenta giovedì).

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

L'esempio seguente estrae il giorno della settimana dalla `birthday` colonna della `squirrels` tabella e restituisce i risultati come output dell'istruzione SELECT. L'output di questa query sarà un elenco di valori del giorno della settimana, uno per ogni riga della `squirrels` tabella, che rappresentano il giorno della settimana per il compleanno di ogni scoiattolo.

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

funzione DAYOFYEAR

La funzione DAYOFYEAR è una funzione di estrazione della data che accetta una data o un timestamp come input e restituisce il giorno dell'anno (un valore compreso tra 1 e 366, a seconda dell'anno e se si tratta di un anno bisestile).

Questa funzione è utile quando è necessario lavorare con componenti specifici di una data o di un timestamp, ad esempio quando si eseguono calcoli basati sulla data, si filtrano i dati o si formattano i valori delle date.

Sintassi

```
dayofyear(date)
```

Arguments (Argomenti)

data

Un'espressione DATE o TIMESTAMP.

Valori restituiti

La funzione DAYOFYEAR restituisce un numero INTERO (compreso tra 1 e 366, a seconda dell'anno e se si tratta di un anno bisestile).

Esempi

L'esempio seguente estrae il giorno dell'anno () dalla data di input. `100 '2016-04-09'`

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

L'esempio seguente estrae il giorno dell'anno dalla `birthday` colonna della `squirrels` tabella e restituisce i risultati come output dell'istruzione SELECT.

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

Funzione EXTRACT

La funzione EXTRACT restituisce una parte di data o di ora da un valore TIMESTAMP, TIMESTAMPTZ, TIME o TIMETZ. Gli esempi includono un giorno, mese, ora, minuto, secondo, millisecondo o microsecondo da un timestamp.

Sintassi

```
EXTRACT(datepart FROM source)
```

Arguments (Argomenti)

datepart

Il sottocampo di una data o ora da estrarre, ad esempio un giorno, un mese, un anno, un'ora, un minuto, un secondo, un millisecondo o un microsecondo. Per un elenco dei valori possibili, consultare Parti di data per funzioni di data e timestamp.

source (origine)

Una colonna o un'espressione che restituisce un tipo di dati TIMESTAMP, TIMESTAMPTZ, TIME o TIMETZ.

Tipo restituito

INTEGER se il valore di origine restituisce un tipo di dati TIMESTAMP, TIME o TIMETZ.

DOUBLE PRECISION se il valore di origine restituisce il tipo di dati TIMESTAMPTZ.

Esempi con TIME

La tabella di esempio seguente TIME_TEST ha una colonna TIME_VAL (tipo TIME) con tre valori inseriti.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
```

```
00:58:00
```

Nell'esempio seguente vengono estratti i minuti da ogni timetz_val.

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

Nell'esempio seguente vengono estratte le ore da ogni time_val.

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

hours
-----------
         20
         0
         0
```

funzione FROM_UTC_TIMESTAMP

La funzione FROM_UTC_TIMESTAMP converte la data di input da UTC (Coordinated Universal Time) al fuso orario specificato.

Questa funzione è utile quando è necessario convertire i valori di data e ora da UTC a un fuso orario specifico. Questo può essere importante quando si lavora con dati che provengono da diverse parti del mondo e devono essere presentati nell'ora locale appropriata.

Sintassi

```
from_utc_timestamp(timestamp, timezone
```

Arguments (Argomenti)

timestamp

Un'espressione TIMESTAMP con un timestamp UTC.

timezone

Un'espressione STRING che è un fuso orario valido in cui convertire la data o il timestamp di input.

Valori restituiti

La funzione FROM_UTC_TIMESTAMP restituisce un TIMESTAMP.

Esempio

L'esempio seguente converte la data di input da UTC al fuso orario specificato (`'Asia/Seoul'`), che in questo caso è 9 ore prima dell'UTC. L'output risultante è la data e l'ora nel fuso orario di Seoul, che è. `2016-08-31 09:00:00`

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

Funzione HOUR

La funzione HOUR è una funzione di estrazione temporale che richiede un orario o un timestamp come input e restituisce il componente orario (un valore compreso tra 0 e 23).

Questa funzione di estrazione temporale è utile quando è necessario lavorare con componenti specifici di un orario o di un timestamp, ad esempio quando si eseguono calcoli basati sul tempo, si filtrano i dati o si formattano i valori temporali.

Sintassi

```
hour(timestamp)
```

Arguments (Argomenti)

timestamp

UN'ESPRESSIONE TIMESTAMP.

Valori restituiti

La funzione HOUR restituisce un valore INTEGER.

Esempio

L'esempio seguente estrae il componente hour (12) dal timestamp di input. `'2009-07-30 12:58:59'`

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

Funzione MINUTE

La funzione MINUTE è una funzione di estrazione temporale che richiede un orario o un timestamp come input e restituisce il componente dei minuti (un valore compreso tra 0 e 60).

Sintassi

```
minute(timestamp)
```

Arguments (Argomenti)

timestamp

Un'espressione TIMESTAMP o una STRING di un formato di timestamp valido.

Valori restituiti

La funzione MINUTE restituisce un valore INTEGER.

Esempio

L'esempio seguente estrae il componente minute (58) dal timestamp di input. `'2009-07-30 12:58:59'`

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

Funzione MONTH

La funzione MONTH è una funzione di estrazione temporale che richiede un orario o un timestamp come input e restituisce il componente del mese (un valore compreso tra 0 e 12).

Sintassi

```
month(date)
```

Arguments (Argomenti)

data

Un'espressione TIMESTAMP o una STRING di un formato di timestamp valido.

Valori restituiti

La funzione MONTH restituisce un valore INTEGER.

Esempio

L'esempio seguente estrae il componente month (7) dal timestamp di input. `'2016-07-30'`

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

Funzione SECOND

La funzione SECOND è una funzione di estrazione temporale che richiede un orario o un timestamp come input e restituisce il secondo componente (un valore compreso tra 0 e 60).

Sintassi

```
second(timestamp)
```

Arguments (Argomenti)

timestamp

Un'espressione TIMESTAMP.

Valori restituiti

La funzione SECOND restituisce un valore INTEGER.

Esempio

L'esempio seguente estrae il secondo componente (59) dal timestamp di input. `'2009-07-30 12:58:59'`

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

Funzione TIMESTAMP

La funzione TIMESTAMP accetta un valore (in genere un numero) e lo converte in un tipo di dati timestamp.

Questa funzione è utile quando è necessario convertire un valore numerico che rappresenta un'ora o una data in un tipo di dati timestamp. Ciò può essere utile quando si lavora con dati archiviati in un formato numerico, ad esempio timestamp Unix o epoch time.

Sintassi

```
timestamp(expr)
```

Arguments (Argomenti)

expr

Qualsiasi espressione che può essere trasmessa a TIMESTAMP.

Valori restituiti

La funzione TIMESTAMP restituisce un TIMESTAMP.

Esempio

L'esempio seguente converte un timestamp numerico Unix (`1632416400`) nel tipo di dati timestamp corrispondente: 22 settembre 2021 alle 12:00:00 UTC.

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

Funzione TO_TIMESTAMP

TO_TIMESTAMP converte una stringa TIMESTAMP in TIMESTAMPTZ.

Sintassi

```
to_timestamp (timestamp)
```

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

Arguments (Argomenti)

timestamp

Una stringa di timestamp o un tipo di dati che può essere inserito in una stringa di timestamp.

format

Una stringa letterale che corrisponde ai modelli datetime di Spark. Per modelli datetime validi, vedi Datetime Patterns for Formatting and Parsing.

Tipo restituito

TIMESTAMP

Esempi

L'esempio seguente dimostra l'utilizzo della funzione TO_TIMESTAMP per convertire una stringa TIMESTAMP in un 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
```

È possibile passare a TO_TIMESTAMP parte di una data. Le parti rimanenti della data sono impostate sui valori predefiniti. L'orario è incluso nell'output:

```
SELECT TO_TIMESTAMP('2017','YYYY');
```

```
to_timestamp
--------------------------
2017-01-01 00:00:00+00
```

La seguente istruzione SQL converte la stringa '2011-12-18 24:38:15 'in un TIMESTAMP. Il risultato è un TIMESTAMP che cade il giorno successivo perché il numero di ore è superiore a 24 ore:

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

to_timestamp
---------------------
2011-12-19 00:38:15+00
```

funzione YEAR

La funzione YEAR è una funzione di estrazione della data che accetta una data o un timestamp come input e restituisce il componente dell'anno (un numero a quattro cifre).

Sintassi

```
year(date)
```

Arguments (Argomenti)

data

Un'espressione DATE o TIMESTAMP.

Valori restituiti

La funzione YEAR restituisce un valore INTEGER.

Esempio

L'esempio seguente estrae il componente year (2016) dalla data di input. `'2016-07-30'`

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

L'esempio seguente estrae il componente year dalla `birthday` colonna della `squirrels` tabella e restituisce i risultati come output dell'istruzione SELECT. L'output di questa query sarà un elenco di valori annuali, uno per ogni riga della `squirrels` tabella, che rappresentano l'anno di nascita di ogni scoiattolo.

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

Parti di data per funzioni di data e timestamp

La tabella seguente identifica i nomi e le abbreviazioni di parti di data e parti di ora accettati come argomenti per le seguenti funzioni:

- DATE_ADD
- DATE_DIFF
- DATE_PART
- EXTRACT

Parte data o parte ora	Abbreviazioni
millennium, millennia	mil, mils
century, centuries	c, cent, cents
decade, decades	dec, decs
epoch	epoca (supportato da EXTRACT)
year, years	y, yr, yrs
quarter, quarters	qtr, qtrs
month, months	mon, mons
week, weeks	w
day of week	dayofweek, dow, dw, weekday (supportate da DATE_PART e Funzione EXTRACT) Restituisce un intero compreso tra 0 e 6, a partire da domenica.

Parte data o parte ora	Abbreviazioni
	Note La parte di data DOW si comporta in modo diverso rispetto alla parte di data day of week (D) utilizzata per stringhe in formato datetime. D si basa su numeri interi 1-7, dove domenica è 1. Per ulteriori informazioni, consulta Stringhe di formato datetime.
day of year	dayofyear, doy, dy, yearday (supportato da EXTRACT)
day, days	d
hour, hours	h, hr, hrs
minute, minutes	m, min, mins
second, seconds	s, sec, secs
millisecond, milliseconds	ms, msec, msecs, msecond, mseconds, millisec, millisecs, millisecon
microsecond, microseconds	microsec, microsecs, microsecond, usecond, useconds, us, usec, usecs
timezone, timezone_hour, timezone_minute	Supportato da EXTRACT solo per il timestamp con fuso orario (TIMESTAMPTZ).

Variazioni nei risultati con secondi, millisecondi e microsecondi

Differenze minori nei risultati delle query si hanno quando funzioni di data differenti specificano secondi, millisecondi o microsecondi come parti di data:

- La funzione EXTRACT restituisce interi solo per la parte di data specificata, ignorando parti di dati di livello superiore e inferiore. Se la parte di data specificata è secondi, millisecondi e microsecondi non sono inclusi nel risultato. Se la parte di data specificata è millisecondi, secondi e microsecondi non sono inclusi nel risultato. Se la parte di data specificata è microsecondi, secondi e millisecondi non sono inclusi nel risultato.

- La funzione DATE_PART restituisce la parte di secondi completa del timestamp, indipendentemente dalla parte di data specificata, restituendo un valore decimale o un intero in base alle necessità.

Note su CENTURY, EPOCH, DECADE e MIL

CENTURY o CENTURIES

AWS Clean Rooms interpreta un CENTURY in modo che inizi con l'anno ## #1 e finisca con l'anno: ###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)
```

EPOCA

L' AWS Clean Rooms implementazione di EPOCH è relativa al 1970-01-01 00:00:00.000 000 indipendentemente dal fuso orario in cui risiede il cluster. È possibile che sia necessario compensare i risultati della differenza in ore a seconda del fuso orario in cui si trova il cluster.

DECADE o DECADES

AWS Clean Rooms interpreta DECADE o DECADES DATEPART in base al calendario comune. Ad esempio, poiché il calendario comune inizia dall'anno 1, il primo decennio (decennio 1) va da 0001-01-01 a 0009-12-31 e il secondo decennio (decennio 2) va da 0010-01-01 a 0019-12-31. Ad esempio, il decennio 201 va da 2000-01-01 a 2009-12-31:

```
select extract(decade from timestamp '1999-02-16 20:38:40');
date_part
-----------
200
(1 row)
```

```
select extract(decade from timestamp '2000-02-16 20:38:40');
date_part
-----------
201
(1 row)

select extract(decade from timestamp '2010-02-16 20:38:40');
date_part
-----------
202
(1 row)
```

MIL o MILS

AWS Clean Rooms interpreta un MIL in modo che inizi con il primo giorno dell'anno #001 e finisca con l'ultimo giorno dell'anno: #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)
```

Funzioni di crittografia e decrittografia

Le funzioni di crittografia e decrittografia aiutano gli sviluppatori SQL a proteggere i dati sensibili dall'accesso non autorizzato o dall'uso improprio convertendoli tra un formato di testo semplice leggibile e un formato di testo cifrato illeggibile.

AWS Clean Rooms Spark SQL supporta le seguenti funzioni di crittografia e decrittografia:

Argomenti

- Funzione AES_ENCRYPT
- Funzione AES_DECRYPT

Funzione AES_ENCRYPT

La funzione AES_ENCRYPT viene utilizzata per crittografare i dati utilizzando l'algoritmo Advanced Encryption Standard (AES).

Sintassi

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

Argomenti

expr

Il valore binario da crittografare.

key

La passphrase da utilizzare per crittografare i dati.

Sono supportate lunghezze di chiave di 16, 24 e 32 bit.

modalità

Speciifica quale modalità di cifratura a blocchi deve essere utilizzata per crittografare i messaggi.

Modalità valide: ECB (Electronic CodeBook), GCM (Galois/Counter Mode), CBC (Cipher-Block Chaining).

imbottitura

Specifica come riempire i messaggi la cui lunghezza non è un multiplo della dimensione del blocco.

Valori validi: PKCS, NONE, DEFAULT.

Il padding DEFAULT indica PKCS (Public Key Cryptography Standards) per ECB, NONE per GCM e PKCS per CBC.

Le combinazioni supportate di (mode, padding) sono ('ECB', 'PKCS'), ('GCM', 'NONE') e ('CBC', 'PKCS').

iv

Vettore di inizializzazione opzionale (IV). Supportato solo per le modalità CBC e GCM.

Valori validi: 12 byte di lunghezza per GCM e 16 byte per CBC.

aad

Dati autenticati aggiuntivi opzionali (AAD). Supportato solo per la modalità GCM. Può essere qualsiasi input in formato libero e deve essere fornito sia per la crittografia che per la decrittografia.

Tipo restituito

La funzione AES_ENCRYPT restituisce un valore crittografato di expr utilizzando AES in una determinata modalità con il padding specificato.

Esempi

L'esempio seguente mostra come utilizzare la funzione Spark SQL AES_ENCRYPT per crittografare in modo sicuro una stringa di dati (in questo caso, la parola «Spark») utilizzando una chiave di crittografia specificata. Il testo cifrato risultante viene quindi codificato in Base64 per semplificare l'archiviazione o la trasmissione.

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

L'esempio seguente mostra come utilizzare la funzione Spark SQL AES_ENCRYPT per crittografare in modo sicuro una stringa di dati (in questo caso, la parola «Spark») utilizzando una chiave di crittografia specificata. Il testo cifrato risultante viene quindi rappresentato in formato esadecimale, che può essere utile per attività come l'archiviazione, la trasmissione o il debug dei dati.

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

L'esempio seguente mostra come utilizzare la funzione Spark SQL AES_ENCRYPT per crittografare in modo sicuro una stringa di dati (in questo caso, «Spark SQL») utilizzando una chiave di crittografia, una modalità di crittografia e una modalità di riempimento specificate. Il testo cifrato risultante viene quindi codificato in Base64 per semplificare l'archiviazione o la trasmissione.

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

Funzione AES_DECRYPT

La funzione AES_DECRYPT viene utilizzata per decrittografare i dati utilizzando l'algoritmo Advanced Encryption Standard (AES).

Sintassi

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

Argomenti

expr

Il valore binario da decifrare.

key

La passphrase da utilizzare per decrittografare i dati.

La passphrase deve corrispondere alla chiave utilizzata originariamente per produrre il valore crittografato ed essere lunga 16, 24 o 32 byte.

modalità

Speciifica quale modalità di cifratura a blocchi deve essere utilizzata per decrittografare i messaggi.

Modalità valide: ECB, GCM, CBC.

imbottitura

Specifica come riempire i messaggi la cui lunghezza non è un multiplo della dimensione del blocco.

Valori validi: PKCS, NONE, DEFAULT.

Il padding DEFAULT indica PKCS per ECB, NONE per GCM e PKCS per CBC.

- aad

Dati autenticati aggiuntivi opzionali (AAD). Supportato solo per la modalità GCM. Può essere qualsiasi input in formato libero e deve essere fornito sia per la crittografia che per la decrittografia.

Tipo restituito

Restituisce un valore decrittografato di expr utilizzando AES in modalità con padding.

Esempi

L'esempio seguente mostra come utilizzare la funzione Spark SQL AES_ENCRYPT per crittografare in modo sicuro una stringa di dati (in questo caso, la parola «Spark») utilizzando una chiave di crittografia specificata. Il testo cifrato risultante viene quindi codificato in Base64 per semplificare l'archiviazione o la trasmissione.

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

L'esempio seguente dimostra come utilizzare la funzione Spark SQL AES_DECRYPT per decrittografare dati precedentemente crittografati e codificati in Base64. Il processo di decrittografia richiede la chiave e i parametri di crittografia corretti (modalità di crittografia e modalità padding) per ripristinare correttamente i dati di testo in chiaro originali.

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

Funzioni hash

Una funzione hash è una funzione matematica che converte un valore di input numerico in un altro valore.

AWS Clean Rooms Spark SQL supporta le seguenti funzioni hash:

Argomenti

- MD5 funzione
- Funzione SHA
- SHA1 funzione
- SHA2 funzione
- HASH64 funzione xx

MD5 funzione

Utilizza la funzione hash MD5 crittografica per convertire una stringa di lunghezza variabile in una stringa di 32 caratteri che è una rappresentazione testuale del valore esadecimale di un checksum a 128 bit.

Sintassi

```
MD5(string)
```

Argomenti

stringa

Una stringa di lunghezza variabile.

Tipo restituito

La MD5 funzione restituisce una stringa di 32 caratteri che è una rappresentazione testuale del valore esadecimale di un checksum a 128 bit.

Esempi

L'esempio seguente mostra il valore a 128 bit per la stringa "AWS Clean Rooms":

```
select md5('AWS Clean Rooms');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

Funzione SHA

Sinonimo di funzione. SHA1

Consultare SHA1 funzione.

SHA1 funzione

La SHA1 funzione utilizza la funzione hash SHA1 crittografica per convertire una stringa di lunghezza variabile in una stringa di 40 caratteri che è una rappresentazione testuale del valore esadecimale di un checksum a 160 bit.

Sintassi

SHA1 è sinonimo di. Funzione SHA

```
SHA1(string)
```

Argomenti

stringa

Una stringa di lunghezza variabile.

Tipo restituito

La SHA1 funzione restituisce una stringa di 40 caratteri che è una rappresentazione testuale del valore esadecimale di un checksum a 160 bit.

Esempio

L'esempio seguente restituisce il valore a 160 bit per la parola 'AWS Clean Rooms':

```
select sha1('AWS Clean Rooms');
```

SHA2 funzione

La SHA2 funzione utilizza la funzione hash SHA2 crittografica per convertire una stringa di lunghezza variabile in una stringa di caratteri. La stringa di caratteri è una rappresentazione testuale del valore esadecimale del checksum con il numero specificato di bit.

Sintassi

```
SHA2(string, bits)
```

Argomenti

stringa

Una stringa di lunghezza variabile.

integer

Numero di bit nelle funzioni hash. I valori validi sono 0 (uguale a 256), 224, 256, 384 e 512.

Tipo restituito

La SHA2 funzione restituisce una stringa di caratteri che è una rappresentazione testuale del valore esadecimale del checksum o una stringa vuota se il numero di bit non è valido.

Esempio

L'esempio seguente restituisce il valore a 256 bit per la parola 'AWS Clean Rooms':

```
select sha2('AWS Clean Rooms', 256);
```

HASH64 funzione xx

La funzione xxhash64 restituisce un valore hash a 64 bit degli argomenti.

La funzione xxhash64 () è una funzione hash non crittografica progettata per essere veloce ed efficiente. Viene spesso utilizzata nelle applicazioni di elaborazione e archiviazione dei dati, in cui è necessario un identificatore univoco per un dato, ma non è necessario mantenere segreto il contenuto esatto dei dati.

Nel contesto di una query SQL, la funzione xxhash64 () può essere utilizzata per vari scopi, ad esempio:

- Generazione di un identificatore univoco per una riga in una tabella
- Partizionamento dei dati in base a un valore hash
- Implementazione di strategie personalizzate di indicizzazione o distribuzione dei dati

Il caso d'uso specifico dipenderebbe dai requisiti dell'applicazione e dai dati da elaborare.

Sintassi

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

Argomenti

espr (1)

Un'espressione di qualsiasi tipo.

expr 2

Un'espressione di qualsiasi tipo.

Valori restituiti

Restituisce un valore hash a 64 bit degli argomenti (BIGINT). L'hash seed è 42.

Esempio

L'esempio seguente genera un valore hash a 64 bit (5602566077635097486) in base all'input fornito. Il primo argomento è un valore di stringa, in questo caso la parola «Spark». Il secondo argomento è un array contenente il valore intero singolo 123. Il terzo argomento è un valore intero che rappresenta il seme della funzione hash.

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

Funzioni Hyperloglog

Le funzioni HyperLogLog (HLL) di SQL forniscono un modo per stimare in modo efficiente il numero di elementi unici (cardinalità) in un set di dati di grandi dimensioni, anche quando l'insieme effettivo di elementi unici non è archiviato.

I principali vantaggi dell'utilizzo delle funzioni HLL sono:

- Efficienza della memoria: gli schizzi HLL richiedono molta meno memoria rispetto all'archiviazione dell'intero set di elementi unici, il che li rende adatti a set di dati di grandi dimensioni.
- Calcolo distribuito: gli schizzi HLL possono essere combinati su più fonti di dati o nodi di elaborazione, consentendo una stima efficiente e distribuita del conteggio.
- Risultati approssimativi: HLL fornisce una stima approssimativa del conteggio univoca, con un compromesso regolabile tra precisione e utilizzo della memoria (tramite il parametro di precisione).

Queste funzioni sono particolarmente utili in scenari in cui è necessario stimare il numero di elementi unici, ad esempio nelle applicazioni di analisi, data warehousing e elaborazione di flussi in tempo reale.

AWS Clean Rooms supporta le seguenti funzioni HLL.

Argomenti

funzione HLL_SKETCH_AGG

La funzione di aggregazione HLL_SKETCH_AGG crea uno sketch HLL dai valori nella colonna specificata. Restituisce un tipo di dati HLLSKETCH che incapsula i valori delle espressioni di input.

La funzione di aggregazione HLL_SKETCH_AGG funziona con qualsiasi tipo di dati e ignora i valori NULL.

Quando non ci sono righe in una tabella o tutte le righe sono NULL, lo schizzo risultante non ha coppie indice-valore come `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}`.

Sintassi

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

Argomento

aggregate_expression

Qualsiasi espressione di tipo INT, BIGINT, STRING o BINARY rispetto alla quale verrà eseguito un conteggio univoco. Tutti i NULL valori vengono ignorati.

LGConfigK

Una costante INT opzionale compresa tra 4 e 21 inclusi con 12 di default. Il log-base-2 di K, dove K è il numero di bucket o slot per lo schizzo.

Tipo restituito

La funzione HLL_SKETCH_AGG restituisce un buffer BINARY non NULL contenente lo sketch calcolato poiché consuma e aggrega tutti i valori di input nel gruppo di aggregazione. HyperLogLog

Esempi

Gli esempi seguenti utilizzano l'algoritmo HyperLogLog (HLL) per stimare il numero distinto di valori nella colonna. `col` La `hll_sketch_agg(col, 12)` funzione aggrega i valori nella colonna col, creando uno schizzo HLL con una precisione di 12. La `hll_sketch_estimate()` funzione viene quindi utilizzata per stimare il numero distinto di valori in base allo schizzo HLL generato. Il risultato finale della query è 3, che rappresenta il conteggio distinto stimato di valori nella `col` colonna. In questo caso, i valori distinti sono 1, 2 e 3.

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

L'esempio seguente utilizza anche l'algoritmo HLL per stimare il numero distinto di valori nella `col` colonna, ma non specifica un valore di precisione per lo sketch HLL. In questo caso, utilizza la precisione predefinita di 14. La `hll_sketch_agg(col)` funzione prende i valori nella `col` colonna e crea uno schizzo HyperLogLog (HLL), che è una struttura di dati compatta che può essere utilizzata per stimare il numero distinto di elementi. La `hll_sketch_estimate(hll_sketch_agg(col))` funzione utilizza lo schizzo HLL creato nel passaggio precedente e calcola una stima del numero distinto di valori nella colonna. `col` Il risultato finale della query è 3, che rappresenta il numero distinto stimato di valori nella colonna. `col` In questo caso, i valori distinti sono 1, 2 e 3.

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

Funzione HLL_SKETCH_ESTIMATE

La funzione HLL_SKETCH_ESTIMATE esegue uno schizzo HLL e stima il numero di elementi unici rappresentati dallo schizzo. Utilizza l'algoritmo HyperLogLog (HLL) per contare un'approssimazione probabilistica del numero di valori univoci in una determinata colonna, utilizzando una rappresentazione binaria nota come buffer di sketch generata in precedenza dalla funzione HLL_SKETCH_AGG e restituendo il risultato come un numero intero grande.

L'algoritmo di disegno HLL fornisce un modo efficiente per stimare il numero di elementi unici, anche per set di dati di grandi dimensioni, senza dover memorizzare l'intero set di valori univoci.

Le `hll_union_agg` funzioni `hll_union` and possono anche combinare gli schizzi utilizzando e unendo questi buffer come input.

Sintassi

