



SQL-Referenz

AWS Clean Rooms



AWS Clean Rooms: SQL-Referenz

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

SQL-Referenz	1
Konventionen für die SQL-Referenz	1
SQL-Namensregeln	2
Namen und Spalten der konfigurierten Tabellenzuordnungen	3
Literale	4
Reservierte Wörter	4
Datentypen	6
Multibyte-Zeichen	8
Numerische Typen	9
Zeichentypen	16
Datum-/Uhrzeittypen	18
Typ BOOLEAN	27
Typ SUPER	30
Verschachtelter Typ	31
Typ VARBYTE	32
Kompatibilität von Typen und Umwandlung zwischen Typen	35
SQL-Befehle	42
SELECT	42
SELECT list	42
WITH-Klausel	44
FROM-Klausel	48
WHERE-Klausel	57
GROUP BY-Klausel	58
HAVING-Klausel	63
Satzoperatoren	64
ORDER BY-Klausel	75
Beispiele für Unterabfragen	78
Korrelierte Unterabfragen	80
SQL-Funktionen	83
Aggregationsfunktionen	83
ANY_VALUE	84
APPROXIMATE PERCENTILE_DISC	86
AVG	88
BOOL_AND	89

BOOL_OR	90
COUNT Funktionen und COUNT DISTINCT	91
COUNT	92
LISTAGG	95
MAX	99
MEDIAN	101
MIN	103
PERCENTILE_CONT	105
STDDEV_SAMP und STDDEV_POP	108
SUM und SUM DISTINCT	110
VAR_SAMP und VAR_POP	111
Array-Funktionen	113
Array	113
array_concat	114
array_flatten	115
get_array_length	116
split_to_array	117
subarray	117
Bedingte Ausdrücke	118
CASE	119
COALESCE Ausdruck	121
GREATEST und LEAST	122
NVL und COALESCE	123
NVL2	125
NULLIF	127
Funktionen für die Datentypformatierung	129
CAST	130
CONVERT	134
TO_CHAR	136
TO_DATE	142
TO_NUMBER	143
Datum-/Uhrzeit-Formatzeichenfolgen	145
Numerische Formatzeichenfolgen	148
Formatierung im Teradata-Stil für numerische Daten	149
Datums- und Zeitfunktionen	156
Zusammenfassung der Datums- und Zeitfunktionen	157

Datums- und Zeitfunktionen in Transaktionen	159
Operator + (Verkettung)	160
ADD_MONTHS	161
CONVERT_TIMEZONE	162
CURRENT_DATE	165
DATEADD	166
DATEDIFF	171
DATE_PART	176
DATE_TRUNC	179
EXTRACT	182
GETDATE Funktion	186
SYSDATE	187
TIMEOFDAY	188
TO_TIMESTAMP	189
Datumsteile für Datums- oder Zeitstempelfunktionen	191
Hash-Funktionen	194
MD5	195
SHA	195
SHA1	196
SHA2	196
MURMUR3_32_HASH	197
JSON-Funktionen	200
CAN_JSON_PARSE	202
JSON_EXTRACT_ARRAY_ELEMENT_TEXT	203
JSON_EXTRACT_PATH_TEXT	204
JSON_PARSE	208
JSON_SERIALIZE	209
JSON_SERIALIZE_TO_VARBYTE	209
Mathematische Funktionen	210
Symbole für mathematische Operatoren	212
ABS	214
ACOS	215
ASIN zugeordnet	216
ATAN	216
ATAN2	217
CBRT	218

CEILING (oder CEIL)	219
COS	219
COT	220
DEGREES	221
DEXP	222
DLOG1	223
DLOG10	223
EXP	224
FLOOR	225
LN	226
LOG	228
MOD	228
PI	231
POWER	231
RADIANS	232
RANDOM	233
ROUND	236
SIGN	237
SIN	238
SQRT	239
TRUNC	241
Zeichenfolgefunktionen	243
Der Operator (Verkettung)	245
BTRIM	247
CHAR_LENGTH	248
CHARACTER_LENGTH	248
CHARINDEX	248
CONCAT	250
LEFT und RIGHT	253
LEN	254
LENGTH	256
LOWER	256
LPAD und RPAD	257
LTRIM	258
POSITION	261
REGEXP_COUNT	262

REGEXP_INSTR	265
REGEXP_REPLACE	268
REGEXP_SUBSTR	271
REPEAT	274
REPLACE	275
REPLICATE	277
REVERSE	277
RTRIM	278
SOUNDEX	280
SPLIT_PART	282
STRPOS	284
SUBSTR	286
SUBSTRING	286
TEXTLEN	290
TRANSLATE	290
TRIM	293
UPPER	294
Funktionen für SUPER-Typinformationen	295
DECIMAL_PRECISION	296
DECIMAL_SCALE	297
IS_ARRAY	298
IS_BIGINT	299
IS_CHAR	300
IS_DECIMAL	301
IS_FLOAT	302
IS_INTEGER	303
IS_OBJECT	304
IS_SCALAR	305
IS_SMALLINT	306
IS_VARCHAR	306
JSON_TYPEOF	307
VARBYTE-Funktionen	308
FROM_HEX	308
FROM_VARBYTE	309
TO_HEX	310
TO_VARBYTE	311

Fensterfunktionen	312
Übersicht über die Syntax von Fensterfunktionen	313
Spezifisches Anordnen von Daten für Fensterfunktionen	317
Unterstützte Funktionen	318
Beispieltabelle mit Beispielen von Fensterfunktionen	320
AVG	320
COUNT	322
CUME_DIST	325
DENSE_RANK	327
FIRST_VALUE	329
LAG	332
LAST_VALUE	334
LEAD	337
LISTAGG	339
MAX	343
MEDIAN	345
MIN	348
NTH_VALUE	350
NTILE	353
PERCENT_RANK	354
PERCENTILE_CONT	356
PERCENTILE_DISC	361
RANK	363
RATIO_TO_REPORT	366
ROW_NUMBER	368
STDDEV_SAMP und STDDEV_POP	370
SUM	372
VAR_SAMP und VAR_POP	376
SQL-Bedingungen	378
Vergleichsbedingungen	378
Nutzungshinweise	379
Beispiele	380
Beispiele mit einer TIME-Spalte	381
Beispiele mit einer TIMETZ-Spalte	382
Logische Bedingungen	383
Syntax	383

Patternmatching-Bedingungen	386
LIKE	387
SIMILAR TO	391
BETWEEN-Bereichsbedingung	395
Syntax	395
Beispiele	395
„Null“-Bedingung	397
Syntax	397
Argumente	397
Beispiel	398
EXISTS-Bedingung	398
Syntax	398
Argumente	398
Beispiel	399
IN-Bedingung	399
Syntax	399
Argumente	399
Beispiele	400
Optimierung bei großen IN-Listen	400
Syntax	401
Verschachtelte Daten abfragen	402
Navigation	402
Aufheben der Verschachtelung von Abfragen	403
Lax-Semantik	405
Arten der Introspektion	406
Dokumentverlauf	408
.....	cdx

Überblick über SQL in AWS Clean Rooms

Willkommen bei der AWS Clean Rooms SQL-Referenz.

AWS Clean Rooms basiert 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, Benennungsregeln und Datentypen:

Themen

- [Konventionen für die SQL-Referenz](#)
- [SQL-Namensregeln](#)
- [Datentypen](#)

Um sich mit den SQL-Befehlen, den Typen von SQL-Funktionen und den SQL-Bedingungen vertraut zu machen, die Sie verwenden können AWS Clean Rooms, lesen Sie sich die folgenden Themen durch:

- [SQL-Befehle in AWS Clean Rooms](#)
- [SQL-Funktionen in AWS Clean Rooms](#)
- [SQL-Bedingungen in AWS Clean Rooms](#)

Weitere Informationen AWS Clean Rooms dazu finden Sie im [AWS Clean Rooms Benutzerhandbuch](#) und in der [AWS Clean Rooms API-Referenz](#).

Konventionen für die SQL-Referenz

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

Zeichen	Beschreibung
GROSSBUCHSTABEN	Wörter in Großbuchstaben sind Schlüsselwörter.
[]	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 aufgelistet 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-Nennungsregeln in erläutertAWS Clean Rooms.

Namen und Spalten der konfigurierten Tabellenzuordnungen

Mitglieder, die Abfragen durchführen können, verwenden konfigurierte Tabellenzuordnungsamen als Tabellennamen in Abfragen. Namen konfigurierter Tabellenzuordnungen und konfigurierte Tabellenspalten können in Abfragen als Alias verwendet werden.

Die folgenden Benennungsregeln gelten für konfigurierte Tabellenzuordnungsamen, konfigurierte Tabellenspaltenamen und Aliase:

- Sie dürfen nur alphanumerische Zeichen, Unterstriche (_) oder Bindestriche (-) verwenden, dürfen aber nicht mit einem Bindestrich beginnen oder enden.
- (Nur benutzerdefinierte Analyseregeln) Sie können das Dollarzeichen (\$) verwenden, aber kein Muster verwenden, 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 sich der Zeichenketteninhalt zusammensetzt
- ein Dollarzeichen (\$)
- das gleiche Etikett, mit dem der Dollarkurs begann
- ein Dollarzeichen

Beispiel: `$$invalid$$`

- Sie dürfen keine aufeinanderfolgenden Bindestriche (-) enthalten.
- Sie können nicht mit einem der folgenden Präfixe beginnen:

`padb_`, `pg_`, `stcs_`, `stl_`, `stll_`, `stv_`, `svcs_`, `svl_`, `svv_`, `sys_`, `systable_`

- Sie dürfen keine umgekehrten Schrägstriche (\), Anführungszeichen (') oder Leerzeichen enthalten, die nicht in doppelten Anführungszeichen stehen.
- 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](#) muss in doppelten Anführungszeichen (" „) stehen.

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

Literale

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

Die folgenden Benennungsregeln gelten für Literale inAWS Clean Rooms:

- Numerische Literale, Zeichen- und Datums-, Uhrzeit- und Zeitstempel-Literale werden unterstützt.
- NurTAB,CARRIAGE RETURN(CR) undLINE FEED(LF) Unicode-Steuerzeichen aus der allgemeinen Unicode-Kategorie (Cc) werden unterstützt.
- Direkte Verweise auf Literale in der Projektionsliste werden in der SELECT-Anweisung nicht unterstützt.

Beispiele:

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

Reservierte Wörter

Im Folgenden finden Sie eine Liste reservierter Wörter inAWS Clean Rooms.

AES128	DELTA32KDESC	LEADING	PRIMARY
--------	--------------	---------	---------

AES256ALL	DISTINCT	LEFTLIKE	RAW
ALLOWOVERWRITEANALYSE	DO	LIMIT	READRATIO
ANALYZE	DISABLE	LOCALTIME	RECOVERREFERENCES
AND	ELSE	LOCALTIMESTAMP	REJECTLOG
ANY	EMPTYASNULLENABLE	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
BLANKSASNULLBOTH	FREEZE	NEW	SYSDATESYSTEM
BYTEDICT	FROM	NOT	TABLE
BZIP2CASE	FULL	NOTNULL	TAG
CAST	GLOBALDICT256	NULL	TDES
CHECK	GLOBALDICT64KGRANT	NULLSOFF	TEXT255
COLLATE	GROUP	OFFLINEOFFSET	TEXT32KTHEN

COLUMN	GZIPHAVING	OID	TIMESTAMP
CONSTRAINT	IDENTITY	OLD	TO
CREATE	IGNOREILIKE	ON	TOPTRAILING
CREDENTIALS LSCROSS	IN	ONLY	TRUE
CURRENT_DATE	INITIALLY	OPEN	TRUNCATEC OLUMNSUNION
CURRENT_TIME	INNER	OR	UNIQUE
CURRENT_TIMESTAMP	INTERSECT	ORDER	UNNEST
CURRENT_USER	INTERVAL	OUTER	USING
CURRENT_USER_ID SER_IDDEFAULT	INTO	OVERLAPS	VERBOSE
DEFERRABLE	IS	PARALLEL PARTITION	WALLETWHEN
DEFLATE	ISNULL	PERCENT	WHERE
DEFRAG	JOIN	PERMISSIONS	WITH
DELTA	LANGUAGE	PIVOTPLACING	WITHOUT

Datentypen

Jeder Wert, der AWS Clean Rooms gespeichert oder abgerufen wird, hat einen Datentyp mit einem festen Satz zugeordneter Eigenschaften. Datentypen werden deklariert, wenn Tabellen erstellt werden. Ein Datentyp beschränkt die Gruppe von Werten, die eine Spalte oder ein Argument enthalten kann.

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

Datentyp	Aliasnamen	Beschreibung
ARRAY	Nicht zutreffend	Verschachtelter Array-Datentyp
BIGINT	Nicht zutreffend	8-Byte-Ganzzahl mit Vorzeichen
BOOLEAN	BOOL	Logischer/Boolescher Wert (wahr/falsch)
CHAR	CHARACTER	Zeichenfolge mit fester Länge
DATUM	Nicht zutreffend	Kalenderdatum (Jahr, Monat, Tag)
DECIMAL	NUMERIC	Genauer Zahlenwert mit wählbarer Genauigkeit
DOUBLE PRECISION	FLOAT8, FLOAT	Gleitkommazahl mit doppelter Genauigkeit
INTEGER	INT	4-Byte-Ganzzahl mit Vorzeichen
MAP	Nicht zutreffend	Ordnen Sie den verschachtelten Datentyp zu
REAL	FLOAT4	Gleitkommazahl mit einfacher Genauigkeit
SMALLINT	Nicht zutreffend	2-Byte-Ganzzahl mit Vorzeichen
STRUCT	Nicht zutreffend	Verschachtelter Datentyp Struct
SUPER	Nicht zutreffend	Superset-Datentyp, der alle skalaren Typen umfasst, AWS Clean Rooms einschließlich

Datentyp	Aliasnamen	Beschreibung
		komplexer Typen wie ARRAY und STRUCTS.
TIME	Nicht zutreffend	Uhrzeit
TIMETZ	Nicht zutreffend	Uhrzeit mit Zeitzone
VARBYTE	VARBINARY, BINARY VARYING	Binärwert mit variabler Länge
VARCHAR	CHARAKTER VARIIEREND	Zeichenfolge mit variabler Länge und benutzerdefinierte m Grenzwert

Note

Die verschachtelten Datentypen ARRAY, STRUCT und MAP sind derzeit nur für die benutzerdefinierte Analyseregeln 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 (Bytensequenzen: ED A0 80–ED BF BF)

Der Datentyp CHAR bietet keine Unterstützung für Multibyte-Zeichen.

Numerische Typen

Themen

- [Ganzzahl-Typen](#)
- [Typ DECIMAL oder NUMERIC](#)
- [Hinweise zur Verwendung von 128-Bit-DECIMAL- oder -NUMERIC-Spalten](#)
- [Gleitkommazahl-Typen](#)
- [Berechnungen mit numerischen Werten](#)

Numerische Datentypen sind Ganzzahlen, Dezimalzahlen und Gleitkommazahlen.

Ganzzahl-Typen

Verwenden Sie die Datentypen SMALLINT, INTEGER und BIGINT, um Ganzzahlen aus verschiedenen Bereichen zu speichern. Sie können keine Werte außerhalb des zulässigen Bereichs für jeden Typ speichern.

Name	Speicher	Bereich
SMALLINT	2 Bytes	-32768 bis +32767
INTEGER oder INT	4 Bytes	-2147483648 bis +2147483647
BIGINT	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 die *Genauigkeit* (precision) und die *Dezimalstellen* bzw. Nachkommastellen (scale) angeben:

```
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 Skalierung 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 Annäherungen 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 FLOAT4 angeben.

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 FLOAT8 angeben.

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 Überlaufer 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 [SQL-Funktionen in AWS Clean Rooms](#).

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.

	KLEINE GANZZAHL	GANZZAHL	GROSSE ZAHL	DECIMAL	FLOAT4	FLOAT8
KLEIN	SMALLINT	INTEGER	BIGINT	DECIMAL	FLOAT8	FLOAT8

GANZZAHL	INTEGER	INTEGER	BIGINT	DECIMAL	FLOAT8	FLOAT8
GROSSE ZAHL	BIGINT	BIGINT	BIGINT	DECIMAL	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT4	FLOAT8
FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8

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 stellen p_1 und s_1 die Genauigkeit und die Dezimalstellen des ersten Operanden einer Berechnung und p_2 und s_2 die Genauigkeit und die Dezimalstellen 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 = $\max(s_1, s_2)$ Genauigkeit = $\max(p_1 - s_1, p_2 - s_2) + 1 + \text{scale}$
*	Skalieren = $s_1 + s_2$ Genauigkeit = $p_1 + p_2 + 1$
/	Skalieren = $\max(4, s_1 + p_2 - s_2 + 1)$ Genauigkeit = $p_1 - s_1 + s_2 + \text{scale}$

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)
```

Eine DEC1-Tabelle mit einer DECIMAL(7,2)-Spalte wird z. B. mit einer DEC2-Tabelle mit einer DECIMAL(15,3)-Spalte verbunden, um eine DEC3-Tabelle zu erstellen. Das Schema von DEC3 zeigt, dass die Spalte zu einer NUMERIC(15,3)-Spalte 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 die 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: $\max(38 + \text{scale} - \text{precision}), \min(4, 100)$

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:

- Ergebnisse von expliziten Umwandlungen in Überlauf Fehlern während der Laufzeit, wenn bestimmte Datenwerte nicht die Genauigkeit oder die Dezimalstellen aufweisen, die in der Umwandlungsfunktion angegeben sind. 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 wird in DECIMAL (5,0) umgewandelt
- INTEGER wird in DECIMAL (10,0) umgewandelt
- BIGINT 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).

Speicherung und Bereiche

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. 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)
CHAR oder CHARACTER	Länge der Zeichenfolge einschließlich der Leerzeichen am Ende (falls vorhanden)	4096 Bytes
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)

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.

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)
```

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 Datum-/Uhrzeittypen gehören DATE, TIME, TIMETZ, TIMESTAMP und TIMESTAMPTZ.

Themen

- [Speicherung und Bereiche](#)
- [DATUM](#)
- [TIME](#)
- [TIMETZ](#)
- [TIMESTAMP \(ZEITSTEMPEL\)](#)
- [TIMESTAMPTZ](#)
- [Beispiele mit Datum-/Uhrzeittypen](#)
- [Datums-, Zeit- und Zeitstempelliterale](#)
- [Intervallliterale](#)

Speicherung und Bereiche

Name	Speicher	Bereich	Behebung
DATE	4 Bytes	4713 v. Chr. bis 294276 n. Chr.	1 Tag
TIME	8 Bytes	00:00:00 bis 24:00:00	1 Mikrosekunde
TIMETZ	8 Bytes	00:00:00+1459 bis 00:00:00+1459	1 Mikrosekunde
TIMESTAMP	8 Bytes	4713 v. Chr. bis 294276 n. Chr.	1 Mikrosekunde
TIMESTAMP TZ	8 Bytes	4713 v. Chr. bis 294276 n. Chr.	1 Mikrosekunde

DATUM

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

TIME

Verwenden Sie den Datentyp TIME, um die Uhrzeit zu speichern.

TIME-Spalten speichern Werte mit einer Genauigkeit von maximal sechs Stellen für Sekundenbruchteile.

Standardmäßig entsprechen TIME-Werte sowohl in Benutzertabellen als auch in AWS Clean Rooms Systemtabellen der koordinierten Weltzeit (Coordinated Universal Time, UTC).

TIMETZ

Verwenden Sie den Datentyp TIMETZ, um die Uhrzeit mit einer Zeitzone zu speichern.

TIMETZ-Spalten speichern Werte mit einer Genauigkeit von maximal sechs Stellen für Sekundenbruchteile.

Standardmäßig sind TIMETZ-Werte sowohl in Benutzertabellen als auch in AWS Clean Rooms Systemtabellen UTC.

TIMESTAMP (ZEITSTEMPEL)

Verwenden sie den Datentyp TIMESTAMP, um vollständige Zeitstempel zu speichern, die das Datum und die Uhrzeit umfassen.

TIMESTAMP-Spalten speichern Werte mit einer Genauigkeit von maximal sechs Stellen für Sekundenbruchteile.

Wenn Sie ein Datum oder ein Datum mit einem unvollständigen Zeitstempelwert in eine TIMESTAMP-Spalte einfügen, wird der Wert implizit in einen vollständigen Zeitstempelwert konvertiert. Dieser vollständige Zeitstempelwert hat Standardwerte (00) für fehlende Stunden, Minuten und Sekunden. Zeitzonewerte in Eingabezeichenfolgen werden ignoriert.

Standardmäßig sind TIMESTAMP-Werte sowohl in Benutzertabellen als auch in AWS Clean Rooms Systemtabellen UTC.

TIMESTAMPTZ

Verwenden sie den Datentyp TIMESTAMPTZ, um vollständige Zeitstempel einzugeben, die das Datum, die Uhrzeit und eine Zeitzone umfassen. Wenn ein Eingabewert eine Zeitzone enthält, AWS Clean Rooms verwendet die Zeitzone, um den Wert in UTC umzurechnen, und speichert den UTC-Wert.

Führen Sie den folgenden Befehl aus, um eine Liste der unterstützten Zeitzonennamen anzuzeigen.

