SQL-Referenz

AWS Clean Rooms

AWS Clean Rooms: SQL-Referenz

Table of Contents

Überblick über SQL in AWS Clean Rooms

Willkommen bei der AWS Clean RoomsSQL-Referenz.

AWS Clean Roomsbasiert auf dem Industriestandard Structured Query Language (SQL), einer Abfragesprache, die aus Befehlen und Funktionen besteht, die Sie für die Arbeit mit Datenbanken und Datenbankobjekten verwenden. SQL setzt auch Regeln für die Verwendung von Datentypen, Ausdrücken und Literalen durch.

Die folgenden Themen enthalten allgemeine Informationen zu den Konventionen und Benennungsregeln, die in dieser SQL-Referenz verwendet werden.

Topics

Die folgenden Abschnitte enthalten Informationen zu den Literalen, Datentypen, SQL-Befehlen, Typen von SQL-Funktionen und SQL-Bedingungen, die Sie in AWS Clean Rooms verwenden können.

Weitere Informationen AWS Clean Rooms dazu finden Sie im AWS Clean RoomsBenutzerhandbuch und in der AWS Clean RoomsAPI-Referenz.

Konventionen für die SQL-Referenz

In diesem Abschnitt werden die Konventionen erklärt, die zum Schreiben der Syntax für die SQL-Ausdrücke, -Befehle und -Funktionen verwendet werden.

Zeichen	Beschreibung
GROSSBUCHSTABEN	Wörter in Großbuchstaben sind Schlüsselwörter.

Zeichen	Beschreibung
[ ]	Eckige Klammern bezeichnen optionale Argumente . Mehrere Argumente in eckigen Klammern zeigen an, dass Sie eine beliebige Anzahl der Argumente verwenden können. Argumente in eckigen Klammern, die jeweils in einer eigenen Zeile stehen, zeigen außerdem an, dass der -Parser die Argumente in der Reihenfolge erwartet, in der sie in der Syntax aufgelist et sind.
{ }	Geschweifte Klammern zeigen an, dass Sie nur eines der Argumente verwenden können, die innerhalb der Klammern stehen.
|	Pipe-Zeichen zeigen an, dass Sie zwischen den Argumenten wählen können.
Kursivschrift	Wörter in Kursivschrift zeigen Platzhalter an. Sie müssen das kursiv formatierte Wort durch den entsprechenden Wert ersetzen.
...	Auslassungspunkte zeigen an, dass Sie das Element davor wiederholen können.
'	Wörter in einfachen Anführungszeichen müssen zusammen mit den Anführungszeichen verwendet werden.

SQL-Namensregeln

In den folgenden Abschnitten werden die SQL-Benennungsregeln unter erklärt AWS Clean Rooms.

Themen

- Namen und Spalten für konfigurierte Tabellenzuordnungen
- Reservierte Wörter

Namen und Spalten für konfigurierte Tabellenzuordnungen

Mitglieder, die Abfragen durchführen können, verwenden konfigurierte Tabellenzuordnungsnamen als Tabellennamen in Abfragen. Konfigurierte Tabellenzuordnungsnamen und konfigurierte Tabellenspalten können in Abfragen mit Aliasnamen versehen werden.

Die folgenden Benennungsregeln gelten für konfigurierte Tabellenzuordnungsnamen, konfigurierte Tabellenspaltennamen und Aliase:

- Sie dürfen nur alphanumerische Zeichen, Unterstriche (_) oder Bindestriche (-) enthalten, dürfen jedoch nicht mit einem Bindestrich beginnen oder enden.
  - (Nur benutzerdefinierte Analyseregel) Sie können das Dollarzeichen ($) verwenden, aber kein Muster, das einer Zeichenkettenkonstante in Dollaranführungszeichen folgt.

    Eine Zeichenkettenkonstante in Dollaranführungszeichen besteht aus:

    - ein Dollarzeichen ($)
    - ein optionales „Tag“ mit null oder mehr Zeichen
    - ein weiteres Dollarzeichen
    - beliebige Zeichenfolge, aus der der Zeichenketteninhalt besteht
    - ein Dollarzeichen ($)
    - das gleiche Etikett, mit dem der Dollarkurs begann
    - ein Dollarzeichen

    Zum Beispiel: `$$invalid$$`
- Sie dürfen keine aufeinanderfolgenden Bindestriche (-) enthalten.
- Sie dürfen mit keinem der folgenden Präfixe beginnen:

  `padb_`, `pg_`, `stcs_`, `stl_`, `stll_`, `stv_`, `svcs_`, `svl_`, `svv_`, `sys_`, `systable_`
- Sie dürfen keine umgekehrten Schrägstriche (\), Anführungszeichen (') oder Leerzeichen ohne doppelte Anführungszeichen enthalten.
- Wenn sie mit einem nicht alphabetischen Zeichen beginnen, müssen sie in doppelten Anführungszeichen (“ „) stehen.
- Wenn sie einen Bindestrich (-) enthalten, müssen sie in doppelten Anführungszeichen (“ „) stehen.
- Sie müssen zwischen 1 und 127 Zeichen lang sein.
- Reservierte Wörter müssen in doppelten Anführungszeichen (“ „) stehen.

- Die folgenden Spaltennamen sind reserviert und können nicht verwendet werden AWS Clean Rooms (auch nicht mit Anführungszeichen):
  - OID
  - Tableoid
  - xmin
  - cmin
  - xmax
  - cmax
  - ctid

Reservierte Wörter

Im Folgenden finden Sie eine Liste der reservierten Wörter 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

Datentypunterstützung durch SQL Engine

AWS Clean Rooms unterstützt mehrere SQL-Engines und Dialekte. Das Verständnis der Datentypsysteme in diesen Implementierungen ist entscheidend für eine erfolgreiche Zusammenarbeit und Analyse von Daten. Die folgenden Tabellen zeigen die entsprechenden Datentypen in AWS Clean Rooms SQL, Snowflake SQL und Spark SQL.

Numerische Datentypen

Numerische Typen stehen für verschiedene Arten von Zahlen, von genauen ganzen Zahlen bis hin zu ungefähren Gleitkommawerten. Die Wahl des numerischen Typs wirkt sich sowohl auf die Speicheranforderungen als auch auf die Rechengenauigkeit aus. Integer-Typen variieren je nach Bytegröße, während Dezimal- und Gleitkommatypen unterschiedliche Genauigkeits- und Skalierungsoptionen bieten.

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
8-Byte-Ganzzahl	BIGINT	Nicht unterstützt	GROSSER GANZZAHL, LANG	Ganzzahlen mit Vorzeichen von -9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807.

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
4-Byte-Ganzzahl	INT	Nicht unterstützt	INT, INTEGER	Ganzzahlen mit Vorzeichen von -2.147.483.648 bis 2.147.483.647
2-Byte-Ganzzahl	SMALLINT	Nicht unterstützt	SMALLINT, KURZ	Ganzzahlen mit Vorzeichen von -32.768 bis 32.767
1-Byte-Ganzzahl	Nicht unterstützt	Nicht unterstützt	WINZIGE GANZZAHL, BYTE	Ganzzahlen mit Vorzeichen von -128 bis 127
Float mit doppelter Genauigkeit	DOPPELTE, DOPPELTE PRÄZISION	FLOAT, FLOAT4, FLOAT8, DOPPELT, DOPPELTE GENAUIGKEIT, REAL	DOUBLE	8-Byte-Gleitkommazahlen mit doppelter Genauigkeit
Gleitkommazahl mit einfacher Genauigkeit	ECHT, SCHWEBEN	Nicht unterstützt	FLOAT	4-Byte-Fließkommazahlen mit einfacher Genauigkeit

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
Dezimal (feste Genauigkeit)	DECIMAL	DEZIMAL, NUMERISCH, ZAHL **Note** Snowflake ordnet exakte numerische Typen mit kleinerer Breite (INT, BIGINT, SMALLINT usw.) automatisch als Alias für NUMBER zu.	DEZIMAL, NUMERISCH,	Vorzeichenbehaftete Dezimalzahlen mit beliebiger Genauigkeit
Dezimalzahl (mit Genauigkeit)	DEZIMAL (p)	DEZIMAL (p), ZAHL (p)	DEZIMAL (p)	Dezimalzahlen mit fester Genauigkeit

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
Dezimalzahl (mit Skala)	DECIMAL (p,s)	DEZIMAL (p, s), ZAHL (p, s)	DECIMAL (p,s)	Dezimalzahlen mit fester Genauigkeit und Skala

Boolesche Datentypen

Boolesche Typen stehen für einfache logische Werte. true/false Diese Typen sind in allen SQL-Engines konsistent und werden häufig für Flags, Bedingungen und logische Operationen verwendet.

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
Boolesch	BOOLEAN	BOOLEAN	BOOLEAN	Stellt Werte dar true/false

Datums- und Uhrzeit-Datentypen

Datums- und Uhrzeittypen verarbeiten Zeitdaten mit unterschiedlicher Genauigkeit und Zeitzonenerkennung. Diese Typen unterstützen verschiedene Formate zum Speichern von Daten, Uhrzeiten und Zeitstempeln sowie Optionen zum Ein- oder Ausschließen von Zeitzoneninformationen.

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
Date	DATUM	DATUM	DATUM	Datumswerte (Jahr, Monat, Tag) ohne Zeitzone

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
Zeit	TIME	Nicht unterstützt	Nicht unterstützt	Tageszeit in UTC, ohne Zeitzone
Zeit mit TZ	TIMETZ	Nicht unterstützt	Nicht unterstützt	Tageszeit in UTC, mit Zeitzone
Zeitstempel	TIMESTAMP (ZEITSTEMPEL)	ZEITSTEMPEL, ZEITSTEMPEL_NTZ	TIMESTAMP_NTZ	Zeitstempel ohne Zeitzone **Note** NTZ steht für „Keine Zeitzone“
Zeitstempel mit TZ	TIMESTAMPTZ	TIMESTAMP_LTZ	ZEITSTEMPEL, TIMESTAMP_LTZ	Zeitstempel mit lokaler Zeitzone **Note** LTZ steht für „Lokale Zeitzone“

Zeichendatentypen

Zeichentypen speichern Textdaten und bieten sowohl Optionen mit fester Länge als auch mit variabler Länge. Diese Typen verarbeiten Textzeichenfolgen und Binärdaten mit optionalen Längenangaben zur Steuerung der Speicherzuweisung.

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
Zeichen mit fester Länge	CHAR	CHAR, CHARACTER	CHAR, CHARACTER	Zeichenfolge mit fester Länge
Zeichen fester Länge mit Länge	CHAR(n)	CHAR(n), CHARACTER(n)	CHAR(n), CHARACTER(n)	Zeichenfolge mit fester Länge und angegebener Länge
Zeichen mit variabler Länge	VARCHAR	VARCHAR, STRING, TEXT	VARCHAR, ZEICHENFOLGE	Zeichenfolge mit variabler Länge
Zeichen mit variabler Länge und Länge	VARCHAR (n)	VARCHAR (n), ZEICHENFOLGE (n), TEXT (n)	VARCHAR (n)	Zeichenfolge mit variabler Länge und Längenbeschränkung
Binär	VARBYTE	BINARY, VARBINARY	BINARY	Binäre Bytefolge
Binär mit Länge	VARBYTE(n)	Nicht unterstützt	Nicht unterstützt	Binäre Bytefolge mit Längenbegrenzung

Strukturierte Datentypen

Strukturierte Typen ermöglichen eine komplexe Datenorganisation, indem mehrere Werte in einzelnen Feldern kombiniert werden. Dazu gehören Arrays für geordnete Sammlungen, Maps für Schlüssel-Wert-Paare und Strukturen zur Erstellung benutzerdefinierter Datenstrukturen mit benannten Feldern.

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
Array	ARRAY <type>	ARRAY (Typ)	ARRAY <type>	Geordnete Reihenfolge von Elementen desselben Typs **Note** Array-Typen müssen Elemente desselben Typs enthalten
Zuordnung	LANDKARTE<key, value>	MAP (Schlüssel, Wert)	LANDKARTE <key, value>	Sammlung von Schlüssel-Wert-Paaren **Note** Kartentypen müssen Elemente desselben

Datentyp	AWS Clean Rooms SQL	Snowflake-SQL	Spark-SQL	Description
				Typs enthalten
Struct	STRUKTUR< field1: type1, field2: type2>	OBJEKT (Feld1 Typ1, Feld2 Typ2)	STRUKTUR< field1: type1, field2: type2 >	Struktur mit benannten Feldern bestimmter Typen **Note** Die Syntax strukturierter Typen kann zwischen den Implementierungen leicht variieren
Super	SUPER	Nicht unterstützt	Nicht unterstützt	Flexibler Typ, der alle Datentypen unterstützt, auch komplexe Typen

AWS Clean Rooms Spark-SQL

AWS Clean Rooms Spark SQL setzt Regeln in Bezug auf die Verwendung von Datentypen, Ausdrücken und Literalen durch.

Weitere Informationen zu AWS Clean Rooms Spark SQL finden Sie im AWS Clean Rooms Benutzerhandbuch und in der AWS Clean Rooms API-Referenz.

Die folgenden Themen enthalten Informationen zu den Literalen, Datentypen, Befehlen, Funktionen und Bedingungen, die in AWS Clean Rooms Spark SQL unterstützt werden.

Themen

Literale

Ein Literal oder eine Konstante ist ein fester Datenwert, bestehend aus einer Zeichenfolge oder einer numerischen Konstante.

AWS Clean Rooms Spark SQL unterstützt verschiedene Arten von Literalen, darunter:

- Numerische Literale für Ganzzahlen, Dezimalzahlen und Gleitkommazahlen.
- Zeichenliterale, auch als Zeichenketten, Zeichenketten oder Zeichenkonstanten bezeichnet, werden zur Angabe eines Zeichenkettenwerts verwendet.
- Datums-, Uhrzeit- und Zeitstempelliterale, die mit Datentypen vom Typ Datetime verwendet werden. Weitere Informationen finden Sie unter Datums-, Zeit- und Zeitstempelliterale.
- Intervallliterale. Weitere Informationen finden Sie unter Intervallliterale.
- Boolesche Literale. Weitere Informationen finden Sie unter Boolesche Literale.
- Null-Literale, die zur Angabe eines Nullwerts verwendet werden.
- Nur TAB, CARRIAGE RETURN (CR) und LINE FEED (LF) Unicode-Steuerzeichen aus der allgemeinen Unicode-Kategorie (Cc) werden unterstützt.

AWS Clean Rooms Spark SQL unterstützt keine direkten Verweise auf Zeichenkettenliterale in der SELECT-Klausel, sie können jedoch in Funktionen wie CAST verwendet werden.

Operator + (Verkettung)

Verkettet numerische Literale, Zeichenkettenliterale und/oder Datetime- und Intervallliterale. Sie befinden sich auf beiden Seiten des +-Symbols und geben basierend auf den Eingaben auf beiden Seiten des +-Symbols unterschiedliche Typen zurück.

Syntax

```
numeric + string
```

```
date + time
```

```
date + timetz
```

Die Reihenfolge der Argumente kann umgekehrt werden.

Argumente

numeric literals

Literale oder Konstanten, die Zahlen darstellen, können Ganzzahlen oder Gleitkommazahlen sein.

string literals

Zeichenketten, Zeichenketten oder Zeichenkonstanten

date

A DATE Spalte oder ein Ausdruck, der implizit in eine umgewandelt wird DATE.

time

A TIME Spalte oder ein Ausdruck, der implizit in eine TIME.

timetz

A TIMETZ Spalte oder ein Ausdruck, der implizit in eine TIMETZ.

Beispiel

Die folgende Beispieltabelle TIME_TEST hat eine Spalte TIME_VAL (Typ TIME) mit drei eingefügten Werten.

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

Datentypen

Jeder Wert, den AWS Clean Rooms Spark SQL speichert oder abruft, hat einen Datentyp mit einem festen Satz von zugehörigen Eigenschaften. Datentypen werden deklariert, wenn Tabellen erstellt werden. Eine Datentyp beschränkt die Gruppe von Werten, den eine Spalte oder ein Argument enthalten kann.

In der folgenden Tabelle sind die Datentypen aufgeführt, die Sie in AWS Clean Rooms Spark SQL verwenden können.

Name des Datentyps	Datentyp	Aliasnamen	Description
ARRAY	the section called “Verschachtelter Typ”	Nicht zutreffend	Verschachtelter Array-Datentyp
BIGINT	the section called “Numerische Typen”	Nicht zutreffend	8-Byte-Ganzzahl mit Vorzeichen
BINARY	the section called “Binärer Typ”	Nicht zutreffend	Werte der Byte-Sequenz
BOOLEAN	the section called “Typ BOOLEAN”	BOOL	Logischer/Boolescher Wert (wahr/falsch)
BYTE	the section called “Numerische Typen”	Nicht zutreffend	1-Byte-Ganzzahlen mit Vorzeichen, von -128 bis 127
CHAR	the section called “Zeichentypen”	CHARACTER	Zeichenfolge mit fester Länge

Name des Datentyps	Datentyp	Aliasnamen	Description
DATE	the section called “Datum-/Uhrzeittypen”	Nicht zutreffend	Kalenderdatum (Jahr, Monat, Tag)
DECIMAL	the section called “Numerische Typen”	NUMERIC	Genauer Zahlenwert mit wählbarer Genauigkeit
FLOAT	the section called “Numerische Typen”	FLOAT8, DOPPELTE GENAUIGKEIT	Double (Gleitkommazahl mit doppelter Genauigkeit)
INTEGER	the section called “Numerische Typen”	INT	4-Byte-Ganzzahl mit Vorzeichen
INTERVAL	the section called “Datum-/Uhrzeittypen”	Nicht zutreffend	Zeitdauer in der Reihenfolge von Tag zu Uhrzeit oder von Jahr zu Monat
LONG	the section called “Numerische Typen”	Nicht zutreffend	8-Byte-Ganzzahlen mit Vorzeichen
MAP	the section called “Verschachtelter Typ”	Nicht zutreffend	Ordnen Sie den verschachtelten Datentyp zu
REAL	the section called “Numerische Typen”	FLOAT4	Gleitkommazahl mit einfacher Genauigkeit
SHORT	the section called “Numerische Typen”	Nicht zutreffend	2-Byte-Ganzzahlen mit Vorzeichen.
SMALLINT	the section called “Numerische Typen”	Nicht zutreffend	2-Byte-Ganzzahl mit Vorzeichen
STRUCT	the section called “Verschachtelter Typ”	Nicht zutreffend	Verschachtelter Struct-Datentyp

Name des Datentyps	Datentyp	Aliasnamen	Description
TIMESTAMP_LTZ	the section called “Datum-/Uhrzeittypen”	Nicht zutreffend	Tageszeit mit lokaler Zeitzone
TIMESTAMP_NTZ	the section called “Datum-/Uhrzeittypen”	Nicht zutreffend	Tageszeit ohne Zeitzone
TINYINT	the section called “Numerische Typen”	Nicht zutreffend	1-Byte-Ganzzahlen mit Vorzeichen, von -128 bis 127
VARCHAR	the section called “Zeichentypen”	ZEICHEN VARIIEREND	Zeichenfolge mit variabler Länge und benutzerdefiniertem Grenzwert

> **Note**
>
> Die verschachtelten Datentypen ARRAY, STRUCT und MAP sind derzeit nur für die benutzerdefinierte Analyseregel aktiviert. Weitere Informationen finden Sie unter Verschachtelter Typ.

Multibyte-Zeichen

Der Datentyp VARCHAR unterstützt Multibyte-UTF-8-Zeichen mit einer Länge von bis zu vier Bytes. Zeichen mit einer Länge von fünf Bytes oder mehr werden nicht unterstützt. Sie berechnen die Größe einer VARCHAR-Spalte, die Multibyte-Zeichen enthält, indem Sie die Anzahl der Zeichen mit der Anzahl der Bytes pro Zeichen multiplizieren. Wenn eine Zeichenfolge z. B. vier chinesischen Zeichen enthält und jedes Zeichen drei Bytes lang ist, dann ist eine VARCHAR(12)-Spalte erforderlich, um die Zeichenfolge zu speichern.

Der Datentyp VARCHAR bietet keine Unterstützung für die folgenden ungültigen UTF-8-Codepunkte:

`0xD800 – 0xDFFF` (Bytesequenzen: `ED A0 80–ED BF BF`)

Der Datentyp CHAR bietet keine Unterstützung für Multibyte-Zeichen.

Numerische Typen

Numerische Datentypen sind Ganzzahlen, Dezimalzahlen und Gleitkommazahlen.

Themen

Ganzzahl-Typen

Verwenden Sie die folgenden Datentypen, um ganze Zahlen verschiedener Bereiche zu speichern. Sie können keine Werte außerhalb des zulässigen Bereichs für jeden Typ speichern.

Name	Speicher	Bereich
SMALLINT	2 Bytes	-32768 bis +32767
SHORT	2 Bytes	-32768 bis +32767
INTEGER oder INT	4 Bytes	-2147483648 bis +2147483647
BIGINT	8 Bytes	-9223372036854775808 bis +9223372036854775807
LONG	8 Bytes	-9223372036854775808 bis +9223372036854775807

Typ DECIMAL oder NUMERIC

Verwenden Sie den Datentyp DECIMAL oder NUMERIC, um Werte mit benutzerdefinierter Genauigkeit zu speichern. Die Schlüsselwörter DECIMAL und NUMERIC können synonym verwendet werden. In diesem Dokument wird der Begriff dezimal für diesen Datentyp bevorzugt. Der Begriff

numerisch wird in der Regel als Oberbegriff für Ganzzahl-, Dezimalzahl- und Gleitkommazahl-Datentypen verwendet.

Speicher	Bereich
Variabel, bis zu 128 Bits für unkomprimierte DECIMAL-Typen	128-Bit-Ganzzahlen mit Vorzeichen und einer Genauigkeit von bis zu 38 Stellen

Definieren Sie eine DECIMAL-Spalte in einer Tabelle, indem Sie ein *precision* und angeben*scale*:

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

precision

Die Anzahl aller signifikanten Stellen im gesamten Wert: die Anzahl der Stellen auf beiden Seiten des Dezimaltrennzeichens. Die Zahl `48.2891` hat z. B. eine Genauigkeit von 6 und 4 Dezimalstellen. Wenn Sie nichts angeben, wird standardmäßig eine Genauigkeit von 18 verwendet. Die maximale Genauigkeit ist 38.

Wenn die Anzahl der Ziffern links vom Dezimaltrennzeichen in einem Eingabewert die Genauigkeit der Spalte abzüglich ihrer Skala überschreitet, kann der Wert nicht in die Spalte kopiert (oder eingefügt oder aktualisiert) werden. Diese Regel gilt für alle Werte, die nicht innerhalb des Bereichs der Spaltendefinition liegen. Der zulässige Wertebereich für eine `numeric(5,2)`-Spalte erstreckt sich z. B. von `-999.99` bis `999.99`.

scale

Die Anzahl aller Dezimalstellen im Nachkommabereich des Wertes bzw. die Anzahl der Stellen auf der rechten Seite des Dezimaltrennzeichens. Ganzzahlen haben keine Dezimalstellen. In einer Spaltenspezifikation muss der Wert für die Dezimalstellen kleiner oder gleich dem Wert für die Genauigkeit sein. Wenn Sie nichts angeben, werden standardmäßig 0 Dezimalstellen verwendet. Es sind maximal 37 Dezimalstellen zulässig.

Wenn ein Eingabewert, der in eine Tabelle geladen wird, mehr Dezimalstellen aufweist, als für die Spalte zulässig sind, wird der Wert auf die angegebene Dezimalstelle gerundet. Die Spalte PRICEPAID in der Tabelle SALES ist z. B. eine DECIMAL(8,2)-Spalte. Wenn ein DECIMAL(8,4)-Wert in die Spalte PRICEPAID eingefügt wird, wird der Wert auf 2 Dezimalstellen gerundet.

```
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)
```

Die Ergebnisse expliziter Umwandlungen von Werten, aus der Tabelle ausgewählt wurden, werden jedoch nicht gerundet.

> **Note**
>
> Der maximale positive Wert, der in eine DECIMAL(19,0)-Spalte eingefügt werden kann, ist `9223372036854775807` (2^{63} -1). Die maximale negative Wert ist `-9223372036854775807`. Wenn versucht wird, den Wert `9999999999999999999` (19 mal die Ziffer Neun) einzufügen, wird ein Überlauffehler verursacht. Unabhängig von der Position des Dezimaltrennzeichens ist `9223372036854775807` die längste Zeichenkette, die AWS Clean Rooms als DECIMAL-Zahl darstellen kann. Der größte Wert, der in eine DECIMAL(19,18)-Spalte geladen werden kann, ist z. B. `9.223372036854775807`. Diese Regeln haben folgenden Grund:
>
> - DEZIMALWERTE mit einer Genauigkeit von 19 oder weniger signifikanten Stellen werden intern als 8-Byte-Ganzzahlen gespeichert.
> - DEZIMALWERTE mit einer Genauigkeit von 20 bis 38 signifikanten Stellen werden als 16-Byte-Ganzzahlen gespeichert.

Hinweise zur Verwendung von 128-Bit-DECIMAL- oder -NUMERIC-Spalten

Weisen Sie DECIMAL-Spalten nur dann maximale Genauigkeit zu, wenn Sie sicher sind, dass Ihre Anwendung diese Präzision erfordert. 128-Bit-Werte belegen doppelt so viel Speicherplatz wie 64-Bit-Werte und können zu langsameren Ausführungszeiten von Abfragen führen.

Gleitkommazahl-Typen

Verwenden Sie die Datentypen REAL oder DOUBLE PRECISION, um numerische Werte mit variabler Genauigkeit zu speichern. Diese Typen sind ungenaue Typen, d. h. manche Werte werden als Annährungen gespeichert, so dass bei der Speicherung und Rückgabe eines bestimmten Wertes leichte Abweichungen auftreten können. Wenn Sie auf genaue Speicherungen und Berechnungen zurückgreifen müssen (z. B. bei Geldbeträgen), verwenden Sie den Datentyp DECIMAL.

REAL steht für das Gleitkommaformat mit einfacher Genauigkeit gemäß dem IEEE-Standard 754 für Gleitkomma-Arithmetik. Es hat eine Genauigkeit von etwa 6 Ziffern und einen Bereich von etwa 1E-37 bis 1E+37. Sie können diesen Datentyp auch als angeben. FLOAT4

DOUBLE PRECISION steht für das Gleitkommaformat mit doppelter Genauigkeit gemäß dem IEEE-Standard 754 für binäre Gleitkommaarithmetik. Es hat eine Genauigkeit von etwa 15 Ziffern und einen Bereich von etwa 1E-307 bis 1E+308. Sie können diesen Datentyp auch als FLOAT oder angeben FLOAT8.

Berechnungen mit numerischen Werten

In bezieht AWS Clean Rooms sich Berechnung auf binäre mathematische Operationen: Addition, Subtraktion, Multiplikation und Division. In diesem Abschnitt werden die erwarteten Ausgabetypen dieser Operationen beschrieben sowie die spezielle Formel, die verwendet wird, um die Genauigkeit und die Dezimalstellen zu ermitteln, wenn DECIMAL-Datentypen involviert sind.

Wenn bei der Verarbeitung von Abfragen numerische Werte berechnet werden, kann es vorkommen, dass eine Berechnung nicht möglich ist und die Abfrage einen numerischen Überlauffehler zurückgibt. Außerdem können Fälle auftreten, in denen die Dezimalstellen berechneter Werte variieren bzw. nicht den Erwartungen entsprechen. Bei manchen Operationen ist es möglich, diese Probleme durch explizite Umwandlungen (Typerweiterung) oder AWS Clean Rooms-Konfigurationsparameter zu umgehen.

Weitere Informationen zu den Ergebnissen ähnlicher Berechnungen mit SQL-Funktionen finden Sie unter AWS Clean Rooms Spark SQL-Funktionen.

Ausgabetypen für Berechnungen

Angesichts der Anzahl der in AWS Clean Rooms unterstützten numerischen Datentypen zeigt die folgende Tabelle die erwarteten Rückgabetypen für Additions-, Subtraktions-, Multiplikations- und Divisionsoperationen. Die erste Spalte links in der Tabelle enthält dabei den ersten Operanden und die oberste Zeile den zweiten Operanden der Berechnung.

Operand 1	Operand 2	Rückgabetyp
SMALLINT oder SHORT	SMALLINT oder SHORT	SMALLINT oder SHORT
SMALLINT oder SHORT	INTEGER	INTEGER
SMALLINT oder SHORT	BIGINT	BIGINT
SMALLINT oder SHORT	DECIMAL	DECIMAL
SMALLINT oder SHORT	FLOAT4	FLOAT8
SMALLINT oder SHORT	FLOAT8	FLOAT8
INTEGER	INTEGER	INTEGER
INTEGER	BIGINT oder LONG	BIGINT oder LONG
INTEGER	DECIMAL	DECIMAL
INTEGER	FLOAT4	FLOAT8
INTEGER	FLOAT8	FLOAT8
BIGINT oder LONG	BIGINT oder LONG	BIGINT oder LONG
BIGINT oder LONG	DECIMAL	DECIMAL
BIGINT oder LONG	FLOAT4	FLOAT8
BIGINT oder LONG	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL
DECIMAL	FLOAT4	FLOAT8
DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8
FLOAT8	FLOAT8	FLOAT8

Genauigkeit und Dezimalstellen der berechneten DECIMAL-Ergebnisse

In der folgenden Tabelle werden die Regeln für die Berechnung der Genauigkeit und der Dezimalstellen zusammengefasst, wenn mathematische Operationen DECIMAL-Ergebnisse ausgeben. In dieser Tabelle `p1` und `s1` stellen die Genauigkeit und Skalierung des ersten Operanden in einer Berechnung dar. `p2`und `s2` stellen die Genauigkeit und den Maßstab des zweiten Operanden dar. (Unabhängig von diesen Berechnungen ist die maximale Genauigkeit eines Ergebnisses 38 und der maximale Wert für die Dezimalstellen 38.)

Operation	Genauigkeit und Dezimalstellen in Ergebnissen
+ oder -	Skalieren = `max(s1,s2)` Genauigkeit = `max(p1-s1,p2-s2)+1+scale`
*	Skalieren = `s1+s2` Genauigkeit = `p1+p2+1`
/	Skalieren = `max(4,s1+p2-s2+1)` Genauigkeit = `p1-s1+ s2+scale`

Die Spalten PRICEPAID und COMMISSION in der Tabelle SALES sind z. B. DECIMAL(8,2)-Spalten. Wenn Sie PRICEPAID durch COMMISSION dividieren (oder umgekehrt), sieht die Formel wie folgt aus:

```
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)
```

Die folgende Berechnung stellt die allgemeine Regel für die Berechnung der Genauigkeit und der Dezimalstellen in Ergebnissen von Operationen dar, die mit DECIMAL-Werten sowie mit Satzoperatoren wie UNION, INTERSECT und EXCEPT oder Funktionen wie COALESCE und DECODE durchgeführt werden:

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

Beispielsweise wird eine DEC1 Tabelle mit einer DECIMAL (7,2) -Spalte mit einer DEC2 Tabelle mit einer DECIMAL (15,3) -Spalte verknüpft, um eine Tabelle zu erstellen. DEC3 Das Schema von DEC3 zeigt, dass die Spalte zu einer NUMERIC-Spalte (15,3) wird.

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

Im Beispiel oben wird die Formel wie folgt angewendet:

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

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

Hinweise für Divisionsoperationen

Bei Divisionsoperationen geben divide-by-zero Bedingungen Fehler zurück.

Für Dezimalstellen gilt ein Grenzwert von 100, nachdem die Genauigkeit und die Dezimalstellen berechnet wurden. Wenn im Ergebnis mehr als 100 Dezimalstellen berechnet wurden, wird das Ergebnis der Division wie folgt skaliert:

- Genauigkeit = `precision - (scale - max_scale)`
- Skalieren = `max_scale`

Wenn die berechnete Genauigkeit über dem maximalen Wert für die Genauigkeit (38) liegt, wird die Genauigkeit auf 38 reduziert, und für die Dezimalstellen wird die folgende Formel angewendet: `max(38 + scale - precision), min(4, 100))`

Überlaufbedingungen

Der Überlauf wird bei allen numerischen Berechnungen geprüft. DECIMAL-Daten mit einer Genauigkeit von 19 oder weniger werden als 64-Bit-Ganzzahlen gespeichert. DECIMAL-Daten mit einer Genauigkeit größer als 19 werden als 128-Bit-Ganzzahlen gespeichert. Die maximale Genauigkeit für alle DECIMAL-Werte beträgt 38, und es sind maximal 37 Dezimalstellen zulässig.

Überlauffehler treten auf, wenn ein Wert diese Grenzwerte überschreitet; diese gelten sowohl für Zwischenergebnissätze als auch für Endergebnissätze:

- Explizites Casting führt zu Laufzeitüberlauffehlern, wenn bestimmte Datenwerte nicht der in der Cast-Funktion angegebenen Genauigkeit oder Skala entsprechen. Sie können beispielsweise nicht alle Werte aus der Spalte PRICEPAID in der SALES-Tabelle (eine Spalte DECIMAL (8,2)) umwandeln und ein DECIMAL-Ergebnis (7,3) zurückgeben:

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

  Dieser Fehler tritt auf, weil einige der größeren Werte in der PRICEPAID-Spalte nicht umgewandelt werden können.
- Multiplikationsoperationen produzieren Ergebnisse, bei denen sich die Anzahl der Dezimalstellen aus der Summe der Dezimalstellen der einzelnen Operanden ergeben. Wenn beide Operanden z. B. 4 Dezimalstellen haben, hat das Ergebnis 8 Dezimalstellen, d. h. es bleiben nur 10 Stellen auf der linken Seite des Dezimaltrennzeichens übrig. Es kann daher relativ schnell passieren, dass Überlaufbedingungen bei der Multiplikation zweier großer Zahlen auftreten, die jeweils eine nicht unerheblich Anzahl von Dezimalstellen aufweisen.

Numerische Berechnungen mit den Typen INTEGER und DECIMAL

Wenn einer der Operanden in einer Berechnung den INTEGER-Datentyp hat und der andere Operand DECIMAL ist, wird der INTEGER-Operand implizit in DECIMAL umgewandelt.

- SMALLINT oder SHORT werden in DECIMAL (5,0) umgewandelt
- INTEGER wird in DECIMAL (10,0) umgewandelt
- BIGINT oder LONG wird als DECIMAL (19,0) umgewandelt

Wenn Sie z. B. SALES.COMMISSION, eine DECIMAL(8,2)-Spalte, mit SALES.QTYSOLD, einer SMALLINT-Spalte multiplizieren, wird diese Berechnung umgewandelt in:

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

Zeichentypen

Zu den Zeichendatentypen gehören die Typen CHAR (character) und VARCHAR (character varying).

Themen

CHAR oder CHARACTER

Verwenden Sie eine CHAR- oder CHARACTER-Spalte, um Zeichenfolgen mit einer festen Länge zu speichern. Diese Zeichenfolgen werden mit Leerzeichen aufgefüllt, sodass eine CHAR(10)-Spalte immer 10 Bytes im Speicher belegt.

```
char(10)
```

Eine CHAR-Spalte ohne Längenangabe wird als CHAR(1)-Spalte umgesetzt.

Die Datentypen CHAR und VARCHAR werden in Bezug auf ihre Bytes definiert, nicht über die Zeichen. Eine CHAR-Spalte kann nur Einzelbyte-Zeichen enthalten, d. h. eine CHAR(10)-Spalte kann eine Zeichenfolge mit einer maximalen Länge von 10 Bytes enthalten.

Name	Speicher	Bereich (Breite der Spalte)
CHAR oder CHARACTER	Länge der Zeichenfolge einschließlich der Leerzeichen am Ende (falls vorhanden)	4096 Bytes

VARCHAR oder CHARACTER VARYING

Verwenden Sie eine VARCHAR- oder CHARACTER VARYING-Spalte, um Zeichenfolgen mit einer variablen Länge und einem festen Grenzwert zu speichern. Diese Zeichenfolgen werden mit Leerzeichen aufgefüllt, d. h. eine VARCHAR(120)-Spalte besteht aus jeweils maximal 120 Einzelbyte-Zeichen, 60 Zeichen mit einer Länge von je 2 Bytes, 40 Zeichen mit einer Länge von je 3 Bytes oder 30 Zeichen mit einer Länge von je 4 Bytes.

```
varchar(120)
```

VARCHAR-Datentypen werden in Byte und nicht in Zeichen definiert. Eine VARCHAR-Spalte kann Multibyte-Zeichen bis zu einer maximalen Länge von vier Bytes pro Zeichen enthalten. Eine VARCHAR(12)-Spalte kann z. B. 12 Einzelbyte-Zeichen, 6 Zeichen mit einer Länge von je 2 Bytes, 4 Zeichen mit einer Länge von je 3 Bytes oder 3 Zeichen mit einer Länge von je 4 Bytes enthalten.

Name	Speicher	Bereich (Breite der Spalte)
VARCHAR oder CHARACTER VARYING	4 Bytes + alle Bytes für Zeichen, wobei jedes Zeichen zwischen 1 und 4 Bytes lang ist.	65535 Bytes (64 K -1)

Die Bedeutung von Leerzeichen am Ende

Die Datentypen CHAR und VARCHAR speichern Zeichenfolgen mit einer Länge von bis zu n Bytes. Der Versuch, eine längere Zeichenfolge in einer Spalte dieser Typen zu speichern, führt zu einem Fehler. Wenn es sich bei den zusätzlichen Zeichen jedoch ausschließlich um Leerzeichen (Leerzeichen) handelt, wird die Zeichenfolge auf die maximale Länge gekürzt. Wenn die Zeichenfolge kürzer als die maximal zulässige Länge ist, werden CHAR-Werte mit Leerzeichen aufgefüllt; VARCHAR-Werte speichern die Zeichenfolge dagegen ohne Leerzeichen.

Leerzeichen am Ende von CHAR-Werten sind semantisch immer ohne Bedeutung. Sie werden beim Vergleich zweier CHAR-Werte ignoriert, werden bei LENGTH-Berechnungen nicht berücksichtigt und werden entfernt, wenn Sie einen CHAR-Wert in einen anderen Zeichenfolgetyp konvertieren.

Leerzeichen am Ende von VARCHAR- und CHAR- Werten werden beim Vergleich von Werten als semantisch ohne Bedeutung behandelt.

Längenberechnungen geben die Länge von VARCHAR-Zeichenfolgen einschließlich der Leerzeichen am Ende zurück. Leerzeichen am Ende werden im Fall von Zeichenfolgen mit fester Länge nicht zu der Länge gezählt.

Datum-/Uhrzeittypen

Zu den Datetime-Datentypen gehören DATE, TIME, TIMESTAMP_LTZ und TIMESTAMP_NTZ.

Themen

DATE

Verwenden sie den Datentyp DATE, um einfache Kalenderdaten ohne Zeitstempel zu speichern.

Name	Speicher	Bereich	Behebung
DATE	4 Bytes	4713 v. Chr. bis 294276 n. Chr.	1 Tag

TIMESTAMP_LTZ

Verwenden Sie den TIMESTAMP_LTZ-Datentyp, um vollständige Zeitstempelwerte zu speichern, die das Datum, die Uhrzeit und die lokale Zeitzone enthalten.

TIMESTAMP steht für Werte, die aus Werten der Felder`year`,,, und `second` bestehen `month` `day` `hour`, `minute` mit der lokalen Zeitzone der Sitzung. Der `timestamp` Wert steht für einen absoluten Zeitpunkt.

TIMESTAMP in Spark ist ein benutzerdefinierter Alias, der einer der Varianten TIMESTAMP_LTZ und TIMESTAMP_NTZ zugeordnet ist. Sie können den Standard-Zeitstempeltyp über die Konfiguration auf TIMESTAMP_LTZ (Standardwert) oder TIMESTAMP_NTZ festlegen. `spark.sql.timestampType`

TIMESTAMP_NTZ

Verwenden Sie den TIMESTAMP_NTZ-Datentyp, um vollständige Zeitstempelwerte zu speichern, die das Datum und die Uhrzeit ohne die lokale Zeitzone enthalten.

TIMESTAMP steht für Werte, die Werte der Felder`year`,,, und enthalten. `month` `day` `hour` `minute` `second` Alle Operationen werden ohne Berücksichtigung einer Zeitzone ausgeführt.

TIMESTAMP in Spark ist ein benutzerdefinierter Alias, der einer der Varianten TIMESTAMP_LTZ und TIMESTAMP_NTZ zugeordnet ist. Sie können den Standard-Zeitstempeltyp über die Konfiguration auf TIMESTAMP_LTZ (Standardwert) oder TIMESTAMP_NTZ festlegen. `spark.sql.timestampType`

Beispiele mit Datum-/Uhrzeittypen

Die folgenden Beispiele zeigen Ihnen, wie Sie mit Datetime-Typen arbeiten, die von unterstützt werden. AWS Clean Rooms

Datumsbeispiele

Die folgenden Beispiele fügen Datumsangaben in verschiedenen Formaten ein und zeigen die Ausgabe an.

```
select * from datetable order by 1;

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

Wenn Sie einen Zeitstempel in eine DATE-Spalte eingeben, wird die Uhrzeit ignoriert, und nur das Datum wird geladen.

Zeit-Beispiele

Die folgenden Beispiele fügen TIME- und TIMETZ-Werte in verschiedenen Formaten ein und zeigen die Ausgabe an.

```
select * from timetable order by 1;
start_time |  end_time
------------------------
```

```
19:11:19  | 20:41:19+00
19:11:19  | 20:41:19+00
```

Datums-, Zeit- und Zeitstempelliterale

Im Folgenden finden Sie Regeln für die Arbeit mit Datums-, Uhrzeit- und Zeitstempelliteralen, die von Spark SQL unterstützt werden. AWS Clean Rooms

Datumsangaben

Die folgende Tabelle zeigt Eingabedaten, die gültige Beispiele für literale Datumswerte sind, die Sie in Tabellen laden können. AWS Clean Rooms Es wird davon ausgegangen, dass der standardmäßige `MDY DateStyle`-Modus aktiviert ist. Dieser Modus bedeutet, dass der Monatswert vor dem Tageswert steht, zum Beispiel in Zeichenfolgen wie `1999-01-08` und `01/02/00`.

> **Note**
>
> Datums- bzw. Zeitstempelliterale müssen in Anführungszeichen stehen, wenn Sie sie in eine Tabelle laden.

Eingegebenes Datum	Vollständiges Datum
January 8, 1999	January 8, 1999
1999-01-08	January 8, 1999
1/8/1999	January 8, 1999
01/02/00	January 2, 2000
2000-Jan-31	January 31, 2000
Jan-31-2000	January 31, 2000
31-Jan-2000	January 31, 2000
20080215	February 15, 2008
080215	February 15, 2008

Eingegebenes Datum	Vollständiges Datum
2008.366	December 31, 2008 (dreistellige Datumskomponente muss zwischen 001 und 366 liegen)

Times

Die folgende Tabelle zeigt Eingabezeiten, die gültige Beispiele für literale Zeitwerte sind, die Sie in AWS Clean Rooms Tabellen laden können.

Eingegebene Zeiten	Beschreibung (der Uhrzeitkomponente)
04:05:06.789	4.05 Uhr und 6,789 Sekunden
04:05:06	4.05 Uhr und 6 Sekunden
04:05	Genau 4.05 Uhr
040506	4.05 Uhr und 6 Sekunden
04:05 AM	Genau 4.05 Uhr, AM ist optional
04:05 PM	Genau 16.05 Uhr, Stundenwert muss kleiner als 12 sein
16:05	Genau 16.05 Uhr

Besondere Datums-/Uhrzeitwerte

Die folgende Tabelle zeigt spezielle Werte, die als Datetime-Literale und als Argumente für Datumsfunktionen verwendet werden können. Sie müssen in einfachen Anführungszeichen (') angegeben werden und werden bei der Verarbeitung der Abfrage in reguläre Zeitstempelwerte umgewandelt.

Sonderwert	Beschreibung
now	Wird zu der Startzeit der aktuellen Transaktion ausgewertet und gibt einen Zeitstempel mit auf

Sonderwert	Beschreibung
	Mikrosekunden genauer Uhrzeitkomponente zurück.
`today`	Wird zu dem entsprechenden Datum ausgewertet und gibt einen Zeitstempel mit Nullen für die Uhrzeitkomponente zurück.
`tomorrow`	Wird zu dem entsprechenden Datum ausgewertet und gibt einen Zeitstempel mit Nullen für die Uhrzeitkomponente zurück.
`yesterday`	Wird zu dem entsprechenden Datum ausgewertet und gibt einen Zeitstempel mit Nullen für die Uhrzeitkomponente zurück.

Die folgenden Beispiele zeigen, wie now und wie die Funktion DATE_ADD `today` funktioniert.

```
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)
```

Intervallliterale

Im Folgenden finden Sie Regeln für die Arbeit mit Intervallliteralen, die von AWS Clean Rooms Spark SQL unterstützt werden.

Mit Intervallliteralen können Zeiträume angegeben werden, beispielsweise `12 hours` oder `6 weeks`. Die Intervallliteralen können in Bedingungen und Berechnungen verwendet werden, die Datums-/Uhrzeitausdrücke enthalten.

> **Note**
>
> Sie können den INTERVAL-Datentyp nicht für Spalten in AWS Clean Rooms Tabellen verwenden.

Ein Intervall wird als Kombination des Schlüsselworts INTERVAL mit einer Zahlenangabe und einer unterstützten Datumskomponente ausgedrückt, zum Beispiel `INTERVAL '7 days'` oder `INTERVAL '59 minutes'`. Sie können Mengenangaben und Einheiten kombinieren und auf diese Weise das Intervall präzisieren. Beispiel: `INTERVAL '7 days, 3 hours, 59 minutes'`. Die Einheiten können abgekürzt und in ihren Pluralformen verwendet werden. Beispiele: `5 s`, `5 second` und `5 seconds` drücken dasselbe Intervall aus.

Wenn keine Datumskomponente angegeben wird, gibt der Intervallwert Sekunden an. Die Mengenangabe kann auch ein Dezimalwert sein. Beispiel: .: `0.5 days`).

Beispiele

Die folgenden Beispiele stellen eine Reihe von Berechnungen mit verschiedenen Intervallwerten dar.

Im folgenden Beispiel wird dem angegebenen Datum 1 Sekunde hinzugefügt.

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

Im folgenden Beispiel wird dem angegebenen Datum 1 Minute hinzugefügt.

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

```
(1 row)
```

Im folgenden Beispiel werden dem angegebenen Datum 3 Stunden und 35 Minuten hinzugefügt.

```
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)
```

Im folgenden Beispiel werden dem angegebenen Datum 52 Wochen hinzugefügt.

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

Im folgenden Beispiel werden dem angegebenen Datum 1 Woche, 1 Stunde, 1 Minute und 1 Sekunde hinzugefügt.

```
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)
```

Im folgenden Beispiel werden dem angegebenen Datum 12 Stunden (ein halber Tag) hinzugefügt.

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

Im folgenden Beispiel werden 4 Monate vom 31. März 2023 abgezogen und das Ergebnis ist der 30. November 2022. Die Berechnung berücksichtigt die Anzahl der Tage in einem Monat.

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

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

Intervalldatentypen und -literale

Sie können einen Intervalldatentyp verwenden, um Zeitdauern in Einheiten wie `seconds`, `minutes`, `hours`, `days`, `months` und `years` zu speichern. Intervalldatentypen und -literale können in Berechnungen von Datum und Uhrzeit verwendet werden, beispielsweise beim Hinzufügen von Intervallen zu Datumsangaben und Zeitstempeln, beim Summieren von Intervallen und beim Subtrahieren eines Intervalls von einem Datum oder Zeitstempel. Intervallliterale können als Eingabewerte für Spalten des Intervalldatentyps in einer Tabelle verwendet werden.

Syntax des Intervalldatentyps

So geben Sie einen Intervalldatentyp zum Speichern einer Zeitdauer in Jahren und Monaten an:

```
INTERVAL year_to_month_qualifier
```

So geben Sie einen Intervalldatentyp zum Speichern einer Zeitdauer in Tagen, Stunden, Minuten und Sekunden an:

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

Syntax des Intervallliterals

So geben Sie ein Intervallliteral zum Festlegen einer Zeitdauer in Jahren und Monaten an:

```
INTERVAL quoted-string year_to_month_qualifier
```

So geben Sie ein Intervallliteral zum Festlegen einer Zeitdauer in Tagen, Stunden, Minuten und Sekunden an:

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

Argumente

quoted-string

Gibt einen positiven oder negativen numerischen Wert an, der eine Menge und die Datums-/Uhrzeiteinheit als Eingabezeichenfolge angibt. Wenn die Zeichenfolge in Anführungszeichen nur eine Zahl enthält, werden die Einheiten anhand von year_to_month_qualifier oder AWS Clean Rooms day_to_second_qualifier bestimmt. `'23' MONTH` beispielsweise steht für `1 year 11 months`, `'-2' DAY` für `-2 days 0 hours 0 minutes 0.0 seconds`, `'1-2' MONTH` für `1 year 2 months` und `'13 day 1 hour 1 minute 1.123 seconds' SECOND` für `13 days 1 hour 1 minute 1.123 seconds`. Weitere Informationen zu Ausgabeformaten eines Intervalls finden Sie unter Intervallstile.

year_to_month_qualifier

Gibt den Bereich des Intervalls an. Wenn Sie einen Qualifier verwenden und ein Intervall mit Zeiteinheiten erstellen, die kleiner als der Qualifier sind, werden die kleineren Teile des Intervalls gekürzt und verworfen. AWS Clean Rooms Gültige Werte für year_to_month_qualifier sind:

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

day_to_second_qualifier

Gibt den Bereich des Intervalls an. Wenn Sie einen Qualifier verwenden und ein Intervall mit Zeiteinheiten erstellen, die kleiner als der Qualifier sind, werden die kleineren Teile des Intervalls AWS Clean Rooms gekürzt und verworfen. Gültige Werte für day_to_second_qualifier sind:

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

- `MINUTE TO SECOND`

  Die Ausgabe des INTERVAL-Literals wird auf die kleinste angegebene INTERVAL-Komponente gekürzt. Wenn Sie beispielsweise einen MINUTE-Qualifier verwenden, werden die Zeiteinheiten, die kleiner als MINUTE sind, AWS Clean Rooms verworfen.

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

  Der resultierende Wert wird auf `'1 day 01:01:00'` gekürzt.

fractional_precision

Optionaler Parameter, der die zulässige Anzahl an Nachkommastellen im Intervall angibt. Das Argument fractional_precision sollte nur angegeben werden, wenn Ihr Intervall SECOND enthält. `SECOND(3)` erzeugt beispielsweise ein Intervall, das nur drei Nachkommastellen erlaubt, z. B. 1,234 Sekunden. Die maximale Anzahl von Nachkommastellen ist sechs.

Die Sitzungskonfiguration `interval_forbid_composite_literals` bestimmt, ob ein Fehler zurückgegeben wird, wenn ein Intervall mit den Teilen YEAR TO MONTH und DAY TO SECOND angegeben wird.

Intervallarithmetik

Sie können Intervallwerte zusammen mit anderen Datums-/Uhrzeitwerten verwenden, um arithmetische Operationen durchzuführen. In den folgenden Tabellen werden die verfügbaren Operationen und die aus den einzelnen Optionen resultierenden Datentypen beschrieben.

> **Note**
>
> Operationen, die sowohl `date`- als auch `timestamp`-Ergebnisse liefern können, tun dies basierend auf der kleinsten Zeiteinheit, die in der Gleichung enthalten ist. Wenn Sie beispielsweise ein `interval` zu einem `date` hinzufügen, ist das Ergebnis ein `date`, wenn es sich um ein YEAR TO MONTH-Intervall handelt, und ein Zeitstempel bei einem DAY TO SECOND-Intervall.

Operationen, bei denen der erste Operand ein `interval` ist, führen zu den folgenden Ergebnissen für den angegebenen zweiten Operanden:

Operator	Date	Zeitstempel	Intervall	Numerischer Wert
-	–	–	Intervall	–
+	Date	Datum/Zeitstempel	Intervall	–
*	–	–	–	Intervall
/	–	–	–	Intervall

Operationen, bei denen der erste Operand ein `date` ist, führen zu den folgenden Ergebnissen für den angegebenen zweiten Operanden:

Operator	Date	Zeitstempel	Intervall	Numerischer Wert
-	Numerischer Wert	Intervall	Datum/Zeitstempel	Date
+	–	–	–	–

Operationen, bei denen der erste Operand ein `timestamp` ist, führen zu den folgenden Ergebnissen für den angegebenen zweiten Operanden:

Operator	Date	Zeitstempel	Intervall	Numerischer Wert
-	Numerischer Wert	Intervall	Zeitstempel	Zeitstempel
+	–	–	–	–

Intervallstile

- `postgres` – folgt dem PostgreSQL-Stil. Dies ist die Standardeinstellung.
- `postgres_verbose` – folgt dem ausführlichen PostgreSQL-Stil.
- `sql_standard` – folgt dem SQL-Standardstil für Intervallliterale.

Der folgende Befehl legt für den Intervallstil `sql_standard` fest.

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

postgres-Ausgabeformat

Im Folgenden sehen Sie das Ausgabeformat für den `postgres`-Intervallstil. Jeder numerische Wert kann negativ sein.

```
'<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
```

postgres_verbose-Ausgabeformat

Die postgres_verbose-Syntax ähnelt der postgres-Syntax, die Ausgaben von postgres_verbose enthalten jedoch auch die Zeiteinheit.

```
'[@] <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
```

sql_standard-Ausgabeformat

Die Werte für Intervalle vom Typ „Year to month“ werden wie folgt formatiert. Bei Angabe eines negativen Vorzeichens vor dem Intervall hat das Intervall einen negativen Wert. Das Vorzeichen gilt für das gesamte Intervall.

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

Die Werte für Intervalle vom Typ „Day to second“ werden wie folgt formatiert.

```
'[-]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
```

Beispiele für den Intervalldatentyp

Die folgenden Beispiele zeigen, wie INTERVAL-Datentypen mit Tabellen verwendet werden.

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

Beispiele für Intervallliterale

Die folgenden Beispiele werden mit dem Intervallstil `postgres` ausgeführt.

Das folgende Beispiel zeigt, wie ein INTERVAL-Literal von einem Jahr erstellt wird.

```
select INTERVAL '1' YEAR

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

Wenn Sie eine quoted-string angeben, die den Qualifier überschreitet, werden die verbleibenden Zeiteinheiten aus dem Intervall gekürzt. Im folgenden Beispiel wird aus einem Intervall von 13 Monaten 1 Jahr und 1 Monat, doch der verbleibende Monat wird aufgrund des Qualifiers YEAR weggelassen.

```
select INTERVAL '13 months' YEAR

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

Wenn Sie einen Qualifier verwenden, der kleiner als Ihre Intervallzeichenfolge ist, werden die verbleibenden Einheiten eingeschlossen.

```
select INTERVAL '13 months' MONTH

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

Wenn Sie eine Genauigkeit in Ihrem Intervall angeben, wird die Anzahl der Nachkommastellen auf die angegebene Genauigkeit gekürzt.

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

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

Wenn Sie keine Genauigkeit angeben, AWS Clean Rooms wird die maximale Genauigkeit von 6 verwendet.

```
select INTERVAL '1.23456789' SECOND

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

Das folgende Beispiel zeigt, wie ein Bereichsintervall erstellt wird.

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

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

Qualifier bestimmen die Einheiten, die Sie angeben. Das folgende Beispiel verwendet zwar dieselbe Zeichenfolge in Anführungszeichen von '2:2' wie im vorherigen Beispiel, AWS Clean Rooms erkennt jedoch, dass aufgrund des Qualifizierers unterschiedliche Zeiteinheiten verwendet werden.

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

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

Abkürzungen und Pluralformen der einzelnen Einheiten werden ebenfalls unterstützt. So sind beispielsweise `5s`, `5 second` und `5 seconds` äquivalente Intervalle. Unterstützte Einheiten sind Jahre, Monate, Stunden, Minuten und Sekunden.

```
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
```

Beispiele für Intervallliterale ohne Qualifier-Syntax

> **Note**
>
> Die folgenden Beispiele zeigen die Verwendung eines Intervallliterals ohne einen `YEAR TO MONTH`- oder `DAY TO SECOND`-Qualifier. Informationen zur Verwendung des empfohlenen Intervallliterals mit einem Qualifier finden Sie unter Intervalldatentypen und -literale.

Mit Intervallliteralen können Zeiträume angegeben werden, beispielsweise `12 hours` oder `6 months`. Die Intervallliterale können in Bedingungen und Berechnungen verwendet werden, die Datums-/Uhrzeitausdrücke enthalten.

Ein Intervallliteral wird als Kombination des Schlüsselworts INTERVAL mit einer Zahlenangabe und einer unterstützten Datumskomponente ausgedrückt, zum Beispiel `INTERVAL '7 days'` oder `INTERVAL '59 minutes'`. Sie können Mengenangaben und Einheiten kombinieren und auf diese Weise das Intervall präzisieren. Beispiel: `INTERVAL '7 days, 3 hours, 59 minutes'`. Die Einheiten können abgekürzt und in ihren Pluralformen verwendet werden. Beispiele: `5 s`, `5 second` und `5 seconds` drücken dasselbe Intervall aus.

Wenn keine Datumskomponente angegeben wird, gibt der Intervallwert Sekunden an. Die Mengenangabe kann auch ein Dezimalwert sein. Beispiel: .: `0.5 days`).

Die folgenden Beispiele stellen eine Reihe von Berechnungen mit verschiedenen Intervallwerten dar.

Im Folgenden wird dem angegebenen Datum 1 Sekunde hinzugefügt.

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

Im Folgenden wird dem angegebenen Datum 1 Minute hinzugefügt.

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

Im Folgenden werden dem angegebenen Datum 3 Stunden und 35 Minuten hinzugefügt.

```
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)
```

Im Folgenden werden dem angegebenen Datum 52 Wochen hinzugefügt.

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

Im Folgenden werden dem angegebenen Datum 1 Woche, 1 Stunde, 1 Minute und 1 Sekunde hinzugefügt.

```
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)
```

Im Folgenden werden dem angegebenen Datum 12 Stunden (ein halber Tag) hinzugefügt.

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

Im Folgenden werden 4 Monate vom 15. Februar 2023 abgezogen und das Ergebnis ist der 15. Oktober 2022.

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

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

Im Folgenden werden 4 Monate vom 31. März 2023 abgezogen und das Ergebnis ist der 30. November 2022. Die Berechnung berücksichtigt die Anzahl der Tage in einem Monat.

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

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

Typ BOOLEAN

Verwenden Sie den Typ BOOLEAN, um die Wahrheitswerte „wahr“ bzw. „falsch“ in einer Einzelbytespalte zu speichern. Die folgende Tabelle enthält Beschreibungen der drei möglichen Zustände der Booleschen Werte und deren entsprechenden Literale. Unabhängig von der eingegebenen Zeichenfolge werden Werte in Booleschen Spalten mit „t“ für den Wahrheitswert „wahr“ und „f“ für den Wahrheitswert „falsch“ gespeichert und angezeigt.

Status	Zulässige Literalwerte	Speicher
Wahr	`TRUE 't' 'true' 'y' 'yes' '1'`	1 Byte
Falsch	`FALSE 'f' 'false' 'n' 'no' '0'`	1 Byte
Unbekannt	`NULL`	1 Byte

Sie können einen IS-Vergleich nur verwenden, um einen booleschen Wert als Prädikat in der WHERE-Klausel zu prüfen. Sie können den IS-Vergleich nicht mit einem booleschen Wert in der SELECT-Liste verwenden.

Beispiele

Sie können eine Spalte vom Typ BOOLEAN verwenden, um den Status „Aktiv/Inaktiv“ für jeden Kunden in einer CUSTOMER-Tabelle zu speichern.

```
select * from customer;
custid | active_flag
```

```
-------+--------------
   100 | t
```

In diesem Beispiel wählt die folgende Abfrage Benutzer aus der USERS-Tabelle aus, die Sport mögen, Theater aber nicht mögen:

```
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)
```

Im folgenden Beispiel werden diejenigen Benutzer aus der Tabelle USERS ausgewählt, von denen nicht bekannt ist, ob sie Rockmusik mögen.

```
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)
```

Im folgenden Beispiel wird ein Fehler zurückgegeben, weil ein IS-Vergleich in der SELECT-Liste verwendet wird.

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

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

Das folgende Beispiel ist erfolgreich, weil statt des Vergleichs ein Gleichheitsvergleich (=) in der SELECT-Liste verwendet wirdIS.