```
HLL_SKETCH_ESTIMATE (hllsketch_expression)
```

Argomento

hllsketch_expression

Un'`BINARY`espressione contenente uno schizzo generato da HLL_SKETCH_AGG

Tipo restituito

La funzione HLL_SKETCH_ESTIMATE restituisce un valore BIGINT che è il conteggio distinto approssimativo rappresentato dallo schizzo di input.

Esempi

Gli esempi seguenti utilizzano l'algoritmo di disegno HyperLogLog (HLL) per stimare la cardinalità (conteggio univoco) dei valori nella colonna. `col` La `hll_sketch_agg(col, 12)` funzione prende la `col` colonna e crea uno schizzo HLL con una precisione di 12 bit. Lo sketch HLL è una struttura di dati approssimativa in grado di stimare in modo efficiente il numero di elementi unici in un set. La `hll_sketch_estimate()` funzione prende lo schizzo HLL creato da `hll_sketch_agg` e stima la cardinalità (conteggio univoco) dei valori rappresentati dallo schizzo. `FROM VALUES (1), (1), (2), (2), (3) tab(col);`Genera un set di dati di test con 5 righe, in cui la `col` colonna contiene i valori 1, 1, 2, 2 e 3. Il risultato di questa query è il conteggio univoco stimato dei valori nella `col` colonna, che è 3.

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

La differenza tra l'esempio seguente e quello precedente è che il parametro di precisione (12 bit) non è specificato nella chiamata di `hll_sketch_agg` funzione. In questo caso, viene utilizzata la precisione predefinita di 14 bit, che può fornire una stima più accurata del conteggio unico rispetto all'esempio precedente che utilizzava 12 bit di precisione.

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

```
3
```

Funzione HLL_UNION

La funzione HLL_UNION combina due schizzi HLL in un unico schizzo unificato. Utilizza l'algoritmo HyperLogLog (HLL) per combinare due schizzi in un unico schizzo. Le query possono utilizzare i buffer risultanti per calcolare conteggi univoci approssimativi sotto forma di numeri interi lunghi con la funzione. `hll_sketch_estimate`

Sintassi

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

Argomento

Exprn

Un'`BINARY`espressione contenente uno schizzo generato da HLL_SKETCH_AGG.

allowDifferentLgConfigK

Un'espressione BOOLEAN opzionale che controlla se consentire l'unione di due schizzi con valori LGConfigK diversi. Il valore predefinito è `false`.

Tipo restituito

La funzione HLL_UNION restituisce un buffer BINARY contenente lo HyperLogLog schizzo calcolato come risultato della combinazione delle espressioni di input. Quando il `allowDifferentLgConfigK` parametro è`true`, lo schizzo del risultato utilizza il più piccolo dei due valori forniti. `lgConfigK`

Esempi

Gli esempi seguenti utilizzano l'algoritmo di disegno HyperLogLog (HLL) per stimare il conteggio univoco dei valori su due colonne `col1` e `col2` in un set di dati.

La `hll_sketch_agg(col1)` funzione crea uno schizzo HLL per i valori univoci nella colonna. `col1`

La `hll_sketch_agg(col2)` funzione crea uno schizzo HLL per i valori univoci nella colonna col2.

La `hll_union(...)` funzione combina i due schizzi HLL creati nei passaggi 1 e 2 in un unico schizzo HLL unificato.

La `hll_sketch_estimate(...)` funzione utilizza lo schizzo HLL combinato e stima il conteggio univoco dei valori per entrambi e. `col1` `col2`

La `FROM VALUES` clausola genera un set di dati di test con 5 righe, dove `col1` contiene i valori 1, 1, 2, 2 e 3 e `col2` contiene i valori 4, 4, 5, 5 e 6.

Il risultato di questa query è il conteggio unico stimato di valori per entrambi `col1` e`col2`, che è 6. L'algoritmo di disegno HLL offre un modo efficiente per stimare il numero di elementi unici, anche per set di dati di grandi dimensioni, senza dover memorizzare l'intero set di valori univoci. In questo esempio, la `hll_union` funzione viene utilizzata per combinare gli schizzi HLL delle due colonne, il che consente di stimare il conteggio univoco per l'intero set di dati, anziché solo per ogni colonna singolarmente.

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

La differenza tra l'esempio seguente e quello precedente è che il parametro di precisione (12 bit) non è specificato nella chiamata alla funzione. `hll_sketch_agg` In questo caso, viene utilizzata la precisione predefinita di 14 bit, che può fornire una stima più accurata del conteggio unico rispetto all'esempio precedente che utilizzava 12 bit di precisione.

```
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);
```

Funzione HLL_UNION_AGG

La funzione HLL_UNION_AGG combina più schizzi HLL in un unico schizzo unificato. Utilizza l'algoritmo HyperLogLog (HLL) per combinare un gruppo di schizzi in uno solo. Le query possono utilizzare i buffer risultanti per calcolare conteggi univoci approssimativi con la funzione. `hll_sketch_estimate`

Sintassi

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

Argomento

expr

Un'`BINARY`espressione contenente uno schizzo generato da HLL_SKETCH_AGG.

allowDifferentLgConfigK

Un'espressione BOOLEAN opzionale che controlla se consentire l'unione di due schizzi con valori LGConfigK diversi. Il valore predefinito è `false`.

Tipo restituito

La funzione HLL_UNION_AGG restituisce un buffer BINARY contenente lo HyperLogLog sketch calcolato come risultato della combinazione delle espressioni di input dello stesso gruppo. Quando il `allowDifferentLgConfigK` parametro è, lo schizzo del risultato utilizza il più `true` piccolo dei due valori forniti. `lgConfigK`

Esempi

I seguenti esempi utilizzano l'algoritmo di sketch HyperLogLog (HLL) per stimare il conteggio univoco dei valori su più schizzi HLL.

Il primo esempio stima il conteggio univoco dei valori in un set di dati.

```
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
```

L'interrogazione interna crea due sketch HLL:

- La prima istruzione SELECT crea uno schizzo a partire da un singolo valore di 1.
- La seconda istruzione SELECT crea uno schizzo a partire da un altro valore singolo di 1, ma con una precisione di 20.

La query esterna utilizza la funzione HLL_UNION_AGG per combinare i due schizzi in un unico schizzo. Quindi applica la funzione HLL_SKETCH_ESTIMATE a questo sketch combinato per stimare il conteggio univoco dei valori.

Il risultato di questa query è il conteggio univoco stimato dei valori nella colonna, che è. `col` 1 Ciò significa che i due valori di input pari a 1 sono considerati unici, anche se hanno lo stesso valore.

Il secondo esempio include un parametro di precisione diverso per la funzione HLL_UNION_AGG. In questo caso, entrambi gli schizzi HLL vengono creati con una precisione di 14 bit, il che consente di combinarli con successo utilizzando il parametro. `hll_union_agg` `true`

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

Il risultato finale della query è il conteggio univoco stimato, che anche in questo caso è. 1 Ciò significa che i due valori di input pari a 1 sono considerati unici, anche se hanno lo stesso valore.

Funzioni JSON

Quando è necessario memorizzare un insieme relativamente piccolo di coppie chiave-valore, è possibile risparmiare spazio memorizzando i dati nel formato JSON. Poiché le stringhe JSON possono essere memorizzate in una singola colonna, l'utilizzo di JSON potrebbe essere più efficiente rispetto all'archiviazione dei dati in formato tabulare.

Example

Ad esempio, supponiamo di avere una tabella sparsa, in cui è necessario disporre di molte colonne per rappresentare appieno tutti gli attributi possibili. Tuttavia, la maggior parte dei valori delle colonne sono NULL per una determinata riga o colonna. Utilizzando JSON per l'archiviazione, potresti essere in grado di archiviare i dati di una riga in coppie chiave-valore in una singola stringa JSON ed eliminare le colonne della tabella scarsamente popolate.

Inoltre, è possibile modificare facilmente le stringhe JSON per memorizzare coppie chiavi:valore aggiuntive senza dover aggiungere colonne a una tabella.

È consigliabile usare un JSON con parsimonia. JSON non è una buona scelta per archiviare set di dati di grandi dimensioni perché, archiviando dati diversi in una singola colonna, JSON non utilizza l'architettura dell'archivio di colonne. AWS Clean Rooms

JSON utilizza stringhe di testo con codifica UTF-8, pertanto le stringhe JSON possono essere memorizzate come tipi di dati CHAR o VARCHAR. Utilizzare VARCHAR se le stringhe includono caratteri multibyte.

Le stringhe JSON devono essere formattate in modo corretto con JSON, in base alle seguenti regole:

- Il JSON di livello radice può essere un oggetto JSON o un array JSON. Un oggetto JSON è un insieme non ordinato di coppie di chiave:valore separate da virgole racchiuse da parentesi graffe.

  Ad esempio, `{"one":1, "two":2}`
- Un array JSON è un insieme ordinato di valori separati da virgola racchiusi tra parentesi.

  Un esempio è quanto segue: `["first", {"one":1}, "second", 3, null]`
- Gli array JSON utilizzano un indice basato su zero; il primo elemento di un array è in posizione 0. In una coppia chiave:valore JSON, la chiave è una stringa racchiusa tra virgolette doppie.
- Un valore JSON può essere uno dei seguenti:
  - Oggetto JSON
  - Array JSON
  - Stringa tra virgolette doppie
  - Numero (intero e a virgola mobile)
  - Booleano
  - Null

- Gli oggetti vuoti e gli array vuoti sono valori JSON validi.
- I campi JSON fanno distinzione tra maiuscole e minuscole.
- Lo spazio bianco tra gli elementi strutturali JSON (ad esempio `{ }, [ ]`) viene ignorato.

Argomenti

Funzione GET_JSON_OBJECT

La funzione GET_JSON_OBJECT estrae un oggetto json da. `path`

Sintassi

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

Argomenti

json_txt

Un'espressione STRING contenente JSON ben formato.

path

Un valore letterale STRING con un'espressione di percorso JSON ben formata.

Valori restituiti

Restituisce una STRING.

Viene restituito un valore NULL se l'oggetto non può essere trovato.

Esempio

L'esempio seguente estrae un valore da un oggetto JSON. Il primo argomento è una stringa JSON che rappresenta un oggetto semplice con una singola coppia chiave-valore. Il secondo argomento è un'espressione di percorso JSON. Il $ simbolo rappresenta la radice dell'oggetto JSON e la `.a` parte specifica che vogliamo estrarre il valore associato alla chiave "»a. L'output della funzione è 'b', che è il valore associato al tasto "a" nell'oggetto JSON di input.

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

Funzione TO_JSON

La funzione TO_JSON converte un'espressione di input in una rappresentazione di stringa JSON. La funzione gestisce la conversione di diversi tipi di dati (come numeri, stringhe e valori booleani) nelle rappresentazioni JSON corrispondenti.

La funzione TO_JSON è utile quando è necessario convertire dati strutturati (come righe di database o oggetti JSON) in un formato più portatile e autodescrittivo come JSON. Ciò può essere particolarmente utile quando è necessario interagire con altri sistemi o servizi che prevedono dati in formato JSON.

Sintassi

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

Argomenti

expr

L'espressione di input che desideri convertire in una stringa JSON. Può essere un valore, una colonna o qualsiasi altra espressione SQL valida.

options

Un set opzionale di opzioni di configurazione che possono essere utilizzate per personalizzare il processo di conversione JSON. Queste opzioni possono includere cose come la gestione di valori nulli, la rappresentazione di valori numerici e il trattamento di caratteri speciali.

Valori restituiti

Restituisce una stringa JSON con un determinato valore di struttura

Esempi

L'esempio seguente converte una struttura denominata (un tipo di dati strutturati) in una stringa JSON. Il primo argomento (`named_struct('a', 1, 'b', 2)`) () è l'espressione di input che viene passata alla `to_json()` funzione. Crea una struttura denominata con due campi: «a» con un valore di 1 e «b» con un valore di 2. La funzione to_json () prende la struttura denominata come

argomento e la converte in una rappresentazione di stringa JSON. L'output è`{"a":1,"b":2}`, che è una stringa JSON valida che rappresentava la struttura denominata.

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

L'esempio seguente converte una struttura denominata che contiene un valore di timestamp in una stringa JSON, con un formato timestamp personalizzato. Il primo argomento (`named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd'))`) crea una struttura denominata con un singolo campo 'time' che contiene il valore del timestamp. Il secondo argomento (`map('timestampFormat', 'dd/MM/yyyy')`) crea una mappa (dizionario chiave-valore) con una singola coppia chiave-valore, dove la chiave è 'timestampFormat' e il valore è ". 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 L'output è `{"time":"26/08/2015"}` una stringa JSON con un singolo campo 'time' che contiene il valore del timestamp nel formato " desiderato. dd/MM/yyyy

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

Funzioni matematiche

Questa sezione descrive gli operatori e le funzioni matematiche supportati in AWS Clean Rooms Spark SQL.

Argomenti

- Simboli degli operatori matematici
- Funzione ABS
- Funzione ACOS
- Funzione ASIN
- Funzione ATAN
- ATAN2 funzione
- Funzione CBRT
- Funzione CEILING (oppure CEIL)

- Funzione COS
- Funzione COT
- Funzione DEGREES
- Funzione DIV
- Funzione EXP
- Funzione FLOOR
- Funzione LN
- Funzione LOG
- Funzione MOD
- Funzione PI
- Funzione POWER
- Funzioni RADIANS
- Funzione RAND
- Funzione RANDOM
- Funzione ROUND
- Funzione SIGN
- Funzione SIN
- Funzione SQRT
- Funzione TRUNC

Simboli degli operatori matematici

La tabella seguente elenca gli operatori matematici supportati.

Operatori supportati

Operatore	Description	Esempio	Risultato
+	addizione	2 + 3	5
-	sottrazione	2 - 3	-1
*	moltiplicazione	2 * 3	6

Operatore	Description	Esempio	Risultato
/	divisione	4 / 2	2
%	modulo	5 % 4	1
^	potenza	2,0 ^ 3,0	8

Esempi

Calcola la commissione pagata più una commissione di gestione di 2,00 USD per una determinata transazione:

```
select commission, (commission + 2.00) as comm
from sales where salesid=10000;

commission | comm
-----------+-------
28.05      | 30.05
(1 row)
```

Calcolare il 20 percento del prezzo di vendita per una determinata transazione:

```
select pricepaid, (pricepaid * .20) as twentypct
from sales where salesid=10000;

pricepaid | twentypct
----------+-----------
187.00    |    37.400
(1 row)
```

Vendite di biglietti previste in base a un modello di crescita continua. In questo esempio, la sottoquery restituisce il numero di biglietti venduti nel 2008. Tale risultato viene moltiplicato in modo esponenziale per un tasso di crescita continuo del 5 percento in 10 anni.

```
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)
```

Trova il prezzo totale pagato e le commissioni per le vendite con un ID data maggiore o uguale a 2.000. Quindi sottrarre la commissione totale dal prezzo totale pagato.

```
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)
```

Funzione ABS

ABS calcola il valore assoluto di un numero, in cui quel numero può essere un valore letterale o un'espressione che valuta un numero.

Sintassi

```
ABS (number)
```

Arguments (Argomenti)

numero

Numero o espressione che valuta un numero. Può essere SMALLINT, INTEGER, BIGINT, DECIMAL o type. FLOAT4 FLOAT8

Tipo restituito

ABS Restituisce lo stesso tipo di dati del suo argomento.

Esempi

Calcolare il valore assoluto di -38:

```
select abs (-38);
abs
-------
38
(1 row)
```

Calcolare il valore assoluto di (14-76):

```
select abs (14-76);
abs
-------
62
(1 row)
```

Funzione ACOS

ACOS è una funzione trigonometrica che restituisce l'arco coseno di un numero. Il valore restituito è in radianti ed è compreso tra `0` e `PI`.

Sintassi

```
ACOS(number)
```

Arguments (Argomenti)

numero

Il parametro di input è un numero `DOUBLE PRECISION`.

Tipo restituito

`DOUBLE PRECISION`

Esempi

Per restituire l'arco coseno di `-1`, utilizza l'esempio seguente.

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Funzione ASIN

ASIN è una funzione trigonometrica che restituisce l'arco seno di un numero. Il valore restituito è in radianti ed è compreso tra `PI/2` e `-PI/2`.

Sintassi

```
ASIN(number)
```

Arguments (Argomenti)

numero

Il parametro di input è un numero `DOUBLE PRECISION`.

Tipo restituito

`DOUBLE PRECISION`

Esempi

Per restituire l'arco seno di 1, utilizza l'esempio seguente.

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
```

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

Funzione ATAN

ATAN è una funzione trigonometrica che restituisce l'arco tangente di un numero. Il valore restituito è in radianti ed è compreso tra `-PI` e `PI`.

Sintassi

```
ATAN(number)
```

Arguments (Argomenti)

numero

Il parametro di input è un numero `DOUBLE PRECISION`.

Tipo restituito

`DOUBLE PRECISION`

Esempi

Per restituire l'arco tangente di 1 e moltiplicarlo per 4, utilizza l'esempio seguente.

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

ATAN2 funzione

ATAN2 è una funzione trigonometrica che restituisce l'arcotangente di un numero diviso per un altro numero. Il valore restituito è in radianti ed è compreso tra `PI/2` e `-PI/2`.

Sintassi

```
ATAN2(number1, number2)
```

Arguments (Argomenti)

number1

Un numero DOUBLE PRECISION.

number2

Un numero DOUBLE PRECISION.

Tipo restituito

DOUBLE PRECISION

Esempi

Per restituire l'arco tangente di 2/2 e moltiplicarlo per 4, utilizza l'esempio seguente.

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Funzione CBRT

La funzione CBRT è una funzione matematica che calcola la radice cubica di un numero.

Sintassi

```
CBRT (number)
```

Argomento

CBRT prende un numero DOUBLE PRECISION come argomento.

Tipo restituito

La funzione CBRT restituisce un numero DOUBLE PRECISION.

Esempi

Calcolare la radice cubica della commissione pagata per una determinata transazione:

```
select cbrt(commission) from sales where salesid=10000;

cbrt
------------------
3.03839539048843
(1 row)
```

Funzione CEILING (oppure CEIL)

La funzione CEILING o CEIL viene utilizzata per arrotondare un numero fino al numero intero successivo. (L Funzione FLOOR arrotonda un numero fino al numero intero successivo.)

Sintassi

```
CEIL | CEILING(number)
```

Arguments (Argomenti)

numero

Il numero o l'espressione che restituisce un numero. Può essere SMALLINT, INTEGER, BIGINT, DECIMAL o type. FLOAT4 FLOAT8

Tipo restituito

CEILING e CEIL restituiscono lo stesso tipo di dati come argomento.

Esempio

Calcolare il tetto della commissione pagata per una determinata transazione di vendita:

```
select ceiling(commission) from sales
where salesid=10000;

ceiling
---------
29
(1 row)
```

Funzione COS

COS è una funzione trigonometrica che restituisce il coseno di un numero. Il valore restituito è in radianti ed è compreso tra `-1` e `1`, inclusi.

Sintassi

```
COS(double_precision)
```

Argomento

numero

Il parametro di input è un numero a precisione doppia.

Tipo restituito

La funzione COS restituisce un numero a precisione doppia.

Esempi

L'esempio seguente restituisce l'arco coseno di 0:

```
select cos(0);
cos
-----
1
(1 row)
```

L'esempio seguente restituisce l'arco coseno di PI:

```
select cos(pi());
cos
-----
-1
(1 row)
```

Funzione COT

COT è una funzione trigonometrica che restituisce la cotangente di un numero. Il parametro di input deve essere diverso da zero.

Sintassi

```
COT(number)
```

Argomento

numero

Il parametro di input è un numero DOUBLE PRECISION.

Tipo restituito

DOUBLE PRECISION

Esempi

Per restituire la cotangente di 1, utilizza l'esempio seguente.

```
SELECT COT(1);

+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

Funzione DEGREES

Converte un angolo in radianti nel suo equivalente in gradi.

Sintassi

```
DEGREES(number)
```

Argomento

numero

Il parametro di input è un numero DOUBLE PRECISION.

Tipo restituito

`DOUBLE PRECISION`

Esempio

Per restituire l'equivalente in gradi di 0,5 radianti, utilizza l'esempio seguente.

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
+-------------------+
```

Per convertire i radianti PI in gradi, utilizza l'esempio seguente.

```
SELECT DEGREES(pi());

+---------+
| degrees |
+---------+
|     180 |
+---------+
```

Funzione DIV

L'operatore DIV restituisce la parte integrale della divisione del dividendo per divisore.

Sintassi

```
dividend div divisor
```

Arguments (Argomenti)

dividendo

Un'espressione che restituisce un valore numerico o un intervallo.

divisore

Un tipo di intervallo corrispondente if `dividend` è un intervallo, altrimenti un valore numerico.