```
select my_timezone_names();
```

Führen Sie den folgenden Befehl aus, um eine Liste der unterstützten Zeitzoneabkürzungen anzuzeigen.

```
select my_timezone_abbrevs();
```

Aktuelle Informationen zu Zeitzonen finden Sie in der [IANA Time Zone Database](#).

In der folgenden Tabelle werden Beispiele zu Zeitzoneformaten aufgeführt.

Format	Beispiel
DD Mon HH:MI:SS YYYY TZ	17 Dec 07:37:16 1997 PST
mm/tt/jjjj hh:mi:ss.ss zz	12/17/1997 07:37:16.00 PST
mm/tt/jjjj hh:mi:ss.ss zz	12/17/1997 07:37:16.00 US/Pacific
yyyy-mm-dd hh:mi:ss+/-tz	1997-12-17 07:37:16-08
tt.mm.jjjj hh:mi:ss zz	17.12.1997 07:37:16.00 PST

TIMESTAMPTZ-Spalten speichern Werte mit einer Genauigkeit von maximal sechs Stellen für Sekundenbruchteile.

Wenn Sie ein Datum oder ein Datum mit einem unvollständigen Zeitstempelwert in eine TIMESTAMPTZ-Spalte einfügen, wird der Wert implizit in einen vollständigen Zeitstempelwert konvertiert. Dieser vollständige Zeitstempelwert hat Standardwerte (00) für fehlende Stunden, Minuten und Sekunden.

TIMESTAMPTZ-Werte in Benutzertabellen entsprechen der Zeitzone UTC.

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
```

Zeitstempelbeispiele

Wenn Sie ein Datum in eine TIMESTAMP- oder TIMESTAMPTZ-Spalte eingeben, wird für die Uhrzeit standardmäßig Mitternacht verwendet. Wenn Sie beispielsweise das Literal 20081231 eingeben, ist der gespeicherte Wert 2008-12-31 00:00:00.

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

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

Datums-, Zeit- und Zeitstempelliterale

Im Folgenden finden Sie Regeln für die Arbeit mit Datums-, Uhrzeit- und Zeitstempelliteralen, die von 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
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

Zeitstempel

Die folgende Tabelle zeigt Eingabezeitstempel, die gültige Beispiele für literale Zeitwerte sind, die Sie in Tabellen laden können. AWS Clean Rooms Alle gültigen Datumsliteralen können mit den folgenden Uhrzeitliteralen kombiniert werden.

Eingegebene Zeitstempel (konkatenierte Datums- und Uhrzeitliteralen)	Beschreibung (der Uhrzeitkomponente)
20080215 04:05:06.789	4.05 Uhr und 6,789 Sekunden
20080215 04:05:06	4.05 Uhr und 6 Sekunden
20080215 04:05	Genau 4.05 Uhr
20080215 040506	4.05 Uhr und 6 Sekunden
20080215 04:05 AM	Genau 4.05 Uhr, AM ist optional

Eingegebene Zeitstempel (konkatenierte Datums- und Uhrzeitlitterale)	Beschreibung (der Uhrzeitkomponente)
20080215 04:05 PM	Genau 16.05 Uhr, Stundenwert muss kleiner als 12 sein
20080215 16:05	Genau 16.05 Uhr
20080215	Mitternacht (durch Standardwert)

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
<code>now</code>	Wird zu der Startzeit der aktuellen Transaktion ausgewertet und gibt einen Zeitstempel mit auf Mikrosekunden genauer Uhrzeitkomponente zurück.
<code>today</code>	Wird zu dem entsprechenden Datum ausgewertet und gibt einen Zeitstempel mit Nullen für die Uhrzeitkomponente zurück.
<code>tomorrow</code>	Wird zu dem entsprechenden Datum ausgewertet und gibt einen Zeitstempel mit Nullen für die Uhrzeitkomponente zurück.
<code>yesterday</code>	Wird zu dem entsprechenden Datum ausgewertet und gibt einen Zeitstempel mit Nullen für die Uhrzeitkomponente zurück.

Die folgenden Beispiele zeigen, wie `now` und `today` mit der `DATEADD`-Funktion zusammenarbeiten.

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

```
date_add
```

```
-----
```

```
2009-11-17 00:00:00
```

```
(1 row)
```

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

```
date_add
```

```
-----
```

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

```
(1 row)
```

Intervallliterale

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

Mit Intervallliteralen können Zeiträume angegeben werden, beispielsweise `12 hours` oder `6 weeks`. Die Intervallliterale 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 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 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
```

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

einggegebenen 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 wird.

```
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	

Typ SUPER

Verwenden Sie den SUPER-Datentyp, um halbstrukturierte Daten oder Dokumente als Werte zu speichern.

Halbstrukturierte Daten entsprechen nicht der starren und tabellarischen Struktur des relationalen Datenmodells, das in SQL-Datenbanken verwendet wird. Der SUPER-Datentyp enthält Tags, die auf unterschiedliche Entitäten innerhalb der Daten verweisen. SUPER-Datentypen können komplexe Werte wie Arrays, verschachtelte Strukturen und andere komplexe Strukturen enthalten, die mit Serialisierungsformaten wie JSON verknüpft sind. Der SUPER-Datentyp ist ein Satz von schemalosen Array- und Strukturwerten, die alle anderen Skalartypen von umfassen. AWS Clean Rooms

Der Datentyp SUPER unterstützt bis zu 1 MB an Daten für ein einzelnes SUPER-Feld oder -Objekt.

Der SUPER-Datentyp hat folgende Eigenschaften:

- Ein skalarer Wert: AWS Clean Rooms

- Ein Nullwert
- Ein boolescher Wert
- Eine Zahl, wie z. B. `smallint`, `integer`, `bigint`, `decimal` oder `floating point` (z. B. `float4` oder `float8`)
- Ein Zeichenfolgenwert, z. B. `varchar` oder `char`
- Ein komplexer Wert:
 - Ein Array von Werten, einschließlich skalarer oder komplexer
 - Eine Struktur, auch bekannt als Tupel oder Objekt, die eine Zuordnung von Attributnamen und -werten (skalar oder komplex) darstellt

Jeder der beiden Typen komplexer Werte enthält eigene Skalare oder komplexe Werte ohne Einschränkungen für die Regelmäßigkeit.

Der SUPER-Datentyp unterstützt die Persistenz von halbstrukturierten Daten in einer schemalosen Form. Obwohl sich das hierarchische Datenmodell ändern kann, können die alten Datenversionen in derselben SUPER-Spalte nebeneinander existieren.

Verschachtelter Typ

AWS Clean Rooms unterstützt Abfragen mit Daten mit verschachtelten Datentypen, insbesondere den Spaltentypen `AWS Glue Struct`, `Array` und `Map`. Nur die benutzerdefinierte Analyseregeln 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.

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 `angegebenvarchar`.

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 unter [Navigation](#).

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](#)

Typ VARBYTE

Verwenden Sie eine VARBYTE-, VARBINARY- oder BINARY VARYING-Spalte, um einen Binärwert mit variabler Länge mit einem festen Grenzwert zu speichern.

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

Die maximale Anzahl von Bytes (n) kann zwischen 1 und 1 024 000 liegen. Der Standardwert ist 64 000.

Hier sind einige Beispiele, in denen es empfehlenswert sein kann, einen VARBYTE-Datentyp zu verwenden:

- Verknüpfen von Tabellen in VARBYTE-Spalten.
- Erstellen materialisierter Ansichten, die VARBYTE-Spalten enthalten. Die inkrementelle Aktualisierung materialisierter Ansichten, die VARBYTE-Spalten enthalten, wird unterstützt. Andere Aggregationsfunktionen als COUNT, MIN und MAX sowie GROUP BY bei VARBYTE-Spalten unterstützen jedoch keine inkrementelle Aktualisierung.

Um sicherzustellen, dass alle Byte druckbare Zeichen sind, AWS Clean Rooms verwendet das Hexadezimalformat zum Drucken von VARBYTE-Werten. Das folgende SQL konvertiert

beispielsweise die Hexadezimalzeichenfolge 6162 in einen Binärwert. Der zurückgegebene Wert ist zwar ein Binärwert, aber die Ergebnisse werden als Hexadezimalzahl 6162 ausgedruckt.

```
select from_hex('6162');

from_hex
-----
6162
```

AWS Clean Rooms unterstützt die Umwandlung zwischen VARBYTE und den folgenden Datentypen:

- CHAR
- VARCHAR
- SMALLINT
- INTEGER
- BIGINT

Mit der folgenden SQL-Anweisung wird eine VARCHAR-Zeichenfolge in VARBYTE umgewandelt. Der zurückgegebene Wert ist zwar ein Binärwert, aber die Ergebnisse werden als Hexadezimalzahl 616263 ausgedruckt.

```
select 'abc'::varbyte;

varbyte
-----
616263
```

Mit der folgenden SQL-Anweisung wird ein CHAR-Wert in einer Spalte in VARBYTE umgewandelt. In diesem Beispiel wird eine Tabelle mit einer CHAR(10)-Spalte (c) erstellt und es werden Zeichenwerte eingefügt, die kürzer als die Länge von 10 sind. Die resultierende Umwandlung füllt das Ergebnis mit Leerzeichen (hex'20') auf die definierte Spaltengröße auf. Der zurückgegebene Wert ist zwar ein Binärwert, aber die Ergebnisse werden als Hexadezimalzahl ausgedruckt.

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

      c
-----
61612020202020202020
```

```
61626320202020202020
```

Mit der folgenden SQL-Anweisung wird eine SMALLINT-Zeichenfolge in VARBYTE umgewandelt. Der zurückgegebene Wert ist zwar ein Binärwert, aber die Ergebnisse werden als Hexadezimalzahl 0005 ausgedruckt, die zwei Bytes oder vier Hexadezimalzeichen entspricht.

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

varbyte
-----
0005
```

Mit der folgenden SQL-Anweisung wird INTEGER in VARBYTE umgewandelt. Der zurückgegebene Wert ist zwar ein Binärwert, aber die Ergebnisse werden als Hexadezimalzahl 00000005 ausgedruckt, die vier Bytes oder acht Hexadezimalzeichen entspricht.

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

varbyte
-----
00000005
```

Mit der folgenden SQL-Anweisung wird BIGINT in VARBYTE umgewandelt. Der zurückgegebene Wert ist zwar ein Binärwert, aber die Ergebnisse werden als Hexadezimalzahl 0000000000000005 ausgedruckt, die acht Bytes oder 16 Hexadezimalzeichen entspricht.

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

varbyte
-----
0000000000000005
```

Einschränkungen bei der Verwendung des VARBYTE-Datentyps mit AWS Clean Rooms

Im Folgenden finden Sie Einschränkungen bei der Verwendung des VARBYTE-Datentyps mit: AWS Clean Rooms

- AWS Clean Rooms unterstützt den VARBYTE-Datentyp nur für Parquet- und ORC-Dateien.

- AWS Clean Rooms Der Abfrageeditor unterstützt den VARBYTE-Datentyp noch nicht vollständig. Verwenden Sie deshalb einen anderen SQL-Client, wenn Sie mit VARBYTE-Ausdrücken arbeiten.

Wenn Sie dennoch den Abfrage-Editor verwenden möchten, gibt es einen Workaround: Sofern die Größe der Daten unter 64 KB liegt und der Inhalt gültiges UTF-8 ist, können Sie die VARBYTE-Werte beispielsweise in VARCHAR umwandeln:

```
select to_varbyte('6162', 'hex')::varchar;
```

- Sie können VARBYTE-Datentypen nicht mit benutzerdefinierten Python- oder Lambda-Funktionen (UDFs) verwenden.
- Sie können aus einer VARBYTE-Spalte keine HLLSKETCH-Spalte erstellen oder für eine VARBYTE-Spalte APPROXIMATE COUNT DISTINCT verwenden.

Kompatibilität von Typen und Umwandlung zwischen Typen

In der folgenden Diskussion wird beschrieben, wie Typkonvertierungsregeln und Datentypkompatibilität in AWS Clean Rooms funktionieren.

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.
- Wenn Sie DATE oder TIMESTAMP in TIMESTAMPTZ konvertieren oder TIME in TIMETZ konvertieren, wird die Zeitzone auf die Zeitzone der aktuellen Sitzung festgelegt. Standardmäßig ist als Zeitzone für Sitzungen UTC festgelegt.
- Analog dazu wird auch TIMESTAMPTZ auf der Grundlage der Zeitzone der aktuellen Sitzung in DATE, TIME oder TIMESTAMP konvertiert. Standardmäßig ist als Zeitzone für Sitzungen UTC festgelegt. Die Zeitzoneneinformationen werden nach der Konvertierung wieder entfernt.
- Zeichenfolgen, die einen Zeitstempel mit angegebener Zeitzone darstellen, werden unter Verwendung der Zeitzone der aktuellen Sitzung in TIMESTAMPTZ konvertiert. Diese ist standardmäßig UTC. Zeichenfolgen, die eine Uhrzeit mit angegebener Zeitzone darstellen, werden unter Verwendung der Zeitzone der aktuellen Sitzung in TIMETZ konvertiert. Diese ist standardmäßig UTC.

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

Von Typ	Zu Typ
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
CHAR	VARCHAR
DATUM	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT
	CHAR
	DOUBLE PRECISION (FLOAT8)
	GANZZAHL (INT)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
DOUBLE PRECISION (FLOAT8)	BIGINT

Von Typ	Zu Typ
	CHAR
	DECIMAL (NUMERIC)
	GANZZAHL (INT)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
GANZZAHL (INT)	BIGINT
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
REAL (FLOAT4)	BIGINT
	CHAR
	DECIMAL (NUMERIC)
	GANZZAHL (INT)
	SMALLINT
	VARCHAR

Von Typ	Zu Typ
SMALLINT	BIGINT
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	GANZZAHL (INT)
	REAL (FLOAT4)
	VARCHAR
TIMESTAMP	CHAR
	DATUM
	VARCHAR
	TIMESTAMPTZ
	TIME
TIMESTAMPTZ	CHAR
	DATUM
	VARCHAR
	TIMESTAMP
	TIMETZ
TIME	VARCHAR
	TIMETZ

Von Typ	Zu Typ
TIMETZ	VARCHAR
	TIME

 Note

Bei impliziten Typumwandlungen zwischen TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ oder Zeichenfolgen wird die Zeitzone der aktuellen Sitzung verwendet. Der Datentyp VARBYTE kann nicht implizit in einen anderen Datentyp umgewandelt werden. Weitere Informationen finden Sie unter [CAST-Funktion](#).

SQL-Befehle in AWS Clean Rooms

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

Themen

- [SELECT](#)

SELECT

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

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

Themen

- [SELECT list](#)
- [WITH-Klausel](#)
- [FROM-Klausel](#)
- [WHERE-Klausel](#)
- [GROUP BY-Klausel](#)
- [HAVING-Klausel](#)
- [Satzoperatoren](#)
- [ORDER BY-Klausel](#)
- [Beispiele für Unterabfragen](#)
- [Korrelierte Unterabfragen](#)

SELECT list

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

Syntax

```
SELECT  
[ TOP number ]
```

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

Parameter

TOP *number* (*Zahl*)

TOP verwendet eine positive Ganzzahl als Argument, das die Anzahl der Zeilen definiert, die an den Client zurückgegeben werden. Das Verhalten mit der -TOPKlausel entspricht dem Verhalten mit der -LIMITKlausel. Die Anzahl der zurückgegebenen Zeilen ist fest, aber der Zeilensatz ist nicht fest. Um einen konsistenten Satz von Zeilen zurückzugeben, verwenden Sie TOP oder LIMIT in Verbindung mit einer -ORDERBYKlausel.

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.

Beispielsweise:

```
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. Beispielsweise:

```
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.

Nutzungshinweise

TOP ist eine SQL-Erweiterung. TOP bietet eine Alternative zum -LIMIT-Verhalten. Sie können TOP und nicht LIMIT in derselben Abfrage verwenden.

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 )
```

Parameter

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 AWS Clean Rooms unterstützt. Siehe [SELECT](#).

Nutzungshinweise

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

- SELECT, WITH, UNION, INTERSECT und EXCEPT

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	venue name	venue city	venue state	venue seats
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 venue name, venue city, sum(pricepaid) as venue name_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
group by venue name, venue city),

top_venues as
(select venue name
from venue_sales
where venue name_sales > 800000)

select venue name, venue city, venue state,
```

```

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 venuevenue in(select venuevenue from top_venues)
group by venuevenue, venuecity, venuestate
order by venuevenue;

```

venuevenue	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](#)
- [Parameter](#)
- [Nutzungshinweise](#)
- [JOIN-Beispiele](#)

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
```

Parameter

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.

Beispielsweise:

```
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 lautet, schlägt die Abfrage fehl `SELECT "tbl"."col" FROM "tbl" AS "t"`, da der Tabellename 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-Beispiele

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. Weitere Informationen zur Syntax einer JOIN-Klausel finden Sie unter [Parameter](#).

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 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 die LISTIDs 2 und 3 nicht zu Verkäufen führten.

```
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		
3		
4	76.00	11.40
5	525.00	78.75

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 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 die 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

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 die LISTIDs 2 und 3 nicht zu Verkäufen führten.

```
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. In den Ergebnissen sind nur Zeilen enthalten, die zu keinen Verkäufen führen (LISTIDs 2 und 3).

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

listid	price	comm
2	NULL	NULL
3	NULL	NULL

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

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 gleicht LISTID-Spaltenwerte in der SALES- und der LISTING-Tabelle für LISTIDs 1, 2, 3, 4 und 5 in beiden Tabellen ab. 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

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
```

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

Die folgende Abfrage ist ein interner Join zweier 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
```

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)

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 im SELECT-Ausdruck eine Aggregatfunktion vorhanden ist, muss sich jede Spalte im SELECT-Ausdruck, die sich nicht in einer Aggregatfunktion befindet, in der GROUP BY-Klausel befinden.

Weitere Informationen finden Sie unter [SQL-Funktionen in AWS Clean Rooms](#).

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 Rooms unterstützt Aggregationserweiterungen, um die Arbeit mehrerer GROUP BY-Operationen in einer einzigen Anweisung auszuführen.

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

Gruppirt 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 Satzoperatoren UNION, INTERSECT und EXCEPT werden verwendet, um die Ergebnisse von zwei getrennten Abfrageausdrücken zu vergleichen und zusammenzuführen. 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 mit den Satzoperatoren UNION, UNION ALL, INTERSECT und EXCEPT zusammengeführt wurden.