```
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    |
```

Boolesche Literale

Die folgenden Regeln beziehen sich auf die Arbeit mit booleschen Literalen, die von Spark SQL unterstützt werden. AWS Clean Rooms

Verwenden Sie ein boolesches Literal, um einen booleschen Wert anzugeben, z. B. oder. `TRUE` `FALSE`

Syntax

```
TRUE | FALSE
```

Beispiel

Das folgende Beispiel zeigt eine Spalte mit dem angegebenen Wert von. TRUE

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

Binärer Typ

Verwenden Sie den BINARY-Datentyp, um nicht interpretierte Binärdaten fester Länge zu speichern und zu verwalten und so effiziente Speicher- und Vergleichsmöglichkeiten für bestimmte Anwendungsfälle bereitzustellen.

Der BINARY-Datentyp speichert eine feste Anzahl von Byte, unabhängig von der tatsächlichen Länge der gespeicherten Daten. Die maximale Länge beträgt in der Regel 255 Byte.

BINARY wird verwendet, um rohe, nicht interpretierte Binärdaten wie Bilder, Dokumente oder andere Dateitypen zu speichern. Die Daten werden genau so gespeichert, wie sie bereitgestellt werden, ohne jegliche Zeichenkodierung oder Interpretation. Binärdaten, die in BINARY-Spalten gespeichert sind, werden verglichen und sortiert byte-by-byte, und zwar auf der Grundlage der tatsächlichen Binärwerte und nicht auf der Grundlage von Zeichenkodierungs- oder Sortierungsregeln.

Die folgende Beispielabfrage zeigt die binäre Darstellung der Zeichenfolge`"abc"`. Jedes Zeichen in der Zeichenfolge wird durch seinen ASCII-Code im Hexadezimalformat dargestellt: „a“ ist 0x61, „b“ ist 0x62 und „c“ ist 0x63. In Kombination bilden diese Hexadezimalwerte die binäre Darstellung. `"616263"`

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

Verschachtelter Typ

AWS Clean Roomsunterstützt Abfragen mit Daten mit verschachtelten Datentypen, insbesondere den Spaltentypen AWS Glue STRUCT, ARRAY und MAP. Nur die benutzerdefinierte Analyseregel unterstützt verschachtelte Datentypen.

Insbesondere entsprechen verschachtelte Datentypen nicht der starren, tabellarischen Struktur des relationalen Datenmodells von SQL-Datenbanken.

Verschachtelte Datentypen enthalten Tags, die auf unterschiedliche Entitäten innerhalb der Daten verweisen. Sie können komplexe Werte wie Arrays, verschachtelte Strukturen und andere komplexe Strukturen enthalten, die Serialisierungsformaten wie JSON zugeordnet sind. Verschachtelte Datentypen unterstützen bis zu 1 MB an Daten für ein einzelnes Feld oder Objekt des verschachtelten Datentyps.

Themen

Typ ARRAY

Verwenden Sie den Typ ARRAY, um Werte darzustellen, die aus einer Folge von Elementen des Typs bestehen`elementType`.

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

Wird verwendet`containsNull`, um anzugeben, ob Elemente in einem ARRAY-Typ `null` Werte haben können.

MAP-Typ

Verwenden Sie den MAP-Typ, um Werte darzustellen, die aus einer Reihe von Schlüssel-Wert-Paaren bestehen.

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

`keyType`: der Datentyp von Schlüsseln

`valueType`: der Datentyp der Werte

Schlüssel dürfen keine `null` Werte haben. Wird verwendet`valueContainsNull`, um anzugeben, ob Werte eines MAP-Werts `null` Werte haben können.

Typ STRUCT

Verwenden Sie den Typ STRUCT, um Werte mit der Struktur darzustellen, die durch eine Folge von StructFields (Feldern) beschrieben wird.

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

StructField(Name, DataType, Nullwert): Stellt ein Feld in einem dar. StructType

`dataType`: der Datentyp eines Feldes

`name`: der Name eines Feldes

Wird verwendet`nullable`, um anzugeben, ob die Werte dieser Felder `null` Werte haben können.

Beispiele für verschachtelte Datentypen

Für den `struct<given:varchar, family:varchar>` Typ gibt es zwei Attributnamen:`given`, und`family`, die jeweils einem `varchar` Wert entsprechen.

Für den `array<varchar>` Typ wird das Array als eine Liste von angegeben`varchar`.

Der `array<struct<shipdate:timestamp, price:double>>` Typ bezieht sich auf eine Liste von Elementen mit `struct<shipdate:timestamp, price:double>` Typ.

Der `map` Datentyp verhält sich wie ein `array` von`structs`, wobei der Attributname für jedes Element im Array mit a bezeichnet wird `key` und ihm zugeordnet wird. `value`

Example

Der `map<varchar(20), varchar(20)>` Typ wird beispielsweise als `array<struct<key:varchar(20), value:varchar(20)>>` „where“ behandelt `key` und `value` bezieht sich auf die Attribute der Map in den zugrunde liegenden Daten.

Hinweise dazu, wie die Navigation in Arrays und Strukturen AWS Clean Rooms ermöglicht wird, finden Sie unterNavigation.

Hinweise dazu, wie die Iteration über Arrays AWS Clean Rooms ermöglicht wird, indem das Array mithilfe der FROM-Klausel einer Abfrage navigiert wird, finden Sie unter. Aufheben der Verschachtelung von Abfragen

Kompatibilität von Typen und Umwandlung zwischen Typen

In den folgenden Themen wird beschrieben, wie Typkonvertierungsregeln und Datentypkompatibilität in AWS Clean Rooms Spark SQL funktionieren.

Themen

- Kompatibilität
- Allgemeine Regeln zur Kompatibilität und zur Umwandlung
- Arten von impliziter Umwandlung

Kompatibilität

Es gibt verschiedene Datenbankoperationen, bei denen die Datentypen passend gemacht und den Literalwerten und Konstanten Datentypen zugewiesen werden. Hierzu gehören die folgenden:

- DML- (Data Manipulation Language-)Operationen über Tabellen
- UNION-, INTERSECT- und EXCEPT-Abfragen
- CASE-Ausdrücke
- Auswertung von Prädikaten wie LIKE oder IN
- Auswertung von SQL-Funktionen, bei denen Vergleiche durchgeführt oder Daten extrahiert werden
- Vergleiche mit mathematischen Operatoren

Die Ergebnisse dieser Operationen hängen von den Regeln zur Umwandlung von Typen und der Kompatibilität zwischen Datentypen ab. Kompatibilität bedeutet, dass ein one-to-one Abgleich eines bestimmten Werts und eines bestimmten Datentyps nicht immer erforderlich ist. Da einige Datentypen kompatibel sind, ist eine implizite Konvertierung oder ein Zwang möglich. Weitere Informationen finden Sie unter Arten von impliziter Umwandlung. Wenn Datentypen inkompatibel sind, können Sie manchmal einen Wert in einen anderen Datentyp umwandeln, indem Sie eine explizite Typumwandlungsfunktion verwenden.

Allgemeine Regeln zur Kompatibilität und zur Umwandlung

Beachten Sie die folgenden Regeln zur Kompatibilität und zur Typumwandlung:

- Datentypen aus derselben Kategorie sind i. d. R. miteinander kompatibel und können implizit ineinander konvertiert werden. Ein Beispiel hierfür sind numerische Datentypen.

  Sie können beispielsweise mit einer impliziten Umwandlung einen Dezimalwert in eine Spalte mit Ganzzahlen einfügen. Dabei werden Dezimalwerte auf eine Ganzzahl gerundet. Sie können auch einen Zahlenwert wie `2008` aus einem Datum extrahieren und den Wert in eine ganzzahlige Spalte einfügen.
- Numerische Datentypen erzwingen Überlaufbedingungen, die auftreten, wenn Sie versuchen, Werte einzufügen. out-of-range Beispielsweise passt ein Dezimalwert mit einer Genauigkeit von 5 Stellen nicht in eine Dezimalspalte mit einer Genauigkeit von 4 Stellen. Eine Ganzzahl oder der gesamte Teil einer Dezimalzahl wird niemals gekürzt. Der Bruchteil einer Dezimalzahl kann jedoch je nach Bedarf auf- oder abgerundet werden. Die Ergebnisse expliziter Umwandlungen von Werten, aus der Tabelle ausgewählt wurden, werden jedoch nicht gerundet.
- Verschiedene Arten von Zeichenketten sind kompatibel. VARCHAR-Spaltenzeichenfolgen, die Einzelbyte-Daten enthalten, und CHAR-Spaltenzeichenfolgen sind vergleichbar und implizit konvertierbar. VARCHAR-Zeichenfolgen mit Multibytedaten können nicht mit CHAR-Spalten verglichen werden. Sie können eine Zeichenfolge auch in einen Datums-, Zeit-, Zeitstempel- oder numerischen Wert konvertieren, wenn es sich bei der Zeichenfolge um einen geeigneten Literalwert handelt. Alle führenden oder nachfolgenden Leerzeichen werden ignoriert. Umgekehrt können Sie auch ein Datum, eine Uhrzeit, einen Zeitstempel oder einen Zahlenwert in eine Zeichenfolge mit fester oder variabler Länge konvertieren.

  > **Note**
  >
  > Wenn Sie eine Zeichenfolge in einen numerischen Typ umwandeln möchten, muss die Zeichenfolge die Zeichendarstellung einer Zahl sein. Sie können die Zeichenketten `'1.0'` beispielsweise in Dezimalwerte `'5.9'` umwandeln, aber Sie können die Zeichenfolge `'ABC'` nicht in einen beliebigen numerischen Typ umwandeln.
- Wenn Sie DEZIMAL-Werte mit Zeichenketten vergleichen, AWS Clean Rooms versucht es, die Zeichenfolge in einen DEZIMALWERT zu konvertieren. Wenn Sie alle anderen numerischen Werte mit Zeichenfolgen vergleichen, werden die numerischen Werte in Zeichenfolgen konvertiert. Um eine Konvertierung in der Gegenrichtung zu erreichen (beispielsweise Zeichenfolgen in

Ganzzahlen oder DECIMAL-Werte in Zahlenfolgen umzuwandeln), müssen Sie eine explizite Funktion wie beispielsweise CAST-Funktion verwenden.

- Wenn Sie einen 64-Bit-Wert vom Typ DECIMAL oder NUMERIC in einen Typ mit einer höheren Genauigkeit umwandeln möchten, müssen Sie eine explizite Funktion verwenden, beispielsweise CAST oder CONVERT.

Arten von impliziter Umwandlung

Es gibt zwei Arten von impliziten Typumwandlungen:

- Implizite Konvertierungen bei Aufgaben, z. B. beim Einstellen von Werten in den Befehlen INSERT oder UPDATE
- Implizite Konvertierungen in Ausdrücken, wie z. B. das Durchführen von Vergleichen in der WHERE-Klausel

In der folgenden Tabelle sind die Datentypen aufgeführt, die implizit in Zuweisungen oder Ausdrücken konvertiert werden können. Sie können diese Konvertierungen auch mit expliziten Umwandlungsfunktionen durchführen.

Von Typ	Zu Typ
BIGINT	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOPPELTE GENAUIGKEIT () FLOAT8
	INTEGER
	ECHT (FLOAT4)
	SMALLINT oder SHORT
	VARCHAR
CHAR	VARCHAR

Von Typ	Zu Typ
DATUM	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT oder LONG
	CHAR
	DOPPELTE GENAUIGKEIT () FLOAT8
	GANZZAHL (INT)
	ECHT (FLOAT4)
	SMALLINT oder SHORT
	VARCHAR
DOPPELTE PRÄZISION () FLOAT8	BIGINT oder LONG
	CHAR
	DECIMAL (NUMERIC)
	GANZZAHL (INT)
	ECHT (FLOAT4)
	SMALLINT oder SHORT
	VARCHAR
GANZZAHL (INT)	BIGINT oder LONG
	BOOLEAN

Von Typ	Zu Typ
	CHAR
	DECIMAL (NUMERIC)
	DOPPELTE GENAUIGKEIT () FLOAT8
	ECHT (FLOAT4)
	SMALLINT oder SHORT
	VARCHAR
ECHT () FLOAT4	BIGINT oder LONG
	CHAR
	DECIMAL (NUMERIC)
	GANZZAHL (INT)
	SMALLINT oder SHORT
	VARCHAR
SMALLINT	BIGINT oder LONG
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOPPELTE GENAUIGKEIT () FLOAT8
	GANZZAHL (INT)
	ECHT (FLOAT4)
	VARCHAR

Von Typ	Zu Typ
TIME	VARCHAR
	TIMETZ

> **Note**
>
> Implizite Konvertierungen zwischen DATE, TIME, TIMESTAMP_LTZ, TIMESTAMP_NTZ oder Zeichenketten verwenden die aktuelle Sitzungszeitzone.
> Der Datentyp VARBYTE kann nicht implizit in einen anderen Datentyp umgewandelt werden. Weitere Informationen finden Sie unter CAST-Funktion.

AWS Clean Rooms Spark-SQL-Befehle

Die folgenden SQL-Befehle werden in AWS Clean Rooms Spark SQL unterstützt:

Themen

- CACHE-TABELLE
- Hinweise
- SELECT

CACHE-TABELLE

Mit dem Befehl CACHE TABLE werden die Daten einer vorhandenen Tabelle zwischengespeichert oder eine neue Tabelle mit Abfrageergebnissen erstellt und zwischengespeichert.

> **Note**
>
> Die zwischengespeicherten Daten bleiben für die gesamte Abfrage bestehen.

Die Syntax, die Argumente und einige Beispiele stammen aus der Apache Spark-SQL-Referenz.

Syntax

Der Befehl CACHE TABLE unterstützt drei Syntaxmuster:

Mit AS (ohne Klammern): Erstellt eine neue Tabelle auf der Grundlage der Abfrageergebnisse und speichert sie im Cache.

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

Mit AS und Klammern: Funktioniert ähnlich wie die erste Syntax, verwendet jedoch Klammern, um die Abfrage explizit zu gruppieren.

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

Ohne AS: Speichert eine bestehende Tabelle im Cache und verwendet die SELECT-Anweisung, um zu filtern, welche Zeilen zwischengespeichert werden sollen.

```
CACHE TABLE cache_table_identifier query;
```

Wobei Folgendes gilt:

- Alle Anweisungen sollten mit einem Semikolon (;) enden
- `query`ist normalerweise eine SELECT-Anweisung
- Klammern um die Abfrage sind bei AS optional
- Das Schlüsselwort AS ist optional

Parameter

cache_table_identifier

Der Name für die zwischengespeicherte Tabelle. Kann einen optionalen Qualifizierer für den Datenbanknamen enthalten.

ALS

Ein Schlüsselwort, das beim Erstellen und Zwischenspeichern einer neuen Tabelle aus Abfrageergebnissen verwendet wird.

query

Eine SELECT-Anweisung oder eine andere Abfrage, die die Daten definiert, die zwischengespeichert werden sollen.

Beispiele

In den folgenden Beispielen bleibt die zwischengespeicherte Tabelle für die gesamte Abfrage bestehen. Nach dem Zwischenspeichern lesen nachfolgende Abfragen, auf die verwiesen *cache_table_identifier* wird, aus der zwischengespeicherten Version, anstatt sie erneut zu berechnen oder aus ihr zu lesen. *sourceTable* Dadurch kann die Abfrageleistung für Daten, auf die häufig zugegriffen wird, verbessert werden.

Erstellen Sie eine gefilterte Tabelle aus Abfrageergebnissen und speichern Sie sie im Cache

Das erste Beispiel zeigt, wie eine neue Tabelle aus Abfrageergebnissen erstellt und zwischengespeichert wird. In diesem Befehl wird das AS Schlüsselwort ohne Klammern um die SELECT Anweisung herum verwendet. Es erstellt eine neue Tabelle mit dem Namen 'cache_table_identifier', die nur die Zeilen von 'sourceTable' enthält, deren Status 'ist. active' Es führt die Abfrage aus, speichert die Ergebnisse in der neuen Tabelle und speichert den Inhalt der neuen Tabelle im Cache. Das ursprüngliche 'sourceTable' bleibt unverändert, und nachfolgende Abfragen müssen auf 'cache_table_identifier' verweisen, um die zwischengespeicherten Daten verwenden zu können.

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

Zwischenspeichern Sie Abfrageergebnisse mit SELECT-Anweisungen in Klammern

Das zweite Beispiel zeigt, wie die Ergebnisse einer Abfrage als neue Tabelle mit einem bestimmten Namen (cache_table_identifier) zwischengespeichert werden, wobei Klammern um die Anweisung herum verwendet werden. SELECT Dieser Befehl erstellt eine neue Tabelle mit dem Namen 'cache_table_identifier', die nur die Zeilen von 'sourceTable' enthält, deren Status 'ist. active' Er führt die Abfrage aus, speichert die Ergebnisse in der neuen Tabelle und speichert den Inhalt der neuen Tabelle im Cache. Das Original 'sourceTable' bleibt unverändert. Nachfolgende Abfragen müssen auf 'cache_table_identifier' verweisen, um die zwischengespeicherten Daten verwenden zu können.

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

Eine vorhandene Tabelle mit Filterbedingungen zwischenspeichern

Das dritte Beispiel zeigt, wie eine vorhandene Tabelle mit einer anderen Syntax zwischengespeichert wird. Bei dieser Syntax, bei der das Schlüsselwort 'AS' und Klammern weggelassen werden, werden in der Regel die angegebenen Zeilen aus einer vorhandenen Tabelle mit dem Namen 'cache_table_identifier' zwischengespeichert, anstatt eine neue Tabelle zu erstellen. Die SELECT Anweisung dient als Filter, um zu bestimmen, welche Zeilen zwischengespeichert werden sollen.

> **Note**
>
> Das genaue Verhalten dieser Syntax ist je nach Datenbanksystem unterschiedlich. Überprüfen Sie immer die richtige Syntax für Ihren spezifischen AWS Dienst.

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

Hinweise

Hinweise für SQL-Analysen enthalten Optimierungsrichtlinien, die als Leitfaden für Strategien zur Abfrageausführung AWS Clean Rooms dienen. So können Sie die Abfrageleistung verbessern und die Rechenkosten senken. Hinweise geben an, wie die Spark-Analyse-Engine ihren Ausführungsplan generieren sollte.

Syntax

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

Hinweise werden mithilfe einer Syntax im Kommentarstil in SQL-Abfragen eingebettet und müssen direkt nach dem SELECT-Schlüsselwort platziert werden.

Unterstützte Hinweistypen

AWS Clean Rooms unterstützt zwei Kategorien von Hinweisen: Join-Hinweise und Partitionierungshinweise.

Themen

- Hinweise zusammenführen
- Hinweise zur Partitionierung

Hinweise zusammenführen

Verbindungshinweise schlagen Verbindungsstrategien für die Abfrageausführung vor. Die Syntax, die Argumente und einige Beispiele stammen aus der Apache Spark-SQL-Referenz mit weiteren Informationen

ÜBERTRAGUNG

Schlägt vor, Broadcast Join zu AWS Clean Rooms verwenden. Die Join-Seite mit dem Hinweis wird unabhängig von autoBroadcastJoin Threshold übertragen. Wenn beide Seiten des Joins die Broadcast-Hinweise haben, wird die Seite mit der kleineren Größe (basierend auf Statistiken) übertragen.

Aliase: BROADCASTJOIN, MAPJOIN

Parameter: Tabellenbezeichner (optional)

Beispiele:

```
-- 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

Schlägt vor, Shuffle Sort Merge Join zu AWS Clean Rooms verwenden.

Aliase: SHUFFLE_MERGE, MERGEJOIN

Parameter: Tabellenbezeichner (optional)

Beispiele:

```
-- 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

Schlägt vor, Shuffle Hash Join AWS Clean Rooms zu verwenden. Wenn beide Seiten die Shuffle-Hash-Hinweise haben, wählt der Abfrageoptimierer die kleinere Seite (basierend auf Statistiken) als Build-Seite.

Parameter: Tabellenbezeichner (optional)

Beispiele:

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

SHUFFLE_REPLICATE_NL

Schlägt vor, Nested Loop Join zu verwenden. AWS Clean Rooms shuffle-and-replicate

Parameter: Tabellenbezeichner (optional)

Beispiele:

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

Hinweise zur Fehlerbehebung in Spark SQL

Die folgende Tabelle zeigt allgemeine Szenarien, in denen Hinweise in SparkSQL nicht angewendet werden. Weitere Informationen finden Sie unter the section called “Überlegungen und Einschränkungen”.

Anwendungsfall	Beispielabfrage
Die Tabellenreferenz wurde nicht gefunden	`SELECT /*+ BROADCAST(fake_table) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Die Tabelle nimmt nicht am Zusammenführungsvorgang teil	`SELECT /*+ BROADCAST(s) */ * FROM students s WHERE s.age > 25;`
Tabellenverweis in einer verschachtelten Unterabfrage	`SELECT /*+ BROADCAST(s) */ * FROM employees e INNER JOIN (SELECT * FROM students s WHERE s.age > 20) sub ON e.eid = sub.sid;`
Spaltenname statt Tabellenverweis	`SELECT /*+ BROADCAST(e.eid) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Hinweis ohne erforderliche Parameter	`SELECT /*+ BROADCAST */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Basistabellenname statt Tabellenalias	`SELECT /*+ BROADCAST(employees) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`

Hinweise zur Partitionierung

Partitionierungshinweise steuern die Datenverteilung zwischen den Executor-Knoten. Wenn mehrere Partitionierungshinweise angegeben sind, werden mehrere Knoten in den logischen Plan eingefügt, aber der Hinweis ganz links wird vom Optimierer ausgewählt.

COALESCE

Reduziert die Anzahl der Partitionen auf die angegebene Anzahl von Partitionen.

Parameter: Numerischer Wert (erforderlich) — muss eine positive Ganzzahl zwischen 1 und 2147483647 sein

Beispiele:

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

VERTEILUNG

Partitioniert Daten mithilfe der angegebenen Partitionierungsausdrücke auf die angegebene Anzahl von Partitionen neu. Verwendet die Round-Robin-Verteilung.

Parameter:

- Numerischer Wert (optional) — Anzahl der Partitionen; muss eine positive Ganzzahl zwischen 1 und 2147483647 sein
- Spaltenbezeichner (optional) — Spalten, nach denen partitioniert werden soll; Diese Spalten müssen im Eingabeschema vorhanden sein.
- Wenn beide angegeben sind, muss der numerische Wert an erster Stelle stehen

Beispiele:

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

-- Repartition by column
```

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

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

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

REPARTITION_BY_RANGE

Partitioniert Daten mithilfe der Bereichspartitionierung der angegebenen Spalten auf die angegebene Anzahl von Partitionen neu.

Parameter:

- Numerischer Wert (optional) — Anzahl der Partitionen; muss eine positive Ganzzahl zwischen 1 und 2147483647 sein
- Spaltenbezeichner (optional) — Spalten, nach denen partitioniert werden soll; Diese Spalten müssen im Eingabeschema vorhanden sein.
- Wenn beide angegeben sind, muss der numerische Wert an erster Stelle stehen

Beispiele:

```
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;
```

NEU AUSBALANCIEREN

Die Ausgabepartitionen der Abfrageergebnisse werden neu verteilt, sodass jede Partition eine angemessene Größe hat (nicht zu klein und nicht zu groß). Dabei handelt es sich um ein Verfahren nach bestem Bemühen: Wenn schiefe Partitionen vorhanden sind, AWS Clean Rooms werden die schiefen Partitionen aufgeteilt, damit sie nicht zu groß werden. Dieser Hinweis ist nützlich, wenn Sie das Ergebnis einer Abfrage in eine Tabelle schreiben müssen, um zu kleine oder zu große Dateien zu vermeiden.

Parameter:

- Numerischer Wert (optional) — Anzahl der Partitionen; muss eine positive Ganzzahl zwischen 1 und 2147483647 sein
- Spaltenbezeichner (optional) — Spalten müssen in der SELECT-Ausgabeliste erscheinen
- Wenn beide angegeben sind, muss der numerische Wert an erster Stelle stehen

Beispiele:

```
-- 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;
```

Kombinieren mehrerer Hinweise

Sie können mehrere Hinweise in einer einzigen Abfrage angeben, indem Sie sie durch Kommas trennen:

```
-- 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;
```

Überlegungen und Einschränkungen

- Hinweise sind Optimierungsvorschläge, keine Befehle. Der Abfrageoptimierer ignoriert möglicherweise Hinweise, die auf Ressourcenbeschränkungen oder Ausführungsbedingungen basieren.
- Hinweise werden sowohl für als auch CreateAnalysisTemplate direkt in SQL-Abfragezeichenfolgen eingebettet. StartProtectedQuery APIs
- Hinweise müssen direkt nach dem SELECT-Schlüsselwort stehen.
- Benannte Parameter werden bei Hinweisen nicht unterstützt und lösen eine Ausnahme aus.
- Spaltennamen in den Hinweisen REPARTITION und REPARTITION_BY_RANGE müssen im Eingabeschema vorhanden sein.
- Die Spaltennamen in den REBALANCE-Hinweisen müssen in der SELECT-Ausgabeliste erscheinen.
- Numerische Parameter müssen positive Ganzzahlen zwischen 1 und 2147483647 sein. Wissenschaftliche Schreibweisen wie 1e1 werden nicht unterstützt
- Hinweise werden in Differential Privacy SQL-Abfragen nicht unterstützt.
- Hinweise für SQL-Abfragen werden in PySpark Jobs nicht unterstützt. Verwenden Sie die Datenrahmen-API, um Anweisungen für Ausführungspläne in einem PySpark Job bereitzustellen. Weitere Informationen finden Sie in den Apache DataFrame Spark-API-Dokumenten.

SELECT

Der SELECT-Befehl gibt Zeilen aus Tabellen und benutzerdefinierten Funktionen zurück.

Die folgenden SELECT-SQL-Befehle, -Klauseln und Mengenoperatoren werden in AWS Clean Rooms Spark SQL unterstützt:

Themen

Die Syntax, die Argumente und einige Beispiele stammen aus der Apache Spark SQL-Referenz.

SELECT list

Die SELECT list Namen der Spalten, Funktionen und Ausdrücke, die die Abfrage zurückgeben soll. Der Liste stellt die Ausgabe der Abfrage dar.

Syntax

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

Parameters

DISTINCT

Eine Option, die duplizierte Zeilen aus dem Ergebnissatz entfernt, basierend auf übereinstimmenden Werten in einer oder mehreren Spalten.

expression

Ein Ausdruck, der aus einer oder mehreren Spalten gebildet wird, die in den Tabellen vorhanden sind, die von der Abfrage referenziert werden. Ein Ausdruck kann SQL-Funktionen enthalten. Beispiel:

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

AS column_alias

Ein temporärer Name für die Spalte, der im endgültigen Ergebnissatz verwendet wird. Das AS-Schlüsselwort ist optional. Beispiel:

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

Wenn Sie keinen Alias für einen Ausdruck angeben, bei dem es sich nicht um einen einfachen Spaltennamen handelt, wendet der Ergebnissatz einen Standardnamen auf diese Spalte an.

> **Note**
>
> Der Alias wird sofort nach seiner Definition in der Zielliste erkannt. Sie können einen Alias nicht in anderen Ausdrücken verwenden, die danach in derselben Zielliste definiert wurden.

WITH-Klausel

Eine WITH-Klausel ist eine optionale Klausel, die der SELECT-Liste in einer Abfrage vorangeht. Die WITH-Klausel definiert einen oder mehrere allgemeine Tabellenausdrücke (CTE). Jeder allgemeine Tabellenausdruck (CTE) definiert eine temporäre Tabelle, die einer Ansichtdefinition ähnelt. Sie können diese temporären Tabellen in der FROM-Klausel referenzieren. Sie werden nur verwendet, während die Abfrage, zu der sie gehören, ausgeführt wird. Jede CTE in der WITH-Klausel gibt einen Tabellennamen, eine optionale Liste von Spaltennamen und einen Abfrageausdruck an, der in eine Tabelle evaluiert wird (eine SELECT-Anweisung).

Unterabfragen mit einer WITH-Klausel sind eine effiziente Art, Tabellen zu definieren, die während der Ausführung einer einzelnen Abfrage verwendet werden können. In allen Fällen können dieselben Ergebnisse erzielt werden, indem im Hauptteil der SELECT-Anweisung Unterabfragen verwendet werden. Unterabfragen mit WITH-Klauseln können jedoch leichter geschrieben und gelesen werden. Wenn möglich, werden Unterabfragen mit WITH-Klauseln, die mehrmals referenziert werden, als gemeinsame Unterausdrücke optimiert. Das bedeutet, dass es möglich sein kann, eine WITH-Unterabfrage einmal zu evaluieren und die Ergebnisse wiederzuverwenden. (Beachten Sie, dass gemeinsame Unterausdrücke nicht auf diejenigen begrenzt sind, die in der WITH-Klausel definiert sind.)

Syntax

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

wobei common_table_expression nicht rekursiv sein kann. Dies ist die nicht-rekursive Form:

```
CTE_table_name AS ( query )
```

Parameters

common_table_expression

Definiert eine temporäre Tabelle, auf die Sie in der FROM-Klausel verweisen können und die nur während der Ausführung der Abfrage verwendet wird, zu der sie gehört.

CTE_table_name

Ein eindeutiger Name für eine temporäre Tabelle, die die Ergebnisse einer Unterabfrage mit WITH-Klausel definiert. Sie können in einer einzelnen WITH-Klausel keine duplizierten Namen verwenden. Jede Unterabfrage muss einen Tabellennamen erhalten, der in der referenziert werden kann FROM-Klausel.

query

Jede SELECT-Abfrage, die unterstützt. AWS Clean Rooms Siehe SELECT.

Nutzungshinweise

Sie können eine WITH-Klausel in der folgenden SQL-Anweisung verwenden:

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

Wenn die FROM-Klausel einer Abfrage, die eine WITH-Klausel enthält, keine der Tabellen referenziert, die von der WITH-Klausel definiert werden, wird die WITH-Klausel ignoriert, und die Abfrage wird wie normal ausgeführt.

Eine Tabelle, die von einer Unterabfrage mit WITH-Klausel definiert ist, kann nur im Bereich der SELECT-Abfrage referenziert werden, die die WITH-Klausel beginnt. Sie können beispielsweise eine solche Tabelle in der FROM-Klausel einer Unterabfrage in der SELECT-Liste, in einer WHERE-Klausel oder in einer HAVING-Klausel referenzieren. Sie können eine WITH-Klausel nicht in einer

Unterabfrage verwenden und ihre Tabelle in der FROM-Klausel der Hauptabfrage oder einer anderen Unterabfrage referenzieren. Dieses Abfragemuster führt zu einer Fehlermeldung der Art `relation table_name doesn't exist` für die Tabelle der WITH-Klausel.

Sie können innerhalb einer Unterabfrage mit WITH-Klausel keine weitere WITH-Klausel angeben.

Sie können keine Vorausreferenzen auf Tabellen erstellen, die durch Unterabfragen mit WITH-Klauseln definiert werden. Die folgende Abfrage gibt beispielsweise aufgrund der Vorausreferenz auf die Tabelle W2 in der Definition der Tabelle W1 einen Fehler zurück:

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

Beispiele

Im folgenden Beispiel wird der einfachste mögliche Fall einer Abfrage gezeigt, die eine WITH-Klausel enthält. Die WITH-Abfrage namens VENUECOPY wählt alle Zeilen aus der Tabelle VENUE aus. Die Hauptabfrage wählt anschließend alle Zeilen aus VENUECOPY aus. Die Tabelle VENUECOPY besteht nur für die Dauer dieser Abfrage.

```
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)
```

Im folgenden Beispiel wird eine WITH-Klausel gezeigt, die zwei Tabellen namens VENUE_SALES und TOP_VENUES erstellt. Die zweite WITH-Abfragetabelle wählt aus der ersten aus. Die WHERE-

Klausel des Hauptabfrageblocks enthält eine Unterabfrage, die die Tabelle TOP_VENUES einschränkt.

```
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)
```

In den folgenden beiden Beispielen werden die Regeln für den Bereich der Tabellenreferenzen auf der Basis von Unterabfragen mit WITH-Klausel gezeigt. Die erste Abfrage wird ausgeführt. Die zweite Abfrage schlägt jedoch mit einem erwarteten Fehler fehl. Die erste Abfrage enthält eine Unterabfrage mit WITH-Klausel innerhalb der SELECT-Liste der Hauptabfrage. Die von der WITH-Klausel definierte Tabelle (HOLIDAYS) wird in der FROM-Klausel der Unterabfrage in der SELECT-Liste referenziert:

```
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)
```

Die zweite Abfrage schlägt fehl, weil sie versucht, die Tabelle HOLIDAYS in der Hauptabfrage und in der Unterabfrage der SELECT-Liste zu referenzieren. Die Referenzen der Hauptabfrage liegen außerhalb des Bereichs.

```
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
```

FROM-Klausel

Die -Klausel in einer Abfrage listet die Tabellenreferenzen (Tabellen, Ansichten und Unterabfragen) auf, aus denen Daten ausgewählt werden. Wenn mehrere Tabellenreferenzen aufgelistet werden, muss ein Join für die Tabellen ausgeführt werden, indem entweder in der FROM-Klausel oder in der WHERE-Klausel die entsprechende Syntax verwendet wird. Wenn keine Join-Kriterien angegeben werden, verarbeitet das System die Abfrage als Kreuz-Join (kartesisches Produkt).

Themen

Syntax

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

wobei table_reference eins der folgenden ist:

```
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

Eine Tabelle, die von einer Unterabfrage in der definiert wird WITH-Klausel.

table_name

Der Name einer Tabelle oder Ansicht.

alias

Der temporäre alternative Name für eine Tabelle oder Ansicht. Für eine Tabelle, die von einer Unterabfrage abgeleitet wird, muss ein Alias bereitgestellt werden. In anderen Tabellenreferenzen sind Aliasnamen optional. Das AS Schlüsselwort ist immer optional. Tabellenaliasnamen stellen eine bequeme Abkürzung für die Identifizierung von Tabellen in anderen Teilen einer Abfrage dar, beispielsweise in der WHERE-Klausel.

Beispiel:

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

Wenn Sie definieren, dass ein Tabellenalias definiert ist, muss der Alias verwendet werden, um in der Abfrage auf diese Tabelle zu verweisen.

Wenn die Abfrage beispielsweise so ist`SELECT "tbl"."col" FROM "tbl" AS "t"`, würde die Abfrage fehlschlagen, weil der Tabellenname jetzt im Wesentlichen überschrieben wird. Eine gültige Abfrage wäre in diesem Fall. `SELECT "t"."col" FROM "tbl" AS "t"`

column_alias

Der temporäre alternative Name für eine Spalte in einer Tabelle oder Ansicht.

subquery

Ein Abfrageausdruck, der zu einer Tabelle evaluiert wird. Die Tabelle ist nur für die Dauer der Abfrage vorhanden und erhält in der Regel einen Namen oder einen Alias. Ein Alias ist jedoch nicht erforderlich. Sie können auch Spaltennamen für Tabellen definieren, die von Unterabfragen abgeleitet werden. Die Vergabe von Spaltenaliasnamen ist wichtig, wenn Sie für die Ergebnisse von Unterabfragen einen Join mit anderen Tabellen ausführen möchten und wenn Sie diese Spalten an anderer Stelle in der Abfrage auswählen oder einschränken möchten.

Eine Unterabfrage kann eine ORDER BY-Klausel enthalten. Diese Klausel hat jedoch keine Auswirkungen, wenn nicht auch eine LIMIT- oder OFFSET-Klausel angegeben ist.

NATURAL

Definiert einen Join, der automatisch alle Paare identisch benannter Spalten in den beiden Tabellen als Joining-Spalten verwendet. Es ist keine explizite Join-Bedingung erforderlich. Wenn die Tabellen CATEGORY und EVENT beispielsweise beide Spalten namens CATID besitzen, ist ein Join ihrer CATID-Spalten ein NATURAL-Join dieser Tabellen.

> **Note**
>
> Wenn ein NATURAL-Join angegeben ist, in den Tabellen, für die ein Join ausgeführt werden soll, jedoch keine identisch benannten Spaltenpaare vorhanden sind, wird für die Abfrage standardmäßig ein Kreuz-Join ausgeführt.

join_type

Geben Sie eine der folgenden Join-Arten an:

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

Kreuz-Joins sind nicht qualifizierte Joins. Sie geben das kartesische Produkt der beiden Tabellen zurück.

Interne und externe Joins sind qualifizierte Joins. Sie sind entweder implizit (in natürlichen Joins), mit der ON- oder USING-Syntax in der FROM-Klausel oder mit einer WHERE-Klauselbedingung qualifiziert.

Ein interner Join gibt nur übereinstimmende Zeilen zurück, basierend auf der Join-Bedingung oder der Liste der Joining-Spalten. Ein externer Join gibt alle Zeilen zurück, die der entsprechende interne Join zurückgeben würde, und zusätzlich nicht übereinstimmende Zeilen aus der Tabelle „links“, aus der Tabelle „rechts“ oder aus beiden Tabellen. Die linke Tabelle wird zuerst aufgelistet. Die rechte Tabelle wird als zweite Tabelle aufgelistet. Die nicht übereinstimmenden Zeilen enthalten NULL-Werte, um die Lücken in den Ausgabespalten zu füllen.

ON join_condition

Eine Join-Spezifikation, in der die Joining-Spalten als eine Bedingung angegeben werden, die dem Schlüsselwort ON folgt. Beispiel:

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

USING (join_column [, ...])

Eine Join-Spezifikation, in der die Joining-Spalten in Klammern angegeben werden. Wenn mehrere Joining-Spalten angegeben werden, werden sie durch Komma abgetrennt. Das Schlüsselwort USING muss der Liste vorangestellt werden. Zum Beispiel:

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

Nutzungshinweise

Joining-Spalten müssen vergleichbare Datentypen haben.

Ein NATURAL- oder -USING-Join enthält jeweils nur eine Spalte jedes Joining-Spaltenpaars im Zwischenergebnissatz.

Ein Join mit der ON-Syntax enthält beide Joining-Spalten im Zwischenergebnissatz.

Weitere Informationen finden Sie auch unter WITH-Klausel.

JOIN-Klausel

Eine SQL JOIN-Klausel wird verwendet, um die Daten aus zwei oder mehr Tabellen basierend auf gemeinsamen Feldern zu kombinieren. Die Ergebnisse können sich je nach festgelegter Join-Methode ändern oder nicht. Externe Joins nach links und rechts behalten die Werte aus einer der Tabellen, für die ein Join ausgeführt wurde, wenn in der anderen Tabelle keine Übereinstimmung gefunden wurde.

Die Kombination aus dem JOIN-Typ und der Join-Bedingung bestimmt, welche Zeilen in der endgültigen Ergebnismenge enthalten sind. Die SELECT- und WHERE-Klauseln steuern dann, welche Spalten zurückgegeben werden und wie die Zeilen gefiltert werden. Das Verständnis der verschiedenen JOIN-Typen und deren effektive Verwendung ist eine wichtige Fähigkeit in SQL, da Sie damit Daten aus mehreren Tabellen auf flexible und leistungsstarke Weise kombinieren können.

Syntax

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

Parameters

WÄHLEN SIE Spalte1, Spalte2,..., SpalteN

Die Spalten, die Sie in die Ergebnismenge aufnehmen möchten. Sie können Spalten aus einer oder beiden der am JOIN beteiligten Tabellen auswählen.

AUS Tabelle1

Die erste (linke) Tabelle in der JOIN-Operation.

[VERKNÜPFEN | INNERE VERKNÜPFUNG | LINKE [ÄUSSERE] VERKNÜPFUNG | RECHTE [ÄUSSERE] VERKNÜPFUNG | VOLLSTÄNDIGE [ÄUSSERE] VERKNÜPFUNG] Tabelle2:

Der Typ des auszuführenden JOINS. JOIN oder INNER JOIN gibt nur die Zeilen mit übereinstimmenden Werten in beiden Tabellen zurück.

LEFT [OUTER] JOIN gibt alle Zeilen aus der linken Tabelle mit übereinstimmenden Zeilen aus der rechten Tabelle zurück.

RIGHT [OUTER] JOIN gibt alle Zeilen aus der rechten Tabelle mit den entsprechenden Zeilen aus der linken Tabelle zurück.

FULL [OUTER] JOIN gibt alle Zeilen aus beiden Tabellen zurück, unabhängig davon, ob eine Übereinstimmung vorliegt oder nicht.

CROSS JOIN erzeugt ein kartesisches Produkt der Zeilen aus den beiden Tabellen.

ON Tabelle1.Spalte = Tabelle2.Spalte

Die Join-Bedingung, die angibt, wie die Zeilen in den beiden Tabellen abgeglichen werden. Die Join-Bedingung kann auf einer oder mehreren Spalten basieren.

WHERE-Bedingung:

Eine optionale Klausel, mit der die Ergebnismenge anhand einer bestimmten Bedingung weiter gefiltert werden kann.

Beispiel

Das folgende Beispiel ist ein Join zwischen zwei Tabellen mit der USING-Klausel. In diesem Fall werden die Spalten listid und eventid als Join-Spalten verwendet. Die Ergebnisse sind auf 5 Zeilen begrenzt.

```
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
```

JOIN-Typen

INNER

Dies ist der Standard-Join-Typ. Gibt die Zeilen zurück, deren Werte in beiden Tabellenverweisen übereinstimmen.

Der INNER JOIN ist der in SQL am häufigsten verwendete Join-Typ. Es ist eine leistungsstarke Methode, um Daten aus mehreren Tabellen auf der Grundlage einer gemeinsamen Spalte oder einer Gruppe von Spalten zu kombinieren.

Syntax:

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

Die folgende Abfrage gibt alle Zeilen zurück, in denen ein übereinstimmender customer_id-Wert zwischen den Tabellen „customers“ und „orders“ vorhanden ist. Das Resultset wird die Spalten customer_id, name, order_id und order_date enthalten.

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

Die folgende Abfrage ist ein innerer Join (ohne das Schlüsselwort JOIN) zwischen den Tabellen LISTING und SALES, wobei die LISTID aus der Tabelle LISTING zwischen 1 und 5 liegt. Diese Abfrage gleicht LISTID-Spaltenwerte in der Tabelle LISTING (linke Tabelle) und der Tabelle SALES (rechte Tabelle) ab. Die Ergebnisse zeigen, dass LISTID 1, 4 und 5 den Kriterien entsprechen.

```
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
```

Bei dem folgenden Beispiel handelt es sich um einen inneren Join mit der ON-Klausel. In diesem Fall werden NULL-Zeilen nicht zurückgegeben.

```
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
```

Die folgende Abfrage ist ein interner Join zweiter Unterabfragen in der FROM-Klausel. Die Abfrage ermittelt die Zahl der verkauften und nicht verkauften Tickets für verschiedene Veranstaltungskategorien (Konzerte und Shows). Die Unterabfragen mit FROM-Klausel sind Tabellen-Unterabfragen und können mehrere Spalten und Zeilen zurückgeben.

```
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
```

LINKS [AUSSEN]

Gibt alle Werte aus der linken Tabellenreferenz und die übereinstimmenden Werte aus der rechten Tabellenreferenz zurück oder hängt NULL an, wenn es keine Übereinstimmung gibt. Es wird auch als Left Outer Join bezeichnet.

Es gibt alle Zeilen aus der linken (ersten) Tabelle und die passenden Zeilen aus der rechten (zweiten) Tabelle zurück. Wenn es in der rechten Tabelle keine Übereinstimmung gibt, enthält die Ergebnismenge NULL-Werte für die Spalten aus der rechten Tabelle. Das Schlüsselwort OUTER kann weggelassen werden, und der Join kann einfach als LEFT JOIN geschrieben werden. Das Gegenteil von LEFT OUTER JOIN ist RIGHT OUTER JOIN, bei dem alle Zeilen aus der rechten Tabelle und die passenden Zeilen aus der linken Tabelle zurückgegeben werden.

Syntax:

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

Die folgende Abfrage gibt alle Zeilen aus der Kundentabelle zusammen mit den entsprechenden Zeilen aus der Bestelltabelle zurück. Wenn ein Kunde keine Bestellungen hat, enthält das Resultset dennoch die Informationen dieses Kunden mit NULL-Werten für die Spalten order_id und 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;
```

Bei der folgenden Abfrage handelt es sich um einen linken, externen Join. Externe Joins nach links und rechts behalten die Werte aus einer der Tabellen, für die ein Join ausgeführt wurde, wenn in der anderen Tabelle keine Übereinstimmung gefunden wurde. Die Tabellen links und rechts werden in der Syntax als erste und zweite Tabelle aufgelistet. Es werden NULL-Werte verwendet, um

die „Lücken“ im Ergebnissatz zu füllen. Diese Abfrage gleicht LISTID-Spaltenwerte in der Tabelle LISTING (linke Tabelle) und der Tabelle SALES (rechte Tabelle) ab. Die Ergebnisse zeigen, dass LISTIDs 2 und 3 zu keinen Verkäufen geführt haben.

```
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
```

RECHTS [AUSSEN]

Gibt alle Werte aus der rechten Tabellenreferenz und die übereinstimmenden Werte aus der linken Tabellenreferenz zurück oder hängt NULL an, wenn es keine Übereinstimmung gibt. Es wird auch als rechter äußerer Join bezeichnet.

Es gibt alle Zeilen aus der rechten (zweiten) Tabelle und die passenden Zeilen aus der linken (ersten) Tabelle zurück. Wenn es in der linken Tabelle keine Übereinstimmung gibt, enthält die Ergebnismenge NULL-Werte für die Spalten aus der linken Tabelle. Das Schlüsselwort OUTER kann weggelassen werden, und der Join kann einfach als RIGHT JOIN geschrieben werden. Das Gegenteil von RIGHT OUTER JOIN ist LEFT OUTER JOIN, bei dem alle Zeilen aus der linken Tabelle und die passenden Zeilen aus der rechten Tabelle zurückgegeben werden.

Syntax:

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

Die folgende Abfrage gibt alle Zeilen aus der Kundentabelle zusammen mit den entsprechenden Zeilen aus der Bestelltabelle zurück. Wenn ein Kunde keine Bestellungen hat, enthält das Resultset dennoch die Informationen dieses Kunden mit NULL-Werten für die Spalten order_id und 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;
```

Bei der folgenden Abfrage handelt es sich um einen rechten, externen Join. Diese Abfrage gleicht LISTID-Spaltenwerte in der Tabelle LISTING (linke Tabelle) und der Tabelle SALES (rechte Tabelle) ab. Die Ergebnisse zeigen, dass LISTIDs 1, 4 und 5 den Kriterien entsprechen.

```
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
```

VOLL [ÄUSSERLICH]

Gibt alle Werte aus beiden Beziehungen zurück und fügt NULL-Werte auf der Seite an, für die es keine Übereinstimmung gibt. Es wird auch als vollständiger äußerer Join bezeichnet.

Es gibt alle Zeilen sowohl aus der linken als auch aus der rechten Tabelle zurück, unabhängig davon, ob eine Übereinstimmung vorliegt oder nicht. Wenn es keine Übereinstimmung gibt, enthält die Ergebnismenge NULL-Werte für die Spalten aus der Tabelle, die keine passende Zeile hat. Das Schlüsselwort OUTER kann weggelassen werden, und der Join kann einfach als FULL JOIN geschrieben werden. Der FULL OUTER JOIN wird seltener verwendet als der LEFT OUTER JOIN oder RIGHT OUTER JOIN, kann aber in bestimmten Szenarien nützlich sein, in denen Sie alle Daten aus beiden Tabellen sehen müssen, auch wenn es keine Treffer gibt.

Syntax:

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

Die folgende Abfrage gibt alle Zeilen aus den Tabellen „Kunden“ und „Bestellungen“ zurück. Wenn ein Kunde keine Bestellungen hat, enthält das Resultset dennoch die Informationen dieses Kunden mit NULL-Werten für die Spalten order_id und order_date. Wenn einer Bestellung kein Kunde zugeordnet ist, enthält das Resultset diese Bestellung mit NULL-Werten für die Spalten customer_id und 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;
```

Bei der folgenden Abfrage handelt es sich um einen vollständigen Join. Vollständige Joins behalten die Werte aus einer der Tabellen bei, für die ein Join ausgeführt wurde, wenn in der anderen Tabelle keine Übereinstimmung gefunden wurde. Die Tabellen links und rechts werden in der Syntax als erste und zweite Tabelle aufgelistet. Es werden NULL-Werte verwendet, um die „Lücken“ im Ergebnissatz zu füllen. Diese Abfrage gleicht LISTID-Spaltenwerte in der Tabelle LISTING (linke Tabelle) und der Tabelle SALES (rechte Tabelle) ab. Die Ergebnisse zeigen, dass LISTIDs 2 und 3 zu keinen Verkäufen geführt haben.

```
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
```

Bei der folgenden Abfrage handelt es sich um einen vollständigen Join. Diese Abfrage gleicht LISTID-Spaltenwerte in der Tabelle LISTING (linke Tabelle) und der Tabelle SALES (rechte Tabelle) ab. Nur Zeilen, die zu keinen Verkäufen führen (LISTIDs 2 und 3), sind in den Ergebnissen enthalten.

```
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
```

[LINKS] HALB

Gibt Werte von der linken Seite der Tabellenreferenz zurück, die mit der rechten Seite übereinstimmen. Es wird auch als linke Semi-Verknüpfung bezeichnet.

Es werden nur die Zeilen aus der linken (ersten) Tabelle zurückgegeben, für die eine entsprechende Zeile in der rechten (zweiten) Tabelle vorhanden ist. Es werden keine Spalten aus der rechten Tabelle zurückgegeben, sondern nur die Spalten aus der linken Tabelle. Der LEFT SEMI JOIN ist nützlich, wenn Sie die Zeilen in einer Tabelle suchen möchten, die eine Übereinstimmung in einer anderen Tabelle haben, ohne Daten aus der zweiten Tabelle zurückgeben zu müssen. Der LEFT SEMI JOIN ist eine effizientere Alternative zur Verwendung einer Unterabfrage mit einer IN- oder EXISTS-Klausel.

Syntax:

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

Die folgende Abfrage gibt nur die Spalten customer_id und name aus der Kundentabelle für die Kunden zurück, die mindestens eine Bestellung in der Bestelltabelle haben. Das Resultset wird keine Spalten aus der Bestelltabelle enthalten.

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

CROSS JOIN

Gibt das kartesische Produkt zweier Beziehungen zurück. Das bedeutet, dass die Ergebnismenge alle möglichen Kombinationen von Zeilen aus den beiden Tabellen enthält, ohne dass eine Bedingung oder ein Filter angewendet wird.

Der CROSS JOIN ist nützlich, wenn Sie alle möglichen Kombinationen von Daten aus zwei Tabellen generieren müssen, z. B. wenn Sie einen Bericht erstellen möchten, der alle möglichen Kombinationen von Kunden- und Produktinformationen anzeigt. Der CROSS JOIN unterscheidet sich von anderen Join-Typen (INNER JOIN, LEFT JOIN usw.), da er in der ON-Klausel keine Join-Bedingung enthält. Die Join-Bedingung ist für einen CROSS JOIN nicht erforderlich.

Syntax:

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

Die folgende Abfrage gibt ein Resultset zurück, das alle möglichen Kombinationen von customer_id, customer_name, product_id und product_name aus den Tabellen Customers und Products enthält. Wenn die Kundentabelle 10 Zeilen und die Produkttabelle 20 Zeilen hat, enthält die Ergebnismenge von CROSS JOIN 10 x 20 = 200 Zeilen.

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

Bei der folgenden Abfrage handelt es sich um einen Cross Join oder kartesischen Join der LISTING- und der SALES-Tabelle mit einem Prädikat zur Begrenzung der Ergebnisse. Diese Abfrage entspricht den LISTID-Spaltenwerten in der SALES-Tabelle und der LISTING-Tabelle für LISTIDs 1, 2, 3, 4 und 5 in beiden Tabellen. Die Ergebnisse zeigen, dass 20 Zeilen den Kriterien entsprechen.

```
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

Gibt die Werte aus der linken Tabellenreferenz zurück, die nicht mit der rechten Tabellenreferenz übereinstimmen. Es wird auch als Left Anti Join bezeichnet.

Der ANTI JOIN ist eine nützliche Operation, wenn Sie die Zeilen in einer Tabelle suchen möchten, für die es in einer anderen Tabelle keine Übereinstimmung gibt.

Syntax:

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

Die folgende Abfrage gibt alle Kunden zurück, die keine Bestellungen aufgegeben haben.

```
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

Gibt an, dass die Zeilen aus den beiden Beziehungen implizit auf Gleichheit für alle Spalten mit übereinstimmenden Namen abgeglichen werden.

Es ordnet automatisch Spalten mit demselben Namen und Datentyp zwischen den beiden Tabellen zu. Sie müssen die Join-Bedingung nicht explizit in der ON-Klausel angeben. Sie kombiniert alle übereinstimmenden Spalten zwischen den beiden Tabellen in der Ergebnismenge.

NATURAL JOIN ist eine praktische Abkürzung, wenn die Tabellen, die Sie verknüpfen, Spalten mit denselben Namen und Datentypen haben. Es wird jedoch generell empfohlen, das explizitere INNER JOIN... zu verwenden ON-Syntax, um die Join-Bedingungen expliziter und verständlicher zu machen.

Syntax:

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

Das folgende Beispiel ist eine natürliche Verknüpfung zwischen zwei Tabellen mit den folgenden Spalten: `employees` `departments`

- `employees`Tabelle: `employee_id``first_name`,`last_name`, `department_id`
- `departments`tabelle:`department_id`, `department_name`

Die folgende Abfrage gibt eine Ergebnismenge zurück, die den Vornamen, den Nachnamen und den Abteilungsnamen für alle übereinstimmenden Zeilen zwischen den beiden Tabellen enthält, basierend auf der `department_id` Spalte.

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

Das folgende Beispiel ist ein NATURAL-Join zwischen zwei Tabellen. In diesem Fall haben die Spalten listid, sellerid, eventid und dateid identische Namen und Datentypen in beiden Tabellen und werden daher als Join-Spalten verwendet. Die Ergebnisse sind auf 5 Zeilen begrenzt.

```
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
```

WHERE-Klausel

Die WHERE-Klausel enthält Bedingungen, die entweder einen Join für Tabellen ausführen oder Prädikate auf Spalten in Tabellen anwenden. Für Tabellen können interne Joins ausgeführt werden, indem entweder in der WHERE-Klausel oder in der FROM-Klausel die entsprechende Syntax verwendet wird. Die Kriterien für externe Joins müssen in der FROM-Klausel angegeben werden.

Syntax

```
[ WHERE condition ]
```

Bedingung

Jede Suchbedingung mit einem Booleschen Ergebnis, wie eine Join-Bedingung oder ein Prädikat für eine Tabellenspalte. In den folgenden Beispielen werden gültige Join-Bedingungen gezeigt:

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

In den folgenden Beispielen werden gültige Bedingungen für Spalten in Tabellen gezeigt:

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

Bedingungen können einfach oder komplex sein. Im Fall komplexer Bedingungen können Sie Klammern verwenden, um logische Einheiten zu isolieren. Im folgenden Beispiel wird die Join-Bedingung durch Klammern umschlossen.

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

Nutzungshinweise

Sie können in der WHERE-Klausel Aliase verwenden, um Auswahllistenausdrücke zu referenzieren.

Sie können die Ergebnisse aggregierter Funktionen in der WHERE-Klausel nicht einschränken. Verwenden Sie für diesen Zweck die HAVING-Klausel.

Spalten, die in der WHERE-Klausel eingeschränkt sind, müssen von Tabellenreferenzen in der FROM-Klausel abgeleitet werden.

Beispiel

Die folgende Abfrage verwendet eine Kombination aus verschiedenen WHERE-Klauseleinschränkungen, einschließlich einer Join-Bedingung für die Tabellen SALES und EVENT, eines Prädikats für die EVENTNAME-Spalte und zweier Prädikate für die STARTTIME-Spalte.

```
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)
```

VALUES-Klausel

Die VALUES-Klausel wird verwendet, um eine Reihe von Zeilenwerten direkt in der Abfrage bereitzustellen, ohne dass auf eine Tabelle verwiesen werden muss.

Die VALUES-Klausel kann in den folgenden Szenarien verwendet werden:

- Sie können die VALUES-Klausel in einer INSERT INTO-Anweisung verwenden, um die Werte für die neuen Zeilen anzugeben, die in eine Tabelle eingefügt werden.
- Sie können die VALUES-Klausel alleine verwenden, um eine temporäre Ergebnismenge oder eine Inline-Tabelle zu erstellen, ohne auf eine Tabelle verweisen zu müssen.
- Sie können die VALUES-Klausel mit anderen SQL-Klauseln wie WHERE, ORDER BY oder LIMIT kombinieren, um die Zeilen in der Ergebnismenge zu filtern, zu sortieren oder einzuschränken.

Diese Klausel ist besonders nützlich, wenn Sie einen kleinen Datensatz direkt in Ihre SQL-Anweisung einfügen, abfragen oder bearbeiten müssen, ohne eine permanente Tabelle erstellen oder darauf verweisen zu müssen. Sie ermöglicht es Ihnen, die Spaltennamen und die entsprechenden Werte für jede Zeile zu definieren, sodass Sie die Flexibilität haben, temporäre Ergebnismengen zu erstellen oder Daten im laufenden Betrieb einzufügen, ohne den Aufwand, eine separate Tabelle verwalten zu müssen.

Syntax

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

Parameter

expression

Ein Ausdruck, der eine Kombination aus einem oder mehreren Werten, Operatoren und SQL-Funktionen angibt, die zu einem Wert führt.

table_alias

Ein Alias, der einen temporären Namen mit einer optionalen Spaltennamenliste angibt.

Beispiel

Im folgenden Beispiel wird eine Inline-Tabelle erstellt, eine temporäre tabellenähnliche Ergebnismenge mit zwei Spalten, und`col1`. `col2` Die einzelne Zeile in der Ergebnismenge enthält jeweils die Werte `"one"` und`1`. Der `SELECT * FROM` Teil der Abfrage ruft einfach alle Spalten und Zeilen aus dieser temporären Ergebnismenge ab. Die Spaltennamen (`col1`und`col2`) werden automatisch vom Datenbanksystem generiert, da die VALUES-Klausel die Spaltennamen nicht explizit spezifiziert.

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

Wenn Sie benutzerdefinierte Spaltennamen definieren möchten, können Sie dies tun, indem Sie nach der VALUES-Klausel eine AS-Klausel verwenden, etwa so:

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

Dadurch würde eine temporäre Ergebnismenge mit den Spaltennamen `name` und `id` anstelle der Standardwerte `col1` und erstelltcol2.

GROUP BY-Klausel

Die GROUP BY-Klausel identifiziert die Gruppierungsspalten für die Abfrage. Gruppierungsspalten müssen deklariert werden, wenn die Abfrage aggregierte Werte mit Standardfunktionen wie SUM, AVG und COUNT berechnet. Wenn der SELECT-Ausdruck eine Aggregatfunktion enthält, muss jede Spalte im SELECT-Ausdruck, die sich nicht in einer Aggregatfunktion befindet, in der GROUP BY-Klausel enthalten sein.

Weitere Informationen finden Sie unter AWS Clean Rooms Spark SQL-Funktionen.

Syntax

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

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

Parameter

expr

Der Liste der Spalten oder Ausdrücke muss der Liste der nicht aggregierten Ausdrücke in der Auswahlliste der Abfrage entsprechen. Betrachten Sie beispielsweise die folgende einfache Abfrage.

```
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 dieser Abfrage besteht die Auswahlliste aus zwei aggregierten Ausdrücken. Der erste verwendet die SUM-Funktion und der zweite verwendet die COUNT-Funktion. Die übrigen beiden Spalten, LISTID und EVENTID, müssen als Gruppierungsspalten deklariert werden.

Ausdrücke in der -Klausel können ebenfalls die Auswahlliste durch Verwendung von Ordinalzahlen referenzieren. Das vorherige Beispiel könnte beispielsweise wie folgt abgekürzt werden.

```
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

Sie können die Aggregationserweiterung ROLLUP verwenden, um die Arbeit mehrerer GROUP BY-Operationen in einer einzigen Anweisung auszuführen. Weitere Informationen zu Aggregationserweiterungen und verwandten Funktionen finden Sie unter Aggregationserweiterungen.

Aggregationserweiterungen

AWS Clean Roomsunterstützt Aggregationserweiterungen, um die Arbeit mehrerer GROUP BY-Operationen in einer einzigen Anweisung zu erledigen.

GROUPING SETS

Berechnet einen oder mehrere Gruppierungssätze in einer einzigen Anweisung. Ein Gruppierungssatz ist die Menge einer einzelnen GROUP BY-Klausel, eine Menge von 0 oder mehr Spalten, nach denen Sie die Ergebnismenge einer Abfrage gruppieren können. GROUP BY GROUPING SETS entspricht der Ausführung einer UNION ALL-Abfrage für eine Ergebnismenge, die nach verschiedenen Spalten gruppiert ist. Beispielsweise entspricht GROUP BY GROUPING SETS((a), (b)) GROUP BY a UNION ALL GROUP BY b.

Das folgende Beispiel gibt die Kosten der Produkte der Bestelltabelle zurück, gruppiert sowohl nach den Produktkategorien als auch nach der Art der verkauften Produkte.

```
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

Geht von einer Hierarchie aus, bei der vorangehende Spalten als übergeordnete Spalten der nachfolgenden Spalten betrachtet werden. ROLLUP gruppiert Daten nach den bereitgestellten Spalten und gibt zusätzlich zu den gruppierten Zeilen weitere Zwischensummenzeilen zurück, die die Summen auf allen Ebenen der Gruppierungsspalten darstellen. Beispielsweise können Sie GROUP BY ROLLUP((a), (b)) verwenden, um eine Ergebnismenge zurückzugeben, die zuerst nach a und dann nach b gruppiert ist, wobei angenommen wird, dass b ein Unterabschnitt von a ist. ROLLUP gibt auch eine Zeile mit der gesamten Ergebnismenge ohne Gruppierungsspalten zurück.

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

Im folgenden Beispiel werden die Kosten der Produkte der Bestelltabelle zurückgegeben, zuerst nach Kategorie und dann nach Produkt gruppiert, wobei „product“ (Produkt) eine Unterteilung von „category“ (Kategorie) darstellt.