Tipo restituito

`BIGINT`

Esempi

L'esempio seguente seleziona due colonne dalla tabella degli scoiattoli: la `id` colonna, che contiene l'identificatore univoco per ogni scoiattolo, e una colonna`age div 2`, che rappresenta la divisione in numeri interi `calculated` della colonna dell'età per 2. Il `age div 2` calcolo esegue la divisione in numeri interi sulla `age` colonna, arrotondando efficacemente l'età al numero intero pari più vicino. Ad esempio, se la `age` colonna contiene valori come 3, 5, 7 e 10, la `age div 2` colonna conterrà rispettivamente i valori 1, 2, 3 e 5.

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

Questa query può essere utile in scenari in cui è necessario raggruppare o analizzare i dati in base alle fasce di età e si desidera semplificare i valori di età arrotondandoli per difetto al numero intero pari più vicino. L'output risultante fornirebbe la `id` e l'età divisa per 2 per ogni scoiattolo nella tabella. `squirrels`

Funzione EXP

La funzione EXP implementa la funzione esponenziale di un'espressione numerica o la base del logaritmo naturale, e, elevato alla potenza dell'espressione. La funzione EXP è l'inverso di Funzione LN.

Sintassi

```
EXP (expression)
```

Argomento

espressione

L'espressione deve essere un tipo di dati numero INTEGER, DECIMAL, o DOUBLE PRECISION.

Tipo restituito

La funzione EXP restituisce un numero DOUBLE PRECISION.

Esempio

Utilizzare la funzione EXP per prevedere le vendite di biglietti in base a un modello di crescita continua. In questo esempio, la sottoquery restituisce il numero di biglietti venduti nel 2008. Questo risultato è moltiplicato per il risultato della funzione EXP, che specifica un tasso di crescita continua del 7% nel corso di 10 anni.

```
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)
```

Funzione FLOOR

La funzione FLOOR arrotonda un numero fino al numero intero successivo.

Sintassi

```
FLOOR (number)
```

Argomento

numero

Il numero o l'espressione che restituisce un numero. Può essere SMALLINT, INTEGER, BIGINT, DECIMAL o type. FLOAT4 FLOAT8

Tipo restituito

FLOOR restituisce lo stesso tipo di dati del suo argomento.

Esempio

L'esempio mostra il valore della commissione pagata per una determinata transazione di vendita prima e dopo l'utilizzo della funzione 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)
```

Funzione LN

La funzione LN restituisce il logaritmo naturale del parametro di input.

Sintassi

```
LN(expression)
```

Argomento

espressione

L'espressione o colonna di destinazione su cui viene eseguita la funzione.

> **Note**
>
> Questa funzione restituisce un errore per alcuni tipi di dati se l'espressione fa riferimento a una tabella AWS Clean Rooms creata dall'utente o a una tabella di sistema AWS Clean Rooms STL o STV.

Le espressioni con i seguenti tipi di dati generano un errore se fanno riferimento a una tabella creata dall'utente o di sistema.

- BOOLEAN
- CHAR

- DATE
- DECIMAL o NUMERIC
- TIMESTAMP
- VARCHAR

Le espressioni con i seguenti tipi di dati vengono eseguite correttamente su tabelle create dall'utente e su tabelle di sistema STL o STV:

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

Tipo restituito

La funzione LN restituisce lo stesso tipo dell'espressione.

Esempio

Nell'esempio seguente viene restituito il logaritmo naturale o il logaritmo di base e del numero 2,718281828:

```
select ln(2.718281828);
ln
--------------------
0.9999999998311267
(1 row)
```

Si noti che la risposta è quasi uguale a 1.

Questo esempio restituisce il logaritmo naturale dei valori nella colonna USERID nella tabella USERS:

```
select username, ln(userid) from users order by userid limit 10;

 username |        ln
----------+-------------------
 JSG99FHE |                 0
```

```
 PGL08LJI | 0.693147180559945
 IFT66TXU |  1.09861228866811
 XDZ38RDD |  1.38629436111989
 AEB55QTM |   1.6094379124341
 NDQ15VBM |  1.79175946922805
 OWY35QYB |  1.94591014905531
 AZG78YIP |  2.07944154167984
 MSD36KVR |  2.19722457733622
 WKW41AIW |  2.30258509299405
(10 rows)
```

Funzione LOG

Restituisce il logaritmo di with. `expr` `base`

Sintassi

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

Argomento

expr

L'espressione deve avere un tipo di dati integer, numero decimale o numero in virgola mobile.

base

La base per il calcolo del logaritmo. Deve essere un numero positivo (diverso da 1) di tipo di dati a doppia precisione.

Tipo restituito

La funzione LOG restituisce un numero a precisione doppia.

Esempio

Il seguente esempio restituisce il logaritmo di base 10 del numero 100:

```
select log(10, 100);
--------
2
(1 row)
```

Funzione MOD

Restituisce il resto di due numeri, altrimenti nota come operazione modulo. Per calcolare il risultato, il primo parametro viene diviso per il secondo.

Sintassi

```
MOD(number1, number2)
```

Arguments (Argomenti)

number1

Il primo parametro di input è un numero INTEGER, SMALLINT, BIGINT, o DECIMAL. Se uno dei parametri è di tipo DECIMAL, anche l'altro parametro deve essere di tipo DECIMAL. Se uno dei parametri è un INTEGER, l'altro parametro può essere un INTEGER, SMALLINT, o BIGINT. Entrambi i parametri possono essere anche SMALLINT o BIGINT, ma un parametro non può essere un SMALLINT se l'altro è un BIGINT.

number2

Il secondo parametro di input è un numero INTEGER, SMALLINT, BIGINT, o DECIMAL. Le stesse regole sui tipi di dati si applicano a number2 così come a number1.

Tipo restituito

I tipi di restituzione validi sono DECIMAL, INT, SMALLINT e BIGINT. Il tipo di restituzione della funzione MOD è lo stesso tipo numerico dei parametri di input, se entrambi i parametri di input sono dello stesso tipo. Se entrambi i parametri di input sono INTEGER, comunque, il tipo di restituzione sarà anche un INTEGER.

Note per l'utilizzo

Puoi utilizzare % come operatore di modulo.

Esempi

L'esempio seguente restituisce il resto quando un numero viene diviso per un altro:

```
SELECT MOD(10, 4);
```

```
 mod
------
 2
```

L'esempio seguente restituisce un risultato decimale:

```
SELECT MOD(10.5, 4);

 mod
------
 2.5
```

Puoi trasmettere i valori dei parametri:

```
SELECT MOD(CAST(16.4 as integer), 5);

 mod
------
 1
```

Controlla se il primo parametro è pari dividendolo per 2:

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

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

Puoi utilizzare % come operatore di modulo:

```
SELECT 11 % 4 as remainder;

 remainder
-----------
 3
```

L'esempio seguente restituisce informazioni per le categorie dispari nella tabella 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)
```

Funzione PI

La funzione PI restituisce il valore di pi con 14 posizioni decimali.

Sintassi

```
PI()
```

Tipo restituito

`DOUBLE PRECISION`

Esempi

Per restituire il valore di pi, utilizza l'esempio seguente.

```
SELECT PI();

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Funzione POWER

La funzione POWER è una funzione esponenziale che eleva un'espressione numerica alla potenza di una seconda espressione numerica. Ad esempio, 2 alla terza è calcolato come `POWER(2,3)`, con risultato 8.

Sintassi

```
{POWER(expression1, expression2)
```

Arguments (Argomenti)

expression1

Espressione numerica da elevare. Deve essere un tipo di dati `INTEGER`, `DECIMAL` o `FLOAT`.

expression2

Potenza da elevare expression1. Deve essere un tipo di dati `INTEGER`, `DECIMAL` o `FLOAT`.

Tipo restituito

`DOUBLE PRECISION`

Esempio

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

Funzioni RADIANS

La funzione RADIANS converte un angolo in gradi nel suo equivalente in radianti.

Sintassi

```
RADIANS(number)
```

Argomento

numero

Il parametro di input è un numero `DOUBLE PRECISION`.

Tipo restituito

`DOUBLE PRECISION`

Esempio

Per restituire l'equivalente in radianti di 180 gradi, utilizza l'esempio seguente.

```
SELECT RADIANS(180);

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

Funzione RAND

La funzione RAND genera un numero casuale a virgola mobile compreso tra 0 e 1. La funzione RAND genera un nuovo numero casuale ogni volta che viene chiamata.

Sintassi

```
RAND()
```

Tipo restituito

RANDOM restituisce un valore DOUBLE.

Esempio

L'esempio seguente genera una colonna di numeri casuali a virgola mobile compresi tra 0 e 1 per ogni riga della tabella. `squirrels` L'output risultante sarebbe una singola colonna contenente un elenco di valori decimali casuali, con un valore per ogni riga della tabella degli scoiattoli.

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

Questo tipo di query è utile quando è necessario generare numeri casuali, ad esempio per simulare eventi casuali o per introdurre la casualità nell'analisi dei dati. Nel contesto della `squirrels` tabella,

potrebbe essere utilizzata per assegnare valori casuali a ciascuno scoiattolo, che potrebbero quindi essere utilizzati per ulteriori elaborazioni o analisi.

Funzione RANDOM

La funzione RANDOM genera un valore casuale compreso tra 0.0 (incluso) e 1.0 (escluso).

Sintassi

```
RANDOM()
```

Tipo restituito

RANDOM restituisce un numero DOUBLE PRECISION.

Esempi

1. Calcolare un valore casuale compreso tra 0 e 99. Se il numero casuale è da 0 a 1, questa query produce un numero casuale compreso tra 0 e 100:

   ```
   select cast (random() * 100 as int);

   INTEGER
   ------
   24
   (1 row)
   ```

2. Recuperare un esempio casuale uniforme di 10 voci:

   ```
   select *
   from sales
   order by random()
   limit 10;
   ```

   Ora recuperare un esempio casuale di 10 voci, ma sceglierle in proporzione al loro prezzo. Ad esempio, una voce il cui prezzo è il doppio di un'altra ha il doppio delle probabilità di apparire nei risultati della query:

   ```
   select *
   from sales
   order by log(1 - random()) / pricepaid
   ```

```
limit 10;
```

3. Questo esempio utilizza il comando SET per impostare un valore SEED in modo che RANDOM generi una sequenza di numeri prevedibile.

   Innanzitutto, restituisce tre interi RANDOM senza prima impostare il valore SEED:

```
select cast (random() * 100 as int);
INTEGER
------
6
(1 row)

select cast (random() * 100 as int);
INTEGER
------
68
(1 row)

select cast (random() * 100 as int);
INTEGER
------
56
(1 row)
```

   Ora impostare il valore SEED su `.25`, e restituire altri tre numeri RANDOM:

```
set seed to .25;
select cast (random() * 100 as int);
INTEGER
------
21
(1 row)

select cast (random() * 100 as int);
INTEGER
------
79
(1 row)

select cast (random() * 100 as int);
INTEGER
------
```

```
12
(1 row)
```

Infine, ripristinare il valore SEED su `.25`, e verificare che RANDOM restituisca gli stessi risultati delle tre chiamate precedenti:

```
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)
```

Funzione ROUND

La funzione ROUND arrotonda i numeri al integer o decimale più vicino.

La funzione ROUND può facoltativamente includere un secondo argomento: un integer per indicare il numero di cifre decimali per l'arrotondamento, in entrambe le direzioni. Quando non si specifica il secondo argomento, la funzione viene arrotondata al numero intero più vicino. Quando il secondo argomento specificato è >n, la funzione viene arrotondata al numero più vicino con n cifre decimali di precisione.

Sintassi

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

Argomento

numero

Un numero o un'espressione che restituisce un numero. Può essere DECIMAL o type. FLOAT8 AWS Clean Rooms può convertire altri tipi di dati secondo le regole di conversione implicite.

integer (facoltativo)

Un intero che indica il numero di posizioni decimali per l'arrotondamento, in entrambe le direzioni.

Tipo restituito

ROUND restituisce lo stesso tipo di dati numerici degli argomenti di input.

Esempi

Arrotondare la commissione pagata per una determinata transazione al numero intero più vicino.

```
select commission, round(commission)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    28
(1 row)
```

Arrotondare la commissione pagata per una determinata transazione al primo posto decimale.

```
select commission, round(commission, 1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |  28.1
(1 row)
```

Per la stessa query, estendere la precisione nella direzione opposta.

```
select commission, round(commission, -1)
from sales where salesid=10000;
```

```
commission | round
-----------+-------
     28.05 |    30
(1 row)
```

Funzione SIGN

La funzione SIGN restituisce il segno (positivo o negativo) di un numero. Il risultato della funzione SIGN è un valore `1`, `-1` o `0` a indicare il segno dell'argomento.

Sintassi

```
SIGN (number)
```

Argomento

numero

Numero o espressione che valuta un numero. Può essere il DECIMALor FLOAT8 tipo. AWS Clean Rooms può convertire altri tipi di dati secondo le regole di conversione implicite.

Tipo restituito

SIGN restituisce lo stesso tipo di dati numerici degli argomenti di input. Se l'input è DECIMAL, l'output è DECIMAL(1,0).

Esempi

Per determinare il segno della commissione pagata per una determinata transazione dalla tabella SALES, utilizza l'esempio seguente.

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

Funzione SIN

SIN è una funzione trigonometrica che restituisce il seno di un numero. Il valore restituito è compreso tra `-1` e `1`.

Sintassi

```
SIN(number)
```

Argomento

numero

Un numero `DOUBLE PRECISION` in radianti.

Tipo restituito

`DOUBLE PRECISION`

Esempio

Per restituire il seno di `-PI`, utilizza l'esempio seguente.

```
SELECT SIN(-PI());

+-------------------------+
|           sin           |
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

Funzione SQRT

La funzione SQRT restituisce la radice quadrata di un valore numerico. La radice quadrata è un numero moltiplicato per sé stesso per ottenere il valore fornito.

Sintassi

```
SQRT (expression)
```

Argomento

espressione

L'espressione deve avere un tipo di dati integer, numero decimale o numero in virgola mobile. L'espressione può includere funzioni. Il sistema potrebbe eseguire conversioni di tipo implicito.

Tipo restituito

La funzione SQRT restituisce un numero DOUBLE PRECISION.

Esempi

L'esempio seguente restituisce la radice quadrata di un numero.

```
select sqrt(16);

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

L'esempio seguente esegue una conversione di tipo implicito.

```
select sqrt('16');

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

L'esempio seguente annida le funzioni per eseguire un'attività più complessa.

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

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

L'esempio seguente restituisce la lunghezza del raggio quando viene fornita l'area di un cerchio. Calcola il raggio in pollici, ad esempio, quando viene fornita l'area in pollici quadrati. L'area dell'esempio è 20.

```
select sqrt(20/pi());
```

Ciò restituisce il valore 5,046265044040321.

L'esempio seguente restituisce la radice quadrata per i valori di COMMISSION dalla tabella SALES. La colonna COMMISSION è una colonna DECIMAL. Questo esempio mostra come utilizzare la funzione in una query con una logica condizionale più complessa.

```
select sqrt(commission)
from sales where salesid < 10 order by salesid;

sqrt
------------------
10.4498803820905
3.37638860322683
7.24568837309472
5.1234753829798
...
```

La seguente query restituisce la radice quadrata arrotondata per lo stesso insieme dei valori di COMMISSION.

```
select salesid, commission, round(sqrt(commission))
from sales where salesid < 10 order by salesid;

salesid | commission | round
--------+------------+-------
      1 |     109.20 |    10
      2 |      11.40 |     3
      3 |      52.50 |     7
      4 |      26.25 |     5
...
```

Per ulteriori informazioni sui dati di esempio in AWS Clean Rooms, consulta Database di esempio.

Funzione TRUNC

La funzione TRUNC tronca i numeri all'intero o al decimale precedente.

La funzione TRUNC può facoltativamente includere un secondo argomento come un intero per indicare il numero di cifre decimali per l'arrotondamento, in entrambe le direzioni. Quando non si

specifica il secondo argomento, la funzione viene arrotondata al numero intero più vicino. Quando viene specificato il secondo argomento >n, la funzione viene arrotondata al numero più vicino con n cifre decimali di precisione. Questa funzione tronca anche un timestamp e restituisce una data.

Sintassi

```
TRUNC (number [ , integer ] |
timestamp )
```

Arguments (Argomenti)

numero

Un numero o un'espressione che restituisce un numero. Può essere il numero DECIMALE o FLOAT8 il tipo. AWS Clean Rooms può convertire altri tipi di dati secondo le regole di conversione implicite.

integer (facoltativo)

Un integer che indica il numero di posizioni decimali di precisione, in entrambe le direzioni. Se non viene fornito un valore integer, il numero viene troncato come numero intero; se viene specificato un valore integer, il numero viene troncato alla posizione decimale specificata.

timestamp

La funzione può anche restituire la data da un timestamp. Per restituire un valore di timestamp con `00:00:00` come ora, eseguire il casting del risultato della funzione su TIMESTAMP.

Tipo restituito

TRUNC restituisce lo stesso tipo di dati del primo argomento di input. Per i timestamp, TRUNC restituisce una data.

Esempi

Troncare la commissione pagata per una determinata transazione di vendita.

```
select commission, trunc(commission)
from sales where salesid=784;

commission | trunc
-----------+-------
```

```
    111.15 |   111

(1 row)
```

Troncare lo stesso valore della commissione alla prima posizione decimale.

```
select commission, trunc(commission,1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 | 111.1

(1 row)
```

Troncare la commissione con un valore negativo per il secondo argomento; `111.15` è arrotondato per difetto a `110`.

```
select commission, trunc(commission,-1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   110
(1 row)
```

Restituisce la parte di data dal risultato della funzione SYSDATE (che restituisce un timestamp):

```
select sysdate;

timestamp
----------------------------
2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc
------------
2011-07-21
(1 row)
```

Applica la funzione TRUNC a una colonna TIMESTAMP. Il tipo restituito è una data.

```
select trunc(starttime) from event
order by eventid limit 1;

trunc
------------
2008-01-25
(1 row)
```

Funzioni scalari

Questa sezione descrive le funzioni scalari supportate in Spark SQL. AWS Clean Rooms Una funzione scalare è una funzione che accetta uno o più valori come input e restituisce un singolo valore come output. Le funzioni scalari operano su singole righe o elementi e producono un unico risultato per ogni input.

Le funzioni scalari, come SIZE, sono diverse dagli altri tipi di funzioni SQL, come le funzioni aggregate (count, sum, avg) e le funzioni di generazione di tabelle (explode, flatten). Questi altri tipi di funzioni operano su più righe o generano più righe, mentre le funzioni scalari funzionano su singole righe o elementi.

Argomenti

- Funzione SIZE

Funzione SIZE

La funzione SIZE accetta una matrice, una mappa o una stringa esistente come argomento e restituisce un singolo valore che rappresenta la dimensione o la lunghezza di quella struttura di dati. Non crea una nuova struttura di dati. Viene utilizzato per interrogare e analizzare le proprietà delle strutture di dati esistenti, anziché per crearne di nuove.

Questa funzione è utile per determinare il numero di elementi in un array o la lunghezza di una stringa. Può essere particolarmente utile quando si lavora con matrici e altre strutture di dati in SQL, perché consente di ottenere informazioni sulla dimensione o sulla cardinalità dei dati.

Sintassi

```
size(expr)
```

Argomenti

expr

Un'espressione ARRAY, MAP o STRING.

Tipo restituito

La funzione SIZE restituisce un valore INTEGER.

Esempio

In questo esempio, la funzione SIZE viene applicata all'array `['b', 'd', 'c', 'a']` e restituisce il valore4, che è il numero di elementi nell'array.

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

In questo esempio, la funzione SIZE viene applicata alla mappa `{'a': 1, 'b': 2}` e restituisce il valore2, che è il numero di coppie chiave-valore nella mappa.

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

In questo esempio, la funzione SIZE viene applicata alla stringa `'hello world'` e restituisce il valore11, che è il numero di caratteri nella stringa.

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

Funzioni stringa

Le funzioni di stringa elaborano e manipolano stringhe di caratteri o espressioni che valutano le stringhe di caratteri. Quando l'argomento stringa in queste funzioni è un valore letterale, deve essere racchiuso tra virgolette singole. I tipi di dati supportati includono CHAR e VARCHAR.

La seguente sezione fornisce i nomi della funzione, la sintassi e le descrizioni per le funzioni supportate. Tutti gli offset in stringhe sono basati su uno.

Argomenti

|| (Concatenamento) Operatore

Concatena due espressioni su entrambi i lati del simbolo || e restituisce l'espressione concatenata.

L'operatore di concatentazione è simile a. Funzione CONCAT

> **Note**
>
> Sia per la funzione CONCAT sia per l'operatore di concatenazione, se una o entrambe le espressioni sono nulle, il risultato della concatenazione è nullo.

Sintassi

```
expression1 || expression2
```

Argomenti

expression1, expression2

Entrambi gli argomenti possono essere stringhe di caratteri o espressioni a lunghezza fissa o a lunghezza variabile.

Tipo restituito

L'operatore || restituisce una stringa. Il tipo di stringa è lo stesso degli argomenti di input.

Esempio

L'esempio seguente concatena i campi FIRSTNAME e LASTNAME dalla tabella 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)
```

Per concatenare le colonne che potrebbero contenere valori null, utilizzare l'espressione Funzioni NVL e COALESCE. Il seguente esempio utilizza NVL per restituire uno 0 ogni volta che si incontra NULL.

```
select venuename || ' seats ' || nvl(venueseats, 0)
from venue where venuestate = 'NV' or venuestate = 'NC'
order by 1
limit 10;

seating
----------------------------------
Ballys Hotel seats 0
Bank of America Stadium seats 73298
Bellagio Hotel seats 0
Caesars Palace seats 0
Harrahs Hotel seats 0
Hilton Hotel seats 0
Luxor Hotel seats 0
Mandalay Bay Hotel seats 0
Mirage Hotel seats 0
New York New York seats 0
```

Funzione BTRIM

La funzione BTRIM riduce una stringa rimuovendo spazi vuoti iniziali e finali o rimuovendo i caratteri iniziali e finali che corrispondono a una stringa specificata facoltativa.

Sintassi

```
BTRIM(string [, trim_chars ] )
```

Argomenti

stringa

La stringa VARCHAR di input da ridurre.

trim_chars

La stringa VARCHAR contenente i caratteri da abbinare.

Tipo restituito

La funzione BTRIM restituisce una stringa VARCHAR.

Esempi

L'esempio seguente riduce gli spazi vuoti iniziali e finali dalla stringa `' abc '`:

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

L'esempio seguente rimuove le stringhe `'xyz'` iniziali e finali dalla stringa `'xyzaxyzbxyzcxyz'`. Le occorrenze iniziali e finali di `'xyz'` vengono rimosse, ma le occorrenze interne alla stringa non vengono rimosse.

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

L'esempio seguente rimuove le parti iniziale e finale dalla stringa `'setuphistorycassettes'` che corrispondono a uno qualsiasi dei caratteri nell'elenco `'tes'` trim_chars. Qualsiasi carattere `t`, e o `s` che si verifica prima di un altro carattere che non è nell'elenco trim_chars all'inizio o alla fine della stringa di input viene rimosso.