Themen

- [Syntax](#)
- [Parameter](#)
- [Reihenfolge der Evaluierung für Satzoperatoren](#)
- [Nutzungshinweise](#)
- [Beispiel für UNION-Abfragen](#)
- [Beispiel für die UNION ALL-Abfrage](#)
- [Beispiel für INTERSECT-Abfragen](#)
- [Beispiel für die EXCEPT-Abfrage](#)

Syntax

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

Parameter

query

Ein Abfrageausdruck, der in Form seiner Auswahlliste einem zweiten Abfrageausdruck entspricht, der dem Operator UNION, INTERSECT oder EXCEPT folgt. Die beiden Ausdrücke müssen die gleiche Zahl von Ausgabespalten mit kompatiblen Datentypen enthalten. Andernfalls können die beiden Ergebnissätze nicht verglichen und zusammengeführt werden. Satzoperationen lassen die

implizite Umwandlung zwischen unterschiedlichen Kategorien von Datentypen nicht zu. 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
```

UNION

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

INTERSECT

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

EXCEPT | MINUS

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. MINUS und EXCEPT sind exakte Synonyme.

ALL

Das Schlüsselwort ALL behält alle duplizierten Zeilen, die von UNION erstellt werden. Wenn das Schlüsselwort ALL nicht verwendet wird, besteht das Standardverhalten darin, diese Duplikate zu verwerfen. INTERSECT ALL, EXCEPT ALL und MINUS ALL werden nicht unterstützt.

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 (Spaltenalias) 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	GX071KOC	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. Die Abfrage gibt für eine spezifische Reihe von Ereignis-IDs 0 oder mehr Zeilen für jeden Verkauf zurück, der mit den einzelnen Ereignissen verknüpft ist, und 0 oder 1 Zeile für jede Auflistung dieses Ereignisses. Die Ereignis-IDs sind für die einzelnen Zeilen in den Tabellen LISTING und EVENT eindeutig. Es gibt jedoch möglicherweise mehrere Verkäufe für dieselbe Kombination von Ereignis- und Auflistungs-IDs in der Tabelle SALES.

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. Die beiden anderen Auflistungen, 501 und 502, generierten keine Verkäufe. Daher stammt die einzige Zeile, die die Abfrage für diese Auflistungs-IDs generiert, aus der Tabelle LISTING (SALESROW = „Nein“).

```
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. Die Abfrage gibt für eine spezifische Reihe von Ereignis-IDs 0 oder mehr Zeilen für jeden Verkauf zurück, der mit den einzelnen Ereignissen verknüpft ist, und 0 oder 1 Zeile für jede Auflistung dieses Ereignisses. Die Ereignis-IDs sind für die einzelnen Zeilen in den Tabellen LISTING und EVENT eindeutig. Es gibt jedoch möglicherweise mehrere Verkäufe für dieselbe Kombination von Ereignis- und Auflistungs-IDs in der Tabelle SALES.

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. Die beiden anderen Auflistungen, 501 und 502, generierten keine Verkäufe. Daher stammt die einzige Zeile, die die Abfrage für diese Auflistungs-IDs generiert, aus der Tabelle LISTING (SALESROW = „Nein“).

```
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 Tabelle CATEGORY 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](#)
- [Parameter](#)
- [Nutzungshinweise](#)
- [Beispiele mit ORDER BY](#)

Syntax

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

Parameter

expression

Ausdruck, der die Sortierreihenfolge des Abfrageergebnisses definiert. Sie 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.
- Wenn ORDER BY in einem AWS Clean Roomsparallelen System wie keine eindeutige Reihenfolge erzeugt, ist die Reihenfolge der Zeilen nicht deterministisch. Das heißt, wenn der ORDER BY-Ausdruck doppelte Werte erzeugt, kann die Rückgabereihenfolge dieser Zeilen von anderen Systemen oder von einer Ausführung von AWS Clean Rooms zur nächsten variieren.
- AWS Clean Rooms unterstützt keine Zeichenfolgeliterale 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 | venue name | venue city | venue state | venue seats
-----+-----+-----+-----+-----
(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 [JOIN-Beispiele](#).

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 Arten von korrelierten Unterabfragen folgen Mustern, die nicht dekoriert werden kann und nicht unterstützt. 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 übersprungene 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.

- Nullsensitive Korrelationsreferenzen auf eine - AWS Clean Rooms Systemtabelle. Beispielsweise:

```
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.) Beispielsweise:

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

SQL-Funktionen in AWS Clean Rooms

AWS Clean Rooms unterstützt die folgenden SQL-Funktionen:

Themen

- [Aggregationsfunktionen](#)
- [Array-Funktionen](#)
- [Bedingte Ausdrücke](#)
- [Funktionen für die Datentypformatierung](#)
- [Datums- und Zeitfunktionen](#)
- [Hash-Funktionen](#)
- [JSON-Funktionen](#)
- [Mathematische Funktionen](#)
- [Zeichenfolgenfunktionen](#)
- [Funktionen für SUPER-Typinformationen](#)
- [VARBYTE-Funktionen](#)
- [Fensterfunktionen](#)

Aggregationsfunktionen

AWS Clean Rooms unterstützt die folgenden Aggregatfunktionen:

Themen

- [Funktion ANY_VALUE](#)
- [Die Funktion APPROXIMATE PERCENTILE_DISC](#)
- [AVG Funktion](#)
- [Die Funktion BOOL_AND](#)
- [Die Funktion BOOL_OR](#)
- [COUNT Funktionen und COUNT DISTINCT](#)
- [Die Funktion COUNT](#)
- [Die Funktion LISTAGG](#)

- [Die Funktion MAX](#)
- [Die Funktion MEDIAN](#)
- [Die Funktion MIN](#)
- [Die Funktion PERCENTILE_CONT](#)
- [Die Funktionen STDDEV_SAMP und STDDEV_POP](#)
- [SUM - und -SUM DISTINCTFunktionen](#)
- [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 ( [ DISTINCT | ALL ] expression )
```

Argumente

DISTINCT | ALL

Geben Sie entweder DISTINCT oder ALL an, um einen beliebigen Wert aus den Eingabeausdruckswerten zurückzugeben. Das Argument DISTINCT hat keine Auswirkung und wird ignoriert.

Ausdruck

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

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL

- DOUBLE PRECISION
- BOOLEAN
- CHAR
- VARCHAR
- DATUM
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

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 Instance eines beliebigen zurückdateid, bei dem eventname istEagles.

```
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
```

Das folgende Beispiel gibt eine Instance eines beliebigen zurückdateid, bei dem oder eventname istEaglesCold 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
```

Die Funktion APPROXIMATE PERCENTILE_DISC

APPROXIMATE PERCENTILE_DISC ist eine Funktion für die inverse Verteilung, die ein diskretes Verteilungsmodell annimmt. Sie empfängt einen Perzentilwert und eine Sortierspezifikation und gibt ein Element aus dem angegebenen Satz zurück. Die Annäherung ermöglicht eine sehr viel schnellere Ausführung der Funktion bei einer niedrigen relativen Fehlerquote von ungefähr 0,5 Prozent.

APPROXIMATE PERCENTILE_DISC verwendet für einen bestimmten Perzentilwert einen zusammenfassenden Quantil-Algorithmus, um das diskrete Perzentil des Ausdrucks in der ORDER BY-Klausel anzunähern. APPROXIMATE PERCENTILE_DISC gibt den Wert mit dem kleinsten kumulativen Verteilungswert (in Bezug auf dieselbe Sortierspezifikation) zurück, der größer als oder gleich Perzentil ist.

APPROXIMATE PERCENTILE_DISC ist eine reine Datenverarbeitungsknoten-Funktion. Die Funktion gibt einen Fehler zurück, wenn die Abfrage nicht auf eine benutzerdefinierte Tabelle oder AWS Clean Rooms Systemtabelle verweist.

Syntax

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

Argumente

percentile

Numerische Konstante zwischen 0 und 1. Null-Werte werden bei der Berechnung ignoriert.

WITHIN GROUP (ORDER BY expr)

Klausel, die numerische oder Datum-/Zeitwerte angibt, um das Perzentil zu sortieren und zu verarbeiten.

Rückgabewert

Derselbe Datentyp wie der ORDER BY-Ausdruck in der WITHIN GROUP-Klausel.

Nutzungshinweise

Wenn die Anweisung APPROXIMATE PERCENTILE_DISC eine GROUP BY-Klausel enthält, ist der Ergebnissatz begrenzt. Das Limit ist vom Knotentyp und der Anzahl der Knoten abhängig. Wenn das Limit überschritten wird, schlägt die Funktion fehl und gibt den folgenden Fehler zurück.

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

Wenn Sie mehr Gruppen auswerten müssen, als das Limit zulässt, sollten Sie die Verwendung von in Betracht ziehen [Die Funktion PERCENTILE_CONT](#).

Beispiele

Im folgenden Beispiel werden die Anzahl der Verkäufe, der Gesamtumsatz und der fünfzigste Perzentilwert für die 10 Topdaten zurückgegeben.

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

caldate	count	sum	percentile_disc
2008-01-07	658	2081400.00	2020.00
2008-01-02	614	2064840.00	2178.00
2008-07-22	593	1994256.00	2214.00
2008-01-26	595	1993188.00	2272.00

2008-02-24		655		1975345.00		2070.00
2008-02-04		616		1972491.00		1995.00
2008-02-14		628		1971759.00		2184.00
2008-09-01		600		1944976.00		2100.00
2008-07-29		597		1944488.00		2106.00
2008-07-23		592		1943265.00		1974.00

AVG Funktion

Die AVG Funktion gibt den Durchschnitt (arithmetischer Mittelwert) der Eingabeausdruckswerte zurück. Die AVG Funktion funktioniert mit numerischen Werten und ignoriert NULL-Werte.

Syntax

```
AVG (column)
```

Argumente

column

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

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- DOUBLE

Datentypen

Die von der AVG Funktion unterstützten Argumenttypen sind SMALLINT, INTEGER, BIGINTDECIMAL, und DOUBLE.

Die von der AVG Funktion unterstützten Rückgabetyper sind:

- BIGINT für jedes Argument vom Typ Ganzzahl
- DOUBLE für ein Gleitkommaargument
- Gibt denselben Datentyp wie Ausdruck für jeden anderen Argumenttyp zurück

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

Beispiel

Ermitteln Sie die durchschnittliche verkaufte Menge pro Transaktion aus der SALES Tabelle.

```
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 Rückgabebetyp 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 Rückgabebetyp 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
```

COUNT Funktionen und COUNT DISTINCT

Die `COUNT` Funktion zählt die durch den Ausdruck definierten Zeilen. Die `COUNT DISTINCT` Funktion berechnet die Anzahl der unterschiedlichen Nicht-`NULL`-Werte in einer Spalte oder einem Ausdruck. Es beseitigt alle duplizierten Werte aus dem angegebenen Ausdruck, bevor die Zählung durchgeführt wird.

Syntax

```
COUNT (column)
```

```
COUNT (DISTINCT column)
```

Argumente

column

Die Zielspalte, für die die Funktion ausgeführt wird.

Datentypen

Die -COUNTFunktion und die -COUNT DISTINCTFunktion unterstützen alle Argumentdatentypen.

Die COUNT DISTINCT Funktion gibt zurückBIGINT.

Beispiele

Zählen Sie alle Benutzer aus dem Bundesstaat Puerto.

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

Zählen Sie alle eindeutigen Veranstaltungs-IDs aus 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.
- APPROXIMATE COUNT DISTINCT ermittelt die ungefähre Anzahl der unterschiedlichen Nicht-NULL-Werte in einer Spalte oder einem Ausdruck.

Syntax

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

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

```
APPROXIMATE COUNT ( DISTINCT 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.

APPROXIMATE

Bei Verwendung mit APPROXIMATE verwendet eine Funktion COUNT DISTINCT einen HyperLogLog Algorithmus, um die Anzahl der unterschiedlichen Nicht-NULL-Werte in einer Spalte oder einem Ausdruck anzunähern. Abfragen mit dem Schlüsselwort APPROXIMATE werden sehr viel schneller bei einer niedrigen relativen Fehlerquote von etwa 2 % ausgeführt. Annäherungen sollten bei Abfragen verwendet werden, die eine große Zahl unterschiedlicher Werte zurückgeben, beispielsweise mehrere Millionen oder mehr pro Abfrage (bzw. pro Gruppe, wenn es eine GROUP BY-Klausel gibt). Bei kleineren Sätzen unterschiedlicher Werte, d. h. mehreren tausend, werden Annäherungen möglicherweise langsamer ausgeführt als präzise Zählungen. APPROXIMATE kann nur mit COUNT DISTINCT verwendet werden.

Rückgabebetyp

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
```

Zählung aller eindeutigen Veranstaltungs-IDs aus der Tabelle EVENT:

```
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
...

```

In den folgenden Beispielen werden die Rückgabewerte und Ausführungszeichen für COUNT und APPROXIMATE COUNT verglichen.

```

select count(distinct pricepaid) from sales;

```

```

count
-----
 4528

```

Time: 48.048 ms

```

select approximate count(distinct pricepaid) from sales;

```

```

count
-----
 4553

```

Time: 21.728 ms

Die Funktion LISTAGG

Die Aggregationsfunktion „LISTAGG“ ordnet die Zeilen der Gruppe in einer Abfrage nach dem ORDER BY-Ausdruck an. Anschließend werden die Werte zu einer einzigen Zeichenfolge verkettet.

LISTAGG ist eine reine Datenverarbeitungsknoten-Funktion. Die Funktion gibt einen Fehler zurück, wenn die Abfrage nicht auf eine benutzerdefinierte Tabelle oder AWS Clean Rooms Systemtabelle verweist.

Syntax

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

Argumente

DISTINCT

(Optional) Eine Klausel, die duplizierte Werte in dem angegebenen Ausdruck beseitigt, bevor die Verkettung vorgenommen wird. Leerzeichen am Ende werden ignoriert, sodass beispielsweise die Zeichenfolgen ' a ' und ' a ' als duplizierte Werte behandelt werden würden. „LISTAGG“ verwendet den ersten registrierten Wert. Weitere Informationen finden Sie unter [Die Bedeutung von Leerzeichen am Ende](#).

aggregate_expression

Ein gültiger Ausdruck (beispielsweise ein Spaltenname), der die Werte bereitstellt, die aggregiert werden sollen. NULL-Werte und leere Zeichenfolgen werden ignoriert.

delimiter

(Optional) Die Zeichenfolgenkonstante, die die verketteten Werte trennt. Der Standardwert ist „NULL“.

AWS Clean Rooms unterstützt jede Anzahl von Leerzeichen am Anfang oder am Ende um ein optionales Komma oder einen Doppelpunkt sowie eine leere Zeichenfolge oder eine beliebige Anzahl von Leerzeichen.

Beispiele für gültige Werte sind:

" , "

" : "

" "

WITHIN GROUP (ORDER BY *order_list*)

(Optional) Eine Klausel, die die Sortierreihenfolge der aggregierten Werte angibt.

Rückgabewert

VARCHAR(MAX). Wenn der Ergebnissatz größer als die maximal zulässige Größe von VARCHAR ist (64.000 – 1 oder 65535), gibt LISTAGG den folgenden Fehler zurück:

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

Nutzungshinweise

Wenn eine Anweisung mehrere LISTAGG-Funktionen enthält, die WITHIN GROUP-Klauseln verwenden, muss jede WITHIN GROUP-Klausel dieselben ORDER BY-Werte verwenden.

Beispielweise wird die folgende Anweisung einen Fehler zurückgeben.

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

Die folgenden Anweisungen werden erfolgreich ausgeführt werden.

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

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

Beispiele

Im folgenden Beispiel werden Verkäufer-IDs aggregiert, geordnet nach Verkäufer-ID.

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

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

Im folgenden Beispiel wird mit „DISTINCT“ eine Liste von einzigartigen Verkäufer-IDs zurückgegeben.

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

```
listagg
```

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

Im folgenden Beispiel werden Verkäufer-IDs aggregiert, geordnet nach Datum.

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

```
listagg
```

```
-----
31141242333
```

Im folgenden Beispiel wird eine durch Pipe-Zeichen getrennte Liste von Verkaufsterminen für Käufer B zurückgegeben.

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

```
listagg
```

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

Im folgenden Beispiel wird eine durch Komma getrennte Liste von Verkaufs-IDs für die einzelnen Käufer-IDs zurückgegeben.

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

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

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 einer der folgenden Datentypen:

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

- DATUM
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

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
-----
2500.000000000
```

(1 row)

Die Funktion MEDIAN

Berechnet den Medianwert für den Wertebereich. NULL-Werte im Bereich werden ignoriert.

MEDIAN ist eine Funktion für die inverse Verteilung, die ein kontinuierliches Verteilungsmodell annimmt.

MEDIAN ist ein Spezialfall von [PERCENTILE_CONT\(.5\)](#).

MEDIAN ist eine reine Datenverarbeitungsknoten-Funktion. Die Funktion gibt einen Fehler zurück, wenn die Abfrage nicht auf eine benutzerdefinierte Tabelle oder AWS Clean Rooms Systemtabelle verweist.

Syntax

```
MEDIAN ( median_expression )
```

Argumente

median_expression

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

Datentypen

Der Rückgabetyt wird durch den Datentyp von median_expression festgelegt. Die folgende Tabelle zeigt den Rückgabetyt für jeden median_expression-Datentyp an.

Input type	Rückgabetyt
NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATUM	DATUM
TIMESTAMP	TIMESTAMP

Input type	Rückgabebetyp
TIMESTAMPTZ	TIMESTAMPTZ

Nutzungshinweise

Wenn das Argument `median_expression` den Datentyp `DECIMAL` hat und mit der maximal zulässigen Präzision von 38 Stellen definiert ist, gibt `MEDIAN` möglicherweise ein falsches Ergebnis oder einen Fehler zurück. Wenn der Rückgabewert der Funktion `MEDIAN` 38 Stellen überschreitet, wird das Ergebnis entsprechend abgekürzt. Dies führt zu einem Genauigkeitsverlust. Wenn während der Interpolierung ein Zwischenergebnis die maximal zulässige Genauigkeit überschreitet, erfolgt ein numerischer Überlauf und die Funktion gibt einen Fehler zurück. Um diese Bedingungen zu vermeiden, werden die Verwendung eines Datentyps mit einer niedrigeren Genauigkeit oder die Umwandlung des Arguments `median_expression` in ein Argument mit niedrigerer Genauigkeit empfohlen.

Wenn eine Anweisung mehrere Aufrufe von sortierbasierten Aggregationsfunktionen enthält (`LISTAGG`, `PERCENTILE_CONT` oder `MEDIAN`), müssen alle dieselben `ORDER BY`-Werte verwenden. Beachten Sie, dass `MEDIAN` implizit eine Reihenfolge nach dem Wert des Ausdrucks anwendet.

Die folgende Anweisung gibt beispielsweise einen Fehler zurück.

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

An error occurred when executing the SQL command:

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

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

Die folgende Anweisung wird erfolgreich ausgeführt.

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

```
median (salesid)
from sales group by salesid, pricepaid;
```

Beispiele

Das folgende Beispiel zeigt, dass MEDIAN dieselben Ergebnisse wie PERCENTILE_CONT(0.5) produziert.

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

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

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 einer der folgenden Datentypen:

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

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
-----
20.000000000
(1 row)
```

Die Funktion PERCENTILE_CONT

PERCENTILE_CONT ist eine Funktion für die inverse Verteilung, die ein kontinuierliches Verteilungsmodell annimmt. Sie empfängt einen Perzentilwert und eine Sortierspezifikation und gibt einen interpolierten Wert zurück, der in Bezug auf die Sortierspezifikation in den angegebenen Perzentilwert fällt.

PERCENTILE_CONT berechnet eine lineare Interpolierung zwischen Werten, nachdem diese der Reihenfolge entsprechend angeordnet wurden. Mithilfe des Perzentilwerts (P) und der Anzahl der Nicht-Null-Zeilen (N) in der Aggregationsgruppe berechnet die Funktion die Anzahl der Zeilen, nachdem die Zeilen entsprechend der Sortierspezifikation angeordnet wurden. Die Anzahl von Zeilen (RN) wird mit der Formel $RN = (1 + (P * (N - 1)))$ berechnet. Das Endergebnis der Aggregationsfunktion wird durch lineare Interpolierung zwischen den Werten aus Zeilen zwischen $CRN = \text{CEILING}(RN)$ und $FRN = \text{FLOOR}(RN)$ berechnet.

Das Ergebnis wird wie folgt aussehen.