```
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

Gruppiert Daten nach den bereitgestellten Spalten und gibt zusätzlich zu den gruppierten Zeilen weitere Zwischensummenzeilen zurück, die die Summen auf allen Ebenen der Gruppierungsspalten darstellen. CUBE gibt dieselben Zeilen wie ROLLUP zurück und fügt zusätzliche Zwischensummenzeilen für jede Kombination von Gruppierungsspalten hinzu, die nicht von ROLLUP abgedeckt wird. Beispielsweise können Sie GROUP BY CUBE ((a), (b)) verwenden, um eine Ergebnismenge zurückzugeben, die zuerst nach a und dann nach b – unter der Annahme, dass b ein Unterabschnitt von a ist – und dann nur nach b gruppiert ist. CUBE gibt auch eine Zeile mit der gesamten Ergebnismenge ohne Gruppierungsspalten zurück.

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

Im folgenden Beispiel werden die Kosten der Produkte der Bestelltabelle zurückgegeben, zuerst nach Kategorie und dann nach Produkt gruppiert, wobei „product“ (Produkt) eine Unterteilung von „category“ (Kategorie) darstellt. Im Gegensatz zum vorherigen Beispiel für ROLLUP gibt die Anweisung Ergebnisse für jede Kombination von Gruppierungsspalten zurück.

```
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)
```

HAVING-Klausel

Die HAVING-Klausel wendet eine Bedingung auf den gruppierten Zwischenergebnissatz an, den eine Abfrage zurückgibt.

Syntax

```
[ HAVING condition ]
```

Sie können beispielsweise die Ergebnisse einer SUM-Funktion einschränken:

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

Die HAVING-Bedingung wird angewendet, nachdem alle WHERE-Klauselbedingungen angewendet wurden und die GROUP BY-Operationen abgeschlossen sind.

Die Bedingung selbst hat das gleiche Format wie eine WHERE-Klauselbedingung.

Nutzungshinweise

- Bei jeder, in einer -Klauselbedingung referenzierten Spalte muss es sich entweder um eine Gruppierungsspalte handeln oder um eine Spalte, die sich auf das Ergebnis einer aggregierten Funktion bezieht.
- In einer HAVING-Klausel können Sie Folgendes nicht angeben:
  - Eine Ordinalzahl, die ein Auswahllistenelement referenziert. Nur die Klauseln GROUP BY und ORDER BY akzeptieren Ordinalzahlen.

Beispiele

Die folgende Abfrage berechnet den Ticket-Gesamtverkauf für alle Veranstaltungen nach Namen. Anschließend werden Veranstaltungen entfernt, deren Gesamtverkauf weniger als 800.000 USD betrug. Die HAVING-Bedingung wird auf die Ergebnisse der Aggregierungsfunktion in der Auswahlliste angewendet: `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)
```

Die folgende Abfrage berechnet einen ähnlichen Ergebnissatz. In diesem Fall wird die HAVING-Bedingung jedoch auf ein Aggregat angewendet, das nicht in der Auswahlliste angegeben ist: `sum(qtysold)`. Veranstaltungen, für weniger als 2.000 Tickets verkauft wurden, werden aus dem Endergebnis entfernt.

```
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)
```

Satzoperatoren

Die Mengenoperatoren werden verwendet, um die Ergebnisse zweier separater Abfrageausdrücke zu vergleichen und zusammenzuführen.

AWS Clean RoomsSpark SQL unterstützt die folgenden Mengenoperatoren, die in der folgenden Tabelle aufgeführt sind.

Set-Operator
INTERSECT

Set-Operator
ALLES ÜBERSCHNEIDEN
EXCEPT
AUSSER ALLEN
UNION
UNION ALL

Wenn Sie beispielsweise wissen möchten, welche Benutzer einer Website sowohl Käufer als auch Verkäufer sind, die Namen jedoch in getrennten Spalten oder Tabellen gespeichert sind, können Sie die Überschneidung zwischen diesen beiden Arten von Benutzern finden. Wenn Sie wissen möchten, welche Benutzer einer Website Käufer, jedoch nicht Verkäufer sind, können Sie den Operator EXCEPT verwenden, um den Unterschied zwischen diesen beiden Listen von Benutzern zu finden. Wenn Sie eine Liste aller Benutzer unabhängig von der Rolle erstellen möchten, können Sie den Operator UNION verwenden.

> **Note**
>
> Die Klauseln ORDER BY, LIMIT, SELECT TOP und OFFSET können nicht in den Abfrageausdrücken verwendet werden, die durch die Mengenoperatoren UNION, UNION ALL, INTERSECT und EXCEPT zusammengeführt werden.

Themen

Syntax

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

Parameters

Unterabfrage1, Unterabfrage2

Ein Abfrageausdruck, der in Form seiner Auswahlliste einem zweiten Abfrageausdruck entspricht, der auf den Operator UNION, UNION ALL, INTERSECT, INTERSECT ALL, EXCEPT oder EXCEPT ALL folgt. Die beiden Ausdrücke müssen die gleiche Zahl von Ausgabespalten mit kompatiblen Datentypen enthalten. Andernfalls können die beiden Ergebnissätze nicht verglichen und zusammengeführt werden. Mengenoperationen ermöglichen keine implizite Konvertierung zwischen verschiedenen Kategorien von Datentypen. Weitere Informationen finden Sie unter Kompatibilität von Typen und Umwandlung zwischen Typen.

Sie können Abfragen erstellen, die eine unbegrenzte Anzahl von Abfrageausdrücken enthalten, und sie mithilfe der Operatoren UNION, INTERSECT und EXCEPT in beliebigen Kombinationen verbinden. Beispielsweise ist die folgende Abfragestruktur gültig, wenn die Tabellen T1, T2 und T3 kompatible Sätze von Spalten enthalten:

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

VEREINIGUNG [ALLE | UNTERSCHIEDLICH]

Satzoperation, die Zeilen aus zwei Abfrageausdrücken zurückgibt, unabhängig davon, ob die Zeilen von einem oder von beiden Ausdrücken abgeleitet werden.

SICH ÜBERSCHNEIDEN [ALLE | UNTERSCHIEDLICH]

Satzoperation, die Zeilen zurückgibt, die von zwei Abfrageausdrücken abgeleitet werden. Zeilen, die nicht von beiden Ausdrücken zurückgegeben werden, werden verworfen.

AUSSER [ALLE | UNTERSCHIEDLICH]

Satzoperation, die Zeilen zurückgibt, die von einem von zwei Abfrageausdrücken abgeleitet werden. Um sich für das Ergebnis zu qualifizieren, dürfen Zeilen zwar in der ersten Ergebnistabelle, nicht jedoch in der zweiten vorhanden sein.

EXCEPT ALL entfernt keine Duplikate aus den Ergebniszeilen.

MINUS und EXCEPT sind exakte Synonyme.

Reihenfolge der Evaluierung für Satzoperatoren

Die Satzoperatoren UNION und EXCEPT sind links-assoziativ. Wenn keine Klammern angegeben werden, um die Reihenfolge zu beeinflussen, wird eine Kombination dieser Satzoperatoren von links nach rechts ausgewertet. Beispielsweise wird in der folgenden Abfrage der Operator UNION von T1 und T2 zuerst ausgewertet. Anschließend wird die Operation EXCEPT für das UNION-Ergebnis ausgeführt:

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

Der Operator INTERSECT hat Vorrang vor den Operatoren UNION und EXCEPT, wenn in derselben Abfrage eine Kombination von Operatoren verwendet wird. Beispielsweise wird in der folgenden Abfrage die Schnittmenge von T2 und T3 ausgewertet und anschließend mit T1 vereinigt:

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

Durch die Hinzufügung von Klammern können Sie eine andere Reihenfolge für die Auswertung erzwingen. Im folgenden Fall wird für das Ergebnis von UNION für T1 und T2 eine Überschneidung mit T3 ausgewertet. Die Abfrage führt wahrscheinlich zu einem anderen Ergebnis.

```
(select * from t1
```

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

Nutzungshinweise

- Die Spaltennamen, die im Ergebnis einer Satzoperationsabfrage zurückgegeben werden, sind die Spaltennamen (Spaltenaliase) aus den Tabellen im ersten Abfrageausdruck. Da diese Spaltennamen potenziell irreführend sein können, da die Werte in der Spalte aus Tabellen auf beiden Seiten des Satzoperators abgeleitet werden, sollten Sie möglicherweise sinnvolle Aliase für den Ergebnissatz bereitstellen.
- Wenn Abfragen mit Satzoperatoren Dezimalergebnisse zurückgeben, geben die entsprechenden Ergebnisspalten Werte mit derselben Genauigkeit und Skalierung zurück. In der folgenden Abfrage, in der T1.REVENUE eine DECIMAL(10,2)-Spalte ist und T2.REVENUE eine DECIMAL(8,4)-Spalte ist, ist das Dezimalergebnis DECIMAL(12,4):

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

  Die Skalierung ist 4, da dies die maximale Skalierung der beiden Spalten ist. Die Genauigkeit ist 12, da T1.REVENUE 8 Stellen links vom Dezimalkomma erfordert (12 – 4 = 8). Dieser Vorgang stellt sicher, dass alle Werte aus beiden Seiten der UNION-Operation in das Ergebnis passen. Für 64-Bit-Werte ist die maximale Ergebnisgenauigkeit 19 und die maximale Ergebnisskalierung 18. Für 128-Bit-Werte ist die maximale Ergebnisgenauigkeit 38 und die maximale Ergebnisskalierung 37.

  Wenn der resultierende Datentyp die AWS Clean Rooms Genauigkeits- und Skalierungsgrenzen überschreitet, gibt die Abfrage einen Fehler zurück.
- Bei Satzoperationen werden zwei Zeilen als identisch behandelt, wenn für jedes korrespondierendes Spaltenpaar die beiden Datenwerte beide gleich oder beide NULL sind. Wenn beispielsweise die Tabellen T1 und T2 beide nur eine Spalte und eine Zeile enthalten und diese Zeile in beiden Tabellen NULL ist, gibt eine INTERSECT-Operation für diese Tabellen diese Zeile zurück.

Beispiel für UNION-Abfragen

In der folgenden UNION-Abfrage werden Zeilen in der Tabelle SALES mit Zeilen in der Tabelle LISTING zusammengeführt. Aus jeder Tabelle werden drei kompatible Spalten ausgewählt. In diesem Fall haben die korrespondierenden Spalten die gleichen Namen und Datentypen.

```
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
```

Das folgende Beispiel zeigt, wie Sie der Ausgabe einer UNION-Abfrage einen Literalwert hinzufügen können, um zu sehen, durch welche Abfrageausdrücke die einzelnen Zeilen im Ergebnissatz jeweils generiert wurden. Die Abfrage identifiziert Zeilen aus dem ersten Abfrageausdruck als „B“ (für Käufer) und Zeilen aus dem zweiten Abfrageausdruck als „S“ (für Verkäufer).

Die Abfrage identifiziert Käufer und Verkäufer für Tickettransaktionen, die einen Wert von mindestens 10.000 USD haben. Der einzige Unterschied zwischen den beiden Abfrageausdrücken auf beiden Seiten des UNION-Operators besteht in der Joining-Spalte für die Tabelle 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
```

Das folgende Beispiel verwendet einen UNION ALL-Operator, da duplizierte Zeilen im Ergebnis beibehalten werden müssen, wenn gefunden. Für eine bestimmte Reihe von Ereignissen IDs gibt die Abfrage 0 oder mehr Zeilen für jeden Verkauf zurück, der mit jedem Ereignis verknüpft ist, und 0 oder 1 Zeile für jede Auflistung dieses Ereignisses. Ereignisse IDs sind für jede Zeile in den Tabellen LISTING und EVENT eindeutig, aber es kann mehrere Verkäufe für dieselbe Kombination aus Ereignis und Angebot IDs in der Tabelle SALES geben.

Die dritte Spalte im Ergebnissatz identifiziert die Quelle der Zeile. Wenn sie aus der Tabelle SALES stammt, wird sie in der Spalte SALESROW mit „Ja“ markiert. (SALESROW ist ein Alias für SALES.LISTID.) Wenn sie aus der Tabelle LISTING stammt, wird sie in der Spalte SALESROW mit „Nein“ markiert.

In diesem Fall besteht der Ergebnissatz aus drei Verkaufszeilen für Auflistung 500, Ereignis 7787. Mit anderen Worten, für diese Kombination von Auflistung und Ereignis fanden drei verschiedene Transaktionen statt. Bei den anderen beiden Auflistungen, 501 und 502, wurden keine Verkäufe erzielt. Daher IDs stammt die einzige Zeile, die die Abfrage für diese Listen generiert, aus der Tabelle 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
```

Wenn Sie die gleiche Abfrage ohne das Schlüsselwort ALL ausführen, gibt das Ergebnis nur eine der Verkaufstransaktionen zurück.

```
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
```

Beispiel für die UNION ALL-Abfrage

Das folgende Beispiel verwendet einen UNION ALL-Operator, da duplizierte Zeilen im Ergebnis beibehalten werden müssen, wenn gefunden. Für eine bestimmte Reihe von Ereignissen IDs gibt die Abfrage 0 oder mehr Zeilen für jeden Verkauf zurück, der mit jeder Veranstaltung verknüpft ist, und 0 oder 1 Zeile für jede Auflistung dieser Veranstaltung. Ereignisse IDs sind für jede Zeile in den Tabellen LISTING und EVENT eindeutig, aber es kann mehrere Verkäufe für dieselbe Kombination aus Ereignis und Angebot IDs in der Tabelle SALES geben.

Die dritte Spalte im Ergebnissatz identifiziert die Quelle der Zeile. Wenn sie aus der Tabelle SALES stammt, wird sie in der Spalte SALESROW mit „Ja“ markiert. (SALESROW ist ein Alias für SALES.LISTID.) Wenn sie aus der Tabelle LISTING stammt, wird sie in der Spalte SALESROW mit „Nein“ markiert.

In diesem Fall besteht der Ergebnissatz aus drei Verkaufszeilen für Auflistung 500, Ereignis 7787. Mit anderen Worten, für diese Kombination von Auflistung und Ereignis fanden drei verschiedene Transaktionen statt. Bei den anderen beiden Auflistungen, 501 und 502, wurden keine Verkäufe erzielt. Daher IDs stammt die einzige Zeile, die die Abfrage für diese Listen generiert, aus der Tabelle 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
```

Wenn Sie die gleiche Abfrage ohne das Schlüsselwort ALL ausführen, gibt das Ergebnis nur eine der Verkaufstransaktionen zurück.

```
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
```

Beispiel für INTERSECT-Abfragen

Vergleichen Sie das folgende Beispiel mit dem ersten UNION-Beispiel. Der einzige Unterschied zwischen den beiden Beispielen besteht im verwendeten Satzoperator. Die Ergebnisse unterscheiden sich jedoch stark. Nur eine Zeile ist identisch:

```
235494 |    23875 |    8771
```

Dies ist die einzige Zeile im begrenzten Ergebnis von 5 Zeilen, die in beiden Tabellen gefunden wurde.

```
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
```

Die folgende Abfrage sucht Veranstaltungen (für die Tickets verkauft wurden), die im März sowohl in New York City als auch in Los Angeles stattfanden. Der Unterschied zwischen den beiden Abfrageausdrücken auf beiden Seiten des UNION-Operators besteht in der Einschränkung für die Spalte 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
```

Beispiel für die EXCEPT-Abfrage

Die CATEGORY-Tabelle in der Datenbank enthält die folgenden 11 Zeilen:

```
 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)
```

Angenommen, eine Tabelle namens CATEGORY_STAGE (eine Staging-Tabelle) enthält eine einzige zusätzliche Zeile:

```
 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)
```

Gibt den Unterschied zwischen den beiden Tabellen zurück. Mit anderen Worten, gibt Zeilen zurück, die in der Tabelle CATEGORY_STAGE, jedoch nicht in der Tabelle CATEGORY enthalten sind:

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

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

Die folgende gleichwertige Abfrage verwendet das Synonym MINUS.

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

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

Wenn Sie die Reihenfolge der SELECT-Ausdrücke umkehren, gibt die Abfrage keine Zeilen zurück.

ORDER BY-Klausel

Die ORDER BY-Klausel sortiert den Ergebnissatz einer Abfrage.

> **Note**
>
> Der äußerste ORDER BY-Ausdruck darf nur Spalten enthalten, die sich in der Auswahlliste befinden.

Themen

- Syntax
- Parameters
- Nutzungshinweise

- Beispiele mit ORDER BY

Syntax

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

Parameters

expression

Ausdruck, der die Sortierreihenfolge des Abfrageergebnisses definiert. Er besteht aus einer oder mehreren Spalten in der Auswahlliste. Die Ergebnisse werden auf der Basis einer binären UTF-8-Reihenfolge zurückgegeben. Sie können auch Folgendes angeben:

- Ordinalzahlen, die die Position der Auswahllisteneinträge darstellen (oder die Position der Spalten in der Tabelle, wenn keine Auswahlliste vorhanden ist)
- Aliase, die Auswahllisteneinträge definieren

Wenn die -Klausel mehrere Ausdrücke enthält, wird der Ergebnissatz nach dem ersten Ausdruck sortiert. Anschließend wird der zweite Ausdruck auf Zeilen mit übereinstimmenden Werten aus dem ersten Ausdruck angewendet usw.

ASC | DESC

Eine Option, die die Sortierreihenfolge für den Ausdruck wie folgt definiert:

- ASC: aufsteigend (beispielsweise niedrig nach hoch für numerische Werte und A bis Z für Zeichenfolgen). Wenn keine Option angegeben wird, werden die Daten standardmäßig in aufsteigender Reihenfolge sortiert.
- DESC: absteigend (beispielsweise hoch nach niedrig für numerische Werte und Z bis A für Zeichenfolgen).

NULLS FIRST | NULLS LAST

Option, die angibt, ob NULL-Werte vor Nicht-Null-Werten oder nach Nicht-Null-Werten aufgelistet werden sollen. Standardmäßig werden NULL-Werte in einer ASC-Reihenfolge an letzter Stelle sortiert und aufgeführt und in einer DESC-Reihenfolge an erster Stelle sortiert und aufgeführt.

LIMIT number | ALL

Option, die die Anzahl der sortierten Zeilen steuert, die von der Abfrage zurückgegeben werden. Bei der LIMIT-Zahl muss es sich um eine positive Ganzzahl handeln. Der maximal zulässige Wert ist 2147483647.

LIMIT 0 gibt keine Zeilen zurück. Sie können diese Syntax für Testzwecke verwenden: um zu prüfen, ob eine Abfrage ausgeführt wird (ohne Zeilen anzuzeigen) oder um eine Spaltenliste aus einer Tabelle zurückzugeben. Eine -Klausel ist redundant, wenn Sie LIMIT 0 verwenden, um eine Spaltenliste zurückzugeben. Der Standardwert ist LIMIT ALL.

OFFSET start

Option, die die Anzahl der Zeilen vor start angibt, die übersprungen werden sollen, bevor Zeilen zurückgegeben werden. Bei der OFFSET-Zahl muss es sich um eine positive Ganzzahl handeln. Der maximal zulässige Wert ist 2147483647. Bei der Verwendung mit der Option LIMIT werden OFFSET-Zeilen übersprungen, bevor die Zahl der LIMIT-Zeilen gezählt werden, die zurückgegeben werden. Wenn die LIMIT-Option nicht verwendet wird, wird die Zahl der Zeilen im Ergebnissatz um die Zahl der übersprungenen Zeilen reduziert. Die von einer OFFSET-Klausel übersprungenen Zeilen müssen dennoch gescannt werden. Daher ist es möglicherweise ineffizient, einen großen OFFSET-Wert zu verwenden.

Nutzungshinweise

Beachten Sie das folgende erwartete Verhalten bei Verwendung von ORDER BY-Klauseln:

- NULL-Werte gelten als „höher“ als alle anderen Werte. Bei Verwendung der standardmäßigen aufsteigenden Sortierfolge befinden sich NULL-Werte am Ende. Um dieses Verhalten zu ändern, wählen Sie die Option NULLS FIRST.
- Wenn eine Anfrage keine ORDER BY-Klausel enthält, gibt das System Ergebnissätze ohne vorhersagbare Anordnung der Zeilen zurück. Wenn dieselbe Abfrage zweimal ausgeführt wird, wird der Ergebnissatz möglicherweise in einer anderen Reihenfolge zurückgegeben.
- Die Optionen LIMIT und OFFSET können ohne ORDER BY-Klausel verwendet werden. Um jedoch einen konsistenten Satz von Zeilen zurückzugeben, verwenden Sie diese Optionen in Verbindung mit ORDER BY.
- In jedem parallel SystemAWS Clean Rooms, wenn ORDER BY keine eindeutige Reihenfolge erzeugt, ist die Reihenfolge der Zeilen nicht deterministisch. Das heißt, wenn der ORDER BY-

Ausdruck doppelte Werte erzeugt, kann die Reihenfolge, in der diese Zeilen zurückgegeben werden, von anderen Systemen oder von einem Lauf AWS Clean Rooms zum nächsten variieren.

- AWS Clean Roomsunterstützt keine Zeichenkettenliterale in ORDER BY-Klauseln.

Beispiele mit ORDER BY

Gibt alle 11 Zeilen aus der Tabelle CATEGORY geordnet nach der zweiten Spalte, CATGROUP, zurück. Ergebnisse, die denselben CATGROUP-Wert haben, ordnen die CATDESC-Spaltenwerte nach der Länge der Zeichenfolge. Dann wird nach Spalten CATID und CATNAME geordnet.

```
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)
```

Gibt ausgewählte Spalten aus der Tabelle SALES zurück, geordnet nach den höchsten QTYSOLD-Werten. Begrenzt das Ergebnis auf die obersten 10 Zeilen:

```
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
```

Gibt unter Verwendung der LIMIT 0-Syntax eine Spaltenliste, aber keine Zeilen zurück:

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

Beispiele für Unterabfragen

In den folgenden Beispielen zeigen verschiedene Möglichkeiten, wie Unterabfragen in SELECT-Abfragen integriert werden können. Ein weiteres Beispiel für die Verwendung von Unterabfragen finden Sie unter Beispiel.

Unterabfragen in der SELECT-Liste

Das folgende Beispiel enthält eine Unterabfrage in der SELECT-Liste. Diese Unterabfrage ist skalar: Sie gibt nur eine Spalte und einen Wert zurück. Dies wird im Ergebnis für jede Zeile wiederholt, die von der umschließenden Abfrage zurückgegeben wird. Die Abfrage vergleicht den von der Unterabfrage berechneten Q1SALES-Wert mit den Verkaufswerten für zwei andere Quartale (2 und 3) im Jahr 2008 wie von der umschließenden Abfrage definiert.

```
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)
```

Unterabfragen in der WHERE-Klausel

Das folgende Beispiel enthält eine Tabellenunterabfrage in der WHERE-Klausel. Diese Unterabfrage produziert mehrere Zeilen. In diesem Fall enthalten die Zeilen nur eine Spalte. Tabellenunterabfragen können jedoch mehrere Spalten und Zeilen enthalten, genau wie jede andere Tabelle.

Die Abfrage sucht die 10 Top-Verkäufer in Bezug die meisten verkauften Tickets. Die Liste der Top 10 wird durch die Unterabfrage eingeschränkt, die Benutzer entfernt, die in Städten mit Ticketverkaufsstellen leben. Diese Abfrage kann auf verschiedene Arten geschrieben werden. Beispielsweise könnte die Unterabfrage als ein Join innerhalb der Hauptabfrage geschrieben werden.

```
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)
```

Unterabfragen in der WITH-Klausel

Siehe WITH-Klausel.

Korrelierte Unterabfragen

Das folgende Beispiel enthält eine korrelierte Unterabfrage in der WHERE-Klausel. Diese Art von Unterabfrage enthält mindestens eine Korrelation zwischen ihren Spalten und den Spalten, die von der umschließenden Abfrage produziert werden. In diesem Fall ist die Korrelation `where`

`s.listid=l.listid`. Die Unterabfrage wird für jede Zeile ausgeführt, die die umschließende Abfrage produziert, um die Zeile zu qualifizieren oder zu disqualifizieren.

```
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)
```

Muster für korrelierte Unterabfragen, die nicht unterstützt werden

Der Abfrageplaner verwendet eine Methode für das Neuschreiben von Abfragen, die als Entkorrelierung von Unterabfragen bezeichnet wird, um verschiedene Muster korrelierter Unterabfragen für die Ausführung in einer MPP-Umgebung zu optimieren. Einige Typen von korrelierten Unterabfragen folgen Mustern, die nicht korreliert AWS Clean Rooms werden können und auch nicht unterstützt werden. Abfragen, die die folgenden Korrelierungsreferenzen enthalten, geben Fehler zurück:

- Korrelierungsreferenzen, die einen Abfrageblock überspringen, auch als „überspringende Korrelierungsreferenzen“ bekannt. Beispielsweise sind in der folgenden Abfrage der Block mit der Korrelierungsreferenz und der übersprungene Block durch ein NOT EXISTS-Prädikat verbunden:

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

  Der übersprungende Block ist in diesem Fall die Unterabfrage für die LISTING-Tabelle. Die Korrelierungsreferenz korreliert die Tabellen EVENT und SALES.

- Korrelierungsreferenzen aus einer Unterabfrage, die Teil einer ON-Klausel in einer externen Abfrage ist:

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

  Die ON-Klausel enthält eine Korrelierungsreferenz aus SALES in der Unterabfrage für EVENT in der umschließenden Abfrage.

- Korrelationsreferenzen, die auf Null reagieren, auf eine Systemtabelle. AWS Clean Rooms Beispiel:

```
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);
```

- Korrelierungsreferenzen aus einer Unterabfrage, die eine Fensterfunktion enthält.

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

- Referenzen in einer GROUP BY-Spalte zu den Ergebnissen einer korrelierten Unterabfrage. Beispiel:

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

- Korrelierungsreferenzen aus einer Unterabfrage mit einer Aggregationsfunktion und einer GROUP BY-Klausel, die durch ein IN-Prädikat mit der umschließenden Abfrage verbunden sind. (Diese Einschränkung gilt nicht für die Aggregationsfunktionen MIN und MAX.) Beispiel:

```
select * from listing where listid in
(select sum(qtysold)
from sales
```

```
where numtickets>4
group by salesid);
```

AWS Clean Rooms Spark SQL-Funktionen

AWS Clean Rooms Spark SQL unterstützt die folgenden SQL-Funktionen:

Themen

Aggregationsfunktionen

Aggregatfunktionen in AWS Clean Rooms Spark SQL werden verwendet, um Berechnungen oder Operationen für eine Gruppe von Zeilen durchzuführen und einen einzelnen Wert zurückzugeben. Sie sind für Datenanalyse- und Zusammenfassungsaufgaben unerlässlich.

AWS Clean Rooms Spark SQL unterstützt die folgenden Aggregatfunktionen:

Themen

- APPROX COUNT_DISTINCT-Funktion
- Funktion „UNGEFÄHRES PERZENTIL“
- AVG Funktion
- Die Funktion BOOL_AND
- Die Funktion BOOL_OR
- CARDINALITY-Funktion
- Funktion COLLECT_LIST
- Funktion COLLECT_SET
- COUNTund COUNT DISTINCT Funktionen
- Die Funktion COUNT
- Die Funktion MAX
- Die Funktion MEDIAN
- Die Funktion MIN
- PERZENTILE-Funktion
- SKEWNESS-Funktion
- Die Funktionen STDDEV_SAMP und STDDEV_POP
- SUMund SUM DISTINCT Funktionen
- Die Funktionen VAR_SAMP und VAR_POP

Funktion ANY_VALUE

Die Funktion ANY_VALUE gibt einen beliebigen Wert aus den Eingabeausdruckswerten nicht deterministisch zurück. Diese Funktion kann NULL zurückgeben, wenn der Eingabeausdruck nicht dazu führt, dass Zeilen zurückgegeben werden.

Syntax

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

Argumente

Ausdruck

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird. Der Ausdruck ist einer der folgenden Datentypen:

isIgnoreNull

Ein boolescher Wert, der bestimmt, ob die Funktion nur Werte zurückgeben soll, die ungleich Null sind.

Rückgabewert

Gibt denselben Datentyp wie expression zurück.

Nutzungshinweise

Wenn eine Anweisung, die die Funktion ANY_VALUE für eine Spalte angibt, auch einen Verweis auf eine zweite Spalte enthält, muss die zweite Spalte in einer GROUP-BY-Klausel oder in einer Aggregationsfunktion enthalten sein.

Beispiele

Das folgende Beispiel gibt eine Instanz von any zurück, `dateid` wo der ist. `eventname` `Eagles`

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

Die Ergebnisse sehen wie folgt aus.

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

Im folgenden Beispiel wird eine Instanz von any zurückgegeben`dateid`, wobei der Wert `Eagles` oder `eventname` ist`Cold War Kids`.

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

Die Ergebnisse sehen wie folgt aus.

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

APPROX COUNT_DISTINCT-Funktion

APPROX COUNT_DISTINCT bietet eine effiziente Methode, um die Anzahl der Einzelwerte in einer Spalte oder einem Datensatz zu schätzen.

Syntax

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

Argumente

expr

Der Ausdruck oder die Spalte, für die Sie die Anzahl der Einzelwerte schätzen möchten.

Dabei kann es sich um eine einzelne Spalte, einen komplexen Ausdruck oder eine Kombination von Spalten handeln.

Verwandte D

Ein optionaler Parameter, der die gewünschte relative Standardabweichung der Schätzung angibt.

Es handelt sich um einen Wert zwischen 0 und 1, der den maximal akzeptablen relativen Fehler der Schätzung darstellt. Ein kleinerer RelativeSD-Wert führt zu einer genaueren, aber langsameren Schätzung.

Wenn dieser Parameter nicht angegeben wird, wird ein Standardwert (normalerweise etwa 0,05 oder 5%) verwendet.

Rückgabewert

Gibt die geschätzte Kardinalität von HyperLogLog ++ zurück. RelativeSD definiert die maximal zulässige relative Standardabweichung.

Beispiel

Die folgende Abfrage schätzt die Anzahl der Einzelwerte in der `col1` Spalte mit einer relativen Standardabweichung von 1% (0,01).

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

Die folgende Abfrage schätzt, dass die `col1` Spalte 3 Einzelwerte enthält (die Werte 1, 2 und 3).

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

Funktion „UNGEFÄHRES PERZENTIL“

APPROX PERCENTILE wird verwendet, um den Perzentilwert eines bestimmten Ausdrucks oder einer bestimmten Spalte zu schätzen, ohne den gesamten Datensatz sortieren zu müssen. Diese Funktion ist in Szenarien nützlich, in denen Sie schnell die Verteilung eines großen Datensatzes verstehen oder auf Perzentilen basierende Metriken verfolgen müssen, ohne den Rechenaufwand für die Durchführung einer exakten Perzentilberechnung aufwenden zu müssen. Es ist jedoch wichtig, die Kompromisse zwischen Geschwindigkeit und Genauigkeit zu verstehen und die richtige Fehlertoleranz auf der Grundlage der spezifischen Anforderungen Ihres Anwendungsfalls auszuwählen.

Syntax

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

Argumente

expr

Der Ausdruck oder die Spalte, für die Sie den Perzentilwert schätzen möchten.

Dabei kann es sich um eine einzelne Spalte, einen komplexen Ausdruck oder eine Kombination von Spalten handeln.

percentile

Der Perzentilwert, den Sie schätzen möchten, ausgedrückt als Wert zwischen 0 und 1.

Beispielsweise würde 0,5 dem 50. Perzentil (Median) entsprechen.

Genauigkeit

Ein optionaler Parameter, der die gewünschte Genauigkeit der Perzentilschätzung angibt. Es handelt sich um einen Wert zwischen 0 und 1, der den maximal akzeptablen relativen Fehler der Schätzung darstellt. Ein kleinerer `accuracy` Wert führt zu einer genaueren, aber langsameren Schätzung. Wenn dieser Parameter nicht angegeben wird, wird ein Standardwert (normalerweise etwa 0,05 oder 5%) verwendet.

Rückgabewert

Gibt das ungefähre Perzentil der numerischen oder ANSI-Intervallspalte col zurück, das der kleinste Wert in den geordneten COL-Werten ist (sortiert vom kleinsten zum größten), sodass nicht mehr als ein Prozentsatz der COL-Werte kleiner als der Wert oder gleich diesem Wert ist.

Der Prozentwert muss zwischen 0,0 und 1,0 liegen. Der Genauigkeitsparameter (Standard: 10000) ist ein positives numerisches Literal, das die Näherungsgenauigkeit auf Kosten des Speichers steuert.

Ein höherer Genauigkeitswert führt zu einer besseren Genauigkeit. Dies `1.0/accuracy` ist der relative Fehler der Näherung.

Wenn es sich bei Prozent um eine Matrix handelt, muss jeder Wert der Prozentmatrix zwischen 0,0 und 1,0 liegen. Gibt in diesem Fall das ungefähre Perzentil-Array der Spalte col bei der angegebenen Prozentzahl zurück.

Beispiele

Die folgende Abfrage schätzt das 95. Perzentil der `response_time` Spalte mit einem maximalen relativen Fehler von 1% (0,01).

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

Mit der folgenden Abfrage werden die Werte für das 50., 40. und 10. Perzentil der Spalte in der Tabelle geschätzt. `col tab`

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

Mit der folgenden Abfrage wird das 50. Perzentil (Median) der Werte in der Spalte Spalte geschätzt.

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

AVG Funktion

Die AVG Funktion gibt den Durchschnitt (das arithmetische Mittel) der eingegebenen Ausdruckswerte zurück. Die AVG Funktion arbeitet mit numerischen Werten und ignoriert NULL-Werte.

Syntax

```
AVG (column)
```

Argumente

column

Die Zielspalte, in der die Funktion ausgeführt wird. Die Spalte ist einer der folgenden Datentypen:

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

Datentypen

Die von der AVG Funktion unterstützten Argumenttypen sind SMALLINTINTEGER,BIGINT,DECIMAL, undDOUBLE.

Die von der AVG Funktion unterstützten Rückgabetypen sind:

- BIGINTfür jedes Argument vom Typ Integer
- DOUBLEfür ein Fließkomma-Argument
- Gibt den gleichen Datentyp wie der Ausdruck für jeden anderen Argumenttyp zurück

Die Standardgenauigkeit für ein AVG Funktionsergebnis mit einem DECIMAL Argument ist 38. Die Ergebnisskala ist die gleiche wie die Skala des Arguments. Beispielsweise gibt der Wert AVG einer DEC(5,2) Spalte einen DEC(38,2) Datentyp zurück.

Beispiel

Suchen Sie in der SALES Tabelle nach der durchschnittlichen Verkaufsmenge pro Transaktion.

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

Die Funktion BOOL_AND

Die Funktion BOOL_AND wird für eine einzige boolesche oder Ganzzahlspalte bzw. einen einzigen booleschen oder Ganzzahlausdruck ausgeführt. Diese Funktion wendet ähnliche Logik auf die Funktionen BIT_AND und BIT_OR an. Für diese Funktion ist der Rückgabetyp ein boolescher Wert (`true` oder `false`).

Wenn alle Werte in einem Satz „true“ sind, gibt die Funktion BOOL_AND `true` (`t`) zurück. Wenn ein Wert „false“ ist, gibt die Funktion `false` (`f`) zurück.

Syntax

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

Argumente

Ausdruck

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird. Dieser Ausdruck muss einen booleschen oder Ganzzahl-Datentyp haben. Der Rückgabewert der Funktion ist BOOLEAN.

DISTINCT | ALL

Mit dem Argument DISTINCT beseitigt die Funktion alle duplizierten Werte für den angegebenen Ausdruck, bevor das Ergebnis berechnet wird. Mit dem Argument ALL behält die Funktion alle duplizierten Werte. ALL ist das Standardargument.

Beispiele

Sie können die booleschen Funktionen auf boolesche Ausdrücke oder Ganzzahlausdrücke anwenden.

Beispielsweise gibt die folgende Abfrage Ergebnisse aus der Standardtabelle USERS in der Datenbank TICKIT zurück, die mehrere boolesche Spalten besitzt.

Die Funktion BOOL_AND gibt für alle fünf Zeilen `false` zurück. Nicht allen Benutzern in diesen Bundesstaaten gefällt 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)
```

Die Funktion BOOL_OR

Die Funktion BOOL_OR wird für eine einzige boolesche oder Ganzzahlspalte bzw. einen einzigen booleschen oder Ganzzahlausdruck ausgeführt. Diese Funktion wendet ähnliche Logik auf die Funktionen BIT_AND und BIT_OR an. Für diese Funktion ist der Rückgabetyp ein boolescher Wert (`true`, `false` oder `NULL`).

Wenn ein Wert in einem Satz `true` lautet, gibt die Funktion BOOL_OR `true` (`t`) zurück. Wenn ein Wert in einem Satz `false` lautet, gibt die Funktion `false` (`f`) zurück. NULL kann zurückgegeben werden, wenn der Wert unbekannt ist.

Syntax

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

Argumente

Ausdruck

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird. Dieser Ausdruck muss einen booleschen oder Ganzzahl-Datentyp haben. Der Rückgabewert der Funktion ist BOOLEAN.

DISTINCT | ALL

Mit dem Argument DISTINCT beseitigt die Funktion alle duplizierten Werte für den angegebenen Ausdruck, bevor das Ergebnis berechnet wird. Mit dem Argument ALL behält die Funktion alle duplizierten Werte. ALL ist das Standardargument.

Beispiele

Sie können die booleschen Funktionen mit booleschen Ausdrücken oder Ganzzahlausdrücken verwenden. Beispielsweise gibt die folgende Abfrage Ergebnisse aus der Standardtabelle USERS in der Datenbank TICKIT zurück, die mehrere boolesche Spalten besitzt.

Die Funktion BOOL_OR gibt für alle fünf Zeilen `true` zurück. Mindestens einem Benutzer in diesen Bundesstaaten gefällt 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)
```

Im folgenden Beispiel wird NULL zurückgegeben.

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

CARDINALITY-Funktion

Die CARDINALITY-Funktion gibt die Größe eines ARRAY- oder MAP-Ausdrucks (expr) zurück.

Diese Funktion ist nützlich, um die Größe oder Länge eines Arrays zu ermitteln.

Syntax

```
cardinality(expr)
```

Argumente

expr

Ein ARRAY- oder MAP-Ausdruck.

Rückgabewert

Gibt die Größe eines Arrays oder einer Map zurück (INTEGER).

Die Funktion gibt NULL bei einer Null-Eingabe zurück, ob `sizeOfNull` sie auf `false` oder gesetzt `enabled` ist`true`.

Andernfalls kehrt `-1` die Funktion bei einer Null-Eingabe zurück. Mit den Standardeinstellungen kehrt die Funktion bei einer `-1` Nulleingabe zurück.

Beispiel

Die folgende Abfrage berechnet die Kardinalität oder die Anzahl der Elemente im angegebenen Array. Das Array (`'b', 'd', 'c', 'a'`) hat 4 Elemente, also wäre die Ausgabe dieser Abfrage. `4`

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

Funktion COLLECT_LIST

Die Funktion COLLECT_LIST sammelt eine Liste von nicht eindeutigen Elementen und gibt sie zurück.

Diese Art von Funktion ist nützlich, wenn Sie mehrere Werte aus einer Reihe von Zeilen in einer einzigen Array- oder Listendatenstruktur sammeln möchten.

> **Note**
>
> Die Funktion ist nicht deterministisch, da die Reihenfolge der gesammelten Ergebnisse von der Reihenfolge der Zeilen abhängt, die nach einer Shuffle-Operation möglicherweise nicht deterministisch ist.

Syntax

```
collect_list(expr)
```

Argumente

expr

Ein Ausdruck beliebigen Typs.

Rückgabewert

Gibt ein ARRAY des Argumenttyps zurück. Die Reihenfolge der Elemente im Array ist nicht deterministisch.

NULL-Werte sind ausgeschlossen.

Wenn DISTINCT angegeben ist, sammelt die Funktion nur eindeutige Werte und ist ein Synonym für `collect_set` Aggregatfunktion.

Beispiel

Die folgende Abfrage sammelt alle Werte aus der Spalte col in einer Liste. Die `VALUES` Klausel wird verwendet, um eine Inline-Tabelle mit drei Zeilen zu erstellen, wobei jede Zeile eine einzelne Spalte col mit den Werten 1, 2 und 1 hat. Die `collect_list()` Funktion wird dann verwendet, um alle Werte aus der Spalte col in einem einzigen Array zu aggregieren. Die Ausgabe dieser SQL-Anweisung wäre das Array`[1,2,1]`, das alle Werte aus der Spalte col in der Reihenfolge enthält, in der sie in den Eingabedaten erschienen sind.

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

Funktion COLLECT_SET

Die Funktion COLLECT_SET sammelt eine Reihe von eindeutigen Elementen und gibt sie zurück.

Diese Funktion ist nützlich, wenn Sie alle unterschiedlichen Werte aus einer Reihe von Zeilen in einer einzigen Datenstruktur sammeln möchten, ohne Duplikate einzubeziehen.

> **Note**
>
> Die Funktion ist nicht deterministisch, da die Reihenfolge der gesammelten Ergebnisse von der Reihenfolge der Zeilen abhängt, die nach einer Shuffle-Operation möglicherweise nicht deterministisch ist.

Syntax

```
collect_set(expr)
```

Argumente

expr

Ein Ausdruck eines beliebigen Typs außer MAP.

Rückgabewert

Gibt ein ARRAY des Argumenttyps zurück. Die Reihenfolge der Elemente im Array ist nicht deterministisch.

NULL-Werte sind ausgeschlossen.

Beispiel

Die folgende Abfrage sammelt alle Einzelwerte aus der Spalte col in einem Satz. Die VALUES Klausel wird verwendet, um eine Inline-Tabelle mit drei Zeilen zu erstellen, wobei jede Zeile eine einzelne Spalte col mit den Werten 1, 2 und 1 hat. Die `collect_set()` Funktion wird dann verwendet, um alle Einzelwerte aus der Spalte col zu einem einzigen Satz zusammenzufassen. Die Ausgabe dieser SQL-Anweisung wäre der Satz`[1,2]`, der die eindeutigen Werte aus der Spalte col enthält. Der doppelte Wert 1 ist nur einmal im Ergebnis enthalten.

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

COUNTund COUNT DISTINCT Funktionen

Die COUNT Funktion zählt die durch den Ausdruck definierten Zeilen. Die COUNT DISTINCT Funktion berechnet die Anzahl der unterschiedlichen Werte, die nicht NULL sind, in einer Spalte oder einem Ausdruck. Sie entfernt alle doppelten Werte aus dem angegebenen Ausdruck, bevor die Zählung durchgeführt wird.

Syntax

```
COUNT (DISTINCT column)
```

Argumente

column

Die Zielspalte, in der die Funktion ausgeführt wird.

Datentypen

Die COUNT Funktion und die COUNT DISTINCT Funktion unterstützen alle Argumentdatentypen.

Die COUNT DISTINCT Funktion kehrt zurückBIGINT.

Beispiele

Zählen Sie alle Benutzer aus dem Bundesstaat Florida.

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

Zählen Sie alle einzigartigen Veranstaltungsorte IDs anhand der EVENT Tabelle.

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

Die Funktion COUNT

Die Funktion COUNT zählt die durch den Ausdruck definierten Zeilen.

Zu der Funktion COUNT gibt es folgende Varianten.

- COUNT (*) zählt alle Zeilen in der Zieltabelle, unabhängig davon, ob sie Null-Werte enthalten oder nicht.
- COUNT (expression) berechnet die Zahl der Zeilen mit Nicht-NULL-Werten in einer spezifischen Spalte oder einem spezifischen Ausdruck.
- COUNT (DISTINCT expression) berechnet die Zahl der unterschiedlichen Nicht-NULL-Werte in einer Spalte oder einem Ausdruck.

Syntax

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

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

Argumente

Ausdruck

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird. Die Funktion COUNT unterstützt alle Argumentdatentypen.

DISTINCT | ALL

Mit dem Argument DISTINCT beseitigt die Funktion alle duplizierten Werte aus dem angegebenen Ausdruck, bevor die Zählung ausgeführt wird. Mit dem Argument ALL behält die Funktion alle duplizierten Werte aus dem angegebenen Ausdruck, um die Zählung auszuführen. ALL ist das Standardargument.

Rückgabetyp

Die Funktion COUNT gibt BIGINT zurück.

Beispiele

Zählung aller Benutzer aus dem Bundesstaat Florida:

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

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

Zählung aller Ereignisnamen aus der EVENT-Tabelle:

```
select count(eventname) from event;

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

Zählung aller Ereignisnamen aus der EVENT-Tabelle:

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

count
-------
```

```
8798
```

Zählen Sie alle einzigartigen Veranstaltungsorte IDs aus der EVENT-Tabelle:

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

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

Zählung der Häufigkeit, mit der die einzelnen Verkäufer Batches von mehr als vier Tickets zum Verkauf aufgelistet haben; Gruppierung der Ergebnisse nach Verkäufer-ID:

```
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
...
```

Die Funktion MAX

Die Funktion MAX gibt den maximal zulässigen Wert in einem Satz von Zeilen zurück. DISTINCT oder ALL könnten zwar verwendet werden, wirken sich jedoch nicht auf das Ergebnis aus.

Syntax

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

Argumente

Ausdruck

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird. Der Ausdruck ist ein beliebiger numerischer Datentyp.

DISTINCT | ALL

Mit dem Argument DISTINCT beseitigt die Funktion alle duplizierten Werte aus dem angegebenen Ausdruck, bevor der maximal zulässige Wert berechnet wird. Mit dem Argument ALL behält die Funktion alle duplizierten Werte aus dem angegebenen Ausdruck, um den maximal zulässigen Wert zu berechnen. ALL ist das Standardargument.

Datentypen

Gibt denselben Datentyp wie expression zurück.

Beispiele

Suche des höchsten Preises, der in allen Verkäufen gezahlt wurde:

```
select max(pricepaid) from sales;

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

Suche des höchsten Preises pro Ticket, der in allen Verkäufen gezahlt wurde:

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

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

Die Funktion MEDIAN

Syntax

```
MEDIAN ( median_expression )
```

Argumente

median_expression

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird.

Die Funktion MIN

Die Funktion MIN gibt den Mindestwert in einem Satz von Zeilen zurück. DISTINCT oder ALL könnten zwar verwendet werden, wirken sich jedoch nicht auf das Ergebnis aus.

Syntax

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

Argumente

Ausdruck

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird. Der Ausdruck ist ein beliebiger numerischer Datentyp.

DISTINCT | ALL

Mit dem Argument DISTINCT beseitigt die Funktion alle duplizierten Werte aus dem angegebenen Ausdruck, bevor der Mindestwert berechnet wird. Mit dem Argument ALL behält die Funktion alle duplizierten Werte aus dem angegebenen Ausdruck, um den Mindestwert zu berechnen. ALL ist das Standardargument.

Datentypen

Gibt denselben Datentyp wie expression zurück.

Beispiele

Suche des niedrigsten Preises, der in allen Verkäufen gezahlt wurde:

```
select min(pricepaid) from sales;

min
-------
```

```
20.00
(1 row)
```

Suche des niedrigsten Preises pro Ticket, der in allen Verkäufen gezahlt wurde:

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

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

PERZENTILE-Funktion

Die PERCENTILE-Funktion wird verwendet, um den exakten Perzentilwert zu berechnen, indem zuerst die Werte in der Spalte sortiert und dann der Wert am angegebenen Wert ermittelt wird. `col percentage`

Die PERZENTILE-Funktion ist nützlich, wenn Sie den exakten Perzentilwert berechnen müssen und der Rechenaufwand für Ihren Anwendungsfall akzeptabel ist. Sie liefert genauere Ergebnisse als die APPROX_PERCENTILE-Funktion, ist aber möglicherweise langsamer, insbesondere bei großen Datensätzen.

Im Gegensatz dazu ist die Funktion APPROX_PERCENTILE eine effizientere Alternative, da sie eine Schätzung des Perzentilwerts mit einer bestimmten Fehlertoleranz liefern kann. Dadurch eignet sie sich besser für Szenarien, in denen Geschwindigkeit eine höhere Priorität als absolute Genauigkeit hat.

Syntax

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

Argumente

Spalte

Der Ausdruck oder die Spalte, für die Sie den Perzentilwert berechnen möchten.

Prozentsatz

Der Perzentilwert, den Sie berechnen möchten, ausgedrückt als Wert zwischen 0 und 1.

Beispielsweise würde 0,5 dem 50. Perzentil (Median) entsprechen.

Frequenz

Ein optionaler Parameter, der die Häufigkeit oder Gewichtung der einzelnen Werte in der `col` Spalte angibt. Falls angegeben, berechnet die Funktion das Perzentil auf der Grundlage der Häufigkeit der einzelnen Werte.

Rückgabewert

Gibt den exakten Perzentilwert der numerischen oder ANSI-Intervallspalte col zum angegebenen Prozentsatz zurück.

Der Prozentwert muss zwischen 0,0 und 1,0 liegen.

Der Wert der Frequenz sollte ein positives Integral sein

Beispiel

Die folgende Abfrage findet den Wert, der größer oder gleich 30% der Werte in der `col` Spalte ist. Da die Werte 0 und 10 sind, ist das 30. Perzentil 3,0, weil es der Wert ist, der größer oder gleich 30% der Daten ist.

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

SKEWNESS-Funktion

Die Funktion SKEWNESS gibt den Wert der Schiefe zurück, der anhand der Werte einer Gruppe berechnet wurde.

Die Schiefe ist ein statistisches Maß, das die Asymmetrie oder den Mangel an Symmetrie in einem Datensatz beschreibt. Sie liefert Informationen über die Form der Datenverteilung.

Diese Funktion kann nützlich sein, um die statistischen Eigenschaften eines Datensatzes zu verstehen und als Grundlage für weitere Analysen oder Entscheidungen zu dienen.

Syntax

```
skewness(expr)
```

Argumente

expr

Ein Ausdruck, der zu einer Zahl ausgewertet wird.

Rückgabewert

Gibt DOUBLE zurück.

Wenn DISTINCT angegeben ist, arbeitet die Funktion nur mit einem eindeutigen Satz von Ausdruckwerten.

Beispiele

Die folgende Abfrage berechnet die Schiefe der Werte in der Spalte. `col` In diesem Beispiel wird die `VALUES` Klausel verwendet, um eine Inline-Tabelle mit vier Zeilen zu erstellen, wobei jede Zeile eine einzelne Spalte `col` mit den Werten -10, -20, 100 und 1000 enthält. Die `skewness()` Funktion wird dann verwendet, um die Schiefe der Werte in der `col` Spalte zu berechnen. Das Ergebnis, 1.1135657469022011, stellt den Grad und die Richtung der Schiefe in den Daten dar. Ein positiver Wert für die Schiefe gibt an, dass die Daten nach rechts geneigt sind, sodass sich der Großteil der Werte auf der linken Seite der Verteilung konzentriert. Ein negativer Wert für die Schiefe gibt an, dass die Daten nach links geneigt sind und sich der Großteil der Werte auf der rechten Seite der Verteilung konzentriert.

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

Die folgende Abfrage berechnet die Schiefe der Werte in der Spalte col. Ähnlich wie im vorherigen Beispiel wird die `VALUES` Klausel verwendet, um eine Inline-Tabelle mit vier Zeilen zu erstellen, wobei jede Zeile eine einzelne Spalte `col` mit den Werten -1000, -100, 10 und 20 enthält. Die `skewness()` Funktion wird dann verwendet, um die Schiefe der Werte in der Spalte zu berechnen. `col` Das Ergebnis, -1,135657469022011, stellt den Grad und die Richtung der Schiefe in den Daten dar. In diesem Fall gibt der negative Wert für die Schiefe an, dass die Daten nach links geneigt sind, sodass sich der Großteil der Werte auf der rechten Seite der Verteilung konzentriert.

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

Die Funktionen STDDEV_SAMP und STDDEV_POP

Die Funktionen STDDEV_SAMP und STDDEV_POP geben die Stichproben- und Populationsstandardabweichungen eines Satzes numerischer Werte (integer, decimal oder floating-point) zurück. Das Ergebnis der Funktion STDDEV_SAMP entspricht der Quadratwurzel der Stichprobenabweichung desselben Satzes von Werten.

STDDEV_SAMP und STDDEV sind Synonyme für dieselbe Funktion.

Syntax

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

Der Ausdruck muss einen numerischen Datentyp haben. Unabhängig vom Datentyp des Ausdrucks ist der Rückgabewert dieser Funktion eine DOUBLE PRECISION-Zahl.

> **Note**
>
> Die Standardabweichung wird mittels Gleitkommaarithmetik berechnet. Dies kann zu einer leichten Ungenauigkeit führen.

Nutzungshinweise

Wenn die Stichprobenstandardabweichung (STDDEV oder STDDEV_SAMP) für einen Ausdruck berechnet wird, der aus einem einzigen Wert besteht, ist das Ergebnis der Funktion NULL und nicht 0.

Beispiele

Die folgende Abfrage gibt den Durchschnitt der Werte in der Spalte VENUESEATS der Tabelle VENUE zurück, gefolgt von der Stichprobenstandardabweichung und der Populationsstandardabweichung desselben Satzes von Werten. VENUESEATS ist eine INTEGER-Spalte. Die Ergebnisskala ist auf 2 Ziffern reduziert.

```
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)
```

Die folgende Abfrage gibt die Stichprobenstandardabweichung für die Spalte COMMISSION in der Tabelle SALES zurück. COMMISSION ist eine DECIMAL-Spalte. Die Ergebnisskala ist auf 10 Ziffern reduziert.

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

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

Die folgende Abfrage gibt die Stichprobenstandardabweichung für die Spalte COMMISSION als Ganzzahl aus.

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

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

Die folgende Abfrage gibt sowohl die Stichprobenstandardabweichung als auch die Quadratwurzel der Stichprobenabweichung für die Spalte COMMISSION zurück. Die Ergebnisse dieser Berechnungen sind identisch.

```
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)
```

SUMund SUM DISTINCT Funktionen

Die SUM Funktion gibt die Summe der Eingabespalten- oder Ausdruckswerte zurück. Die SUM Funktion arbeitet mit numerischen Werten und ignoriert NULL Werte.

Die SUM DISTINCT Funktion entfernt alle doppelten Werte aus dem angegebenen Ausdruck, bevor die Summe berechnet wird.

Syntax

```
SUM (DISTINCT column )
```

Argumente

column

Die Zielspalte, in der die Funktion ausgeführt wird. Bei der Spalte handelt es sich um beliebige numerische Datentypen.

Beispiele

Ermitteln Sie die Summe aller gezahlten Provisionen anhand der SALES Tabelle.

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

Ermitteln Sie die Summe aller einzelnen gezahlten Provisionen aus der SALES Tabelle.

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

Die Funktionen VAR_SAMP und VAR_POP

Die Funktionen VAR_SAMP und VAR_POP geben die Stichproben- und Populationsabweichung eines Satzes numerischer Werte (integer, decimal oder floating-point) zurück. Das Ergebnis der Funktion VAR_SAMP entspricht der Quadratwurzel der Stichprobenstandardabweichung desselben Satzes von Werten.

VAR_SAMP und VARIANCE sind Synonyme für dieselbe Funktion.

Syntax

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

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

Der Ausdruck muss einen Ganzzahl-, Dezimal- oder Gleitkommadatentyp haben. Unabhängig vom Datentyp des Ausdrucks ist der Rückgabewert dieser Funktion eine DOUBLE PRECISION-Zahl.

> **Note**
>
> Die Ergebnisse dieser Funktionen sind je nach Data Warehouse-Cluster verschieden, abhängig von der Konfiguration des jeweiligen Clusters.

Nutzungshinweise

Wenn die Stichprobenabweichung (VARIANCE oder VAR_SAMP) für einen Ausdruck berechnet wird, der aus einem einzigen Wert besteht, ist das Ergebnis der Funktion NULL und nicht 0.

Beispiele

Die folgende Abfrage gibt die gerundete Stichproben- und Populationsabweichung für die Spalte NUMTICKETS in der Tabelle LISTING zurück.

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

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

Die folgende Abfrage führt dieselben Berechnungen aus, gibt die Ergebnisse jedoch als Dezimalwerte aus.

```
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)
```

Array-Funktionen

In diesem Abschnitt werden die Array-Funktionen für SQL beschrieben, die in AWS Clean Rooms unterstützt werden.

Themen

ARRAY-Funktion

Erzeugt ein Array mit den angegebenen Elementen.

Syntax

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

Argument

expr1, expr2

Ausdrücke aller Datentypen außer Datums- und Uhrzeittypen. Die Argumente müssen nicht denselben Datentyp haben.

Rückgabetyp

Die Array-Funktion gibt ein ARRAY mit den Elementen im Ausdruck zurück.

Beispiel

Das folgende Beispiel zeigt ein Array mit numerischen Werten und ein Array mit verschiedenen Datentypen.

```
--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)
```

Funktion ARRAY_CONTAINS

Die Funktion ARRAY_CONTAINS kann verwendet werden, um grundlegende Mitgliedschaftsprüfungen für Array-Datenstrukturen durchzuführen. Die Funktion ARRAY_CONTAINS ist nützlich, wenn Sie überprüfen müssen, ob ein bestimmter Wert in einem Array vorhanden ist.

Syntax

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

Argumente

Array

Ein zu durchsuchendes ARRAY.

Wert

Ein Ausdruck mit einem Typ, der den Typ, der den Array-Elementen am wenigsten gemeinsam ist.

Rückgabetyp

Die Funktion ARRAY_CONTAINS gibt einen Wert vom Typ BOOLEAN zurück.

Wenn der Wert NULL ist, ist das Ergebnis NULL.

Wenn ein Element im Array NULL ist, ist das Ergebnis NULL, wenn der Wert keinem anderen Element zugeordnet ist.

Beispiele

Im folgenden Beispiel wird geprüft, ob das Array den Wert `[1, 2, 3]` enthält4. Da das `[1, 2, 3` [Array] den Wert nicht enthält4, gibt die Funktion array_contains zurück. `false`

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

Im folgenden Beispiel wird geprüft, ob das Array den Wert `[1, 2, 3]` enthält. 2 Da das Array den Wert `[1, 2, 3]` enthält2, gibt die Funktion array_contains zurück. `true`

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

ARRAY_DISTINCT-Funktion

Die Funktion ARRAY_DISTINCT kann verwendet werden, um doppelte Werte aus einem Array zu entfernen. Die Funktion ARRAY_DISTINCT ist nützlich, wenn Sie Duplikate aus einem Array entfernen und nur mit den eindeutigen Elementen arbeiten müssen. Dies kann in Szenarien hilfreich sein, in denen Sie Operationen oder Analysen an einem Datensatz durchführen möchten, ohne dass sich wiederholte Werte gegenseitig beeinflussen.

Syntax

```
array_distinct(array)
```

Argumente

Array

Ein ARRAY-Ausdruck.

Rückgabetyp

Die Funktion ARRAY_DISTINCT gibt ein ARRAY zurück, das nur die eindeutigen Elemente aus dem Eingabearray enthält.

Beispiele

In diesem Beispiel `[1, 2, 3, null, 3]` enthält das Eingabearray einen doppelten Wert von. `3` Die `array_distinct` Funktion entfernt diesen doppelten Wert `3` und gibt ein neues Array mit den eindeutigen Elementen zurück:`[1, 2, 3, null]`.

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

In diesem Beispiel `[1, 2, 2, 3, 3, 3]` enthält das Eingabearray doppelte Werte von `2` und`3`. Die `array_distinct` Funktion entfernt diese Duplikate und gibt ein neues Array mit den eindeutigen Elementen zurück:`[1, 2, 3]`.

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

ARRAY_EXCEPT-Funktion

Die Funktion ARRAY_EXCEPT verwendet zwei Arrays als Argumente und gibt ein neues Array zurück, das nur die Elemente enthält, die im ersten Array vorhanden sind, aber nicht im zweiten Array.

Die ARRAY_EXCEPT ist nützlich, wenn Sie die Elemente finden müssen, die für ein Array im Vergleich zu einem anderen einzigartig sind. Dies kann in Szenarien hilfreich sein, in denen Sie mengenähnliche Operationen an Arrays ausführen müssen, z. B. um den Unterschied zwischen zwei Datensätzen zu ermitteln.

Syntax

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

Argumente

Matrix1

Ein ARRAY beliebigen Typs mit vergleichbaren Elementen.

array2

Ein ARRAY von Elementen, deren Typ mit den Elementen von array1 am wenigsten gemeinsam ist.

Rückgabetyp

Die Funktion ARRAY_EXCEPT gibt ein ARRAY zurück, dessen Typ mit Array1 übereinstimmt, ohne Duplikate.