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
-----------------
 uphistoryca
```

Funzione CONCAT

La funzione CONCAT concatena due espressioni e restituisce l'espressione risultante. Per concatenare più di due espressioni, utilizzare le funzioni CONCAT nidificate. L'operatore di concatenazione (| |) tra due espressioni produce gli stessi risultati della funzione CONCAT.

> **Note**
>
> Sia per la funzione CONCAT sia per l'operatore di concatenazione, se una o entrambe le espressioni sono nulle, il risultato della concatenazione è nullo.

Sintassi

```
CONCAT ( expression1, expression2 )
```

Argomenti

expression1, expression2

Entrambi gli argomenti possono essere una stringa di caratteri a lunghezza fissa, una stringa di caratteri a lunghezza variabile, un'espressione binaria o un'espressione che valuta uno di questi input.

Tipo restituito

CONCAT restituisce un'espressione. Il tipo di dati dell'espressione è lo stesso tipo degli argomenti di input.

Se le espressioni di input sono di tipi diversi, AWS Clean Rooms prova a digitare implicitamente genera una delle espressioni. Se non è possibile eseguire il cast di valori, viene restituito il valore nullo.

Esempi

L'esempio seguente concatena due letterali di caratteri:

```
select concat('December 25, ', '2008');
```

```
concat
-------------------
December 25, 2008
(1 row)
```

La seguente query, utilizzando l'operatore || invece di CONCAT, produce lo stesso risultato:

```
select 'December 25, '||'2008';

concat
-------------------
December 25, 2008
(1 row)
```

Nell'esempio seguente vengono utilizzate due funzioni CONCAT per concatenare tre stringhe di caratteri:

```
select concat('Thursday, ', concat('December 25, ', '2008'));

concat
----------------------------
Thursday, December 25, 2008
(1 row)
```

Per concatenare le colonne che potrebbero contenere valori null, utilizzare la Funzioni NVL e COALESCE. Il seguente esempio utilizza NVL per restituire uno 0 ogni volta che si incontra NULL.

```
select concat(venuename, concat(' seats ', nvl(venueseats, 0))) as seating
from venue where venuestate = 'NV' or venuestate = 'NC'
order by 1
limit 5;

seating
---------------------------------
Ballys Hotel seats 0
Bank of America Stadium seats 73298
Bellagio Hotel seats 0
Caesars Palace seats 0
Harrahs Hotel seats 0
(5 rows)
```

La query seguente concatena i valori CITY e STATE dalla tabella VENUE:

```
select concat(venuecity, venuestate)
from venue
where venueseats > 75000
order by venueseats;

concat
-------------------
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

La seguente query utilizza funzioni CONCAT nidificate. La query concatena i valori CITY e STATE dalla tabella VENUE ma delimita la stringa risultante con una virgola e uno spazio:

```
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)
```

Funzione FORMAT_STRING

La funzione FORMAT_STRING crea una stringa formattata sostituendo i segnaposto in una stringa modello con gli argomenti forniti. Restituisce una stringa formattata da stringhe di formato in stile printf.

La funzione FORMAT_STRING funziona sostituendo i segnaposto nella stringa del modello con i valori corrispondenti passati come argomenti. Questo tipo di formattazione delle stringhe può essere utile quando è necessario creare dinamicamente stringhe che includono una combinazione di testo statico e dati dinamici, ad esempio quando si generano messaggi di output, report o altri tipi di testo

informativo. La funzione FORMAT_STRING fornisce un modo conciso e leggibile per creare questi tipi di stringhe formattate, semplificando la manutenzione e l'aggiornamento del codice che genera l'output.

Sintassi

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

Argomenti

strfmt

Un'espressione STRING.

obj

Una STRINGA o un'espressione numerica.

Tipo restituito

FORMAT_STRING restituisce una STRING.

Esempio

L'esempio seguente contiene una stringa modello che contiene due segnaposto: `%d` per un valore decimale (intero) e per un valore di stringa. `%s` Il `%d` segnaposto viene sostituito con il valore decimale (intero) () e il segnaposto %s viene sostituito con il valore di stringa (`100`). `"days"` L'output è una stringa modello con i segnaposto sostituiti dagli argomenti forniti:. `"Hello World 100 days"`

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

Funzioni LEFT e RIGHT

Queste funzioni restituiscono il numero specificato di caratteri più a sinistra o più a destra da una stringa di caratteri.

Il numero si basa sul numero di caratteri, non di byte, pertanto i caratteri multibyte vengono contati come caratteri singoli.

Sintassi

```
LEFT ( string,  integer )

RIGHT ( string,  integer )
```

Argomenti

stringa

Qualsiasi stringa di caratteri o espressione che valuti una stringa di caratteri.

integer

Un integer positivo.

Tipo restituito

LEFT e RIGHT restituiscono una stringa VARCHAR.

Esempio

L'esempio seguente restituisce i 5 caratteri più a sinistra e i 5 caratteri più a destra dei nomi di eventi con un valore compreso tra 1000 e 1005: IDs

```
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)
```

Funzione LENGTH

Funzione LOWER

Converte una stringa in minuscolo. LOWER supporta caratteri multibyte UTF-8, fino a un massimo di quattro byte per carattere.

Sintassi

```
LOWER(string)
```

Argomento

stringa

Il parametro di input è una stringa VARCHAR (o qualsiasi altro tipo di dati, ad esempio CHAR, che può essere convertito implicitamente in VARCHAR).

Tipo restituito

La funzione LOWER restituisce una stringa di caratteri che appartiene allo stesso tipo di dati della stringa di input.

Esempi

L'esempio seguente converte il campo CATNAME in lettere minuscole:

```
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)
```

Funzioni LPAD e RPAD

Queste funzioni antepongono o aggiungono caratteri a una stringa, in base a una lunghezza specificata.

Sintassi

```
LPAD (string1, length, [ string2 ])
```

```
RPAD (string1, length, [ string2 ])
```

Argomenti

string1

Una stringa di caratteri o un'espressione che valuta una stringa di caratteri, come il nome di una colonna di caratteri.

length

Un integer che definisce la lunghezza del risultato della funzione. La lunghezza di una stringa si basa sul numero di caratteri, non di byte, pertanto i caratteri multibyte vengono contati come caratteri singoli. Se string1 è più lunga della lunghezza specificata, viene troncata (a destra). Se lunghezza è un numero negativo, il risultato della funzione è una stringa vuota.

string2

Uno o più caratteri anteposti o aggiunti a string1. Questo argomento è facoltativo; se non è specificato, gli spazi vengono usati.

Tipo restituito

Queste funzioni restituiscono un tipo di dati VARCHAR.

Esempi

Troncare un insieme specificato di nomi di eventi a 20 caratteri e anteporre ai nomi più brevi gli spazi:

```
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)
```

Troncare lo stesso insieme specificato di nomi di eventi a 20 caratteri ma aggiungere ai nomi più brevi `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)
```

Funzione LTRIM

Taglia i caratteri dall'inizio di una stringa. Rimuove la stringa più lunga contenente solo i caratteri nell'elenco dei caratteri di taglio. Il taglio è completo quando un carattere di taglio non appare nella stringa di input.

Sintassi

```
LTRIM( string [, trim_chars] )
```

Argomenti

stringa

Una stringa, una colonna, un'espressione o una stringa letterale da tagliare.

trim_chars

Una colonna o un'espressione di stringhe o un valore letterale di stringa che rappresenta i caratteri da tagliare dall'inizio della stringa. Se non specificato, viene utilizzato uno spazio come carattere di taglio.

Tipo restituito

La funzione LTRIM restituisce una stringa di caratteri che appartiene allo stesso tipo di dati della stringa di input (CHAR o VARCHAR).

Esempi

L'esempio seguente taglia l'anno dalla colonna `listime`. I caratteri di taglio nel valore letterale di stringa `'2008-'` indicano i caratteri da tagliare da sinistra. Se si utilizzano i caratteri di taglio `'028-'`, si ottiene lo stesso risultato.

```
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 rimuove tutti i caratteri in trim_chars se questi si trovano all'inizio di stringa. L'esempio seguente riduce i caratteri "C", "D" e "G" quando si trovano all'inizio di VENUENAME che è una colonna 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
```

L'esempio seguente utilizza il carattere di taglio 2 che viene recuperato dalla colonna `venueid`.

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

L'esempio seguente non taglia alcun carattere perché prima del carattere di taglio `'0'` è presente un 2.

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

L'esempio seguente utilizza il carattere di taglio dello spazio predefinito e taglia i due spazi dall'inizio della stringa.

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

Funzione POSITION

Restituisce la posizione della sottostringa specificata all'interno di una stringa.

Sintassi

```
POSITION(substring IN string )
```

Argomenti

sottostringa

La sottostringa da cercare all'interno della stringa.

stringa

La stringa o colonna da ricercare.

Tipo restituito

La funzione POSITION restituisce un integer corrispondente alla posizione della sottostringa (basata su uno, non su zero). La posizione si basa sul numero di caratteri, non di byte, pertanto i caratteri multibyte vengono contati come caratteri singoli.

Note per l'utilizzo

POSITION restituisce 0 se la sottostringa non si trova all'interno della stringa:

```
select position('dog' in 'fish');

position
----------
 0
(1 row)
```

Esempi

L'esempio seguente mostra la posizione della stringa `fish` all'interno della parola `dogfish`:

```
select position('fish' in 'dogfish');
```

```
position
----------
 4
(1 row)
```

L'esempio seguente restituisce il numero di transazioni di vendita con una COMMISSION superiore a 999,00 dalla tabella SALES:

```
select distinct position('.' in commission), count (position('.' in commission))
from sales where position('.' in commission) > 4 group by position('.' in commission)
order by 1,2;

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

Funzione REGEXP_COUNT

Cerca una stringa per un modello di espressione regolare e restituisce un integer che indica il numero di volte in cui il modello si verifica nella stringa. Se non viene trovata alcuna corrispondenza, la funzione restituisce 0.

Sintassi

```
REGEXP_COUNT ( source_string, pattern [, position [, parameters ] ] )
```

Argomenti

source_string

Un'espressione di stringa, come ad esempio un nome di colonna, da cercare.

pattern

Un valore letterale di stringa che rappresenta un modello di espressione regolare.

posizione

Un integer positivo che indica la posizione all'interno di source_string per iniziare la ricerca. La posizione si basa sul numero di caratteri, non di byte, pertanto i caratteri multibyte vengono contati

come caratteri singoli. Il valore di default è 1. Se posizione è inferiore a 1, la ricerca inizia con il primo carattere di source_string. Se posizione è maggiore rispetto al numero di caratteri in source_string, il risultato è 0.

parameters

Uno o più letterali di stringa che indicano come la funzione corrisponde al modello. Di seguito sono riportati i valori possibili:

- c: eseguire una corrispondenza in base a maiuscole e minuscole. L'impostazione predefinita è utilizzare la corrispondenza con distinzione tra maiuscole e minuscole.
- i: eseguire una corrispondenza senza distinzione tra maiuscole e minuscole.
- p: interpreta il modello con il dialetto Perl Compatible Regular Expression (PCRE).

Tipo restituito

Numero intero

Esempio

L'esempio seguente conta il numero di volte in cui si verifica una sequenza di tre lettere.

```
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');

 regexp_count
 --------------
            8
```

L'esempio seguente conta il numero di volte in cui il nome di dominio di livello superiore è `org` oppure `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
```

Nell'esempio seguente vengono conteggiate le ricorrenze della stringa F0X utilizzando una corrispondenza senza distinzione tra maiuscole e minuscole.

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

 regexp_count
 --------------
            1
```

Nell'esempio seguente viene utilizzato un modello scritto in dialetto PCRE per individuare le parole contenenti almeno un numero e una lettera minuscola. Utilizza l'operatore ?=, che ha una connotazione look-ahead specifica in PCRE. In questo esempio viene contato il numero di ricorrenze di tali parole, con la corrispondenza tra maiuscole e minuscole.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

 regexp_count
 --------------
            2
```

Nell'esempio seguente viene utilizzato un modello scritto in dialetto PCRE per individuare le parole contenenti almeno un numero e una lettera minuscola. Utilizza l'operatore ?=, che ha una connotazione specifica in PCRE. In questo esempio viene contato il numero di ricorrenze di tali parole, ma differisce dall'esempio precedente in quanto utilizza la corrispondenza senza distinzione tra maiuscole e minuscole.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');

 regexp_count
 --------------
            3
```

Funzione REGEXP_INSTR

Cerca una stringa per un modello di espressione regolare e restituisce un integer che indica la posizione iniziale o finale della sottostringa corrispondente. Se non viene trovata alcuna corrispondenza, la funzione restituisce 0. REGEXP_INSTR è simile alla funzione POSITION, ma consente di cercare una stringa per un modello di espressione regolare.

Sintassi

```
REGEXP_INSTR ( source_string, pattern [, position [, occurrence] [, option
 [, parameters ] ] ] ] )
```

Argomenti

source_string

Un'espressione di stringa, come ad esempio un nome di colonna, da cercare.

pattern

Un valore letterale di stringa che rappresenta un modello di espressione regolare.

posizione

Un integer positivo che indica la posizione all'interno di source_string per iniziare la ricerca. La posizione si basa sul numero di caratteri, non di byte, pertanto i caratteri multibyte vengono contati come caratteri singoli. Il valore di default è 1. Se posizione è inferiore a 1, la ricerca inizia con il primo carattere di source_string. Se posizione è maggiore rispetto al numero di caratteri in source_string, il risultato è 0.

occorrenza

Un integer positivo che indica quale occorrenza del modello utilizzare. REGEXP_INSTR salta le prime corrispondenze occorrenza -1. Il valore di default è 1. Se occorrenza è inferiore a 1 oppure maggiore rispetto al numero di caratteri in source_string, la ricerca viene ignorata e il risultato è 0.

option

Un valore che indica se restituire la posizione del primo carattere della corrispondenza (`0`) o la posizione del primo carattere dopo la fine della corrispondenza (1). Un valore diverso da zero equivale a 1. Il valore predefinito è 0.

parameters

Uno o più letterali di stringa che indicano come la funzione corrisponde al modello. Di seguito sono riportati i valori possibili:

- c: eseguire una corrispondenza in base a maiuscole e minuscole. L'impostazione predefinita è utilizzare la corrispondenza con distinzione tra maiuscole e minuscole.
- i: eseguire una corrispondenza senza distinzione tra maiuscole e minuscole.
- e: estrarre una sottostringa usando una sottoespressione.

Se modello include una sottoespressione, REGEXP_INSTR corrisponde a una sottostringa che utilizza la prima sottoespressione in modello. REGEXP_INSTR considera solo la prima sottoespressione; le sottoespressioni aggiuntive vengono ignorate. Se il modello non ha una sottoespressione, REGEXP_INSTR ignora il parametro "e".

- p: interpreta il modello con il dialetto Perl Compatible Regular Expression (PCRE).

Tipo restituito

Numero intero

Esempio

Nell'esempio seguente viene cercato il carattere @ che inizia un nome di dominio e restituisce la posizione iniziale della prima corrispondenza.

```
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
```

Nell'esempio seguente vengono cercate le varianti della parola `Center` e viene restituita la posizione iniziale della prima corrispondenza.

```
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
```

Nell'esempio seguente viene trovata la posizione iniziale della prima ricorrenza della stringa FOX utilizzando una logica di associazione senza distinzione tra maiuscole e minuscole.

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

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

Nell'esempio seguente viene utilizzato un modello scritto in dialetto PCRE per individuare le parole contenenti almeno un numero e una lettera minuscola. Utilizza l'operatore ?=, che ha una connotazione look-ahead specifica in PCRE. In questo esempio viene trovata la posizione iniziale della seconda parola di questo tipo.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

 regexp_instr
 --------------
           21
```

Nell'esempio seguente viene utilizzato un modello scritto in dialetto PCRE per individuare le parole contenenti almeno un numero e una lettera minuscola. Utilizza l'operatore ?=, che ha una connotazione look-ahead specifica in PCRE. In questo esempio viene trovata la posizione iniziale della seconda parola, ma differisce dall'esempio precedente in quanto utilizza la corrispondenza senza distinzione tra maiuscole e minuscole.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');

 regexp_instr
 --------------
           15
```

Funzione REGEXP_REPLACE

Cerca una stringa per un modello di espressione regolare e sostituisce ogni occorrenza del modello con la stringa specificata. REGEXP_REPLACE è simile a Funzione REPLACE, ma consente di cercare una stringa per un modello di espressione regolare.

REGEXP_REPLACE è simile a Funzione TRANSLATE e a Funzione REPLACE, ad eccezione del fatto che TRANSLATE esegue più sostituzioni a carattere singolo e REPLACE sostituisce un'intera stringa con un'altra stringa, mentre REGEXP_REPLACE consente di cercare una stringa per un modello di espressione regolare.

Sintassi

```
REGEXP_REPLACE ( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

Argomenti

source_string

Un'espressione di stringa, come ad esempio un nome di colonna, da cercare.

pattern

Un valore letterale di stringa che rappresenta un modello di espressione regolare.

replace_string

Un'espressione di stringa, ad esempio un nome di colonna, che sostituirà ogni occorrenza del modello. L'impostazione predefinita è una stringa vuota ("").

posizione

Un integer positivo che indica la posizione all'interno di source_string per iniziare la ricerca. La posizione si basa sul numero di caratteri, non di byte, pertanto i caratteri multibyte vengono contati come caratteri singoli. Il valore di default è 1. Se posizione è inferiore a 1, la ricerca inizia con il primo carattere di source_string. Se posizione è maggiore rispetto al numero di caratteri in source_string, il risultato è source_string.

parameters

Uno o più letterali di stringa che indicano come la funzione corrisponde al modello. Di seguito sono riportati i valori possibili:

- c: eseguire una corrispondenza in base a maiuscole e minuscole. L'impostazione predefinita è utilizzare la corrispondenza con distinzione tra maiuscole e minuscole.
- i: eseguire una corrispondenza senza distinzione tra maiuscole e minuscole.
- p: interpreta il modello con il dialetto Perl Compatible Regular Expression (PCRE).

Tipo restituito

VARCHAR

Se modello oppure replace_string è NULL, il risultato è NULL.

Esempio

L'esempio seguente elimina @ e il nome di dominio dagli indirizzi 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
```

Nell'esempio seguente sono sostituiti i nomi di dominio degli indirizzi e-mail con questo valore: `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
```

Nell'esempio seguente vengono sostituite tutte le ricorrenze della stringa `FOX` con il valore `quick brown fox` utilizzando una corrispondenza senza distinzione tra maiuscole e minuscole.

```
SELECT regexp_replace('the fox', 'FOX', 'quick brown fox', 1, 'i');
```

```
   regexp_replace
---------------------
 the quick brown fox
```

Nell'esempio seguente viene utilizzato un modello scritto in dialetto PCRE per individuare le parole contenenti almeno un numero e una lettera minuscola. Utilizza l'operatore ?=, che ha una connotazione look-ahead specifica in PCRE. In questo esempio viene sostituita ogni ricorrenza di tale parola con il valore [hidden].

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

        regexp_replace
------------------------------
 [hidden] plain A1234 [hidden]
```

Nell'esempio seguente viene utilizzato un modello scritto in dialetto PCRE per individuare le parole contenenti almeno un numero e una lettera minuscola. Utilizza l'operatore ?=, che ha una connotazione look-ahead specifica in PCRE. In questo esempio viene sostituita ogni ricorrenza di tale parola con il valore [hidden], ma differisce dall'esempio precedente in quanto utilizza la corrispondenza senza distinzione tra maiuscole e minuscole.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

          regexp_replace
---------------------------------
 [hidden] plain [hidden] [hidden]
```

Funzione REGEXP_SUBSTR

Restituisce i caratteri da una stringa cercando un modello di espressione regolare. REGEXP_SUBSTR è simile alla funzione Funzione SUBSTRING ma consente di cercare una stringa per un modello di espressione regolare. Se la funzione non riesce a far corrispondere l'espressione regolare ad alcun carattere della stringa, restituisce una stringa vuota.

Sintassi

```
REGEXP_SUBSTR ( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

Argomenti

source_string

Un'espressione della stringa da ricercare.

pattern

Un valore letterale di stringa che rappresenta un modello di espressione regolare.

posizione

Un integer positivo che indica la posizione all'interno di source_string per iniziare la ricerca. La posizione si basa sul numero di caratteri, non di byte, pertanto i caratteri multibyte vengono contati come caratteri singoli. Il valore di default è 1. Se posizione è inferiore a 1, la ricerca inizia con il primo carattere di source_string. Se posizione è maggiore rispetto al numero di caratteri in source_string, il risultato è una stringa vuota ("").

occorrenza

Un integer positivo che indica quale occorrenza del modello utilizzare. REGEXP_SUBSTR salta le prime corrispondenze occorrenza -1. Il valore di default è 1. Se occorrenza è inferiore a 1 oppure maggiore rispetto al numero di caratteri in source_string, la ricerca viene ignorata e il risultato è NULL.

parameters

Uno o più letterali di stringa che indicano come la funzione corrisponde al modello. Di seguito sono riportati i valori possibili:

- c: eseguire una corrispondenza in base a maiuscole e minuscole. L'impostazione predefinita è utilizzare la corrispondenza con distinzione tra maiuscole e minuscole.
- i: eseguire una corrispondenza senza distinzione tra maiuscole e minuscole.
- e: estrarre una sottostringa usando una sottoespressione.

  Se modello include una sottoespressione, REGEXP_SUBSTR corrisponde a una sottostringa che utilizza la prima sottoespressione in modello. Un'espressione secondaria è un'espressione all'interno del modello racchiusa tra parentesi. Ad esempio, per il modello `'This is a (\\w+)'` corrisponde alla prima espressione con la stringa `'This is a '` seguita da una parola. Invece di restituire un modello, REGEXP_SUBSTR con il parametro e restituisce solo la stringa all'interno dell'espressione secondaria.

REGEXP_SUBSTR considera solo la prima sottoespressione; le sottoespressioni aggiuntive vengono ignorate. Se il modello non ha una sottoespressione, REGEXP_SUBSTR ignora il parametro "e".

- p: interpreta il modello con il dialetto Perl Compatible Regular Expression (PCRE).

Tipo restituito

VARCHAR

Esempio

L'esempio seguente restituisce la porzione di un indirizzo email tra il carattere @ e l'estensione del dominio.

```
SELECT email, regexp_substr(email,'@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

                     email                      |      regexp_substr
------------------------------------------------+--------------------------
 Etiam.laoreet.libero@sodalesMaurisblandit.edu | @sodalesMaurisblandit
 Suspendisse.tristique@nonnisiAenean.edu       | @nonnisiAenean
 amet.faucibus.ut@condimentumegetvolutpat.ca   | @condimentumegetvolutpat
 sed@lacusUtnec.ca                             | @lacusUtnec
```

Nell'esempio seguente viene restituita la porzione dell'input corrispondente alla prima ricorrenza della stringa F0X utilizzando una corrispondenza senza distinzione tra maiuscole e minuscole.

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

 regexp_substr
---------------
 fox
```

L'istruzione di esempio seguente restituisce la prima parte dell'input che inizia con lettere minuscole. Dal punto di vista funzionale è identica alla medesima istruzione SELECT senza il parametro c.

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');
```

```
 regexp_substr
---------------
 abc
```

Nell'esempio seguente viene utilizzato un modello scritto in dialetto PCRE per individuare le parole contenenti almeno un numero e una lettera minuscola. Utilizza l'operatore ?=, che ha una connotazione look-ahead specifica in PCRE. In questo esempio viene restituita la parte dell'input corrispondente alla seconda parola di questo tipo.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

Nell'esempio seguente viene utilizzato un modello scritto in dialetto PCRE per individuare le parole contenenti almeno un numero e una lettera minuscola. Utilizza l'operatore ?=, che ha una connotazione look-ahead specifica in PCRE. In questo esempio viene restituita la parte di input corrispondente alla seconda parola, ma differisce dall'esempio precedente in quanto si utilizza la corrispondenza senza distinzione tra maiuscole e minuscole.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
---------------
 A1234
```

L'esempio seguente utilizza un'espressione secondaria per trovare la seconda stringa che corrisponde al modello `'this is a (\\w+)'` utilizzando la corrispondenza senza distinzione tra maiuscole e minuscole. Restituisce l'espressione secondaria tra parentesi.