Wenn ($CRN = FRN = RN$), ist das Ergebnis (value of expression from row at RN)

Andernfalls sieht das Ergebnis wie folgt aus:

$(CRN - RN) * (\text{value of expression for row at } FRN) + (RN - FRN) * (\text{value of expression for row at } CRN).$

PERCENTILE_CONT ist eine reine Datenverarbeitungsknoten-Funktion. Die Funktion gibt einen Fehler zurück, wenn die Abfrage nicht auf eine benutzerdefinierte Tabelle oder AWS Clean Rooms Systemtabelle verweist.

Syntax

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

Argumente

percentile

Numerische Konstante zwischen 0 und 1. Null-Werte werden bei der Berechnung ignoriert.

WITHIN GROUP (ORDER BY *expr*)

Gibt numerische oder Datum-/Zeitwerte an, nach denen das Perzentil sortiert und berechnet werden soll.

Rückgabewert

Der Rückgabewert wird durch den Datentyp des ORDER BY-Ausdrucks in der WITHIN GROUP-Klausel festgelegt. Die folgende Tabelle zeigt den Rückgabewert für jeden ORDER BY-Datentyp an.

Input type	Rückgabewert
SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATUM	DATUM
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

Nutzungshinweise

Wenn das Argument ORDER BY den Datentyp DECIMAL hat und mit der maximal zulässigen Präzision von 38 Stellen definiert ist, gibt PERCENTILE_CONT möglicherweise ein falsches Ergebnis oder einen Fehler zurück. Wenn der Rückgabewert der Funktion PERCENTILE_CONT 38 Stellen

überschreitet, wird das Ergebnis entsprechend abgekürzt. Dies führt zu einem Genauigkeitsverlust. Wenn während der Interpolierung ein Zwischenergebnis die maximal zulässige Genauigkeit überschreitet, erfolgt ein numerischer Überlauf und die Funktion gibt einen Fehler zurück. Um diese Bedingungen zu vermeiden, werden die Verwendung eines Datentyps mit einer niedrigeren Genauigkeit oder die Umwandlung des Ausdrucks ORDER BY in einen Ausdruck mit niedrigerer Genauigkeit empfohlen.

Wenn eine Anweisung mehrere Aufrufe von sortierbasierten Aggregationsfunktionen enthält (LISTAGG, PERCENTILE_CONT oder MEDIAN), müssen alle dieselben ORDER BY-Werte verwenden. Beachten Sie, dass MEDIAN implizit eine Reihenfolge nach dem Wert des Ausdrucks anwendet.

Die folgende Anweisung gibt beispielsweise einen Fehler zurück.

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

An error occurred when executing the SQL command:

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

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

Die folgende Anweisung wird erfolgreich ausgeführt.

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

Beispiele

Das folgende Beispiel zeigt, dass MEDIAN dieselben Ergebnisse wie PERCENTILE_CONT(0.5) produziert.

```
select top 10 distinct sellerid, qtysold,
```

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

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

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 Ganzzahl-, Dezimal- oder Gleitkommatyp 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)
```

SUM - und -SUM DISTINCTFunktionen

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

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

Syntax

```
SUM (column)
```

```
SUM (DISTINCT column )
```

Argumente

column

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

- SMALLINT

- INTEGER
- BIGINT
- DECIMAL
- DOUBLE

Datentypen

Die von der SUM Funktion unterstützten Argumenttypen sind SMALLINT, INTEGER, BIGINTDECIMAL, und DOUBLE.

Die SUM Funktion unterstützt die folgenden Rückgabetypen:

- BIGINT für die INTEGER Argumente BIGINTSMALLINT, und
- DOUBLE für Gleitkomma-Argumente
- Gibt denselben Datentyp wie Ausdruck für jeden anderen Argumenttyp zurück

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

Beispiele

Ermitteln Sie die Summe aller gezahlten Provisionen aus 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 Gleitkommatyp 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](#)
- [array_concat-Funktion](#)
- [array_flatten-Funktion](#)
- [get_array_length-Funktion](#)
- [split_to_array-Funktion](#)
- [subarray-Funktion](#)

array-Funktion

Erstellt ein Array des SUPER-Datentyps.

Syntax

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

Argument

expr1, expr2

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

Rückgabebetyp

Die array-Funktion gibt den Datentyp SUPER zurück.

Beispiel

Das folgende Beispiel zeigt eine Reihe von numerischen Werten und eine Reihe verschiedener 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)
```

array_concat-Funktion

Die array_concat-Funktion verkettet zwei Arrays, um ein Array zu erstellen, das alle Elemente im ersten Array enthält, gefolgt von allen Elementen im zweiten Array. Beide Argumente müssen gültige Arrays sein.

Syntax

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

Argumente

super_expr1

Der Wert, der das erste der beiden zu verkettenden Arrays angibt.

super_expr2

Der Wert, der das zweite der beiden zu verkettenden Arrays angibt.

Rückgabebetyp

Die `array_concat`-Funktion gibt einen SUPER-Datenwert zurück.

Beispiel

Das folgende Beispiel zeigt die Verkettung von zwei Arrays desselben Typs und die Verkettung von zwei Arrays unterschiedlichen Typs.

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

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

array_flatten-Funktion

Führt mehrere Arrays in einem einzelnen Array vom SUPER-Typ zusammen.

Syntax

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

Argumente

`super_expr1`,`super_expr2`

Ein gültiger SUPER-Ausdruck der Array-Form.

Rückgabebetyp

Die `array_flatten`-Funktion gibt einen SUPER-Datenwert zurück.

Beispiel

Das folgende Beispiel zeigt eine `array_flatten`-Funktion.

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

get_array_length-Funktion

Gibt die Länge des angegebenen Arrays zurück. Die Funktion `GET_ARRAY_LENGTH` gibt die Länge eines an ein bestimmtes Objekt übergebenen SUPER-Arrays oder eines Array-Pfads an.

Syntax

```
get_array_length( super_expr )
```

Argumente

`super_expr`

Ein gültiger SUPER-Ausdruck der Array-Form.

Rückgabebetyp

Die `get_array_length`-Funktion gibt einen BIGINT zurück.

Beispiel

Das folgende Beispiel zeigt eine `get_array_length`-Funktion.

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

split_to_array-Funktion

Verwendet ein Trennzeichen als optionalen Parameter. Wenn kein Trennzeichen vorhanden ist, ist der Standardwert ein Komma.

Syntax

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

Argumente

string

Die Eingabezeichenfolge, die geteilt werden soll.

delimiter

Ein optionaler Wert nach dem die Eingabezeichenfolge getrennt wird. Standardmäßig wird ein Komma verwendet.

Rückgabebetyp

Die `split_to_array`-Funktion gibt einen SUPER-Datenwert zurück.

Beispiel

Das folgende Beispiel zeigt eine `split_to_array`-Funktion.

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

subarray-Funktion

Manipuliert Arrays, um eine Teilmenge der Eingabe-Arrays zurückzugeben.

Syntax

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

Argumente

`super_expr`

Ein gültiger SUPER-Ausdruck in Array-Form.

`start_position`

Die Position innerhalb des Arrays, an der die Extrahierung gestartet werden soll, beginnend mit der Indexposition 0. Eine negative Position zählt vom Ende des Arrays rückwärts.

`length`

Die Anzahl der Element, die extrahiert werden soll (die Länge der Unterzeichenfolge).

Rückgabebetyp

Die subarray-Funktion gibt einen SUPER-Datenwert zurück.

Beispiel

Das folgende Beispiel zeigt eine subarray-Funktion.

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

Bedingte Ausdrücke

AWS Clean Rooms unterstützt die folgenden bedingten Ausdrücke:

Themen

- [Der bedingte Ausdruck CASE](#)
- [COALESCE Ausdruck](#)
- [Funktionen GREATEST und LEAST](#)
- [NVL- und COALESCE-Funktionen](#)

- [Funktion NVL2](#)
- [NULLIF-Funktion](#)

Der bedingte Ausdruck CASE

Der CASE-Ausdruck ist ein bedingter Ausdruck, vergleichbar den IF-/THEN-/ELSE-Anweisungen anderer Sprachen. 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 Ausgabetyt 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
...

```

COALESCE Ausdruck

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

Wenden Sie den COALESCE Ausdruck auf zwei Spalten an.

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

Der Standardspaltenname für einen NVL-Ausdruck ist COALESCE. Die folgende Abfrage gibt dieselben Ergebnisse zurück.

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

Funktionen GREATEST und LEAST

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)

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.

Rückgabebetyp

AWS Clean Rooms bestimmt den Datentyp des zurückgegebenen Werts basierend auf den Eingabeausdrücken. 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"
```

Funktion NVL2

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.

Rückgabetyt

Der NVL2-Rückgabetyt wird wie folgt festgelegt:

- 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

Für `NVL2` hat die Rückgabe entweder den Wert des Parameters `not_null_return_value` oder `null_return_value`, je nachdem, was von der Funktion ausgewählt wurde, hat jedoch den Datentyp `not_null_return_value`.

Wenn beispielsweise `column1` `NULL` ist, geben die folgenden Abfragen denselben Wert zurück. Der Datentyp des `DECODE`-Rückgabewerts ist jedoch `INTEGER` und der Datentyp des `NVL2`-Rückgabewerts ist `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.com
August Ashley   consectetuer.euismod@example.com
Karyn Austin    ipsum.primis.in@example.com
Lucas Ayers     at@example.com
```

NULLIF-Funktion

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 Rückgabebetyp ist mit dem Typ des ersten Ausdrucks identisch. Der Standardspaltenname des NULLIF-Ergebnisses ist der Spaltenname des ersten Ausdrucks.

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)

Funktionen für die Datentypformatierung

Mithilfe einer Funktion zur Formatierung von Datentypen können Sie Werte von einem Datentyp in einen anderen konvertieren. Für jede 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 unterstützt mehrere Formatierungsfunktionen für Datentypen.

Themen

- [CAST-Funktion](#)
- [CONVERT-Funktion](#)
- [TO_CHAR](#)
- [TO_DATE-Funktion](#)
- [TO_NUMBER](#)
- [Datum-/Uhrzeit-Formatzeichenfolgen](#)
- [Numerische Formatzeichenfolgen](#)

- [TeradataFormatierung von Zeichen im Stil von -Zeichen für numerische Daten](#)

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 Quelltable nicht ändert. Dieser wird nur im Kontext der Abfrage geändert.

Die CAST-Funktion ist [the section called "CONVERT"](#) insofern sehr ähnlich, als beide Funktionen einen Datentyp in einen anderen konvertieren. Die beiden Funktionen werden jedoch unterschiedlich aufgerufen.

Bestimmte Datentypen erfordern eine explizite Konvertierung in andere Datentypen unter Verwendung der Funktionen CAST oder CONVERT. Andere Datentypen können implizit als Teil eines anderen Befehls konvertiert werden, ohne CAST oder CONVERT zu verwenden. 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 )  
expression :: 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 [Datentypen](#), mit Ausnahme der VARYTE-, BINARY- und BINARY-Datentypen.

Rückgabebetyp

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 salesid, pricepaid::decimal(38,2)*100000000000000000000
as value from sales where salesid<10 order by salesid;
```

salesid	value
1	72800000000000000000000000000000.00
2	76000000000000000000000000000000.00
3	35000000000000000000000000000000.00
4	17500000000000000000000000000000.00
5	15400000000000000000000000000000.00
6	39400000000000000000000000000000.00
7	78800000000000000000000000000000.00
8	19700000000000000000000000000000.00
9	59100000000000000000000000000000.00

(9 rows)

CONVERT-Funktion

Wie die [CAST-Funktion](#) konvertiert die CONVERT-Funktion einen Datentyp in einen anderen kompatiblen Datentyp. Sie können beispielsweise eine Zeichenfolge in ein Datum oder einen numerischen Typ in eine Zeichenfolge konvertieren. CONVERT führt eine Laufzeitkonvertierung durch, was bedeutet, dass die Konvertierung den Datentyp eines Werts in einer Quelltable nicht ändert. Dieser wird nur im Kontext der Abfrage geändert.

Bestimmte Datentypen erfordern eine explizite Konvertierung in andere Datentypen unter Verwendung der CONVERT-Funktion. Andere Datentypen können implizit als Teil eines anderen Befehls konvertiert werden, ohne CAST oder CONVERT zu verwenden. Siehe [Kompatibilität von Typen und Umwandlung zwischen Typen](#).

Syntax

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

Argumente

Typ

Einer der unterstützten [Datentypen](#), mit Ausnahme der VARYTE-, BINARY- und BINARY-Datentypen.

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.

Rückgabebetyp

CONVERT 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 CONVERT(decimal(2,1), 123.456);
```

oder eine INTEGER-Konvertierung, die einen Overflow verursacht:

```
SELECT CONVERT(smallint, 12345678);
```

Beispiele

Die folgende Abfrage verwendet die CONVERT-Funktion, um eine Spalte mit Dezimalzahlen in Ganzzahlen zu konvertieren.

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

In diesem Beispiel wird eine Ganzzahl in eine Zeichenfolge konvertiert.

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

In diesem Beispiel werden das aktuelle Datum und die aktuelle Uhrzeit in einen variablen Zeichendatentyp konvertiert:

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



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

In diesem Beispiel wird die Saletime-Spalte in eine reine Zeitspalte konvertiert, wobei die Datumsangaben aus jeder Zeile entfernt werden.

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

Im folgenden Beispiel werden variable Zeichendaten in ein Datetime-Objekt konvertiert.

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

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](#).

Rückgabebetyp

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, '199999D99') 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(string, format)
```

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

Argumente

string

Eine Zeichenfolge, die konvertiert werden soll.

format

Ein Zeichenfolgeliteral, das das Format der Zeichenfolge in der Eingabezeichenfolge in Bezug auf die Datumsabschnitte definiert. Eine Liste der gültigen Formate für Tag, Monat und Jahr finden Sie unter [Datum-/Uhrzeit-Formatzeichenfolgen](#).

is_strict

Ein optionaler boolescher Wert, der angibt, ob ein Fehler zurückgegeben wird, wenn ein Eingabedatumswert außerhalb des zulässigen Bereichs liegt. Wenn is_strict auf TRUE gesetzt wird, wird ein Fehler zurückgegeben, wenn ein Wert außerhalb des zulässigen Bereichs liegt. Wenn is_strict auf FALSE gesetzt wird, was die Standardeinstellung ist, sind Überlaufwerte zulässig.

Rückgabebetyp

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 Mon YYYY');
```

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

Die folgende SQL-Anweisung konvertiert die Zeichenfolge 20010631 in ein Datum.

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

Das Ergebnis ist der 1. Juli 2001, da der Juni nur 30 Tage hat.

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

Die folgende SQL-Anweisung konvertiert die Zeichenfolge 20010631 in ein Datum:

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

Das Ergebnis ist ein Fehler, da der Juni nur 30 Tage hat.

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

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](#).

Rückgabebetyp

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
```


Datum-/Uhrzeit-Formatzeichenfolgen

Die folgenden Zeichenketten im Datetime-Format gelten für Funktionen wie TO_CHAR. Diese Zeichenfolgen können Datum-/Uhrzeittrennzeichen (wie -, / oder :) sowie die folgenden „Datumsteile“ oder „Zeitteile“ enthalten.


Beispiele für das Formatieren von Datumsangaben als Zeichenfolgen finden Sie unter [TO_CHAR](#).

Datumsteil oder Zeiteil	Bedeutung
BC oder B.C., AD oder A.D., b.c. oder bc, ad oder a.d.	Anzeigen für Zeitalter in Groß- und Kleinbuchstaben
CC	Jahrhundertzahl mit zwei Ziffern
YYYY, YYY, YY, Y	Jahreszahl mit 4, 3, 2 oder 1 Ziffer
Y,YYY	Jahreszahl mit 4 Ziffern und Komma
IYYY, IYY, IY, I	Jahreszahl der internationalen Organisation für Normung (International Organization for Standardization, ISO) mit 4, 3, 2 oder 1 Ziffer
Q	Quartalszahl (1 bis 4)
MONAT, Monat, monat	Name des Monats (Großbuchstaben, Groß- und Kleinbuchstaben, mit leeren Stellen auf 9 Zeichen aufgefüllt)

Datumsteil oder Zeitteil	Bedeutung
MON, Mon, mon	Abgekürzter Name des Monats (Großbuchstaben, Groß- und Kleinbuchstaben, mit leeren Stellen auf 3 Zeichen aufgefüllt)
MM	Monatszahl (01 bis 12)
RM, rm	Monat in römischen Ziffern (I–XII, wobei I Januar ist, Groß- oder Kleinbuchstaben)
W	Woche des Monats (1–5; die erste Woche beginnt am ersten Tag des Monats)
WW	Woche des Jahres (1–53; die erste Woche beginnt am ersten Tag des Jahres)
IW	Woche des Jahres nach ISO (der erste Donnerstag des neuen Jahres liegt in Woche 1)
DAY, Day, day	Name des Tages (Großbuchstaben, Groß- und Kleinbuchstaben, mit leeren Stellen auf 9 Zeichen aufgefüllt)
DY, Dy, dy	Abgekürzter Name des Tages (Großbuchstaben, Groß- und Kleinbuchstaben, mit leeren Stellen auf 3 Zeichen aufgefüllt)
DDD	Tag des Jahres (001–366)
IDDD	Tag der Woche des Jahres nach ISO 8601 (001–371; Tag 1 des Jahres ist Montag der ersten ISO-Woche)
DD	Tag des Monats als Zahl (01–31)

Datumsteil oder Zeitteil	Bedeutung
D	<p>Wochentag (1–7; wobei Sonntag 1 ist)</p> <div style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p> Note</p> <p>Der Datumsteil D verhält sich anders als der Datumsteil Wochentag (DOW), der für die Datum-/Uhrzeitfunktionen DATE_PART and EXTRACT verwendet wird. DOW basiert auf den Ganzzahlen 0–6, wobei die 0 für den Sonntag steht. Weitere Informationen finden Sie unter Datumsteile für Datums- oder Zeitstempelfunktionen.</p> </div>
ID	Tag der Woche nach ISO 8601, Montag (1) bis Sonntag (7)
J	Julianischer Tag (Tage seit dem 1. Januar 4712 BC)
HH24	Stunde (24-Stunden-Uhr, 00–23)
HH oder HH12	Stunde (12-Stunden-Uhr, 01–12)
MI	Minuten (00–59)
SS	Sekunden (00–59)
MS	Millisekunden (,000)
US	Mikrosekunden (,000000)
AM oder PM, A.M. oder P.M., a.m. oder p.m., am oder pm	Meridiananzeigen für Zeitalter in Groß- und Kleinbuchstaben (für die 12-Stunden-Uhr)
TZ, tz	Zeitzoneabkürzung in Groß- und Kleinbuchstaben, nur für TIMESTAMPTZ gültig

Datumsteil oder Zeitteil	Bedeutung
OF	Unterschied zu UTC; nur für TIMESTAMPTZ gültig

 Note

Sie müssen datetime-Separatoren (wie etwa '-', '/' oder ':') mit einzelnen Anführungszeichen umgeben, die in der vorherigen Tabelle aufgeführten "dateparts" und "timeparts" müssen allerdings in doppelten Anführungszeichen stehen.

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 unter [TO_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).
FM	Füllmodus. Unterdrückt ausfüllende Leerzeichen und Nullen.

Format	Beschreibung
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.

TeradataFormatierung von Zeichen im Stil von -Zeichen für numerische Daten

In diesem Thema erfahren Sie, wie die Funktionen `TEXT_TO_INT_ALT` und `TEXT_TO_NUMERIC_ALT` die Zeichen in der Zeichenfolge für den Eingabeausdruck interpretieren. In der folgenden Tabelle finden Sie auch eine Liste der Zeichen, die Sie in der Formatphrase angeben können. Darüber hinaus finden Sie eine Beschreibung der Unterschiede zwischen der Formatierung im Teradata-Stil und der AWS Clean Rooms Formatierungsoption.