Beispiele

In diesem Beispiel `[1, 2, 3]` enthält das erste Array die Elemente 1, 2 und 3. Das zweite Array `[2, 3, 4]` enthält die Elemente 2, 3 und 4. Die `array_except` Funktion entfernt die Elemente 2 und 3 aus dem ersten Array, da sie auch im zweiten Array vorhanden sind. Die resultierende Ausgabe ist das Array`[1]`.

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

In diesem Beispiel `[1, 2, 3]` enthält das erste Array die Elemente 1, 2 und 3. Das zweite Array `[1, 3, 5]` enthält die Elemente 1, 3 und 5. Die `array_except` Funktion entfernt die Elemente 1 und 3 aus dem ersten Array, da sie auch im zweiten Array vorhanden sind. Die resultierende Ausgabe ist das Array`[2]`.

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

Funktion ARRAY_INTERSECT

Die Funktion ARRAY_INTERSECT verwendet zwei Arrays als Argumente und gibt ein neues Array zurück, das die Elemente enthält, die in beiden Eingabearrays vorhanden sind. Diese Funktion ist nützlich, wenn Sie die gemeinsamen Elemente zwischen zwei Arrays suchen müssen. Dies kann in Szenarien hilfreich sein, in denen Sie mengenähnliche Operationen an Arrays ausführen müssen, z. B. um die Schnittmenge zwischen zwei Datensätzen zu ermitteln.

Syntax

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

Argumente

Matrix1

Ein ARRAY beliebigen Typs mit vergleichbaren Elementen.

array2

Ein ARRAY von Elementen, deren Typ mit den Elementen von array1 am wenigsten gemeinsam ist.

Rückgabetyp

Die Funktion ARRAY_INTERSECT gibt ein ARRAY zurück, dessen Typ mit Array1 übereinstimmt, ohne Duplikate und ohne Elemente, die sowohl in array1 als auch array2 enthalten sind.

Beispiele

In diesem Beispiel enthält das erste Array die Elemente 1, 2 und 3. `[1, 2, 3]` Das zweite Array `[1, 3, 5]` enthält die Elemente 1, 3 und 5. Die Funktion ARRAY_INTERSECT identifiziert die gemeinsamen Elemente zwischen den beiden Arrays, nämlich 1 und 3. Das resultierende Ausgabearray ist. `[1, 3]`

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

ARRAY_JOIN-Funktion

Die ARRAY_JOIN-Funktion benötigt zwei Argumente: Das erste Argument ist das Eingabearray, das verknüpft werden soll. Das zweite Argument ist die Trennzeichenfolge, die verwendet wird, um die Array-Elemente zu verketten. Diese Funktion ist nützlich, wenn Sie ein Array von Zeichenketten (oder einen anderen Datentyp) in eine einzelne verkettete Zeichenfolge konvertieren müssen. Dies kann in Szenarien hilfreich sein, in denen Sie ein Array von Werten als einzelne formatierte Zeichenfolge darstellen möchten, z. B. zu Anzeigezwecken oder zur Verwendung bei der weiteren Verarbeitung.

Syntax

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

Argumente

Array

Jeder ARRAY-Typ, aber seine Elemente werden als Zeichenketten interpretiert.

delimiter

Ein STRING, der verwendet wird, um die verketteten Array-Elemente zu trennen.

Ersatz durch Null

Ein STRING, der verwendet wird, um einen NULL-Wert im Ergebnis auszudrücken.

Rückgabetyp

Die Funktion ARRAY_JOIN gibt einen STRING zurück, bei dem die Elemente des Arrays durch ein Trennzeichen getrennt sind und Nullelemente ersetzt werden. `nullReplacement` Wenn nicht angegeben`nullReplacement`, werden Elemente herausgefiltert`null`. Wenn ein Argument ja ist`NULL`, ist das Ergebnis`NULL`.

Beispiele

In diesem Beispiel verwendet die Funktion ARRAY_JOIN das Array `['hello', 'world']` und verbindet die Elemente mithilfe des Trennzeichens `' '` (eines Leerzeichens). Die resultierende Ausgabe ist die Zeichenfolge. `'hello world'`

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

In diesem Beispiel verwendet die ARRAY_JOIN-Funktion das Array `['hello', null, 'world']` und verbindet die Elemente mithilfe des Trennzeichens `' '` (eines Leerzeichens). Der `null` Wert wird durch die angegebene Ersatzzeichenfolge `','` (ein Komma) ersetzt. Die resultierende Ausgabe ist die Zeichenfolge`'hello , world'`.

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

Funktion ARRAY_REMOVE

Die Funktion ARRAY_REMOVE benötigt zwei Argumente: Das erste Argument ist das Eingabearray, aus dem die Elemente entfernt werden. Das zweite Argument ist der Wert, der aus dem Array entfernt wird. Diese Funktion ist nützlich, wenn Sie bestimmte Elemente aus einem Array entfernen müssen. Dies kann in Szenarien hilfreich sein, in denen Sie eine Datenbereinigung oder Vorverarbeitung für ein Array von Werten durchführen müssen.

Syntax

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

Argumente

Array

Ein ARRAY.

element

Ein Ausdruck eines Typs, der den seltensten Typ mit den Elementen eines Arrays gemeinsam hat.

Rückgabetyp

Die Funktion ARRAY_REMOVE gibt den Ergebnistyp zurück, der dem Typ des Arrays entspricht. Wenn das zu entfernende Element ist`NULL`, ist das Ergebnis. `NULL`

Beispiele

In diesem Beispiel verwendet die Funktion ARRAY_REMOVE das Array `[1, 2, 3, null, 3]` und entfernt alle Vorkommen des Werts 3. Die resultierende Ausgabe ist das Array. `[1, 2, null]`

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

ARRAY_UNION-Funktion

Die Funktion ARRAY_UNION verwendet zwei Arrays als Argumente und gibt ein neues Array zurück, das die eindeutigen Elemente aus beiden Eingabearrays enthält. Diese Funktion ist nützlich, wenn Sie zwei Arrays kombinieren und alle doppelten Elemente entfernen müssen. Dies kann in Szenarien hilfreich sein, in denen Sie mengenähnliche Operationen an Arrays ausführen müssen, z. B. um die Verbindung zwischen zwei Datensätzen zu ermitteln.

Syntax

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

Argumente

Matrix1

Ein ARRAY.

Array 2

Ein ARRAY desselben Typs wie array1.

Rückgabetyp

Die Funktion ARRAY_UNION gibt ein ARRAY desselben Typs wie ein Array zurück.

Beispiel

In diesem Beispiel `[1, 2, 3]` enthält das erste Array die Elemente 1, 2 und 3. Das zweite Array `[1, 3, 5]` enthält die Elemente 1, 3 und 5. Die Funktion ARRAY_UNION kombiniert die eindeutigen Elemente aus beiden Arrays, sodass das Ausgabe-Array entsteht. `[1, 2, 3, 5]` T

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

EXPLODE-Funktion

Die EXPLODE-Funktion wird verwendet, um eine einzelne Zeile mit einem Array oder einer Zuordnungsspalte in mehrere Zeilen umzuwandeln, wobei jede Zeile einem einzelnen Element aus dem Array oder der Map entspricht.

Syntax

```
explode(expr)
```

Argumente

expr

Ein Array-Ausdruck oder ein Map-Ausdruck.

Rückgabetyp

Die EXPLODE-Funktion gibt eine Reihe von Zeilen zurück, wobei jede Zeile ein einzelnes Element aus dem Eingabe-Array oder der Eingabe-Map darstellt.

Der Datentyp der Ausgabezeilen hängt vom Datentyp der Elemente im Eingabe-Array oder der Eingabe-Map ab.

Beispiele

Im folgenden Beispiel wird das einzeilige Array [10, 20] in zwei separate Zeilen umgewandelt, die jeweils eines der Array-Elemente (10 und 20) enthalten.

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

Im ersten Beispiel wurde das Eingabe-Array direkt als Argument an übergeben. `explode()` In diesem Beispiel wird das Eingabearray mithilfe der => Syntax angegeben, wobei der Spaltenname (`collection`) explizit angegeben wird.

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

Beide Ansätze sind gültig und führen zu demselben Ergebnis, aber die zweite Syntax kann nützlicher sein, wenn Sie eine Spalte aus einem größeren Datensatz auflösen müssen, als nur ein einfaches Array-Literal.

Funktion FLATTEN

Die FLATTEN-Funktion wird verwendet, um eine verschachtelte Array-Struktur zu einem einzigen flachen Array zu „glätten“.

Syntax

```
flatten(arrayOfArrays)
```

Argumente

arrayOfArrays

Ein Array von Arrays.

Rückgabetyp

Die FLATTEN-Funktion gibt ein Array zurück.

Beispiel

In diesem Beispiel ist die Eingabe ein verschachteltes Array mit zwei inneren Arrays, und die Ausgabe ist ein einzelnes flaches Array, das alle Elemente aus den inneren Arrays enthält. Die

FLATTEN-Funktion verwendet das verschachtelte Array `[[1, 2], [3, 4]]` und kombiniert alle Elemente zu einem einzigen Array. `[1, 2, 3, 4]`

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

Bedingte Ausdrücke

In SQL werden bedingte Ausdrücke verwendet, um Entscheidungen auf der Grundlage bestimmter Bedingungen zu treffen. Sie ermöglichen es Ihnen, den Fluss Ihrer SQL-Anweisungen zu steuern und verschiedene Werte zurückzugeben oder verschiedene Aktionen auszuführen, die auf der Auswertung einer oder mehrerer Bedingungen basieren.

AWS Clean Rooms unterstützt die folgenden bedingten Ausdrücke:

Themen

Der bedingte Ausdruck CASE

Der CASE-Ausdruck ist ein bedingter Ausdruck, der if/then/else Aussagen in anderen Sprachen ähnelt. CASE wird verwendet, um ein Ergebnis anzugeben, wenn es mehrere Bedingungen gibt. Verwenden Sie CASE, wenn ein SQL-Ausdruck gilt, z. B. in einem SELECT-Befehl.

Es gibt zwei Arten von CASE-Ausdrücken: einfach und gesucht.

- In einfachen CASE-Ausdrücken wird ein Ausdruck mit einem Wert verglichen. Wenn keine Übereinstimmung gefunden wird, wird die in der THEN-Klausel angegebene Aktion angewendet.

Wenn keine Übereinstimmung gefunden wird, wird die in der ELSE-Klausel angegebene Aktion angewendet.

- In gesuchten CASE-Ausdrücken wird jeder CASE-Ausdruck auf der Basis eines booleschen Ausdrucks evaluiert und die CASE-Anweisung gibt den ersten übereinstimmenden CASE-Ausdruck zurück. Wenn in den WHEN-Klauseln kein übereinstimmender Ausdruck gefunden wird, wird die Aktion in der ELSE-Klausel zurückgegeben.

Syntax

Einfache CASE-Anweisung, um übereinstimmende Bedingungen zu finden:

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

Gesuchte CASE-Anweisung, um jede Bedingung auszuwerten:

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

Argumente

expression

Ein Spaltenname oder ein gültiger Ausdruck.

Wert

Wert, mit dem der Ausdruck verglichen wird, wie eine numerische Konstante oder eine Zeichenfolge.

Ergebnis

Der Zielwert oder -ausdruck, der zurückgegeben wird, wenn ein Ausdruck oder eine boolesche Bedingung ausgewertet werden. Die Datentypen aller Ergebnisausdrücke müssen in einen einzigen Ausgabetyp konvertierbar sein.

condition

Ein boolescher Ausdruck, der mit true oder false ausgewertet wird. Wenn die Bedingung mit true ausgewertet wird, ist der Wert des CASE-Ausdrucks das Ergebnis, das auf die Bedingung folgt, und der Rest des CASE-Ausdrucks wird nicht verarbeitet. Wenn die Bedingung mit false ausgewertet wird, werden alle nachfolgenden WHEN-Klauseln ausgewertet. Wenn keine Ergebnisse der WHEN-Bedingung mit true ausgewertet werden, ist der Wert des CASE-Ausdrucks das Ergebnis der ELSE-Klausel. Wenn die ELSE-Klausel ausgelassen wurde und keine Bedingung mit true ausgewertet wird, ist das Ergebnis null.

Beispiele

Verwenden Sie einen einfachen CASE-Ausdruck, um `New York City` durch `Big Apple` in einer für die Tabelle VENUE ausgeführten Abfrage zu ersetzen. Alle anderen Städtenamen werden durch ersetzt `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
...
```

Verwendet einen gesuchten CASE-Ausdruck, um Gruppennummern basierend auf dem PRICEPAID-Wert für einzelne Ticketverkäufe zuzuweisen:

```
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
...
```

COALESCEAusdruck

Ein COALESCE Ausdruck gibt den Wert des ersten Ausdrucks in der Liste zurück, der nicht Null ist. Wenn alle Ausdrücke null sind, ist das Ergebnis null. Wenn ein Nicht-Null-Wert gefunden wird, werden die verbleibenden Ausdrücke in der Liste nicht ausgewertet.

Diese Art von Ausdruck ist nützlich, wenn Sie einen Sicherungswert für etwas zurückgeben möchten, wenn der bevorzugte Wert fehlt oder null ist. Beispielsweise kann eine Abfrage eine von drei Telefonnummern zurückgeben (mobil, Festnetz oder beruflich; in dieser Reihenfolge), je nachdem, welche Telefonnummer in der Tabelle zuerst gefunden wird (nicht null).

Syntax

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

Beispiele

Wendet den COALESCE Ausdruck auf zwei Spalten an.

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

Der Standardspaltenname für einen NVL-Ausdruck lautetCOALESCE. Die folgende Abfrage gibt dieselben Ergebnisse zurück.

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

GRÖSSTER und KLEINSTER Ausdruck

Gibt den größten oder kleinsten Wert aus einer Liste einer beliebigen Zahl von Ausdrücken zurück.

Syntax

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

Parameter

expression_list

Eine durch Komma getrennte Liste von Ausdrücken, wie beispielsweise Spaltennamen. Die Ausdrücke müssen alle in einen gemeinsamen Datentyp konvertierbar sein. NULL-Werte in der Liste werden ignoriert. Wenn alle Ausdrücke zu NULL ausgewertet werden, ist das Ergebnis NULL.

Rückgabewert

Gibt den größten Wert (bei GREATEST) oder den kleinsten Wert (bei LEAST) aus der angegebenen Liste von Ausdrücken zurück.

Beispiel

Im folgenden Beispiel wird der höchste Wert alphabetisch für `firstname` oder `lastname` zurückgegeben.

```
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)
```

IF-Ausdruck

Die Bedingungsfunktion IF gibt einen von zwei Werten zurück, die auf einer Bedingung basieren.

Bei dieser Funktion handelt es sich um eine gängige Kontrollflussanweisung, die in SQL verwendet wird, um Entscheidungen zu treffen und auf der Grundlage der Auswertung einer Bedingung unterschiedliche Werte zurückzugeben. Sie ist nützlich, um einfache If-Else-Logik in einer Abfrage zu implementieren.

Syntax

```
if(expr1, expr2, expr3)
```

Argumente

expr1

Die Bedingung oder der Ausdruck, der ausgewertet wird. Ist dies der Fall`true`, gibt die Funktion den Wert von expr2 zurück. Wenn expr1 gleich ist*`false`*, gibt die Funktion den Wert von expr3 zurück.

Ausdruck2

Der Ausdruck, der ausgewertet und zurückgegeben wird, wenn expr1 `true`

Ausdruck3

Der Ausdruck, der ausgewertet und zurückgegeben wird, wenn expr1 `false`

Rückgabewert

Wenn als Ergebnis `expr1` ausgewertet wird`true`, kehrt es zurück`expr2`; andernfalls kehrt es zurück. `expr3`

Beispiel

Im folgenden Beispiel wird die `if()` Funktion verwendet, um basierend auf einer Bedingung einen von zwei Werten zurückzugeben. Die ausgewertete Bedingung ist`1 < 2`, was bedeutet`true`, dass der erste Wert zurückgegeben `'a'` wird.

```
SELECT if(1 < 2, 'a', 'b');
 a]
```

IS_NULL-Ausdruck

Der IS_NULL bedingte Ausdruck wird verwendet, um zu überprüfen, ob ein Wert Null ist.

Dieser Ausdruck ist ein Synonym für. IS NULL

Syntax

```
is_null(expr)
```

Argumente

expr

Ein Ausdruck beliebigen Typs.

Rückgabewert

Der IS_NULL bedingte Ausdruck gibt einen booleschen Wert zurück. Wenn NULL `expr1` ist, wird zurückgegeben`true`, andernfalls wird zurückgegeben. `false`

Beispiele

Das folgende Beispiel prüft, ob der Wert Null 1 ist, und gibt das boolesche Ergebnis zurück, `true` da 1 ein gültiger Wert ungleich Null ist.

```
SELECT is not null(1);
 true
```

Im folgenden Beispiel wird die `id` Spalte aus der `squirrels` Tabelle ausgewählt, jedoch nur für die Zeilen, in denen sich die Altersspalte befindet. `null`

```
SELECT id FROM squirrels WHERE is_null(age)
```

IS_NOT_NULL-Ausdruck

Der IS_NOT_NULL bedingte Ausdruck wird verwendet, um zu überprüfen, ob ein Wert nicht Null ist.

Dieser Ausdruck ist ein Synonym für. IS NOT NULL

Syntax

```
is_not_null(expr)
```

Argumente

expr

Ein Ausdruck beliebigen Typs.

Rückgabewert

Der IS_NOT_NULL bedingte Ausdruck gibt einen booleschen Wert zurück. Wenn nicht NULL `expr1` ist, wird zurückgegeben`true`, andernfalls zurückgegeben. `false`

Beispiele

Das folgende Beispiel prüft, ob der Wert nicht Null 1 ist, und gibt das boolesche Ergebnis zurück, `true` da 1 ein gültiger Wert ungleich Null ist.

```
SELECT is not null(1);
 true
```

Im folgenden Beispiel wird die `id` Spalte aus der `squirrels` Tabelle ausgewählt, jedoch nur für die Zeilen, in denen sich die Altersspalte nicht befindet. `null`

```
SELECT id FROM squirrels WHERE is_not_null(age)
```

NVL- und COALESCE-Funktionen

Gibt den Wert des ersten Ausdrucks in einer Reihe von Ausdrücken zurück, der nicht null ist. Wenn ein Nicht-Null-Wert gefunden wird, werden die verbleibenden Ausdrücke in der Liste nicht ausgewertet.

NVL ist identisch mit COALESCE. Es sind Synonyme. Unter diesem Thema finden Sie eine Erläuterung der Syntax sowie Beispiele für beide.

Syntax

```
NVL( expression, expression, ... )
```

Die Syntax für COALESCE ist identisch:

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

Wenn alle Ausdrücke null sind, ist das Ergebnis null.

Diese Funktionen sind hilfreich, wenn Sie einen Sekundärwert zurückgeben möchten, falls ein Primärwert fehlt oder null ist. Eine Abfrage könnte beispielsweise die erste von drei verfügbaren Telefonnummern zurückgeben: Mobiltelefonnummer, private oder geschäftliche Telefonnummer. Die Reihenfolge der Ausdrücke in der Funktion bestimmt die Reihenfolge der Auswertung.

Argumente

expression

Ein Ausdruck (beispielsweise ein Spaltenname), der hinsichtlich des Null-Status ausgewertet werden soll.

Rückgabetyp

AWS Clean Rooms bestimmt den Datentyp des zurückgegebenen Werts auf der Grundlage der Eingabeausdrücke. Wenn die Datentypen der Eingabeausdrücke keinen gemeinsamen Typ haben, wird ein Fehler zurückgegeben.

Beispiele

Wenn die Liste Ausdrücke mit Ganzzahlen enthält, gibt die Funktion eine Ganzzahl zurück.

```
SELECT COALESCE(NULL, 12, NULL);

coalesce
--------------
12
```

Dieses Beispiel, das im Gegensatz zum vorherigen Beispiel NVL verwendet, gibt dasselbe Ergebnis zurück.

```
SELECT NVL(NULL, 12, NULL);

coalesce
--------------
12
```

Im folgenden Beispiel wird einen Zeichenfolgetyp zurückgegeben.

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', NULL);

coalesce
--------------
AWS Clean Rooms
```

Das folgende Beispiel führt zu einem Fehler, da die Datentypen in der Ausdrucksliste unterschiedlich sind. In diesem Fall enthält die Liste sowohl einen Zeichenfolgetyp als auch einen Zahlentyp.

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', 12);
ERROR: invalid input syntax for integer: "AWS Clean Rooms"
```

NVL2 Funktion

Gibt einen von zwei Werten aus, je nachdem, ob ein angegebener Ausdruck zu NULL oder zu NOT NULL aufgelöst wird.

Syntax

```
NVL2 ( expression, not_null_return_value, null_return_value )
```

Argumente

expression

Ein Ausdruck (beispielsweise ein Spaltenname), der hinsichtlich des Null-Status ausgewertet werden soll.

not_null_return_value

Der Wert, der zurückgegeben wird, wenn expression zu NOT NULL ausgewertet wird. Der Wert not_null_return_value muss entweder denselben Datentyp wie expression haben oder implizit in diesen Datentyp konvertiert werden können.

null_return_value

Der Wert, der zurückgegeben wird, wenn expression zu NULL ausgewertet wird. Der Wert null_return_value muss entweder denselben Datentyp wie expression haben oder implizit in diesen Datentyp konvertiert werden können.

Rückgabetyp

Der NVL2 Rückgabetyp wird wie folgt bestimmt:

- Wenn not_null_return_value oder null_return_value null ist, wird der Datentyp des Nicht-Null-Ausdrucks zurückgegeben.

Wenn sowohl not_null_return_value als auch null_return_value nicht null sind:

- Wenn not_null_return_value und null_return_value denselben Datentyp haben, wird dieser Datentyp zurückgegeben.
- Wenn not_null_return_value und null_return_value unterschiedliche numerische Datentypen haben, wird der kleinste kompatible numerische Datentyp zurückgegeben.
- Wenn not_null_return_value und null_return_value unterschiedliche Datum-/Uhrzeit-Datentypen haben, wird ein Zeitstempeldatentyp zurückgegeben.
- Wenn not_null_return_value und null_return_value unterschiedliche Zeichendatentypen haben, wird der Datentyp von not_null_return_value zurückgegeben.
- Wenn not_null_return_value und null_return_value gemischte numerische und nicht numerische Datentypen haben, wird der Datentyp von not_null_return_value zurückgegeben.

> **Important**
>
> In den letzten beiden Fällen, in denen der Datentyp von not_null_return_value zurückgegeben wird, wird null_return_value implizit in diesen Datentyp umgewandelt. Wenn die Datentypen nicht kompatibel sind, schlägt die Funktion fehl.

Nutzungshinweise

Denn NVL2 die Rückgabe hat entweder den Wert des Parameters not_null_return_value oder null_return_value, je nachdem, welcher Wert von der Funktion ausgewählt wurde, hat aber den Datentyp not_null_return_value.

Wenn beispielsweise column1 NULL ist, geben die folgenden Abfragen denselben Wert zurück. Der DECODE-Rückgabewert-Datentyp ist jedoch NVL2 INTEGER und der Rückgabewert-Datentyp VARCHAR.

```
select decode(column1, null, 1234, '2345');
select nvl2(column1, '2345', 1234);
```

Beispiel

Im folgenden Beispiel werden einige Beispieldaten modifiziert und anschließend zwei Felder ausgewertet, um die richtigen Kontaktinformationen für Benutzer bereitzustellen:

```
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
```

NULLIF-Funktion

Vergleicht zwei Argumente und gibt null zurück, wenn die Argumente gleich sind. Wenn sie nicht gleich sind, wird das erste Argument zurückgegeben.

Syntax

Der NULLIF-Ausdruck vergleicht zwei Argumente und gibt null zurück, wenn die Argumente gleich sind. Wenn sie nicht gleich sind, wird das erste Argument zurückgegeben. Dieser Ausdruck ist die Umkehrung des NVL- oder COALESCE-Ausdrucks.

```
NULLIF ( expression1, expression2 )
```

Argumente

expression1, expression2

Die Zielspalten oder -ausdrücke, die verglichen werden. Der Rückgabetyp ist mit dem Typ des ersten Ausdrucks identisch.

Beispiele

Im folgenden Beispiel gibt die Abfrage die Zeichenfolge `first` zurück, da die Argumente nicht identisch sind.

```
SELECT NULLIF('first', 'second');

case
-------
first
```

Im folgenden Beispiel gibt die Abfrage NULL zurück, da die Argumente des Zeichenfolgeliterals identisch sind.

```
SELECT NULLIF('first', 'first');

case
-------
```

```
NULL
```

Im folgenden Beispiel gibt die Abfrage 1 zurück, da die Ganzzahlargumente nicht identisch sind.

```
SELECT NULLIF(1, 2);

case
-------
1
```

Im folgenden Beispiel gibt die Abfrage NULL zurück, da die Ganzzahlargumente identisch sind.

```
SELECT NULLIF(1, 1);

case
-------
NULL
```

Im folgenden Beispiel gibt die Abfrage null zurück, wenn die LISTID- und SALESID-Werte übereinstimmen:

```
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)
```

Konstruktor-Funktionen

Eine SQL-Konstruktorfunktion ist eine Funktion, die verwendet wird, um neue Datenstrukturen wie Arrays oder Maps zu erstellen.

Sie nehmen einige Eingabewerte und geben ein neues Datenstrukturobjekt zurück. Konstruktorfunktionen werden normalerweise nach dem Datentyp benannt, den sie erstellen, z. B. ARRAY oder MAP.

Konstruktorfunktionen unterscheiden sich von Skalarfunktionen oder Aggregatfunktionen, die mit vorhandenen Daten arbeiten und einen einzelnen Wert zurückgeben. Konstruktorfunktionen werden verwendet, um neue Datenstrukturen zu erstellen, die dann für die weitere Datenverarbeitung oder Analyse verwendet werden können.

AWS Clean Rooms unterstützt die folgenden Konstruktorfunktionen:

Themen

MAP-Konstruktorfunktion

Die MAP-Konstruktorfunktion erstellt eine Map mit den angegebenen Schlüssel/Wert-Paaren.

Konstruktorfunktionen wie MAP sind nützlich, wenn Sie neue Datenstrukturen programmgesteuert in Ihren SQL-Abfragen erstellen müssen. Sie ermöglichen es Ihnen, komplexe Datenstrukturen zu erstellen, die für die weitere Datenverarbeitung oder Analyse verwendet werden können.

Syntax

```
map(key0, value0, key1, value1, ...)
```

Argumente

Schlüssel 0

Ein Ausdruck eines vergleichbaren Typs. Alle key0 müssen einen am wenigsten gemeinsamen Typ haben.

Wert0

Ein Ausdruck beliebigen Typs. Alle ValueN müssen einen Typ haben, der am wenigsten gemeinsam ist.

Rückgabewert

Die MAP-Funktion gibt ein MAP zurück, bei dem die Schlüssel als der seltenste Typ von key0 und die Werte als der seltenste Typ von value0 eingegeben wurden.

Beispiele

Im folgenden Beispiel wird eine neue Map mit zwei Schlüssel-Wert-Paaren erstellt: Der Schlüssel ist mit dem Wert verknüpft. `1.0 '2'` Der Schlüssel `3.0` ist mit dem Wert verknüpft. `'4'` Die resultierende Map wird dann als Ausgabe der SQL-Anweisung zurückgegeben.

```
SELECT map(1.0, '2', 3.0, '4');
 {1.0:"2",3.0:"4"}
```

Konstruktorfunktion NAMED_STRUCT

Die Konstruktorfunktion NAMED_STRUCT erstellt eine Struktur mit den angegebenen Feldnamen und Werten.

Konstruktorfunktionen wie NAMED_STRUCT sind nützlich, wenn Sie neue Datenstrukturen programmgesteuert in Ihren SQL-Abfragen erstellen müssen. Sie ermöglichen es Ihnen, komplexe Datenstrukturen wie Strukturen oder Datensätze zu erstellen, die für die weitere Datenverarbeitung oder Analyse verwendet werden können.

Syntax

```
named_struct(name1, val1, name2, val2, ...)
```

Argumente

Name1

Ein STRING-literales Benennungsfeld 1.

Wert 1

Ein Ausdruck beliebigen Typs, der den Wert für Feld 1 angibt.

Rückgabewert

Die Funktion NAMED_STRUCT gibt eine Struktur zurück, bei der Feld 1 dem Typ von Val1 entspricht.

Beispiele

Im folgenden Beispiel wird eine neue Struktur mit drei benannten Feldern erstellt: Dem Feld "a" wird der Wert zugewiesen. 1 Dem Feld "b" wird der Wert zugewiesen. 2. Dem Feld "c" wird der Wert zugewiesen3. Die resultierende Struktur wird dann als Ausgabe der SQL-Anweisung zurückgegeben.

```
SELECT named_struct("a", 1, "b", 2, "c", 3);
 {"a":1,"b":2,"c":3}
```

STRUCT-Konstruktorfunktion

Die STRUCT-Konstruktorfunktion erstellt eine Struktur mit den angegebenen Feldwerten.

Konstruktorfunktionen wie STRUCT sind nützlich, wenn Sie neue Datenstrukturen programmgesteuert in Ihren SQL-Abfragen erstellen müssen. Sie ermöglichen es Ihnen, komplexe Datenstrukturen wie Strukturen oder Datensätze zu erstellen, die für die weitere Datenverarbeitung oder -analyse verwendet werden können.

Syntax

```
struct(col1, col2, col3, ...)
```

Argumente

Spalte 1

Ein Spaltenname oder ein gültiger Ausdruck.

Rückgabewert

Die STRUCT-Funktion gibt eine Struktur zurück, bei der Feld1 dem Typ von expr1 entspricht.

Wenn es sich bei den Argumenten um benannte Referenzen handelt, werden die Namen zur Benennung des Felds verwendet. Andernfalls erhalten die Felder den Namen colN, wobei N die Position des Feldes in der Struktur ist.

Beispiele

Das folgende Beispiel erstellt eine neue Struktur mit drei Feldern: Dem ersten Feld wird der Wert 1 zugewiesen. Dem zweiten Feld wird der Wert 2 zugewiesen. Dem dritten Feld wird der Wert 3

zugewiesen. Standardmäßig werden die Felder in der resultierenden Struktur auf der Grundlage ihrer Position in der Argumentliste mit`col3`, und benannt`col1`. `col2` Die resultierende Struktur wird dann als Ausgabe der SQL-Anweisung zurückgegeben.

```
SELECT struct(1, 2, 3);
 {"col1":1,"col2":2,"col3":3}
```

Funktionen für die Datentypformatierung

Mithilfe einer Funktion zur Formatierung von Datentypen können Sie Werte von einem Datentyp in einen anderen konvertieren. Bei jeder dieser Funktionen ist das erste Argument immer der zu formatierende Wert, und das zweite Argument enthält die Vorlage für das neue Format.

AWS Clean Rooms Spark SQL unterstützt mehrere Funktionen zur Formatierung von Datentypen.

Themen

- BASE64 Funktion
- CAST-Funktion
- DECODE-Funktion
- ENCODE-Funktion
- HEX-Funktion
- STR_TO_MAP-Funktion
- TO_CHAR
- TO_DATE-Funktion
- TO_NUMBER
- UNBASE64 Funktion
- UNHEX-Funktion
- Datum-/Uhrzeit-Formatzeichenfolgen
- Numerische Formatzeichenfolgen

BASE64 Funktion

Die BASE64 Funktion konvertiert einen Ausdruck mithilfe der RFC2045 Base64-Übertragungskodierung für MIME in eine Base64-Zeichenfolge.

Syntax

```
base64(expr)
```

Argumente

expr

Ein BINARY-Ausdruck oder ein STRING, den die Funktion als BINARY interpretiert.

Rückgabetyp

`STRING`

Beispiel

Verwenden Sie das folgende Beispiel, um die angegebene Zeichenketteneingabe in ihre Base64-kodierte Darstellung zu konvertieren. Das Ergebnis ist die Base64-kodierte Darstellung der Eingabezeichenfolge 'Spark SQL', die 'u3bhcmSgU1fm' ist.

```
SELECT base64('Spark SQL');
 U3BhcmsgU1FM
```

CAST-Funktion

Die CAST-Funktion konvertiert einen Datentyp in einen anderen kompatiblen Datentyp. Sie können beispielsweise eine Zeichenfolge in ein Datum oder einen numerischen Typ in eine Zeichenfolge konvertieren. CAST führt eine Laufzeitkonvertierung durch, was bedeutet, dass die Konvertierung den Datentyp eines Werts in einer Quelltabelle nicht ändert. Dieser wird nur im Kontext der Abfrage geändert.

Bestimmte Datentypen erfordern eine explizite Konvertierung in andere Datentypen mithilfe der CAST-Funktion. Andere Datentypen können implizit als Teil eines anderen Befehls konvertiert werden, ohne dass CAST verwendet wird. Siehe Kompatibilität von Typen und Umwandlung zwischen Typen.

Syntax

Verwenden Sie eine dieser beiden gleichwertigen Syntaxformate, um Ausdrücke von einem Datentyp in einen anderen umzuwandeln.

```
CAST ( expression AS type )
```

Argumente

expression

Ein Ausdruck, der einen oder mehrere Werte auswertet, beispielsweise ein Spaltenname oder ein Literal. Die Konvertierung von Null-Werten gibt Null-Werte zurück. Der Ausdruck darf keine leeren oder leeren Zeichenfolgen enthalten.

Typ

Einer der unterstützten DatentypenDatentypen, mit Ausnahme der Datentypen BINARY und BINARY VARYING.

Rückgabetyp

CAST gibt den Datentyp zurück, der durch das Argument type angegeben ist.

> **Note**
>
> AWS Clean Rooms gibt einen Fehler zurück, wenn Sie versuchen, eine problematische Konvertierung durchzuführen, z. B. eine DECIMAL-Konvertierung, die an Genauigkeit verliert, wie die folgende:
>
> ```
> select 123.456::decimal(2,1);
> ```
>
> oder eine INTEGER-Konvertierung, die einen Overflow verursacht:
>
> ```
> select 12345678::smallint;
> ```

Beispiele

Die folgenden beiden Abfragen sind gleichwertig. Beide wandeln einen Dezimalwert in eine Ganzzahl um:

```
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)
```

Das Folgende führt zu einem ähnlichen Ergebnis. Für die Ausführung sind keine Beispieldaten erforderlich:

```
select cast(162.00 as integer) as pricepaid;

pricepaid
-----------
162
(1 row)
```

In diesem Beispiel werden die Werte in einer Zeitstempelspalte in Datumsangaben umgewandelt, was dazu führt, dass die Uhrzeit aus jedem Ergebnis entfernt wird:

```
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)
```

Wenn Sie CAST nicht wie im vorherigen Beispiel dargestellt verwendet haben, würden die Ergebnisse die Uhrzeit umfassen: 2008-02-18 02:36:48.

Die folgende Abfrage wandelt variable Zeichendaten in ein Datum um. Für die Ausführung sind keine Beispieldaten erforderlich.

```
select cast('2008-02-18 02:36:48' as date) as mysaletime;

mysaletime
--------------------
2008-02-18
(1 row)
```

In diesem Beispiel werden die Werte in einer Datumsspalte in Zeitstempel umgewandelt:

```
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 einem Fall wie dem vorherigen Beispiel können Sie zusätzliche Kontrolle über die Ausgabeformatierung erlangen, indem Sie TO_CHAR

In diesem Beispiel wird eine Ganzzahl in eine Zeichenfolge umgewandelt:

```
select cast(2008 as char(4));
```

```
bpchar
--------
2008
```

In diesem Beispiel wird ein DECIMAL(6,3)-Wert in einen DECIMAL(4,1)-Wert umgewandelt:

```
select cast(109.652 as decimal(4,1));

numeric
---------
109.7
```

Dieses Beispiel zeigt einen komplexeren Ausdruck. Die Spalte PRICEPAID (eine DECIMAL(8,2)-Spalte) in der Tabelle SALES wird in eine DECIMAL(38,2)-Spalte umgewandelt und die Werte werden mit 100000000000000000000 multipliziert:

```
select salesid, pricepaid::decimal(38,2)*100000000000000000000
as value from sales where salesid<10 order by salesid;


 salesid |           value
---------+----------------------------
       1 | 72800000000000000000000.00
       2 |  76000000000000000000000.00
       3 | 35000000000000000000000.00
       4 | 17500000000000000000000.00
       5 | 15400000000000000000000.00
       6 | 39400000000000000000000.00
       7 | 78800000000000000000000.00
       8 | 19700000000000000000000.00
       9 | 59100000000000000000000.00

(9 rows)
```

DECODE-Funktion

Die DECODE-Funktion ist das Gegenstück zur ENCODE-Funktion, mit der eine Zeichenfolge mithilfe einer bestimmten Zeichenkodierung in ein Binärformat konvertiert wird. Die DECODE-Funktion nimmt die Binärdaten und konvertiert sie unter Verwendung der angegebenen Zeichenkodierung wieder in ein lesbares Zeichenkettenformat.

Diese Funktion ist nützlich, wenn Sie mit in einer Datenbank gespeicherten Binärdaten arbeiten und diese in einem für Menschen lesbaren Format präsentieren müssen oder wenn Sie Daten zwischen verschiedenen Zeichenkodierungen konvertieren müssen.

Syntax

```
decode(expr, charset)
```

Argumente

expr

Ein BINÄRER Ausdruck, der im Zeichensatz codiert ist.

Zeichensatz

Ein STRING-Ausdruck.

Unterstützte Zeichensatzkodierungen (ohne Berücksichtigung von Groß- und Kleinschreibung): `'US-ASCII'`,,`'ISO-8859-1'`, `'UTF-8''UTF-16BE'`, `'UTF-16LE'` und. `'UTF-16'`

Rückgabetyp

Die DECODE-Funktion gibt einen STRING zurück.

Beispiel

Im folgenden Beispiel wird eine Tabelle `messages` mit einer Spalte namens aufgerufen`message_text`, in der Nachrichtendaten in einem Binärformat unter Verwendung der UTF-8-Zeichenkodierung gespeichert werden. Die DECODE-Funktion konvertiert die Binärdaten zurück in ein lesbares Zeichenkettenformat. Die Ausgabe dieser Abfrage ist der lesbare Text der in der Nachrichtentabelle gespeicherten Nachricht mit der ID`123`, der unter Verwendung der `'utf-8'` Kodierung vom Binärformat in eine Zeichenfolge umgewandelt wurde.

```
SELECT decode(message_text, 'utf-8') AS message
FROM messages
WHERE message_id = 123;
```

ENCODE-Funktion

Die ENCODE-Funktion wird verwendet, um eine Zeichenfolge unter Verwendung einer bestimmten Zeichenkodierung in ihre binäre Darstellung zu konvertieren.

Diese Funktion ist nützlich, wenn Sie mit Binärdaten arbeiten oder wenn Sie zwischen verschiedenen Zeichenkodierungen konvertieren müssen. Sie können die ENCODE-Funktion beispielsweise verwenden, wenn Sie Daten in einer Datenbank speichern, die Binärspeicher benötigt, oder wenn Sie Daten zwischen Systemen übertragen müssen, die unterschiedliche Zeichenkodierungen verwenden.

Syntax

```
encode(str, charset)
```

Argumente

str

Ein STRING-Ausdruck, der codiert werden soll.

Zeichensatz

Ein STRING-Ausdruck, der die Kodierung angibt.

Unterstützte Zeichensatzkodierungen (ohne Berücksichtigung von Groß- und Kleinschreibung):`'US-ASCII'`,,`'ISO-8859-1'`, `'UTF-8''UTF-16BE'`, `'UTF-16LE'` und. `'UTF-16'`

Rückgabetyp

Die ENCODE-Funktion gibt einen BINÄRWERT zurück.

Beispiel

Im folgenden Beispiel wird die Zeichenfolge `'abc'` mithilfe der `'utf-8'` Kodierung in ihre binäre Darstellung konvertiert, was in diesem Fall dazu führt, dass die ursprüngliche Zeichenfolge zurückgegeben wird. Das liegt daran, dass es sich bei der `'utf-8'` Kodierung um eine Zeichenkodierung mit variabler Breite handelt, die den gesamten ASCII-Zeichensatz (einschließlich der Buchstaben `'a''b'`, und`'c'`) mit einem einzigen Byte pro Zeichen darstellen kann. Daher entspricht die binäre Darstellung von `'abc'` using `'utf-8'` der ursprünglichen Zeichenfolge.

```
SELECT encode('abc', 'utf-8');
```

```
 abc
```

HEX-Funktion

Die HEX-Funktion konvertiert einen numerischen Wert (entweder eine Ganzzahl oder eine Gleitkommazahl) in die entsprechende hexadezimale Zeichenkettendarstellung.

Hexadezimal ist ein Zahlensystem, das 16 verschiedene Symbole (0-9 und A-F) verwendet, um numerische Werte darzustellen. Es wird häufig in der Informatik und Programmierung verwendet, um Binärdaten in einem kompakteren und für Menschen lesbaren Format darzustellen.

Syntax

```
hex(expr)
```

Argumente

expr

Ein BIGINT-, BINARY- oder STRING-Ausdruck.

Rückgabetyp

HEX gibt einen STRING zurück. Die Funktion gibt die hexadezimale Darstellung des Arguments zurück.

Beispiel

Im folgenden Beispiel wird der Integer-Wert 17 als Eingabe verwendet und die Funktion HEX () darauf angewendet. Die Ausgabe ist11, was die hexadezimale Darstellung des Eingabewerts ist. 17

```
SELECT hex(17);
 11
```

Im folgenden Beispiel wird die Zeichenfolge in ihre `'Spark_SQL'` hexadezimale Darstellung konvertiert. Die Ausgabe ist`537061726B2053514C`, das ist die hexadezimale Darstellung der Eingabezeichenfolge. `'Spark_SQL'`

```
SELECT hex('Spark_SQL');
 537061726B2053514C
```

In diesem Beispiel wird die Zeichenfolge 'Spark_SQL' wie folgt konvertiert:

- 'S' -> 53
- 'p' -> 70
- 'a' -> 61
- 'r' -> 72 '
- 'k' -> 6 B
- '_' -> 20
- 'S' -> 53
- 'Q' -> 51
- 'L' -> 4C

Die Verkettung dieser Hexadezimalwerte ergibt die endgültige Ausgabe ". `537061726B2053514C`"

STR_TO_MAP-Funktion

Die STR_TO_MAP-Funktion ist eine Konvertierungsfunktion. string-to-map Sie konvertiert eine Zeichenkettendarstellung einer Karte (oder eines Wörterbuchs) in eine tatsächliche Kartendatenstruktur.

Diese Funktion ist nützlich, wenn Sie mit Kartendatenstrukturen in SQL arbeiten müssen, die Daten jedoch zunächst als Zeichenfolge gespeichert werden. Indem Sie die Zeichenkettendarstellung in eine tatsächliche Map konvertieren, können Sie dann Operationen und Manipulationen an den Kartendaten durchführen.

Syntax

```
str_to_map(text[, pairDelim[, keyValueDelim]])
```

Argumente

Text

Ein STRING-Ausdruck, der die Map darstellt.

PairDelim

Ein optionales STRING-Literal, das angibt, wie Einträge getrennt werden sollen. Es ist standardmäßig ein Komma (). `','`

keyValueDelim

Ein optionales STRING-Literal, das angibt, wie jedes Schlüssel-Wert-Paar getrennt werden soll. Standardmäßig wird ein Doppelpunkt () verwendet. `':'`

Rückgabetyp

Die STR_TO_MAP-Funktion gibt sowohl für Schlüssel als auch für Werte einen MAP-Wert vom Typ STRING zurück. Sowohl PairDelim als auch werden als reguläre Ausdrücke behandelt. keyValueDelim

Beispiel

Das folgende Beispiel verwendet die Eingabezeichenfolge und die beiden Trennzeichenargumente und konvertiert die Zeichenfolgendarstellung in eine tatsächliche Kartendatenstruktur. In diesem speziellen Beispiel `'a:1,b:2,c:3'` stellt die Eingabezeichenfolge eine Map mit den folgenden Schlüssel-Wert-Paaren dar: `'a'` ist der Schlüssel und `'1'` ist der Wert. `'b'`ist der Schlüssel und `'2'` ist der Wert. `'c'`ist der Schlüssel und `'3'` ist der Wert. Das `','` Trennzeichen wird verwendet, um die Schlüssel-Wert-Paare zu trennen, und das `':'` Trennzeichen wird verwendet, um den Schlüssel und den Wert innerhalb jedes Paares zu trennen. Die Ausgabe dieser Abfrage ist:. `{"a":"1","b":"2","c":"3"}` Dies ist die resultierende Kartendatenstruktur, in der die Schlüssel `'a' 'b''c'`, und und und die entsprechenden Werte sind `'1''2'`, und`'3'`.

```
SELECT str_to_map('a:1,b:2,c:3', ',', ':');
 {"a":"1","b":"2","c":"3"}
```

Das folgende Beispiel zeigt, dass die STR_TO_MAP-Funktion erwartet, dass die Eingabezeichenfolge ein bestimmtes Format hat, wobei die Schlüssel-Wert-Paare korrekt abgegrenzt sind. Wenn die Eingabezeichenfolge nicht dem erwarteten Format entspricht, versucht die Funktion trotzdem, eine Map zu erstellen, aber die resultierenden Werte entsprechen möglicherweise nicht den Erwartungen.

```
SELECT str_to_map('a');
 {"a":null}
```

TO_CHAR

TO_CHAR konvertiert einen Zeitstempel oder numerischen Ausdruck in ein Zeichenfolgendatenformat.

Syntax

```
TO_CHAR (timestamp_expression | numeric_expression , 'format')
```

Argumente

timestamp_expression

Ein Ausdruck, der einen TIMESTAMP- oder TIMESTAMPTZ-Typwert als Ergebnis hat oder einen Wert, der implizit zu einem Zeitstempel gezwungen werden kann.

numeric_expression

Ein Ausdruck, der einen numerischen Datentypwert als Ergebnis hat oder einen Wert, der implizit zu einem numerischen Typ gezwungen werden kann. Weitere Informationen finden Sie unter Numerische Typen. „TO_CHAR“ fügt links von der Zahlenfolge ein Leerzeichen ein.

> **Note**
>
> TO_CHAR unterstützt keine 128-Bit-DEZIMALWERTE.

format

Das Format für den neuen Wert. Informationen zu gültigen Formaten finden Sie unter Datum-/Uhrzeit-Formatzeichenfolgen und Numerische Formatzeichenfolgen.

Rückgabetyp

VARCHAR

Beispiele

Im folgenden Beispiel wird ein Zeitstempel in einen Wert mit Datum und Uhrzeit konvertiert, dessen Format den Namen des Monats auf neun Zeichen aufgefüllt, den Namen des Wochentages und die Tagesnummer des Monats enthält.

```
select to_char(timestamp '2009-12-31 23:15:59', 'MONTH-DY-DD-YYYY HH12:MIPM');
to_char
-------------------------
DECEMBER -THU-31-2009 11:15PM
```

Im folgenden Beispiel wird ein Zeitstempel in einen Wert mit Tageszahl des Jahres konvertiert.

```
select to_char(timestamp '2009-12-31 23:15:59', 'DDD');

to_char
-------------------------
365
```

Im folgenden Beispiel wird ein Zeitstempel in einen Wert mit ISO-Tageszahl der Woche konvertiert.

```
select to_char(timestamp '2022-05-16 23:15:59', 'ID');

to_char
-------------------------
1
```

Im folgenden Beispiel wird der Monat aus einem Datumswert extrahiert.

```
select to_char(date '2009-12-31', 'MONTH');

to_char
-------------------------
DECEMBER
```

Im folgenden Beispiel wird jeder STARTTIME-Wert in der Tabelle EVENT in eine Zeichenfolge konvertiert, die aus Stunden, Minuten und Sekunden besteht.

```
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)
```

Im folgenden Beispiel wird ein ganzer Zeitstempelwert in ein anderes Format konvertiert.

```
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)
```

Im folgenden Beispiel wird ein Zeitstempelliteral in eine Zeichenfolge konvertiert.

```
select to_char(timestamp '2009-12-31 23:15:59','HH24:MI:SS');
to_char
----------
23:15:59
(1 row)
```

Im folgenden Beispiel wird eine Zahl in eine Zeichenfolge mit dem Minuszeichen am Ende konvertiert.

```
select to_char(-125.8, '999D99S');
to_char
---------
125.80-
(1 row)
```

Im folgenden Beispiel wird eine Zahl in eine Zeichenfolge mit dem Währungssymbol konvertiert.

```
select to_char(-125.88, '$S999D99');
to_char
---------
$-125.88
(1 row)
```

Im folgenden Beispiel wird eine Zahl in eine Zeichenfolge konvertiert, bei dem Eckige Klammern als negative Zahlen verwendet werden.

```
select to_char(-125.88, '$999D99PR');
to_char
---------
$<125.88>
(1 row)
```

Im folgenden Beispiel wird eine Zahl in eine Zeichenfolge römischer Zahlen konvertiert.

```
select to_char(125, 'RN');
to_char
---------
CXXV
(1 row)
```

Im folgenden Beispiel wird der Wochentag angezeigt.

```
SELECT to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS');
              to_char
-----------------------
Wednesday, 31 09:34:26
```

Im folgenden Beispiel wird das Ordnungszahlsuffix für eine Zahl angezeigt.

```
SELECT to_char(482, '999th');
              to_char
-----------------------
 482nd
```

Im folgenden Beispiel wird in der Tabelle SALES die Provision vom gezahlten Preis abgezogen. Die Differenz wird dann aufgerundet und in eine römische Zahl umgewandelt, die in der folgenden Spalte angezeigt wird: `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)
```

Im folgenden Beispiel wird das Währungssymbol zu den in der `to_char` Spalte angezeigten Differenzwerten hinzugefügt:

```
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)
```

Im folgenden Beispiel wird das Jahrhundert aufgelistet, in dem die einzelnen Verkäufe ausgeführt wurden.

```
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)
```

Im folgenden Beispiel wird jeder STARTTIME-Wert in der Tabelle EVENT in eine Zeichenfolge konvertiert, die aus Stunden, Minuten, Sekunden und Zeitzone besteht.

```
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)
```

Im folgenden Beispiel wird die Formatierung für Sekunden, Millisekunden und Mikrosekunden gezeigt.

```
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
```

TO_DATE-Funktion

TO_DATE konvertiert ein Datum in einer Zeichenfolge in den Datentyp DATE.

Syntax

```
TO_DATE (date_str)
```

```
TO_DATE (date_str, format)
```

Argumente

date_str

Eine Datumszeichenfolge oder ein Datentyp, der in eine Datumszeichenfolge umgewandelt werden kann.

format

Ein Zeichenkettenliteral, das den Datetime-Mustern von Spark entspricht. Gültige Datetime-Muster finden Sie unter Datetime-Muster für Formatierung und Analyse.

Rückgabetyp

TO_DATE gibt ein DATE zurück, abhängig vom Formatwert.

Wenn die Konvertierung in das Format fehlschlägt, wird ein Fehler zurückgegeben.

Beispiele

Die folgende SQL-Anweisung konvertiert das Datum `02 Oct 2001` in einem Datumsdatentyp.

```
select to_date('02 Oct 2001', 'dd MMM yyyy');

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

Die folgende SQL-Anweisung konvertiert die Zeichenfolge `20010631` in ein Datum.

```
select to_date('20010631', 'yyyyMMdd');
```

Die folgende SQL-Anweisung konvertiert die Zeichenfolge `20010631` in ein Datum:

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

Das Ergebnis ist ein Nullwert, da der Juni nur 30 Tage hat.

```
to_date
------------
```

```
NULL
```

TO_NUMBER

TO_NUMBER konvertiert eine Zeichenfolge in einen numerischen Wert (Dezimalwert).

Syntax

```
to_number(string, format)
```

Argumente

string

Die Zeichenfolge, die konvertiert werden soll. Das Format muss ein Literalwert sein.

format

Das zweite Argument ist eine Formatzeichenfolge, die anzeigt, wie die Zeichenfolge analysiert werden muss, um den numerischen Wert zu generieren. Beispielsweise gibt das Format `'99D999'` an, dass die Zeichenfolge, die konvertiert werden soll, aus fünf Ziffern mit dem Dezimalzeichen an dritter Position besteht. Beispielsweise gibt `to_number('12.345','99D999')` `12.345` als einen numerischen Wert zurück. Die Liste der gültigen Formate finden Sie unter Numerische Formatzeichenfolgen.

Rückgabetyp

TO_NUMBER gibt eine Dezimalzahl zurück.

Wenn die Konvertierung in das Format fehlschlägt, wird ein Fehler zurückgegeben.

Beispiele

Im folgenden Beispiel wird die Zeichenfolge `12,454.8-` in eine Zahl konvertiert:

```
select to_number('12,454.8-', '99G999D9S');

to_number
-----------
-12454.8
```

Im folgenden Beispiel wird die Zeichenfolge `$ 12,454.88` in eine Zahl konvertiert:

```
select to_number('$ 12,454.88', 'L 99G999D99');

to_number
-----------
12454.88
```

Im folgenden Beispiel wird die Zeichenfolge `$ 2,012,454.88` in eine Zahl konvertiert:

```
select to_number('$ 2,012,454.88', 'L 9,999,999.99');

to_number
-----------
2012454.88
```

UNBASE64 Funktion

Die UNBASE64 Funktion konvertiert ein Argument von einer Base-64-Zeichenfolge in eine Binärdatei.

Die Base64-Kodierung wird häufig verwendet, um Binärdaten (wie Bilder, Dateien oder verschlüsselte Informationen) in einem Textformat darzustellen, das für die Übertragung über verschiedene Kommunikationskanäle (wie E-Mail, URL-Parameter oder Datenbankspeicher) sicher ist.

Mit dieser UNBASE64 Funktion können Sie diesen Vorgang rückgängig machen und die ursprünglichen Binärdaten wiederherstellen. Diese Art von Funktionalität kann in Szenarien nützlich sein, in denen Sie mit Daten arbeiten müssen, die im Base64-Format codiert wurden, z. B. bei der Integration mit externen Systemen oder bei APIs denen Base64 als Datenübertragungsmechanismus verwendet wird.

Syntax

```
unbase64(expr)
```

Argumente

expr

Ein STRING-Ausdruck in einem Base64-Format.

Rückgabetyp

`BINARY`

Beispiel

Im folgenden Beispiel wird die Base64-kodierte Zeichenfolge wieder in `'U3BhcmsgU1FM'` die ursprüngliche Zeichenfolge konvertiert. `'Spark SQL'`

```
SELECT unbase64('U3BhcmsgU1FM');
 Spark SQL
```

UNHEX-Funktion

Die UNHEX-Funktion konvertiert eine hexadezimale Zeichenfolge zurück in ihre ursprüngliche Zeichenfolgendarstellung.

Diese Funktion kann in Szenarien nützlich sein, in denen Sie mit Daten arbeiten müssen, die in einem Hexadezimalformat gespeichert oder übertragen wurden, und Sie die ursprüngliche Zeichenkettendarstellung für die weitere Verarbeitung oder Anzeige wiederherstellen müssen.

Die UNHEX-Funktion ist das Gegenstück zur HEX-Funktion.

Syntax

```
unhex(expr)
```

Argumente

expr

Ein STRING-Ausdruck mit Hexadezimalzeichen.

Rückgabetyp

UNHEX gibt einen BINÄRWERT zurück.

Wenn die Länge von expr ungerade ist, wird das erste Zeichen verworfen und das Ergebnis mit einem Null-Byte aufgefüllt. Wenn expr Zeichen enthält, die keine Hexadezimalzahlen sind, ist das Ergebnis NULL.

Beispiel

Im folgenden Beispiel wird eine hexadezimale Zeichenfolge wieder in ihre ursprüngliche Zeichenkettendarstellung konvertiert, indem die Funktionen UNHEX () und DECODE () zusammen verwendet werden. Im ersten Teil der Abfrage wird die Funktion UNHEX () verwendet, um die

hexadezimale Zeichenfolge '537061726B2053514C' in ihre binäre Darstellung zu konvertieren. Der zweite Teil der Abfrage verwendet die Funktion DECODE (), um die mit der UNHEX () -Funktion erhaltenen Binärdaten wieder in eine Zeichenfolge zu konvertieren, wobei die Zeichenkodierung 'UTF-8' verwendet wird. Die Ausgabe der Abfrage ist die ursprüngliche Zeichenfolge 'Spark_SQL', die in eine Hexadezimalzahl und dann wieder in eine Zeichenfolge konvertiert wurde.

```
SELECT decode(unhex('537061726B2053514C'), 'UTF-8');
 Spark SQL
```

Datum-/Uhrzeit-Formatzeichenfolgen

Sie können Datetime-Muster in den folgenden gängigen Szenarien verwenden:

- Bei der Arbeit mit CSV- und JSON-Datenquellen zum Analysieren und Formatieren von Datums-/Uhrzeitinhalten
- Bei der Konvertierung zwischen Zeichenfolgentypen und Datums- oder Zeitstempeltypen mithilfe von Funktionen wie:
  - unix_timestamp
  - date_format
  - to_unix_timestamp
  - von_unixtime
  - to_date
  - to_timestamp
  - from_utc_timestamp
  - to_utc_timestamp

Verwenden Sie die Musterbuchstaben in der folgenden Tabelle für die Analyse und Formatierung von Datum und Zeitstempel.

Datumsteil oder Zeitteil	Bedeutung	Beispiele
a	AM oder PM des Tages, dargestellt als AM-PM	PM
D	Tag des Jahres, dargestellt als dreistellige Zahl	189

Datumsteil oder Zeitteil	Bedeutung	Beispiele
d	Tag des Monats, dargestellt als zweistellige Zahl	28
E	Wochentag, als Text dargestellt	Di Dienstag
F	Einheitlicher Wochentag im Monat, dargestellt als einstellige Zahl	3
G	Äraindikator, dargestellt als Text	AD Anno Domini
h	Uhrzeit von AM oder PM, dargestellt als zweistellige Zahl	12
H	Stunde des Tages, dargestellt als zweistellige Zahl zwischen 0 und 23	0
k	Uhrzeit des Tages, dargestellt als zweistellige Zahl von 1—24	1
K	Uhrzeit von AM oder PM, dargestellt als zweistellige Zahl von 0—11	0
m	Minute der Stunde, dargestellt als zweistellige Zahl	30

Datumsteil oder Zeitteil	Bedeutung	Beispiele
M/L	Monat des Jahres, dargestellt als Monat	7 07 Juli Juli
O	Lokalisierter Zonenversatz von UTC	GMT+8 GMT+ 8:00 UTC-08:00
q/Q	Quartal des Jahres, dargestellt als Zahl (1 bis 4) oder Text	3 03 Q3 3. Quartal
S	Sekunde der Minute, dargestellt als zweistellige Zahl	55
S	Bruchteil einer Sekunde, dargestellt als Bruchteil	978
V	Zeitzonen-ID, dargestellt als Zonen-ID	Amerika/Los_Angeles Z 08:30

Datumsteil oder Zeitteil	Bedeutung	Beispiele
x	Zonenversatz von UTC (Offset-X)	+0000 -08 -0830 - 08:30 -083015 - 08:30:15
X	Zonenversatz von UTC; wobei Z für Null steht	Z -08 -0830 - 08:30 -083015 - 08:30:15
y	Jahr, als Jahr dargestellt	2020 20
z	Name der Zeitzone, als Text dargestellt	Pacific Standard Time PST
Z	Zonenversatz von UTC (Offset-Z)	+0000 -0800 - 08:00
'	Escape für Text, dargestellt als Trennzeichen	N/A

Datumsteil oder Zeitteil	Bedeutung	Beispiele
''	Einfaches Anführungszeichen, wortwörtlich dargestellt	'
[	Optionaler Beginn des Abschnitts	N/A
]	Optionales Ende des Abschnitts	N/A

Die Anzahl der Musterbuchstaben bestimmt den Formattyp:

Textformat

- Verwenden Sie 1—3 Buchstaben für die abgekürzte Form (z. B. „Mon“ für Montag)
- Verwenden Sie genau 4 Buchstaben für das vollständige Formular (z. B. „Montag“)
- Verwenden Sie nicht 5 oder mehr Buchstaben - dies führt zu einem Fehler

Zahlenformat (n)

- Der Wert n steht für die maximal zulässige Anzahl von Buchstaben
- Für Muster mit einzelnen Buchstaben:
  - Die Ausgabe verwendet mindestens Ziffern ohne Auffüllung
- Für mehrere Buchstabenmuster:
  - Die Ausgabe wird mit Nullen aufgefüllt, um der Breite der Buchstabenzahl zu entsprechen
- Beim Parsen muss die Eingabe die genaue Anzahl von Ziffern enthalten

Zahlen-/Textformat

- Folgen Sie bei 3 oder mehr Buchstaben den Regeln für das Textformat
- Folgen Sie den Regeln für das Zahlenformat, um weniger Buchstaben zu erhalten

Format für Brüche

- Verwenden Sie 1—9 S-Zeichen (z. B. SSSSSS)

- Zum Parsen:
  - Akzeptiere Brüche zwischen 1 und der Anzahl der S-Zeichen
- Für die Formatierung:
  - Geben Sie Nullen ein, um der Anzahl der S-Zeichen zu entsprechen
- Unterstützt bis zu 6 Ziffern für eine Genauigkeit im Mikrosekundenbereich
- Kann Nanosekunden analysieren, schneidet aber zusätzliche Ziffern ab

Jahresformat

- Die Buchstabenzahl legt die minimale Feldbreite für den Innenabstand fest
- Für zwei Buchstaben:
  - Druckt die letzten beiden Ziffern
  - Analysiert Jahre zwischen 2000 und 2099
- Für weniger als vier Buchstaben (außer zwei):
  - Zeigt das Vorzeichen nur für negative Jahre
- Verwenden Sie nicht 7 oder mehr Buchstaben - dies führt zu einem Fehler

Format des Monats

- Verwenden Sie 'M' für das Standardformular oder 'L' für das eigenständige Formular
- Einfaches 'M' oder 'L':
  - Zeigt die Monatszahlen 1—12 ohne Polsterung
- 'MM' oder 'LL':
  - Zeigt die Monatszahlen 01—12 mit Polsterung
- 'MMM':
  - Zeigt den abgekürzten Monatsnamen in Standardform
  - Muss Teil eines vollständigen Datumsmusters sein
- „LLL“:
  - Zeigt den abgekürzten Monatsnamen in eigenständiger Form
  - Wird nur für die monatliche Formatierung verwendet
- 'MMMM':

  - Zeigt den vollständigen Monatsnamen in Standardform
  - Wird für Datums- und Zeitstempel verwendet
- 'LLLL':
  - Zeigt den vollständigen Monatsnamen in eigenständiger Form
  - Nur für die monatliche Formatierung verwenden

Zeitzonenformate

- am-pm: Verwenden Sie nur einen Buchstaben
- Zonen-ID (V): Verwenden Sie nur 2 Buchstaben
- Zonennamen (z):
  - 1—3 Buchstaben: Zeigt den Kurznamen
  - 4 Buchstaben: Zeigt den vollständigen Namen
  - Verwenden Sie nicht 5 oder mehr Buchstaben

Offset-Formate

- X und x:
  - 1 Buchstabe: Zeigt Stunde (+01) oder Stundenminute (+0130)
  - 2 Buchstaben: Zeigt die Stunde und Minute ohne Doppelpunkt an (+0130)
  - 3 Buchstaben: Zeigt die Stunde und Minute mit Doppelpunkt an (+ 01:30)
  - 4 Buchstaben: Wird hour-minute-second ohne Doppelpunkt angezeigt (+013015)
  - 5 Buchstaben: Wird hour-minute-second mit Doppelpunkt angezeigt (+ 01:30:15)
  - X verwendet 'Z' für einen Nullversatz
  - x verwendet '+00', '+0000' oder '+ 00:00 'für einen Nullversatz
- O:
  - 1 Buchstabe: Zeigt die Kurzform an (GMT+8)
  - 4 Buchstaben: Zeigt die vollständige Form an (GMT+ 08:00)
- Z:
  - 1-3 Buchstaben: Zeigt die Stunde und Minute ohne Doppelpunkt an (+0130)
  - 4 Buchstaben: Zeigt die vollständige lokalisierte Form
  - 5 Buchstaben: Wird hour-minute-second mit Doppelpunkt angezeigt

Optionale Abschnitte

- Verwenden Sie eckige Klammern [], um optionale Inhalte zu markieren
- Sie können optionale Abschnitte verschachteln
- Alle gültigen Daten werden in der Ausgabe angezeigt
- Bei der Eingabe können ganze optionale Abschnitte weggelassen werden

> **Note**
>
> Die Symbole 'E', 'F', 'q' und 'Q' funktionieren nur für die Formatierung von Datum und Uhrzeit (wie date_format). Verwenden Sie sie nicht für die Datetime-Analyse (wie to_timestamp).