```
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
```

Funzione REPEAT

Ripete una stringa il numero specificato di volte. Se il parametro di input è numerico, REPEAT lo considera come una stringa.

Sintassi

```
REPEAT(string, integer)
```

Argomenti

stringa

Il primo parametro di input è la stringa da ripetere.

integer

Il secondo parametro è un integer che indica il numero di volte in cui ripetere la stringa.

Tipo restituito

La funzione REPEAT restituisce una stringa.

Esempi

L'esempio seguente ripete il valore della colonna CATID nella tabella CATEGORY tre volte:

```
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)
```

Funzione REPLACE

Sostituisce tutte le occorrenze di un insieme di caratteri all'interno di una stringa esistente con altri caratteri specificati.

REPLACE è simile a Funzione TRANSLATE e a Funzione REGEXP_REPLACE, ad eccezione del fatto che TRANSLATE esegue più sostituzioni a carattere singolo e REGEXP_REPLACE consente di cercare una stringa per un modello di espressione regolare, mentre REPLACE sostituisce un'intera stringa con un'altra stringa.

Sintassi

```
REPLACE(string1, old_chars, new_chars)
```

Argomenti

stringa

La stringa CHAR o VARCHAR da cercare in ricerca

old_chars

La stringa CHAR o VARCHAR da sostituire.

new_chars

Nuova stringa CHAR o VARCHAR che sostituisce la old_string.

Tipo restituito

VARCHAR

Se old_chars oppure new_chars è NULL, il risultato è NULL.

Esempi

L'esempio seguente converte la stringa `Shows` in `Theatre` nel campo CATGROUP:

```
select catid, catgroup,
replace(catgroup, 'Shows', 'Theatre')
from category
order by 1,2,3;
```

```
 catid | catgroup | replace
-------+----------+----------
     1 | Sports   | Sports
     2 | Sports   | Sports
     3 | Sports   | Sports
     4 | Sports   | Sports
     5 | Sports   | Sports
     6 | Shows    | Theatre
     7 | Shows    | Theatre
     8 | Shows    | Theatre
     9 | Concerts | Concerts
    10 | Concerts | Concerts
    11 | Concerts | Concerts
(11 rows)
```

Funzione REVERSE

La funzione REVERSE funziona su una stringa e restituisce i caratteri in ordine inverso. Ad esempio, `reverse('abcde')` restituisce `edcba`. Questa funzione funziona su tipi di dati numerici e di date, così come su tipi di dati di carattere; tuttavia, nella maggior parte dei casi ha un valore pratico per le stringhe di caratteri.

Sintassi

```
REVERSE ( expression )
```

Argomento

espressione

Un'espressione con un carattere, una data, un timestamp o un tipo di dati numerici che rappresenta la destinazione dell'inversione di caratteri. Tutte le espressioni sono implicitamente convertite in stringhe di caratteri a lunghezza variabile. Gli spazi finali in stringhe di caratteri a larghezza fissa vengono ignorati.

Tipo restituito

REVERSE restituisce una VARCHAR.

Esempi

Selezionare cinque nomi di città distinti e i corrispondenti nomi invertiti dalla tabella 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)
```

Seleziona cinque vendite IDs e il corrispondente IDs cast invertito come stringhe di caratteri:

```
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)
```

Funzione RTRIM

La funzione RTRIM riduce un insieme specificato di caratteri dalla fine di una stringa. Rimuove la stringa più lunga contenente solo i caratteri nell'elenco dei caratteri di taglio. Il taglio è completo quando un carattere di taglio non appare nella stringa di input.

Sintassi

```
RTRIM( string, trim_chars )
```

Argomenti

stringa

Una stringa, una colonna, un'espressione o una stringa letterale da tagliare.

trim_chars

Una colonna o un'espressione di stringa o un valore letterale di stringa che rappresenta i caratteri da tagliare dall'inizio della stringa. Se non specificato, viene utilizzato uno spazio come carattere di taglio.

Tipo restituito

Una stringa che è lo stesso tipo di dati dell'argomento stringa.

Esempio

L'esempio seguente riduce gli spazi vuoti iniziali e finali dalla stringa `' abc '`:

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

L'esempio seguente rimuove le stringhe `'xyz'` iniziali dalla stringa `'xyzaxyzbxyzcxyz'`. Le occorrenze finali di `'xyz'` vengono rimosse, ma le occorrenze interne alla stringa non vengono rimosse.

```
select 'xyzaxyzbxyzcxyz' as untrim,
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

L'esempio seguente rimuove le parti finali dalla stringa `'setuphistorycassettes'` che corrispondono a uno qualsiasi dei caratteri nell'elenco `'tes'` trim_chars. Qualsiasi carattere `t`, `e` o `s` che si verifica prima di un altro carattere che non è nell'elenco trim_chars alla fine della stringa di input viene rimosso.

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

L'esempio seguente riduce i caratteri "Parco"dalla fine di VENUENAME laddove presente:

```
select venueid, venuename, rtrim(venuename, 'Park')
from venue
order by 1, 2, 3
limit 10;

venueid |         venuename          |          rtrim
--------+----------------------------+-------------------------
      1 | Toyota Park                | Toyota
      2 | Columbus Crew Stadium      | Columbus Crew Stadium
      3 | RFK Stadium                | RFK Stadium
      4 | CommunityAmerica Ballpark  | CommunityAmerica Ballp
      5 | Gillette Stadium           | Gillette Stadium
      6 | New York Giants Stadium    | New York Giants Stadium
      7 | BMO Field                  | BMO Field
      8 | The Home Depot Center      | The Home Depot Cente
      9 | Dick's Sporting Goods Park | Dick's Sporting Goods
     10 | Pizza Hut Park             | Pizza Hut
```

Si noti che RTRIM rimuove tutti i caratteri in `P`, `a` , `r` oppure `k` quando appaiono alla fine di un VENUENAME.

Funzione SPLIT

La funzione SPLIT consente di estrarre sottostringhe da una stringa più grande e utilizzarle come matrice. La funzione SPLIT è utile quando è necessario suddividere una stringa in singoli componenti in base a un delimitatore o uno schema specifico.

Sintassi

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

Argomenti

str

Un'espressione stringa da dividere.

regex

Una stringa che rappresenta un'espressione regolare. La stringa regex deve essere un'espressione regolare Java.

limite

Un'espressione intera che controlla il numero di volte in cui viene applicata l'espressione regolare.

- limit > 0: la lunghezza dell'array risultante non sarà superiore al limite e l'ultima voce dell'array risultante conterrà tutti gli input oltre l'ultima espressione regolare corrispondente.
- limit <= 0: l'espressione regolare verrà applicata il maggior numero di volte possibile e l'array risultante può essere di qualsiasi dimensione.

Tipo restituito

<STRING>La funzione SPLIT restituisce un ARRAY.

Se`limit > 0`: la lunghezza dell'array risultante non sarà superiore al limite e l'ultima voce dell'array risultante conterrà tutti gli input oltre l'ultima espressione regolare corrispondente.

If`limit <= 0`: regex verrà applicato il maggior numero di volte possibile e l'array risultante può essere di qualsiasi dimensione.

Esempio

In questo esempio, la funzione SPLIT divide la stringa di input `'oneAtwoBthreeC'` ovunque incontri i caratteri `'A'` o `'C'` (come specificato dal modello di espressione regolare). `'B' '[ABC]'` L'output risultante è una matrice di quattro elementi:`"one"`, `"two""three"`, e una stringa vuota. `""`

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

Funzione SPLIT_PART

Divide una stringa sul delimitatore specificato e restituisce la parte nella posizione specificata.

Sintassi

```
SPLIT_PART(string, delimiter, position)
```

Argomenti

stringa

Una stringa, una colonna, un'espressione o una stringa letterale da dividere. La stringa può essere CHAR o VARCHAR.

delimiter

La stringa del delimitatore che indica le sezioni della stringa di input.

Se delimiter è un letterale, racchiuderlo tra virgolette singole.

posizione

Posizione della porzione di stringa da restituire (contando da 1). Deve essere un integer superiore a 0. Se posizione è maggiore del numero di porzioni di stringa, SPLIT_PART restituisce una stringa vuota. Se il delimitatore non si trova nella stringa, il valore restituito contiene i contenuti della parte specificata, che possono essere l'intera stringa o un valore vuoto.

Tipo restituito

Una stringa CHAR o VARCHAR, uguale al parametro di stringa.

Esempi

L'esempio seguente divide una stringa letterale in parti utilizzando il delimitatore $ e restituisce la seconda parte.

```
select split_part('abc$def$ghi','$',2)

split_part
----------
def
```

L'esempio seguente divide una stringa letterale in parti utilizzando il delimitatore $. Restituisce una stringa vuota perché la parte 4 non viene trovata.

```
select split_part('abc$def$ghi','$',4)

split_part
----------
```

L'esempio seguente divide una stringa letterale in parti utilizzando il delimitatore #. Restituisce l'intera stringa, che è la prima parte, perché il delimitatore non è stato trovato.

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

L'esempio seguente divide il campo timestamp LISTTIME in componenti anno, mese e data.

```
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
```

L'esempio seguente seleziona il campo timestamp LISTTIME e lo divide sul carattere `'-'` per ottenere il mese (la seconda parte della stringa LISTTIME), quindi conta il numero di voci per ogni mese:

```
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
```

Funzione SUBSTRING

Restituisce il sottoinsieme di una stringa basata su una posizione iniziale specificata.

Se l'input è una stringa di carattere, la posizione iniziale e il numero di caratteri estratti si basano sui caratteri, non di byte, pertanto i caratteri multibyte vengono contati come caratteri singoli. Se l'input è un'espressione binaria, la posizione iniziale e la sottostringa estratta sono basate su byte. Non è possibile specificare una lunghezza negativa, ma è possibile specificare una posizione di partenza negativa.

Sintassi

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

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

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

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

Argomenti

stringa di caratteri

La stringa da cercare. I tipi di dati non carattere sono trattati come una stringa.

start_position

La posizione all'interno della stringa per iniziare l'estrazione, a partire da 1. La start_position si basa sul numero di caratteri, non di byte, pertanto i caratteri multibyte vengono contati come caratteri singoli. Questo numero può essere negativo.

caratteri numerici

Il numero di caratteri da estrarre (la lunghezza della sottostringa). Il numbecharacters si basa sul numero di caratteri, non sui byte, in modo che i caratteri multibyte vengano contati come caratteri singoli. Questo numero non può essere negativo.

start_byte

La posizione all'interno dell'espressione binaria per iniziare l'estrazione, a partire da 1. Questo numero può essere negativo.

numero di byte

Il numero di byte da estrarre, ovvero, la lunghezza della sottostringa. Questo numero non può essere negativo.

Tipo restituito

VARCHAR

Note di utilizzo per le stringhe di caratteri

L'esempio seguente restituisce una stringa di quattro caratteri che inizia con il sesto carattere.

```
select substring('caterpillar',6,4);
substring
-----------
pill
(1 row)
```

Se start_position + numbecharacters supera la lunghezza della stringa, SUBSTRING restituisce una sottostringa a partire da start_position fino alla fine della stringa. Per esempio:

```
select substring('caterpillar',6,8);
substring
-----------
pillar
```

```
(1 row)
```

Se la `start_position` è negativa o pari a 0, la funzione SUBSTRING restituisce una sottostringa che inizia dal primo carattere di stringa con una lunghezza di `start_position` + `numbecharacters` -1. Ad esempio:

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

Se `start_position` + `numbecharacters` -1 è inferiore o pari a zero, SUBSTRING restituisce una stringa vuota. Ad esempio:

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

Esempi

L'esempio seguente restituisce il mese dalla stringa LISTTIME nella tabella 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)
```

L'esempio seguente è lo stesso di sopra, ma utilizza l'opzione 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)
```

Non è possibile utilizzare SUBSTRING per estrarre in modo prevedibile il prefisso di una stringa che potrebbe contenere caratteri multibyte poiché è necessario specificare la lunghezza di una stringa multibyte in base al numero di byte, non al numero di caratteri. Per estrarre il segmento iniziale di una stringa in base alla lunghezza in byte, è possibile eseguire il CAST della stringa come VARCHAR(byte_length) per troncare la stringa, laddove byte_length è la lunghezza necessaria. L'esempio seguente estrae i primi 5 byte dalla stringa `'Fourscore and seven'`.

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
Fours
```

L'esempio seguente restituisce il nome `Ana` che appare dopo l'ultimo spazio nella stringa di input `Silva, Ana`.

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))

 reverse
-----------
 Ana
```

Funzione TRANSLATE

Per una data espressione, sostituisce tutte le occorrenze di caratteri specificati con sostituti specificati. I caratteri esistenti sono mappati per i caratteri sostitutivi in base alla loro posizione negli argomenti characters_to_replace e characters_to_substitute. Se vengono specificati più caratteri nell'argomento characters_to_replace rispetto all'argomento characters_to_substitute, i caratteri extra dall'argomento characters_to_replace vengono omessi nel valore di restituzione.

TRANSLATE è simile a Funzione REPLACE e a Funzione REGEXP_REPLACE, ad eccezione del fatto che REPLACE sostituisce un'intera stringa con un'altra stringa e REGEXP_REPLACE consente di cercare una stringa per un modello di espressione regolare, mentre TRANSLATE realizza più sostituzioni a carattere singolo.

Se qualsiasi argomento è null, la restituzione è NULL.

Sintassi

```
TRANSLATE ( expression, characters_to_replace, characters_to_substitute )
```

Argomenti

espressione

L'espressione da tradurre.

characters_to_replace

Una stringa contenente i caratteri da sostituire.

characters_to_substitute

Una stringa contenente i caratteri da sostituire.

Tipo restituito

VARCHAR

Esempi

L'esempio seguente sostituisce diversi caratteri in una stringa:

```
select translate('mint tea', 'inea', 'osin');

translate
-----------
most tin
```

L'esempio seguente sostituisce il simbolo at (@) con un punto per tutti i valori in una colonna:

```
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
```

L'esempio seguente sostituisce gli spazi con caratteri di sottolineatura ed elimina i punti per tutti i valori in una colonna:

```
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
```

Funzione TRIM

Riduce una stringa rimuovendo spazi vuoti iniziali e finali o rimuovendo i caratteri iniziali e finali che corrispondono a una stringa specificata facoltativa.

Sintassi

```
TRIM( [ BOTH ] [ trim_chars FROM ] string
```

Argomenti

trim_chars

(Facoltativo) I caratteri da ridurre dalla stringa. Se questo parametro viene omesso, gli spazi vuoti vengono ridotti.

stringa

La stringa da ridurre.

Tipo restituito

La funzione TRIM restituisce una stringa VARCHAR o CHAR. Se si utilizza la funzione TRIM con un comando SQL, converte implicitamente i risultati in VARCHAR. AWS Clean Rooms Se si utilizza la funzione TRIM nell'elenco SELECT per una funzione SQL, AWS Clean Rooms non converte implicitamente i risultati e potrebbe essere necessario eseguire una conversione esplicita per evitare un errore di mancata corrispondenza del tipo di dati. Vedi la Funzione CAST funzione per informazioni sulle conversioni esplicite.

Esempio

L'esempio seguente riduce gli spazi vuoti iniziali e finali dalla stringa `' abc '`:

```
select '     abc    ' as untrim, trim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

Nell'esempio seguente vengono rimosse le virgolette doppie che circondano la stringa `"dog"`:

```
select trim('"' FROM '"dog"');

btrim
-------
dog
```

TRIM rimuove tutti i caratteri in trim_chars se questi si trovano all'inizio di stringa. L'esempio seguente riduce i caratteri "C", "D" e "G" quando si trovano all'inizio di VENUENAME che è una colonna 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
```

Funzione UPPER

Converte una stringa in maiuscolo. UPPER supporta caratteri multibyte UTF-8, fino a un massimo di quattro byte per carattere.

Sintassi

```
UPPER(string)
```

Argomenti

stringa

Il parametro di input è una stringa VARCHAR (o qualsiasi altro tipo di dati, ad esempio CHAR, che può essere convertito implicitamente in VARCHAR).

Tipo restituito

La funzione UPPER restituisce una stringa di caratteri che appartiene allo stesso tipo di dati della stringa di input.

Esempi

L'esempio seguente converte il campo CATNAME in lettere maiuscole:

```
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)
```

Funzione UUID

La funzione UUID genera un identificatore univoco universale (UUID).

UUIDs sono identificatori univoci globali che vengono comunemente utilizzati per fornire identificatori univoci per vari scopi, come:

- Identificazione dei record del database o di altre entità di dati.
- Generazione di nomi o chiavi univoci per file, directory o altre risorse.
- Monitoraggio e correlazione dei dati tra sistemi distribuiti.
- Fornitura di identificatori univoci per pacchetti di rete, componenti software o altre risorse digitali.

La funzione UUID genera un valore UUID unico con una probabilità molto elevata, anche su sistemi distribuiti e per lunghi periodi di tempo. UUIDs vengono generalmente generati utilizzando una combinazione del timestamp corrente, dell'indirizzo di rete del computer e di altri dati casuali o pseudo-casuali, garantendo che è altamente improbabile che ogni UUID generato entri in conflitto con altri UUID.

Nel contesto di una query SQL, la funzione UUID può essere utilizzata per generare identificatori univoci per nuovi record da inserire in un database o per fornire chiavi univoche per il partizionamento dei dati, l'indicizzazione o altri scopi in cui è richiesto un identificatore univoco.

> **Note**
>
> La funzione UUID non è deterministica.

Sintassi

```
uuid()
```

Argomenti

La funzione UUID non accetta argomenti.

Tipo restituito

UUID restituisce una stringa di identificazione univoca universale (UUID). Il valore viene restituito come stringa UUID canonica di 36 caratteri.

Esempio

L'esempio seguente genera un identificatore univoco universale (UUID). L'output è una stringa di 36 caratteri che rappresenta un identificatore univoco universale.

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

Funzioni relative alla privacy

AWS Clean Rooms fornisce funzioni per aiutarti a rispettare la conformità relativa alla privacy per le seguenti specifiche.

- Global Privacy Platform (GPP): una specifica dell'Interactive Advertising Bureau (IAB) che stabilisce un framework globale e standardizzato per la privacy e l'uso dei dati online. Per ulteriori informazioni sulle specifiche tecniche del GPP, consulta la documentazione della Global Privacy Platform su. GitHub
- Transparency and Consent Framework (TCF) — Un componente chiave del GPP, lanciato nel 2020, che fornisce un quadro tecnico standardizzato per aiutare le aziende a rispettare le normative sulla privacy come il Regolamento generale sulla protezione dei dati (GDPR) dell'UE. Il TCF consente ai clienti di concedere o negare il consenso alla raccolta e al trattamento dei dati. Per ulteriori informazioni sulle specifiche tecniche di TCF, consulta la documentazione TCF su. GitHub

Argomenti

- funzione consent_gpp_v1_decode
- funzione consent_tcf_v2_decode

funzione consent_gpp_v1_decode

La `consent_gpp_v1_decode` funzione viene utilizzata per decodificare i dati di consenso della Global Privacy Platform (GPP) v1. Prende come input la stringa di consenso codificata e restituisce i dati di consenso decodificati, che includono informazioni sulle preferenze di privacy e sulle scelte di consenso dell'utente. Questa funzione è utile quando si lavora con dati che includono informazioni sul consenso GPP v1, in quanto consente di accedere e analizzare i dati di consenso in un formato strutturato.

Sintassi

```
consent_gpp_v1_decode(gpp_string)
```

Argomenti

gpp_string

La stringa di consenso GPP v1 codificata.

Valori restituiti

Il dizionario restituito include le seguenti coppie chiave-valore:

- `version`: La versione della specifica GPP utilizzata (attualmente 1).
- `cmpId`: L'ID della piattaforma di gestione del consenso (CMP) che ha codificato la stringa di consenso.
- `cmpVersion`: la versione della CMP che ha codificato la stringa di consenso.
- `consentScreen`: L'ID della schermata nell'interfaccia utente CMP in cui l'utente ha fornito il consenso.
- `consentLanguage`: Il codice della lingua delle informazioni sul consenso.
- `vendorListVersion`: la versione dell'elenco dei fornitori utilizzata.
- `publisherCountryCode`: il codice del paese dell'editore.
- `purposeConsent`: un elenco di numeri interi che rappresentano gli scopi per i quali l'utente ha acconsentito.
- `purposeLegitimateInterest`: Un elenco di scopi IDs per i quali l'interesse legittimo dell'utente è stato comunicato in modo trasparente.

- `specialFeatureOptIns`: Un elenco di numeri interi che rappresentano le funzioni speciali che l'utente ha scelto.
- `vendorConsent`: un elenco di fornitori a IDs cui l'utente ha acconsentito.
- `vendorLegitimateInterest`: Un elenco di fornitori IDs per i quali l'interesse legittimo dell'utente è stato comunicato in modo trasparente.

Esempio

L'esempio seguente utilizza un singolo argomento, che è la stringa di consenso codificata. Restituisce un dizionario contenente i dati di consenso decodificati, comprese le informazioni sulle preferenze di privacy dell'utente, le scelte di consenso e altri metadati.

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

La struttura di base dei dati di consenso restituiti include informazioni sulla versione della stringa di consenso, i dettagli della CMP (Consent Management Platform), il consenso dell'utente e le scelte di interesse legittimo per scopi e fornitori diversi e altri metadati.

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

funzione consent_tcf_v2_decode

La `consent_tcf_v2_decode` funzione viene utilizzata per decodificare i dati di consenso di Transparency and Consent Framework (TCF) v2. Prende come input la stringa di consenso codificata e restituisce i dati di consenso decodificati, che includono informazioni sulle preferenze di privacy e

sulle scelte di consenso dell'utente. Questa funzione è utile quando si lavora con dati che includono informazioni sul consenso TCF v2, in quanto consente di accedere e analizzare i dati di consenso in un formato strutturato.

Sintassi

```
consent_tcf_v2_decode(tcf_string)
```

Argomenti

tcf_string

La stringa di consenso TCF v2 codificata.

Valori restituiti

La `consent_tcf_v2_decode` funzione restituisce un dizionario contenente i dati di consenso decodificati da una stringa di consenso Transparency and Consent Framework (TCF) v2.

Il dizionario restituito include le seguenti coppie chiave-valore:

Segmento principale

- `version`: La versione della specifica TCF utilizzata (attualmente 2).
- `created`: La data e l'ora in cui è stata creata la stringa di consenso.
- `lastUpdated`: la data e l'ora dell'ultimo aggiornamento della stringa di consenso.
- `cmpId`: l'ID della piattaforma di gestione del consenso (CMP) che ha codificato la stringa di consenso.
- `cmpVersion`: la versione della CMP che ha codificato la stringa di consenso.
- `consentScreen`: L'ID della schermata nell'interfaccia utente CMP in cui l'utente ha fornito il consenso.
- `consentLanguage`: Il codice della lingua delle informazioni sul consenso.
- `vendorListVersion`: la versione dell'elenco dei fornitori utilizzata.
- `tcfPolicyVersion`: La versione della politica TCF su cui si basa la stringa di consenso.
- `isServiceSpecific`: Un valore booleano che indica se il consenso è specifico per un particolare servizio o si applica a tutti i servizi.

- `useNonStandardStacks`: Un valore booleano che indica se vengono utilizzati pile non standard.
- `specialFeatureOptIns`: Un elenco di numeri interi che rappresentano le funzioni speciali che l'utente ha scelto di utilizzare.
- `purposeConsent`: Un elenco di numeri interi che rappresentano gli scopi per i quali l'utente ha acconsentito.
- `purposesLITransparency`: Un elenco di numeri interi che rappresentano gli scopi per i quali l'utente ha garantito la trasparenza degli interessi legittimi.
- `purposeOneTreatment`: Un valore booleano che indica se l'utente ha richiesto lo «scopo unico del trattamento» (ovvero, tutti gli scopi sono trattati allo stesso modo).
- `publisherCountryCode`: Il codice del paese dell'editore.
- `vendorConsent`: un elenco di fornitori a IDs cui l'utente ha acconsentito.
- `vendorLegitimateInterest`: Un elenco di fornitori IDs per i quali l'interesse legittimo dell'utente è stato comunicato in modo trasparente.
- `pubRestrictionEntry`: Un elenco di restrizioni per gli editori. Questo campo contiene l'ID dello scopo, il tipo di restrizione e l'elenco dei fornitori IDs soggetti a tale restrizione.

Segmento di fornitore divulgato

- `disclosedVendors`: un elenco di numeri interi che rappresentano i fornitori che sono stati resi noti all'utente.

Segmento relativo agli scopi del publisher

- `pubPurposesConsent`: un elenco di numeri interi che rappresentano gli scopi specifici dell'editore per i quali l'utente ha dato il consenso.
- `pubPurposesLITransparency`: Un elenco di numeri interi che rappresentano gli scopi specifici dell'editore per i quali l'utente ha garantito la trasparenza degli interessi legittimi.
- `customPurposesConsent`: Un elenco di numeri interi che rappresentano gli scopi personalizzati per i quali l'utente ha dato il consenso.
- `customPurposesLITransparency`: Un elenco di numeri interi che rappresentano gli scopi personalizzati per i quali l'utente ha garantito la trasparenza degli interessi legittimi.

Questi dati di consenso dettagliati possono essere utilizzati per comprendere e rispettare le preferenze di privacy dell'utente quando lavora con dati personali.

Esempio

L'esempio seguente utilizza un singolo argomento, che è la stringa di consenso codificata. Restituisce un dizionario contenente i dati di consenso decodificati, comprese le informazioni sulle preferenze di privacy dell'utente, le scelte di consenso e altri metadati.