Format	Beschreibung
G	<p>Wir nicht als Gruppentrennzeichen für die expression-Eingabezeichenfolge unterstützt. Sie können dieses Zeichen nicht in der format-Phrase angeben.</p>
D	<p>Dezimaltrennzeichen. Sie können dieses Zeichen in der format-Phrase angeben. Dieses Zeichen entspricht dem . (Punkt).</p> <p>Das Radix-Symbol darf nicht in einer Formatphrase vorkommen, die eines der folgenden Zeichen enthält:</p> <ul style="list-style-type: none"> • . (Punkt) • S (großgeschriebenes S) • V (großgeschriebenes V)
/ , : %	<p>Einfügungszeichen / (Schrägstrich); , (Komma); : (Doppelpunkt) und % (Prozentzeichen).</p> <p>Sie können diese Zeichen nicht in der format-Phrase angeben.</p> <p>AWS Clean Rooms ignoriert diese Zeichen in der Zeichenfolge für den Eingabeausdruck.</p>
.	<p>Punkt als Grundzeichen, d. h. als Dezimalzeichen.</p> <p>Dieses Zeichen kann nicht in einer format-Phrase vorkommen, die eines der folgenden Zeichen enthält:</p> <ul style="list-style-type: none"> • D (großgeschriebenes D) • S (großgeschriebenes S)

Format	Beschreibung
	<ul style="list-style-type: none"> • V (großgeschriebenes V)
B	<p>Sie können kein Leerraumzeichen (B) in der format-Phrase angeben. In der expression-Zeichenfolge werden Leerzeichen am Anfang und Ende ignoriert und Leerzeichen zwischen Ziffern sind nicht zulässig.</p>
+ -	<p>Sie können kein Plus- oder Minuszeichen (+ oder -) in der format-Phrase angeben. Plus- und Minuszeichen werden jedoch implizit als Teil des numerischen Wertes geparkt, wenn sie in der expression-Eingabezeichenfolge auftauchen.</p>
V	<p>Indikator für die Position des Dezimaltrennzeichens.</p> <p>Dieses Zeichen kann nicht in einer format-Phrase vorkommen, die eines der folgenden Zeichen enthält:</p> <ul style="list-style-type: none"> • D (großgeschriebenes D) • . (Punkt)
Z	<p>Dezimalziffer mit Nullunterdrückung. AWS Clean Rooms schneidet führende Nullen ab. Das Z-Zeichen darf keiner 9 folgen. Das Z-Zeichen muss links vom Dezimaltrennzeichen stehen, wenn die Nachkommastelle eine 9 enthält.</p>
9	<p>Dezimalstelle.</p>

Format	Beschreibung
CHAR(n)	<p>Für dieses Format können Sie für Folgendes angeben:</p> <ul style="list-style-type: none">• CHAR besteht aus Z oder 9 Zeichen. AWS Clean Rooms unterstützt kein + (Plus) oder - (Minus) im CHAR-Wert.• n ist eine Ganzzahlkonstante, I oder F. Bei I ist dies die Anzahl der Zeichen, die erforderlich sind, um den Ganzzahlteil numerischer oder ganzzahliger Daten anzuzeigen. Bei F ist dies die Anzahl der Zeichen, die erforderlich sind, um die Nachkommastellen numerischer Daten anzuzeigen.
-	<p>Bindestrich-Zeichen (-).</p> <p>Sie dieses Zeichen nicht in der format-Phrase angeben.</p> <p>AWS Clean Rooms ignoriert dieses Zeichen in der Zeichenfolge für den Eingabeausdruck.</p>

Format	Beschreibung
S	<p>Dezimalzahl mit Vorzeichen in Zonen. Das S-Zeichen muss auf die letzte Dezimalstelle in der format-Phrase folgen. Das letzte Zeichen der expression-Eingabezeichenfolge und die entsprechende numerische Konvertierung finden Sie unter Datenformatierungszeichen für numerische Datenformatierung im Teradata-Stil mit Zoneneinteilung mit Vorzeichen .</p> <p>Das S-Zeichen kann nicht in einer format-Phrase vorkommen, die eines der folgenden Zeichen enthält:</p> <ul style="list-style-type: none">• + (Pluszeichen)• . (Punkt)• D (großgeschriebenes D)• Z (großgeschriebenes Z)• F (großgeschriebenes F)• E (großgeschriebenes E)
E	<p>Exponentialnotation. Die expression-Eingabezeichenfolge kann ein Exponentenzeichen enthalten. E kann nicht als Exponentenzeichen in einer format-Phrase angegeben werden.</p>
FN9	<p>Wird nicht unterstützt in AWS Clean Rooms.</p>
FNE	<p>Wird nicht unterstützt in AWS Clean Rooms.</p>

Format	Beschreibung
\$, USD, US-Dollar	<p>Dollarzeichen (\$), ISO-Währungssymbol (USD) und der Währungsname US-Dollar.</p> <p>Beim ISO-Währungssymbol USD und beim Währungsnamen US-Dollar wird zwischen Groß- und Kleinschreibung unterschieden. AWS Clean Rooms unterstützt nur die USD-Währung. Die expression-Eingabezeichenfolge kann Leerzeichen zwischen dem Dollar-Währungssymbol und dem numerischen Wert umfassen, zum Beispiel „\$ 123E2“ oder „123E2 \$“.</p>
L	<p>Währungssymbol. Dieses Währungssymbol kann in der format-Phrase nur einmal vorhanden sein. Es ist nicht möglich, mehrere Währungssymbole anzugeben.</p>
C	<p>ISO-Währungssymbol. Dieses Währungssymbol kann in der format-Phrase nur einmal vorhanden sein. Es ist nicht möglich, mehrere Währungssymbole anzugeben.</p>
N	<p>Vollständiger Währungsname. Dieses Währungssymbol kann in der format-Phrase nur einmal vorhanden sein. Es ist nicht möglich, mehrere Währungssymbole anzugeben.</p>
O	<p>Doppeltes Währungssymbol. Sie können dieses Zeichen nicht in der format-Phrase angeben.</p>
U	<p>Doppeltes ISO-Währungssymbol. Sie können dieses Zeichen nicht in der format-Phrase angeben.</p>

Format	Beschreibung
A	Vollständiger doppelter Währungsname. Sie können dieses Zeichen nicht in der format-Phrase angeben.

Datenformatierungszeichen für numerische Datenformatierung im Teradata-Stil mit Zoneneinteilung mit Vorzeichen

Sie können die folgenden Zeichen in der format-Phrase der TEXT_TO_INT_ALT- und TEXT_TO_NUMERIC_ALT-Funktionen für einen Signed-Zoned-Decimal-Wert verwenden.

Letztes Zeichen der Eingabezeichenfolge	Numerische Konvertierung
{ oder 0	n ... 0
A oder 1	n ... 1
B oder 2	n ... 2
C oder 3	n ... 3
D oder 4	n ... 4
E oder 5	n ... 5
F oder 6	n ... 6
G oder 7	n ... 7
H oder 8	n ... 8
I oder 9	n ... 9
}	-n ... 0
J	-n ... 1
K	-n ... 2

Letztes Zeichen der Eingabezeichenfolge	Numerische Konvertierung
L	-n ... 3
M	-n ... 4
N	-n ... 5
O	-n ... 6
P	-n ... 7
Q	-n ... 8
R	-n ... 9

Datums- und Zeitfunktionen

AWS Clean Rooms unterstützt die folgenden Datums- und Uhrzeitfunktionen:

Themen

- [Zusammenfassung der Datums- und Zeitfunktionen](#)
- [Datums- und Zeitfunktionen in Transaktionen](#)
- [Operator + \(Verkettung\)](#)
- [Funktion ADD_MONTHS](#)
- [Funktion CONVERT_TIMEZONE](#)
- [Funktion CURRENT_DATE](#)
- [Funktion DATEADD](#)
- [Funktion DATEDIFF](#)
- [Funktion DATE_PART](#)
- [Funktion DATE_TRUNC](#)
- [Funktion EXTRACT](#)
- [GETDATE Funktion](#)
- [Funktion SYSDATE](#)

- [Funktion TIMEOFDAY](#)
- [Funktion TO_TIMESTAMP](#)
- [Datumsteile für Datums- oder Zeitstempelfunktionen](#)


Zusammenfassung der Datums- und Zeitfunktionen

Die folgende Tabelle enthält eine Zusammenfassung der Datums- und Uhrzeitfunktionen, die in verwendet werden AWS Clean Rooms.

Funktion	Syntax	Rückgabewert
<p>Operator + (Verkettung)</p> <p>Verkettet ein Datum mit einer Uhrzeit auf beiden Seiten des Pluszeichens (+) und gibt einen TIMESTAMP oder TIMESTAMPTZ zurück.</p>	date + time	TIMESTAMP oder TIMESTAMPTZ
<p>ADD_MONTHS</p> <p>Fügt die angegebene Anzahl von Monaten zu einem Datums- oder Zeitstempel hinzu.</p>	ADD_MONTHS ({date timestamp}, integer)	TIMESTAMP
<p>Funktion CURRENT_DATE</p> <p>Gibt ein Datum in der Zeitzone der aktuellen Sitzung (standardmäßig UTC) für den Beginn der aktuellen Transaktion aus.</p>	CURRENT_DATE	DATE
<p>DATEADD</p> <p>Erhöht ein Datum oder eine Uhrzeit um ein bestimmtes Intervall.</p>	DATEADD (datepart, interval, {date time timetz timestamp})	TIMESTAMP oder TIME oder TIMETZ
<p>DATEDIFF</p> <p>Gibt die Differenz zwischen zwei Datumsangaben oder Uhrzeiten für einen bestimmten</p>	DATEDIFF (datepart, {date time timetz timestamp}, {date time timetz timestamp})	BIGINT

Funktion	Syntax	Rückgabewert
Datumsteil, etwa einen Tag oder einen Monat, aus.		
<p>DATE_PART</p> <p>Extrahiert einen Datumsteilwert aus einem Datum oder einer Uhrzeit.</p>	DATE_PART (datepart, {date timestamp})	DOUBLE
<p>DATE_TRUNC</p> <p>Verkürzt einen Zeitstempel auf der Grundlage eines Datumsteils.</p>	DATE_TRUNC ('datepart', timestamp)	TIMESTAMP
<p>EXTRACT</p> <p>Extrahiert einen Datums- oder Uhrzeitteil von einem Timestamp-, Timestamptz-, Time- oder Timetz-Wert.</p>	EXTRACT (datepart AUS source)	INTEGER or DOUBLE
<p>GETDATE Funktion</p> <p>Gibt das aktuelle Datum und die aktuelle Uhrzeit in der Zeitzone der aktuellen Sitzung (standardmäßig UTC) aus. Die Klammern sind erforderlich.</p>	GETDATE()	TIMESTAMP
<p>SYSDATE</p> <p>Gibt das Datum und die Uhrzeit nach UTC für den Beginn der aktuellen Transaktion aus.</p>	SYSDATE	TIMESTAMP

Funktion	Syntax	Rückgabewert
<p>TIMEOFDAY</p> <p>Gibt den aktuellen Wochentag, das aktuelle Datum und die aktuelle Uhrzeit in der Zeitzone der aktuellen Sitzung (standardmäßig UTC) als Zeichenfolgenwert aus.</p>	TIMEOFDAY()	VARCHAR
<p>TO_TIMESTAMP</p> <p>Gibt einen Zeitstempel mit Zeitzone für das angegebene Zeitstempelformat mit Zeitzone aus.</p>	TO_TIMESTAMP ('timestamp', 'format')	TIMESTAMP TZ

 Note

Sprungsekunden werden bei Berechnungen der verstrichenen Zeit nicht berücksichtigt.

Datums- und Zeitfunktionen in Transaktionen

Wenn Sie die folgenden Funktionen mit einem Transaktionsblock (BEGINN ... END) ausführen, gibt die Funktion das Startdatum bzw. die Startzeit der aktuellen Transaktion aus, nicht den Beginn der aktuellen Anweisung.

- SYSDATE
- TIMESTAMP
- CURRENT_DATE

Die folgenden Funktionen geben immer das Startdatum oder die Startzeit der aktuellen Anweisung aus, selbst wenn sie sich innerhalb eines Transaktionsblocks befinden.

- GETDATE

- TIMEOFDAY

Operator + (Verkettung)

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

Syntax

```
numeric + string
```

```
date + time
```

```
date + timetz
```

Die Reihenfolge der Argumente kann umgekehrt werden.

Argumente

numerische Literale

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

Zeichenfolgeliterale

Zeichenfolgen, Zeichenfolgen oder Zeichenkonstanten

date

Eine DATE Spalte oder ein Ausdruck, die/der implizit zu einem konvertiert wirdDATE.

variieren

Eine TIME Spalte oder ein Ausdruck, die/der implizit zu einem konvertiert wirdTIME.

timetz

Eine TIMETZ Spalte oder ein Ausdruck, die/der implizit zu einem konvertiert wirdTIMETZ.

Beispiel

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

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

Funktion ADD_MONTHS

ADD_MONTHS fügt die angegebene Zahl von Monaten zu einem Datums- oder Zeitstempelwert bzw. -ausdruck hinzu. Die Funktion [DATEADD](#) 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.

Rückgabebetyp

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.

Rückgabebetyp

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
```

Das folgende Beispiel konvertiert eine Zeitstempel-LISTTIME-Spalte von der Standardzeitzone UTC zur Zeitzone US/Pacific time. 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 aus: JJJJ-MM-TT.

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
```

Rückgabebetyp

DATUM

Beispiel

Im folgenden Beispiel wird das aktuelle Datum (in der AWS-Region, in der die Funktion ausgeführt wird) zurückgegeben.

```
select current_date;
```

```
date
-----
2008-10-01
```

Funktion DATEADD

Erhöht einen DATE-, TIME-, TIMETZ- oder TIMESTAMP-Wert um ein bestimmtes Intervall.

Syntax

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

Argumente

datepart

Der Datumsteil (z. B. Jahr, Monat, Tag oder Stunde), für den die Funktion gilt. Weitere Informationen finden Sie unter [Datumsteile für Datums- oder Zeitstempelfunktionen](#).

Intervall

Eine Ganzzahl, die das Intervall angibt (z. B. eine Anzahl von Tagen), das dem Zielausdruck hinzugefügt werden soll. Bei einer negativen Ganzzahl wird das Intervall subtrahiert.

date|time|timetz|timestamp

Eine DATE-, TIME-, TIMETZ- oder TIMESTAMP-Spalte bzw. ein entsprechender Ausdruck, die/der implizit zu einem DATE, TIME, TIMETZ oder TIMESTAMP konvertiert wird. Der DATE-, TIME-, TIMETZ- oder TIMESTAMP-Ausdruck muss den angegebenen Datumsteil enthalten.

Rückgabetyt

TIMESTAMP oder TIME oder TIMEZ abhängig vom Eingabedatentyp.

Beispiele mit einer DATE-Spalte

Im folgenden Beispiel werden 30 Tage zu jedem Datum hinzugefügt, das in der DATE-Tabelle vorhanden ist.

```
select dateadd(day,30,caldate) as novplus30
```

```

from date
where month='NOV'
order by dateid;

novplus30
-----
2008-12-01 00:00:00
2008-12-02 00:00:00
2008-12-03 00:00:00
...
(30 rows)

```

Im folgenden Beispiel werden 18 Monate zu einem Literal-Datumswert hinzugefügt.

```

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

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

```

Der Standard-Spaltenname für eine DATEADD-Funktion ist DATE_ADD. Der Standard-Zeitstempel für einen Datumswert ist 00:00:00.

Im folgenden Beispiel werden 30 Minuten zu einem Datumswert hinzugefügt, der keinen Zeitstempel angibt.

```

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

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

```

Sie können Datumsteile ausschreiben oder abkürzen. In diesem Fall steht das m für Minuten, nicht für Monate.

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 jedem TIME_VAL in der TIME_TEST-Tabelle 5 Minuten hinzugefügt.

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

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

Im folgenden Beispiel werden 8 Stunden zu einem Literal-Zeitwert hinzugefügt.

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

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

Das folgende Beispiel wird angezeigt, wenn eine Zeit über 24:00:00 oder unter 00:00:00 liegt.

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

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

Beispiele mit einer TIMETZ-Spalte

Die Ausgabewerte in diesen Beispielen sind in der Standardzeitzone UTC angegeben.

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 jedem TIMETZ_VAL in der TIMETZ_TEST-Tabelle 5 Minuten hinzugefügt.

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

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

Im folgenden Beispiel werden 2 Stunden zu einem Literal-Timetz-Wert hinzugefügt.

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

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

Beispiele mit einer TIMESTAMP-Spalte

Die Ausgabewerte in diesen Beispielen sind in der Standardzeitzone UTC angegeben.

Die folgende Beispieltabelle TIMESTAMP_TEST enthält eine Spalte TIMESTAMP_VAL (Typ TIMESTAMP) mit drei eingefügten Werten.

```
SELECT timestamp_val FROM timestamp_test;
```

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

Im folgenden Beispiel werden nur den `TIMESTAMP_VAL`-Werten in `TIMESTAMP_TEST` aus der Zeit vor dem Jahr 2000 20 Jahre hinzugefügt.

```
SELECT dateadd(year,20,timestamp_val)
FROM timestamp_test
WHERE timestamp_val < to_timestamp('2000-01-01 00:00:00', 'YYYY-MM-DD HH:MI:SS');

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

Im folgenden Beispiel werden einem literalen Zeitstempelwert, der ohne Sekundenanzeige geschrieben wurde, 5 Sekunden hinzugefügt.

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

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

Nutzungshinweise

Die Funktionen `DATEADD(month, ...)` und `ADD_MONTHS` behandeln Daten am Ende von Monaten in unterschiedlicher Weise:

- `ADD_MONTHS`: Wenn das Datum, das Sie hinzufügen, der letzte Tag des Monats ist, ist das Ergebnis immer der letzte Tag des Ergebnismonats, unabhängig von der Länge des Monats. Zum Beispiel: 30. April + 1 Monat = 31. Mai.

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

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

- `DATEADD`: Wenn das Datum, zu dem Sie hinzufügen, weniger Tage enthält als der Ergebnismonat, entspricht das Ergebnis dem Tag des Ergebnismonats, nicht dem letzten Tag des Monats. Zum Beispiel: 30. April + 1 Monat = 30. Mai.

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

```
date_add
-----
2008-05-30 00:00:00
(1 row)
```

Die Funktion DATEADD behandelt das Schaltjahrdatum 02-29 anders als DATEADD(month, 12, ...) oder DATEADD(year, 1, ...).

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

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

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

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

Funktion DATEDIFF

DATEDIFF gibt die Differenz zwischen den Datumsteilen zweier Datums- oder Uhrzeitausdrücke aus.

Syntax

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

Argumente

datepart

Der spezifische Teil des Datums- oder Zeitwerts (Jahr, Monat oder Tag, Stunde, Minute, Sekunde, Millisekunde oder Mikrosekunde) auf den die Funktion angewendet wird. Weitere Informationen finden Sie unter [Datumsteile für Datums- oder Zeitstempelfunktionen](#).

Insbesondere bestimmt DATEDIFF die Anzahl von Datumsteilgrenzen, die zwischen zwei Ausdrücken überschritten werden. Nehmen wir an, dass Sie den Jahresdifferenz zwischen zwei Daten berechnen möchten, 12-31-2008 und 01-01-2009. In diesem Fall gibt die Funktion

1 Jahr zurück, obwohl die Daten nur einen Tag auseinanderliegen. Wenn Sie die Differenz in Stunden zwischen zwei Zeitstempeln, `01-01-2009 8:30:00` und `01-01-2009 10:00:00` ermitteln, ist das Ergebnis 2 Stunden. Wenn Sie die Differenz in Stunden zwischen zwei Zeitstempeln, `8:30:00` und `10:00:00` ermitteln, ist das Ergebnis 2 Stunden.

`date|time|timetz|timestamp`

Eine DATE-, TIME-, TIMETZ- oder TIMESTAMP-Spalte bzw. ein entsprechender Ausdruck, die/der implizit zu einem DATE, TIME, TIMETZ oder TIMESTAMP konvertiert wird. Beide Ausdrücke müssen den angegebenen Datums- oder Zeiteil enthalten. Wenn das zweite Datum bzw. die zweite Uhrzeit größer als das/die erste ist, ist das Ergebnis positiv. Wenn das zweite Datum bzw. die zweite Uhrzeit vor dem/der ersten liegt, ist das Ergebnis negativ.

Rückgabebetyp

BIGINT

Beispiele mit einer DATE-Spalte

Im folgenden Beispiel wird die Differenz als Anzahl von Wochen zwischen zwei Literal-Datumswerten berechnet.