Numerische Formatzeichenfolgen

Die folgenden Zeichenketten im numerischen Format gelten für Funktionen wie TO_NUMBER und TO_CHAR.

- Beispiele für das Formatieren von Zeichenfolgen als Zahlen finden Sie unter TO_NUMBER.
- Beispiele für das Formatieren von Zahlen als Zeichenfolgen finden Sie unterTO_CHAR.

Format	Beschreibung
9	Numerischer Wert mit der angegebenen Anzahl von Stellen.
0	Numerischer Wert mit Nullen zu Beginn.
. (Punkt), D	Dezimalpunkt.
, (Komma)	Tausendertrennzeichen.
CC	Jahrhundertcode. Das 21. Jahrhundert begann beispielsweise am 01.01.2001 (wird nur für TO_CHAR unterstützt).

Format	Beschreibung
FM	Füllmodus. Unterdrückt ausfüllende Leerzeichen und Nullen.
PR	Negativer Wert in Winkelklammern.
S	Vorzeichen, das mit einer Zahl fest verbunden ist.
L	Währungssymbol an der angegebenen Position.
G	Gruppentrennzeichen.
MI	Minuszeichen an der angegebenen Position für Zahlen kleiner als 0.
PL	Pluszeichen an der angegebenen Position für Zahlen größer als 0.
SG	Plus- oder Minuszeichen an der angegebenen Position.
RN	Römische Zahl zwischen 1 und 3999 (wird nur für TO_CHAR unterstützt).
TH oder th	Ordnungszahlsuffix. Konvertiert keine Bruchzahlen oder Werte kleiner als null.

Datums- und Zeitfunktionen

Mit Datums- und Uhrzeitfunktionen können Sie eine Vielzahl von Vorgängen mit Datums- und Uhrzeitdaten ausführen, z. B. Teile eines Datums extrahieren, Datumsberechnungen durchführen, Datums- und Uhrzeitdaten formatieren und mit dem aktuellen Datum und der aktuellen Uhrzeit arbeiten. Diese Funktionen sind für Aufgaben wie Datenanalyse, Berichterstattung und Datenmanipulation mit Zeitdaten unerlässlich.

AWS Clean Rooms unterstützt die folgenden Datums- und Uhrzeitfunktionen:

Themen

Funktion ADD_MONTHS

ADD_MONTHS fügt die angegebene Zahl von Monaten zu einem Datums- oder Zeitstempelwert bzw. -ausdruck hinzu. Die Funktion DATE_ADD bietet eine ähnliche Funktionalität.

Syntax

```
ADD_MONTHS( {date | timestamp}, integer)
```

Argumente

date | timestamp

Eine Datums- oder Zeitstempelspalte bzw. ein entsprechender Ausdruck, die/der implizit zu einem Datum oder Zeitstempel konvertiert wird. Wenn das Datum der letzte Tag des Monats ist, oder wenn der resultierende Monat kürzer ist, gibt die Funktion im Ergebnis den letzten Tag des Monats aus. Für andere Datumsangaben enthält das Ergebnis die gleiche Tagesnummer wie der Datumsausdruck.

integer

Eine positive oder negative Ganzzahl. Verwenden Sie eine negative Zahl, um Monate von Datumsangaben abzuziehen.

Rückgabetyp

TIMESTAMP

Beispiel

Die folgende Abfrage verwendet die Funktion ADD_MONTHS innerhalb einer TRUNC-Funktion. Die TRUNC-Funktion entfernt die Tageszeit aus dem Ergebnis von ADD_MONTHS. Die Funktion ADD_MONTHS fügt jedem Wert aus der Spalte CALDATE 12 Monate hinzu.

```
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)
```

Die folgenden Beispiele illustrieren die Verhaltensweise, wenn die Funktion ADD_MONTHS für Datumsangaben verwendet wird, die Monate mit unterschiedlichen Anzahlen von Tagen enthalten.

```
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)
```

Funktion CONVERT_TIMEZONE

CONVERT_TIMEZONE konvertiert einen Zeitstempel von einer Zeitzone zu einer anderen. Die Funktion passt sich automatisch an die Sommerzeit an.

Syntax

```
CONVERT_TIMEZONE ( ['source_timezone',] 'target_timezone', 'timestamp')
```

Argumente

source_timezone

(Optional) Die Zeitzone des aktuellen Zeitstempels. Der Standardwert ist UTC.

target_timezone

Die Zeitzone für den neuen Zeitstempel.

timestamp

Eine Zeitstempelspalte bzw. ein entsprechender Ausdruck, die/der implizit zu einem Zeitstempel konvertiert wird.

Rückgabetyp

TIMESTAMP

Beispiele

Das folgende Beispiel konvertiert den Zeitstempelwert von der Standardzeitzone UTC zu PST.

```
select convert_timezone('PST', '2008-08-21 07:23:54');

 convert_timezone
-----------------------
2008-08-20 23:23:54
```

Das folgende Beispiel konvertiert den Zeitstempelwert in der Spalte LISTTIME von der Standardzeitzone UTC zu PST. Obwohl der Zeitstempel in der Sommerzeitzone liegt, wird er zur Standardzeit konvertiert, da die Zielzeitzone als Abkürzung (PST) angegeben ist.

```
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
```

Im folgenden Beispiel wird eine LISTTIME-Spalte mit einem Zeitstempel von der Standard-UTC-Zeitzone in eine Zeitzone konvertiert US/Pacific . Die Zielzeitzone verwendet einen Zeitzonennamen, und der Zeitstempel liegt im Sommerzeitzeitraum, weshalb die Funktion die Sommerzeit ausgibt.

```
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
```

Das folgende Beispiel konvertiert eine Zeitstempelzeichenfolge von EST zu PST:

```
select convert_timezone('EST', 'PST', '20080305 12:25:29');

 convert_timezone
-------------------
2008-03-05 09:25:29
```

Das folgende Beispiel konvertiert einen Zeitstempel zu US Eastern Standard Time, da die Zielzeitzone einen Zeitzonennamen (America/New York) verwendet und der Zeitstempel im Standardzeitzeitraum liegt.

```
select convert_timezone('America/New_York', '2013-02-01 08:00:00');

 convert_timezone
---------------------
2013-02-01 03:00:00
(1 row)
```

Das folgende Beispiel konvertiert einen Zeitstempel zu US Eastern Daylight Time, da die Zielzeitzone einen Zeitzonennamen (America/New York) verwendet und der Zeitstempel im Sommerzeitzeitraum liegt.

```
select convert_timezone('America/New_York', '2013-06-01 08:00:00');

 convert_timezone
---------------------
2013-06-01 04:00:00
(1 row)
```

Das folgende Beispiel illustriert die Verwendung von Verschiebungen.

```
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)
```

Funktion CURRENT_DATE

CURRENT_DATE gibt ein Datum in der Zeitzone der aktuellen Sitzung (standardmäßig UTC) im Standardformat zurück:. YYYY-MM-DD

> **Note**
>
> CURRENT_DATE gibt das Startdatum für die aktuelle Transaktion aus, nicht für den Start der aktuellen Anweisung. Angenommen, Sie starten eine mehrere Anweisungen umfassende Transaktion am 01.10.08 um 23:59 Uhr und die Anweisung mit CURRENT_DATE wird am 02.10.08 um 00:00 Uhr ausgeführt. CURRENT_DATE gibt dann `10/01/08` zurück, nicht `10/02/08`.

Syntax

```
CURRENT_DATE
```

Rückgabetyp

DATUM

Beispiel

Das folgende Beispiel gibt das aktuelle Datum zurück (in AWS-Region dem die Funktion ausgeführt wird).

```
select current_date;

   date
------------
2008-10-01
```

CURRENT_TIMESTAMP-Funktion

CURRENT_TIMESTAMP gibt das aktuelle Datum und die aktuelle Uhrzeit zurück, einschließlich Datum, Uhrzeit und (optional) der Millisekunden oder Mikrosekunden.

Diese Funktion ist nützlich, wenn Sie das aktuelle Datum und die aktuelle Uhrzeit abrufen müssen, um beispielsweise den Zeitstempel eines Ereignisses aufzuzeichnen, zeitbasierte Berechnungen durchzuführen oder Spalten aufzufüllen. date/time

Syntax

```
current_timestamp()
```

Rückgabetyp

Die CURRENT_TIMESTAMP-Funktion gibt ein DATUM zurück.

Beispiel

Das folgende Beispiel gibt das aktuelle Datum und die aktuelle Uhrzeit zum Zeitpunkt der Ausführung der Abfrage zurück, also am 25. April 2020 um 15:49:11.914 (15:49:11.914 Uhr).

```
SELECT current_timestamp();
 2020-04-25 15:49:11.914
```

Im folgenden Beispiel werden das aktuelle Datum und die aktuelle Uhrzeit für jede Zeile in der Tabelle abgerufen. `squirrels`

```
SELECT current_timestamp() FROM squirrels
```

DATE_ADD-Funktion

Gibt das Datum zurück, das num_days nach start_date liegt.

Syntax

```
date_add(start_date, num_days)
```

Argumente

start_date

Der Wert für das Startdatum.

Anzahl_Tage

Die Anzahl der hinzuzufügenden Tage (Ganzzahl). Eine positive Zahl addiert Tage, eine negative Zahl subtrahiert Tage.

Rückgabetyp

DATUM

Beispiele

Das folgende Beispiel fügt einem Datum einen Tag hinzu:

```
SELECT date_add('2016-07-30', 1);

Result:
2016-07-31
```

Im folgenden Beispiel werden mehrere Tage hinzugefügt.

```
SELECT date_add('2016-07-30', 5);

Result:
2016-08-04
```

Nutzungshinweise

Diese Dokumentation bezieht sich auf die DATE_ADD-Funktion von Spark SQL, die im Vergleich zu einigen anderen SQL-Varianten eine einfachere Schnittstelle zum Hinzufügen von Tagen zu Daten bietet. Für das Hinzufügen anderer Intervalle wie Monate oder Jahre sind möglicherweise andere Funktionen erforderlich.

DATE_DIFF-Funktion

DATE_DIFF gibt die Differenz zwischen den Datumsteilen zweier Datums- oder Uhrzeitausdrücke zurück.

Syntax

```
date_diff(endDate, startDate)
```

Argumente

endDate

Ein DATE-Ausdruck.

startDate

Ein DATE-Ausdruck.

Rückgabetyp

BIGINT

Beispiele mit einer DATE-Spalte

Im folgenden Beispiel wird die Differenz als Anzahl von Wochen zwischen zwei Literal-Datumswerten berechnet.

```
select date_diff(week,'2009-01-01','2009-12-31') as numweeks;

numweeks
----------
52
(1 row)
```

Im folgenden Beispiel wird die Differenz in Stunden zwischen zwei Literal-Datumswerten ermittelt. Wenn Sie den Zeitwert für ein Datum nicht angeben, wird standardmäßig 00:00:00 verwendet.

```
select date_diff(hour, '2023-01-01', '2023-01-03 05:04:03');

date_diff
----------
53
(1 row)
```

Im folgenden Beispiel wird die Differenz in Tagen zwischen zwei TIMESTAMETZ-Literalwerten ermittelt.

```
Select date_diff(days, 'Jun 1,2008  09:59:59 EST', 'Jul 4,2008  09:59:59 EST')

date_diff
----------
33
```

Im folgenden Beispiel wird die Differenz in Tagen zwischen zwei Daten in derselben Zeile einer Tabelle ermittelt.

```
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)
```

Im folgenden Beispiel wird die Differenz als Anzahl von Quartalen zwischen einem in der Vergangenheit liegenden Literalwert und dem heutigen Datum berechnet. Bei diesem Beispiel wird davon ausgegangen, dass das aktuelle Datum der 5. Juni 2008 ist. Sie können Datumsteile ausschreiben oder abkürzen. Der Standardspaltenname für die DATE_DIFF-Funktion ist DATE_DIFF.

```
select date_diff(qtr, '1998-07-01', current_date);

date_diff
-----------
40
(1 row)
```

Das folgende Beispiel verbindet die Tabellen SALES und LISTING zur Berechnung, wie viel Tage nach ihrer Auflistung Tickets für die Auflistungen 1000 bis 1005 verkauft wurden. Die längste Wartezeit für den Verkauf dieser Auflistungen betrug 15 Tage, und die kürzeste lag unter einem Tag (0 Tage).

```
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)
```

Dieses Beispiel berechnet die durchschnittliche Zahl von Stunden, für die Verkäufer auf alle Ticketverkäufe warteten.

```
select avg(date_diff(hours, listtime, saletime)) as avgwait
from sales, listing
where sales.listid = listing.listid;

avgwait
---------
465
(1 row)
```

Beispiele mit einer TIME-Spalte

Die folgende Beispieltabelle TIME_TEST enthält eine Spalte TIME_VAL (Typ TIME) mit drei eingefügten Werten.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

Im folgenden Beispiel wird die Differenz als Anzahl von Stunden zwischen der TIME_VAL-Spalte und einem Zeitliteral berechnet.

```
select date_diff(hour, time_val, time '15:24:45') from time_test;

 date_diff
-----------
        -5
        15
        15
```

Im folgenden Beispiel wird die Differenz als Anzahl von Minuten zwischen zwei Literal-Zeitwerten berechnet.

```
select date_diff(minute, time '20:00:00', time '21:00:00') as nummins;

nummins
```

```
----------
60
```

Beispiele mit einer TIMETZ-Spalte

Die folgende Beispieltabelle TIMETZ_TEST enthält eine Spalte TIMETZ_VAL (Typ TIMETZ) mit drei eingefügten Werten.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

Im folgenden Beispiel werden die Differenzen als Anzahl von Stunden zwischen dem TIMETZ-Literal und timetz_val berechnet.

```
select date_diff(hours, timetz '20:00:00 PST', timetz_val) as numhours from
 timetz_test;

numhours
----------
0
-4
1
```

Im folgenden Beispiel wird die Differenz als Anzahl von Stunden zwischen zwei Literal-TIMETZ-Werten berechnet.

```
select date_diff(hours, timetz '20:00:00 PST', timetz '00:58:00 EST') as numhours;

numhours
----------
1
```

Funktion DATE_PART

DATE_PART extrahiert Datumsteilwerte aus einem Ausdruck. DATE_PART ist synonym mit der Funktion PGDATE_PART.

Syntax

```
datepart(field, source)
```

Argumente

field

Welcher Teil der Quelle extrahiert werden soll, und die unterstützten Zeichenkettenwerte sind dieselben wie die Felder der entsprechenden Funktion EXTRACT.

source

Eine DATE- oder INTERVAL-Spalte, aus der das Feld extrahiert werden soll.

Rückgabetyp

Wenn das Feld 'SECOND' ist, eine DEZIMALZAHL (8, 6). In allen anderen Fällen eine Ganzzahl.

Beispiel

Im folgenden Beispiel wird der Tag des Jahres (DOY) aus einem Datumswert extrahiert. Die Ausgabe zeigt, dass der Tag des Jahres für das Datum „2019-08-12" ist. 224 Das bedeutet, dass der 12. August 2019 der 224. Tag des Jahres 2019 ist.

```
SELECT datepart('doy', DATE'2019-08-12');
 224
```

Funktion DATE_TRUNC

Die Funktion DATE_TRUNC verkürzt alle Zeitstempelausdrücke oder Literale auf der Grundlage des angegebenen Datumsteils, beispielsweise Stunde, Tag oder Monat.

Syntax

```
date_trunc(format, datetime)
```

Argumente

format

Das Format, das die Einheit darstellt, auf die gekürzt werden soll. Gültige Formate sind folgende:

- „YEAR“, „YYYY“, „YY“ — kürzt auf das erste Datum des Jahres, in das das TS fällt, der Zeitteil wird auf Null gesetzt
- „QUARTER“ — kürzt auf das erste Datum des Quartals, in das das TS fällt, der Zeitteil wird auf Null gesetzt
- „MONTH“, „MM“, „MON“ — kürzen Sie den Wert auf das erste Datum des Monats, in den das TS fällt, und der Zeitteil wird auf Null gesetzt
- „WOCHE“ — wird auf den Montag der Woche gekürzt, in den das TS fällt, der Zeitteil wird auf Null gesetzt
- „DAY“, „DD“ — setzt den Zeitteil auf Null
- „HOUR“ — setzt die Minute und die Sekunde mit Bruchteilen auf Null
- „MINUTE“ — setzt die Sekunde mit Bruchteil auf Null
- „SECOND“ — setzt den zweiten Bruchteil auf Null
- „MILLISECOND“ — setzt die Mikrosekunden auf Null
- „MIKROSEKUNDE“ — alles bleibt

ts

Ein Datetime-Wert

Rückgabetyp

Gibt den Zeitstempel ts zurück, gekürzt auf die im Formatmodell angegebene Einheit

Beispiele

Im folgenden Beispiel wird ein Datumswert auf den Jahresanfang gekürzt. Die Ausgabe zeigt, dass das Datum „2015-03-05“ auf „2015-01-01“ gekürzt wurde, was dem Beginn des Jahres 2015 entspricht.

```
SELECT date_trunc('YEAR', '2015-03-05');

 date_trunc
-----------
2015-01-01
```

DAY-Funktion

Die DAY-Funktion gibt den Tag des Monats des Datums/Zeitstempels zurück.

Datumsextraktionsfunktionen sind nützlich, wenn Sie mit bestimmten Komponenten eines Datums oder Zeitstempels arbeiten müssen, z. B. wenn Sie datumsbasierte Berechnungen durchführen, Daten filtern oder Datumswerte formatieren.

Syntax

```
day(date)
```

Argumente

date

Ein DATE- oder TIMESTAMP-Ausdruck.

Rückgabewert

Die DAY-Funktion gibt einen INTEGER-Wert zurück.

Beispiele

Im folgenden Beispiel wird der Tag des Monats (`30`) aus dem Eingabedatum extrahiert`'2009-07-30'`.

```
SELECT day('2009-07-30');
 30
```

Im folgenden Beispiel wird der Tag des Monats aus der `birthday` `squirrels` Tabellenspalte extrahiert und die Ergebnisse als Ausgabe der SELECT-Anweisung zurückgegeben. Die Ausgabe dieser Abfrage ist eine Liste von Tageswerten, einer für jede Zeile in der `squirrels` Tabelle, die den Tag des Monats darstellt, an dem jedes Eichhörnchen Geburtstag hat.

```
SELECT day(birthday) FROM squirrels
```

DAYOFMONTH-Funktion

Die Funktion DAYOFMONTH gibt den Tag des Monats von zurück date/timestamp (ein Wert zwischen 1 und 31, abhängig von Monat und Jahr).

Die DAYOFMONTH-Funktion ähnelt der DAY-Funktion, hat jedoch leicht unterschiedliche Namen und ein leicht unterschiedliches Verhalten. Die DAY-Funktion wird häufiger verwendet, aber die

DAYOFMONTH-Funktion kann als Alternative verwendet werden. Diese Art von Abfrage kann nützlich sein, wenn Sie eine datumsbasierte Analyse oder Filterung für eine Tabelle durchführen müssen, die Datums- oder Zeitstempeldaten enthält, z. B. wenn Sie bestimmte Komponenten eines Datums für die weitere Verarbeitung oder Berichterstattung extrahieren müssen.

Syntax

```
dayofmonth(date)
```

Argumente

date

Ein DATE- oder TIMESTAMP-Ausdruck.

Rückgabewert

Die Funktion DAYOFMONTH gibt einen INTEGER-Wert zurück.

Beispiel

Im folgenden Beispiel wird der Tag des Monats (`30`) aus dem Eingabedatum extrahiert. `'2009-07-30'`

```
SELECT dayofmonth('2009-07-30');
 30
```

Im folgenden Beispiel wird die Funktion DAYOFMONTH auf die `birthday` Spalte der `squirrels` Tabelle angewendet. Für jede Zeile in der `squirrels` Tabelle wird der Tag des Monats aus der `birthday` Spalte extrahiert und als Ausgabe der SELECT-Anweisung zurückgegeben. Die Ausgabe dieser Abfrage ist eine Liste von Tageswerten, einer für jede Zeile in der `squirrels` Tabelle, die den Tag des Monats darstellt, an dem jedes Eichhörnchen Geburtstag hat.

```
SELECT dayofmonth(birthday) FROM squirrels
```

DAYOFWEEK-Funktion

Die DAYOFWEEK-Funktion verwendet ein Datum oder einen Zeitstempel als Eingabe und gibt den Wochentag als Zahl zurück (1 für Sonntag, 2 für Montag,..., 7 für Samstag).

Diese Datumsextraktionsfunktion ist nützlich, wenn Sie mit bestimmten Komponenten eines Datums oder Zeitstempels arbeiten müssen, z. B. wenn Sie datumsbasierte Berechnungen durchführen, Daten filtern oder Datumswerte formatieren.

Syntax

```
dayofweek(date)
```

Argumente

date

Ein DATE- oder TIMESTAMP-Ausdruck.

Rückgabewert

Die DAYOFWEEK-Funktion gibt einen INTEGER-Wert zurück, wobei

1 = Sonntag

2 = Montag

3 = Dienstag

4 = Mittwoch

5 = Donnerstag

6 = Freitag

7 = Samstag

Beispiele

Im folgenden Beispiel wird der Wochentag aus diesem Datum extrahiert, das 5 ist (für Donnerstag).

```
SELECT dayofweek('2009-07-30');
 5
```

Im folgenden Beispiel wird der Wochentag aus der `birthday` Spalte der `squirrels` Tabelle extrahiert und die Ergebnisse als Ausgabe der SELECT-Anweisung zurückgegeben. Die Ausgabe dieser Abfrage ist eine Liste mit Wochentagswerten, einer für jede Zeile in der `squirrels` Tabelle, die den Wochentag für den Geburtstag jedes Eichhörnchens darstellt.

```
SELECT dayofweek(birthday) FROM squirrels
```

DAYOFYEAR-Funktion

Die DAYOFYEAR-Funktion ist eine Datumsextraktionsfunktion, die ein Datum oder einen Zeitstempel als Eingabe verwendet und den Tag des Jahres zurückgibt (ein Wert zwischen 1 und 366, abhängig vom Jahr und davon, ob es sich um ein Schaltjahr handelt).

Diese Funktion ist nützlich, wenn Sie mit bestimmten Komponenten eines Datums oder Zeitstempels arbeiten müssen, z. B. wenn Sie datumsbasierte Berechnungen durchführen, Daten filtern oder Datumswerte formatieren.

Syntax

```
dayofyear(date)
```

Argumente

date

Ein DATE- oder TIMESTAMP-Ausdruck.

Rückgabewert

Die DAYOFYEAR-Funktion gibt einen INTEGER-Wert zurück (zwischen 1 und 366, abhängig vom Jahr und davon, ob es sich um ein Schaltjahr handelt).

Beispiele

Im folgenden Beispiel wird der Tag des Jahres (`100`) aus dem Eingabedatum extrahiert. `'2016-04-09'`

```
SELECT dayofyear('2016-04-09');
 100
```

Im folgenden Beispiel wird der Tag des Jahres aus der `birthday` `squirrels` Tabellenspalte extrahiert und die Ergebnisse als Ausgabe der SELECT-Anweisung zurückgegeben.

```
SELECT dayofyear(birthday) FROM squirrels
```

Funktion EXTRACT

Die EXTRACT-Funktion gibt einen Datums- oder Uhrzeitteil von einem TIMESTAMP-, TIMESTAMPTZ-, TIME- oder TIMETZ-Wert zurück. Beispiele hierfür sind ein Tag, Monat, Jahr, eine Stunde, Minute, Sekunde, Millisekunde oder Mikrosekunde aus einem Zeitstempel.

Syntax

```
EXTRACT(datepart FROM source)
```

Argumente

datepart

Das zu extrahierende Unterfeld eines Datums- oder Uhrzeitwerts, z. B. Tag, Monat, Jahr, Stunde, Minute, Sekunde, Millisekunde oder Mikrosekunde. Für mögliche Werte vgl. Datumsteile für Datums- oder Zeitstempelfunktionen.

source

Eine Spalte oder ein Ausdruck, der zum Datentyp TIMESTAMP, TIMESTAMPTZ, TIME oder TIMETZ ausgewertet wird.

Rückgabetyp

INTEGER, wenn der Wert source zum Datentyp TIMESTAMP, TIME oder TIMETZ ausgewertet wird.

DOUBLE PRECISION, wenn der Wert source zum Datentyp TIMESTAMPTZ ausgewertet wird.

Beispiele mit TIME

Die folgende Beispieltabelle TIME_TEST enthält eine Spalte TIME_VAL (Typ TIME) mit drei eingefügten Werten.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

Im folgenden Beispiel werden die Minuten aus jedem time_val extrahiert.

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

Im folgenden Beispiel werden die Stunden aus jedem time_val extrahiert.

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

hours
-----------
         20
         0
         0
```

FROM_UTC_TIMESTAMP-Funktion

Die Funktion FROM_UTC_TIMESTAMP konvertiert das Eingabedatum von UTC (Coordinated Universal Time) in die angegebene Zeitzone.

Diese Funktion ist nützlich, wenn Sie Datums- und Uhrzeitwerte von UTC in eine bestimmte Zeitzone konvertieren müssen. Dies kann wichtig sein, wenn Sie mit Daten arbeiten, die aus verschiedenen Teilen der Welt stammen und in der entsprechenden Ortszeit präsentiert werden müssen.

Syntax

```
from_utc_timestamp(timestamp, timezone
```

Argumente

timestamp

Ein TIMESTAMP-Ausdruck mit einem UTC-Zeitstempel.

Zeitzone

Ein STRING-Ausdruck, der eine gültige Zeitzone darstellt, in die das Eingabedatum oder der eingegebene Zeitstempel konvertiert werden soll.

Rückgabewert

Die Funktion FROM_UTC_TIMESTAMP gibt einen TIMESTAMP zurück.

Beispiel

Das folgende Beispiel konvertiert das Eingabedatum von UTC in die angegebene Zeitzone (`'Asia/Seoul'`), die in diesem Fall 9 Stunden vor UTC liegt. Die resultierende Ausgabe ist das Datum und die Uhrzeit in der Zeitzone von Seoul, also`2016-08-31 09:00:00`.

```
SELECT from_utc_timestamp('2016-08-31', 'Asia/Seoul');
 2016-08-31 09:00:00
```

HOUR-Funktion

Die HOUR-Funktion ist eine Zeitextraktionsfunktion, die eine Zeit oder einen Zeitstempel als Eingabe verwendet und die Stundenkomponente (einen Wert zwischen 0 und 23) zurückgibt.

Diese Zeitextraktionsfunktion ist nützlich, wenn Sie mit bestimmten Komponenten eines Zeit- oder Zeitstempels arbeiten müssen, z. B. wenn Sie zeitbasierte Berechnungen durchführen, Daten filtern oder Zeitwerte formatieren.

Syntax

```
hour(timestamp)
```

Argumente

timestamp

Ein TIMESTAMP-Ausdruck.

Rückgabewert

Die HOUR-Funktion gibt einen INTEGER-Wert zurück.

Beispiel

Im folgenden Beispiel wird die Stundenkomponente (12) aus dem Eingabezeitstempel `'2009-07-30 12:58:59'` extrahiert.

```
SELECT hour('2009-07-30 12:58:59');
 12
```

MINUTE-Funktion

Die MINUTE-Funktion ist eine Zeitextraktionsfunktion, die eine Zeit oder einen Zeitstempel als Eingabe verwendet und die Minutenkomponente (einen Wert zwischen 0 und 60) zurückgibt.

Syntax

```
minute(timestamp)
```

Argumente

timestamp

Ein TIMESTAMP-Ausdruck oder ein STRING mit einem gültigen Zeitstempelformat.

Rückgabewert

Die MINUTE-Funktion gibt einen INTEGER-Wert zurück.

Beispiel

Im folgenden Beispiel wird die Minutenkomponente (58) aus dem Eingabezeitstempel `'2009-07-30 12:58:59'` extrahiert.

```
SELECT minute('2009-07-30 12:58:59');
 58
```

MONTH-Funktion

Die MONTH-Funktion ist eine Zeitextraktionsfunktion, die eine Zeit oder einen Zeitstempel als Eingabe verwendet und die Monatskomponente (einen Wert zwischen 0 und 12) zurückgibt.

Syntax

```
month(date)
```

Argumente

date

Ein TIMESTAMP-Ausdruck oder ein STRING mit einem gültigen Zeitstempelformat.

Rückgabewert

Die MONTH-Funktion gibt einen INTEGER-Wert zurück.

Beispiel

Im folgenden Beispiel wird die Monatskomponente (7) aus dem Eingabezeitstempel `'2016-07-30'` extrahiert.

```
SELECT month('2016-07-30');
 7
```

SECOND-Funktion

Die SECOND-Funktion ist eine Zeitextraktionsfunktion, die eine Zeit oder einen Zeitstempel als Eingabe verwendet und die zweite Komponente zurückgibt (einen Wert zwischen 0 und 60).

Syntax

```
second(timestamp)
```

Argumente

timestamp

Ein TIMESTAMP-Ausdruck.

Rückgabewert

Die SECOND-Funktion gibt einen INTEGER-Wert zurück.

Beispiel

Im folgenden Beispiel wird die zweite Komponente (59) aus dem Eingabezeitstempel `'2009-07-30 12:58:59'` extrahiert.

```
SELECT second('2009-07-30 12:58:59');
 59
```

TIMESTAMP-Funktion

Die TIMESTAMP-Funktion nimmt einen Wert (normalerweise eine Zahl) und konvertiert ihn in einen Timestamp-Datentyp.

Diese Funktion ist nützlich, wenn Sie einen numerischen Wert, der eine Uhrzeit oder ein Datum darstellt, in einen Timestamp-Datentyp konvertieren müssen. Dies kann hilfreich sein, wenn Sie mit Daten arbeiten, die in einem numerischen Format gespeichert sind, z. B. Unix-Zeitstempel oder Epochenzeit.

Syntax

```
timestamp(expr)
```

Argumente

expr

Jeder Ausdruck, der in TIMESTAMP umgewandelt werden kann.

Rückgabewert

Die TIMESTAMP-Funktion gibt einen TIMESTAMP zurück.

Beispiel

Das folgende Beispiel konvertiert einen numerischen Unix-Zeitstempel (`1632416400`) in den entsprechenden Timestamp-Datentyp: 22. September 2021 um 12:00:00 Uhr UTC.

```
SELECT timestamp(1632416400);
 2021-09-22 12:00:00 UTC
```

Funktion TO_TIMESTAMP

TO_TIMESTAMP konvertiert eine TIMESTAMP-Zeichenfolge zu TIMESTAMPTZ.

Syntax

```
to_timestamp (timestamp)
```

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

Argumente

timestamp

Eine Zeitstempelzeichenfolge oder ein Datentyp, der in eine Zeitstempelzeichenfolge umgewandelt werden kann.

format

Ein Zeichenkettenliteral, das den Datetime-Mustern von Spark entspricht. Gültige Datetime-Muster finden Sie unter Datetime-Muster für Formatierung und Analyse.

Rückgabetyp

TIMESTAMP

Beispiele

Das folgende Beispiel zeigt die Verwendung der TO_TIMESTAMP-Funktion zur Konvertierung einer TIMESTAMP-Zeichenfolge in eine TIMESTAMP-Zeichenfolge.

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

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

Es ist möglich, den TO_TIMESTAMP-Teil eines Datums zu übergeben. Die übrigen Datumsteile werden auf die Standardwerte gesetzt. Die Uhrzeit ist in der Ausgabe enthalten:

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp
--------------------------
```

```
2017-01-01 00:00:00+00
```

Die folgende SQL-Anweisung konvertiert die Zeichenfolge '2011-12-18 24:38:15 'in einen TIMESTAMP. Das Ergebnis ist ein TIMESTAMP, der auf den nächsten Tag fällt, weil die Anzahl der Stunden mehr als 24 Stunden beträgt:

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

to_timestamp
---------------------
2011-12-19 00:38:15+00
```

YEAR-Funktion

Die YEAR-Funktion ist eine Datumsextraktionsfunktion, die ein Datum oder einen Zeitstempel als Eingabe verwendet und die Jahreskomponente (eine vierstellige Zahl) zurückgibt.

Syntax

```
year(date)
```

Argumente

date

Ein DATE- oder TIMESTAMP-Ausdruck.

Rückgabewert

Die YEAR-Funktion gibt einen INTEGER-Wert zurück.

Beispiel

Im folgenden Beispiel wird die Jahreskomponente (`2016`) aus dem Eingabedatum extrahiert`'2016-07-30'`.

```
SELECT year('2016-07-30');
 2016
```

Im folgenden Beispiel wird die Jahreskomponente aus der `birthday` Spalte der `squirrels` Tabelle extrahiert und die Ergebnisse als Ausgabe der SELECT-Anweisung zurückgegeben. Die Ausgabe

dieser Abfrage ist eine Liste von Jahreswerten, einer für jede Zeile in der `squirrels` Tabelle, die das Geburtsjahr jedes Eichhörnchens darstellt.

```
SELECT year(birthday) FROM squirrels
```

Datumsteile für Datums- oder Zeitstempelfunktionen

Die folgende Tabelle identifiziert die Namen und Abkürzungen von Datumsteilen und Uhrzeitteilen, die als Argumente für die folgenden Funktionen verwendet werden können:

- DATE_ADD
- DATE_DIFF
- DATE_PART
- EXTRACT

Datumsteil oder Uhrzeitteil	Abkürzungen
millennium, millennia	mil, mils
century, centuries	c, cent, cents
decade, decades	dec, decs
Epoche	epoch (unterstützt von EXTRACT)
year, years	y, yr, yrs
quarter, quarters	qtr, qtrs
month, months	mon, mons
week, weeks	w
Tag der Woche	dayofweek, dow, dw, weekday (unterstützt von DATE_PART und Funktion EXTRACT) Gibt eine Ganzzahl von 0–6 aus, beginnend mit Sonntag.

Datumsteil oder Uhrzeitteil	Abkürzungen
	Note Der Datumsteil DOW verhält sich anders als der Datumsteil „Wochentag (D)“ für Datumsteilformatze ichenfolgen. D basiert auf den Ganzzahlen 1–7, wobei die 1 für den Sonntag steht. Weitere Informationen finden Sie unter Datum-/Uhrzeit-Formatzeichenfolgen.
Tag des Jahres	dayofyear, doy, dy, yearday (unterstützt von 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	Unterstützt von EXTRACT nur für Zeitstempel mit Zeitzone (TIMESTAMPTZ).

Abweichungen bei den Ergebnissen mit Sekunden, Millisekunden und Mikrosekunden

Kleinere Differenzen treten auf, wenn verschiedene Datumsfunktionen Sekunden, Millisekunden oder Mikrosekunden als Datumsteile angeben:

- Die Funktion EXTRACT gibt nur für den angegebenen Datumsteilen Ganzzahlen aus, wobei Datumsteile auf höheren und niedrigeren Ebenen ignoriert werden. Wenn der angegebene Datumsteil „Sekunden“ ist, werden Millisekunden und Mikrosekunden in dem Ergebnis nicht berücksichtigt. Wenn der angegebene Datumsteil „Millisekunden“ ist, werden Sekunden und

Mikrosekunden in dem Ergebnis nicht berücksichtigt. Wenn der angegebene Datumsteil „Mikrosekunden“ ist, werden Sekunden und Millisekunden in dem Ergebnis nicht berücksichtigt.

- Die Funktion DATE_PART gibt den vollständigen Sekundenteil des Zeitstempels aus, unabhängig davon, welcher Datumsteil angegeben wurde; dabei wird je nach Bedarf entweder eine Dezimal- oder eine Ganzzahl ausgegeben.

Anmerkungen zu CENTURY, EPOCH, DECADE und MIL

CENTURY oder CENTURIES

AWS Clean Rooms interpretiert ein CENTURY so, dass es mit dem Jahr ## #1 beginnt und mit dem Jahr endet: ###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)
```

EPOCHE

Die AWS Clean Rooms Implementierung von EPOCH erfolgt relativ zu 1970-01-01 00:00:00.000 000 unabhängig von der Zeitzone, in der sich der Cluster befindet. Möglicherweise müssen Sie die Ergebnisse um die Differenz in Stunden verschieben, je nach der Zeitzone, in der sich das Cluster befindet.

DECADE oder DECADES

AWS Clean Rooms interpretiert den DATEPART DECADE oder DECADES auf der Grundlage des gemeinsamen Kalenders. Zum Beispiel: Da der gewöhnliche Kalender mit dem Jahr 1 beginnt, ist die erste Dekade (Dekade 1) 0001-01-01 bis 0009-12-31, und die zweite Dekade (Dekade 2) ist 0010-01-01 bis 0019-12-31. Beispielsweise reicht Dekade 201 von 2000-01-01 bis 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 oder MILS

AWS Clean Rooms interpretiert eine MIL so, dass sie mit dem ersten Tag des Jahres #001 beginnt und mit dem letzten Tag des Jahres endet: #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)
```

Verschlüsselungs- und Entschlüsselungsfunktionen

Verschlüsselungs- und Entschlüsselungsfunktionen helfen SQL-Entwicklern, sensible Daten vor unberechtigtem Zugriff oder Missbrauch zu schützen, indem sie sie zwischen einer lesbaren Klartextform und einer unlesbaren Chiffretextform konvertieren.

AWS Clean Rooms Spark SQL unterstützt die folgenden Verschlüsselungs- und Entschlüsselungsfunktionen:

Themen

AES_ENCRYPT-Funktion

Die AES_ENCRYPT-Funktion wird zum Verschlüsseln von Daten mit dem Advanced Encryption Standard (AES) -Algorithmus verwendet.

Syntax

```
aes_encrypt(expr, key[, mode[, padding[, iv[, aad]]]])
```

Argumente

expr

Der zu verschlüsselnde Binärwert.

key

Die Passphrase, die zum Verschlüsseln der Daten verwendet werden soll.

Schlüssellängen von 16, 24 und 32 Bit werden unterstützt.

Modus

Gibt an, welcher Blockchiffriermodus zum Verschlüsseln von Nachrichten verwendet werden soll.

Gültige Modi: ECB (Electronic CodeBook), GCM (Galois/Counter Mode), CBC (Cipher-Block Chaining).

Polsterung

Gibt an, wie Nachrichten aufgefüllt werden, deren Länge kein Vielfaches der Blockgröße ist.

Gültige Werte: PKCS, NONE, DEFAULT.

Das DEFAULT-Padding bedeutet PKCS (Public Key Cryptography Standards) für ECB, NONE für GCM und PKCS für CBC.

Unterstützte Kombinationen von (Mode, Padding) sind ('ECB', 'PKCS'), ('GCM', 'NONE') und ('CBC', 'PKCS').

iv

Optionaler Initialisierungsvektor (IV). Wird nur für die Modi CBC und GCM unterstützt.

Gültige Werte: 12 Byte lang für GCM und 16 Byte für CBC.

aad

Optionale zusätzliche authentifizierte Daten (AAD). Wird nur für den GCM-Modus unterstützt. Dies kann jede beliebige Eingabe in freier Form sein und muss sowohl für die Verschlüsselung als auch für die Entschlüsselung bereitgestellt werden.

Rückgabetyp

Die Funktion AES_ENCRYPT gibt unter Verwendung von AES im angegebenen Modus mit der angegebenen Auffüllung den verschlüsselten Wert expr zurück.

Beispiele

Das folgende Beispiel zeigt, wie die Spark-SQL-Funktion AES_ENCRYPT verwendet wird, um eine Datenfolge (in diesem Fall das Wort „Spark“) mit einem angegebenen Verschlüsselungsschlüssel sicher zu verschlüsseln. Der resultierende Chiffretext wird dann Base64-kodiert, um das Speichern oder Übertragen zu erleichtern.

```
SELECT base64(aes_encrypt('Spark', 'abcdefghijklmnop'));
  4A5jOAh9FNGwoMeuJukfllrLdHEZxA2DyuSQAWz77dfn
```

Das folgende Beispiel zeigt, wie die Spark-SQL-Funktion AES_ENCRYPT verwendet wird, um eine Datenfolge (in diesem Fall das Wort „Spark“) mit einem angegebenen Verschlüsselungsschlüssel sicher zu verschlüsseln. Der resultierende Chiffretext wird dann im Hexadezimalformat dargestellt, was für Aufgaben wie Datenspeicherung, Übertragung oder Debugging nützlich sein kann.

```
SELECT hex(aes_encrypt('Spark', '0000111122223333'));
 83F16B2AA704794132802D248E6BFD4E380078182D1544813898AC97E709B28A94
```

Das folgende Beispiel zeigt, wie Sie mit der Funktion AES_ENCRYPT von Spark SQL eine Datenfolge (in diesem Fall „Spark SQL“) mithilfe eines angegebenen Verschlüsselungsschlüssels,

Verschlüsselungsmodus und Füllmodus sicher verschlüsseln können. Der resultierende Chiffretext wird dann Base64-kodiert, um das Speichern oder Übertragen zu erleichtern.

```
SELECT base64(aes_encrypt('Spark SQL', '1234567890abcdef', 'ECB', 'PKCS'));
 3lmwu+Mw0H3fi5NDvcu9lg==
```

AES_DECRYPT-Funktion

Die AES_DECRYPT-Funktion wird zum Entschlüsseln von Daten mit dem Advanced Encryption Standard (AES) -Algorithmus verwendet.

Syntax

```
aes_decrypt(expr, key[, mode[, padding[, aad]]])
```

Argumente

expr

Der zu entschlüsselnde Binärwert.

key

Die Passphrase, die zum Entschlüsseln der Daten verwendet werden soll.

Die Passphrase muss mit dem Schlüssel übereinstimmen, der ursprünglich zur Erzeugung des verschlüsselten Werts verwendet wurde, und 16, 24 oder 32 Byte lang sein.

Modus

Gibt an, welcher Blockchiffriermodus zum Entschlüsseln von Nachrichten verwendet werden soll.

Gültige Modi: ECB, GCM, CBC.

Polsterung

Gibt an, wie Nachrichten aufgefüllt werden, deren Länge kein Vielfaches der Blockgröße ist.

Gültige Werte: PKCS, NONE, DEFAULT.

Das DEFAULT-Padding bedeutet PKCS für ECB, NONE für GCM und PKCS für CBC.

aad

Optionale zusätzliche authentifizierte Daten (AAD). Wird nur für den GCM-Modus unterstützt. Dies kann jede beliebige Eingabe in freier Form sein und muss sowohl für die Verschlüsselung als auch für die Entschlüsselung bereitgestellt werden.

Rückgabetyp

Gibt einen entschlüsselten Wert von expr zurück, der AES im Modus mit Auffüllung verwendet.

Beispiele

Das folgende Beispiel zeigt, wie die Spark-SQL-Funktion AES_ENCRYPT verwendet wird, um eine Datenfolge (in diesem Fall das Wort „Spark“) mit einem angegebenen Verschlüsselungsschlüssel sicher zu verschlüsseln. Der resultierende Chiffretext wird dann Base64-kodiert, um das Speichern oder Übertragen zu erleichtern.

```
SELECT base64(aes_encrypt('Spark', 'abcdefghijklmnop'));
  4A5jOAh9FNGwoMeuJukfllrLdHEZxA2DyuSQAWz77dfn
```

Das folgende Beispiel zeigt, wie die Spark-SQL-Funktion AES_DECRYPT verwendet wird, um Daten zu entschlüsseln, die zuvor verschlüsselt und Base64-kodiert wurden. Der Entschlüsselungsprozess erfordert den richtigen Verschlüsselungsschlüssel und die richtigen Parameter (Verschlüsselungsmodus und Füllmodus), um die ursprünglichen Klartextdaten erfolgreich wiederherzustellen.

```
SELECT aes_decrypt(unbase64('3lmwu+Mw0H3fi5NDvcu9lg=='), '1234567890abcdef', 'ECB',
 'PKCS');
 Spark SQL
```

Hash-Funktionen

Eine Hash-Funktion ist eine mathematische Funktion, mit der ein numerischer Eingabewert in einen anderen Wert umgewandelt wird.

AWS Clean Rooms Spark SQL unterstützt die folgenden Hash-Funktionen:

Themen

- MD5 Funktion

- Die Funktion SHA
- SHA1 Funktion
- SHA2 Funktion
- HASH64 xx-Funktion

MD5 Funktion

Verwendet die MD5 kryptografische Hashfunktion, um eine Zeichenfolge variabler Länge in eine 32-stellige Zeichenfolge zu konvertieren, die eine Textdarstellung des Hexadezimalwerts einer 128-Bit-Prüfsumme ist.

Syntax

```
MD5(string)
```

Argumente

string

Eine Zeichenfolge mit variabler Länge.

Rückgabetyp

Die MD5 Funktion gibt eine 32-stellige Zeichenfolge zurück, die eine Textdarstellung des Hexadezimalwerts einer 128-Bit-Prüfsumme ist.

Beispiele

Im folgenden Beispiel wird der 128-Bit-Wert für die Zeichenfolge „AWS Clean Rooms“ gezeigt:

```
select md5('AWS Clean Rooms');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

Die Funktion SHA

Synonym für Funktion. SHA1

Siehe SHA1 Funktion.

SHA1 Funktion

Die SHA1 Funktion verwendet die SHA1 kryptografische Hashfunktion, um eine Zeichenfolge mit variabler Länge in eine 40-stellige Zeichenfolge zu konvertieren, die eine Textdarstellung des Hexadezimalwerts einer 160-Bit-Prüfsumme ist.

Syntax

SHA1 Die Funktion SHAist ein Synonym für.

```
SHA1(string)
```

Argumente

string

Eine Zeichenfolge mit variabler Länge.

Rückgabetyp

Die SHA1 Funktion gibt eine 40-stellige Zeichenfolge zurück, die eine Textdarstellung des Hexadezimalwerts einer 160-Bit-Prüfsumme ist.

Beispiel

Im folgenden Beispiel wird der 160-Bit-Wert für das Wort „AWS Clean Rooms“ zurückgegeben:

```
select sha1('AWS Clean Rooms');
```

SHA2 Funktion

Die SHA2 Funktion verwendet die SHA2 kryptografische Hash-Funktion, um eine Zeichenfolge variabler Länge in eine Zeichenfolge umzuwandeln. Die Zeichenkette ist eine Textdarstellung des hexadezimalen Wertes der Prüfsumme mit der angegebenen Anzahl von Bits.

Syntax

```
SHA2(string, bits)
```

Argumente

string

Eine Zeichenfolge mit variabler Länge.

integer

Die Anzahl der Bits in den Hash-Funktionen. Gültige Werte sind 0 (identisch mit 256), 224, 256, 384 und 512.

Rückgabetyp

Die SHA2 Funktion gibt eine Zeichenfolge zurück, die eine Textdarstellung des Hexadezimalwerts der Prüfsumme ist, oder eine leere Zeichenfolge, wenn die Anzahl der Bits ungültig ist.

Beispiel

Im folgenden Beispiel wird der 256-Bit-Wert für das Wort „AWS Clean Rooms“ zurückgegeben:

```
select sha2('AWS Clean Rooms', 256);
```

HASH64 xx-Funktion

Die Funktion xxhash64 gibt einen 64-Bit-Hashwert der Argumente zurück.

Die Funktion xxhash64 () ist eine nicht-kryptografische Hash-Funktion, die darauf ausgelegt ist, schnell und effizient zu sein. Sie wird häufig in Datenverarbeitungs- und Speicheranwendungen verwendet, bei denen eine eindeutige Kennung für ein Datenelement benötigt wird, der genaue Inhalt der Daten jedoch nicht geheim gehalten werden muss.

Im Kontext einer SQL-Abfrage könnte die Funktion xxhash64 () für verschiedene Zwecke verwendet werden, wie zum Beispiel:

- Generieren eines eindeutigen Bezeichners für eine Zeile in einer Tabelle
- Partitionierung von Daten auf der Grundlage eines Hashwerts
- Implementierung benutzerdefinierter Indizierungs- oder Datenverteilungsstrategien

Der spezifische Anwendungsfall würde von den Anforderungen der Anwendung und den verarbeiteten Daten abhängen.

Syntax

```
xxhash64(expr1, expr2, ...)
```

Argumente

expr1

Ein Ausdruck beliebigen Typs.

expr2

Ein Ausdruck beliebigen Typs.

Rückgabewert

Gibt einen 64-Bit-Hashwert der Argumente zurück (BIGINT). Der Hash-Seed ist 42.

Beispiel

Das folgende Beispiel generiert einen 64-Bit-Hashwert (5602566077635097486) auf der Grundlage der bereitgestellten Eingabe. Das erste Argument ist ein Zeichenkettenwert, in diesem Fall das Wort „Spark“. Das zweite Argument ist ein Array, das den einzelnen Integer-Wert 123 enthält. Das dritte Argument ist ein Integer-Wert, der den Startwert für die Hash-Funktion darstellt.

```
SELECT xxhash64('Spark', array(123), 2);
 5602566077635097486
```

Hyperloglog-Funktionen

Die HyperLogLog (HLL) -Funktionen in SQL bieten eine Möglichkeit, die Anzahl der eindeutigen Elemente (Kardinalität) in einem großen Datensatz effizient zu schätzen, selbst wenn der tatsächliche Satz eindeutiger Elemente nicht gespeichert ist.

Die Hauptvorteile der Verwendung von HLL-Funktionen sind:

- Speichereffizienz: HLL-Skizzen benötigen viel weniger Speicherplatz als das Speichern des gesamten Satzes einzigartiger Elemente, sodass sie für große Datensätze geeignet sind.
- Verteiltes Rechnen: HLL-Skizzen können über mehrere Datenquellen oder Verarbeitungsknoten hinweg kombiniert werden, was eine effiziente Schätzung der verteilten eindeutigen Anzahl ermöglicht.

- Ungefähre Ergebnisse: HLL bietet eine ungefähre Schätzung der individuellen Anzahl mit einem einstellbaren Kompromiss zwischen Genauigkeit und Speicherverbrauch (über den Präzisionsparameter).

Diese Funktionen sind besonders nützlich in Szenarien, in denen Sie die Anzahl der einzelnen Elemente schätzen müssen, z. B. in Analyse-, Data Warehousing- und Echtzeit-Stream-Verarbeitungsanwendungen.

AWS Clean Rooms unterstützt die folgenden HLL-Funktionen.

Themen

HLL_SKETCH_AGG-Funktion

Die Aggregatfunktion HLL_SKETCH_AGG erstellt eine HLL-Skizze aus den Werten in der angegebenen Spalte. Sie gibt einen HLLSKETCH-Datentyp zurück, der die Werte der Eingabeausdrücke kapselt.

Die HLL_SKETCH_AGG-Agg-Aggregatfunktion funktioniert mit jedem Datentyp und ignoriert NULL-Werte.

Wenn keine Zeilen in einer Tabelle vorhanden sind oder alle Zeilen NULL sind, enthält die resultierende Skizze keine Index-Wert-Paare wie zum Beispiel `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}`.

Syntax

```
HLL_SKETCH_AGG (aggregate_expression[, lgConfigK ] )
```

Argument

aggregate_expression

Jeder Ausdruck vom Typ INT, BIGINT, STRING oder BINARY, für den eine eindeutige Zählung erfolgt. Alle NULL Werte werden ignoriert.

LgConfigK

Eine optionale INT-Konstante zwischen 4 und 21 (einschließlich) mit dem Standardwert 12. Die Log-Base-2 von K, wobei K die Anzahl der Buckets oder Slots für die Skizze ist.

Rückgabetyp

Die Funktion HLL_SKETCH_AGG gibt einen BINARY-Puffer zurück, der nicht NULL ist und die Skizze enthält, die aufgrund der Verwendung und Aggregation aller Eingabewerte in der HyperLogLog Aggregationsgruppe berechnet wurde.

Beispiele

In den folgenden Beispielen wird der Algorithmus HyperLogLog (HLL) verwendet, um die eindeutige Anzahl der Werte in der Spalte zu schätzen. `col` Die `hll_sketch_agg(col, 12)` Funktion aggregiert die Werte in der Spalte col und erstellt so eine HLL-Skizze mit einer Genauigkeit von 12. Die `hll_sketch_estimate()` Funktion wird dann verwendet, um die eindeutige Anzahl von Werten auf der Grundlage der generierten HLL-Skizze zu schätzen. Das Endergebnis der Abfrage ist 3, was der geschätzten eindeutigen Anzahl von Werten in der `col` Spalte entspricht. In diesem Fall sind die unterschiedlichen Werte 1, 2 und 3.

```
SELECT hll_sketch_estimate(hll_sketch_agg(col, 12))
    FROM VALUES (1), (1), (2), (2), (3) tab(col);
  3
```

Im folgenden Beispiel wird auch der HLL-Algorithmus verwendet, um die eindeutige Anzahl von Werten in der `col` Spalte zu schätzen, aber es wird kein Genauigkeitswert für die HLL-Skizze angegeben. In diesem Fall wird die Standardgenauigkeit von 14 verwendet. Die `hll_sketch_agg(col)` Funktion verwendet die Werte in der `col` Spalte und erstellt eine HyperLogLog (HLL-) Skizze, bei der es sich um eine kompakte Datenstruktur handelt, mit der die unterschiedliche Anzahl von Elementen geschätzt werden kann. Die `hll_sketch_estimate(hll_sketch_agg(col))` Funktion berechnet anhand der im vorherigen Schritt erstellten HLL-Skizze eine Schätzung der unterschiedlichen Anzahl von Werten in der Spalte. `col` Das Endergebnis der Abfrage ist 3, was der geschätzten eindeutigen Anzahl von Werten in der `col` Spalte entspricht. In diesem Fall sind die unterschiedlichen Werte 1, 2 und 3.

```
SELECT hll_sketch_estimate(hll_sketch_agg(col))
FROM VALUES (1), (1), (2), (2), (3) tab(col);
3
```

Funktion HLL_SKETCH_ESTIMATE

Die Funktion HLL_SKETCH_ESTIMATE verwendet eine HLL-Skizze und schätzt die Anzahl der eindeutigen Elemente, die durch die Skizze dargestellt werden. Sie verwendet den HyperLogLog (HLL) -Algorithmus, um eine probabilistische Annäherung an die Anzahl der Einzelwerte in einer bestimmten Spalte zu zählen. Dabei wird eine binäre Darstellung, ein sogenannter Sketch-Puffer, verwendet, der zuvor von der HLL_SKETCH_AGG-Funktion generiert wurde, und das Ergebnis als große Ganzzahl zurückgegeben.

Der HLL-Skizzieralgorithmus bietet eine effiziente Methode zur Schätzung der Anzahl eindeutiger Elemente, selbst bei großen Datensätzen, ohne dass der gesamte Satz von Einzelwerten gespeichert werden muss.

Mit den `hll_union_agg` Funktionen `hll_union` und können Skizzen auch miteinander kombiniert werden, indem sie diese Puffer als Eingaben verwenden und zusammenführen.

Syntax

```
HLL_SKETCH_ESTIMATE (hllsketch_expression)
```

Argument

hllsketch_expression

Ein `BINARY` Ausdruck, der eine von HLL_SKETCH_AGG generierte Skizze enthält

Rückgabetyp

Die Funktion HLL_SKETCH_ESTIMATE gibt einen BIGINT-Wert zurück, der der ungefähren Anzahl unterschiedlicher Werte entspricht, die durch die Eingabeskizze dargestellt wird.

Beispiele

In den folgenden Beispielen wird der Skizzieralgorithmus HyperLogLog (HLL) verwendet, um die Kardinalität (eindeutige Anzahl) der Werte in der Spalte zu schätzen. `col` Die `hll_sketch_agg(col, 12)` Funktion verwendet die `col` Spalte und erstellt eine HLL-Skizze mit einer Genauigkeit von 12 Bit. Die HLL-Skizze ist eine ungefähre Datenstruktur, mit der die Anzahl der eindeutigen Elemente in einem Satz effizient geschätzt werden kann. Die `hll_sketch_estimate()` Funktion verwendet die HLL-Skizze, die von erstellt wurde,

`hll_sketch_agg` und schätzt die Kardinalität (eindeutige Anzahl) der durch die Skizze repräsentierten Werte. Die `FROM VALUES (1), (1), (2), (2), (3) tab(col);` generiert einen Testdatensatz mit 5 Zeilen, wobei die `col` Spalte die Werte 1, 1, 2, 2 und 3 enthält. Das Ergebnis dieser Abfrage ist die geschätzte eindeutige Anzahl der Werte in der `col` Spalte, die 3 ist.

```
SELECT hll_sketch_estimate(hll_sketch_agg(col, 12))
    FROM VALUES (1), (1), (2), (2), (3) tab(col);
  3
```

Der Unterschied zwischen dem folgenden und dem vorherigen Beispiel besteht darin, dass der Präzisionsparameter (12 Bit) im `hll_sketch_agg` Funktionsaufruf nicht angegeben ist. In diesem Fall wird die Standardgenauigkeit von 14 Bit verwendet, was im Vergleich zum vorherigen Beispiel, bei dem eine Genauigkeit von 12 Bit verwendet wurde, zu einer genaueren Schätzung der Anzahl von Einzelstücken führen kann.

```
SELECT hll_sketch_estimate(hll_sketch_agg(col))
FROM VALUES (1), (1), (2), (2), (3) tab(col);
3
```

HLL_UNION-Funktion

Die Funktion HLL_UNION kombiniert zwei HLL-Skizzen zu einer einzigen, einheitlichen Skizze. Sie verwendet den HyperLogLog (HLL) -Algorithmus, um zwei Skizzen zu einer einzigen Skizze zu kombinieren. Abfragen können die resultierenden Puffer verwenden, um mit der Funktion ungefähre Einzelzahlen als lange Ganzzahlen zu berechnen. `hll_sketch_estimate`

Syntax

```
HLL_UNION (( expr1, expr2 [, allowDifferentLgConfigK ] ))
```

Argument

ExprN

Ein `BINARY` Ausdruck, der eine von HLL_SKETCH_AGG generierte Skizze enthält.

allowDifferentLgConfigK

Ein optionaler BOOLESCHER Ausdruck, der steuert, ob das Zusammenführen von zwei Skizzen mit unterschiedlichen LgConfigK-Werten zulässig ist. Der Standardwert ist `false`.