```
from aws_clean_rooms.functions import consent_tcf_v2_decode

consent_string = "CO1234567890abcdef"
consent_data = consent_tcf_v2_decode(consent_string)

print(consent_data)
```

La struttura di base dei dati di consenso restituiti include informazioni sulla versione della stringa di consenso, i dettagli della CMP (Consent Management Platform), il consenso dell'utente e le scelte di interesse legittimo per scopi e fornitori diversi e altri metadati.

```
    /** 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],
};
```

Funzioni finestra

Le funzioni finestra ti consentono di creare query aziendali analitiche in modo più efficiente. Le funzioni finestra operano su una partizione o "finestra" di un insieme di risultati e restituiscono un valore per ogni riga in quella finestra. Tuttavia, le funzioni non finestra eseguono i calcoli in relazione a ogni riga del set di risultati. A differenza delle funzioni di gruppo che aggregano le righe dei risultati, le funzioni finestra mantengono tutte le righe nell'espressione della tabella.

I valori restituiti sono calcolati utilizzando i valori dai set di righe in quella finestra. Per ogni riga nella tabella, la finestra definisce un set di righe che viene utilizzato per calcolare gli attributi aggiuntivi. Una finestra viene definita utilizzando una specifica della finestra (la clausola OVER) e si basa su tre concetti principali:

- Partizionamento della finestra, che forma gruppi di righe (clausola PARTITION)
- Ordinamento della finestra, che definisce un ordine o una sequenza di righe all'interno di ciascuna partizione (clausola ORDER BY)
- Frame della finestra, che sono definiti in relazione a ciascuna riga per restringere ulteriormente l'insieme di righe (specifica ROWS)

Le funzioni finestra sono l'ultimo insieme di operazioni eseguite in una query ad eccezione della clausola ORDER BY finale. Tutti i join e tutte le clausole WHERE, GROUP BY e HAVING vengono completati prima che le funzioni finestra vengano elaborate. Pertanto, le funzioni finestra possono essere visualizzate solo nell'elenco di selezione o nella clausola ORDER BY. È possibile utilizzare più funzioni finestra all'interno di una singola query con diverse clausole del frame. È inoltre possibile utilizzare le funzioni finestra in altre espressioni scalari, come ad esempio CASE.

Riepilogo della sintassi della funzione finestra

Le funzioni della finestra seguono una sintassi standard, che è la seguente.

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

Qui, function è una delle funzioni descritte in questa sezione.

L'expr_list è il seguente.

```
expression | column_name [, expr_list ]
```

L'order_list è il seguente.

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

La frame_clause è la seguente.

```
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 }}
```

Arguments (Argomenti)

funzione

La funzione finestra. Per informazioni dettagliate, vedere le descrizioni della singola funzione.

OVER

La clausola che definisce la specifica della finestra. La clausola OVER è obbligatoria per le funzioni finestra e le contraddistingue da altre funzioni SQL.

PARTITION BY expr_list

(Facoltativo) La clausola PARTITION BY suddivide il set di risultati in partizioni in modo molto simile alla clausola GROUP BY. Se è presente una clausola di partizione, la funzione viene calcolata per le righe in ogni partizione. Se non viene specificata alcuna clausola di partizione,

una singola partizione contiene l'intera tabella e la funzione viene calcolata per quella tabella completa.

Le funzioni di classificazione DENSE_RANK, NTILE, RANK e ROW_NUMBER richiedono un confronto globale di tutte le righe nell'insieme di risultati. Quando viene utilizzata una clausola PARTITION BY, l'ottimizzatore della query può eseguire ciascuna aggregazione in parallelo distribuendo il carico di lavoro su più sezioni in base alle partizioni. Se la clausola PARTITION BY non è presente, la fase di aggregazione deve essere eseguita in serie su una singola sezione, che può avere un impatto negativo significativo sulle prestazioni, in particolare per i cluster di grandi dimensioni.

AWS Clean Roomsnon supporta stringhe letterali nelle clausole PARTITION BY.

ORDER BY order_list

(Facoltativo) La funzione finestra viene applicata alle righe all'interno di ciascuna partizione ordinata in base alla specifica dell'ordine in ORDER BY. Questa clausola ORDER BY è distinta e completamente non correlata a una clausola ORDER BY in una frame_clause. La clausola ORDER BY può essere utilizzata senza la clausola PARTITION BY.

Per le funzioni di classificazione, la clausola ORDER BY identifica le misure per i valori di classificazione. Per le funzioni di aggregazione, le righe partizionate devono essere ordinate prima che la funzione di aggregazione sia calcolata per ciascun frame. Per ulteriori informazioni sui tipi di funzione finestra, consultare Funzioni finestra.

Gli identificatori o le espressioni di colonna che valutano gli identificatori di colonna sono obbligatori nell'elenco degli ordini. Né le costanti né le espressioni costanti possono essere utilizzate come sostituti dei nomi delle colonne.

I valori NULL vengono trattati come il proprio gruppo, ordinati e classificati in base all'opzione NULLS FIRST o NULLS LAST. Per impostazione predefinita, i valori NULL vengono ordinati e classificati per ultimi in ordine ASC e ordinati e classificati per primi in ordine DESC.

AWS Clean Roomsnon supporta stringhe letterali nelle clausole ORDER BY.

Se viene omessa la clausola ORDER BY, l'ordine delle righe non è deterministico.

> **Note**
>
> In qualsiasi sistema paralleloAWS Clean Rooms, ad esempio quando una clausola ORDER BY non produce un ordinamento unico e totale dei dati, l'ordine delle righe non

è deterministico. In altre parole, se l'espressione ORDER BY produce valori duplicati (un ordinamento parziale), l'ordine di restituzione di tali righe potrebbe variare da una sequenza all'altra. AWS Clean Rooms A loro volta, le funzioni finestra potrebbero restituire risultati inattesi o incoerenti. Per ulteriori informazioni, consulta Ordinamento univoco dei dati per le funzioni finestra.

column_name

Nome di una colonna da partizionare o da ordinare.

ASC | DESC

Opzione che definisce l'ordinamento per l'espressione, come segue:

- ASC: crescente (ad esempio, dal più piccolo al più grande per i valori numerici e da 'A' a 'Z' per le stringhe di caratteri). Se non viene specificata alcuna opzione, i dati vengono ordinati in ordine crescente per impostazione predefinita.
- DESC: decrescente (ad esempio, dal più grande al più piccolo per i valori numerici e da 'Z' ad 'A' per le stringhe).

NULLS FIRST | NULLS LAST

Opzione che specifica se NULLS deve essere ordinato per primo, prima di valori non null, o per ultimo, dopo valori non null. Per impostazione predefinita, i NULLS vengono ordinati e classificati per ultimi in ordine ASC e ordinati e classificati per primi in ordine DESC.

frame_clause

Per le funzioni di aggregazione, la clausola frame perfeziona ulteriormente l'insieme di righe in una finestra della funzione quando si utilizza ORDER BY. Fornisce la capacità di includere o escludere set di righe all'interno del risultato ordinato. La clausola frame è composta dalla parola chiave ROWS e dagli specificatori associati.

La clausola frame non si applica alle funzioni di classificazione. Inoltre, la clausola frame non è richiesta quando non viene utilizzata alcuna clausola ORDER BY nella clausola OVER per una funzione di aggregazione. Se una clausola ORDER BY viene utilizzata per una funzione di aggregazione, è necessaria una clausola del frame esplicita.

Quando non viene specificata alcuna clausola ORDER BY, il frame implicito è illimitato: equivalente a ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

ROWS

Questa clausola definisce il frame della finestra specificando una compensazione fisica dalla riga corrente.

Questa clausola specifica le righe nella finestra o partizione corrente con cui deve essere combinato il valore nella riga corrente. Usa argomenti che specificano la posizione della riga, che può essere prima o dopo la riga corrente. Il punto di riferimento per tutti i frame della finestra è la riga corrente. Ogni riga diventa a sua volta la riga corrente mentre il frame della finestra scorre in avanti nella partizione.

Il frame può essere un semplice insieme di righe fino a includere la riga corrente:

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

Oppure può essere un insieme di righe tra due limiti.

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING indica che la finestra inizia nella prima riga della partizione; offset PRECEDING indica che la finestra inizia un numero di righe equivalente al valore di compensazione prima della riga corrente. UNBOUNDED PRECEDING è il valore predefinito.

CURRENT ROW indica che la finestra inizia o termina nella riga corrente.

UNBOUNDED FOLLOWING indica che la finestra termina nell'ultima riga della partizione; offset FOLLOWING indica che la finestra termina un numero di righe equivalente al valore di compensazione dopo la riga corrente.

offset identifica un numero fisico di righe prima o dopo la riga corrente. In questo caso, offset deve essere una costante che valuta un valore numerico positivo. Ad esempio, 5 FOLLOWING terminerà il frame 5 righe dopo la riga corrente.

Laddove BETWEEN non viene specificato, il frame è implicitamente limitato dalla riga corrente. Ad esempio, `ROWS 5 PRECEDING` è uguale a `ROWS BETWEEN 5 PRECEDING AND CURRENT ROW`. Inoltre, `ROWS UNBOUNDED FOLLOWING` è uguale a `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`.

> **Note**
>
> Non è possibile specificare un frame in cui il limite iniziale è maggiore del limite finale. Ad esempio, non è possibile specificare nessuno di questi frame:
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

Ordinamento univoco dei dati per le funzioni finestra

Se una clausola ORDER BY per una funzione finestra non produce un ordinamento univoco e totale dei dati, l'ordine delle righe è non deterministico. Se l'espressione ORDER BY produce valori duplicati (un ordinamento parziale), l'ordine di restituzione di tali righe può variare in più esecuzioni. In questo caso, le funzioni finestra possono restituire risultati inattesi o incoerenti.

Ad esempio, la seguente query restituisce risultati diversi su più esecuzioni. Si ottengono questi risultati diversi perché `order by dateid` non genera un ordinamento univoco dei dati per la funzione finestra 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
...
```

In questo caso, l'aggiunta di una seconda colonna ORDER BY alla funzione finestra potrebbe risolvere il problema.

```
select dateid, pricepaid,
sum(pricepaid) over(order by dateid, pricepaid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid | sumpaid
--------+-----------+---------
1827 |    234.00 |  234.00
1827 |    337.00 |  571.00
1827 |    347.00 |  918.00
...
```

Funzioni supportate

AWS Clean RoomsSpark SQL supporta due tipi di funzioni delle finestre: aggregate e di classificazione.

Queste sono le funzioni di aggregazione supportate:

- Funzione finestra CUME_DIST
- Funzione finestra DENSE_RANK
- Funzione FIRST window
- Funzione finestra FIRST_VALUE
- Funzione finestra LAG
- funzione LAST window
- Funzione finestra LAST_VALUE
- Funzione finestra LEAD

Queste sono le funzioni di classificazione supportate:

- Funzione finestra DENSE_RANK

- Funzione finestra PERCENT_RANK
- Funzione finestra RANK
- Funzione finestra ROW_NUMBER

Tabella di esempio per gli esempi della funzione finestra

Sono presenti esempi di funzione finestra specifici con la descrizione di ogni funzione. Alcuni esempi utilizzano una tabella denominata WINSALES, che contiene 11 righe, come illustrato nella tabella seguente.

SALESID	DATEID	SELLERID	BUYERID	QTÀ	QTY_SHIPPED
30001	2/8/2003	3	B	10	10
10001	24/12/2003	1	C	10	10
10005	24/12/2003	1	A	30	
40001	9/1/2004	4	A	40	
10006	18/01/2004	1	C	10	
20001	12/2/2004	2	B	20	20
40005	12/2/2004	4	A	10	10
20002	16/2/2004	2	C	20	20
30003	18/4/2004	3	B	15	
30004	18/4/2004	3	B	20	
30007	7/9/2004	3	C	30	

Funzione finestra CUME_DIST

Calcola la distribuzione cumulativa di un valore all'interno di una finestra o partizione. Supponendo l'ordinamento ascendente, la distribuzione cumulativa è determinata utilizzando questa formula:

```
count of rows with values <= x / count of rows in the window or partition
```

laddove x è uguale al valore nella riga corrente della colonna specificata nella clausola ORDER BY. Il seguente insieme di dati dimostra l'uso di questa formula:

```
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
```

L'intervallo del valore di restituzione è compreso tra 0 e 1, con questi valori compresi.

Sintassi

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Arguments (Argomenti)

OVER

Una clausola che specifica il partizionamento della finestra. La clausola OVER non può contenere una specifica del frame della finestra.

PARTITION BY partition_expression

Opzionale. Un'espressione che imposta l'intervallo di registrazioni per ciascun gruppo nella clausola OVER.

ORDER BY order_list

L'espressione su cui calcolare la distribuzione cumulativa. L'espressione deve avere o un tipo di dati numerici o essere implicitamente convertibile in uno. Se ORDER BY viene omesso, il valore di restituzione è 1 per tutte le righe.

Se ORDER BY non produce un ordinamento univoco, l'ordine delle righe è non deterministico. Per ulteriori informazioni, consulta Ordinamento univoco dei dati per le funzioni finestra.

Tipo restituito

FLOAT8

Esempi

L'esempio seguente calcola la distribuzione cumulativa della quantità per ciascun venditore:

```
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
```

Per una descrizione della tabella WINSALES, consultare Tabella di esempio per gli esempi della funzione finestra.

Funzione finestra DENSE_RANK

La funzione finestra DENSE_RANK determina la classificazione di un valore in un gruppo di valori, in base all'espressione ORDER BY nella clausola OVER. Se è presente la clausola PARTITION BY facoltativa, le classificazioni vengono ripristinate per ciascun gruppo di righe. Righe con valori uguali per i criteri di classificazione ricevono la stessa classificazione. La funzione DENSE_RANK differisce da RANK per un aspetto: se due o più righe si legano, non c'è spazio nella sequenza dei valori classificati. Ad esempio, se due righe sono classificate come 1, il livello successivo è 2.

È possibile avere funzioni di classificazione con diverse clausole PARTITION BY e ORDER BY nella stessa query.

Sintassi

```
DENSE_RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Arguments (Argomenti)

( )

La funzione non accetta argomenti, ma le parentesi vuote sono obbligatorie.

OVER

Le clausole finestra per la funzione DENSE_RANK.

PARTITION BY expr_list

Opzionale. Una o più espressioni che definiscono la finestra.

ORDER BY order_list

Opzionale. L'espressione su cui si basano i valori di classificazione. Se non viene specificato nessun PARTITION BY, ORDER BY utilizza l'intera tabella. Se ORDER BY viene omesso, il valore di restituzione è 1 per tutte le righe.

Se ORDER BY non produce un ordinamento univoco, l'ordine delle righe è non deterministico. Per ulteriori informazioni, consulta Ordinamento univoco dei dati per le funzioni finestra.

Tipo restituito

INTEGER

Esempi

Nel seguente esempio viene ordinata la tabella in base alla quantità venduta (in ordine decrescente) e viene assegnata una classificazione densa e una classificazione regolare a ciascuna riga. I risultati vengono ordinati dopo aver applicato i risultati della funzione finestra.

```
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)
```

Notare la differenza nelle classificazioni assegnate allo stesso insieme di righe quando le funzioni DENSE_RANK e RANK vengono utilizzate fianco a fianco nella stessa query. Per una descrizione della tabella WINSALES, consultare Tabella di esempio per gli esempi della funzione finestra.

Nel seguente esempio la tabella viene partizionata per SELLERID, ciascuna partizione viene ordinata in base alla quantità (in ordine decrescente) e viene assegnata una classificazione densa a ciascuna riga. I risultati vengono ordinati dopo aver applicato i risultati della funzione finestra.

```
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)
```

Per una descrizione della tabella WINSALES, consultare Tabella di esempio per gli esempi della funzione finestra.

Funzione FIRST window

Dato un insieme ordinato di righe, FIRST restituisce il valore dell'espressione specificata rispetto alla prima riga nella cornice della finestra.

Per informazioni sulla selezione dell'ultima riga nel frame, consulta funzione LAST window.

Sintassi

```
FIRST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Arguments (Argomenti)

espressione

L'espressione o colonna di destinazione su cui viene eseguita la funzione.

IGNORE NULLS

Quando questa opzione viene utilizzata con FIRST, la funzione restituisce il primo valore nel frame che non è NULL (o NULL se tutti i valori sono NULL).

RESPECT NULLS

Indica che AWS Clean Rooms deve includere valori nulli nella determinazione della riga da utilizzare. RESPECT NULLS è supportato come impostazione predefinita se non si specifica IGNORE NULLS.

OVER

Presenta le clausole finestra per la funzione.

PARTITION BY expr_list

Definisce la finestra per la funzione in termini di una o più espressioni.

ORDER BY order_list

Ordina le righe all'interno di ogni partizione. Se non viene specificata nessuna clausola PARTITION BY, ORDER BY ordina l'intera tabella. Se si specifica una clausola ORDER BY, è necessario anche specificare una frame_clause.

I risultati della funzione FIRST dipendono dall'ordine dei dati. I risultati sono non deterministici nei seguenti casi:

- Quando non è specificata alcuna clausola ORDER BY e una partizione contiene due valori diversi per un'espressione
- Quando l'espressione valuta valori diversi che corrispondono allo stesso valore nell'elenco ORDER BY.

frame_clause

Se una clausola ORDER BY viene utilizzata per una funzione di aggregazione, è necessaria una clausola del frame esplicita. La clausola frame raffina l'insieme di righe in una finestra della funzione, includendo o escludendo insieme di righe nel risultato ordinato. La clausola frame è composta dalla parola chiave ROWS e dagli specificatori associati. Per informazioni, consulta Riepilogo della sintassi della funzione finestra.

Tipo restituito

Queste funzioni supportano espressioni che utilizzano tipi di AWS Clean Rooms dati primitivi. Il tipo restituito è lo stesso del tipo di dati di expression.

Esempi

L'esempio seguente restituisce la capacità di posto per ciascuna posizione nella tabella VENUE, con i risultati ordinati in base alla capacità (da alta a bassa). La funzione FIRST viene utilizzata per selezionare il nome della sede che corrisponde alla prima fila del riquadro: in questo caso, la fila con il maggior numero di posti. I risultati sono partizionati per stato, quindi quando il valore VENUESTATE cambia, viene selezionato un nuovo primo valore. Il frame della finestra è illimitato, quindi lo stesso primo valore è selezionato per ogni riga in ogni partizione.

Per la California, `Qualcomm Stadium` ha il più alto numero di posti (`70561`), quindi questo nome è il primo valore per tutte le righe nella partizione `CA`.

```
select venuestate, venueseats, venuename,
first(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |          venuename            |         first
-----------+------------+-------------------------------
+------------------------------
CA         |      70561 | Qualcomm Stadium              | Qualcomm Stadium
CA         |      69843 | Monster Park                  | Qualcomm Stadium
CA         |      63026 | McAfee Coliseum               | Qualcomm Stadium
CA         |      56000 | Dodger Stadium                | Qualcomm Stadium
CA         |      45050 | Angel Stadium of Anaheim      | Qualcomm Stadium
CA         |      42445 | PETCO Park                    | Qualcomm Stadium
CA         |      41503 | AT&T Park                     | Qualcomm Stadium
CA         |      22000 | Shoreline Amphitheatre        | Qualcomm Stadium
CO         |      76125 | INVESCO Field                 | INVESCO Field
CO         |      50445 | Coors Field                   | INVESCO Field
DC         |      41888 | Nationals Park                | Nationals Park
FL         |      74916 | Dolphin Stadium               | Dolphin Stadium
FL         |      73800 | Jacksonville Municipal Stadium | Dolphin Stadium
FL         |      65647 | Raymond James Stadium         | Dolphin Stadium
FL         |      36048 | Tropicana Field               | Dolphin Stadium
...
```

Funzione finestra FIRST_VALUE

Dato un insieme ordinato di righe, FIRST_VALUE restituisce il valore dell'espressione specificata rispetto alla prima riga nel frame della finestra.

Per informazioni sulla selezione dell'ultima riga nel frame, consulta Funzione finestra LAST_VALUE.

Sintassi

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Arguments (Argomenti)

espressione

L'espressione o colonna di destinazione su cui viene eseguita la funzione.

IGNORE NULLS

Quando questa opzione viene utilizzata con FIRST_VALUE, la funzione restituisce il primo valore nel frame che non è NULL (o NULL se tutti i valori sono NULL).

RESPECT NULLS

Indica che AWS Clean Rooms deve includere valori nulli nella determinazione della riga da utilizzare. RESPECT NULLS è supportato come impostazione predefinita se non si specifica IGNORE NULLS.

OVER

Presenta le clausole finestra per la funzione.

PARTITION BY expr_list

Definisce la finestra per la funzione in termini di una o più espressioni.

ORDER BY order_list

Ordina le righe all'interno di ogni partizione. Se non viene specificata nessuna clausola PARTITION BY, ORDER BY ordina l'intera tabella. Se si specifica una clausola ORDER BY, è necessario anche specificare una frame_clause.

I risultati della funzione FIRST_VALUE dipendono dall'ordinamento dei dati. I risultati sono non deterministici nei seguenti casi:

- Quando non è specificata alcuna clausola ORDER BY e una partizione contiene due valori diversi per un'espressione
- Quando l'espressione valuta valori diversi che corrispondono allo stesso valore nell'elenco ORDER BY.

frame_clause

Se una clausola ORDER BY viene utilizzata per una funzione di aggregazione, è necessaria una clausola del frame esplicita. La clausola frame raffina l'insieme di righe in una finestra della funzione, includendo o escludendo insieme di righe nel risultato ordinato. La clausola frame è

composta dalla parola chiave ROWS e dagli specificatori associati. Per informazioni, consulta Riepilogo della sintassi della funzione finestra.