```
select datediff(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 datediff(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 datediff(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 datediff(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 Standard-Spaltenname für die DATEDIFF-Funktion ist DATE_DIFF.

```
select datediff(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,
datediff(day, listtime, saletime) as wait
from sales, listing where sales.listid = listing.listid
and sales.listid between 1000 and 1005
order by wait desc, priceperticket desc;
```

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

Dieses Beispiel berechnet die durchschnittliche Zahl von Stunden, für die Verkäufer auf alle Ticketverkäufe warteten.

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

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

Argumente

datepart

Ein Bezeichnerliteral oder eine Zeichenfolge des spezifischen Teils des Datumswertes (z. B. Jahr, Monat oder Tag), für den die Funktion gilt. Weitere Informationen finden Sie unter [Datumsteile für Datums- oder Zeitstempelfunktionen](#).

{*date*|*timestamp*}

Eine Datums- oder Zeitstempelspalte bzw. ein entsprechender Ausdruck, die/der implizit zu einem Datum oder Zeitstempel konvertiert wird. Die Spalte bzw. der Ausdruck unter *date* oder *timestamp* muss den in *datepart* angegebenen Datumsteil enthalten.

Rückgabebetyp

DOUBLE

Beispiele

Der Standard-Spaltenname für die DATE_PART-Funktion ist pgdate_part.

Im folgenden Beispiel wird der Minutenwert aus einem Zeitstempelliteral ermittelt.

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

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

Im folgenden Beispiel wird der Wochenwert aus einem Zeitstempelliteral ermittelt. Die Berechnung der Wochenzahl erfolgt gemäß ISO-Standard 8601. Weitere Informationen finden Sie unter [ISO 8601](#) in Wikipedia.

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

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

Im folgenden Beispiel wird der Tag des Monats aus einem Zeitstempelliteral ermittelt.

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

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

Im folgenden Beispiel wird der Wochentag aus einem Zeitstempelliteral ermittelt. Die Berechnung der Wochenzahl erfolgt gemäß ISO-Standard 8601. Weitere Informationen finden Sie unter [ISO 8601](#) in Wikipedia.

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

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

Das folgende Beispiel ermittelt das Jahrhundert aus einem Zeitstempelliteral. Die Berechnung des Jahrhunderts erfolgt gemäß ISO-Standard 8601. Weitere Informationen finden Sie unter [ISO 8601](#) in Wikipedia.

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

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

Im folgenden Beispiel wird das Jahrtausend aus einem Zeitstempelliteral ermittelt. Die Berechnung des Jahrtausends erfolgt gemäß ISO-Standard 8601. Weitere Informationen finden Sie unter [ISO 8601](#) in Wikipedia.

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

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

Im folgenden Beispiel werden die Mikrosekunden aus einem Zeitstempelliteral ermittelt. Die Berechnung der Mikrosekunden erfolgt gemäß ISO-Standard 8601. Weitere Informationen finden Sie unter [ISO 8601](#) in Wikipedia.

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

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

Im folgenden Beispiel wird der Monat aus einem Datumsliteral ermittelt.

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

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

Im folgenden Beispiel wird die Funktion DATE_PART auf eine Spalte in einer Tabelle angewendet.

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

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

Sie können Datumsteile ausschreiben oder abkürzen; in diesem Fall steht w für Wochen.

Der Datumsteil „Wochentag“ gibt eine Ganzzahl zwischen 0 und 6 aus, beginnend mit Sonntag. Verwenden Sie DATE_PART mit dow (DAYOFWEEK) zur Anzeige von Ereignissen an einem Samstag.

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

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

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('datepart', timestamp)
```

Argumente

datepart

Der Datumsteil, auf den der Zeitstempelwert verkürzt werden soll. Die Eingabe timestamp wird entsprechend der Genauigkeit der Eingabe datepart verkürzt. Bei Verwendung von month beispielsweise wird auf den ersten Tag des Monats verkürzt. Gültige Formate sind folgende:

- Mikrosekunde, Mikrosekunden
- Millisekunde, Millisekunden
- Sekunde, Sekunden
- Minute, Minuten
- Stunde, Stunden
- Tag, Tage
- Woche, Wochen
- Monat, Monate
- Quartal, Quartale
- Jahr, Jahre
- Dekade, Dekaden
- Jahrhundert, Jahrhunderte
- Jahrtausend, Jahrtausende

Weitere Informationen zu Abkürzungen einiger Formate finden Sie unter [Datumsteile für Datums- oder Zeitstempelfunktionen](#)

timestamp

Eine Zeitstempelspalte bzw. ein entsprechender Ausdruck, die/der implizit zu einem Zeitstempel konvertiert wird.

Rückgabebetyp

TIMESTAMP

Beispiele

Verkürzen des Eingabezeitstempels auf die Sekunde

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

Verkürzen des Eingabezeitstempels auf die Minute

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

Verkürzen des Eingabezeitstempels auf die Stunde

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

Verkürzen des Eingabezeitstempels auf den Tag

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

Verkürzen des Eingabezeitstempels auf den ersten Tag eines Monats

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

Verkürzen des Eingabezeitstempels auf den ersten Tag eines Quartals

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

Verkürzen des Eingabezeitstempels auf den ersten Tag eines Jahres

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

```
2020-01-01 00:00:00
```

Verkürzen des Eingabezeitstempels auf den ersten Tag eines Jahrhunderts

```
SELECT DATE_TRUNC('millennium', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2001-01-01 00:00:00
```

Verkürzen des Eingabezeitstempels auf den Montag der Woche.

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

Im folgenden Beispiel verwendet die Funktion DATE_TRUNC den Datumsteil „Woche“ zur Rückgabe des Datums des Montags jeder Woche.

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

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

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.

Rückgabebetyp

`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 `TIMESTAMP`

Im folgenden Beispiel werden die Wochennummern für Verkäufe bestimmt, bei denen der gezahlte Preis 10 000 USD oder mehr betrug.

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

salesid	weeknum
159073	6
160318	8
161723	26

Im folgenden Beispiel wird der Minutenwert aus einem Literal-Zeitstempel-Wert zurückgegeben.

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

date_part
--

Im folgenden Beispiel wird der Millisekundenwert aus einem Literal-Timestamp-Wert zurückgegeben.

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

date_part
-----
101
```

Beispiele mit TIMESTAMPTZ

Im folgenden Beispiel wird der Jahreswert aus einem Literal-Timestamptz-Wert zurückgegeben.

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

date_part
-----
1997
```

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
```

Im folgenden Beispiel wird Millisekunden aus einem Literalwert extrahiert.

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

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

Beispiele mit TIMETZ

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 Stunden aus jedem `timetz_val` extrahiert.

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

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

Im folgenden Beispiel wird Millisekunden aus einem Literalwert extrahiert. Literale werden nicht in UTC konvertiert, bevor die Extraktion verarbeitet wurde.

```
select extract(ms from timetz '18:25:33.123456 EST');  
  
date_part  
-----  
      123
```

Im folgenden Beispiel wird die Stunde der Zeitonenabweichung von UTC aus einem Literal-Timetz-Wert zurückgegeben.

```
select extract(timezone_hour from timetz '1.12.1997 07:37:16.00 PDT');  
  
date_part  
-----  
      -7
```

GETDATE Funktion

Die GETDATE Funktion gibt das aktuelle Datum und die aktuelle Uhrzeit in der Zeitzone der aktuellen Sitzung (standardmäßig UTC) zurück.

Es gibt das Startdatum oder die Uhrzeit der aktuellen Anweisung zurück, auch wenn es sich innerhalb eines Transaktionsblocks befindet.

Syntax

```
GETDATE()
```

Die Klammern sind erforderlich.

Rückgabebetyp

TIMESTAMP

Beispiel

Im folgenden Beispiel wird die GETDATE Funktion verwendet, um den vollständigen Zeitstempel für das aktuelle Datum zurückzugeben.

```
select getdate();
```

Funktion SYSDATE

SYSDATE gibt das aktuelle Datum und die aktuelle Uhrzeit in der Zeitzone der aktuellen Sitzung (standardmäßig UTC) aus.

Note

SYSDATE gibt das Startdatum und die Uhrzeit für die aktuelle Transaktion aus, nicht für den Start der aktuellen Anweisung.

Syntax

```
SYSDATE
```

Für diese Funktion sind keine Argumente erforderlich.

Rückgabebetyp

TIMESTAMP

Beispiele

Das folgende Beispiel verwendet die Funktion SYSDATE zur Rückgabe des vollständigen Zeitstempels für das aktuelle Datum.

```
select sysdate;

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

Das folgende Beispiel verwendet die Funktion SYSDATE innerhalb der Funktion TRUNC zur Rückgabe des aktuellen Datums ohne die Uhrzeit.

```
select trunc(sysdate);
```

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

Die folgende Abfrage gibt Vertriebsinformationen für Daten zurück, die zwischen dem Datum der Ausgabe der Abfrage und dem 120 Tage davor liegenden Datum liegen.

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

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

Funktion TIMEOFDAY

TIMEOFDAY ist ein spezielles Alias zur Ausgabe von Wochentag, Datum und Uhrzeit als Zeichenfolgenwert. Es gibt die Tageszeitzeichenfolge für die aktuelle Anweisung zurück, auch wenn es sich innerhalb eines Transaktionsblocks befindet.

Syntax

```
TIMEOFDAY()
```

Rückgabebetyp

VARCHAR

Beispiele

Das folgende Beispiel zeigt die Ausgabe des aktuellen Datums und der Uhrzeit mit der Funktion TIMEOFDAY.

```
select timeofday();
```

```
timeofday
-----
Thu Sep 19 22:53:50.333525 2013 UTC
(1 row)
```

Funktion TO_TIMESTAMP

TO_TIMESTAMP konvertiert eine TIMESTAMP-Zeichenfolge zu TIMESTAMPTZ.

Syntax

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

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

Argumente

timestamp

Eine Zeichenfolge, die für einen Zeitstempelwert in dem von format angegebenen Format steht. Wenn dieses Argument leer gelassen wird, wird der Zeitstempelwert standardmäßig auf `0001-01-01 00:00:00` gesetzt.

format

Ein Zeichenfolgeliteral, das das Format des timestamp-Werts definiert. Formate, die eine Zeitzone (**TZ**, **tz** oder **OF**) enthalten, werden als Eingabe nicht unterstützt. Für gültige Zeitstempelformate vgl. [Datum-/Uhrzeit-Formatzeichenfolgen](#).

is_strict

Ein optionaler boolescher Wert, der angibt, ob ein Fehler zurückgegeben wird, wenn ein Eingabezeitstempelwert außerhalb des zulässigen Bereichs liegt. Wenn `is_strict` auf `TRUE` gesetzt wird, wird ein Fehler zurückgegeben, wenn ein Wert außerhalb des zulässigen Bereichs liegt. Wenn `is_strict` auf `FALSE` gesetzt wird, was die Standardeinstellung ist, sind Überlaufwerte zulässig.

Rückgabebetyp

TIMESTAMPTZ

Beispiele

Das folgende Beispiel zeigt die Verwendung der Funktion `TO_TIMESTAMP` zur Konvertierung einer `TIMESTAMP`-Zeichenfolge in einen `TIMESTAMPTZ`.

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

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

Es 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 `TIMESTAMPTZ`-Wert. Das Ergebnis ist ein `TIMESTAMPTZ`-Wert, der auf den nächsten Tag fällt, da die Anzahl der Stunden 24 übersteigt:

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

```
to_timestamp
-----
2011-12-19 00:38:15+00
```

Die folgende SQL-Anweisung konvertiert die Zeichenfolge `'2011-12-18 24:38:15'` in einen `TIMESTAMPTZ`-Wert. Das Ergebnis ist ein Fehler, da der Zeitwert im Zeitstempel 24 Stunden übersteigt:

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

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

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:

- DATEADD
- DATEDIFF
- 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	<p>dayofweek, dow, dw, weekday (unterstützt von DATE_PART und Funktion EXTRACT)</p> <p>Gibt eine Ganzzahl von 0–6 aus, beginnend mit Sonntag.</p>

Note

Der Datumsteil DOW verhält sich anders als der Datumsteil „Wochentag (D)“ für Datumsteilformate ickenfolgen. D basiert auf den Ganzzahlen 1–7, wobei die

Datumsteil oder Uhrzeitteil	Abkürzungen
	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 ist relativ zum 1970-01-01 00:00:00.00000, 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 DECADE oder DECADES DATEPART basierend auf dem gemeinsamen Kalender. 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 ein MIL so, dass es 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)

```

Hash-Funktionen

Eine Hash-Funktion ist eine mathematische Funktion, die einen numerischen Eingabewert in einen anderen Wert umwandelt. AWS Clean Roomsunterstützt die folgenden Hashfunktionen:

Themen

- [Die Funktion MD5](#)
- [Die Funktion SHA](#)
- [Die Funktion SHA1](#)
- [Die Funktion SHA2](#)
- [MURMUR3_32_HASH](#)

Die Funktion MD5

Verwendet die kryptografische Hash-Funktion MD5, um eine Zeichenfolge mit variabler Länge in eine Zeichenfolge mit 32 Zeichen zu konvertieren, die eine Textdarstellung des hexadezimalen Werts einer 128-Bit-Prüfsumme ist.

Syntax

```
MD5(string)
```

Argumente

string

Eine Zeichenfolge mit variabler Länge.

Rückgabebetyp

Die MD5-Funktion gibt eine Zeichenfolge mit 32 Zeichen zurück, die eine Textdarstellung des hexadezimalen Werts 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 der SHA1-Funktion.

Siehe [Die Funktion SHA1](#).

Die Funktion SHA1

Die SHA1-Funktion verwendet die kryptografische Hash-Funktion SHA1, um eine Zeichenfolge mit variabler Länge in eine Zeichenfolge mit 40 Zeichen zu konvertieren, die eine Textdarstellung des hexadezimalen Werts einer 160-Bit-Prüfsumme ist.

Syntax

SHA1 ist ein Synonym für. [Die Funktion SHA](#)

```
SHA1(string)
```

Argumente

string

Eine Zeichenfolge mit variabler Länge.

Rückgabetyt

Die SHA1-Funktion gibt eine Zeichenfolge mit 40 Zeichen zurück, die eine Textdarstellung des hexadezimalen Werts 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');
```

Die Funktion SHA2

Die SHA2-Funktion verwendet die kryptografische Hash-Funktion SHA2, um eine Zeichenfolge mit variabler Länge in eine Zeichenkette zu konvertieren. 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.

Rückgabetyt

Die SHA2-Funktion gibt eine Zeichenkette zurück, die eine Textdarstellung des Hexadezimalwerts der Prüfsumme oder eine leere Zeichenfolge ist, 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);
```

MURMUR3_32_HASH

Die Funktion MURMUR3_32_HASH berechnet den nicht kryptografischen 32-Bit-Murmur3A-Hash für alle gängigen Datentypen, einschließlich numerischer Datentypen und Zeichenfolgentypen.

Syntax

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

Argumente

Wert

Der Eingabewert für den Hash. AWS Clean Roomshasht die binäre Darstellung des Eingabewerts. Dieses Verhalten ähnelt FNV_HASH, aber der Wert wird in die binäre Darstellung konvertiert, die in der 32-Bit-Murmur3-Hash-Spezifikation von [Apache Iceberg](#) spezifiziert ist.

Seed

Der INT-Seed der Hash-Funktion. Dieses Argument ist optional. Falls nicht angegeben, wird der Standardstartwert 0 AWS Clean Rooms verwendet. Dies ermöglicht eine Kombination des Hashs mehrerer Spalten ohne Konvertierungen oder Verkettungen.

Rückgabebetyp

Die Funktion gibt einen INT-Wert zurück.

Beispiel

Die folgenden Beispiele geben den Murmur3-Hash einer Zahl, die Zeichenfolge 'AWS Clean Rooms' und die Verkettung der beiden zurück.

```
select MURMUR3_32_HASH(1);

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

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

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

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

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

Nutzungshinweise

Um den Hash einer Tabelle mit mehreren Spalten zu berechnen, können Sie den Murmur3-Hash der ersten Spalte berechnen und ihn als Seed an den Hash der zweiten Spalte übergeben. Dann wird der Murmur3-Hash der zweiten Spalte als Seed an den Hash der dritten Spalte übergeben.

Im folgenden Beispiel werden Seeds erstellt, um eine Tabelle mit mehreren Spalten zu hashen.

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

Mit derselben Eigenschaft kann der Hash einer Verkettung von Zeichenfolgen berechnet werden.

```
select MURMUR3_32_HASH('abcd');
```

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

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

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

Die Hash-Funktion verwendet den Typ der Eingabe, um die Anzahl der zu hashenden Bytes zu bestimmen. Verwenden Sie `Übertragen`, um einen bestimmten Typ zu erzwingen, falls erforderlich.

In den folgenden Beispielen werden verschiedene Eingabetypen verwendet, um unterschiedliche Ergebnisse zu erzielen.

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

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

```
select MURMUR3_32_HASH(1);
```



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

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

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

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.

Die AWS Clean Rooms-JSON-Funktionen und der AWS Clean Rooms-COPY-Befehl verwenden dieselben Methoden, um mit Daten im JSON-Format zu arbeiten.

Themen

- [Funktion CAN_JSON_PARSE](#)
- [Die Funktion „JSON_EXTRACT_ARRAY_ELEMENT_TEXT“](#)
- [Die Funktion JSON_EXTRACT_PATH_TEXT](#)
- [Funktion JSON_PARSE](#)

- [Funktion JSON_SERIALISE](#)
- [Funktion JSON_SERIALIZE_TO_VARBYTE](#)

Funktion CAN_JSON_PARSE

Die Funktion `CAN_JSON_PARSE` analysiert Daten im JSON-Format und gibt `true` zurück, wenn das Ergebnis mithilfe der Funktion `JSON_PARSE` in einen SUPER-Wert konvertiert werden kann.

Syntax

```
CAN_JSON_PARSE(json_string)
```

Argumente

`json_string`

Ein Ausdruck, der serialisierte JSON-Datentypen im VARBYTE- oder VARCHAR-Formular zurückgibt.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um festzustellen, ob das JSON-Array `[10001, 10002, "abc"]` in den Datentyp SUPER konvertiert werden kann.

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

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

Die Funktion „JSON_EXTRACT_ARRAY_ELEMENT_TEXT“

Die Funktion `JSON_EXTRACT_ARRAY_ELEMENT_TEXT` gibt ein JSON-Array-Element im äußersten Array einer JSON-Zeichenfolge unter Verwendung eines nullbasierten Index zurück. Das erste Element in einem Array befindet sich an Position 0. Wenn der Index negativ ist oder sich außerhalb des Bereichs befindet, gibt `JSON_EXTRACT_ARRAY_ELEMENT_TEXT` eine leere Zeichenfolge zurück. Wenn das Argument `null_if_invalid` auf `true` gesetzt und die JSON-Zeichenfolge ungültig ist, gibt die Funktion anstatt eines Fehlers „NULL“ zurück.

Weitere Informationen finden Sie unter [JSON-Funktionen](#).

Syntax

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

Argumente

`json_string`

Eine korrekt formatierte JSON-Zeichenfolge.

`pos`

Eine Ganzzahl, die unter Verwendung eines nullbasierten Array-Index den Index des Array-Elements darstellt, das zurückgegeben werden soll.

`null_if_invalid`

Ein Boolescher Wert, der angibt, ob anstatt eines Fehlers „NULL“ zurückgegeben wird, wenn die JSON-Eingabezeichenfolge ungültig ist. Geben Sie `true` (t) an, damit „NULL“ zurückgegeben wird, wenn die JSON-Eingabezeichenfolge ungültig ist. Geben Sie `false` (f) an, damit ein Fehler zurückgegeben wird, wenn die JSON-Eingabezeichenfolge ungültig ist. Der Standardwert ist `false`.

Rückgabebetyp

Eine VARCHAR-Zeichenfolge, die das JSON-Array-Element darstellt, das von `pos` referenziert wird.

Beispiel

Das folgende Beispiel gibt ein Array-Element an Position 2 zurück, das das dritte Element eines null-basierten Array-Index ist:

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

json_extract_array_element_text
-----
113
```

Im folgenden Beispiel wird ein Fehler zurückgegeben, weil die JSON-Eingabezeichenfolge ungültig ist.