Rückgabetyp

Die Funktion HLL_UNION gibt einen BINARY-Puffer zurück, der die HyperLogLog Skizze enthält, die als Ergebnis der Kombination der Eingabeausdrücke berechnet wurde. Wenn der `allowDifferentLgConfigK` Parameter gleich ist`true`, verwendet die Ergebnisskizze den kleineren der beiden angegebenen Werte. `lgConfigK`

Beispiele

In den folgenden Beispielen wird der Skizzieralgorithmus HyperLogLog (HLL) verwendet, um die eindeutige Anzahl von Werten in zwei Spalten `col1` und in einem `col2` Datensatz zu schätzen.

Die `hll_sketch_agg(col1)` Funktion erstellt eine HLL-Skizze für die Einzelwerte in der Spalte. `col1`

Die `hll_sketch_agg(col2)` Funktion erstellt eine HLL-Skizze für die Einzelwerte in der Spalte col2.

Die `hll_union(...)` Funktion kombiniert die beiden in den Schritten 1 und 2 erstellten HLL-Skizzen zu einer einzigen, einheitlichen HLL-Skizze.

Die `hll_sketch_estimate(...)` Funktion verwendet die kombinierte HLL-Skizze und schätzt die eindeutige Anzahl der Werte für sowohl als auch. `col1` `col2`

Die `FROM VALUES` Klausel generiert einen Testdatensatz mit 5 Zeilen, der die Werte 1, 1, 2, 2 und 3 sowie die Werte 4, 4, 5, 5 und 6 `col2` enthält. `col1`

Das Ergebnis dieser Abfrage ist die geschätzte eindeutige Anzahl von Werten für beide `col1` und`col2`, die 6 ist. Der HLL-Skizzieralgorithmus bietet eine effiziente Methode zur Schätzung der Anzahl einzigartiger Elemente, selbst bei großen Datensätzen, ohne dass der gesamte Satz von Einzelwerten gespeichert werden muss. In diesem Beispiel wird die `hll_union` Funktion verwendet, um die HLL-Skizzen aus den beiden Spalten zu kombinieren, sodass die eindeutige Anzahl für den gesamten Datensatz geschätzt werden kann und nicht nur für jede Spalte einzeln.

```
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
```

Der Unterschied zwischen dem folgenden und dem vorherigen Beispiel besteht darin, dass der Präzisionsparameter (12 Bit) im `hll_sketch_agg` Funktionsaufruf nicht angegeben ist. In diesem Fall wird die Standardgenauigkeit von 14 Bit verwendet, was im Vergleich zum vorherigen Beispiel, bei dem eine Genauigkeit von 12 Bit verwendet wurde, zu einer genaueren Schätzung der Anzahl von Einzelstücken führen kann.

```
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);
```

HLL_UNION_AGG-Funktion

Die Funktion HLL_UNION_AGG kombiniert mehrere HLL-Skizzen zu einer einzigen, einheitlichen Skizze. Sie verwendet den HyperLogLog (HLL) -Algorithmus, um eine Gruppe von Skizzen zu einer einzigen zu kombinieren. Abfragen können die resultierenden Puffer verwenden, um ungefähre Einzelzahlen mit der Funktion zu berechnen. `hll_sketch_estimate`

Syntax

```
HLL_UNION_AGG ( expr [, allowDifferentLgConfigK ] )
```

Argument

expr

Ein `BINARY` Ausdruck, der eine von HLL_SKETCH_AGG generierte Skizze enthält.

allowDifferentLgConfigK

Ein optionaler BOOLESCHER Ausdruck, der steuert, ob das Zusammenführen von zwei Skizzen mit unterschiedlichen LgConfigK-Werten zulässig ist. Der Standardwert ist `false`.

Rückgabetyp

Die Funktion HLL_UNION_AGG gibt einen BINARY-Puffer zurück, der die HyperLogLog Skizze enthält, die als Ergebnis der Kombination der Eingabeausdrücke derselben Gruppe berechnet wurde. Wenn der `allowDifferentLgConfigK` Parameter gleich ist`true`, verwendet die Ergebnisskizze den kleineren der beiden angegebenen Werte. `lgConfigK`

Beispiele

In den folgenden Beispielen wird der Skizzieralgorithmus HyperLogLog (HLL) verwendet, um die eindeutige Anzahl von Werten in mehreren HLL-Skizzen zu schätzen.

Im ersten Beispiel wird die eindeutige Anzahl von Werten in einem Datensatz geschätzt.

```
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
```

Die innere Abfrage erstellt zwei HLL-Skizzen:

- Die erste SELECT-Anweisung erstellt eine Skizze aus einem einzelnen Wert von 1.
- Die zweite SELECT-Anweisung erstellt eine Skizze aus einem anderen Einzelwert von 1, jedoch mit einer Genauigkeit von 20.

Die äußere Abfrage verwendet die Funktion HLL_UNION_AGG, um die beiden Skizzen zu einer einzigen Skizze zu kombinieren. Anschließend wendet sie die Funktion HLL_SKETCH_ESTIMATE auf diese kombinierte Skizze an, um die eindeutige Anzahl von Werten zu schätzen.

Das Ergebnis dieser Abfrage ist die geschätzte eindeutige Anzahl der Werte in der Spalte, d. h. `col` 1 Das bedeutet, dass die beiden Eingabewerte von 1 als eindeutig betrachtet werden, obwohl sie denselben Wert haben.

Das zweite Beispiel beinhaltet einen anderen Präzisionsparameter für die HLL_UNION_AGG-Funktion. In diesem Fall werden beide HLL-Skizzen mit einer Genauigkeit von 14 Bit erstellt, sodass sie erfolgreich mit dem Parameter kombiniert werden können. `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
```

Das Endergebnis der Abfrage ist die geschätzte eindeutige Anzahl, was in diesem Fall auch der Fall ist. 1 Das bedeutet, dass die beiden Eingabewerte von 1 als eindeutig betrachtet werden, obwohl sie denselben Wert haben.

JSON-Funktionen

Wenn Sie einen vergleichsweise kleinen Satz von Schlüssel-Wert-Paaren speichern müssen, können Sie vielleicht Platz sparen, indem Sie die Daten im JSON-Format speichern. Da JSON-Zeichenfolgen in einer einzigen Spalte gespeichert werden können, kann die Verwendung von JSON effizienter als das Speichern Ihrer Daten im Tabellenformat sein.

Example

Nehmen wir zum Beispiel an, Sie haben eine Tabelle mit geringer Dichte, in der Sie viele Spalten benötigen, um alle möglichen Attribute vollständig darzustellen. Die meisten Spaltenwerte sind jedoch für eine bestimmte Zeile oder Spalte NULL. Wenn Sie JSON als Speicher verwenden, können Sie die Daten für eine Zeile möglicherweise in Schlüssel-Wert-Paaren in einer einzigen JSON-Zeichenfolge speichern und die spärlich gefüllten Tabellenspalten eliminieren.

Zusätzlich können Sie JSON-Zeichenfolgen leicht ändern, sodass diese weitere Schlüssel:Wert-Paare speichern, ohne einer Tabelle Spalten hinzufügen zu müssen.

Sie sollten JSON nur in bestimmten Fällen verwenden. JSON ist keine gute Wahl für das Speichern größerer Datensätze, da JSON beim Speichern unterschiedlicher Daten in einer einzigen Spalte nicht die Spaltenspeicherarchitektur verwendet. AWS Clean Rooms

JSON verwendet UTF-8-kodierte Textzeichenfolgen. Daher können JSON-Zeichenfolgen als CHAR- oder VARCHAR-Datentypen gespeichert werden. Sie verwenden VARCHAR, wenn die Zeichenfolgen Multibyte-Zeichen enthalten.

JSON-Zeichenfolgen müssen ein korrektes JSON-Format aufweisen, das den folgenden Regeln entspricht:

- Der JSON-Wert kann auf Stammverzeichnisebene ein JSON-Objekt oder ein JSON-Array sein. Ein JSON-Objekt ist ein nicht geordneter Satz von durch Komma getrennten Schlüssel:Wert-Paaren, eingeschlossen in geschweiften Klammern.

  Beispiel: `{"one":1, "two":2}`
- Ein JSON-Array ist ein geordneter Satz von durch Komma getrennten Werten, eingeschlossen in eckigen Klammern.

  Ein Beispiel ist folgendes: `["first", {"one":1}, "second", 3, null]`
- JSON-Arrays verwenden einen nullbasierten Index. Das erste Element in einem Array befindet sich an Position 0. In einem Schlüssel:Wert-Paar in JSON ist der Schlüssel eine Zeichenfolge in doppelten Anführungszeichen.
- Ein JSON-Wert kann jeder der folgenden Werte sein:
  - JSON-Objekt
  - JSON-Array
  - Zeichenfolge in doppelten Anführungszeichen
  - Zahl (Ganzzahl und Gleitkommazahl)
  - Boolesch
  - Null
- Leere Objekte und leere Arrays sind gültige JSON-Werte.
- JSON-Felder unterscheiden zwischen Groß- und Kleinschreibung.
- Leerzeichen zwischen JSON-Strukturelementen (wie `{ }, [ ]`) werden ignoriert.

Themen

Funktion GET_JSON_OBJECT

Die Funktion GET_JSON_OBJECT extrahiert ein JSON-Objekt aus. `path`

Syntax

```
get_json_object(json_txt, path)
```

Argumente

json_txt

Ein STRING-Ausdruck, der wohlgeformtes JSON enthält.

path

Ein STRING-Literal mit einem wohlgeformten JSON-Pfadausdruck.

Rückgabewert

Gibt einen STRING zurück.

Ein NULL-Wert wird zurückgegeben, wenn das Objekt nicht gefunden werden kann.

Beispiel

Das folgende Beispiel extrahiert einen Wert aus einem JSON-Objekt. Das erste Argument ist eine JSON-Zeichenfolge, die ein einfaches Objekt mit einem einzigen Schlüssel-Wert-Paar darstellt. Das zweite Argument ist ein JSON-Pfadausdruck. Das $ Symbol steht für die Wurzel des JSON-Objekts, und der `.a` Teil gibt an, dass wir den Wert extrahieren möchten, der dem Schlüssel "a" zugeordnet ist. Die Ausgabe der Funktion ist 'b', das ist der Wert, der dem Schlüssel "a" im JSON-Eingabeobjekt zugeordnet ist.

```
SELECT get_json_object('{"a":"b"}', '$.a');
 b
```

TO_JSON-Funktion

Die TO_JSON-Funktion konvertiert einen Eingabeausdruck in eine JSON-Zeichenfolgendarstellung. Die Funktion verarbeitet die Konvertierung verschiedener Datentypen (wie Zahlen, Zeichenketten und Boolesche Werte) in die entsprechenden JSON-Repräsentationen.

Die TO_JSON-Funktion ist nützlich, wenn Sie strukturierte Daten (wie Datenbankzeilen oder JSON-Objekte) in ein portableres, sich selbst beschreibendes Format wie JSON konvertieren müssen. Dies kann besonders hilfreich sein, wenn Sie mit anderen Systemen oder Diensten interagieren müssen, die Daten im JSON-Format erwarten.

Syntax

```
to_json(expr[, options])
```

Argumente

expr

Der Eingabeausdruck, den Sie in eine JSON-Zeichenfolge konvertieren möchten. Es kann ein Wert, eine Spalte oder ein anderer gültiger SQL-Ausdruck sein.

options

Ein optionaler Satz von Konfigurationsoptionen, mit denen der JSON-Konvertierungsprozess angepasst werden kann. Diese Optionen können Dinge wie die Behandlung von Nullwerten, die Darstellung numerischer Werte und die Behandlung von Sonderzeichen beinhalten.

Rückgabewert

Gibt eine JSON-Zeichenfolge mit einem bestimmten Strukturwert zurück

Beispiele

Das folgende Beispiel konvertiert eine benannte Struktur (eine Art strukturierter Daten) in eine JSON-Zeichenfolge. Das erste Argument (`named_struct('a', 1, 'b', 2)` () ist der Eingabeausdruck, der an die `to_json()` Funktion übergeben wird. Es erstellt eine benannte Struktur mit zwei Feldern: „a“ mit einem Wert von 1 und „b“ mit einem Wert von 2. Die Funktion to_json () verwendet die benannte Struktur als Argument und konvertiert sie in eine JSON-Zeichenkettendarstellung. Die Ausgabe ist eine gültige JSON-Zeichenfolge`{"a":1,"b":2}`, die die benannte Struktur darstellt.

```
SELECT to_json(named_struct('a', 1, 'b', 2));
 {"a":1,"b":2}
```

Das folgende Beispiel konvertiert eine benannte Struktur, die einen Zeitstempelwert enthält, in eine JSON-Zeichenfolge mit einem benutzerdefinierten Zeitstempelformat. Das erste Argument (`named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd'))`) erstellt eine benannte Struktur mit einem einzigen Feld „time“, das den Zeitstempelwert enthält. Das zweite Argument (`map('timestampFormat', 'dd/MM/yyyy')`) erstellt eine Map (Schlüssel-Wert-Wörterbuch) mit einem einzigen Schlüssel-Wert-Paar, wobei der Schlüssel 'TimeStampFormat' und der Wert '' ist. 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 Die Ausgabe

ist eine JSON-Zeichenfolge mit einem einzigen Feld „Zeit“`{"time":"26/08/2015"}`, das den Zeitstempelwert im gewünschten Format „“ enthält. 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"}
```

Mathematische Funktionen

In diesem Abschnitt werden die mathematischen Operatoren und Funktionen beschrieben, die in AWS Clean Rooms Spark SQL unterstützt werden.

Themen

- RAND-Funktion
- Die Funktion RANDOM
- Die Funktion ROUND
- Die Funktion SIGN
- Die Funktion SIN
- Die Funktion SQRT
- Die Funktion TRUNC

Symbole für mathematische Operatoren

In der folgenden Tabelle werden die unterstützten mathematischen Operatoren aufgeführt.

Unterstützte Operatoren

Operator	Beschreibung	Beispiel	Ergebnis
+	Addition	2 + 3	5
-	Subtraktion	2 - 3	-1
*	Multiplikation	2 * 3	6
/	Division	4 / 2	2
%	Modulo	5 % 4	1
^	Potenzierung	2,0 ^ 3,0	8

Beispiele

Berechnet die gezahlte Provision zuzüglich einer Bearbeitungsgebühr von 2,00$ für eine bestimmte Transaktion:

```
select commission, (commission + 2.00) as comm
```

```
from sales where salesid=10000;

commission | comm
-----------+-------
28.05      | 30.05
(1 row)
```

Berechnet 20 Prozent des Verkaufspreises für eine bestimmte Transaktion:

```
select pricepaid, (pricepaid * .20) as twentypct
from sales where salesid=10000;

pricepaid | twentypct
----------+-----------
187.00    |    37.400
(1 row)
```

Voraussichtliche Ticketverkäufe auf der Basis eines kontinuierlichen Wachstumsmusters. In diesem Beispiel gibt die Unterabfrage die Anzahl der Tickets zurück, die 2008 verkauft wurden. Dieses Ergebnis wird exponentiell mit einer kontinuierlichen Wachstumsrate von 5 Prozent über einen Zeitraum von 10 Jahren multipliziert.

```
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)
```

Ermitteln Sie den gezahlten Gesamtpreis und die Provision für Verkäufe mit einer Datum-ID, die größer oder gleich 2.000 ist. Anschließend wird die Gesamtprovision vom gezahlten Gesamtpreis abgezogen.

```
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)
```

Funktion ABS

ABS berechnet den absoluten Wert einer Zahl, wobei diese Zahl ein Literal oder ein Ausdruck sein kann, der zu einer Zahl ausgewertet wird.

Syntax

```
ABS (number)
```

Argumente

number (Zahl

Zahl oder Ausdruck, der zu einer Zahl ausgewertet wird. Dabei kann es sich um den Typ SMALLINT, INTEGER, BIGINT FLOAT4, DECIMAL oder FLOAT8 handeln.

Rückgabetyp

ABS gibt denselben Datentyp wie sein Argument zurück.

Beispiele

Berechnet den absoluten Wert von -38:

```
select abs (-38);
abs
-------
38
```

```
(1 row)
```

Berechnet den absoluten Wert von (14 - 76):

```
select abs (14-76);
abs
-------
62
(1 row)
```

Die Funktion ACOS

ACOS ist eine trigonometrische Funktion, die den Arcuscosinus einer Zahl zurückgibt. Der Rückgabewert wird in Radianten ausgedrückt und liegt zwischen `0` und `PI`.

Syntax

```
ACOS(number)
```

Argumente

number (Zahl

Der Eingabeparameter ist eine `DOUBLE PRECISION`-Zahl.

Rückgabetyp

`DOUBLE PRECISION`

Beispiele

Verwenden Sie das folgende Beispiel, um den Arcuscosinus von `-1` zurückzugeben.

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Die Funktion ASIN

ASIN ist eine trigonometrische Funktion, die den Arcussinus einer Zahl zurückgibt. Der Rückgabewert wird in Radianten ausgedrückt und liegt zwischen `PI/2` und `-PI/2`.

Syntax

```
ASIN(number)
```

Argumente

number (Zahl

Der Eingabeparameter ist eine `DOUBLE PRECISION`-Zahl.

Rückgabetyp

`DOUBLE PRECISION`

Beispiele

Verwenden Sie das folgende Beispiel, um den Arcussinus von 1 zurückzugeben.

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

Die Funktion ATAN

ATAN ist eine trigonometrische Funktion, die den Arcustangens einer Zahl zurückgibt. Der Rückgabewert wird in Radianten ausgedrückt und liegt zwischen `-PI` und `PI`.

Syntax

```
ATAN(number)
```

Argumente

number (Zahl

Der Eingabeparameter ist eine DOUBLE PRECISION-Zahl.

Rückgabetyp

DOUBLE PRECISION

Beispiele

Verwenden Sie das folgende Beispiel, um den Arcustangens von 1 zurückzugeben und mit 4 multipliziert.

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

ATAN2 Funktion

ATAN2 ist eine trigonometrische Funktion, die den Arkustangens einer Zahl geteilt durch eine andere Zahl zurückgibt. Der Rückgabewert wird in Radianten ausgedrückt und liegt zwischen PI/2 und -PI/2.

Syntax

```
ATAN2(number1, number2)
```

Argumente

number1

Eine DOUBLE PRECISION-Zahl.

number2

Eine DOUBLE PRECISION-Zahl.

Rückgabetyp

DOUBLE PRECISION

Beispiele

Verwenden Sie das folgende Beispiel, um den Arcustangens von 2/2 zurückzugeben und mit 4 multipliziert.

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Die Funktion CBRT

Die CBRT-Funktion ist eine mathematische Funktion, die die Kubikwurzel einer Zahl berechnet.

Syntax

```
CBRT (number)
```

Argument

CBRT hat eine DOUBLE PRECISION-Zahl als Argument.

Rückgabetyp

CBRT gibt eine DOUBLE PRECISION-Zahl zurück.

Beispiele

Berechnet die Kubikwurzel der Provision, die für eine bestimmte Transaktion gezahlt wurde:

```
select cbrt(commission) from sales where salesid=10000;

cbrt
------------------
```

```
3.03839539048843
(1 row)
```

Die Funktion CEILING (oder CEIL)

Die CEILING- oder CEIL-Funktion wird verwendet, um eine Zahl auf die nächste ganze Zahl aufzurunden. (Die Die Funktion FLOOR rundet eine Zahl auf die nächste ganze Zahl ab.)

Syntax

```
CEIL | CEILING(number)
```

Argumente

number (Zahl

Die Zahl oder der Ausdruck, der zu einer Zahl ausgewertet wird. Dabei kann es sich um den Typ SMALLINT, INTEGER, BIGINT FLOAT4, DECIMAL oder handeln. FLOAT8

Rückgabetyp

CEILING und CEIL geben denselben Datentyp wie ihr Argument zurück.

Beispiel

Berechnet die Decke der Provision, die für eine bestimmte Verkaufstransaktion gezahlt wird:

```
select ceiling(commission) from sales
where salesid=10000;

ceiling
---------
29
(1 row)
```

Die Funktion COS

COS ist eine trigonometrische Funktion, die den Cosinus einer Zahl zurückgibt. Der Rückgabewert wird in Radianten ausgedrückt und liegt zwischen -1 und 1, jeweils einschließlich.

Syntax

```
COS(double_precision)
```

Argument

number (Zahl

Der Eingabeparameter ist eine Doppelpräzisionszahl.

Rückgabetyp

Die COS-Funktion gibt eine Doppelpräzisionszahl zurück.

Beispiele

Im folgenden Beispiel wird der Cosinus von 0 zurückgegeben:

```
select cos(0);
cos
-----
1
(1 row)
```

Im folgenden Beispiel wird der Cosinus von PI zurückgegeben:

```
select cos(pi());
cos
-----
-1
(1 row)
```

Die Funktion COT

COT ist eine trigonometrische Funktion, die den Kotangens einer Zahl zurückgibt. Der Eingabeparameter darf nicht null sein.

Syntax

```
COT(number)
```

Argument

number (Zahl

Der Eingabeparameter ist eine DOUBLE PRECISION-Zahl.

Rückgabetyp

DOUBLE PRECISION

Beispiele

Verwenden Sie das folgende Beispiel, um den Kotangens von 1 zurückzugeben.

```
SELECT COT(1);

+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

Die Funktion DEGREES

Konvertiert einen Winkel in Radianten in die Entsprechung in Grad.

Syntax

```
DEGREES(number)
```

Argument

number (Zahl

Der Eingabeparameter ist eine DOUBLE PRECISION-Zahl.

Rückgabetyp

DOUBLE PRECISION

Beispiel

Verwenden Sie das folgende Beispiel, um die Entsprechung in Grad des Radianten 0,5 zurückzugeben.

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
+-------------------+
```

Verwenden Sie das folgende Beispiel, um PI-Radianten in Grad zu konvertieren.

```
SELECT DEGREES(pi());

+---------+
| degrees |
+---------+
|     180 |
+---------+
```

DIV-Funktion

Der DIV-Operator gibt den integralen Teil der Division der Dividende durch den Divisor zurück.

Syntax

```
dividend div divisor
```

Argumente

Dividende

Ein Ausdruck, der zu einer Zahl oder einem Intervall ausgewertet wird.

Divisor

Ein passender Intervalltyp, wenn `dividend` es sich um ein Intervall handelt, andernfalls um eine Zahl.

Rückgabetyp

`BIGINT`

Beispiele

Im folgenden Beispiel werden zwei Spalten aus der Eichhörnchen-Tabelle ausgewählt: die `id` Spalte, die den eindeutigen Bezeichner für jedes Eichhörnchen enthält, und eine `calculated` Spalte`age div 2`, die die ganzzahlige Division der Altersspalte durch 2 darstellt. `age div 2`Bei der Berechnung wird die `age` Spalte durch eine Ganzzahl dividiert, wodurch das Alter auf die nächste gerade Ganzzahl abgerundet wird. Wenn die `age` Spalte beispielsweise Werte wie 3, 5, 7 und 10 enthält, würde die `age div 2` Spalte jeweils die Werte 1, 2, 3 und 5 enthalten.

```
SELECT id, age div 2 FROM squirrels
```

Diese Abfrage kann in Szenarien nützlich sein, in denen Sie Daten nach Altersbereichen gruppieren oder analysieren müssen und Sie die Alterswerte vereinfachen möchten, indem Sie sie auf die nächste gerade Ganzzahl abrunden. Die resultierende Ausgabe würde für jedes Eichhörnchen in der Tabelle das Alter `id` und das `squirrels` Alter geteilt durch 2 ergeben.

Die Funktion EXP

Die EXP-Funktion implementiert die Exponentialfunktion für einen numerischen Ausdruck, oder die Basis des natürlichen Logarithmus, `e`, potenziert mit dem Ausdruck. Die EXP-Funktion ist die Umkehrung von Die Funktion LN.

Syntax

```
EXP (expression)
```

Argument

expression

Der Ausdruck muss den Datentyp INTEGER, DECIMAL oder DOUBLE PRECISION haben.

Rückgabetyp

EXP gibt eine DOUBLE PRECISION-Zahl zurück.

Beispiel

Die EXP-Funktion wird verwendet, um Ticketverkäufe auf der Basis eines kontinuierlichen Wachstumsmusters zu prognostizieren. In diesem Beispiel gibt die Unterabfrage die Anzahl der Tickets zurück, die 2008 verkauft wurden. Dieses Ergebnis wird mit dem Ergebnis der EXP-Funktion multipliziert, das eine kontinuierliche Wachstumsrate von 7 % über 10 Jahre angibt.

```
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)
```

Die Funktion FLOOR

Die FLOOR-Funktion rundet eine Zahl auf die nächste ganze Zahl ab.

Syntax

```
FLOOR (number)
```

Argument

number (Zahl

Die Zahl oder der Ausdruck, der zu einer Zahl ausgewertet wird. Dabei kann es sich um den Typ SMALLINT, INTEGER, BIGINT FLOAT4, DECIMAL oder handeln. FLOAT8

Rückgabetyp

FLOOR gibt denselben Datentyp wie sein Argument zurück.

Beispiel

Das Beispiel zeigt den Wert der Provision, die für eine bestimmte Verkaufstransaktion vor und nach Verwendung der FLOOR-Funktion bezahlt wurde.

```
select commission from sales
```

```
where salesid=10000;

floor
-------
28.05
(1 row)

select floor(commission) from sales
where salesid=10000;

floor
-------
28
(1 row)
```

Die Funktion LN

Die LN-Funktion gibt den natürlichen Logarithmus des Eingabeparameters zurück.

Syntax

```
LN(expression)
```

Argument

expression

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird.

> **Note**
>
> Diese Funktion gibt für einige Datentypen einen Fehler zurück, wenn der Ausdruck auf eine AWS Clean Rooms vom Benutzer erstellte Tabelle oder eine AWS Clean Rooms STL- oder STV-Systemtabelle verweist.

Ausdrücke mit den folgenden Datentypen führen zu einem Fehler, wenn sie eine benutzererstellte oder eine Systemtabelle referenzieren.

- BOOLEAN
- CHAR

- DATUM
- DECIMAL oder NUMERIC
- TIMESTAMP
- VARCHAR

Ausdrücke mit den folgenden Datentypen werden für benutzererstellte und STL- oder STV-Systemtabellen erfolgreich ausgeführt:

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

Rückgabetyp

Die LN-Funktion gibt denselben Typ wie der Ausdruck zurück.

Beispiel

Im folgenden Beispiel wird der natürliche Logarithmus bzw. Basis-e-Logarithmus der Zahl 2,718281828 zurückgegeben:

```
select ln(2.718281828);
ln
--------------------
0.9999999998311267
(1 row)
```

Beachten Sie, dass die Antwort beinahe gleich 1 ist.

In diesem Beispiel wird der natürliche Logarithmus der Werte in der Spalte USERID in der Tabelle USERS zurückgegeben:

```
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)
```

Die Funktion LOG

Gibt den Logarithmus von mit zurück. `expr` `base`

Syntax

```
LOG(base, expr)
```

Argument

expr

Der Ausdruck muss einen Ganzzahl-, Dezimal- oder Gleitkommadatentyp haben.

base

Die Basis für die Logarithmusberechnung. Muss eine positive Zahl (ungleich 1) vom Datentyp doppelter Genauigkeit sein.

Rückgabetyp

Die LOG-Funktion gibt eine Doppelpräzisionszahl zurück.

Beispiel

Im folgenden Beispiel wird der Logarithmus der Zahl 100 zur Basis 10 zurückgegeben:

```
select log(10, 100);
--------
```

```
2
(1 row)
```

Die Funktion MOD

Gibt den Rest von zwei Zahlen zurück, auch bekannt als Modulo-Operation. Um das Ergebnis zu berechnen, wird der erste Parameter durch den zweiten geteilt.

Syntax

```
MOD(number1, number2)
```

Argumente

number1

Der erste Eingabeparameter ist eine INTEGER-, SMALLINT-, BIGINT- oder DECIMAL-Zahl. Wenn es sich bei einem der beiden Parameter um einen Parameter des Typs DECIMAL handelt, muss es sich beim anderen Parameter ebenfalls um einen Parameter des Typs DECIMAL handeln. Wenn es sich bei einem der beiden Parameter um einen Parameter des Typs INTEGER handelt, kann es sich beim anderen Parameter um einen Parameter des Typs INTEGER, SMALLINT oder BIGINT handeln. Beide Parameter können auch den Typ SMALLINT oder BIGINT haben. Wenn ein Parameter jedoch den Typ BIGINT hat, kann der andere Parameter nicht den Typ SMALLINT haben.

number2

Der zweite Parameter ist eine INTEGER-, SMALLINT-, BIGINT- oder DECIMAL-Zahl. Die gleichen Datentypregeln gelten für number2 wie für number1.

Rückgabetyp

Gültige Rückgabetypen sind DECIMAL, INT, SMALLINT und BIGINT. Der Rückgabetyp der MOD-Funktion ist der gleiche numerische Typ wie die Eingabeparameter, wenn beide Eingabeparameter denselben Datentyp haben. Wenn es sich bei einem der Eingabeparameter um einen INTEGER handelt, ist der Rückgabetyp auch ein INTEGER.

Nutzungshinweise

Sie können % als Modulo-Operator verwenden.

Beispiele

Im folgenden Beispiel wird der Rest einer Division von zwei Zahlen zurückgegeben:

```
SELECT MOD(10, 4);

 mod
------
 2
```

Im folgenden Beispiel wird ein Dezimalergebnis zurückgegeben:

```
SELECT MOD(10.5, 4);

 mod
------
 2.5
```

Sie können Parameterwerte umwandeln:

```
SELECT MOD(CAST(16.4 as integer), 5);

 mod
------
 1
```

Überprüfen Sie, ob der erste Parameter gerade ist, indem Sie ihn durch 2 teilen:

```
SELECT mod(5,2) = 0 as is_even;

 is_even
--------
 false
```

Sie können % als Modulo-Operator verwenden:

```
SELECT 11 % 4 as remainder;

 remainder
-----------
 3
```

Das folgende Beispiel gibt Informationen zu Kategorien mit ungeraden Nummern in der Tabelle CATEGORY zurück:

```
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)
```

Die Funktion PI

Die PI-Funktion gibt den Wert von Pi auf 14 Dezimalstellen zurück.

Syntax

```
PI()
```

Rückgabetyp

`DOUBLE PRECISION`

Beispiele

Verwenden Sie das folgende Beispiel, um den Wert von Pi zurückzugeben.

```
SELECT PI();

+------------------+
|        pi        |
+------------------+
| 3.141592653589793 |
+------------------+
```

Die Funktion POWER

Die POWER-Funktion ist eine Exponentialfunktion, die einen numerischen Ausdruck mit der Potenz eines zweiten numerischen Ausdrucks potenziert. Beispielsweise wird 2 in der dritten Potenz als `POWER(2,3)` berechnet. Das Ergebnis ist 8.

Syntax

```
{POWER(expression1, expression2)
```

Argumente

expression1

Der numerische Ausdruck, der potenziert werden soll. Muss ein `INTEGER`-, `DECIMAL`- oder `FLOAT`-Datentyp sein.

expression2

Potenz, mit der expression1potenziert werden soll. Muss ein `INTEGER`-, `DECIMAL`- oder `FLOAT`-Datentyp sein.

Rückgabetyp

`DOUBLE PRECISION`

Beispiel

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

Die Funktion RADIANS

Die RADIANS-Funktion konvertiert einen Winkel in Grad in die Entsprechung im Bogenmaß.

Syntax

```
RADIANS(number)
```

Argument

number (Zahl

Der Eingabeparameter ist eine DOUBLE PRECISION-Zahl.

Rückgabetyp

DOUBLE PRECISION

Beispiel

Verwenden Sie das folgende Beispiel, um die Entsprechung in 180 Grad des Radianten zurückzugeben.

```
SELECT RADIANS(180);

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

RAND-Funktion

Die RAND-Funktion generiert eine zufällige Gleitkommazahl zwischen 0 und 1. Die RAND-Funktion generiert bei jedem Aufruf eine neue Zufallszahl.

Syntax

```
RAND()
```

Rückgabetyp

RANDOM gibt einen Wert vom Typ DOUBLE zurück.

Beispiel

Im folgenden Beispiel wird für jede Zeile in der Tabelle eine Spalte mit zufälligen Gleitkommazahlen zwischen 0 und 1 generiert. `squirrels` Die resultierende Ausgabe wäre eine einzelne Spalte, die eine Liste zufälliger Dezimalwerte mit einem Wert für jede Zeile in der Squirrels-Tabelle enthält.

```
SELECT rand() FROM squirrels
```

Dieser Abfragetyp ist nützlich, wenn Sie Zufallszahlen generieren müssen, um beispielsweise zufällige Ereignisse zu simulieren oder Zufälligkeit in Ihre Datenanalyse einzubeziehen. Im Kontext der `squirrels` Tabelle kann sie verwendet werden, um jedem Eichhörnchen Zufallswerte zuzuweisen, die dann für die weitere Verarbeitung oder Analyse verwendet werden könnten.

Die Funktion RANDOM

Die RANDOM-Funktion generiert einen zufälligen Wert zwischen 0,0 (einschließlich) und 1,0 (ausschließlich).

Syntax

```
RANDOM()
```

Rückgabetyp

RANDOM gibt eine DOUBLE PRECISION-Zahl zurück.

Beispiele

1. Berechnet einen zufälligen Wert zwischen 0 und 99. Wenn die zufällige Zahl 0 bis 1 ist, produziert diese Abfrage eine zufällige Zahl zwischen 0 und 100:

   ```
   select cast (random() * 100 as int);

   INTEGER
   ------
   24
   (1 row)
   ```

2. Rufen Sie eine einheitliche zufällige Stichprobe von 10 Elementen ab:

   ```
   select *
   from sales
   ```

```
order by random()
limit 10;
```

Rufen Sie jetzt eine zufällige Stichprobe von 10 Elementen ab, wählen Sie die Elemente jedoch im Verhältnis zu deren Preis aus. Beispiel: Ein Element, das doppelt so teuer wie ein anderes Element ist, wird doppelt so wahrscheinlich in den Abfrageergebnissen angezeigt:

```
select *
from sales
order by log(1 - random()) / pricepaid
limit 10;
```

3. In diesem Beispiel wird der SET-Befehl verwendet, um einen SEED-Wert festzulegen, sodass RANDOM eine vorhersehbare Zahlenfolge generiert.

   Geben Sie zunächst drei RANDOM-Ganzzahlen zurück, ohne zuerst den SEED-Wert festzulegen:

```
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)
```

   Legen Sie nun den SEED-Wert auf `.25` fest und geben Sie drei weitere RANDOM-Zahlen zurück:

```
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)
```

Setzen Sie zum Schluss den SEED-Wert auf `.25` zurück und überprüfen Sie, ob RANDOM dieselben Ergebnisse wie in den vorherigen drei Aufrufen zurückgibt:

```
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)
```

Die Funktion ROUND

Die ROUND-Funktion rundet Zahlen auf den nächsten Ganzzahl- oder Dezimalwert auf.

Die ROUND-Funktion kann optional ein zweites Argument als Ganzzahl umfassen, um die Anzahl der Dezimalstellen für die Rundung in beide Richtungen anzugeben. Wenn Sie das zweite Argument nicht angeben, wird die Funktion auf die nächste ganze Zahl gerundet. Wenn das zweite Argument >n angegeben wurde, wird die Funktion auf die nächste Zahl mit einer Genauigkeit von n Dezimalstellen gerundet.

Syntax

```
ROUND (number [ , integer ] )
```

Argument

number (Zahl

Eine Zahl oder ein Ausdruck, der zu einer Zahl ausgewertet wird. Dabei kann es sich um DECIMAL oder FLOAT8 Type handeln. AWS Clean Rooms kann andere Datentypen gemäß den impliziten Konvertierungsregeln konvertieren.

integer (optional)

Eine Ganzzahl, die die Zahl der Dezimalstellen für das Runden in beide Richtungen angibt.

Rückgabetyp

ROUND gibt denselben numerischen Datentyp wie das/die Eingabeargument(e) zurück.

Beispiele

Rundet die für eine bestimmte Transaktion gezahlte Vergütung auf die nächste ganze Zahl.

```
select commission, round(commission)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    28
(1 row)
```

Rundet die für eine bestimmte Transaktion gezahlte Vergütung auf die erste Dezimalstelle.

```
select commission, round(commission, 1)
```

```
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |  28.1
(1 row)
```

Erweitert für dieselbe Abfrage die Präzision in die entgegengesetzte Richtung.

```
select commission, round(commission, -1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    30
(1 row)
```

Die Funktion SIGN

Die SIGN-Funktion gibt das Vorzeichen (positiv oder negativ) einer Zahl zurück. Das Ergebnis der SIGN-Funktion ist `1`, `-1` oder `0`, was das Vorzeichen des Arguments anzeigt.

Syntax

```
SIGN (number)
```

Argument

number (Zahl

Zahl oder Ausdruck, der zu einer Zahl ausgewertet wird. Es kann der DECIMALor FLOAT8 Typ sein. AWS Clean Rooms kann andere Datentypen gemäß den impliziten Konvertierungsregeln konvertieren.

Rückgabetyp

SIGN gibt denselben numerischen Datentyp wie das/die Eingabeargument(e) zurück. Wenn die Eingabe DECIMAL ist, ist die Ausgabe DECIMAL (1,0).

Beispiele

Verwenden Sie das folgende Beispiel, um das Vorzeichen der Decke der Provision zu bestimmten, die für eine bestimmte Verkaufstransaktion aus der Tabelle SALES gezahlt wird.

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

Die Funktion SIN

SIN ist eine trigonometrische Funktion, die den Sinus einer Zahl zurückgibt. Der zurückgegebene Wert liegt zwischen -1 und 1.

Syntax

```
SIN(number)
```

Argument

number (Zahl

Eine DOUBLE PRECISION-Zahl im Bogenmaß.

Rückgabetyp

DOUBLE PRECISION

Beispiel

Verwenden Sie das folgende Beispiel, um den Sinus von -PI zurückzugeben.

```
SELECT SIN(-PI());

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

```
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

Die Funktion SQRT

Die SQRT-Funktion gibt die Quadratwurzel eines numerischen Werts zurück. Die Quadratwurzel ist eine Zahl, die mit sich selbst multipliziert den angegebenen Wert ergibt.

Syntax

```
SQRT (expression)
```

Argument

expression

Der Ausdruck muss einen Ganzzahl-, Dezimal- oder Gleitkommadatentyp haben. Der Ausdruck kann Funktionen enthalten. Das System könnte implizite Typumwandlungen durchführen.

Rückgabetyp

SQRT gibt eine DOUBLE PRECISION-Zahl zurück.

Beispiele

Im folgenden Beispiel wird die Quadratwurzel einer Zahl zurückgegeben.

```
select sqrt(16);

sqrt
---------------
4
```

Im folgenden Beispiel wird eine implizite Typumwandlung durchgeführt.

```
select sqrt('16');

sqrt
---------------
```

```
4
```

Im folgenden Beispiel werden Funktionen verschachtelt, um eine komplexere Aufgabe auszuführen.

```
select sqrt(round(16.4));

sqrt
---------------
4
```

Das folgende Beispiel ergibt die Länge des Radius, wenn die Fläche eines Kreises gegeben ist. Der Radius wird beispielsweise in Zoll berechnet, wenn die Fläche in Quadratzoll angegeben ist. Die Fläche in dem Beispiel beträgt 20.

```
select sqrt(20/pi());
```

Der Wert 5,046265044040321 wird zurückgegeben.

Im folgenden Beispiel wird die Quadratwurzel für COMMISSION-Werte aus der Tabelle SALES zurückgegeben. Die COMMISSION-Spalte ist eine DECIMAL-Spalte. Dieses Beispiel zeigt, wie Sie die Funktion in einer Abfrage mit komplexerer bedingter Logik verwenden können.

```
select sqrt(commission)
from sales where salesid < 10 order by salesid;

sqrt
------------------
10.4498803820905
3.37638860322683
7.24568837309472
5.1234753829798
...
```

Die folgende Abfrage gibt die gerundete Quadratwurzel für denselben Satz von COMMISSION-Werten zurück.

```
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
...
```

Weitere Informationen zu Beispieldaten finden Sie AWS Clean Rooms unter Beispieldatenbank.

Die Funktion TRUNC

Die TRUNC-Funktion verkürzt Zahlen auf die vorherige Ganz- oder Dezimalzahlen.

Die TRUNC-Funktion kann optional ein zweites Argument als Ganzzahl umfassen, um die Anzahl der Dezimalstellen für die Rundung in beide Richtungen anzugeben. Wenn Sie das zweite Argument nicht angeben, wird die Funktion auf die nächste ganze Zahl gerundet. Wenn das zweite Argument >n angegeben wurde, wird die Funktion auf die nächste Zahl mit einer Genauigkeit von >n Dezimalstellen gerundet. Die Funktion verkürzt auch einen Zeitstempel und gibt ein Datum zurück.

Syntax

```
TRUNC (number [ , integer ] |
timestamp )
```

Argumente

number (Zahl

Eine Zahl oder ein Ausdruck, der zu einer Zahl ausgewertet wird. Dabei kann es sich um DECIMAL oder FLOAT8 Type handeln. AWS Clean Rooms kann andere Datentypen gemäß den impliziten Konvertierungsregeln konvertieren.

integer (optional)

Eine Ganzzahl, die die Zahl der Dezimalstellen der Präzision in beide Richtungen anzeigt. Wenn keine Ganzzahl angegeben wird, wird die Zahl zu einer ganzen Zahl abgeschnitten. Wenn eine Ganzzahl angegeben wird, wird die Zahl an der angegebenen Dezimalstelle abgeschnitten.

timestamp

Die Funktion kann auch das Datum aus einem Zeitstempel zurückgeben. (Um einen Zeitstempelwert mit `00:00:00` als Uhrzeit zurückzugeben, wandeln Sie das Funktionsergebnis in einen Zeitstempel um.)

Rückgabetyp

TRUNC gibt denselben Datentyp wie das erste Eingabeargument zurück. Für Zeitstempel gibt TRUNC ein Datum zurück.

Beispiele

Schneidet die Provision ab, die für eine bestimmte Verkaufstransaktion gezahlt wird.

```
select commission, trunc(commission)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   111

(1 row)
```

Schneidet denselben Provisionswert an der ersten Dezimalstelle ab.

```
select commission, trunc(commission,1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 | 111.1

(1 row)
```

Schneidet die Provision mit einem negativen Wert für das zweite Argument ab; `111.15` wird auf `110` abgerundet.

```
select commission, trunc(commission,-1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   110
(1 row)
```

Gibt den Datumsabschnitt aus dem Ergebnis der SYSDATE-Funktion zurück (die einen Zeitstempel zurückgibt):

```
select sysdate;

timestamp
----------------------------
2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc
------------
2011-07-21
(1 row)
```

Wendet die TRUNC-Funktion auf eine TIMESTAMP-Spalte an. Der Rückgabetyp ist ein Datum.

```
select trunc(starttime) from event
order by eventid limit 1;

trunc
------------
2008-01-25
(1 row)
```

Skalarfunktionen

In diesem Abschnitt werden die in AWS Clean Rooms Spark SQL unterstützten Skalarfunktionen beschrieben. Eine Skalarfunktion ist eine Funktion, die einen oder mehrere Werte als Eingabe verwendet und einen einzelnen Wert als Ausgabe zurückgibt. Skalarfunktionen arbeiten mit einzelnen Zeilen oder Elementen und erzeugen für jede Eingabe ein einzelnes Ergebnis.

Skalarfunktionen wie SIZE unterscheiden sich von anderen Typen von SQL-Funktionen wie Aggregatfunktionen (Anzahl, Summe, Durchschnitt) und Funktionen zur Tabellengenerierung (Explode, Flatten). Diese anderen Funktionstypen arbeiten mit mehreren Zeilen oder generieren mehrere Zeilen, wohingegen Skalarfunktionen auf einzelne Zeilen oder Elemente angewendet werden.

Themen

- SIZE-Funktion

SIZE-Funktion

Die SIZE-Funktion verwendet ein vorhandenes Array, eine Map oder eine Zeichenfolge als Argument und gibt einen einzelnen Wert zurück, der die Größe oder Länge dieser Datenstruktur darstellt. Sie erstellt keine neue Datenstruktur. Es wird zum Abfragen und Analysieren der Eigenschaften vorhandener Datenstrukturen verwendet, anstatt neue zu erstellen.

Diese Funktion ist nützlich, um die Anzahl der Elemente in einem Array oder die Länge einer Zeichenfolge zu bestimmen. Sie kann besonders hilfreich sein, wenn Sie mit Arrays und anderen Datenstrukturen in SQL arbeiten, da Sie damit Informationen über die Größe oder Kardinalität der Daten abrufen können.

Syntax

```
size(expr)
```

Argumente

expr

Ein ARRAY-, MAP- oder STRING-Ausdruck.

Rückgabetyp

Die SIZE-Funktion gibt einen INTEGER-Wert zurück.

Beispiel

In diesem Beispiel wird die SIZE-Funktion auf das Array `['b', 'd', 'c', 'a']` angewendet und gibt den Wert zurück4, der der Anzahl der Elemente im Array entspricht.

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

In diesem Beispiel wird die SIZE-Funktion auf die Map `{'a': 1, 'b': 2}` angewendet und sie gibt den Wert zurück2, der der Anzahl der Schlüssel-Wert-Paare in der Map entspricht.

```
SELECT size(map('a', 1, 'b', 2));
```

```
 2
```

In diesem Beispiel wird die SIZE-Funktion auf die Zeichenfolge `'hello world'` angewendet und sie gibt den Wert zurück11, der der Anzahl der Zeichen in der Zeichenfolge entspricht.

```
SELECT size('hello world');
11
```

Zeichenfolgenfunktionen

Zeichenfolgefunktionen verarbeiten und bearbeiten Zeichenfolgen oder Ausdrücke, die zu Zeichenfolgen ausgewertet werden. Wenn das Argument string in diesen Funktionen ein Literalwert ist, muss es in einfache Anführungszeichen eingeschlossen werden. Die unterstützten Datentypen sind CHAR und VARCHAR.

Im folgenden Abschnitt werden Funktionsnamen, Syntax und Beschreibungen der unterstützten Funktionen bereitgestellt. Alle Offsets in Zeichenfolgen sind eins-basiert.

Themen

- Die Funktion REPLACE
- Die Funktion REVERSE
- Die Funktion RTRIM
- SPLIT-Funktion
- Die Funktion SPLIT_PART
- Die Funktion SUBSTRING
- Die Funktion TRANSLATE
- Die Funktion TRIM
- Die Funktion UPPER
- UUID-Funktion

Der Operator || (Verkettung)

Verkettet zwei Ausdrücke auf beiden Seiten des Symbols || und gibt den verketteten Ausdruck zurück.

Der Verkettungsoperator ist ähnlich wie. Funktion CONCAT

> **Note**
>
> Für die Funktion CONCAT und den Verkettungsoperator gilt, dass das Ergebnis der Verkettung null ist, wenn einer oder beide Ausdrücke null sind.

Syntax

```
expression1 || expression2
```

Argumente

expression1, expression2

Bei beiden Argumenten kann es sich um Zeichenfolgen oder Ausdrücke mit fester Länge oder mit variabler Länge handeln.

Rückgabetyp

Der Operator || gibt eine Zeichenfolge zurück. Der Zeichenfolgetyp ist derselbe wie die Eingabeargumente.

Beispiel

Im folgenden Beispiel werden die Felder FIRSTNAME und LASTNAME aus der Tabelle USERS verkettet:

```
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)
```

Um Spalten zu verketten, die möglicherweise Null-Werte enthalten, verwenden Sie den Ausdruck NVL- und COALESCE-Funktionen. Im folgenden Beispiel wird NVL verwendet, um eine 0 zurückzugeben, wenn NULL gefunden wird.

```
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
```

Die Funktion BTRIM

Die BTRIM-Funktion kürzt eine Zeichenfolge durch Entfernen von Leerzeichen am Anfang und am Ende oder durch Entfernen von Zeichen am Anfang und am Ende, die mit einer optionalen angegebenen Zeichenfolge übereinstimmen.

Syntax

```
BTRIM(string [, trim_chars ] )
```

Argumente

string

Die VARCHAR-Eingabezeichenfolge, die gekürzt werden soll.

trim_chars

Die VARCHAR-Zeichenfolge, die die Zeichen für die Übereinstimmung enthält.

Rückgabetyp

Die BTRIM-Funktion gibt eine VARCHAR-Zeichenfolge zurück.

Beispiele

Im folgenden Beispiel werden Leerzeichen am Anfang und am Ende aus der Zeichenfolge entfernt `' abc '`:

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

Im folgenden Beispiel werden die Zeichenfolgen `'xyz'` am Anfang und am Ende aus der Zeichenfolge `'xyzaxyzbxyzcxyz'` entfernt. Die Zeichenfolgen `'xyz'` am Anfang und am Ende werden entfernt, entsprechende Zeichenfolgen innerhalb dieser Zeichenfolge jedoch nicht.

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

Im folgenden Beispiel werden die Teile am Anfang und am Ende der Zeichenfolge `'setuphistorycassettes'` entfernt, die mit einem der Zeichen in der trim_chars-Liste `'tes'` übereinstimmen. Alle `t`, `e` oder `s` am Anfang oder Ende der Eingabezeichenfolge, die vor einem anderen Zeichen stehen, das nicht in der trim_chars-Liste enthalten ist, werden entfernt.

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
-----------------
 uphistoryca
```

Funktion CONCAT

Die CONCAT-Funktion verkettet zwei Ausdrücke und gibt den Ergebnisausdruck zurück. Um mehr als zwei Ausdrücke zu verketten, verwenden Sie verschachtelte CONCAT-Funktionen. Der Verkettungsoperator (`||`) zwischen zwei Ausdrücken generiert dieselben Ergebnisse wie die CONCAT-Funktion.

> **Note**
>
> Für die Funktion CONCAT und den Verkettungsoperator gilt, dass das Ergebnis der Verkettung null ist, wenn einer oder beide Ausdrücke null sind.

Syntax

```
CONCAT ( expression1, expression2 )
```

Argumente

expression1, expression2

Beide Argumente können eine Zeichenfolge mit fester Länge, eine Zeichenfolge variabler Länge, ein binärer Ausdruck oder ein Ausdruck sein, der für eine dieser Eingaben ausgewertet wird.

Rückgabetyp

CONCAT gibt einen Ausdruck zurück. Der Datentyp des Ausdrucks ist derselbe Typ wie die Eingabeargumente.

Wenn die Eingabeausdrücke unterschiedlichen Typs sind, wird AWS Clean Rooms versucht, einen der Ausdrücke implizit umzuwandeln. Wenn Werte nicht umgewandelt werden können, wird ein Fehler zurückgegeben.

Beispiele

Im folgenden Beispiel werden zwei Zeichenliterale verkettet:

```
select concat('December 25, ', '2008');

concat
-------------------
December 25, 2008
(1 row)
```

Die folgende Abfrage verwendet anstelle von || den Operator CONCAT und generiert dasselbe Ergebnis:

```
select 'December 25, '||'2008';

concat
-------------------
December 25, 2008
(1 row)
```

Im folgenden Beispiel werden zwei CONCAT-Funktionen verwendet, um drei Zeichenfolgen zu verketten:

```
select concat('Thursday, ', concat('December 25, ', '2008'));
```

```
concat
----------------------------
Thursday, December 25, 2008
(1 row)
```

Um Spalten zu verketten, die möglicherweise Null-Werte enthalten, verwenden Sie NVL- und COALESCE-Funktionen. Im folgenden Beispiel wird NVL verwendet, um eine 0 zurückzugeben, wenn NULL gefunden wird.

```
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)
```

Die folgende Abfrage verkettet CITY- und STATE-Werte aus der Tabelle VENUE:

```
select concat(venuecity, venuestate)
from venue
where venueseats > 75000
order by venueseats;

concat
------------------
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

Die folgende Abfrage verwendet verschachtelte CONCAT-Funktionen. Die Abfrage verkettet CITY- und STATE-Werte aus der Tabelle, trennt die Ergebniszeichenfolge jedoch durch ein Komma und ein Leerzeichen:

```
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)
```

Funktion FORMAT_STRING

Die FORMAT_STRING-Funktion erstellt eine formatierte Zeichenfolge, indem sie Platzhalter in einer Vorlagenzeichenfolge durch die angegebenen Argumente ersetzt. Sie gibt eine formatierte Zeichenfolge aus Formatzeichenfolgen im Printf-Stil zurück.

Die Funktion FORMAT_STRING ersetzt die Platzhalter in der Vorlagenzeichenfolge durch die entsprechenden Werte, die als Argumente übergeben wurden. Diese Art der Zeichenkettenformatierung kann nützlich sein, wenn Sie dynamisch Zeichenfolgen erstellen müssen, die eine Mischung aus statischem Text und dynamischen Daten enthalten, z. B. beim Generieren von Ausgabenachrichten, Berichten oder anderen Arten von informativem Text. Die FORMAT_STRING-Funktion bietet eine präzise und lesbare Möglichkeit, diese Arten von formatierten Zeichenfolgen zu erstellen, wodurch es einfacher wird, den Code, der die Ausgabe generiert, zu verwalten und zu aktualisieren.

Syntax

```
format_string(strfmt, obj, ...)
```

Argumente

strfmt

Ein STRING-Ausdruck.

obj

Ein STRING- oder numerischer Ausdruck.

Rückgabetyp

FORMAT_STRING gibt einen STRING zurück.

Beispiel

Das folgende Beispiel enthält eine Vorlagenzeichenfolge, die zwei Platzhalter enthält: `%d` für einen Dezimalwert (Ganzzahl) und `%s` für einen Zeichenkettenwert. Der `%d` Platzhalter wird durch den Dezimalwert (Ganzzahl) (`100`) ersetzt, und der Platzhalter %s wird durch den Zeichenfolgenwert () ersetzt. `"days"` Die Ausgabe ist eine Vorlagenzeichenfolge, bei der die Platzhalter durch die angegebenen Argumente ersetzt wurden:. `"Hello World 100 days"`

```
SELECT format_string("Hello World %d %s", 100, "days");
 Hello World 100 days
```

Die Funktionen LEFT und RIGHT

Diese Funktionen geben die angegebene Zahl der Zeichen am weitesten links oder am weitesten rechts in einer Zeichenfolge zurück.

Die Zahl basiert auf der Anzahl der Zeichen, nicht der Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt.

Syntax

```
LEFT ( string,  integer )

RIGHT ( string,  integer )
```

Argumente

string

Jede Zeichenfolge oder jeder Ausdruck, der zu einer Zeichenfolge ausgewertet wird.

integer

Eine positive Ganzzahl.

Rückgabetyp

LEFT und RIGHT geben eine VARCHAR-Zeichenfolge zurück.

Beispiel

Das folgende Beispiel gibt die 5 Zeichen ganz links und die 5 ganz rechts von Ereignisnamen zurück, die IDs zwischen 1000 und 1005 liegen:

```
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)
```

Die Funktion LENGTH

Die Funktion LOWER

Konvertiert eine Zeichenfolge in Kleinbuchstaben. LOWER unterstützt UTF-8-Multibyte-Zeichen bis zu einer maximalen Länge von vier Bytes pro Zeichen.

Syntax

```
LOWER(string)
```

Argument

string

Der Eingabeparameter ist eine VARCHAR-Zeichenfolge (oder ein anderer Datentyp wie CHAR, der implizit in VARCHAR konvertiert werden kann).

Rückgabetyp

Die LOWER-Funktion gibt eine Zeichenfolge zurück, die den gleichen Datentyp wie die Eingabezeichenfolge hat.

Beispiele

Im folgenden Beispiel wird das Feld „CATNAME“ in Kleinbuchstaben konvertiert:

```
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)
```

Die Funktionen LPAD und RPAD

Diese Funktionen fügen vor oder nach einer Zeichenfolge Zeichen an, basierend auf einer angegebenen Länge.

Syntax

```
LPAD (string1, length, [ string2 ])
```

```
RPAD (string1, length, [ string2 ])
```

Argumente

string1

Eine Zeichenfolge oder ein Ausdruck, der zu einer Zeichenfolge ausgewertet wird, beispielsweise der Name einer Zeichenspalte.

length

Eine Ganzzahl, die die Länge des Ergebnisses der Funktion definiert. Die Länge einer Zeichenfolge basiert auf der Anzahl der Zeichen, nicht der Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt. Wenn string1 länger als die angegebene Länge ist, wird sie abgeschnitten (rechts). Wenn length eine negative Zahl ist, ist das Ergebnis der Funktion eine leere Zeichenfolge.

string2

Ein oder mehrere Zeichen, die vor oder nach string1 angefügt werden. Dieses Argument ist optional. Wenn es nicht angegeben wird, werden Leerzeichen verwendet.

Rückgabetyp

Diese Funktionen geben einen VARCHAR-Datentyp zurück.

Beispiele

Schneidet einen angegebenen Satz von Veranstaltungsnamen auf 20 Zeichen ab und fügt vor den kürzeren Namen Leerzeichen an:

```
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)
```

Schneidet denselben Satz von Veranstaltungsnamen auf 20 Zeichen ab, fügt vor den kürzeren Namen jedoch an `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)
```

Die Funktion LTRIM

Kürzt Zeichen ab dem Anfang einer Zeichenfolge. Entfernt die längste Zeichenfolge, die nur Zeichen aus der Liste der Trimm-Zeichen enthält. Das Kürzen ist abgeschlossen, wenn in der Eingabezeichenfolge kein Kürzungszeichen vorkommt.

Syntax

```
LTRIM( string [, trim_chars] )
```

Argumente

string

Eine Zeichenfolgenspalte, ein Ausdruck oder ein Zeichenfolgenliteral, die/der/das gekürzt werden soll.

trim_chars

Eine Zeichenfolgenspalte, ein Ausdruck oder ein Zeichenfolgenliteral, die/der/das die Zeichen darstellt, die ab dem Anfang von string gekürzt werden sollen. Wenn nicht angegeben, wird ein Leerzeichen als Trimm-Zeichen verwendet.

Rückgabetyp

Die LTRIM-Funktion gibt eine Zeichenfolge zurück, die denselben Datentyp wie die Eingabezeichenfolge (string) hat (CHAR oder VARCHAR).

Beispiele

Im folgenden Beispiel wird das Jahr aus der `listime`-Spalte gekürzt. Die Trimm-Zeichen im Zeichenfolgenliteral `'2008-'` geben die Zeichen an, die von links gekürzt werden sollen. Bei Verwendung der Trimm-Zeichen `'028-'` erzielen Sie dasselbe Ergebnis.

```
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 entfernt alle Zeichen in trim_chars, wenn sie sich am Anfang von string befinden. Im folgenden Beispiel werden die Zeichen „C“, „D“ und „G“gekürzt, wenn sie sich am Anfang von VENUENAME befinden. Dabei handelt es sich um eine VARCHAR-Spalte.

```
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
```

Im folgenden Beispiel wird das Trimm-Zeichen 2 verwendet, das aus der`venueid`-Spalte abgerufen wird.

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

Im folgenden Beispiel werden keine Zeichen gekürzt, da vor dem Trimm-Zeichen `'0'` eine 2 enthalten ist.

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

Im folgenden Beispiel werden standardmäßige Leerzeichen als Trimm-Zeichen verwendet und die beiden Leerzeichen zu Beginn der Zeichenfolge werden gekürzt.

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

Die Funktion POSITION

Gibt den Ort der angegebenen Unterzeichenfolge innerhalb einer Zeichenfolge zurück.

Syntax

```
POSITION(substring IN string )
```

Argumente

substring

Die Unterzeichenfolge, die innerhalb der Zeichenfolge gesucht werden soll.

string

Die Zeichenfolge oder Spalte, die durchsucht werden soll.

Rückgabetyp

Die POSITION-Funktion gibt eine Ganzzahl zurück, die der Position der Unterzeichenfolge entspricht (eins-basiert, nicht null-basiert). Die Position basiert auf der Anzahl der Zeichen, nicht der Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt.

Nutzungshinweise

POSITION gibt 0 zurück, wenn die Unterzeichenfolge nicht innerhalb der Zeichenfolge gefunden wird:

```
select position('dog' in 'fish');

position
----------
 0
(1 row)
```

Beispiele

Im folgenden Beispiel wird die Position der Zeichenfolge `fish` innerhalb des Worts `dogfish` gezeigt:

```
select position('fish' in 'dogfish');

position
----------
 4
(1 row)
```

Im folgenden Beispiel wird die Zahl der Verkaufstransaktionen mit einer COMMISSION von mehr als 999,00 aus der Tabelle SALES zurückgegeben:

```
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)
```

Die Funktion REGEXP_COUNT

Durchsucht eine Zeichenfolge nach einem regulären Ausdrucksmuster und gibt eine Ganzzahl zurück, die die Häufigkeit angibt, mit der das Muster in der Zeichenfolge auftritt. Wenn keine Übereinstimmung gefunden wird, gibt die Funktion 0 zurück.

Syntax

```
REGEXP_COUNT ( source_string, pattern [, position [, parameters ] ] )
```

Argumente

source_string

Ein Zeichenfolgenausdruck (beispielsweise ein Spaltenname), der gesucht werden soll.

pattern

Ein Zeichenfolgenliteral, das ein Muster für reguläre Ausdrücke darstellt.

position

Eine positive Ganzzahl, die die Position innerhalb von source_string angibt, an der die Suche gestartet werden soll. Die Position basiert auf der Anzahl der Zeichen, nicht der Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt. Der Standardwert ist 1. Wenn position kleiner als 1 ist, beginnt die Suche mit dem ersten Zeichen von source_string. Wenn position größer als die Anzahl der Zeichen in source_string ist, ist das Ergebnis 0.

parameters (Parameter

Ein oder mehrere Zeichenfolgenliterale, die angeben, wie die Funktion mit dem Muster übereinstimmt. Die folgenden Werte sind möglich:

- c – Übereinstimmung mit Unterscheidung von Groß- und Kleinschreibung durchführen. Die Standardeinstellung ist, beim Abgleich die Groß- und Kleinschreibung zu beachten.
- i – Übereinstimmung ohne Unterscheidung von Groß- und Kleinschreibung durchführen.

- p – Das Musters mit einem PCRE-Dialekt (Perl Compatible Regular Expression) interpretieren.

Rückgabetyp

Ganzzahl

Beispiel

Im folgenden Beispiel wird die Häufigkeit gezählt, mit der eine Folge aus drei Buchstaben auftritt.