Tipo restituito

Queste funzioni supportano espressioni che utilizzano tipi di AWS Clean Rooms dati primitivi. Il tipo restituito è lo stesso del tipo di dati di expression.

Esempi

L'esempio seguente restituisce la capacità di posto per ciascuna posizione nella tabella VENUE, con i risultati ordinati in base alla capacità (da alta a bassa). La funzione FIRST_VALUE viene utilizzata per selezionare il nome del luogo corrispondente alla prima riga nel frame: in questo caso, la riga con il numero più alto di posti. I risultati sono partizionati per stato, quindi quando il valore VENUESTATE cambia, viene selezionato un nuovo primo valore. Il frame della finestra è illimitato, quindi lo stesso primo valore è selezionato per ogni riga in ogni partizione.

Per la California, `Qualcomm Stadium` ha il più alto numero di posti (`70561`), quindi questo nome è il primo valore per tutte le righe nella partizione `CA`.

```
select venuestate, venueseats, venuename,
first_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |           venuename            |         first_value
-----------+------------+--------------------------------
+-----------------------------
CA         |      70561 | Qualcomm Stadium               | Qualcomm Stadium
CA         |      69843 | Monster Park                   | Qualcomm Stadium
CA         |      63026 | McAfee Coliseum                | Qualcomm Stadium
CA         |      56000 | Dodger Stadium                 | Qualcomm Stadium
CA         |      45050 | Angel Stadium of Anaheim       | Qualcomm Stadium
CA         |      42445 | PETCO Park                     | Qualcomm Stadium
CA         |      41503 | AT&T Park                      | Qualcomm Stadium
CA         |      22000 | Shoreline Amphitheatre         | Qualcomm Stadium
CO         |      76125 | INVESCO Field                  | INVESCO Field
CO         |      50445 | Coors Field                    | INVESCO Field
```

```
DC         |      41888 | Nationals Park                 | Nationals Park
FL         |      74916 | Dolphin Stadium                | Dolphin Stadium
FL         |      73800 | Jacksonville Municipal Stadium | Dolphin Stadium
FL         |      65647 | Raymond James Stadium          | Dolphin Stadium
FL         |      36048 | Tropicana Field                | Dolphin Stadium
...
```

Funzione finestra LAG

La funzione finestra LAG restituisce i valori per una riga a una data compensazione sopra (prima) la riga corrente nella partizione.

Sintassi

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Arguments (Argomenti)

value_expr

L'espressione o colonna di destinazione su cui viene eseguita la funzione.

offset

Un parametro facoltativo che specifica il numero di righe prima della riga corrente per le quali restituire i valori. La compensazione può essere un integer costante o un'espressione che valuta un integer. Se non si specifica un offset, AWS Clean Rooms viene utilizzato 1 come valore predefinito. Una compensazione di `0` indica la riga corrente.

IGNORE NULLS

Una specifica facoltativa che indica che i valori nulli AWS Clean Rooms devono essere ignorati nella determinazione della riga da utilizzare. I valori null sono inclusi se IGNORE NULLS non è elencato.

> **Note**
>
> È possibile utilizzare un'espressione NVL o COALESCE per sostituire i valori null con un altro valore.

RESPECT NULLS

Indica che AWS Clean Rooms deve includere valori nulli nella determinazione della riga da utilizzare. RESPECT NULLS è supportato come impostazione predefinita se non si specifica IGNORE NULLS.

OVER

Specifica il partizionamento e l'ordinamento della finestra. La clausola OVER non può contenere una specifica del frame della finestra.

PARTITION BY window_partition

Un argomento facoltativo che imposta l'intervallo di registrazioni per ciascun gruppo nella clausola OVER.

ORDER BY window_ordering

Ordina le righe all'interno di ogni partizione.

La funzione della finestra LAG supporta espressioni che utilizzano qualsiasi tipo di AWS Clean Rooms dati. Il tipo di restituzione è lo stesso del tipo di dati di value_expr.

Esempi

L'esempio seguente mostra la quantità di biglietti venduti all'acquirente con un ID acquirente di 3 e il tempo in cui l'acquirente 3 ha acquistato i biglietti. Per confrontare ogni vendita con la vendita precedente per l'acquirente 3, la query restituisce la quantità venduta per ogni vendita precedente. Poiché non è stato effettuato alcun acquisto prima del 16/01/2008, il primo valore venduto precedentemente è null:

```
select buyerid, saletime, qtysold,
lag(qtysold,1) over (order by buyerid, saletime) as prev_qtysold
from sales where buyerid = 3 order by buyerid, saletime;

buyerid |      saletime       | qtysold | prev_qtysold
---------+---------------------+---------+--------------
3 | 2008-01-16 01:06:09 |       1 |
3 | 2008-01-28 02:10:01 |       1 |            1
3 | 2008-03-12 10:39:53 |       1 |            1
3 | 2008-03-13 02:56:07 |       1 |            1
3 | 2008-03-29 08:21:39 |       2 |            1
3 | 2008-04-27 02:39:01 |       1 |            2
```

```
3 | 2008-08-16 07:04:37 |       2 |            1
3 | 2008-08-22 11:45:26 |       2 |            2
3 | 2008-09-12 09:11:25 |       1 |            2
3 | 2008-10-01 06:22:37 |       1 |            1
3 | 2008-10-20 01:55:51 |       2 |            1
3 | 2008-10-28 01:30:40 |       1 |            2
(12 rows)
```

funzione LAST window

Dato un insieme ordinato di righe, la funzione LAST restituisce il valore dell'espressione rispetto all'ultima riga del frame.

Per informazioni sulla selezione della prima riga nel frame, consulta Funzione FIRST window.

Sintassi

```
LAST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Arguments (Argomenti)

espressione

L'espressione o colonna di destinazione su cui viene eseguita la funzione.

IGNORE NULLS

La funzione restituisce l'ultimo valore nel frame che non è NULL (o NULL se tutti i valori sono NULL).

RESPECT NULLS

Indica che AWS Clean Rooms deve includere valori nulli nella determinazione della riga da utilizzare. RESPECT NULLS è supportato come impostazione predefinita se non si specifica IGNORE NULLS.

OVER

Presenta le clausole finestra per la funzione.

PARTITION BY expr_list

Definisce la finestra per la funzione in termini di una o più espressioni.

ORDER BY order_list

Ordina le righe all'interno di ogni partizione. Se non viene specificata nessuna clausola PARTITION BY, ORDER BY ordina l'intera tabella. Se si specifica una clausola ORDER BY, è necessario anche specificare una frame_clause.

I risultati dipendono dall'ordinamento dei dati. I risultati sono non deterministici nei seguenti casi:

- Quando non è specificata alcuna clausola ORDER BY e una partizione contiene due valori diversi per un'espressione
- Quando l'espressione valuta valori diversi che corrispondono allo stesso valore nell'elenco ORDER BY.

frame_clause

Se una clausola ORDER BY viene utilizzata per una funzione di aggregazione, è necessaria una clausola del frame esplicita. La clausola frame raffina l'insieme di righe in una finestra della funzione, includendo o escludendo insieme di righe nel risultato ordinato. La clausola frame è composta dalla parola chiave ROWS e dagli specificatori associati. Per informazioni, consulta Riepilogo della sintassi della funzione finestra.

Tipo restituito

Queste funzioni supportano espressioni che utilizzano tipi di AWS Clean Rooms dati primitivi. Il tipo restituito è lo stesso del tipo di dati di expression.

Esempi

L'esempio seguente restituisce la capacità di posto per ciascuna posizione nella tabella VENUE, con i risultati ordinati in base alla capacità (da alta a bassa). La funzione LAST viene utilizzata per selezionare il nome della sede che corrisponde all'ultima riga del riquadro: in questo caso, la fila con il minor numero di posti. I risultati sono partizionati per stato, quindi quando il valore VENUESTATE cambia, viene selezionato un nuovo ultimo valore. Il frame della finestra è illimitato, quindi lo stesso ultimo valore è selezionato per ogni riga in ogni partizione.

Per la California, `Shoreline Amphitheatre` viene restituito per ogni riga nella partizione perché ha il numero più basso di posti (`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
...
```

Funzione finestra LAST_VALUE

In un set di righe ordinato, la funzione LAST_VALUE restituisce il valore dell'espressione rispetto all'ultima riga nel frame.

Per informazioni sulla selezione della prima riga nel frame, consulta Funzione finestra FIRST_VALUE.

Sintassi

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
```

```
)
```

Arguments (Argomenti)

espressione

L'espressione o colonna di destinazione su cui viene eseguita la funzione.

IGNORE NULLS

La funzione restituisce l'ultimo valore nel frame che non è NULL (o NULL se tutti i valori sono NULL).

RESPECT NULLS

Indica che AWS Clean Rooms deve includere valori nulli nella determinazione della riga da utilizzare. RESPECT NULLS è supportato come impostazione predefinita se non si specifica IGNORE NULLS.

OVER

Presenta le clausole finestra per la funzione.

PARTITION BY expr_list

Definisce la finestra per la funzione in termini di una o più espressioni.

ORDER BY order_list

Ordina le righe all'interno di ogni partizione. Se non viene specificata nessuna clausola PARTITION BY, ORDER BY ordina l'intera tabella. Se si specifica una clausola ORDER BY, è necessario anche specificare una frame_clause.

I risultati dipendono dall'ordinamento dei dati. I risultati sono non deterministici nei seguenti casi:

- Quando non è specificata alcuna clausola ORDER BY e una partizione contiene due valori diversi per un'espressione
- Quando l'espressione valuta valori diversi che corrispondono allo stesso valore nell'elenco ORDER BY.

frame_clause

Se una clausola ORDER BY viene utilizzata per una funzione di aggregazione, è necessaria una clausola del frame esplicita. La clausola frame raffina l'insieme di righe in una finestra della

funzione, includendo o escludendo insieme di righe nel risultato ordinato. La clausola frame è composta dalla parola chiave ROWS e dagli specificatori associati. Per informazioni, consulta Riepilogo della sintassi della funzione finestra.

Tipo restituito

Queste funzioni supportano espressioni che utilizzano tipi di AWS Clean Rooms dati primitivi. Il tipo restituito è lo stesso del tipo di dati di expression.

Esempi

L'esempio seguente restituisce la capacità di posto per ciascuna posizione nella tabella VENUE, con i risultati ordinati in base alla capacità (da alta a bassa). La funzione LAST_VALUE viene utilizzata per selezionare il nome del luogo corrispondente all'ultima riga nel frame: in questo caso, la riga con il numero più basso di posti. I risultati sono partizionati per stato, quindi quando il valore VENUESTATE cambia, viene selezionato un nuovo ultimo valore. Il frame della finestra è illimitato, quindi lo stesso ultimo valore è selezionato per ogni riga in ogni partizione.

Per la California, `Shoreline Amphitheatre` viene restituito per ogni riga nella partizione perché ha il numero più basso di posti (`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
...
```

Funzione finestra LEAD

La funzione finestra LEAD restituisce i valori per una riga a una data compensazione sotto (dopo) la riga corrente nella partizione.

Sintassi

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Arguments (Argomenti)

value_expr

L'espressione o colonna di destinazione su cui viene eseguita la funzione.

offset

Un parametro facoltativo che specifica il numero di righe sotto la riga corrente per le quali restituire i valori. La compensazione può essere un integer costante o un'espressione che valuta un integer. Se non si specifica un offset, AWS Clean Rooms viene utilizzato 1 come valore predefinito. Una compensazione di `0` indica la riga corrente.

IGNORE NULLS

Una specifica facoltativa che indica che i valori nulli AWS Clean Rooms devono essere ignorati nella determinazione della riga da utilizzare. I valori null sono inclusi se IGNORE NULLS non è elencato.

> **Note**
>
> È possibile utilizzare un'espressione NVL o COALESCE per sostituire i valori null con un altro valore.

RESPECT NULLS

Indica che AWS Clean Rooms deve includere valori nulli nella determinazione della riga da utilizzare. RESPECT NULLS è supportato come impostazione predefinita se non si specifica IGNORE NULLS.

OVER

Specifica il partizionamento e l'ordinamento della finestra. La clausola OVER non può contenere una specifica del frame della finestra.

PARTITION BY window_partition

Un argomento facoltativo che imposta l'intervallo di registrazioni per ciascun gruppo nella clausola OVER.

ORDER BY window_ordering

Ordina le righe all'interno di ogni partizione.

La funzione della finestra LEAD supporta espressioni che utilizzano qualsiasi tipo di AWS Clean Rooms dati. Il tipo di restituzione è lo stesso del tipo di dati di value_expr.

Esempi

L'esempio seguente fornisce la commissione per gli eventi nella tabella SALES per cui i biglietti sono stati venduti il 1° gennaio 2008 e il 2 gennaio 2008 e la commissione pagata per le vendite dei biglietti per la vendita successiva.

```
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)
```

Funzione finestra PERCENT_RANK

Calcola la classificazione percentuale di una data riga. La classificazione percentuale è determinata utilizzando questa formula:

`(x - 1) / (the number of rows in the window or partition - 1)`

laddove x è la classificazione della riga corrente. Il seguente insieme di dati dimostra l'uso di questa formula:

```
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
```

L'intervallo del valore di restituzione è compreso tra 0 e 1, con questi valori compresi. La prima riga in qualsiasi insieme ha un PERCENT_RANK di 0.

Sintassi

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Arguments (Argomenti)

( )

La funzione non accetta argomenti, ma le parentesi vuote sono obbligatorie.

OVER

Una clausola che specifica il partizionamento della finestra. La clausola OVER non può contenere una specifica del frame della finestra.

PARTITION BY partition_expression

Opzionale. Un'espressione che imposta l'intervallo di registrazioni per ciascun gruppo nella clausola OVER.

ORDER BY order_list

Opzionale. L'espressione su cui calcolare la classificazione percentuale. L'espressione deve avere o un tipo di dati numerici o essere implicitamente convertibile in uno. Se ORDER BY viene omesso, il valore di restituzione è 0 per tutte le righe.

Se ORDER BY non produce un ordinamento univoco, l'ordine delle righe non è deterministico. Per ulteriori informazioni, consulta Ordinamento univoco dei dati per le funzioni finestra.

Tipo restituito

FLOAT8

Esempi

L'esempio seguente calcola la classificazione in percentuale delle quantità di vendita per ciascun venditore:

```
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
```

Per una descrizione della tabella WINSALES, consultare Tabella di esempio per gli esempi della funzione finestra.

Funzione finestra RANK

La funzione finestra RANK determina la classificazione di un valore in un gruppo di valori, in base all'espressione ORDER BY nella clausola OVER. Se è presente la clausola PARTITION BY facoltativa, le classificazioni vengono ripristinate per ciascun gruppo di righe. Le righe con valori uguali per i criteri di classificazione ricevono lo stesso rango. AWS Clean Roomsaggiunge il numero di righe legate alla classifica pareggiata per calcolare la classifica successiva e quindi i ranghi potrebbero non essere numeri consecutivi. Ad esempio, se due righe sono classificate come 1, il livello successivo è 3.

RANK è diverso dalla Funzione finestra DENSE_RANK per un aspetto: Per DENSE_RANK, se due o più righe si legano, non c'è spazio nella sequenza dei valori classificati. Ad esempio, se due righe sono classificate come 1, il livello successivo è 2.

È possibile avere funzioni di classificazione con diverse clausole PARTITION BY e ORDER BY nella stessa query.

Sintassi

```
RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Arguments (Argomenti)

( )

La funzione non accetta argomenti, ma le parentesi vuote sono obbligatorie.

OVER

Le clausole finestra per la funzione RANK.

PARTITION BY expr_list

Opzionale. Una o più espressioni che definiscono la finestra.

ORDER BY order_list

Opzionale. Definisce le colonne su cui si basano i valori di classificazione. Se non viene specificato nessun PARTITION BY, ORDER BY utilizza l'intera tabella. Se ORDER BY viene omesso, il valore di restituzione è 1 per tutte le righe.

Se ORDER BY non produce un ordinamento univoco, l'ordine delle righe non è deterministico. Per ulteriori informazioni, consulta Ordinamento univoco dei dati per le funzioni finestra.

Tipo restituito

INTEGER

Esempi

Nel seguente esempio la tabella viene ordinata in base alla quantità venduta (in ordine crescente per impostazione predefinita) e viene assegnata una classificazione a ciascuna riga. Un valore di classificazione pari a 1 è il valore di classificazione più alto. I risultati vengono ordinati dopo aver applicato i risultati della funzione finestra:

```
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)
```

Si noti che la clausola ORDER BY esterna in questo esempio include le colonne 2 e 1 per garantire che vengano AWS Clean Rooms restituiti risultati ordinati in modo coerente ogni volta che viene eseguita questa query. Ad esempio, le righe con vendite IDs 10001 e 10006 hanno valori QTY e RNK identici. Ordinare il risultato finale impostato in base alla colonna 1 assicura che la riga 10001 cada sempre prima di 10006. Per una descrizione della tabella WINSALES, consultare Tabella di esempio per gli esempi della funzione finestra.

Nel seguente esempio, l'ordinamento è invertito per la funzione finestra (`order by qty desc`). Ora il valore di classificazione più alto si applica al valore QTY più alto.

```
select salesid, qty,
rank() over (order by qty desc) as rank
from winsales
order by 2,1;

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

Per una descrizione della tabella WINSALES, consultare Tabella di esempio per gli esempi della funzione finestra.

Nel seguente esempio la tabella viene partizionata per SELLERID, ciascuna partizione viene ordinata in base alla quantità (in ordine decrescente) e viene assegnata una classificazione a ciascuna riga. I risultati vengono ordinati dopo aver applicato i risultati della funzione finestra.

```
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)
```

Funzione finestra ROW_NUMBER

Determina il numero ordinale di una riga corrente in un gruppo di righe, contando da 1, in base all'espressione ORDER BY nella clausola OVER. Se è presente la clausola PARTITION BY facoltativa, i numeri ordinali vengono ripristinati per ciascun gruppo di righe. Le righe con valori uguali per le espressioni ORDER BY ricevono i numeri di riga diversi in modo non deterministico.

Sintassi

```
ROW_NUMBER () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Arguments (Argomenti)

( )

La funzione non accetta argomenti, ma le parentesi vuote sono obbligatorie.

OVER

Le clausole finestra per la funzione ROW_NUMBER.

PARTITION BY expr_list

Opzionale. Una o più espressioni che definiscono la funzione ROW_NUMBER.

ORDER BY order_list

Opzionale. L'espressione che definisce le colonne su cui si basano i numeri di riga. Se non viene specificato nessun PARTITION BY, ORDER BY utilizza l'intera tabella.

Se ORDER BY non produce un ordinamento univoco o viene omesso, l'ordine delle righe non è deterministico. Per ulteriori informazioni, consulta Ordinamento univoco dei dati per le funzioni finestra.

Tipo restituito

BIGINT

Esempi

L'esempio seguente esegue la partizione della tabella in SELLERID e ordina ciascuna partizione in base a QTY (in ordine crescente), quindi assegna un numero di riga a ogni riga. I risultati vengono ordinati dopo aver applicato i risultati della funzione finestra.

```
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)
```

Per una descrizione della tabella WINSALES, consultare Tabella di esempio per gli esempi della funzione finestra.

AWS Clean Rooms Condizioni Spark SQL

Le condizioni sono dichiarazioni di una o più espressioni e operatori logici che restituiscono vero, falso o sconosciuto. A volte le condizioni vengono anche chiamate predicati.

Sintassi

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

> **Note**
>
> Tutte le corrispondenze tra modelli LIKE e i confronti di stringhe fanno distinzione tra maiuscole e minuscole. Ad esempio, "A" e "a" non corrispondono. Tuttavia, è possibile effettuare una corrispondenza tra modelli che non distingue tra maiuscole e minuscole usando il predicato ILIKE.

Le seguenti condizioni SQL sono supportate in AWS Clean Rooms Spark SQL.

Argomenti

- Operatori di confronto
- Condizioni logiche
- Condizioni di corrispondenza di modelli
- Condizione di intervallo BETWEEN
- Condizione Null

- Condizione EXISTS
- Condizione IN

Operatori di confronto

Le condizioni di confronto esprimono le relazioni logiche tra due valori. Tutte le condizioni di confronto sono operatori binari con un tipo restituito booleano.

AWS Clean Rooms Spark SQL supporta gli operatori di confronto descritti nella tabella seguente.

Operatore	Sintassi	Descrizione
!	`!expression`	L'`NOT`operatore logico. Utilizzato per negare un'espressione booleana, ossia restituisce l'opposto del valore dell'espressione. Il! L'operatore può anche essere combinato con altri operatori logici, come AND e OR, per creare espressioni booleane più complesse.
<	`a < b`	L'operatore less than comparison. Utilizzato per confrontare due valori e determinare se il valore a sinistra è inferiore a quello a destra.
>	`a > b`	L'operatore maggiore di confronto. Utilizzato per confrontare due valori e determinare se il valore a sinistra è maggiore del valore a destra.

Operatore	Sintassi	Descrizione
<=	`a <= b`	L'operatore di confronto minore o uguale. Utilizzato per confrontare due valori e restituisce `true` se il valore a sinistra è minore o uguale al valore a destra e in `false` altro modo.
>=	`a >= b`	L'operatore di confronto maggiore o uguale a. Utilizzato per confrontare due valori e determinare se il valore a sinistra è maggiore o uguale al valore a destra.
=	`a = b`	L'operatore di confronto delle uguaglianze, che confronta due valori e restituisce `true` se sono uguali e in `false` altro modo.
<> o **!=**	`a <> b` o `a != b`	L'operatore di confronto non uguale a, che confronta due valori e restituisce `true` se non sono uguali e `false` in altro modo.

Operatore	Sintassi	Descrizione
==	a == b	L'operatore di confronto delle uguaglianze standard, che confronta due valori e restituisce `true` se sono uguali e `false` in altro modo. **Note** L'operatore == fa distinzione tra maiuscole e minuscole quando si confrontano i valori delle stringhe. Se è necessario eseguire un confronto senza distinzione tra maiuscole e minuscole, è possibile utilizzare funzioni come UPPER () o LOWER () per convertire i valori nello stesso formato maiuscolo/minuscolo prima del confronto.

Esempi

Di seguito sono elencati alcuni semplici esempi di condizioni di confronto:

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

La seguente query restituisce i valori id per tutti gli scoiattoli che attualmente non stanno cercando cibo.

```
SELECT id FROM squirrels
WHERE !is_foraging
```

La seguente query restituisce le sedi con più di 10.000 posti nella tabella 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)
```

Questo esempio seleziona dalla tabella USERS gli utenti (USERID) ai quali piace la musica rock:

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
13
16
(5 rows)
```

Questo esempio seleziona dalla tabella USERS gli utenti (USERID) per i quali si sa se gli piace la musica 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
```

Esempi con una colonna TIME

La tabella di esempio seguente TIME_TEST ha una colonna TIME_VAL (tipo TIME) con tre valori inseriti.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

Nell'esempio seguente vengono estratte le ore da ogni timetz_val.

```
select time_val from time_test where time_val < '3:00';
   time_val
---------------
 00:00:00.5550
 00:58:00
```

L'esempio seguente confronta due valori letterali temporali.