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

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

Im folgenden Beispiel wird `null_if_invalid` auf `true` gesetzt, sodass die Anweisung anstatt eines Fehlers „NULL“ zurückgibt, wenn die JSON-Eingabezeichenfolge ungültig ist.

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

json_extract_array_element_text
-----
```

Die Funktion JSON_EXTRACT_PATH_TEXT

Die Funktion `JSON_EXTRACT_PATH_TEXT` gibt den Wert für das Schlüssel:Wert-Paar zurück, auf das in einer Reihe von Pfadelementen in einer JSON-Zeichenfolge verwiesen wird. Der JSON-Pfad kann bis zu einer Tiefe von fünf Ebenen verschachtelt sein. Pfadelemente unterscheiden zwischen Groß- und Kleinschreibung. Wenn in der JSON-Zeichenfolge ein Pfadelement nicht vorhanden ist, gibt `JSON_EXTRACT_PATH_TEXT` eine leere Zeichenfolge zurück. Wenn das Argument `null_if_invalid` auf `true` gesetzt und die JSON-Zeichenfolge ungültig ist, gibt die Funktion anstatt eines Fehlers „NULL“ zurück.

Informationen zu zusätzlichen JSON-Funktionen finden Sie unter [JSON-Funktionen](#).

Syntax

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

Argumente

json_string

Eine korrekt formatierte JSON-Zeichenfolge.

path_elem

Ein Pfadelement in einer JSON-Zeichenfolge. Es ist mindestens ein Pfadelement erforderlich. Es können zusätzliche Pfadelemente angegeben werden, bis zu einer Tiefe von fünf Ebenen.

null_if_invalid

Ein Boolescher Wert, der angibt, ob anstatt eines Fehlers „NULL“ zurückgegeben wird, wenn die JSON-Eingabezeichenfolge ungültig ist. Geben Sie `true` (t) an, damit „NULL“ zurückgegeben wird, wenn die JSON-Eingabezeichenfolge ungültig ist. Geben Sie `false` (f) an, damit ein Fehler zurückgegeben wird, wenn die JSON-Eingabezeichenfolge ungültig ist. Der Standardwert ist `false`.

AWS Clean Rooms erkennt in einer JSON-Zeichenfolge `\n` als Zeichen für neue Zeilen und `\t` als Tabulatorzeichen. Um einen Backslash zu laden, muss ein Backslash als Escape-Zeichen verwendet werden (`\\`).

Rückgabebetyp

Eine VARCHAR-Zeichenfolge, die den JSON-Wert darstellt, der von den Pfadelementen referenziert wird.

Beispiel

Im folgenden Beispiel wird der Wert für den Pfad `'f4'`, `'f6'` zurückgegeben.

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

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

Im folgenden Beispiel wird ein Fehler zurückgegeben, weil die JSON-Eingabezeichenfolge ungültig ist.

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

An error occurred when executing the SQL command:

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

Im folgenden Beispiel wird `null_if_invalid` auf `true` gesetzt, sodass die Anweisung anstatt eines Fehlers „NULL“ zurückgibt, wenn die JSON-Eingabezeichenfolge ungültig ist.

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

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

Im folgenden Beispiel wird der Wert für den Pfad `'farm', 'barn', 'color'` zurückgegeben, wobei sich der abgerufene Wert auf der dritten Ebene befindet. Dieses Beispiel ist mit einem JSON-Lint-Tool formatiert, um das Lesen zu vereinfachen.

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

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

Das folgende Beispiel gibt NULL zurück, da das `'color'`-Element fehlt. Dieses Beispiel ist mit einem JSON-Lint-Tool formatiert.

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

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

Wenn das JSON-Format gültig ist, wird beim Versuch, ein fehlendes Element zu extrahieren, NULL zurückgegeben.

Im folgenden Beispiel wird der Wert für den Pfad 'house', 'appliances', 'washing machine', 'brand' zurückgegeben.

```
select json_extract_path_text('{
  "house": {
    "address": {
      "street": "123 Any St.",
      "city": "Any Town",
      "state": "FL",
      "zip": "32830"
    },
    "bathroom": {
      "color": "green",
      "shower": true
    },
    "appliances": {
      "washing machine": {
        "brand": "Any Brand",
        "color": "beige"
      },
      "dryer": {
        "brand": "Any Brand",
        "color": "white"
      }
    }
  }
}', 'house', 'appliances', 'washing machine', 'brand');
```

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


Any Brand

Funktion JSON_PARSE

Die Funktion `JSON_PARSE` analysiert Daten im JSON-Format und konvertiert sie in die SUPER-Darstellung.

Verwenden Sie die `JSON_PARSE`-Funktion, um mit dem Befehl `INSERT` oder `UPDATE` in den SUPER-Datentyp aufzunehmen. Wenn Sie `JSON_PARSE()` zum Parsing von JSON-Zeichenfolgen in SUPER-Werte verwenden, gelten bestimmte Einschränkungen.

Syntax

```
JSON_PARSE(json_string)
```

Argumente

`json_string`

Ein Ausdruck, der eine serialisierte JSON-Zeichenfolge als Datentyp `varbyte` oder `varchar` zurückgibt.

Rückgabebetyp

SUPER

Beispiel

Im Folgenden sehen Sie ein Beispiel für die `JSON_PARSE`-Funktion.

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

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

```
(1 row)
```

Funktion JSON_SERIALIZE

Die Funktion `JSON_SERIALIZE` serialisiert einen SUPER-Ausdruck in eine textbasierte JSON-Darstellung gemäß RFC 8259. Weitere Informationen zu diesem RFC finden Sie unter [The JavaScript Object Notation \(JSON\) Data Interchange Format](#).

Das SUPER-Größenlimit entspricht ungefähr dem Blocklimit, und das varchar-Limit ist kleiner als das SUPER-Größenlimit. Daher gibt die Funktion `JSON_SERIALIZE` einen Fehler zurück, wenn das JSON-Format das varchar-Limit des Systems überschreitet.

Syntax

```
JSON_SERIALIZE(super_expression)
```

Argumente

`super_expression`

Ein SUPER-Ausdruck oder eine SUPER-Spalte.

Rückgabebetyp

varchar

Beispiel

Im folgenden Beispiel wird ein SUPER-Wert zu einer Zeichenfolge serialisiert.

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

Funktion JSON_SERIALIZE_TO_VARBYTE

Die Funktion `JSON_SERIALIZE_TO_VARBYTE` konvertiert einen SUPER-Wert in eine ähnliche JSON-Zeichenfolge wie bei `JSON_SERIALIZE()`, jedoch in einem VARBYTE-Wert gespeichert.

Syntax

```
JSON_SERIALIZE_TO_VARBYTE(super_expression)
```

Argumente

super_expression

Ein SUPER-Ausdruck oder eine SUPER-Spalte.

Rückgabebetyp

varbyte

Beispiel

Im folgenden Beispiel wird ein SUPER-Wert serialisiert und das Ergebnis im VARBYTE-Format zurückgegeben.

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

```
json_serialize_to_varbyte
```

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

Im folgenden Beispiel wird ein SUPER-Wert serialisiert und das Ergebnis in VARCHAR-Format umgewandelt.

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

```
json_serialize_to_varbyte
```

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

Mathematische Funktionen

In diesem Abschnitt werden die mathematischen Operatoren und Funktionen beschrieben, die in AWS Clean Rooms unterstützt werden.

Themen

- [Symbole für mathematische Operatoren](#)
- [Funktion ABS](#)
- [Die Funktion ACOS](#)
- [Die Funktion ASIN](#)
- [Die Funktion ATAN](#)
- [Die Funktion ATAN2](#)
- [Die Funktion CBRT](#)
- [Die Funktion CEILING \(oder CEIL\)](#)
- [Die Funktion COS](#)
- [Die Funktion COT](#)
- [Die Funktion DEGREES](#)
- [Die Funktion DEXP](#)
- [Die Funktion DLOG1](#)
- [Die Funktion DLOG10](#)
- [Die Funktion EXP](#)
- [Die Funktion FLOOR](#)
- [Die Funktion LN](#)
- [Die Funktion LOG](#)
- [Die Funktion MOD](#)
- [Die Funktion PI](#)
- [Die Funktion POWER](#)
- [Die Funktion RADIANS](#)
- [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
/	Quadratwurzel	/ 25,0	5
/	Kubikwurzel	/ 27,0	3
@	Absoluter Wert	@ -5,0	5

Beispiele

Berechnen Sie die gezahlte Provision zuzüglich einer Bearbeitungsgebühr von 2,00 USD 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 10 Jahre 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)
```

Suchen Sie den gezahlten Gesamtpreis und die Provision für Verkäufe mit einer Datums-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, DECIMAL, FLOAT4 oder FLOAT8 handeln.

Rückgabebetyp

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.

Rückgabebetyp

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.

Rückgabebetyp

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 π .

Syntax

```
ATAN(number)
```

Argumente

number (Zahl)

Der Eingabeparameter ist eine DOUBLE PRECISION-Zahl.

Rückgabebetyp

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 |
+-----+
```

Die Funktion ATAN2

ATAN2 ist eine trigonometrische Funktion, die den Arcustangens einer Zahl dividiert durch eine andere Zahl zurückgibt. Der Rückgabewert wird in Radianten ausgedrückt und liegt zwischen $\text{PI}/2$ und $-\text{PI}/2$.

Syntax

```
ATAN2(number1, number2)
```

Argumente

number1

Eine DOUBLE PRECISION-Zahl.

number2

Eine DOUBLE PRECISION-Zahl.

Rückgabebetyp

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.

Rückgabebetyp

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, DECIMAL, FLOAT4 oder FLOAT8 handeln.

Rückgabebetyp

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.

Rückgabebetyp

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.

Rückgabebetyp

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.

Rückgabebetyp

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-Radianen in Grad zu konvertieren.

```
SELECT DEGREES(pi());
```

```
+-----+
| degrees |
+-----+
| 180 |
+-----+
```

Die Funktion DEXP

Die DEXP-Funktion gibt den exponentiellen SPLI-Wert in wissenschaftlicher Notierung für eine Doppelpräzisionsnummer zurück. Der einzige Unterschied zwischen den Funktionen DEXP und EXP besteht darin, dass es sich beim Parameter für DEXP um eine `DOUBLE PRECISION` handeln muss.

Syntax

```
DEXP(number)
```

Argument

`number` (Zahl)

Der Eingabeparameter ist eine `DOUBLE PRECISION`-Zahl.

Rückgabebetyp

DOUBLE PRECISION

Beispiel

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

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

Die Funktion DLOG1

Die DLOG1-Funktion gibt den natürlichen Logarithmus des Eingabeparameters zurück.

Die Funktion DLOG1 ist ein Synonym von [Die Funktion LN](#).

Die Funktion DLOG10

Die DLOG10-Funktion gibt den Logarithmus des Eingabeparameters zur Basis 10 zurück.

Die Funktion DLOG10 ist ein Synonym von [Die Funktion LOG](#).

Syntax

```
DLOG10(number)
```

Argument

number (Zahl)

Der Eingabeparameter ist eine Doppelpräzisionszahl.

Rückgabebetyp

Die DLOG10-Funktion gibt eine Doppelpräzisionszahl zurück.

Beispiel

Im folgenden Beispiel wird der Logarithmus der Zahl 100 zur Basis 10 zurückgegeben:

```
select dlog10(100);

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

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.

Rückgabebetyp

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, DECIMAL, FLOAT4 oder FLOAT8 handeln.

Rückgabebetyp

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.

Die LN-Funktion ist ein Synonym von [Die Funktion DLOG1](#).

Syntax

```
LN(expression)
```

Argument

expression

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

Note

Diese Funktion gibt einen Fehler für einige Datentypen zurück, wenn der Ausdruck auf eine vom AWS Clean Rooms 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

Rückgabebetyp

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 einer Zahl zur Basis 10 zurück.

Synonym von [Die Funktion DLOG10](#).

Syntax

```
LOG(number)
```

Argument

number (Zahl

Der Eingabeparameter ist eine Doppelpräzisionszahl.

Rückgabebetyp

Die LOG-Funktion gibt eine Doppelpräzisionszahl zurück.

Beispiel

Im folgenden Beispiel wird der Logarithmus der Zahl 100 zur Basis 10 zurückgegeben:

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

Rückgabetyt

Gültige Rückgabetyt sind DECIMAL, INT, SMALLINT und BIGINT. Der Rückgabetyt 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 Rückgabetyt 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()
```

Rückgabebetyp

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

```
{POW | POWER}(expression1, expression2)
```


Argumente

expression1

Der numerische Ausdruck, der potenziert werden soll. Muss ein INTEGER-, DECIMAL- oder FLOAT-Datentyp sein.

expression2

Potenz, mit der expression1 potenziert werden soll. Muss ein INTEGER-, DECIMAL- oder FLOAT-Datentyp sein.

Rückgabebetyp

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.

Rückgabebetyp

DOUBLE PRECISION

Beispiel

Verwenden Sie das folgende Beispiel, um die Entsprechung in 180 Grad des Radianten zurückzugeben.

```
SELECT RADIANS(180);
```

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

Die Funktion RANDOM

Die RANDOM-Funktion generiert einen zufälligen Wert zwischen 0,0 (einschließlich) und 1,0 (ausschließlich).

Syntax

```
RANDOM()
```

Rückgabebetyp

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 Befehl SET verwendet, um einen SEED-Wert festzulegen, sodass RANDOM eine vorhersehbare Folge von Zahlen 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. Es kann der Typ DECIMAL oder FLOAT8 sein. 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.

Rückgabebetyp

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 Typ DECIMAL oder FLOAT8 sein. AWS Clean Rooms kann andere Datentypen gemäß den impliziten Konvertierungsregeln konvertieren.

Rückgabebetyp

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 bestimmen, 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.

Rückgabebetyp

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 Gleitkommatyp haben. Der Ausdruck kann Funktionen enthalten. Das System könnte implizite Typumwandlungen durchführen.

Rückgabebetyp

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 in finden Sie AWS Clean Roomsunter [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. Es kann der Typ DECIMAL oder FLOAT8 sein. 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.)

Rückgabebetyp

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 Rückgabebetyp ist ein Datum.

```

select trunc(starttime) from event
order by eventid limit 1;

trunc
-----
2008-01-25
(1 row)

```

Zeichenfolgenfunktionen

Themen

- [Der Operator || \(Verkettung\)](#)
- [Die Funktion BTRIM](#)
- [Die Funktion CHAR_LENGTH](#)
- [Die Funktion CHARACTER_LENGTH](#)

- [Funktion CHARINDEX](#)
- [Funktion CONCAT](#)
- [Die Funktionen LEFT und RIGHT](#)
- [Die Funktion LEN](#)
- [Die Funktion LENGTH](#)
- [Die Funktion LOWER](#)
- [Die Funktionen LPAD und RPAD](#)
- [Die Funktion LTRIM](#)
- [Die Funktion POSITION](#)
- [Die Funktion REGEXP_COUNT](#)
- [Die Funktion REGEXP_INSTR](#)
- [Die Funktion REGEXP_REPLACE](#)
- [Die Funktion REGEXP_SUBSTR](#)
- [Die Funktion REPEAT](#)
- [Die Funktion REPLACE](#)
- [Die Funktion REPLICATE](#)
- [Die Funktion REVERSE](#)
- [Die Funktion RTRIM](#)
- [Funktion SOUNDEX](#)
- [Die Funktion SPLIT_PART](#)
- [Die Funktion STRPOS](#)
- [Funktion SUBSTRING](#)
- [Die Funktion SUBSTRING](#)
- [Die Funktion TEXTLEN](#)
- [Die Funktion TRANSLATE](#)
- [Die Funktion TRIM](#)
- [Die Funktion UPPER](#)

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.

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.

Rückgabetyt

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 verketteten, 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.

Rückgabebetyp

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 ' xyzaxyzbxyzxyz ' entfernt. Die Zeichenfolgen ' xyz ' am Anfang und am Ende werden entfernt, entsprechende Zeichenfolgen innerhalb dieser Zeichenfolge jedoch nicht.

```
select 'xyzaxyzbxyzxyz' as untrim,
```



```
btrim('xyzaxyzbxyzxyz', 'xyz') as trim;
```

```

      untrim      |      trim
-----+-----
xyzaxyzbxyzxyz | 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

```

Die Funktion CHAR_LENGTH

Synonym mit der Funktion LEN.

Siehe [Die Funktion LEN](#).

Die Funktion CHARACTER_LENGTH

Synonym mit der Funktion LEN.

Siehe [Die Funktion LEN](#).

Funktion CHARINDEX

Gibt den Ort der angegebenen Unterzeichenfolge innerhalb einer Zeichenfolge zurück.

Ähnliche Funktionen finden Sie unter [Die Funktion POSITION](#) und [Die Funktion STRPOS](#).

Syntax

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

Argumente

substring

Die Unterzeichenfolge, die innerhalb der Zeichenfolge gesucht werden soll.

string

Die Zeichenfolge oder Spalte, die durchsucht werden soll.

Rückgabebetyp

Die CHARINDEX-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

CHARINDEX gibt 0 zurück, wenn die Unterzeichenfolge nicht innerhalb des gefunden wird `string`:

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

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

Beispiele

Im folgenden Beispiel wird die Position der Zeichenfolge `fish` innerhalb des Worts `dogfish` gezeigt:

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

```
charindex
-----
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 charindex('.', commission), count (charindex('.', commission))
from sales where charindex('.', commission) > 4 group by charindex('.', commission)
```

```
order by 1,2;

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

Funktion CONCAT

Die CONCAT-Funktion verkettet zwei Ausdrücke und gibt den Ergebnisausdruck zurück. Um mehr als zwei Ausdrücke zu verketteten, 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.

Rückgabebetyp

CONCAT gibt einen Ausdruck zurück. Der Datentyp des Ausdrucks ist derselbe Typ wie die Eingabeargumente.

Wenn die Eingabeausdrücke von unterschiedlichen Typen sind, AWS Clean Rooms wandelt einen der Ausdrücke implizit um. Wenn Werte nicht umgewandelt werden können, wird ein Fehler zurückgegeben.

Beispiele

Im folgenden Beispiel werden zwei Zeichenlitterale 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)
```

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.

Rückgabebetyp

LEFT und RIGHT geben eine VARCHAR-Zeichenfolge zurück.

Beispiel

Im folgenden Beispiel werden die 5 Zeichen in Veranstaltungsnamen mit IDs zwischen 1000 und 1005 zurückgegeben, die sich am weitesten links oder am weitesten rechts befinden:

```
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 LEN

Gibt die Länge der angegebenen Zeichenfolge durch die Anzahl der Zeichen an.

Syntax

LEN ist synonym mit [Die Funktion LENGTH](#), [Die Funktion CHAR_LENGTH](#), [Die Funktion CHARACTER_LENGTH](#) und [Die Funktion TEXTLEN](#).

```
LEN(expression)
```

Argument

expression

Der Eingabeparameter ist ein CHAR oder VARCHAR oder ein Alias eines der gültigen Eingabetypen.

Rückgabebetyp

Die LEN-Funktion gibt eine Ganzzahl zurück, die die Anzahl der Zeichen in der Eingabezeichenfolge anzeigt.

Wenn es sich um eine Folge von Zeichen handelt, gibt die LEN-Funktion die tatsächliche Anzahl der Zeichen in Multibyte-Zeichenfolgen zurück, nicht die Anzahl der Bytes. Beispielsweise ist eine VARCHAR(12)-Spalte erforderlich, um drei chinesische Zeichen mit vier Bytes zu speichern. Die LEN-Funktion gibt für diese Zeichenfolge 3 zurück.

Nutzungshinweise

Längenberechnungen berücksichtigen im Fall von Zeichenfolgen mit fester Länge keine Leerzeichen am Ende. Im Fall von Zeichenfolgen mit variabler Länge werden sie jedoch berücksichtigt.

Beispiel

Im folgenden Beispiel wird die Anzahl der Bytes und die Anzahl der Zeichen der Zeichenfolge zurückgegeben français.