```
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');

 regexp_count
 --------------
            8
```

Im folgenden Beispiel wird die Häufigkeit gezählt, mit der der Name der obersten Domäne entweder `org` oder `edu` ist.

```
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
```

Im folgenden Beispiel wird die Anzahl der Vorkommen der Zeichenfolge `FOX` gezählt, wobei nicht zwischen Groß- und Kleinschreibung unterschieden wird.

```
SELECT regexp_count('the fox', 'FOX', 1, 'i');

 regexp_count
 --------------
            1
```

Im folgenden Beispiel wird ein im PCRE-Dialekt geschriebenes Muster verwendet, um Wörter mit mindestens einer Zahl und einem Kleinbuchstaben zu finden. Hierfür wird der Operator `?=` verwendet, der eine bestimmte Lookahead-Konnotation in PCRE hat. In diesem Beispiel wird

die Anzahl der Vorkommen solcher Wörter gezählt, wobei zwischen Groß- und Kleinschreibung unterschieden wird.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

 regexp_count
 --------------
            2
```

Im folgenden Beispiel wird ein im PCRE-Dialekt geschriebenes Muster verwendet, um Wörter mit mindestens einer Zahl und einem Kleinbuchstaben zu finden. Hierfür wird der Operator ?= verwendet, der eine bestimmte Konnotation in PCRE hat. In diesem Beispiel wird die Anzahl der Vorkommen solcher Wörter gezählt. Dies unterscheidet sich insofern vom vorherigen Beispiel, als dass nicht zwischen Groß- und Kleinschreibung unterschieden wird.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');

 regexp_count
 --------------
            3
```

Die Funktion REGEXP_INSTR

Durchsucht eine Zeichenfolge nach einem regulären Ausdrucksmuster und gibt eine Ganzzahl zurück, die die Anfangs- oder Endposition der übereinstimmenden Unterzeichenfolge angibt. Wenn keine Übereinstimmung gefunden wird, gibt die Funktion 0 zurück. REGEXP_INSTR ist der Funktion POSITION ähnlich. Sie können damit jedoch eine Zeichenfolge nach einem regulären Ausdrucksmuster durchsuchen.

Syntax

```
REGEXP_INSTR ( source_string, pattern [, position [, occurrence] [, option
 [, parameters ] ] ] ] )
```

Argumente

source_string

Ein Zeichenfolgenausdruck (beispielsweise ein Spaltenname), der gesucht werden soll.

pattern

Ein Zeichenfolgenliteral, das ein Muster für reguläre Ausdrücke darstellt.

position

Eine positive Ganzzahl, die die Position innerhalb von source_string angibt, an der die Suche gestartet werden soll. Die Position basiert auf der Anzahl der Zeichen, nicht der Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt. Der Standardwert ist 1. Wenn position kleiner als 1 ist, beginnt die Suche mit dem ersten Zeichen von source_string. Wenn position größer als die Anzahl der Zeichen in source_string ist, ist das Ergebnis 0.

occurrence

Eine positive Ganzzahl, die angibt, welches Vorkommen des Musters verwendet werden soll. REGEXP_INSTR überspringt die erste occurrence -1 Übereinstimmungen. Der Standardwert ist 1. Wenn occurrence kleiner als 1 oder größer als die Anzahl der Zeichen in source_string ist, wird die Suche ignoriert und das Ergebnis ist 0.

option

Ein Wert, der angibt, ob die Position des ersten Zeichens der Übereinstimmung (0) oder die Position des ersten Zeichens nach dem Ende der Übereinstimmung (1) zurückgegeben werden soll. Ein Wert ungleich null entspricht 1. Der Standardwert lautet 0.

parameters (Parameter

Ein oder mehrere Zeichenfolgenliterale, die angeben, wie die Funktion mit dem Muster übereinstimmt. Die folgenden Werte sind möglich:

- c – Übereinstimmung mit Unterscheidung von Groß- und Kleinschreibung durchführen. Die Standardeinstellung ist, beim Abgleich die Groß- und Kleinschreibung zu beachten.
- i – Übereinstimmung ohne Unterscheidung von Groß- und Kleinschreibung durchführen.
- e – Teilzeichenfolge mittels eines Unterausdrucks extrahieren.

  Wenn pattern einen Unterausdruck enthält, sucht REGEXP_INSTR nach einer Teilzeichenfolge, die mit dem ersten Unterausdruck in pattern übereinstimmt. REGEXP_INSTR berücksichtigt nur den ersten Unterausdruck. Zusätzliche Unterausdrücke werden ignoriert. Wenn das Muster über keinen Unterausdruck verfügt, ignoriert REGEXP_INSTR den Parameter 'e'.
- p – Das Musters mit einem PCRE-Dialekt (Perl Compatible Regular Expression) interpretieren.

Rückgabetyp

Ganzzahl

Beispiel

Im folgenden Beispiel wird nach dem Zeichen @ gesucht, mit dem Domänennamen beginnen. Anschließend wird die Anfangsposition der ersten Übereinstimmung zurückgegeben.

```
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
```

Im folgenden Beispiel wird nach Varianten des Worts `Center` gesucht. Anschließend wird die Anfangsposition der ersten Übereinstimmung zurückgegeben.

```
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
```

Im folgenden Beispiel wird die Anfangsposition des ersten Vorkommens der Zeichenfolge `FOX` gefunden, wobei nicht zwischen Groß- und Kleinschreibung unterschieden wird.

```
SELECT regexp_instr('the fox', 'FOX', 1, 1, 0, 'i');

 regexp_instr
```

```
--------------
           5
```

Im folgenden Beispiel wird ein im PCRE-Dialekt geschriebenes Muster verwendet, um Wörter mit mindestens einer Zahl und einem Kleinbuchstaben zu finden. Hierfür wird der Operator ?= verwendet, der eine bestimmte Lookahead-Konnotation in PCRE hat. In diesem Beispiel wird die Anfangsposition des zweiten Wortes gefunden.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

 regexp_instr
 --------------
           21
```

Im folgenden Beispiel wird ein im PCRE-Dialekt geschriebenes Muster verwendet, um Wörter mit mindestens einer Zahl und einem Kleinbuchstaben zu finden. Hierfür wird der Operator ?= verwendet, der eine bestimmte Lookahead-Konnotation in PCRE hat. In diesem Beispiel wird die Anfangsposition des zweiten Worts gefunden. Dies unterscheidet sich insofern vom vorherigen Beispiel, als dass nicht zwischen Groß- und Kleinschreibung unterschieden wird.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');

 regexp_instr
 --------------
           15
```

Die Funktion REGEXP_REPLACE

Durchsucht eine Zeichenfolge nach einem regulären Ausdrucksmuster und ersetzt jedes Vorkommen des Musters durch die angegebene Zeichenfolge. REGEXP_REPLACE ist Die Funktion REPLACE ähnlich. Sie können jedoch eine Zeichenfolge nach einem regulären Ausdrucksmuster durchsuchen.

REGEXP_REPLACE ist Die Funktion TRANSLATE und Die Funktion REPLACE ähnlich. TRANSLATE führt jedoch mehrere Einzelzeichenersetzungen durch und REPLACE ersetzt eine ganze Zeichenfolge durch eine andere Zeichenfolge. Mit REGEXP_REPLACE können Sie dagegen eine Zeichenfolge nach einem regulären Ausdrucksmuster durchsuchen.

Syntax

```
REGEXP_REPLACE ( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

Argumente

source_string

Ein Zeichenfolgenausdruck (beispielsweise ein Spaltenname), der gesucht werden soll.

pattern

Ein Zeichenfolgenliteral, das ein Muster für reguläre Ausdrücke darstellt.

replace_string

Ein Zeichenfolgenausdruck (beispielsweise ein Spaltenname), der jedes Vorkommen eines Musters ersetzt. Der Standardwert ist eine leere Zeichenfolge ("").

position

Eine positive Ganzzahl, die die Position innerhalb von source_string angibt, an der die Suche gestartet werden soll. Die Position basiert auf der Anzahl der Zeichen, nicht der Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt. Der Standardwert ist 1. Wenn position kleiner als 1 ist, beginnt die Suche mit dem ersten Zeichen von source_string. Wenn position größer als die Anzahl der Zeichen in source_string ist, ist das Ergebnis source_string.

parameters (Parameter

Ein oder mehrere Zeichenfolgenliterale, die angeben, wie die Funktion mit dem Muster übereinstimmt. Die folgenden Werte sind möglich:

- c – Übereinstimmung mit Unterscheidung von Groß- und Kleinschreibung durchführen. Die Standardeinstellung ist, beim Abgleich die Groß- und Kleinschreibung zu beachten.
- i – Übereinstimmung ohne Unterscheidung von Groß- und Kleinschreibung durchführen.
- p – Das Musters mit einem PCRE-Dialekt (Perl Compatible Regular Expression) interpretieren.

Rückgabetyp

VARCHAR

Wenn pattern oder replace_string NULL sind, ist der Rückgabewert NULL.

Beispiel

Im folgenden Beispiel werden @ und der Domänenname aus E-Mail-Adressen gelöscht.

```
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
```

Im folgenden Beispiel werden die Domänennamen von E-Mail-Adressen durch diesen Wert ersetzt: `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
```

Im folgenden Beispiel werden alle Vorkommen der Zeichenfolge `FOX` innerhalb des Werts `quick brown fox` ersetzt, wobei nicht zwischen Groß- und Kleinschreibung unterschieden wird.

```
SELECT regexp_replace('the fox', 'FOX', 'quick brown fox', 1, 'i');

   regexp_replace
---------------------
```

```
 the quick brown fox
```

Im folgenden Beispiel wird ein im PCRE-Dialekt geschriebenes Muster verwendet, um Wörter mit mindestens einer Zahl und einem Kleinbuchstaben zu finden. Hierfür wird der Operator ?= verwendet, der eine bestimmte Lookahead-Konnotation in PCRE hat. In diesem Beispiel werden alle Vorkommen eines solchen Worts mit dem Wert ersetzt `[hidden]`.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

       regexp_replace
------------------------------
 [hidden] plain A1234 [hidden]
```

Im folgenden Beispiel wird ein im PCRE-Dialekt geschriebenes Muster verwendet, um Wörter mit mindestens einer Zahl und einem Kleinbuchstaben zu finden. Hierfür wird der Operator ?= verwendet, der eine bestimmte Lookahead-Konnotation in PCRE hat. In diesem Beispiel werden alle Vorkommen eines solchen Worts mit dem Wert `[hidden]` ersetzt. Dies unterscheidet sich insofern vom vorherigen Beispiel, als dass nicht zwischen Groß- und Kleinschreibung unterschieden wird.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

         regexp_replace
--------------------------------
 [hidden] plain [hidden] [hidden]
```

Die Funktion REGEXP_SUBSTR

Gibt Zeichen aus einer Zeichenfolge zurück, indem diese nach einem regulären Ausdrucksmuster durchsucht wird. REGEXP_SUBSTR ist der Funktion Die Funktion SUBSTRING ähnlich. Sie können jedoch eine Zeichenfolge nach einem regulären Ausdrucksmuster durchsuchen. Wenn die Funktion den regulären Ausdruck keinem Zeichen in der Zeichenfolge zuordnen kann, wird eine leere Zeichenfolge zurückgegeben.

Syntax

```
REGEXP_SUBSTR ( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

Argumente

source_string

Ein Zeichenfolgeausdruck, der durchsucht werden soll.

pattern

Ein Zeichenfolgenliteral, das ein Muster für reguläre Ausdrücke darstellt.

position

Eine positive Ganzzahl, die die Position innerhalb von source_string angibt, an der die Suche gestartet werden soll. Die Position basiert auf der Anzahl der Zeichen, nicht der Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt. Der Standardwert ist 1. Wenn position kleiner als 1 ist, beginnt die Suche mit dem ersten Zeichen von source_string. Wenn position größer als die Anzahl der Zeichen in source_string ist, ist das Ergebnis eine leere Zeichenfolge ("").

occurrence

Eine positive Ganzzahl, die angibt, welches Vorkommen des Musters verwendet werden soll. REGEXP_SUBSTR überspringt die erste occurrence -1 Übereinstimmungen. Der Standardwert ist 1. Wenn occurrence kleiner als 1 oder größer als die Anzahl der Zeichen in source_string ist, wird die Suche ignoriert und das Ergebnis ist NULL.

parameters (Parameter

Ein oder mehrere Zeichenfolgenliterale, die angeben, wie die Funktion mit dem Muster übereinstimmt. Die folgenden Werte sind möglich:

- c – Übereinstimmung mit Unterscheidung von Groß- und Kleinschreibung durchführen. Die Standardeinstellung ist, beim Abgleich die Groß- und Kleinschreibung zu beachten.
- i – Übereinstimmung ohne Unterscheidung von Groß- und Kleinschreibung durchführen.
- e – Teilzeichenfolge mittels eines Unterausdrucks extrahieren.

  Wenn pattern einen Unterausdruck enthält, sucht REGEXP_SUBSTR nach einer Teilzeichenfolge, die mit dem ersten Unterausdruck in pattern übereinstimmt. Ein Unterausdruck ist ein Ausdruck innerhalb des Musters, der in Klammern gesetzt ist. Bei dem Muster `'This is a (\\w+)'` beispielsweise wird der erste Ausdruck mit der Zeichenfolge `'This is a '`, gefolgt von einem Wort abgeglichen. Anstatt ein Muster zurückzugeben, gibt REGEXP_SUBSTR mit dem Parameter e nur die Zeichenfolge innerhalb des Unterausdrucks zurück.

REGEXP_SUBSTR berücksichtigt nur den ersten Unterausdruck. Zusätzliche Unterausdrücke werden ignoriert. Wenn das Muster über keinen Unterausdruck verfügt, ignoriert REGEXP_SUBSTR den Parameter 'e'.

- p – Das Musters mit einem PCRE-Dialekt (Perl Compatible Regular Expression) interpretieren.

Rückgabetyp

VARCHAR

Beispiel

Im folgenden Beispiel wird der E-Mail-Adresse-Abschnitt zwischen dem Zeichen @ und der Domänenerweiterung zurückgegeben.

```
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
```

Im folgenden Beispiel wird der Teil der Eingabe zurückgegeben, der dem ersten Vorkommen der Zeichenfolge F0X entspricht, wobei nicht zwischen Groß- und Kleinschreibung unterschieden wird.

```
SELECT regexp_substr('the fox', 'FOX', 1, 1, 'i');

 regexp_substr
---------------
 fox
```

Das folgende Beispiel gibt den ersten Teil der Eingabe zurück, der mit Kleinbuchstaben beginnt. Dies ist funktionell identisch mit derselben SELECT-Anweisung ohne den c-Parameter.

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');
```

```
 regexp_substr
---------------
 abc
```

Im folgenden Beispiel wird ein im PCRE-Dialekt geschriebenes Muster verwendet, um Wörter mit mindestens einer Zahl und einem Kleinbuchstaben zu finden. Hierfür wird der Operator ?= verwendet, der eine bestimmte Lookahead-Konnotation in PCRE hat. In diesem Beispiel wird der Teil der Eingabe zurückgegeben, der dem zweiten Wort entspricht.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

Im folgenden Beispiel wird ein im PCRE-Dialekt geschriebenes Muster verwendet, um Wörter mit mindestens einer Zahl und einem Kleinbuchstaben zu finden. Hierfür wird der Operator ?= verwendet, der eine bestimmte Lookahead-Konnotation in PCRE hat. In diesem Beispiel wird der der Teil der Eingabe zurückgegeben, der dem zweiten Wort entspricht. Dies unterscheidet sich insofern vom vorherigen Beispiel, als dass nicht zwischen Groß- und Kleinschreibung unterschieden wird.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
---------------
 A1234
```

Im folgenden Beispiel wird ein Unterausdruck verwendet, um die zweite Zeichenfolge zu finden, die dem Muster `'this is a (\\w+)'` entspricht, wobei nicht zwischen Groß- und Kleinschreibung unterschieden wird. Der Unterausdruck in Klammern wird zurückgegeben.

```
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
```

Die Funktion REPEAT

Wiederholt eine Zeichenfolge mit der angegebenen Häufigkeit. Wenn der Eingabeparameter numerisch ist, wird er von REPEAT als Zeichenfolge behandelt.

Syntax

```
REPEAT(string, integer)
```

Argumente

string

Der erste Eingabeparameter ist die Zeichenfolge, die wiederholt werden soll.

integer

Der zweite Parameter ist eine Ganzzahl, die die Häufigkeit angibt, mit der die Zeichenfolge wiederholt werden soll.

Rückgabetyp

Die REPEAT-Funktion gibt eine Zeichenfolge zurück.

Beispiele

Im folgenden Beispiel wird der Wert der Spalte CATID in der Tabelle CATEGORY dreimal wiederholt:

```
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)
```

Die Funktion REPLACE

Ersetzt alle Vorkommen eines Satzes von Zeichen innerhalb einer vorhandenen Zeichenfolge durch andere angegebene Zeichen.

REPLACE ist Die Funktion TRANSLATE und Die Funktion REGEXP_REPLACE ähnlich. TRANSLATE führt jedoch mehrere Einzelzeichenersetzungen durch und REPLACE ersetzt eine ganze Zeichenfolge durch eine andere Zeichenfolge. REPLACE ersetzt dagegen eine ganze Zeichenfolge durch eine andere Zeichenfolge.

Syntax

```
REPLACE(string1, old_chars, new_chars)
```

Argumente

string

Die CHAR- oder VARCHAR-Zeichenfolge, die durchsucht werden soll.

old_chars

Die CHAR- oder VARCHAR-Zeichenfolge, die ersetzt werden soll.

new_chars

Die neue CHAR- oder VARCHAR-Zeichenfolge, die old_string ersetzt.

Rückgabetyp

VARCHAR

Wenn old_chars oder new_chars NULL sind, ist der Rückgabewert NULL.

Beispiele

Im folgenden Beispiel wird die Zeichenfolge `Shows` in `Theatre` im Feld CATGROUP konvertiert:

```
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)
```

Die Funktion REVERSE

Die REVERSE-Funktion wird für eine Zeichenfolge ausgeführt und gibt die Zeichen in umgekehrter Reihenfolge wieder. Beispielsweise gibt `reverse('abcde')` `edcba` zurück. Diese Funktion kann auf numerische und Datumsdatentypen sowie Zeichendatentypen angewendet werden. In den meisten Fällen hat sie jedoch für Zeichenfolgen mit Zeichen praktischen Nutzen.

Syntax

```
REVERSE ( expression )
```

Argument

expression

Ein Ausdruck mit einem Zeichen-, Datums-, Zeitstempel- oder numerischen Datentyp, der das Ziel der Zeichenumkehrung darstellt. Alle Ausdrücke werden implizit in Zeichenfolgen mit variabler Länge konvertiert. Leerzeichen am Ende von Zeichenfolgen mit fester Breite werden ignoriert.

Rückgabetyp

REVERSE gibt einen VARCHAR zurück.

Beispiele

Wählt fünf verschiedene Namen von Städten und die entsprechenden Umkehrungen der Namen aus der Tabelle USERS aus:

```
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)
```

Wählen Sie fünf Buchstaben IDs und die entsprechende umgekehrte IDs Zeichenkette aus:

```
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)
```

Die Funktion RTRIM

Die RTRIM-Funktion kürzt einen angegebenen Satz von Zeichen ab dem Ende einer Zeichenfolge. Entfernt die längste Zeichenfolge, die nur Zeichen aus der Liste der Trimm-Zeichen enthält. Das Kürzen ist abgeschlossen, wenn in der Eingabezeichenfolge kein Kürzungszeichen vorkommt.

Syntax

```
RTRIM( string, trim_chars )
```

Argumente

string

Eine Zeichenfolgenspalte, ein Ausdruck oder ein Zeichenfolgenliteral, die/der/das gekürzt werden soll.

trim_chars

Eine Zeichenfolgenspalte, ein Ausdruck oder ein Zeichenfolgenliteral, die/der/das die Zeichen darstellt, die am Ende von string gekürzt werden sollen. Wenn nicht angegeben, wird ein Leerzeichen als Trimm-Zeichen verwendet.

Rückgabetyp

Eine Zeichenfolge mit demselben Datentyp wie das string-Argument.

Beispiel

Im folgenden Beispiel werden Leerzeichen am Anfang und am Ende aus der Zeichenfolge entfernt `' abc '`:

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

Im folgenden Beispiel werden die Zeichenfolgen `'xyz'` am Ende der Zeichenfolge `'xyzaxyzbxyzcxyz'` entfernt. Die Zeichenfolgen `'xyz'` am Ende werden entfernt, entsprechende Zeichenfolgen innerhalb dieser Zeichenfolge jedoch nicht.

```
select 'xyzaxyzbxyzcxyz' as untrim,
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

Im folgenden Beispiel werden die Teile am Ende der Zeichenfolge `'setuphistorycassettes'` entfernt, die mit einem der Zeichen in der trim_chars-

Liste `'tes'` übereinstimmen. Alle `t`, `e` oder `s` am Ende der Eingabezeichenfolge, die vor einem anderen Zeichen stehen, das nicht in der trim_chars-Liste enthalten ist, werden entfernt.

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

Im folgenden Beispiel werden die Zeichen „Park“ ab dem Ende von VENUENAME gekürzt, wenn vorhanden:

```
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
```

Beachten Sie, dass RTRIM alle `P`, `a`, `r` oder `k` entfernt, wenn sie sich am Ende eines VENUENAME befinden.

SPLIT-Funktion

Die SPLIT-Funktion ermöglicht es Ihnen, Teilstrings aus einer größeren Zeichenfolge zu extrahieren und mit ihnen als Array zu arbeiten. Die SPLIT-Funktion ist nützlich, wenn Sie eine Zeichenfolge anhand eines bestimmten Trennzeichens oder Musters in einzelne Komponenten aufteilen müssen.

Syntax

```
split(str, regex, limit)
```

Argumente

str

Ein Zeichenkettenausdruck, der aufgeteilt werden soll.

regex

Eine Zeichenfolge, die einen regulären Ausdruck darstellt. Die Regex-Zeichenfolge sollte ein regulärer Java-Ausdruck sein.

limit

Ein Integer-Ausdruck, der steuert, wie oft die Regex angewendet wird.

- limit > 0: Die Länge des resultierenden Arrays wird den Grenzwert nicht überschreiten, und der letzte Eintrag des resultierenden Arrays enthält alle Eingaben, die über die letzte übereinstimmende Regex hinausgehen.
- limit <= 0: Regex wird so oft wie möglich angewendet, und das resultierende Array kann eine beliebige Größe haben.

Rückgabetyp

<STRING>Die SPLIT-Funktion gibt ein ARRAY zurück.

Falls`limit > 0`: Die Länge des resultierenden Arrays wird den Grenzwert nicht überschreiten, und der letzte Eintrag des resultierenden Arrays enthält alle Eingaben, die über den letzten übereinstimmenden regulären Ausdruck hinausgehen.

Wenn`limit <= 0`: Regex wird so oft wie möglich angewendet, und das resultierende Array kann eine beliebige Größe haben.

Beispiel

In diesem Beispiel teilt die SPLIT-Funktion die Eingabezeichenfolge `'oneAtwoBthreeC'` überall dort auf, wo sie auf die Zeichen `'A''B'`, oder trifft `'C'` (wie im Muster für reguläre Ausdrücke angegeben). `'[ABC]'` Die resultierende Ausgabe ist ein Array aus vier Elementen:`"one"`, `"two""three"`, und einer leeren Zeichenfolge`""`.

```
SELECT split('oneAtwoBthreeC', '[ABC]');
 ["one","two","three",""]
```

Die Funktion SPLIT_PART

Teilt eine Zeichenfolge am angegebenen Trennzeichen und gibt den Teil an der angegebenen Position zurück.

Syntax

```
SPLIT_PART(string, delimiter, position)
```

Argumente

string

Eine Zeichenfolgenspalte, ein Ausdruck oder ein Zeichenfolgenliteral, die/der/das geteilt werden soll. Die Zeichenfolge kann CHAR oder VARCHAR sein.

delimiter

Die Trennzeichen-Zeichenfolge, die Abschnitte des Eingabe-string angibt.

Wenn delimiter ein Literal ist, schließen Sie es in einfache Anführungszeichen ein.

position

Position des string-Abschnitts, der zurückgegeben werden soll (gezählt ab 1). Es muss sich um eine Ganzzahl größer als 0 handeln. Wenn position größer als die Anzahl der Zeichenfolgenabschnitte ist, gibt SPLIT_PART eine leere Zeichenfolge zurück. Wenn delimiter nicht in string gefunden wird, enthält der zurückgegebene Wert den Inhalt des angegebenen Teils. Dabei kann es sich um die gesamte Zeichenfolge oder einen leeren Wert handeln.

Rückgabetyp

Eine CHAR- oder VARCHAR-Zeichenfolge, identisch mit dem Parameter string.

Beispiele

Im folgenden Beispiel wird ein Zeichenfolgenliteral mithilfe des Trennzeichens $ in Teile aufgeteilt und der zweite Teil zurückgegeben.

```
select split_part('abc$def$ghi','$',2)

split_part
----------
def
```

Im folgenden Beispiel wird ein Zeichenfolgenliteral mithilfe des Trennzeichens $ in Teile aufgeteilt. Es wird eine leere Zeichenfolge zurückgegeben, da der Teil 4 nicht gefunden wurde.

```
select split_part('abc$def$ghi','$',4)

split_part
----------

```

Im folgenden Beispiel wird ein Zeichenfolgenliteral mithilfe des Trennzeichens # in Teile aufgeteilt. Da das Trennzeichen nicht gefunden wurde, wird die gesamte Zeichenfolge zurückgegeben, wobei es sich um den ersten Teil handelt.

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

Im folgenden Beispiel wird das Zeitstempelfeld LISTTIME in die Komponenten Jahr, Monat und Datum aufgeteilt.

```
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
```

Im folgenden Beispiel wird das Zeitstempelfeld LISTTIME ausgewählt und am Zeichen `'-'` getrennt, um den Monat zu erhalten (den zweiten Teil der Zeichenfolge LISTTIME). Anschließend wird die Zahl der Einträge für jeden Monat gezählt:

```
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
```

Die Funktion SUBSTRING

Gibt die Teilmenge einer Zeichenfolge basierend auf der angegebenen Startposition zurück.

Wenn es sich bei der Eingabe um eine Zeichenfolge handelt, basieren die Startposition und die Anzahl der extrahierten Zeichen auf Zeichen, nicht auf Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt. Wenn es sich bei der Eingabe um einen binären Ausdruck handelt, basieren die Startposition und die extrahierte Teilzeichenfolge auf Bytes. Sie können keine negative Länge angeben. Sie können jedoch eine negative Startposition angeben.

Syntax

```
SUBSTRING(charactestring FROM start_position [ FOR numbecharacters ] )
```

```
SUBSTRING(charactestring, start_position, numbecharacters )
```

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

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

Argumente

Zeichenkette

Die Zeichenfolge, die durchsucht werden soll. Datentypen, die keine Zeichen sind, werden als Zeichenfolge behandelt.

start_position

Die Position innerhalb der Zeichenfolge, an der die Extrahierung gestartet werden soll, beginnend mit 1. Die start_position basiert auf der Anzahl der Zeichen, nicht der Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt. Diese Zahl kann negativ sein.

Zahl der Zeichen

Die Anzahl der Zeichen, die extrahiert werden soll (die Länge der Unterzeichenfolge). Die Zahl der Zeichen basiert auf der Anzahl der Zeichen, nicht auf der Anzahl der Byte, sodass Multibyte-Zeichen als Einzelzeichen gezählt werden. Diese Zahl darf nicht negativ sein.

start_byte

Die Position innerhalb des Binärausdrucks, an der die Extrahierung gestartet werden soll, beginnend mit 1. Diese Zahl kann negativ sein.

Anzahl Byte

Die Anzahl der Bytes, die extrahiert werden sollen, also die Länge der Unterzeichenfolge. Diese Zahl darf nicht negativ sein.

Rückgabetyp

VARCHAR

Nutzungshinweise für Zeichenfolgen

Im folgenden Beispiel wird eine Zeichenfolge mit vier Zeichen zurückgegeben, beginnend mit dem sechsten Zeichen.

```
select substring('caterpillar',6,4);
```

```
substring
-----------
pill
(1 row)
```

Wenn start_position + numbecharacters die Länge der Zeichenfolge überschreitet, gibt SUBSTRING eine Teilzeichenfolge zurück, die von der Startposition bis zum Ende der Zeichenfolge beginnt. Zum Beispiel:

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

Wenn `start_position` negativ oder 0 ist, gibt die Funktion SUBSTRING eine Unterzeichenfolge ab dem ersten Zeichen der Zeichenfolge mit der Länge `start_position` + `numbecharacters` -1 zurück. Beispiel:

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

Wenn `start_position` + `numbecharacters` -1 gleich oder kleiner als null ist, gibt SUBSTRING eine leere Zeichenfolge zurück. Beispiel:

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

Beispiele

Im folgenden Beispiel wird der Monat aus der Zeichenfolge LISTTIME in der Tabelle LISTING zurückgegeben:

```
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)
```

Im folgenden Beispiel wird das Gleiche wie oben gezeigt, jedoch mit der Option 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)
```

Sie können SUBSTRING nicht verwenden, um das Präfix einer Zeichenfolge, die möglicherweise Multibyte-Zeichen enthält, auf vorhersehbare Weise zu extrahieren, da Sie die Länge einer Multibyte-Zeichenfolge anhand der Anzahl der Bytes und nicht anhand der Anzahl der Zeichen

angeben müssen. Um das Anfangssegment einer Zeichenfolge auf der Basis der Länge in Bytes zu extrahieren, können Sie die Zeichenfolge in (byte_length) umwandeln, um die Zeichenfolge abzuschneiden, wobei byte_length die erforderliche Länge ist. Im folgenden Beispiel werden die ersten 5 Bytes aus der Zeichenfolge extrahiert `'Fourscore and seven'`.

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
Fours
```

Das folgende Beispiel gibt den Vornamen `Ana` zurück, der nach dem letzten Leerzeichen in der Eingabezeichenfolge `Silva, Ana` erscheint.

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))

 reverse
-----------
 Ana
```

Die Funktion TRANSLATE

Ersetzt für einen bestimmten Ausdruck alle Vorkommen von angegebenen Zeichen durch angegebene Ersatzzeichen. Vorhandene Zeichen werden aufgrund Ihrer Positionen in den Argumenten characters_to_replace und characters_to_substitute zu Ersatzzeichen zugeordnet. Wenn im Argument characters_to_replace mehr Zeichen als im Argument characters_to_substitute angegeben sind, werden die zusätzlichen Zeichen aus dem Argument characters_to_replace im Rückgabewert ausgelassen.

TRANSLATE ist Die Funktion REPLACE und Die Funktion REGEXP_REPLACE ähnlich. Während REPLACE jedoch eine ganze Zeichenfolge durch eine andere Zeichenfolge ersetzt und REGEXP_REPLACE eine Zeichenfolge nach einem regulären Ausdrucksmuster durchsucht, führt TRANSLATE mehrere Einzelzeichenersetzungen aus.

Wenn ein Argument null ist, ist der Rückgabewert NULL.

Syntax

```
TRANSLATE ( expression, characters_to_replace, characters_to_substitute )
```

Argumente

expression

Der Ausdruck, der übersetzt werden soll.

characters_to_replace

Eine Zeichenfolge, die die Zeichen enthält, die ersetzt werden sollen.

characters_to_substitute

Eine Zeichenfolge, die die Zeichen enthält, die ersetzt werden sollen.

Rückgabetyp

VARCHAR

Beispiele

Im folgenden Beispiel werden mehrere Zeichen in einer Zeichenfolge ersetzt:

```
select translate('mint tea', 'inea', 'osin');

translate
-----------
most tin
```

Im folgenden Beispiel wird für alle Werte in einer Spalte das Zeichen @ durch einen Punkt ersetzt:

```
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
```

Im folgenden Beispiel werden für alle Werte in einer Spalte Leerzeichen durch Unterstriche ersetzt und Punkte entfernt:

```
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
```

Die Funktion TRIM

Kürzt eine Zeichenfolge durch Entfernen von Leerzeichen am Anfang und am Ende oder durch Entfernen von Zeichen am Anfang und am Ende, die mit einer optionalen angegebenen Zeichenfolge übereinstimmen.

Syntax

```
TRIM( [ BOTH ] [ trim_chars FROM ] string
```

Argumente

trim_chars

(Optional) Die Zeichen, die aus der Zeichenfolge gekürzt werden sollen. Wenn dieser Parameter ausgelassen wird, werden Leerzeichen ausgeschnitten.

string

Die Zeichenfolge, die gekürzt werden soll.

Rückgabetyp

Die TRIM-Funktion gibt eine VARCHAR- oder eine CHAR_Zeichenfolge zurück. Wenn Sie die TRIM-Funktion mit einem SQL-Befehl verwenden, werden die Ergebnisse implizit in VARCHAR konvertiert. AWS Clean Rooms Wenn Sie die TRIM-Funktion in der SELECT-Liste für eine SQL-Funktion verwenden, werden die Ergebnisse AWS Clean Rooms nicht implizit konvertiert, und Sie müssen möglicherweise eine explizite Konvertierung durchführen, um zu vermeiden, dass ein Datentypkonflikt auftritt. Informationen zu expliziten Konvertierungen finden Sie in der CAST-Funktion Funktion.

Beispiel

Im folgenden Beispiel werden Leerzeichen am Anfang und am Ende aus der Zeichenfolge entfernt `' abc '`:

```
select '     abc    ' as untrim, trim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

Im folgenden Beispiel werden die doppelten Anführungszeichen entfernt, die die Zeichenfolge umgeben `"dog"`:

```
select trim('"' FROM '"dog"');
```

```
btrim
-------
dog
```

TRIM entfernt alle Zeichen in trim_chars, wenn sie sich am Anfang von string befinden. Im folgenden Beispiel werden die Zeichen „C“, „D“ und „G“gekürzt, wenn sie sich am Anfang von VENUENAME befinden. Dabei handelt es sich um eine VARCHAR-Spalte.

```
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
```

Die Funktion UPPER

Konvertiert eine Zeichenfolge in Großbuchstaben. UPPER unterstützt UTF-8-Multibyte-Zeichen bis zu einer maximalen Länge von vier Bytes pro Zeichen.

Syntax

```
UPPER(string)
```

Argumente

string

Der Eingabeparameter ist eine VARCHAR-Zeichenfolge (oder ein anderer Datentyp wie CHAR, der implizit in VARCHAR konvertiert werden kann).

Rückgabetyp

Die UPPER-Funktion gibt eine Zeichenfolge zurück, die den gleichen Datentyp wie die Eingabezeichenfolge hat.

Beispiele

Im folgenden Beispiel wird das Feld CATNAME in Großbuchstaben konvertiert:

```
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)
```

UUID-Funktion

Die UUID-Funktion generiert einen Universally Unique Identifier (UUID).

UUIDs sind global eindeutige Identifikatoren, die üblicherweise verwendet werden, um eindeutige Identifikatoren für verschiedene Zwecke bereitzustellen, z. B.:

- Identifizieren von Datenbankeinträgen oder anderen Dateneinheiten.
- Generierung eindeutiger Namen oder Schlüssel für Dateien, Verzeichnisse oder andere Ressourcen.
- Verfolgen und Korrelieren von Daten in verteilten Systemen.
- Bereitstellung eindeutiger Kennungen für Netzwerkpakete, Softwarekomponenten oder andere digitale Ressourcen.

Die UUID-Funktion generiert einen UUID-Wert, der mit sehr hoher Wahrscheinlichkeit einzigartig ist, selbst in verteilten Systemen und über lange Zeiträume. UUIDs werden in der Regel anhand einer Kombination aus dem aktuellen Zeitstempel, der Netzwerkadresse des Computers und anderen zufälligen oder pseudozufälligen Daten generiert, wodurch sichergestellt wird, dass es sehr unwahrscheinlich ist, dass jede generierte UUID mit einer anderen UUID in Konflikt gerät.

Im Kontext einer SQL-Abfrage kann die UUID-Funktion verwendet werden, um eindeutige Bezeichner für neue Datensätze zu generieren, die in eine Datenbank eingefügt werden, oder um eindeutige Schlüssel für die Datenpartitionierung, Indizierung oder andere Zwecke bereitzustellen, bei denen ein eindeutiger Bezeichner erforderlich ist.

> **Note**
>
> Die UUID-Funktion ist nicht deterministisch.

Syntax

```
uuid()
```

Argumente

Die UUID-Funktion benötigt kein Argument.

Rückgabetyp

UUID gibt eine UUID-Zeichenfolge (Universally Unique Identifier) zurück. Der Wert wird als kanonische UUID-Zeichenfolge mit 36 Zeichen zurückgegeben.

Beispiel

Im folgenden Beispiel wird ein Universally Unique Identifier (UUID) generiert. Die Ausgabe ist eine 36-stellige Zeichenfolge, die einen Universally Unique Identifier darstellt.

```
SELECT uuid();
 46707d92-02f4-4817-8116-a4c3b23e6266
```

Funktionen im Zusammenhang mit dem Datenschutz

AWS Clean Rooms stellt Funktionen bereit, die Sie bei der Einhaltung der Datenschutzbestimmungen für die folgenden Spezifikationen unterstützen.

- Global Privacy Platform (GPP) — Eine Spezifikation des Interactive Advertising Bureau (IAB), die einen globalen, standardisierten Rahmen für Online-Datenschutz und Datennutzung festlegt. Weitere Informationen zu den technischen Spezifikationen des GPP finden Sie in der Dokumentation der Global Privacy Platform unter. GitHub
- Transparency and Consent Framework (TCF) — Eine Schlüsselkomponente des GPP, das 2020 eingeführt wurde und einen standardisierten technischen Rahmen bietet, der Unternehmen bei der Einhaltung von Datenschutzbestimmungen wie der Datenschutz-Grundverordnung (DSGVO) der EU unterstützt. Das TCF ermöglicht es Kunden, die Zustimmung zur Datenerhebung und -verarbeitung zu erteilen oder zu verweigern. Weitere Informationen zu den technischen Spezifikationen von TCF finden Sie in der TCF-Dokumentation unter. GitHub

Themen

Funktion consent_gpp_v1_decode

Die `consent_gpp_v1_decode` Funktion wird verwendet, um Einwilligungsdaten der Global Privacy Platform (GPP) v1 zu dekodieren. Sie verwendet die kodierte Einwilligungszeichenfolge als Eingabe und gibt die dekodierten Einwilligungsdaten zurück, die Informationen über die Datenschutzpräferenzen und Einwilligungsoptionen des Benutzers enthalten. Diese Funktion ist nützlich, wenn Sie mit Daten arbeiten, die GPP v1-Einwilligungsinformationen enthalten, da Sie damit auf die Einwilligungsdaten in einem strukturierten Format zugreifen und diese analysieren können.

Syntax

```
consent_gpp_v1_decode(gpp_string)
```

Argumente

gpp_string

Die kodierte GPP v1-Zustimmungszeichenfolge.

Rückgabewert

Das zurückgegebene Wörterbuch enthält die folgenden Schlüssel-Wert-Paare:

- `version`: Die verwendete Version der GPP-Spezifikation (derzeit 1).
- `cmpId`: Die ID der Consent Management Platform (CMP), die die Zustimmungszeichenfolge codiert hat.
- `cmpVersion`: Die Version der CMP, die die Zustimmungszeichenfolge codiert hat.
- `consentScreen`: Die ID des Bildschirms in der CMP-Benutzeroberfläche, auf dem der Benutzer seine Zustimmung gegeben hat.
- `consentLanguage`: Der Sprachcode der Einwilligungsinformationen.
- `vendorListVersion`: Die verwendete Version der Lieferantenliste.
- `publisherCountryCode`: Die Landesvorwahl des Herausgebers.
- `purposeConsent`: Eine Liste von ganzen Zahlen, die die Zwecke darstellen, denen der Benutzer zugestimmt hat.
- `purposeLegitimateInterest`: Eine Liste von Zwecken, IDs für die das berechtigte Interesse des Benutzers transparent mitgeteilt wurde.
- `specialFeatureOptIns`: Eine Liste von Ganzzahlen, die die speziellen Funktionen darstellen, für die sich der Benutzer entschieden hat.
- `vendorConsent`: Eine Liste der Anbieter IDs , denen der Benutzer zugestimmt hat.
- `vendorLegitimateInterest`: Eine Liste von Anbietern, IDs für die das berechtigte Interesse des Benutzers transparent mitgeteilt wurde.

Beispiel

Das folgende Beispiel verwendet ein einzelnes Argument, nämlich die kodierte Zustimmungszeichenfolge. Es gibt ein Wörterbuch zurück, das die dekodierten Einwilligungsdaten enthält, einschließlich Informationen über die Datenschutzeinstellungen, Einwilligungsoptionen und andere Metadaten des Benutzers.

```
SELECT * FROM consent_gpp_v1_decode('ABCDEFGHIJK');
```

Die grundlegende Struktur der zurückgegebenen Einwilligungsdaten umfasst Informationen über die Version der Einwilligungszeichenfolge, die CMP-Details (Consent Management Platform), die Zustimmung des Benutzers und die Optionen seiner berechtigten Interessen für verschiedene Zwecke und Anbieter sowie andere Metadaten.

```
{
    "version": 1,
```

```
    "cmpId": 12,
    "cmpVersion": 34,
    "consentScreen": 5,
    "consentLanguage": "en",
    "vendorListVersion": 89,
    "publisherCountryCode": "US",
    "purposeConsent": [1],
    "purposeLegitimateInterests": [1],
    "specialFeatureOptins": [1],
    "vendorConsent": [1],
    "vendorLegitimateInterests": [1]}
}
```

Funktion consent_tcf_v2_decode

Die `consent_tcf_v2_decode` Funktion wird verwendet, um Zustimmungsdaten des Transparency and Consent Framework (TCF) v2 zu dekodieren. Sie verwendet die kodierte Einwilligungszeichenfolge als Eingabe und gibt die dekodierten Einwilligungsdaten zurück, die Informationen über die Datenschutzpräferenzen und Einwilligungsoptionen des Benutzers enthalten. Diese Funktion ist nützlich, wenn Sie mit Daten arbeiten, die TCF v2-Einwilligungsinformationen enthalten, da Sie damit auf die Einwilligungsdaten in einem strukturierten Format zugreifen und diese analysieren können.

Syntax

```
consent_tcf_v2_decode(tcf_string)
```

Argumente

tcf_string

Die kodierte TCF v2-Zustimmungszeichenfolge.

Rückgabewert

Die `consent_tcf_v2_decode` Funktion gibt ein Wörterbuch zurück, das die dekodierten Zustimmungsdaten aus einer TCF (Transparency and Consent Framework) v2-Zustimmungszeichenfolge enthält.

Das zurückgegebene Wörterbuch enthält die folgenden Schlüssel-Wert-Paare:

Kernsegment

- `version`: Die verwendete Version der TCF-Spezifikation (derzeit 2).
- `created`: Datum und Uhrzeit der Erstellung der Zustimmungszeichenfolge.
- `lastUpdated`: Datum und Uhrzeit der letzten Aktualisierung der Zustimmungszeichenfolge.
- `cmpId`: Die ID der Consent Management Platform (CMP), die die Zustimmungszeichenfolge codiert hat.
- `cmpVersion`: Die Version der CMP, die die Zustimmungszeichenfolge codiert hat.
- `consentScreen`: Die ID des Bildschirms in der CMP-Benutzeroberfläche, auf dem der Benutzer seine Zustimmung gegeben hat.
- `consentLanguage`: Der Sprachcode der Einwilligungsinformationen.
- `vendorListVersion`: Die verwendete Version der Lieferantenliste.
- `tcfPolicyVersion`: Die Version der TCF-Richtlinie, auf der die Zustimmungszeichenfolge basiert.
- `isServiceSpecific`: Ein boolescher Wert, der angibt, ob die Zustimmung für einen bestimmten Dienst spezifisch ist oder für alle Dienste gilt.
- `useNonStandardStacks`: Ein boolescher Wert, der angibt, ob Stacks verwendet werden, die nicht dem Standard entsprechen.
- `specialFeatureOptIns`: Eine Liste von Ganzzahlen, die die speziellen Funktionen darstellen, für die sich der Benutzer entschieden hat.
- `purposeConsent`: Eine Liste von Ganzzahlen, die die Zwecke darstellen, denen der Benutzer zugestimmt hat.
- `purposesLITransparency`: Eine Liste von ganzen Zahlen, die die Zwecke darstellen, für die der Benutzer seine berechtigten Interessen transparent gemacht hat.
- `purposeOneTreatment`: Ein boolescher Wert, der angibt, ob der Benutzer die „Einzelbehandlung“ angefordert hat (d. h., alle Zwecke werden gleich behandelt).
- `publisherCountryCode`: Die Landesvorwahl des Herausgebers.
- `vendorConsent`: Eine Liste der Anbieter IDs , denen der Benutzer zugestimmt hat.
- `vendorLegitimateInterest`: Eine Liste von Anbietern, IDs für die das berechtigte Interesse des Benutzers transparent mitgeteilt wurde.
- `pubRestrictionEntry`: Eine Liste mit Einschränkungen für Herausgeber. Dieses Feld enthält die Verwendungs-ID, den Einschränkungstyp und die Liste der Anbieter, für die IDs diese Verwendungsbeschränkung gilt.

Offengelegtes Lieferantensegment

- `disclosedVendors`: Eine Liste von ganzen Zahlen, die die Anbieter repräsentieren, die dem Benutzer bekannt gegeben wurden.

Segment für Zwecke des Herausgebers

- `pubPurposesConsent`: Eine Liste von ganzen Zahlen, die die verlagsspezifischen Zwecke darstellen, für die der Benutzer seine Zustimmung erteilt hat.
- `pubPurposesLITransparency`: Eine Liste von ganzen Zahlen, die die verlegerspezifischen Zwecke darstellen, für die der Nutzer seine berechtigten Interessen transparent gemacht hat.
- `customPurposesConsent`: Eine Liste von Ganzzahlen, die die benutzerdefinierten Zwecke darstellen, für die der Benutzer seine Zustimmung erteilt hat.
- `customPurposesLITransparency`: Eine Liste von Ganzzahlen, die die benutzerdefinierten Zwecke darstellen, für die der Benutzer seine berechtigten Interessen transparent gemacht hat.

Diese detaillierten Einwilligungsdaten können verwendet werden, um die Datenschutzpräferenzen des Benutzers bei der Arbeit mit personenbezogenen Daten zu verstehen und zu respektieren.

Beispiel

Das folgende Beispiel verwendet ein einzelnes Argument, nämlich die kodierte Zustimmungszeichenfolge. Es gibt ein Wörterbuch zurück, das die dekodierten Einwilligungsdaten enthält, einschließlich Informationen über die Datenschutzeinstellungen, Einwilligungsoptionen und andere Metadaten des Benutzers.

```
from aws_clean_rooms.functions import consent_tcf_v2_decode

consent_string = "CO1234567890abcdef"
consent_data = consent_tcf_v2_decode(consent_string)

print(consent_data)
```

Die grundlegende Struktur der zurückgegebenen Einwilligungsdaten umfasst Informationen über die Version der Einwilligungszeichenfolge, die CMP-Details (Consent Management Platform), die Zustimmung des Benutzers und die Optionen seiner berechtigten Interessen für verschiedene Zwecke und Anbieter sowie andere Metadaten.

```
    /** 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],
};
```

Fensterfunktionen

Mit Fensterfunktionen können Sie analytische geschäftliche Abfragen effizienter erstellen. Fensterfunktionen werden für eine Partition bzw. ein „Fenster“ eines Ergebnissatzes ausgeführt und geben für jede Zeile in diesem Fenster einen Wert zurück. Funktionen ohne Fenster führen ihre Berechnungen dagegen für alle Zeilen des Ergebnissatzes aus. Im Gegensatz zu Gruppenfunktionen, die die Ergebniszeilen aggregieren, behalten Fensterfunktionen alle Zeilen im Tabellenausdruck bei.

Die zurückgegebenen Werte werden mithilfe von Werten aus den Sätzen von Zeilen in diesem Fenster berechnet. Das Fenster definiert für jede Zeile in der Tabelle einen Satz von Zeilen, der für die Verarbeitung zusätzlicher Attribute verwendet wird. Ein Fenster wird mithilfe einer Fensterspezifikation (der OVER-Klausel) definiert und basiert auf drei Hauptkonzepten:

- Fensterpartitionierung, die Gruppen von Zeilen bildet (PARTITION-Klausel)
- Fensteranordnung, die eine Reihenfolge oder Sequenz von Zeilen innerhalb der einzelnen Partitionen definiert (ORDER BY-Klausel)
- Fensterrahmen, die in Bezug auf die einzelnen Zeilen definiert werden, um den Satz von Zeilen weiter einzuschränken (ROWS-Spezifikation)

Fensterfunktionen sind der letzte Satz von Operationen, die in einer Abfrage ausgeführt werden, abgesehen von der abschließenden ORDER BY-Klausel. Alle Joins und alle -, - und - Klauseln werden abgeschlossen, bevor die Fensterfunktionen verarbeitet werden. Daher können Fensterfunktionen nur in der Auswahlliste oder in der ORDER BY-Klausen enthalten sein. Innerhalb einer einzelnen Abfrage können mehrere Fensterfunktionen mit unterschiedlichen Rahmenklauseln verwendet werden. Außerdem können Sie Fensterfunktionen in anderen skalaren Ausdrücken verwenden, beispielsweise CASE.

Übersicht über die Syntax von Fensterfunktionen

Fensterfunktionen folgen einer Standardsyntax, die wie folgt lautet.

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

Hier ist function eine der in diesem Abschnitt beschriebenen Funktionen.

Die expr_list lautet wie folgt.

```
expression | column_name [, expr_list ]
```

Die order_list lautet wie folgt.

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

Die frame_clause lautet wie folgt.

```
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 }}
```

Argumente

Funktion

Die Fensterfunktion. Details finden Sie in den Beschreibungen der einzelnen Funktionen.

OVER

Die Klausel, die die Fensterspezifikation definiert. Die OVER-Klausel ist für Fensterfunktionen obligatorisch und differenziert Fensterfunktionen von anderen SQL-Funktionen.

PARTITION BY expr_list

(Optional) Die PARTITION-BY-Klausel unterteilt den Ergebnissatz in Partitionen, ähnlich wie die GROUP-BY-Klausel. Wenn eine Partitionsklausel vorhanden ist, wird die Funktion für die Zeilen in den einzelnen Partitionen berechnet. Wenn keine Partitionsklausel angegeben ist, enthält eine einzige Partition die gesamte Tabelle und die Funktion wird für die gesamte Tabelle berechnet.

Die Rangfestlegungsfunktionen DENSE_RANK, NTILE, RANK und ROW_NUMBER erfordern einen globalen Vergleich aller Zeilen im Ergebnissatz. Wenn eine PARTITION BY-Klausel verwendet wird, kann die Abfrageoptimierung die einzelnen Aggregationen parallel ausführen, indem der Workload entsprechend den Partitionen über mehrere Slices verteilt wird. Wenn die PARTITION BY-Klausel nicht vorhanden ist, muss der Aggregationsschritt seriell für einen einzelnen Slice ausgeführt werden. Dies kann erhebliche negative Auswirkungen auf die Leistung haben, besonders für größere Cluster.

AWS Clean Roomsunterstützt keine Zeichenkettenliterale in PARTITION BY-Klauseln.

ORDER BY order_list

(Optional) Die Fensterfunktion wird auf die Zeilen innerhalb der einzelnen Partitionen angewendet, sortiert entsprechend der Reihenfolgenspezifikation in ORDER BY. Diese ORDER BY-Klausel unterscheidet sich von der ORDER BY-Klausel in der frame_clauseund ist mit dieser in keiner

Weise verwandt. Die ORDER BY-Klausel kann ohne die PARTITION BY-Klausel verwendet werden.

Für Rangfestlegungsfunktionen identifiziert die ORDER BY-Klausel die Messwerte für die Rangfestlegungswerte. Für Aggregationsfunktionen müssen die partitionierten Zeilen angeordnet werden, bevor die jeweilige Aggregationsfunktion für die einzelnen Rahmen berechnet wird. Weitere Informationen zu den Arten von Windowsfunktionen finden Sie unter Fensterfunktionen.

In der Reihenfolgenliste werden Spaltenbezeichner oder Ausdrücke, die zu Spaltenbezeichnern ausgewertet werden, benötigt. Konstanten oder Konstantenausdrücke können nicht als Ersatz für Spaltennamen verwendet werden.

NULL-Werte werden als eigene Gruppe behandelt und entsprechend der Option NULLS FIRST oder NULLS LAST sortiert und angeordnet. Standardmäßig werden NULL-Werte in einer ASC-Reihenfolge an letzter Stelle sortiert und aufgeführt und in einer DESC-Reihenfolge an erster Stelle sortiert und aufgeführt.

AWS Clean Roomsunterstützt keine Zeichenkettenliterale in ORDER BY-Klauseln.

Wenn die ORDER BY-Klausel ausgelassen wird, ist die Reihenfolge der Zeilen nicht deterministisch.

> **Note**
>
> In jedem parallel SystemAWS Clean Rooms, z. B. wenn eine ORDER BY-Klausel keine eindeutige und vollständige Reihenfolge der Daten erzeugt, ist die Reihenfolge der Zeilen nicht deterministisch. Das heißt, wenn der ORDER BY-Ausdruck doppelte Werte erzeugt (eine teilweise Reihenfolge), kann die Reihenfolge der Rückgabe dieser Zeilen von einem Lauf AWS Clean Rooms zum nächsten variieren. In diesem Fall können Fensterfunktionen unerwartete oder inkonsistente Ergebnisse zurückgeben. Weitere Informationen finden Sie unter Spezifisches Anordnen von Daten für Fensterfunktionen.

column_name

Der Name einer Spalte, nach der die Partitionierung oder Anordnung erfolgen soll.

ASC | DESC

Eine Option, die die Sortierreihenfolge für den Ausdruck wie folgt definiert:

- ASC: aufsteigend (beispielsweise niedrig nach hoch für numerische Werte und A bis Z für Zeichenfolgen). Wenn keine Option angegeben wird, werden die Daten standardmäßig in aufsteigender Reihenfolge sortiert.
- DESC: absteigend (beispielsweise hoch nach niedrig für numerische Werte und Z bis A für Zeichenfolgen).

NULLS FIRST | NULLS LAST

Option, die angibt, ob NULL-Werte an erster Stelle vor Nicht-Null-Werten oder an letzter Stelle nach Nicht-Null-Werten aufgelistet werden sollen. Standardmäßig werden NULL-Werte in einer ASC-Reihenfolge an letzter Stelle sortiert und aufgeführt und in einer DESC-Reihenfolge an erster Stelle sortiert und aufgeführt.

frame_clause

Die Rahmenklausel gibt für Aggregationsfunktionen den Satz von Zeilen im Fenster einer Funktion bei Verwendung von ORDER BY noch genauer an. Sie ermöglicht das Ein- oder Ausschließen von Sätzen von Zeilen innerhalb des geordneten Ergebnisses. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren.

Die Rahmenklausel kann nicht auf Rangfestlegungsfunktionen angewendet werden. Außerdem ist sie nicht erforderlich, wenn in der ORDER-BY-Klausel für eine Aggregationsfunktion keine OVER-Klausel verwendet wird. Wenn eine ORDER BY-Klausel für eine Aggregationsfunktion verwendet wird, ist eine explizite Rahmenklausel erforderlich.

Wenn keine ORDER-BY-Klausel angegeben ist, ist der implizierte Rahmen unbegrenzt, äquivalent zu ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

ROWS

Diese Klausel definiert den Fensterrahmen durch Angabe eines physischen Offsets von der aktuellen Zeile.

Diese Klausel gibt die Zeilen im aktuellen Fenster oder in der aktuellen Partition an, mit denen der Wert in der aktuellen Zeile kombiniert werden soll. Sie verwendet Argumente, die die Zeilenposition angeben. Diese kann sich vor oder nach der aktuellen Zeile befinden. Der Referenzpunkt für alle Fensterrahmen ist die aktuelle Zeile. Alle Zeilen werden nacheinander zur aktuellen Zeile, während der Fensterrahmen in der Partition vorwärts gleitet.

Beim Rahmen kann es sich um einen einfachen Satz von Zeilen bis zur und einschließlich der aktuellen Zeile handeln.

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

Es kann sich auch um einen Satz von Zeilen zwischen zwei Grenzen handeln.

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING zeigt an, dass das Fenster an der ersten Zeile der Partition beginnt; offset PRECEDING zeigt an, dass das Fenster um eine Zahl von Reihen vor der aktuellen Zeile beginnt, die dem Offset-Wert entspricht. UNBOUNDED PRECEDING ist der Standardwert.

CURRENT ROW zeigt an, dass das Fenster an der aktuellen Zeile beginnt oder endet.

UNBOUNDED FOLLOWING zeigt an, dass das Fenster an der letzten Zeile der Partition endet; offset FOLLOWING zeigt an, dass das Fenster um eine Zahl von Reihen nach der aktuellen Zeile endet, die dem Offset-Wert entspricht.

offset bezeichnet eine physische Anzahl von Zeilen vor oder nach der aktuellen Zeile. In diesem Fall muss offset eine Konstante sein, der zu einem positiven numerischen Wert ausgewertet wird. Beispielsweise wird bei 5 FOLLOWING der Rahmen fünf Zeilen nach der aktuellen Zeile beendet.

Wenn BETWEEN nicht angegeben ist, wird der Rahmen implizit von der aktuellen Zeile begrenzt. Beispielsweise ist `ROWS 5 PRECEDING` gleich `ROWS BETWEEN 5 PRECEDING AND CURRENT ROW`. Ebenso ist `ROWS UNBOUNDED FOLLOWING` gleich `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`.

> **Note**
>
> Sie können keinen Rahmen angeben, in dem die Startgrenze größer als die Endgrenze ist. Sie können beispielsweise keinen der folgenden Rahmen angeben.
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

Spezifisches Anordnen von Daten für Fensterfunktionen

Wenn eine ORDER-BY-Klausel für eine Fensterfunktion keine spezifische und globale Anordnung der Daten generiert, ist die Reihenfolge der Zeilen nicht deterministisch. Wenn der ORDER-BY-Ausdruck duplizierte Werte generiert (eine partielle Anordnung), kann sich die Rückgabereihenfolge dieser Zeilen zwischen verschiedenen Ausführungen unterscheiden. In diesem Fall geben Fensterfunktionen möglicherweise unerwartete oder inkonsistente Ergebnisse zurück.

Beispielsweise gibt die folgende Abfrage in verschiedenen Ausführen unterschiedliche Ergebnisse zurück. Diese unterschiedlichen Ergebnisse treten auf, da `order by dateid` keine spezifische Reihenfolge der Daten für die SUM-Fensterfunktion erzeugt.

```
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 diesem Fall kann das Hinzufügen einer zweiten ORDER-BY-Spalte zur Fensterfunktion das Problem lösen.

```
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
...
```

Unterstützte Funktionen

AWS Clean RoomsSpark SQL unterstützt zwei Arten von Fensterfunktionen: Aggregat- und Rangfunktionen.

Die folgenden Aggregationsfunktionen werden unterstützt:

- CUME_DIST-Fensterfunktion
- Die Fensterfunktion DENSE_RANK
- Funktion „ERSTES Fenster“
- Die Fensterfunktion FIRST_VALUE
- Die Fensterfunktion LAG
- Funktion „LETZTES Fenster“
- Die Fensterfunktion LAST_VALUE
- Die Fensterfunktion LEAD

Die folgenden Rangfestlegungsfunktionen werden unterstützt:

- Die Fensterfunktion DENSE_RANK
- Die Fensterfunktion PERCENT_RANK
- Die Fensterfunktion RANK
- Die Fensterfunktion ROW_NUMBER

Beispieltabelle mit Beispielen von Fensterfunktionen

Zu jeder Funktionsbeschreibung gehören spezifische Fensterfunktionsbeispiele. In einigen Beispielen wird eine Tabelle mit dem Namen WINSALES verwendet, die 11 Zeilen enthält, wie in der folgenden Tabelle dargestellt.