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
 t
```

Esempi con una colonna TIMETZ

La tabella di esempio seguente TIMETZ_TEST ha una colonna TIMETZ_VAL (tipo TIMETZ) con tre valori inseriti.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

Nell'esempio seguente vengono selezionati solo i valori TIMETZ inferiori a `3:00:00` UTC. Il confronto viene effettuato dopo aver convertito il valore in UTC.

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

L'esempio seguente confronta due valori letterali TIMETZ. Il fuso orario viene ignorato per il confronto.

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

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

Condizioni logiche

Le condizioni logiche combinano il risultato di due condizioni per produrre un singolo risultato. Tutte le condizioni logiche sono operatori binari con un tipo restituito booleano.

Sintassi

```
expression
{ AND | OR }
expression
NOT expression
```

Le condizioni logiche usano una logica booleana a tre valori in cui il valore null rappresenta una relazione sconosciuta. Nella tabella riportata di seguito sono descritti i risultati delle condizioni logiche, dove E1 ed E2 rappresentano espressioni:

E1	E2	E1 ed E2	E1 o E2	NO E2
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

L'operatore NOT viene valutato prima di AND e l'operatore AND viene valutato prima dell'operatore OR. Eventuali parentesi utilizzate potrebbero sostituire questo ordine di valutazione predefinito.

Esempi

L'esempio seguente restituisce USERID e USERNAME dalla tabella USERS se agli utenti piacciono sia Las Vegas sia gli sport:

```
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)
```

L'esempio successivo restituisce USERID e USERNAME dalla tabella USERS se agli utenti piacciono Las Vegas o gli sport o entrambi. Questa query restituisce tutti gli output dell'esempio precedente più gli utenti a cui piacciono solo Las Vegas o gli sport.

```
select userid, username from users
where likevegas = 1 or likesports = 1
order by userid;

userid | username
--------+----------
1 | JSG99FHE
2 | PGL08LJI
3 | IFT66TXU
5 | AEB55QTM
6 | NDQ15VBM
9 | MSD36KVR
10 | WKW41AIW
13 | QTF33MCG
15 | OWU78MTR
16 | ZMG93CDD
22 | RHT62AGI
27 | KOY02CVE
```

```
29 | HUH27PKK
...
(18968 rows)
```

La query seguente usa le parentesi intorno alla condizione OR per trovare i luoghi a New York o in California in cui è stato rappresentato Macbeth:

```
select distinct venuename, venuecity
from venue join event on venue.venueid=event.venueid
where (venuestate = 'NY' or venuestate = 'CA') and eventname='Macbeth'
order by 2,1;

venuename                |   venuecity
-----------------------------------------+---------------
Geffen Playhouse                         | Los Angeles
Greek Theatre                            | Los Angeles
Royce Hall                               | Los Angeles
American Airlines Theatre                | New York City
August Wilson Theatre                    | New York City
Belasco Theatre                          | New York City
Bernard B. Jacobs Theatre                | New York City
...
```

La rimozione delle parentesi in questo esempio modifica la logica e i risultati della query.

L'esempio seguente utilizza l'operatore NOT:

```
select * from category
where not catid=1
order by 1;

catid | catgroup |  catname  |                 catdesc
-------+----------+-----------+--------------------------------------------
2 | Sports   | NHL       | National Hockey League
3 | Sports   | NFL       | National Football League
4 | Sports   | NBA       | National Basketball Association
5 | Sports   | MLS       | Major League Soccer
...
```

L'esempio seguente utilizza una condizione NOT seguita da una condizione AND:

```
select * from category
```

```
where (not catid=1) and catgroup='Sports'
order by catid;

catid | catgroup | catname |             catdesc
-------+----------+---------+---------------------------------
2 | Sports   | NHL     | National Hockey League
3 | Sports   | NFL     | National Football League
4 | Sports   | NBA     | National Basketball Association
5 | Sports   | MLS     | Major League Soccer
(4 rows)
```

Condizioni di corrispondenza di modelli

Un operatore di corrispondenza dei modelli cerca in una stringa uno schema specificato nell'espressione condizionale e restituisce true o false a seconda che trovi o meno una corrispondenza. AWS Clean Rooms Spark SQL utilizza i seguenti metodi per la corrispondenza dei modelli:

- Espressioni LIKE

  L'operatore LIKE confronta un'espressione di stringa, come il nome di colonna, con un modello che usa i caratteri jolly % (percentuale) e _ (sottolineatura). La corrispondenza di modello LIKE copre sempre l'intera stringa. LIKE esegue una corrispondenza con distinzione tra maiuscole e minuscole.

Argomenti

- LIKE
- RLIKE

LIKE

L'operatore LIKE confronta un'espressione di stringa, come il nome di colonna, con un modello che usa i caratteri jolly % (percentuale) e _ (sottolineatura). La corrispondenza di modello LIKE copre sempre l'intera stringa. Per trovare la corrispondenza con una sequenza in qualsiasi parte di una stringa, è necessario che il modello inizi e termini con un segno di percentuale.

LIKE distingue tra maiuscole e minuscole.

Sintassi

```
expression [ NOT ] LIKE | pattern [ ESCAPE 'escape_char' ]
```

Argomenti

espressione

Un'espressione di caratteri UTF-8 valida, come un nome di colonna.

LIKE

LIKE esegue una corrispondenza di modello che fa distinzione tra maiuscole e minuscole. Per eseguire una corrispondenza di modello che non fa distinzione tra maiuscole e minuscole per caratteri multibyte, utilizzare la funzione LOWER sull'espressione e modello con una condizione LIKE.

A differenza dei predicati di confronto, come = e <>, i predicati LIKE non ignorano implicitamente gli spazi finali. Per ignorare gli spazi finali, utilizzare RTRIM o trasmettere in modo esplicito una colonna CHAR a VARCHAR.

L'~~operatore è equivalente a LIKE. Inoltre l'!~~operatore è equivalente a NOT LIKE.

pattern

Un'espressione di caratteri UTF-8 valida con il modello da associare.

escape_char

Un'espressione di caratteri che eseguirà l'escape dei metacaratteri nel modello. Per impostazione predefinita sono due barre rovesciate ("\\").

Se il modello non contiene metacaratteri, allora il modello rappresenta solo la stringa stessa; in questo caso LIKE agisce come l'operatore di uguaglianza.

Entrambe le espressioni di caratteri possono essere tipi di dati CHAR o VARCHAR. Se differiscono, AWS Clean Rooms converte il modello al tipo di dati dell'espressione.

LIKE supporta i seguenti metacaratteri di corrispondenza di modelli:

Operatore	Descrizione
%	Abbina qualsiasi sequenza di zero o più caratteri.

Operatore	Descrizione
_	Abbina qualsiasi carattere singolo.

Esempi

La tabella seguente mostra esempi di corrispondenza di modelli usando LIKE:

Expression	Valori restituiti
`'abc' LIKE 'abc'`	True
`'abc' LIKE 'a%'`	True
`'abc' LIKE '_B_'`	False
`'abc' LIKE 'c%'`	False

L'esempio seguente trova tutte le città il cui nome inizia per "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
...
```

L'esempio seguente trova tutti gli utenti il cui cognome contiene "ten":

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
```

```
-------------
Christensen
Wooten
...
```

L'esempio seguente trova le città il cui terzo e quarto carattere sono «ea» . :

```
select distinct city from users where city like '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

Nell'esempio seguente viene usata la stringa con caratteri escape predefinita (\\) per ricercare stringhe che contengono "start_" (il testo `start` seguito da una sottolineatura _):

```
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)
```

L'esempio seguente specifica "^" come carattere di escape, quindi usa il carattere di escape per ricercare stringhe che contengono "start_" (il testo `start` seguito da una sottolineatura _):

```
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

L'operatore RLIKE consente di verificare se una stringa corrisponde a un modello di espressione regolare specificato.

Restituisce `true` se str corrisponde `regexp` o `false` meno.

Sintassi

```
rlike(str, regexp)
```

Argomenti

str

Un'espressione di tipo stringa

regexp

Un'espressione stringa. La stringa regex deve essere un'espressione regolare Java.

I valori letterali delle stringhe (inclusi i modelli regex) non sono inclusi nel nostro parser SQL. Ad esempio, per corrispondere a «\ abc», un'espressione regolare per regexp può essere «^\ abc$».

Esempi

L'esempio seguente imposta il valore del parametro di configurazione su. `spark.sql.parser.escapedStringLiterals` `true` Questo parametro è specifico del motore SQL Spark. Il `spark.sql.parser.escapedStringLiterals` parametro in Spark SQL controlla il modo in cui il parser SQL gestisce le stringhe letterali con escape. Se impostato su`true`, il parser interpreterà i caratteri della barra rovesciata (\) all'interno delle stringhe letterali come caratteri di

escape, consentendoti di includere caratteri speciali come nuove righe, tabulazioni e virgolette nei valori delle stringhe.

```
SET spark.sql.parser.escapedStringLiterals=true;
spark.sql.parser.escapedStringLiterals  true
```

Ad esempio, con`spark.sql.parser.escapedStringLiterals=true`, è possibile utilizzare la seguente stringa letterale nella query SQL:

```
SELECT 'Hello, world!\n'
```

Il carattere di nuova riga \n verrebbe interpretato come un carattere letterale di nuova riga nell'output.

L'esempio seguente esegue una corrispondenza del modello di espressione regolare. Il primo argomento viene passato all'operatore RLIKE. È una stringa che rappresenta il percorso di un file, in cui il nome utente effettivo viene sostituito dal pattern '****'. Il secondo argomento è il modello di espressione regolare usato per la corrispondenza. L'output (`true`) indica che la prima stringa (`'%SystemDrive%\Users\****'`) corrisponde all'espressione regolare pattern (`'%SystemDrive%\\Users.*'`).

```
SELECT rlike('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

Condizione di intervallo BETWEEN

Una condizione `BETWEEN` testa le espressioni per l'inclusione in un intervallo di valori, usando le parole chiave `BETWEEN` e `AND`.

Sintassi

```
expression [ NOT ] BETWEEN expression AND expression
```

Le espressioni possono essere numeriche, di caratteri o datetime, ma è necessario che siano compatibili. L'intervallo è inclusivo.

Esempi

Il primo esempio conta quante transazioni hanno registrato vendite di 2, 3 o 4 biglietti:

```
select count(*) from sales
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

La condizione di intervallo comprende i valori di inizio e di fine.

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

È necessario che la prima espressione in una condizione di intervallo sia il valore minore e la seconda espressione il valore maggiore. L'esempio seguente restituirà sempre zero righe a causa dei valori delle espressioni:

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

Tuttavia, l'applicazione del modificatore NOT invertirà la logica e produrrà un conto di tutte le righe:

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
172456
(1 row)
```

La query seguente restituisce un elenco di sedi con 20.000-50.000 posti a sedere:

```
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)
```

L'esempio seguente mostra l'utilizzo di BETWEEN per i valori di data:

```
select salesid, qtysold, pricepaid, commission, saletime
from sales
where eventid between 1000 and 2000
   and saletime between '2008-01-01' and '2008-01-03'
order by saletime asc;

salesid | qtysold | pricepaid | commission |   saletime
--------+---------+-----------+------------+---------------
  65082 |       4 |       472 |       70.8 | 1/1/2008 06:06
 110917 |       1 |       337 |      50.55 | 1/1/2008 07:05
 112103 |       1 |       241 |      36.15 | 1/2/2008 03:15
 137882 |       3 |      1473 |     220.95 | 1/2/2008 05:18
  40331 |       2 |        58 |        8.7 | 1/2/2008 05:57
 110918 |       3 |      1011 |     151.65 | 1/2/2008 07:17
  96274 |       1 |       104 |       15.6 | 1/2/2008 07:18
 150499 |       3 |       135 |      20.25 | 1/2/2008 07:20
  68413 |       2 |       158 |       23.7 | 1/2/2008 08:12
```

È importante notare che, sebbene l'intervallo di BETWEEN sia inclusivo, le date hanno un valore di ora predefinito di 00:00:00. L'unica riga valida del 3 gennaio per la query di esempio sarebbe una riga con saletime `1/3/2008 00:00:00`.

Condizione Null

Il NULL test di condizione per valori nulli, quando un valore è mancante o sconosciuto.

Sintassi

```
expression IS [ NOT ] NULL
```

Argomenti

espressione

Qualsiasi espressione come una colonna.

IS NULL

È true quando il valore dell'espressione è null e false quando ha un valore.

IS NOT NULL

È false quando il valore dell'espressione è null e true quando ha un valore.

Esempio

Questo esempio indica quante volte la tabella SALES contiene un valore null nel campo QTYSOLD:

```
select count(*) from sales
where qtysold is null;
count
-------
0
(1 row)
```

Condizione EXISTS

Le condizioni EXISTS testano l'esistenza di righe in una subquery e restituiscono true se una subquery restituisce almeno una riga. Se viene specificato NOT, la condizione restituisce true se una subquery restituisce nessuna riga.

Sintassi

```
[ NOT ] EXISTS (table_subquery)
```

Argomenti

EXISTS

È true se table_subquery restituisce almeno una riga.

NOT EXISTS

È true se table_subquery restituisce nessuna riga.

table_subquery

Una subquery che viene valutata una tabella con una o più colonne e una o più righe.

Esempio

Questo esempio restituisce tutti gli identificatori di data, una volta ciascuno, per ogni data che ha registrato una vendita di qualsiasi tipo:

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
1827
1828
1829
...
```

Condizione IN

Una IN condizione verifica l'appartenenza di un valore a un set di valori o a una sottoquery.

Sintassi

```
expression [ NOT ] IN (expr_list | table_subquery)
```

Arguments (Argomenti)

espressione

Un'espressione datetime, di caratteri o numerica che viene valutata rispetto a expr_list o table_subquery e deve essere compatibile con il tipo di dati di quell'elenco o subquery.

expr_list

Una o più espressioni delimitate da virgola o uno o più set di espressioni delimitate da virgola racchiusi tra parentesi.

table_subquery

Una subquery che viene valutata una tabella con una o più righe ma che è limitata a una sola colonna nel suo elenco di selezione.

IN | NOT IN

IN restituisce true se l'espressione è un membro della query o dell'elenco di espressioni. NOT IN restituisce true se l'espressione non è un membro. IN e NOT IN restituiscono NULL e nessuna riga nei casi seguenti: se l'espressione genera un valore null; se non ci sono valori expr_list o table_subquery corrispondenti e almeno una di queste righe di confronto restituisce un valore null.

Esempi

Le condizioni seguenti sono true solo per quei valori elencati:

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

Ottimizzazione per grandi elenchi IN

Per ottimizzare l'esecuzione delle query, un elenco IN che comprende più di 10 valori viene internamente valutato come un array scalare. Gli elenchi IN con meno di 10 valori vengono valutati come una serie di predicati OR. Questa ottimizzazione è supportata per i tipi di dati SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP e TIMESTAMPTZ.

Guarda l'output di EXPLAIN per la query per vedere l'effetto di questa ottimizzazione. Ad esempio:

```
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)
```

Interrogazione di dati annidati

AWS Clean Rooms offre un accesso compatibile con SQL ai dati relazionali e annidati.

AWS Clean Rooms utilizza la notazione punteggiata e l'array subscript per la navigazione dei percorsi quando si accede ai dati annidati. Inoltre, abilita il FROM elementi di clausola da iterare sugli array e utilizzare per operazioni di disaggregazione. I seguenti argomenti forniscono le descrizioni dei diversi modelli di interrogazione che combinano l'uso del tipo di array/struct/map dati con la navigazione tra percorsi e array, l'unnesting e i join.

Argomenti

Navigazione

AWS Clean Rooms consente la navigazione in matrici e strutture utilizzando rispettivamente la notazione `[...]` tra parentesi e punti. Inoltre, è possibile combinare la navigazione in strutture utilizzando la notazione a punti e gli array utilizzando la notazione con parentesi.

Example

Ad esempio, la seguente query di esempio presuppone che la colonna di dati dell'`c_orders`array sia una matrice con una struttura e un attributo è denominato. `o_orderkey`

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

È possibile utilizzare le notazioni con punti e parentesi in tutti i tipi di query, ad esempio filtraggio, join e aggregazione. È possibile utilizzare queste notazioni in una query in cui ci sono normalmente riferimenti di colonna.

Example

Nell'esempio seguente viene utilizzata un'istruzione SELECT che filtra i risultati.

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0].o_orderkey IS NOT NULL;
```

Example

Nell'esempio seguente viene utilizzata la navigazione con parentesi e punti nelle clausole GROUP BY e ORDER BY.

```
SELECT c_orders[0].o_orderdate,
       c_orders[0].o_orderstatus,
       count(*)
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderkey IS NOT NULL
GROUP BY c_orders[0].o_orderstatus,
         c_orders[0].o_orderdate
ORDER BY c_orders[0].o_orderdate;
```

Annullamento di query

Per annullare le interrogazioni, AWS Clean Rooms abilita l'iterazione sugli array. Lo fa navigando nell'array utilizzando la clausola FROM di una query.

Example

Utilizzando l'esempio precedente, nell'esempio seguente vengono eseguite interazioni sui valori dell'attributo per `c_orders`.

```
SELECT o FROM customer_orders_lineitem c, c.c_orders o;
```

La sintassi di annullamento nidificazione è un'estensione della clausola FROM. In SQL standard, la clausola FROM `x (AS) y` significa che in `y` vengono eseguite iterazioni su ogni tupla in relazione `x`. In questo caso, `x` si riferisce a una relazione e `y` si riferisce a un alias per relazione `x`. Allo stesso modo, la sintassi di unnesting che utilizza l'elemento della clausola FROM `x (AS) y` significa che `y` esegue un'iterazione su ogni valore nell'espressione dell'array. `x` In questo caso, `x` è un'espressione di matrice ed `y` è un alias per. `x`

Per la navigazione regolare, con l'operando sinistro si può anche utilizzare la notazione con punti e parentesi.

Example

Nell'esempio precedente:

- `customer_orders_lineitem c` è l'iterazione sulla tabella di `customer_order_lineitem` base
- `c.c_orders o` è l'iterazione su `c.c_orders array`

Per eseguire iterazioni sull'attributo o`_lineitems`, che è un array all'interno di un array, si aggiungono più clausole.

```
SELECT o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

AWS Clean Rooms supporta anche un indice di matrice quando si esegue l'iterazione sull'array utilizzando il AT parola chiave. La clausola esegue un'`x AS y AT z` iterazione sull'array x e genera il campo`z`, che è l'indice dell'array.

Example

Nell'esempio seguente viene illustrato il funzionamento di un indice di array:

```
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

Nell'esempio seguente sono eseguite iterazioni su un array scalare.

```
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

Nell'esempio seguente viene eseguita un'iterazione su un array di più livelli. L'esempio utilizza più clausole unnest per eseguire l'iterazione negli array più interni. Il `f.multi_level_array` AS l'array ripete. `multi_level_array` L'array AS elemento è l'iterazione sugli array all'interno. `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)
```

Semantica permissiva

Per impostazione predefinita, le operazioni di navigazione sui valori di dati annidati restituiscono null anziché restituire un errore quando la navigazione non è valida. La navigazione tra oggetti non è valida se il valore dei dati nidificati non è un oggetto o se il valore dei dati nidificati è un oggetto ma non contiene il nome dell'attributo utilizzato nella query.

Example

Ad esempio, la seguente query accede a un nome di attributo non valido nella colonna di dati nidificati: `c_orders`

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

La navigazione tra matrici restituisce null se il valore dei dati annidati non è un array o l'indice dell'array non è compreso nei limiti.

Example

La seguente query restituisce null perché `c_orders[1][1]` è fuori dai limiti.

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

Tipi di introspezione

Le colonne con tipi di dati annidati supportano funzioni di ispezione che restituiscono il tipo e altri tipi di informazioni relative al valore. AWS Clean Rooms supporta le seguenti funzioni booleane per le colonne di dati annidate:

- DECIMAL_PRECISION
- DECIMAL_SCALE
- IS_ARRAY
- IS_BIGINT
- IS_CHAR
- IS_DECIMAL
- IS_FLOAT
- IS_INTEGER
- IS_OBJECT
- IS_SCALARE
- IS_SMALLINT
- IS_VARCHAR
- JSON_TYPEOF

Tutte queste funzioni restituiscono false se il valore di input è null. IS_SCALAR, IS_OBJECT e IS_ARRAY si escludono a vicenda e coprono tutti i valori possibili ad eccezione di null. Per dedurre

i tipi corrispondenti ai dati, AWS Clean Rooms utilizza la funzione JSON_TYPEOF che restituisce il tipo (il livello più alto del) valore dei dati annidati, come mostrato nell'esempio seguente:

```
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
```

Cronologia dei documenti per AWS Clean Rooms SQL Reference

La tabella seguente descrive le versioni della documentazione per AWS Clean Rooms SQL Reference.

Per ricevere notifiche sugli aggiornamenti della documentazione, puoi sottoscrivere il feed RSS. Per iscriverti agli aggiornamenti RSS, abilita un plug-in RSS nel browser in uso.

Modifica	Descrizione	Data
Spark SQL supporta Hints	AWS Clean Rooms Spark SQL supporta i suggerimenti di interrogazione per ottimizzare le prestazioni delle query e ridurre i costi di elaborazione.	20 gennaio 2026
Spark SQL supporta CACHE TABLE	AWS Clean Rooms Spark SQL supporta il comando CACHE TABLE, che consente ai clienti di memorizzare nella cache le tabelle esistenti o di creare e memorizzare nella cache nuove tabelle dai risultati delle query per migliorare le prestazioni delle query.	22 ottobre 2025
Spark SQL supporta le funzioni FIRST e LAST Window	AWS Clean Rooms Spark SQL supporta le seguenti funzioni Window: FIRST e LAST.	12 giugno 2025
Aggiornamenti della documentazione della funzione Spark SQL	Aggiornamento della sola documentazione per riflettere accuratamente le funzioni Spark SQL supportate. È stata	20 maggio 2025

	rimossa la documentazione per 25 funzioni non supportat e, tra cui <=> operator, SIMILAR TO, LISTAGG e ARRAY_INSERT. Nomi delle funzioni corretti da DATEADD a DATE_ADD, DATEDIFF a DATE_DIFF, ISNULL a IS_NULL e ISNOTNULL a IS_NOT_NULL. È stato corretto un errore di battitura negli esempi DATE_PART.	
AWS Clean Rooms Spark SQL	I clienti possono ora eseguire query utilizzando alcune condizioni, funzioni, comandi e convenzioni SQL supportat i dal motore di analisi SQL Spark.	29 ottobre 2024
Comandi SQL e funzioni SQL: aggiornamento	Sono stati aggiunti esempi per la clausola JOIN, l'operatore di set EXCEPT, l'espressione condizion ale CASE e le seguenti funzioni: ANY_VALUE, NVL e COALESCE, NULLIF, CAST, CONVERT, CONVERT_T IMEZONE, EXTRACT, MOD, SIGN, CONCAT, FIRST_VAL UE e LAST_VALUE.	28 febbraio 2024

Funzioni SQL: aggiornamento	AWS Clean Rooms ora supporta le seguenti funzioni SQL: Array, SUPER e VARBYTE. Sono ora supportate le seguenti funzioni matematiche: ACOS, ASIN, ATAN, COT, DEXP ATAN2, PI, POW, RADIANS e SIN. Sono ora supportate le seguenti funzioni JSON: CAN_JSON_PARSE, JSON_PARSE e JSON_SERIALIZE.	6 ottobre 2023
Supporto per tipi di dati annidati	AWS Clean Rooms ora supporta i tipi di dati annidati.	30 agosto 2023
Regole di denominazione SQL: aggiornamento	Modifica solo della documentazione per chiarire i nomi delle colonne riservate.	16 agosto 2023
Disponibilità generale	AWS Clean Rooms SQL Reference è ora disponibile a livello generale.	31 luglio 2023

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.