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
30001	8/2/2003	3	B	10	10
10001	12/24/2003	1	C	10	10
10005	12/24/2003	1	A	30	
40001	1/9/2004	4	A	40	
10006	1/18/2004	1	C	10	
20001	2/12/2004	2	B	20	20
40005	2/12/2004	4	A	10	10
20002	2/16/2004	2	C	20	20
30003	4/18/2004	3	B	15	
30004	4/18/2004	3	B	20	
30007	9/7/2004	3	C	30	

CUME_DIST-Fensterfunktion

Berechnet die kumulative Verteilung eines Werts in einem Fenster oder einer Partition. Bei aufsteigender Anordnung wird die kumulative Verteilung anhand der folgenden Formel festgelegt:

```
count of rows with values <= x / count of rows in the window or partition
```

wobei x gleich dem Wert in der aktuellen Zeile der Spalte ist, die in der ORDER BY-Klausel angegeben wird. Der folgende Datensatz zeigt die Verwendung dieser Formel:

```
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
```

Der Rückgabewertbereich ist >0 bis 1 (einschließlich).

Syntax

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Argumente

OVER

Eine Klausel, die die Fensterpartitionierung angibt. Die OVER-Klausel darf keine Fensterrahmenspezifikation enthalten.

PARTITION BY partition_expression

Optional. Ein Ausdruck, der den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel festlegt.

ORDER BY order_list

Der Ausdruck, anhand dessen die kumulative Verteilung berechnet wird. Der Datentyp des Ausdrucks muss entweder numerisch sein oder implizit in einen solchen konvertierbar sein. Wenn ORDER BY ausgelassen wird, ist der Rückgabewert für alle Zeilen 1.

Wenn ORDER-BY nicht zu einer spezifischen Reihenfolge führt, ist die Reihenfolge der Zeilen nicht deterministisch. Weitere Informationen finden Sie unter Spezifisches Anordnen von Daten für Fensterfunktionen.

Rückgabetyp

FLOAT8

Beispiele

Im folgenden Beispiel wird die kumulative Verteilung der Menge für die einzelnen Verkäufer berechnet:

```
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
```

Eine Beschreibung der Tabelle WINSALES finden Sie unter Beispieltabelle mit Beispielen von Fensterfunktionen.

Die Fensterfunktion DENSE_RANK

Die Fensterfunktion DENSE_RANK legt den Rang eines Werts in einer Gruppe von Werten fest, basierend auf dem ORDER BY-Ausdruck in der OVER-Klausel. Wenn die optionale PARTITION BY-Klausel vorhanden ist, wird die Rangfolge für jede Gruppe von Zeilen neu festgelegt. Zeilen mit gleichen Werten in Bezug auf die Rangfestlegungskriterien erhalten den gleichen Rang. Die Funktion DENSE_RANK unterscheidet sich nur in einer Hinsicht von RANK: Wenn zwei oder mehr Zeilen den gleichen Rang erhalten, entsteht in der Rangfolge der Werte keine Lücke. Wenn beispielsweise zwei Zeilen den Rang 1 erhalten, ist der nächste Rang 2.

Sie können in derselben Abfrage Rangfestlegungsfunktionen mit unterschiedlichen PARTITION BY- und ORDER BY-Klauseln verwenden.

Syntax

```
DENSE_RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumente

( )

Die Funktion verwendet keine Argumente. Es ist jedoch eine leere Klammer erforderlich.

OVER

Die Fensterklauseln für die Funktion DENSE_RANK.

PARTITION BY expr_list

Optional. Ein oder mehrere Ausdrücke, der/die das Fenster definiert/definieren.

ORDER BY order_list

Optional. Der Ausdruck, auf dem die Rangfestlegungwerte basieren. Wenn PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle. Wenn ORDER BY ausgelassen wird, ist der Rückgabewert für alle Zeilen 1.

Wenn ORDER-BY nicht zu einer spezifischen Reihenfolge führt, ist die Reihenfolge der Zeilen nicht deterministisch. Weitere Informationen finden Sie unter Spezifisches Anordnen von Daten für Fensterfunktionen.

Rückgabetyp

INTEGER

Beispiele

Im folgenden Beispiel wird die Tabelle nach der verkauften Menge (in absteigender Reihenfolge) geordnet und jeder Zeile ein DENSE_RANK-Wert und ein regulärer Rang zugewiesen. Die Ergebnisse werden sortiert, nachdem die Fensterfunktionsergebnisse angewendet wurden.

```
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)
```

Beachten Sie den Unterschied bei den Rängen, die demselben Satz von Zeilen zugewiesen werden, wenn die Funktionen DENSE_RANK und RANK zusammen in derselben Umfrage verwendet werden. Eine Beschreibung der Tabelle WINSALES finden Sie unter Beispieltabelle mit Beispielen von Fensterfunktionen.

Im folgenden Beispiel wird die Tabelle nach SELLERID partitioniert, die einzelnen Partitionen nach Menge (in absteigender Reihenfolge) geordnet und jeder Zeile ein DENSE_RANK-Wert zugewiesen. Die Ergebnisse werden sortiert, nachdem die Fensterfunktionsergebnisse angewendet wurden.

```
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)
```

Eine Beschreibung der Tabelle WINSALES finden Sie unter Beispieltabelle mit Beispielen von Fensterfunktionen.

Funktion „ERSTES Fenster“

Bei einer bestimmten Anzahl von Zeilen gibt FIRST den Wert des angegebenen Ausdrucks in Bezug auf die erste Zeile im Fensterrahmen zurück.

Informationen zur Auswahl der letzten Zeile im Rahmen finden Sie unter Funktion „LETZTES Fenster“.

Syntax

```
FIRST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumente

expression

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird.

IGNORE NULLS

Wenn diese Option mit FIRST verwendet wird, gibt die Funktion den ersten Wert im Frame zurück, der nicht NULL ist (oder NULL, wenn alle Werte NULL sind).

RESPECT NULLS

Gibt an, dass Nullwerte bei der Bestimmung der zu verwendenden Zeile berücksichtigt werden AWS Clean Rooms sollen. Wenn Sie IGNORE NULLS nicht angeben, wird RESPECT NULLS standardmäßig unterstützt.

OVER

Führt die Fensterklauseln für die Funktion ein.

PARTITION BY expr_list

Definiert das Fenster für die Funktion in Bezug auf mindestens einen Ausdruck.

ORDER BY order_list

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn die PARTITION BY-Klausel nicht angegeben ist, sortiert ORDER BY die gesamte Tabelle. Wenn Sie eine ORDER BY-Klausel angeben, müssen Sie auch eine frame_clause angeben.

Die Ergebnisse der FIRST-Funktion hängen von der Reihenfolge der Daten ab. Die Ergebnisse sind in den folgenden Fällen nicht deterministisch:

- Wenn keine ORDER BY-Klausel angegeben ist und eine Partition zwei verschiedene Werte für einen Ausdruck enthält
- Wenn der Ausdruck zu verschiedenen Werten ausgewertet wird, die demselben Wert in der ORDER BY-Liste entsprechen

frame_clause

Wenn eine ORDER BY-Klausel für eine Aggregationsfunktion verwendet wird, ist eine explizite Rahmenklausel erforderlich. Die Rahmenklausel gibt den Satz von Zeilen im Fenster einer Funktion genauer an, einschließlich oder ausschließlich Sätzen von Zeilen im geordneten Ergebnis. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe Übersicht über die Syntax von Fensterfunktionen.

Rückgabetyp

Diese Funktionen unterstützen Ausdrücke, die primitive AWS Clean Rooms Datentypen verwenden. Der Rückgabetyp ist mit dem Datentyp von expression identisch.

Beispiele

Im folgenden Beispiel wird die Sitzplatzkapazität für die einzelnen Veranstaltungsorte in der Tabelle VENUE zurückgegeben, wobei die Ergebnisse nach Kapazität (hoch zu niedrig) geordnet sind. Die FIRST-Funktion wird verwendet, um den Namen des Veranstaltungsorts auszuwählen, der der ersten Reihe im Frame entspricht: in diesem Fall der Zeile mit der höchsten Anzahl von Sitzplätzen. Die Ergebnisse werden nach Bundesstaat partitioniert. Wenn der Wert für VENUESTATE geändert wird, wird daher ein neuer erster Wert ausgewählt. Der Fensterrahmen ist unbegrenzt. Daher wird für jede Zeile in jeder Partition derselbe erste Wert ausgewählt.

Im Fall von Kalifornien hat `Qualcomm Stadium` die größte Zahl von Sitzplätzen (`70561`). Daher ist dieser Name der erste Wert für alle Zeilen in der Partition `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
...
```

Die Fensterfunktion FIRST_VALUE

Bei einem geordneten Satz von Zeilen gibt FIRST_VALUE den Wert des angegebenen Ausdrucks in Bezug auf die erste Zeile im Fensterrahmen zurück.

Informationen zur Auswahl der letzten Zeile im Rahmen finden Sie unter Die Fensterfunktion LAST_VALUE.

Syntax

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumente

expression

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird.

IGNORE NULLS

Bei Verwendung dieser Option für FIRST_VALUE gibt die Funktion den ersten Wert im Rahmen zurück, der nicht NULL ist (oder NULL, wenn alle Werte NULL sind).

RESPECT NULLS

Gibt an, dass bei der Bestimmung der zu verwendenden Zeile Nullwerte berücksichtigt werden AWS Clean Rooms sollen. Wenn Sie IGNORE NULLS nicht angeben, wird RESPECT NULLS standardmäßig unterstützt.

OVER

Führt die Fensterklauseln für die Funktion ein.

PARTITION BY expr_list

Definiert das Fenster für die Funktion in Bezug auf mindestens einen Ausdruck.

ORDER BY order_list

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn die PARTITION BY-Klausel nicht angegeben ist, sortiert ORDER BY die gesamte Tabelle. Wenn Sie eine ORDER BY-Klausel angeben, müssen Sie auch eine frame_clause angeben.

Die Ergebnisse der Funktion FIRST_VALUE sind von der Anordnung der Daten abhängig. Die Ergebnisse sind in den folgenden Fällen nicht deterministisch:

- Wenn keine ORDER BY-Klausel angegeben ist und eine Partition zwei verschiedene Werte für einen Ausdruck enthält
- Wenn der Ausdruck zu verschiedenen Werten ausgewertet wird, die demselben Wert in der ORDER BY-Liste entsprechen

frame_clause

Wenn eine ORDER BY-Klausel für eine Aggregationsfunktion verwendet wird, ist eine explizite Rahmenklausel erforderlich. Die Rahmenklausel gibt den Satz von Zeilen im Fenster einer Funktion genauer an, einschließlich oder ausschließlich Sätzen von Zeilen im geordneten Ergebnis. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe Übersicht über die Syntax von Fensterfunktionen.

Rückgabetyp

Diese Funktionen unterstützen Ausdrücke, die primitive AWS Clean Rooms Datentypen verwenden. Der Rückgabetyp ist mit dem Datentyp von expression identisch.

Beispiele

Im folgenden Beispiel wird die Sitzplatzkapazität für die einzelnen Veranstaltungsorte in der Tabelle VENUE zurückgegeben, wobei die Ergebnisse nach Kapazität (hoch zu niedrig) geordnet sind. Die Funktion FIRST_VALUE wird verwendet, um den Namen des Veranstaltungsorts auszuwählen, der der ersten Zeile im Rahmen entspricht, in diesem Fall der Zeile mit der größten Zahl von Sitzplätzen. Die Ergebnisse werden nach Bundesstaat partitioniert. Wenn der Wert für VENUESTATE geändert wird, wird daher ein neuer erster Wert ausgewählt. Der Fensterrahmen ist unbegrenzt. Daher wird für jede Zeile in jeder Partition derselbe erste Wert ausgewählt.

Im Fall von Kalifornien hat `Qualcomm Stadium` die größte Zahl von Sitzplätzen (`70561`). Daher ist dieser Name der erste Wert für alle Zeilen in der Partition `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
...
```

Die Fensterfunktion LAG

Die Fensterfunktion LAG gibt die Werte für eine Zeile in einem bestimmten Offset oberhalb (vor) der aktuellen Zeile in der Partition zurück.

Syntax

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Argumente

value_expr

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird.

offset

Ein optionaler Parameter, der die Anzahl der Zeilen vor der aktuellen Zeile angibt, für die Werte zurückgegeben werden sollen. Beim Offset kann es sich um eine ganzzahlige Konstante oder um einen Ausdruck handeln, der zu einer Ganzzahl ausgewertet wird. Wenn Sie keinen Offset angeben, AWS Clean Rooms verwendet 1 als Standardwert. Ein Offset von `0` gibt die aktuelle Zeile an.

IGNORE NULLS

Eine optionale Angabe, die angibt, dass Nullwerte bei der Bestimmung der zu verwendenden Zeile übersprungen werden AWS Clean Rooms sollen. Wenn IGNORE NULLS nicht angegeben wird, werden Null-Werte berücksichtigt.

> **Note**
>
> Sie können einen NVL- oder COALESCE-Ausdruck verwenden, um die Null-Werte durch einen anderen Wert zu ersetzen.

RESPECT NULLS

Gibt an, dass Nullwerte bei der Bestimmung der zu verwendenden Zeile berücksichtigt werden AWS Clean Rooms sollen. Wenn Sie IGNORE NULLS nicht angeben, wird RESPECT NULLS standardmäßig unterstützt.

OVER

Gibt die Fensterpartitionierung und -anordnung an. Die OVER-Klausel darf keine Fensterrahmenspezifikation enthalten.

PARTITION BY window_partition

Ein optionales Argument, das den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel festlegt.

ORDER BY window_ordering

Sortiert die Zeilen innerhalb der einzelnen Partitionen.

Die LAG-Fensterfunktion unterstützt Ausdrücke, die einen der AWS Clean Rooms Datentypen verwenden. Der Rückgabetyp ist mit dem Typ von value_expr identisch.

Beispiele

Im folgenden Beispiel wird die Menge der Tickets gezeigt, die an den Käufer mit der Käufer-ID 3 verkauft wurden, sowie die Uhrzeit, zu der Käufer 3 die Tickets gekauft hat. Um jeden Verkauf mit dem vorherigen Kauf für Käufer 3 zu vergleichen, gibt die Abfrage für jeden Verkauf die vorherige Menge zurück, die verkauft wurde. Da vor dem 16.01.2008 kein Kauf stattfand, ist der erste Wert für die vorherige verkaufte Menge 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)
```

Funktion „LETZTES Fenster“

Bei einer bestimmten Anzahl von Zeilen gibt die Funktion LAST den Wert des Ausdrucks in Bezug auf die letzte Zeile im Frame zurück.

Informationen zur Auswahl der ersten Zeile im Rahmen finden Sie unter Funktion „ERSTES Fenster“.

Syntax

```
LAST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Argumente

expression

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird.

IGNORE NULLS

Die Funktion gibt den letzten Wert im Rahmen zurück, der nicht NULL ist (oder NULL, wenn alle Werte NULL sind).

RESPECT NULLS

Gibt an, dass bei der Bestimmung der zu verwendenden Zeile Nullwerte berücksichtigt werden AWS Clean Rooms sollen. Wenn Sie IGNORE NULLS nicht angeben, wird RESPECT NULLS standardmäßig unterstützt.

OVER

Führt die Fensterklauseln für die Funktion ein.

PARTITION BY expr_list

Definiert das Fenster für die Funktion in Bezug auf mindestens einen Ausdruck.

ORDER BY order_list

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn die PARTITION BY-Klausel nicht angegeben ist, sortiert ORDER BY die gesamte Tabelle. Wenn Sie eine ORDER BY-Klausel angeben, müssen Sie auch eine frame_clause angeben.

Die Ergebnisse sind von der Anordnung der Daten abhängig. Die Ergebnisse sind in den folgenden Fällen nicht deterministisch:

- Wenn keine ORDER BY-Klausel angegeben ist und eine Partition zwei verschiedene Werte für einen Ausdruck enthält
- Wenn der Ausdruck zu verschiedenen Werten ausgewertet wird, die demselben Wert in der ORDER BY-Liste entsprechen

frame_clause

Wenn eine ORDER BY-Klausel für eine Aggregationsfunktion verwendet wird, ist eine explizite Rahmenklausel erforderlich. Die Rahmenklausel gibt den Satz von Zeilen im Fenster einer Funktion genauer an, einschließlich oder ausschließlich Sätzen von Zeilen im geordneten Ergebnis. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe Übersicht über die Syntax von Fensterfunktionen.

Rückgabetyp

Diese Funktionen unterstützen Ausdrücke, die primitive AWS Clean Rooms Datentypen verwenden. Der Rückgabetyp ist mit dem Datentyp von expression identisch.

Beispiele

Im folgenden Beispiel wird die Sitzplatzkapazität für die einzelnen Veranstaltungsorte in der Tabelle VENUE zurückgegeben, wobei die Ergebnisse nach Kapazität (hoch zu niedrig) geordnet sind. Die Funktion LAST wird verwendet, um den Namen des Veranstaltungsorts auszuwählen, der der letzten Zeile im Frame entspricht: in diesem Fall der Zeile mit der geringsten Anzahl von Sitzplätzen. Die Ergebnisse werden nach Bundesstaat partitioniert. Wenn der Wert für VENUESTATE geändert wird, wird daher ein neuer letzter Wert ausgewählt. Der Fensterrahmen ist unbegrenzt. Daher wird für jede Zeile in jeder Partition derselbe letzte Wert ausgewählt.

Im Fall von Kalifornien wird `Shoreline Amphitheatre` für jede Zeile in der Partition zurückgegeben, da es die kleinste Zahl von Sitzplätzen hat (`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
...
```

Die Fensterfunktion LAST_VALUE

Bei einem geordneten Satz von Zeilen gibt die Funktion LAST_VALUE den Wert des Ausdrucks in Bezug auf die letzte Zeile im Rahmen zurück.

Informationen zur Auswahl der ersten Zeile im Rahmen finden Sie unter Die Fensterfunktion FIRST_VALUE.

Syntax

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
```

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

Argumente

expression

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird.

IGNORE NULLS

Die Funktion gibt den letzten Wert im Rahmen zurück, der nicht NULL ist (oder NULL, wenn alle Werte NULL sind).

RESPECT NULLS

Gibt an, dass bei der Bestimmung der zu verwendenden Zeile Nullwerte berücksichtigt werden AWS Clean Rooms sollen. Wenn Sie IGNORE NULLS nicht angeben, wird RESPECT NULLS standardmäßig unterstützt.

OVER

Führt die Fensterklauseln für die Funktion ein.

PARTITION BY expr_list

Definiert das Fenster für die Funktion in Bezug auf mindestens einen Ausdruck.

ORDER BY order_list

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn die PARTITION BY-Klausel nicht angegeben ist, sortiert ORDER BY die gesamte Tabelle. Wenn Sie eine ORDER BY-Klausel angeben, müssen Sie auch eine frame_clause angeben.

Die Ergebnisse sind von der Anordnung der Daten abhängig. Die Ergebnisse sind in den folgenden Fällen nicht deterministisch:

- Wenn keine ORDER BY-Klausel angegeben ist und eine Partition zwei verschiedene Werte für einen Ausdruck enthält
- Wenn der Ausdruck zu verschiedenen Werten ausgewertet wird, die demselben Wert in der ORDER BY-Liste entsprechen

frame_clause

Wenn eine ORDER BY-Klausel für eine Aggregationsfunktion verwendet wird, ist eine explizite Rahmenklausel erforderlich. Die Rahmenklausel gibt den Satz von Zeilen im Fenster einer Funktion genauer an, einschließlich oder ausschließlich Sätzen von Zeilen im geordneten Ergebnis. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe Übersicht über die Syntax von Fensterfunktionen.

Rückgabetyp

Diese Funktionen unterstützen Ausdrücke, die primitive AWS Clean Rooms Datentypen verwenden. Der Rückgabetyp ist mit dem Datentyp von expression identisch.

Beispiele

Im folgenden Beispiel wird die Sitzplatzkapazität für die einzelnen Veranstaltungsorte in der Tabelle VENUE zurückgegeben, wobei die Ergebnisse nach Kapazität (hoch zu niedrig) geordnet sind. Die Funktion LAST_VALUE wird verwendet, um den Namen des Veranstaltungsorts auszuwählen, der der letzten Zeile im Rahmen entspricht, in diesem Fall der Zeile mit der geringsten Anzahl von Sitzplätzen. Die Ergebnisse werden nach Bundesstaat partitioniert. Wenn der Wert für VENUESTATE geändert wird, wird daher ein neuer letzter Wert ausgewählt. Der Fensterrahmen ist unbegrenzt. Daher wird für jede Zeile in jeder Partition derselbe letzte Wert ausgewählt.

Im Fall von Kalifornien wird `Shoreline Amphitheatre` für jede Zeile in der Partition zurückgegeben, da es die kleinste Zahl von Sitzplätzen hat (`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
...
```

Die Fensterfunktion LEAD

Die Fensterfunktion LEAD gibt die Werte für eine Zeile in einem bestimmten Offset unterhalb (nach) der aktuellen Zeile in der Partition zurück.

Syntax

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Argumente

value_expr

Die Zielspalte oder der Ausdruck, für die/den die Funktion ausgeführt wird.

offset

Ein optionaler Parameter, der die Anzahl der Zeilen unterhalb der aktuellen Zeile angibt, für die Werte zurückgegeben werden sollen. Beim Offset kann es sich um eine ganzzahlige Konstante oder um einen Ausdruck handeln, der zu einer Ganzzahl ausgewertet wird. Wenn Sie keinen Offset angeben, AWS Clean Rooms verwendet 1 als Standardwert. Ein Offset von `0` gibt die aktuelle Zeile an.

IGNORE NULLS

Eine optionale Angabe, die angibt, dass Nullwerte bei der Bestimmung der zu verwendenden Zeile übersprungen werden AWS Clean Rooms sollen. Wenn IGNORE NULLS nicht angegeben wird, werden Null-Werte berücksichtigt.

> **Note**
>
> Sie können einen NVL- oder COALESCE-Ausdruck verwenden, um die Null-Werte durch einen anderen Wert zu ersetzen.

RESPECT NULLS

Gibt an, dass Nullwerte bei der Bestimmung der zu verwendenden Zeile berücksichtigt werden AWS Clean Rooms sollen. Wenn Sie IGNORE NULLS nicht angeben, wird RESPECT NULLS standardmäßig unterstützt.

OVER

Gibt die Fensterpartitionierung und -anordnung an. Die OVER-Klausel darf keine Fensterrahmenspezifikation enthalten.

PARTITION BY window_partition

Ein optionales Argument, das den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel festlegt.

ORDER BY window_ordering

Sortiert die Zeilen innerhalb der einzelnen Partitionen.

Die LEAD-Fensterfunktion unterstützt Ausdrücke, die einen der AWS Clean Rooms Datentypen verwenden. Der Rückgabetyp ist mit dem Typ von value_expr identisch.

Beispiele

Im folgenden Beispiel wird die Provision für Veranstaltungen in der Tabelle SALES angegeben, für die am 1. und 2. Januar 2008 Tickets verkauft wurden, sowie die Provision, die für verkaufte Tickets im anschließenden Verkauf gezahlt wurden.

```
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)
```

Die Fensterfunktion PERCENT_RANK

Berechnet den prozentualen Rang einer bestimmten Zeile. Der prozentuale Rang wird anhand der folgenden Formel festgelegt:

`(x - 1) / (the number of rows in the window or partition - 1)`

wobei x der Rang der aktuellen Zeile ist. Der folgende Datensatz zeigt die Verwendung dieser Formel:

```
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
```

Der Rückgabewertbereich ist 0 bis 1 (einschließlich). Die erste Zeile in jedem Satz besitzt den PERCENT_RANK 0.

Syntax

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Argumente

( )

Die Funktion verwendet keine Argumente. Es ist jedoch eine leere Klammer erforderlich.

OVER

Eine Klausel, die die Fensterpartitionierung angibt. Die OVER-Klausel darf keine Fensterrahmenspezifikation enthalten.

PARTITION BY partition_expression

Optional. Ein Ausdruck, der den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel festlegt.

ORDER BY order_list

Optional. Der Ausdruck, anhand dessen der prozentuale Rang berechnet wird. Der Datentyp des Ausdrucks muss entweder numerisch sein oder implizit in einen solchen konvertierbar sein. Wenn ORDER BY ausgelassen wird, ist der Rückgabewert für alle Zeilen 0.

Wenn ORDER BY nicht zu einer spezifischen Reihenfolge führt, ist die Reihenfolge der Zeilen nicht deterministisch. Weitere Informationen finden Sie unter Spezifisches Anordnen von Daten für Fensterfunktionen.

Rückgabetyp

FLOAT8

Beispiele

Im folgenden Beispiel wird der prozentuale Rang der Verkaufsmengen für die einzelnen Verkäufer berechnet:

```
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
```

Eine Beschreibung der Tabelle WINSALES finden Sie unter Beispieltabelle mit Beispielen von Fensterfunktionen.

Die Fensterfunktion RANK

Die Fensterfunktion RANK legt den Rang eines Werts in einer Gruppe von Werten fest, basierend auf dem ORDER BY-Ausdruck in der OVER-Klausel. Wenn die optionale PARTITION BY-Klausel vorhanden ist, wird die Rangfolge für jede Gruppe von Zeilen neu festgelegt. Zeilen mit gleichen Werten für die Rangkriterien erhalten denselben Rang. AWS Clean Roomsaddiert die Anzahl der gleichwertigen Zeilen zum gleichen Rang, um den nächsten Rang zu berechnen, sodass es sich bei den Rängen möglicherweise nicht um fortlaufende Zahlen handelt. Wenn beispielsweise zwei Zeilen den Rang 1 erhalten, ist der nächste Rang 3.

RANK unterscheidet sich in einer Hinsicht von Die Fensterfunktion DENSE_RANK: Wenn zwei oder mehr Zeilen den gleichen Rang erhalten, entsteht bei DENSE_RANK in der Rangfolge der Werte keine Lücke. Wenn beispielsweise zwei Zeilen den Rang 1 erhalten, ist der nächste Rang 2.

Sie können in derselben Abfrage Rangfestlegungsfunktionen mit unterschiedlichen PARTITION BY- und ORDER BY-Klauseln verwenden.

Syntax

```
RANK () OVER
(
[ PARTITION BY expr_list ]
```

```
[ ORDER BY order_list ]
)
```

Argumente

( )

Die Funktion verwendet keine Argumente. Es ist jedoch eine leere Klammer erforderlich.

OVER

Die Fensterklauseln für die Funktion RANK.

PARTITION BY expr_list

Optional. Ein oder mehrere Ausdrücke, der/die das Fenster definiert/definieren.

ORDER BY order_list

Optional. Definiert die Spalten, auf denen die Rangfestlegungswerte basieren. Wenn PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle. Wenn ORDER BY ausgelassen wird, ist der Rückgabewert für alle Zeilen 1.

Wenn ORDER BY nicht zu einer spezifischen Reihenfolge führt, ist die Reihenfolge der Zeilen nicht deterministisch. Weitere Informationen finden Sie unter Spezifisches Anordnen von Daten für Fensterfunktionen.

Rückgabetyp

INTEGER

Beispiele

Im folgenden Beispiel wird die Tabelle nach der verkauften Menge (standardmäßig in aufsteigender Reihenfolge) geordnet und jeder Zeile einen Rang zugewiesen. Der Rangwert 1 ist der Wert mit dem höchsten Rang. Die Ergebnisse werden sortiert, nachdem die Fensterfunktionsergebnisse angewendet wurden:

```
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)
```

Beachten Sie, dass die äußere ORDER BY-Klausel in diesem Beispiel die Spalten 2 und 1 enthält, um sicherzustellen, dass bei jeder Ausführung dieser Abfrage konsistent sortierte Ergebnisse AWS Clean Rooms zurückgegeben werden. Zeilen mit den Umsätzen IDs 10001 und 10006 haben beispielsweise identische QTY- und RNK-Werte. Durch die Anordnung des endgültigen Ergebnissatzes nach Spalte 1 wird sichergestellt, dass die Zeile 10001 stets vor der Zeile 10006 angeordnet wird. Eine Beschreibung der Tabelle WINSALES finden Sie unter Beispieltabelle mit Beispielen von Fensterfunktionen.

Im folgenden Beispiel wird die Anordnung für die Fensterfunktion () umgekehrt. (`order by qty desc`). Jetzt wird der höchste Rangwert auf den größten QTY-Wert angewendet.

```
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)
```

Eine Beschreibung der Tabelle WINSALES finden Sie unter Beispieltabelle mit Beispielen von Fensterfunktionen.

Im folgenden Beispiel wird die Tabelle nach SELLERID partitioniert, die einzelnen Partitionen nach Menge (in absteigender Reihenfolge) geordnet und jeder Zeile ein Rang zugewiesen. Die Ergebnisse werden sortiert, nachdem die Fensterfunktionsergebnisse angewendet wurden.

```
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)
```

Die Fensterfunktion ROW_NUMBER

Legt die Ordnungszahl der aktuellen Zeile innerhalb einer Gruppe von Zeilen fest, ab 1 zählend, basierend auf dem ORDER BY-Ausdruck in der OVER-Klausel. Wenn die optionale PARTITION BY-Klausel vorhanden ist, werden die Ordnungszahlen für jede Gruppe von Zeilen neu festgelegt. Zeilen mit gleichen Werten für die ORDER BY-Ausdrücke erhalten auf nicht deterministische Weise unterschiedliche Zeilenzahlen.

Syntax

```
ROW_NUMBER () OVER
```

```
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Argumente

( )

Die Funktion verwendet keine Argumente. Es ist jedoch eine leere Klammer erforderlich.

OVER

Die Fensterklauseln für die Funktion ROW_NUMBER.

PARTITION BY expr_list

Optional. Ein oder mehrere Ausdrücke, der/die die Funktion ROW_NUMBER definiert/definieren.

ORDER BY order_list

Optional. Der Ausdruck, der die Spalten definiert, auf denen die Zeilennummern basieren. Wenn PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle.

Wenn ORDER BY nicht zu einer eindeutigen Reihenfolge führt oder ausgelassen wird, ist die Reihenfolge der Zeilen nicht deterministisch. Weitere Informationen finden Sie unter Spezifisches Anordnen von Daten für Fensterfunktionen.

Rückgabetyp

BIGINT

Beispiele

Im folgenden Beispiel werden die Tabelle nach SELLERID partitioniert und die einzelnen Partitionen nach QTY angeordnet (in aufsteigender Reihenfolge). Anschließend wird jeder Zeile eine Zeilennummer zugewiesen. Die Ergebnisse werden sortiert, nachdem die Fensterfunktionsergebnisse angewendet wurden.

```
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)
```

Eine Beschreibung der Tabelle WINSALES finden Sie unter Beispieltabelle mit Beispielen von Fensterfunktionen.

AWS Clean Rooms Spark-SQL-Bedingungen

Bedingungen sind Aussagen aus einem oder mehreren Ausdrücken und logischen Operatoren, die als Ergebnis „Wahr“, „Falsch“ oder „Unbekannt“ ausgewertet werden. Bedingungen werden manchmal auch als Prädikate bezeichnet.

Syntax

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

> **Note**
>
> Bei Vergleichen von Zeichenfolgen und bei LIKE-Patternmatches wird die Groß-/Kleinschreibung berücksichtigt. Zum Beispiel entsprechen sich 'A' und 'a' nicht. Wenn Sie

beim Patternmatching die Groß-/Kleinschreibung nicht berücksichtigen möchten, werden Sie statt LIKE das Prädikat ILIKE.

Die folgenden SQL-Bedingungen werden in AWS Clean Rooms Spark SQL unterstützt.

Themen

Vergleichsoperatoren

Vergleichsbedingungen machen eine Aussage bezüglich der logischen Beziehungen zwischen zwei Werten. Alle Vergleichsbedingungen sind binäre Operatoren mit einem Booleschen Rückgabewert.

AWS Clean Rooms Spark SQL unterstützt die in der folgenden Tabelle beschriebenen Vergleichsoperatoren.

Operator	Syntax	Beschreibung
!	`!expression`	Der logische `NOT` Operator. Wird verwendet, um einen booleschen Ausdruck zu negieren, was bedeutet, dass er das Gegenteil des Werts des Ausdrucks zurückgibt. Das! Der Operator kann auch mit anderen logischen Operatoren wie AND und OR kombiniert werden,

Operator	Syntax	Beschreibung
		um komplexere boolesche Ausdrücke zu erzeugen.
<	`a < b`	Der Vergleichsoperator „Weniger als“. Wird verwendet, um zwei Werte zu vergleichen und festzustellen, ob der Wert auf der linken Seite kleiner als der Wert auf der rechten Seite ist.
>	`a > b`	Der Vergleichsoperator „Größer als“. Wird verwendet, um zwei Werte zu vergleichen und festzustellen, ob der Wert auf der linken Seite größer als der Wert auf der rechten Seite ist.
<=	`a <= b`	Der Vergleichsoperator „kleiner als“ oder „gleich“. Wird verwendet, um zwei Werte zu vergleichen, und gibt zurück, `true` ob der Wert auf der linken Seite kleiner oder gleich dem Wert auf der rechten Seite ist, und `false` andernfalls.
>=	`a >= b`	Der Vergleichsoperator „Größer als oder gleich“. Wird verwendet, um zwei Werte zu vergleichen und festzustellen, ob der Wert auf der linken Seite größer oder gleich dem Wert auf der rechten Seite ist.

Operator	Syntax	Beschreibung
=	`a = b`	Der Gleichheitsverglei chsoperator, der zwei Werte vergleicht und zurückgibt, `true` ob sie gleich sind, und `false` andernfalls.
<> oder !=	`a <> b` oder `a != b`	Der Vergleichsoperator „ungleich“, der zwei Werte vergleicht und zurückgibt, `true` wenn sie nicht gleich sind, und `false` andernfalls.

Operator	Syntax	Beschreibung
==	`a == b`	Der Standardoperator für den Gleichheitsvergleich, der zwei Werte vergleicht und zurückgibt, `true` ob sie gleich sind, und `false` andernfalls. **Note** Der Operator == unterscheidet beim Vergleich von Zeichenkettenwerte n zwischen Groß- und Kleinschreibung. Wenn Sie einen Vergleich ohne Berücksichtigung der Groß- und Kleinschr eibung durchführ en müssen, können Sie Funktionen wie UPPER () oder LOWER () verwenden, um die Werte vor dem Vergleich in dieselbe Groß- und Kleinschr eibung umzuwandeln.

Beispiele

Einige einfache Beispiele für Vergleichsbedingungen:

```
a = 5
a < b
```

```
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

Die folgende Abfrage gibt die ID-Werte für alle Eichhörnchen zurück, die derzeit nicht auf Futtersuche sind.

```
SELECT id FROM squirrels
WHERE !is_foraging
```

Die folgende Abfrage gibt Veranstaltungsorte mit mehr als 10.000 Sitzplätzen aus der VENUE-Tabelle zurück:

```
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)
```

In diesem Beispiel werden diejenigen Benutzer (USERID) aus der Tabelle USERS ausgewählt, die Rockmusik schätzen:

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
```

```
13
16
(5 rows)
```

In diesem Beispiel werden diejenigen Benutzer(USERID) aus der Tabelle USERS ausgewählt, von denen nicht bekannt ist, ob sie Rockmusik schätzen:

```
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
```

Beispiele mit einer TIME-Spalte

Die folgende Beispieltabelle TIME_TEST enthält eine Spalte TIME_VAL (Typ TIME) mit drei eingefügten Werten.

```
select time_val from time_test;

time_val
--------------------
20:00:00
00:00:00.5550
00:58:00
```

Im folgenden Beispiel werden die Stunden aus jedem timetz_val extrahiert.

```
select time_val from time_test where time_val < '3:00';
```

```
   time_val
---------------
 00:00:00.5550
 00:58:00
```

Im folgenden Beispiel werden zwei Zeitliterale verglichen.

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
 t
```

Beispiele mit einer TIMETZ-Spalte

Die folgende Beispieltabelle TIMETZ_TEST enthält eine Spalte TIMETZ_VAL (Typ TIMETZ) mit drei eingefügten Werten.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

Im folgenden Beispiel werden nur die TIMETZ-Werte ausgewählt, die kleiner als sind `3:00:00 UTC`. Der Vergleich erfolgt nach der Umwandlung des Wertes in UTC.

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

Im folgenden Beispiel werden zwei TIMETZ-Literale verglichen. Beim Vergleich wird die Zeitzone ignoriert.

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

 ?column?
----------
```

```
t
```

Logische Bedingungen

Logische Bedingungen führen die Ergebnisse zweier Bedingungen zu einem Ergebnis zusammen. Alle logischen Bedingungen sind binäre Operatoren mit einem Booleschen Rückgabewert.

Syntax

```
expression
{ AND | OR }
expression
NOT expression
```

Bei logischen Bedingungen wird eine dreiwertige Boolesche Logik verwendet, bei der der Wert „Null“ als „unbekannt“ interpretiert wird. Die folgende Tabelle beschreibt die Ergebnisse von logischen Bedingungen, wobei `E1` und `E2` Ausdrücke sind:

E1	E2	E1 AND E2	E1 OR E2	NOT E2
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

Der Operator NOT wird vor AND ausgewertet, und AND vor OR. Diese Auswertungsreihenfolge kann durch Klammerung außer Kraft gesetzt werden.

Beispiele

In dem folgenden Beispiel werden USERID und USERNAME aus der Tabelle USERS zurückgegeben, die sowohl Las Vegas als auch Sport mögen:

```
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)
```

In dem nächsten Beispiel werden USERID und USERNAME aus der Tabelle USERS zurückgegeben, die Las Vegas oder Sport mögen: Diese Abfrage gibt alle Ergebnisse des vorangehenden Beispiels, zuzüglich der Benutzer, die Las Vegas mögen, zuzüglich der Benutzer, die Sport mögen.

```
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)
```

In der folgenden Abfrage wird die Bedingung OR in Klammern gesetzt, um alle Veranstaltungen zu suchen, die in New York oder in Kalifornien stattfinden, und bei denen Macbeth gegeben wird:

```
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
...
```

Wenn die Klammerung in dem vorangehenden Beispiel entfernt wird, ändert sich die der bei der Auswertung ermittelte Wert und damit das Ergebnis der Abfrage.

In dem folgenden Beispiel wird der Operator NOT verwendet.

```
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
...
```

Im folgenden Beispiel wird eine NOT-Bedingung verwendet, gefolgt von einer AND-Bedingung:

```
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)
```

Patternmatching-Bedingungen

Ein Mustervergleichsoperator durchsucht eine Zeichenfolge nach einem im bedingten Ausdruck angegebenen Muster und gibt je nachdem, ob er eine Übereinstimmung findet, wahr oder falsch zurück. AWS Clean Rooms Spark SQL verwendet die folgenden Methoden für den Mustervergleich:

- LIKE-Ausdrücke

  Der LIKE-Operator vergleicht einen Zeichenfolgenausdruck (beispielsweise einen Spaltennamen) mit einem Muster, in dem die Platzhalterzeichen % (Prozentzeichen) und _ (Unterstrich) verwendet werden können. Beim LIKE-Patternmatching wird jeweils die gesamte Zeichenfolge durchsucht. LIKE führt eine Übereinstimmung unter Berücksichtigung der Groß-/Kleinschreibung durch.

Themen

- LIKE
- RLIKE

LIKE

Der LIKE-Operator vergleicht einen Zeichenfolgenausdruck (beispielsweise einen Spaltennamen) mit einem Muster, in dem die Platzhalterzeichen % (Prozentzeichen) und _ (Unterstrich) verwendet werden können. Beim LIKE-Patternmatching wird jeweils die gesamte Zeichenfolge durchsucht. Um

für ein Muster anzugeben, dass es an einer beliebigen Stelle innerhalb der Zeichenfolge auftreten kann, muss es in Prozentzeichen eingeschlossen werden.

LIKE unterscheidet zwischen Groß- und Kleinschreibung.

Syntax

```
expression [ NOT ] LIKE | pattern [ ESCAPE 'escape_char' ]
```

Argumente

expression

Ein gültiger UTF8-Zeichenfolgenausdruck (beispielsweise ein Spaltenname).

LIKE

Bei LIKE wird beim Patternmatching die Groß-/Kleinschreibung berücksichtigt. Zur Durchführung eines Patternmatchingvorgangs ohne Berücksichtigung der Groß-/Kleinschreibung verwenden Sie die LOWER-Funktion für expression und pattern mit einer LIKE-Bedingung.

Im Gegensatz zu Vergleichsprädikaten wie = und <> ignorieren LIKE-Prädikate nachfolgende Leerzeichen nicht implizit. Um nachfolgende Leerzeichen zu ignorieren, verwenden Sie RTRIM, oder konvertieren Sie eine CHAR-Spalte explizit zu VARCHAR.

Der Operator entspricht LIKE. ~~ Außerdem entspricht der !~~ Operator NOT LIKE.

pattern

Ein gültiger UTF8-Zeichenfolgenausdruck mit dem Muster für das Patternmatching.

escape_char

Ein Zeichenfolgenausdruck zur Kennzeichnung von Metazeichen im Muster als Literal. Dies ist standardmäßig die Zeichenfolge \\ (doppelter umgekehrter Schrägstrich).

Wenn das Muster pattern keine Metazeichen enthält, ist wird das Muster als die Zeichenfolge selbst interpretiert. In diesem Fall liefert LIKE dasselbe Ergebnis wie der Gleichheitsoperator.

Die Zeichenfolgenausdrücke können vom Datentyp CHAR oder VARCHAR sein. Wenn unterschiedliche Datentypen verwendet werden, konvertiert AWS Clean Rooms pattern in den Datentyp des Ausdrucks expression.

LIKE unterstützt die folgenden Metazeichen in Mustern:

Operator	Beschreibung
%	Entspricht einer Folge von 0 oder mehr Zeichen.
_	Entspricht einem beliebigen Zeichen.

Beispiele

In der folgenden Tabelle werden Beispiele für Patternmatching mit LIKE dargestellt.

Ausdruck	Rückgabewert
`'abc' LIKE 'abc'`	Wahr
`'abc' LIKE 'a%'`	True
`'abc' LIKE '_B_'`	False
`'abc' LIKE 'c%'`	False

Das folgende Beispiel listet alle Städte auf, die mit „E“ beginnen:

```
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
...
```

Das folgende Beispiel listet alle Benutzer auf, deren Nachname „ten“ enthält:

```
select distinct lastname from users
```

```
where lastname like '%ten%' order by lastname;
lastname
-------------
Christensen
Wooten
...
```

Das folgende Beispiel findet Städte, deren drittes und viertes Zeichen „ea“ sind . :

```
select distinct city from users where city like '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

Im folgenden Beispiel wird die Standard-Escape-Zeichenfolge (\\) verwendet, um nach Zeichenfolgen zu suchen, die „start_“ (den Text `start` gefolgt von einem Unterstrich _) enthalten:

```
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)
```

Im folgenden Beispiel wird als Escape-Zeichenfolge ^ (das Caret-Zeichen) verwendet, und dann wird nach Zeichenfolgen gesucht, die „start_“ (den Text `start` gefolgt von einem Unterstrich _) enthalten:

```
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

Mit dem RLIKE-Operator können Sie überprüfen, ob eine Zeichenfolge einem angegebenen Muster für reguläre Ausdrücke entspricht.

Gibt zurück`true`, ob str übereinstimmt`regexp`, oder `false` nicht.

Syntax

```
rlike(str, regexp)
```

Argumente

str

Ein Zeichenkettenausdruck

Regexp

Ein Zeichenkettenausdruck. Die Regex-Zeichenfolge sollte ein regulärer Java-Ausdruck sein.

Zeichenkettenliterale (einschließlich Regex-Muster) sind in unserem SQL-Parser nicht maskiert. Um beispielsweise „\ abc“ zu entsprechen, kann ein regulärer Ausdruck für Regexp „^\ abc$“ lauten.

Beispiele

Im folgenden Beispiel wird der Wert des Konfigurationsparameters auf festgelegt. `spark.sql.parser.escapedStringLiterals` `true` Dieser Parameter ist spezifisch für die Spark-SQL-Engine. Der `spark.sql.parser.escapedStringLiterals` Parameter in Spark

SQL steuert, wie der SQL-Parser mit Escape-Zeichenkettenliteralen umgeht. Wenn dieser Wert auf gesetzt ist`true`, interpretiert der Parser Backslash-Zeichen (\) in Zeichenkettenliteralen als Escape-Zeichen, sodass Sie Sonderzeichen wie Zeilenumbrüche, Tabulatoren und Anführungszeichen in Ihre Zeichenkettenwerte aufnehmen können.

```
SET spark.sql.parser.escapedStringLiterals=true;
spark.sql.parser.escapedStringLiterals  true
```

Mit `spark.sql.parser.escapedStringLiterals=true` könnten Sie beispielsweise das folgende Zeichenkettenliteral in Ihrer SQL-Abfrage verwenden:

```
SELECT 'Hello, world!\n'
```

Das Zeilenumbruchzeichen \n würde in der Ausgabe als wörtliches Zeilenumbruchzeichen interpretiert werden.

Im folgenden Beispiel wird ein Musterabgleich mit regulären Ausdrücken durchgeführt. Das erste Argument wird an den RLIKE-Operator übergeben. Es ist eine Zeichenfolge, die einen Dateipfad darstellt, wobei der tatsächliche Benutzername durch das Muster '****' ersetzt wird. Das zweite Argument ist das Muster für reguläre Ausdrücke, das für den Abgleich verwendet wird. Die Ausgabe (`true`) gibt an, dass die erste Zeichenfolge (`'%SystemDrive%\Users\****'`) dem regulären Ausdrucksmuster (`'%SystemDrive%\\Users.*'`) entspricht.

```
SELECT rlike('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

BETWEEN-Bereichsbedingung

Eine `BETWEEN`-Bedingung überprüft, ob Ausdrücke Elemente aus einem Bereich von Werten enthalten, der über die Schlüsselwörter `BETWEEN` und `AND` angegeben wird.

Syntax

```
expression [ NOT ] BETWEEN expression AND expression
```

Der Datentyp der Ausdrücke kann ein numerischer, ein Zeichen- oder ein Datum/Uhrzeit-Typ sein, die Typen müssen jedoch untereinander kompatibel sein. Der angegebene Bereich versteht sich inklusive der angegebenen Werte.

Beispiele

Im ersten Beispiel werden die Transaktionen, bei denen 2, 3, oder 4 Tickets verkauft wurden, gezählt:

```
select count(*) from sales
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

Bei der Bereichsbedingung werden die Anfangs- und Endwerte mitgezählt (inklusiver Bereich).

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

Bei einer Bereichsbedingung muss der erste Wert stets der kleinere und der zweite der größere sein. In dem folgenden Beispiel werden immer 0 Zeilen zurückgegeben, weil die Werte in dem Bedingungsausdruck vertauscht wurden:

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

Wenn die Bedingung mit NOT negiert wird, werden nicht 0, sondern alle Zeilen gezählt:

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
```

```
172456
(1 row)
```

Die folgende Abfrage gibt eine Liste der Events mit 20.000 bis 50.000 Plätzen zurück:

```
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)
```

Das folgende Beispiel zeigt die Verwendung von BETWEEN für Datumswerte:

```
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
```

Beachten Sie, dass sich der BETWEEN-Bereich zwar inklusive der angegebenen Werte versteht, die Datumsangaben jedoch standardmäßig einen Zeitwert von 00:00:00 haben. Die einzige gültige Zeile für 3. Januar bei der Beispielabfrage wäre eine Zeile mit der Saletime (Verkaufszeit) `1/3/2008 00:00:00`.

„Null“-Bedingung

Das Tool NULL Bedingungstests auf Nullen, wenn ein Wert fehlt oder unbekannt ist.

Syntax

```
expression IS [ NOT ] NULL
```

Argumente

expression

Ein Ausdruck, beispielsweise eine Spalte.

IS NULL

Gibt „wahr“ zurück, wenn der Wert des Ausdrucks „Null“ ist, und „falsch“, wenn der Ausdruck einen Wert hat.

IS NOT NULL

Gibt „falsch“ zurück, wenn der Wert des Ausdrucks „Null“ ist, und „wahr“, wenn der Ausdruck einen Wert hat.

Beispiel

Dieses Beispiel gibt an, wie oft die Tabelle SALES im Feld QTYSOLD „Null“ enthält:

```
select count(*) from sales
where qtysold is null;
count
-------
0
(1 row)
```

EXISTS-Bedingung

Die EXISTS-Bedingung überprüft, ob eine Unterabfrage Zeilen zurückgibt, und gibt „wahr“ zurück, wenn die Unterabfrage mindestens eine Zeile zurückgibt. Bei Voranstellung von NOT wird gibt die Bedingung „wahr“ zurück, wenn die Unterabfrage 0 Zeilen zurückgibt.

Syntax

```
[ NOT ] EXISTS (table_subquery)
```

Argumente

EXISTS

Ist „wahr“, wenn die Unterabfrage table_subquery wenigstens eine Zeile zurückgibt.

NOT EXISTS

Ist „wahr“, wenn die Unterabfrage table_subquery keine Zeilen zurückgibt.

table_subquery

Eine Unterabfrage, die zu einer Tabelle mit einer oder mehreren Spalten und einer oder mehreren Zeilen ausgewertet wird.

Beispiel

In diesem Beispiel werden nacheinander die Identifier für jedes Datum aufgelistet, an dem ein Verkauf stattgefunden hat:

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
1827
1828
1829
...
```

IN-Bedingung

Importieren in &S3; IN Die Bedingung testet einen Wert auf seine Zugehörigkeit zu einer Gruppe von Werten oder zu einer Unterabfrage.

Syntax

```
expression [ NOT ] IN (expr_list | table_subquery)
```

Argumente

expression

Ein numerischer, Zeichen- oder Datum/Uhrzeit-Ausdruck, der anhand der Ausdrucksliste expr_list oder der Unterabfrage table_subquery ausgewertet wird, und der mit dem Datentyp der Liste bzw. Abfrage kompatibel sein muss.

expr_list

Ein oder mehrere kommagetrennte Ausdrücke oder ein oder mehrere Mengen von kommagetrennten Ausdrücken, als Klammerausdruck.

table_subquery

Eine Unterabfrage, die zu einer Tabelle mit einer oder mehreren Zeilen ausgewertet wird, aber höchstens eine Spalte in ihrer SELECT-Liste enthält.

IN | NOT IN

In gibt „wahr“ zurück, wenn der Ausdruck Element der Ausdrucksliste oder der Abfrage ist. NOT IN gibt „wahr“ zurück, wenn der Ausdruck darin nicht enthalten ist. IN und NOT IN geben NULL und keine Zeilen zurück, wenn der Ausdruck expression zu „Null“ ausgewertet wird, oder wenn in der Ausdrucksliste expr_list bzw. der Unterabfrage table_subquery keine übereinstimmenden Werte gefunden wurden und mindestens eine der verglichenen Zeilen als Ergebnis „Null“ zurückgegeben hat.

Beispiele

Die folgenden Bedingungen sind nur für die aufgelisteten Werte wahr:

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

Optimierung bei großen IN-Listen

Um die Abfrageleistung zu optimieren, werden IN-Listen mit mehr als 10 Werten intern als Zahlenarray ausgewertet. IN-Listen mit weniger Werten werden als Reihe von OR-Prädikaten ausgewertet. Diese Optimierung wird für die Datentypen SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP und TIMESTAMPTZ unterstützt.

Den Effekt dieser Optimierung verdeutlicht die Ausgabe, wenn ein EXPLAIN über der Abfrage ausgeführt wird. Beispiel:

```
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)
```

Verschachtelte Daten abfragen

AWS Clean Rooms bietet SQL-kompatiblen Zugriff auf relationale und verschachtelte Daten.

AWS Clean Rooms verwendet Punktnotation und Array-Index für die Pfadnavigation beim Zugriff auf verschachtelte Daten. Es ermöglicht auch FROM Klauselelemente, die über Arrays iteriert und für Operationen ohne Verschachtelung verwendet werden können. Die folgenden Themen enthalten Beschreibungen der verschiedenen Abfragemuster, die die Verwendung des array/struct/map Datentyps mit Pfad- und Arraynavigation, Entschachtelung und Verknüpfungen kombinieren.

Themen

Navigation

AWS Clean Rooms ermöglicht die Navigation in Arrays und Strukturen unter Verwendung der `[...]` Klammern- bzw. Punktnotation. Darüber hinaus können Sie die Navigation mithilfe von Punktschreibweise und Arrays mithilfe der Klammernotation in Strukturen mischen.

Example

In der folgenden Beispielabfrage wird beispielsweise davon ausgegangen, dass es sich bei der `c_orders` Array-Datenspalte um ein Array mit einer Struktur handelt und ein Attribut benannt `o_orderkey` ist.

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

Sie können die Punkt- und Klammernotationen in allen Arten von Abfragen verwenden, z. B. Filtern, Verknüpfen und Aggregation. Sie können diese Notationen in einer Abfrage verwenden, in der normalerweise Spaltenverweise vorhanden sind.

Example

Im folgenden Beispiel wird eine SELECT-Anweisung verwendet, die Ergebnisse filtert.

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0].o_orderkey IS NOT NULL;
```

Example

Im folgenden Beispiel wird die Klammer- und Punktnavigation in den Klauseln GROUP BY und ORDER BY verwendet.

```
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;
```

Aufheben der Verschachtelung von Abfragen

AWS Clean Rooms Aktiviert die Iteration über Arrays, um Abfragen zu entfernen. Dazu navigiert es im Array mithilfe der FROM-Klausel einer Abfrage.

Example

Das folgende Beispiel nutzt das vorherige Beispiel und iteriert über die Attributwerte für `c_orders`.

```
SELECT o FROM customer_orders_lineitem c, c.c_orders o;
```

Die Unnesting-Syntax ist eine Erweiterung der FROM-Klausel. In Standard-SQL bedeutet die FROM-Klausel `x (AS) y`, dass `y` über jedes Tupel in Beziehung `x` iteriert. In diesem Fall bezieht sich `x` auf eine Beziehung und `y` bezieht sich auf einen Alias für Beziehung `x`. In ähnlicher Weise `x (AS) y` bedeutet die Syntax des Aufhebens von Verschachtelungen mithilfe des FROM-Klauselelements, dass über jeden Wert im `y` Array-Ausdruck iteriert wird. `x` In diesem Fall `x` handelt es sich um einen Array-Ausdruck und `y` ist ein Alias für. `x`

Der linke Operand kann auch die Punkt- und Klammernotation für die reguläre Navigation verwenden.

Example

Im vorherigen Beispiel:

- `customer_orders_lineitem c` ist die Iteration über die `customer_order_lineitem` Basistabelle
- `c.c_orders o` ist die Iteration über `c.c_orders array`

Um über das Attribut `o_lineitems` zu iterieren, also ein Array innerhalb eines Arrays, fügen Sie mehrere Klauseln hinzu.

```
SELECT o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

AWS Clean Rooms unterstützt auch einen Array-Index bei der Iteration über das Array mit dem AT Schlüsselwort. Die Klausel `x AS y AT z` iteriert über ein Array `x` und generiert das Feld`z`, das den Array-Index darstellt.

Example

Das folgende Beispiel zeigt die Funktionsweise eines Array-Index.

```
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

Das folgende Beispiel iteriert über ein skalares Array.

```
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

Im folgenden Beispiel wird über ein Array mit mehreren Ebenen iteriert. Das Beispiel nutzt mehrere Klauseln zum Aufheben der Verschachtelung, um in die innersten Arrays zu iterieren. Die `f.multi_level_array` AS Ein Array iteriert über. `multi_level_array` Das Array AS Element ist die Iteration über die darin enthaltenen Arrays. `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)
```

Lax-Semantik

Standardmäßig geben Navigationsoperationen mit verschachtelten Datenwerten Null zurück, anstatt einen Fehler zurückzugeben, wenn die Navigation ungültig ist. Die Objektnavigation ist ungültig, wenn der verschachtelte Datenwert kein Objekt ist oder wenn der verschachtelte Datenwert ein Objekt ist, das aber nicht den in der Abfrage verwendeten Attributnamen enthält.

Example

Die folgende Abfrage greift beispielsweise auf einen ungültigen Attributnamen in der verschachtelten Datenspalte zu: `c_orders`

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

Die Array-Navigation gibt Null zurück, wenn der verschachtelte Datenwert kein Array ist oder der Array-Index außerhalb der Grenzen liegt.

Example

Die folgende Abfrage gibt Null zurück, weil die `c_orders[1][1]` Grenzwerte überschritten wurden.

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

Arten der Introspektion

Verschachtelte Datentypspalten unterstützen Inspektionsfunktionen, die den Typ und andere Typinformationen über den Wert zurückgeben. AWS Clean Rooms unterstützt die folgenden booleschen Funktionen für verschachtelte Datenspalten:

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

Alle diese Funktionen geben false zurück, wenn der Eingabewert null ist. IS_SCALAR, IS_OBJECT und IS_ARRAY schließen sich gegenseitig aus und decken alle möglichen Werte mit Ausnahme von null ab. Um die Typen abzuleiten, die den Daten entsprechen, AWS Clean Rooms verwendet die Funktion JSON_TYPEOF, die den Typ (die oberste Ebene) des verschachtelten Datenwerts zurückgibt, wie im folgenden Beispiel gezeigt:

```
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
```

Dokumenthistorie für die AWS Clean Rooms SQL-Referenz

In der folgenden Tabelle werden die Dokumentationsversionen für die AWS Clean Rooms SQL-Referenz beschrieben.

Um Benachrichtigungen über Aktualisierungen dieser Dokumentation zu erhalten, können Sie den RSS-Feed abonnieren. Um RSS-Updates zu abonnieren, müssen Sie ein RSS-Plugin für den von Ihnen verwendeten Browser aktiviert haben.

Änderung	Beschreibung	Datum
Spark SQL unterstützt Hints	AWS Clean Rooms Spark SQL unterstützt Abfragehinweise, um die Abfrageleistung zu optimieren und die Rechenkosten zu senken.	20. Januar 2026
Spark SQL unterstützt CACHE TABLE	AWS Clean Rooms Spark SQL unterstützt den Befehl CACHE TABLE, der es Kunden ermöglicht, bestehende Tabellen zwischenzuspeichern oder neue Tabellen aus Abfrageergebnissen zu erstellen und zwischenzuspeichern, um die Abfrageleistung zu verbessern.	22. Oktober 2025
Spark SQL unterstützt die Funktionen FIRST und LAST Window	AWS Clean Rooms Spark SQL unterstützt die folgenden Fensterfunktionen: FIRST und LAST.	12. Juni 2025
Aktualisierungen der Dokumentation zu den Funktionen von Spark SQL	Rein dokumentationsbezogenes Update, um die unterstützten Spark-SQL-	20. Mai 2025

	Funktionen genau wiederzug eben. Die Dokumentation für 25 nicht unterstützte Funktione n, darunter den Operator **<=>**, SIMILAR TO, LISTAGG und ARRAY_INSERT, wurde entfernt. Die Funktions namen von DATEADD zu DATE_ADD, DATEDIFF zu DATE_DIFF, ISNULL zu IS_NULL und ISNOTNULL zu IS_NOT_NULL wurden korrigiert. Ein Tippfehler in den DATE_PART-Beispielen wurde behoben.	
AWS Clean Rooms Spark SQL	Kunden können jetzt Abfragen mit einigen SQL-Bedingungen, Funktionen, Befehlen und Konventionen ausführen, die von der Spark SQL Analytics-Engine unterstützt werden.	29. Oktober 2024
SQL-Befehle und SQL-Funkt ionen — Update	Es wurden Beispiele für die JOIN-Klausel, den SET-Operator EXCEPT, den bedingten Ausdruck CASE und die folgenden Funktione n hinzugefügt: ANY_VALUE , NVL und COALESCE, NULLIF, CAST, CONVERT, CONVERT_TIMEZONE, EXTRACT, MOD, SIGN, CONCAT, FIRST_VALUE und LAST_VALUE.	28. Februar 2024

SQL-Funktionen - Update	AWS Clean Rooms unterstützt jetzt die folgenden SQL-Funktionen: Array, SUPER und VARBYTE. Die folgenden mathematischen Funktionen werden jetzt unterstützt: ACOS, ASIN, ATAN, COT ATAN2, DEXP, PI, POW, RADIANS und SIN. Die folgenden JSON-Funktionen werden jetzt unterstützt: CAN_JSON_PARSE, JSON_PARSE und JSON_SERIALIZE.	06. Oktober 2023
Unterstützung für verschachtelte Datentypen	AWS Clean Rooms unterstützt jetzt verschachtelte Datentypen.	30. August 2023
SQL-Namensregeln — Update	Änderung nur in der Dokumentation, um reservierte Spaltennamen zu verdeutlichen.	16. August 2023
Allgemeine Verfügbarkeit	Die AWS Clean Rooms SQL-Referenz ist jetzt allgemein verfügbar.	31. Juli 2023

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.