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

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

Im folgenden Beispiel wird die Anzahl der Zeichen in den Zeichenfolgen cat ohne Leerzeichen am Ende und in cat mit drei Leerzeichen am Ende zurückgegeben:

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

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

Im folgenden Beispiel werden die zehn längsten VENUENAME-Einträge in der Tabelle VENUE zurückgegeben:

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

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


Die Funktion LENGTH

Synonym mit der Funktion LEN.

Siehe [Die Funktion LEN](#).

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).

Rückgabetyt

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.

Rückgabebetyp

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. Die Kürzung ist abgeschlossen, wenn in der Eingabezeichenfolge kein Trimm-Zeichen enthalten ist.

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.

Rückgabebetyp

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 `listtime`-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(venueid, 'CDG')
from venue
where venueid like '%Park'
order by 2
limit 7;
```

venueid	venueid	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.

Ähnliche Funktionen finden Sie unter [Funktion CHARINDEX](#) und [Die Funktion STRPOS](#).

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.

Rückgabebetyp

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 Zeichenfolgenlitterale, 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.

Rückgabebetyp

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@lacusUt nec.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 Zeichenfolgenliterals, 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 Muster mit einem PCRE-Dialekt (Perl Compatible Regular Expression) interpretieren.

Rückgabebetyp

Ganzzahl

Beispiel

Im folgenden Beispiel wird nach dem Zeichen @ gesucht, mit dem Domännennamen 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@lacusUt nec.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 Zeichenfolgenliterals, 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.

Rückgabebetyp

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@lacusUt nec.ca	sed

Im folgenden Beispiel werden die Domännennamen 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@lacusUt nec.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 Zeichenfolgenliterals, 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.

Rückgabebetyp

VARCHAR

Beispiel

Im folgenden Beispiel wird der E-Mail-Adresse-Abschnitt zwischen dem Zeichen @ und der Domänenenerweiterung 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@lacusUtnecc.ca	@lacusUtnecc

Im folgenden Beispiel wird der Teil der Eingabe zurückgegeben, der dem ersten Vorkommen der Zeichenfolge FOX 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 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.

Synonym mit [Die Funktion REPLICATE](#).

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.

Rückgabebetyp

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.

Rückgabebetyp

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 REPLICATE

Synonym mit der Funktion REPEAT.

Siehe [Die Funktion REPEAT](#).

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.

Rückgabotyp

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)

```

Wählt fünf Vertriebs-IDs aus und die entsprechenden umgekehrten IDs umgewandelt in Zeichenfolgen 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. Die Kürzung ist abgeschlossen, wenn in der Eingabezeichenfolge kein Trimm-Zeichen enthalten ist.

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.

Rückgabebetyp

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 'xyzaxyzbxyzxyz' entfernt. Die Zeichenfolgen 'xyz' am Ende werden entfernt, entsprechende Zeichenfolgen innerhalb dieser Zeichenfolge jedoch nicht.

```
select 'xyzaxyzbxyzxyz' as untrim,
rtrim('xyzaxyzbxyzxyz', 'xyz') as trim;

untrim   | trim
-----+-----
xyzaxyzbxyzxyz | 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.

Funktion SOUNDEX

Die Funktion SOUNDEX gibt den amerikanischen Soundex-Wert zurück, der aus dem ersten Buchstaben gefolgt von einer 3-stelligen Kodierung der Laute besteht, die die englische Aussprache der angegebenen Zeichenfolge repräsentieren.

Syntax

```
SOUNDEX(string)
```

Argumente

string

Sie geben eine CHAR- oder VARCHAR-Zeichenfolge an, die Sie in einen amerikanischen Soundex-Codewert konvertieren möchten.

Rückgabetyp

Die Funktion SOUNDEX gibt eine VARCHAR(4)-Zeichenfolge zurück, die aus einem Großbuchstaben gefolgt von einer dreistelligen Kodierung der Laute besteht, die die englische Aussprache repräsentieren.

Nutzungshinweise

Die Funktion SOUNDEX konvertiert nur englische alphabetische ASCII-Zeichen in Klein- und Großbuchstaben, einschließlich a–z und A–Z. SOUNDEX ignoriert andere Zeichen. SOUNDEX gibt einen einzelnen Soundex-Wert für eine Zeichenfolge aus mehreren Wörtern zurück, die durch Leerzeichen getrennt sind.

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

```
soundex  
-----  
A252
```

SOUNDEX gibt eine leere Zeichenfolge zurück, wenn die Eingabezeichenfolge keine englischen Buchstaben enthält.

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

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

Beispiel

Im folgenden Beispiel wird der Soundex A525 für das Wort Amazon zurückgegeben.

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

```
soundex
```

```
-----
```

```
A525
```

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.

Rückgabebetyp

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 STRPOS

Gibt die Position einer Unterzeichenfolge innerhalb einer angegebenen Zeichenfolge zurück.

Ähnliche Funktionen finden Sie unter [Funktion CHARINDEX](#) und [Die Funktion POSITION](#).

Syntax

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

Argumente

string

Der erste Eingabeparameter ist die Zeichenfolge, die durchsucht werden soll.

substring

Der zweite Parameter ist die Unterzeichenfolge, nach der innerhalb der Zeichenfolge gesucht werden soll.

Rückgabebetyp

Die STRPOS-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

STRPOS gibt 0 zurück, wenn die Unterzeichenfolge nicht innerhalb der Zeichenfolge gefunden wird:

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

Beispiele

Im folgenden Beispiel wird die Position der Zeichenfolge fish innerhalb des Worts dogfish gezeigt:

```
select strpos('dogfish', 'fish');
strpos
-----
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 strpos(commission, '.'),
```

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

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

Funktion SUBSTRING

Synonym mit der Funktion SUBSTRING.

Siehe [Die Funktion SUBSTRING](#).

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

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

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

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

Argumente

character_string

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.

number_characters

Die Anzahl der Zeichen, die extrahiert werden soll (die Länge der Unterzeichenfolge). Die number_characters basiert auf der Anzahl der Zeichen, nicht der Bytes. Daher werden Zeichen mit mehreren Bytes als einzelne Zeichen gezählt. 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.

number_bytes

Die Anzahl der Bytes, die extrahiert werden sollen, also die Länge der Unterzeichenfolge. Diese Zahl darf nicht negativ sein.

Rückgabebetyp

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 die Startposition + number_characters die Länge der Zeichenfolge überschreiten, gibt SUBSTRING eine Unterzeichenfolge ab start_position bis zum Ende der Zeichenfolge zurück.

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 + number_characters -1 zurück. Beispiel:

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

Wenn start_position + number_characters -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 TEXTLEN

Synonym mit der Funktion LEN.

Siehe [Die Funktion LEN](#).

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.

Rückgabebetyp

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_Albens
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.

Rückgabebetyp

Die TRIM-Funktion gibt eine VARCHAR- oder eine CHAR_Zeichenfolge zurück. Wenn Sie die TRIM-Funktion mit einem SQL-Befehl verwenden, konvertiert die Ergebnisse AWS Clean Rooms implizit in VARCHAR. Wenn Sie die TRIM-Funktion in der SELECT-Liste für eine SQL-Funktion verwenden, konvertiert die Ergebnisse nicht AWS Clean Rooms implizit und Sie müssen möglicherweise eine explizite Konvertierung durchführen, um einen Datentypfehler zu vermeiden. Weitere Informationen zu expliziten Konvertierungen finden Sie in den Abschnitten zu den Funktionen [CAST-Funktion](#) und [CONVERT-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(venueid, 'CDG')
from venue
where venueid like '%Park'
order by 2
limit 7;
```

venueid	venueid	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).

Rückgabetyt

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)

Funktionen für SUPER-Typinformationen

In diesem Abschnitt werden die Informationsfunktionen für SQL beschrieben, mit denen dynamische Informationen aus Eingaben des SUPER Datentyps abgeleitet werden können, der in unterstützt wird.

AWS Clean Rooms

Themen

- [Die Funktion DECIMAL_PRECISION](#)
- [Die Funktion DECIMAL_SCALE](#)
- [Die Funktion IS_ARRAY](#)
- [Die Funktion IS_BIGINT](#)
- [Die Funktion IS_CHAR](#)
- [Die Funktion IS_DECIMAL](#)
- [Die Funktion IS_FLOAT](#)
- [Die Funktion IS_INTEGER](#)
- [Die Funktion IS_OBJECT](#)
- [Die Funktion IS_SCALAR](#)
- [Die Funktion IS_SMALLINT](#)
- [Die Funktion IS_VARCHAR](#)
- [Die Funktion JSON_TYPEOF](#)

Die Funktion DECIMAL_PRECISION

Überprüft die Genauigkeit der maximalen Gesamtzahl der zu speichernden Dezimalstellen. Diese Zahl enthält sowohl die Vor- als auch Nachkommastellen. Die Genauigkeit liegt zwischen 1 und 38, wobei der Standardwert 38 ist.

Syntax

```
DECIMAL_PRECISION(super_expression)
```

Argumente

super_expression

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

INTEGER

Beispiel

Verwenden Sie das folgende Beispiel, um die Funktion `DECIMAL_PRECISION` auf die Tabelle `t` anzuwenden.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_PRECISION(s) FROM t;
```

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

Die Funktion `DECIMAL_SCALE`

Überprüft die Anzahl der zu speichernden Dezimalstellen nach dem Dezimaltrennzeichen. Die Skalierung liegt zwischen 0 und dem Genauigkeitspunkt, wobei der Standardwert 0 ist.

Syntax

```
DECIMAL_SCALE(super_expression)
```

Argumente

`super_expression`

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

INTEGER

Beispiel

Verwenden Sie das folgende Beispiel, um die Funktion `DECIMAL_SCALE` auf die Tabelle `t` anzuwenden.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_SCALE(s) FROM t;

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

Die Funktion IS_ARRAY

Überprüft, ob eine Variable ein Array ist. Die Funktion gibt `true` zurück, wenn die Variable ein Array ist. Die Funktion umfasst auch leere Arrays. Andernfalls gibt die Funktion `false` für alle anderen Werte zurück, einschließlich null.

Syntax

```
IS_ARRAY(super_expression)
```

Argumente

super_expression

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der IS_ARRAY-Funktion zu überprüfen, ob `[1, 2]` ein Array ist.

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

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

Die Funktion IS_BIGINT

Überprüft, ob ein Wert ein BIGINT ist. Die Funktion IS_BIGINT gibt `true` für Zahlen der Skala 0 im 64-Bit-Bereich zurück. Andernfalls gibt die Funktion `false` für alle anderen Werte zurück, einschließlich null und Gleitkommazahlen.

Die Funktion IS_BIGINT ist eine Obermenge von IS_INTEGER.

Syntax

```
IS_BIGINT(super_expression)
```

Argumente

super_expression

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der IS_BIGINT-Funktion zu überprüfen, ob 5 ein BIGINT ist.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_BIGINT(s) FROM t;
```

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

Die Funktion IS_CHAR

Überprüft, ob ein Wert ein CHAR ist. Die Funktion `IS_CHAR` gibt `true` für Zeichenfolgen zurück, die nur ASCII-Zeichen enthalten, da der CHAR-Typ nur Zeichen im ASCII-Format speichern kann. Für alle anderen Werte gibt die Funktion `false` zurück.

Syntax

```
IS_CHAR(super_expression)
```

Argumente

`super_expression`

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der `IS_CHAR`-Funktion zu überprüfen, ob `t` ein CHAR ist.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('t');

SELECT s, IS_CHAR(s) FROM t;

+-----+-----+
| s | is_char |
+-----+-----+
```

```
+-----+-----+
| "t" | true  |
+-----+-----+
```

Die Funktion IS_DECIMAL

Überprüft, ob ein Wert ein DECIMAL ist. Die Funktion IS_DECIMAL gibt `true` für Zahlen zurück, die keine Gleitkommazahlen sind. Für alle anderen Werte, einschließlich null, gibt die Funktion `false` zurück.

Die Funktion IS_DECIMAL ist eine Obermenge von IS_BIGINT.

Syntax

```
IS_DECIMAL(super_expression)
```

Argumente

super_expression

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der IS_DECIMAL-Funktion zu überprüfen, ob 1.22 ein DECIMAL ist.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (1.22);

SELECT s, IS_DECIMAL(s) FROM t;

+-----+-----+
| s     | is_decimal |
+-----+-----+
```

```
+-----+-----+
| 1.22 | true   |
+-----+-----+
```

Die Funktion IS_FLOAT

Überprüft, ob ein Wert eine Gleitkommazahl ist. Die Funktion IS_FLOAT gibt `true` für Gleitkommazahlen (FLOAT4 und FLOAT8) zurück. Für alle anderen Werte gibt die Funktion `false` zurück.

Die Mengen IS_DECIMAL und IS_FLOAT sind voneinander getrennt.

Syntax

```
IS_FLOAT(super_expression)
```

Argumente

super_expression

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der IS_FLOAT-Funktion zu überprüfen, ob `2.22::FLOAT` ein FLOAT ist.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES(2.22::FLOAT);

SELECT s, IS_FLOAT(s) FROM t;

+-----+-----+
| s     | is_float |
+-----+-----+
```

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

Die Funktion IS_INTEGER

Gibt `true` für Zahlen der Skala 0 im 32-Bit-Bereich zurück und `false` für alles andere (einschließlich Null- und Gleitkommazahlen).

Die Funktion `IS_INTEGER` ist eine Obermenge der Funktion `IS_SMALLINT`.

Syntax

```
IS_INTEGER(super_expression)
```

Argumente

`super_expression`

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der `IS_INTEGER`-Funktion zu überprüfen, ob 5 ein `INTEGER` ist.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_INTEGER(s) FROM t;

+---+-----+
| s | is_integer |
+---+-----+
```



```
| 5 | true |
+---+-----+
```

Die Funktion IS_OBJECT

Überprüft, ob eine Variable ein Objekt ist. Die Funktion `IS_OBJECT` gibt `true` für Objekte zurück, einschließlich leerer Objekte. Für alle anderen Werte, einschließlich `null`, gibt die Funktion `false` zurück.

Syntax

```
IS_OBJECT(super_expression)
```

Argumente

`super_expression`

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der `IS_OBJECT`-Funktion zu überprüfen, ob `{"name": "Joe"}` ein Objekt ist.

```
CREATE TABLE t(s super);

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

SELECT s, IS_OBJECT(s) FROM t;

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

Die Funktion IS_SCALAR

Überprüft, ob eine Variable ein Skalar ist. Die Funktion `IS_SCALAR` gibt `true` für jeden Wert zurück, der kein Array oder Objekt ist. Für alle anderen Werte, einschließlich `null`, gibt die Funktion `false` zurück.

Die Menge von `IS_ARRAY`, `IS_OBJECT` und `IS_SCALAR` deckt alle Werte mit Ausnahme von `null` ab.

Syntax

```
IS_SCALAR(super_expression)
```

Argumente

`super_expression`

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der `IS_SCLALR`-Funktion zu überprüfen, ob `{"name": "Joe"}` ein Skalar ist.

```
CREATE TABLE t(s SUPER);

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

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

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

Die Funktion IS_SMALLINT

Überprüft, ob eine Variable ein SMALLINT ist. Die Funktion IS_SMALLINT gibt `true` für Zahlen der Skala 0 im 16-Bit-Bereich zurück. Für alle anderen Werte, einschließlich null und Gleitkommazahlen, gibt die Funktion `false` zurück.

Syntax

```
IS_SMALLINT(super_expression)
```

Argumente

`super_expression`

Ein SUPER-Ausdruck oder eine Spalte.

Ergebnis

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der IS_SMALLINT-Funktion zu überprüfen, ob 5 ein SMALLINT ist.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_SMALLINT(s) FROM t;
```

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

Die Funktion IS_VARCHAR

Überprüft, ob eine Variable ein VARCHAR ist. Die Funktion IS_VARCHAR gibt für alle Zeichenfolgen `true` zurück. Für alle anderen Werte gibt die Funktion `false` zurück.

Die Funktion `IS_VARCHAR` ist eine Obermenge der Funktion `IS_CHAR`.

Syntax

```
IS_VARCHAR(super_expression)
```

Argumente

`super_expression`

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

BOOLEAN

Beispiel

Verwenden Sie das folgende Beispiel, um mit der `IS_VARCHAR`-Funktion zu überprüfen, ob `abc` ein VARCHAR ist.

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('abc');

SELECT s, IS_VARCHAR(s) FROM t;
```

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

Die Funktion JSON_TYPEOF

Die Skalarfunktion `JSON_TYPEOF` gibt einen VARCHAR mit den Werten „boolean“, „number“, „string“, „object“, „array“ oder „null“ zurück, abhängig vom dynamischen Typ des SUPER-Wertes.

Syntax

```
JSON_TYPEOF(super_expression)
```

Argumente

super_expression

Ein SUPER-Ausdruck oder eine Spalte.

Rückgabebetyp

VARCHAR

Beispiel

Verwenden Sie das folgende Beispiel, um den JSON-Typ für das Array [1, 2] zu überprüfen.

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

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

VARBYTE-Funktionen

AWS Clean Roomsunterstützt die folgenden VARBYTE-Funktionen.

Themen

- [Funktion FROM_HEX](#)
- [Funktion FROM_VARBYTE](#)
- [Funktion TO_HEX](#)
- [Funktion TO_VARBYTE](#)

Funktion FROM_HEX

FROM_HEX konvertiert einen Hexadezimalwert in einen Binärwert.

Syntax

```
FROM_HEX(hex_string)
```

Argumente

hex_string

Die hexadezimale Zeichenfolge des Datentyps VARCHAR oder TEXT, die konvertiert werden soll. Das Format muss ein Literalwert sein.

Rückgabebetyp

VARBYTE

Beispiel

Verwenden Sie das folgende Beispiel, um die hexadezimale Darstellung von '6162' in einen Binärwert zu konvertieren. Das Ergebnis wird automatisch als hexadezimale Darstellung des Binärwerts angezeigt.

```
SELECT FROM_HEX('6162');
```

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

Funktion FROM_VARBYTE

FROM_VARBYTE konvertiert einen Binärwert in eine Zeichenfolge im angegebenen Format.

Syntax

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

Argumente

binary_value

Ein Binärwert des Datentyps VARBYTE.

format

Das Format der zurückgegebenen Zeichenfolge. Gültige Werte, bei denen die Groß-/Kleinschreibung nicht beachtet wird, sind `hex`, `binary`, `utf-8` und `utf8`.

Rückgabebetyp

VARCHAR

Beispiel

Verwenden Sie das folgende Beispiel, um den Binärwert 'ab' in einen Hexadezimalwert zu konvertieren.

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

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

Funktion TO_HEX

TO_HEX konvertiert eine Zahl oder einen Binärwert in eine hexadezimale Darstellung.

Syntax

```
TO_HEX(value)
```

Argumente

Wert

Entweder eine Zahl oder ein Binärwert (VARBYTE), die/der konvertiert werden soll.

Rückgabebetyp

VARCHAR

Beispiel

Verwenden Sie das folgende Beispiel, um eine Zahl in ihre hexadezimale Darstellung zu konvertieren.

```
SELECT TO_HEX(2147676847);
```

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

```
| to_hex |
```

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

```
| 8002f2af |
```

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

Funktion TO_VARBYTE

TO_VARBYTE konvertiert eine Zeichenfolge in einem angegebenen Format in einen Binärwert.

Syntax

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

Argumente

string

Eine CHAR- oder VARCHAR-Zeichenfolge.

format

Das Format der Eingabezeichenfolge. Gültige Werte, bei denen die Groß-/Kleinschreibung nicht beachtet wird, sind hex, binary, utf-8 und utf8.

Rückgabebetyp

VARBYTE

Beispiel

Verwenden Sie das folgende Beispiel, um den Hexadezimalwert 6162 in einen Binärwert zu konvertieren. Das Ergebnis wird automatisch als hexadezimale Darstellung des Binärwerts angezeigt.

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

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

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-Klauseln 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 Rooms unterstützt keine Zeichenfolgeliterale 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_clause* und 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 Rooms unterstützt keine Zeichenfolgeliterale in ORDER BY-Klauseln.

Wenn die ORDER BY-Klausel ausgelassen wird, ist die Reihenfolge der Zeilen nicht deterministisch.

Note

Wenn in einem parallelen System wie AWS Clean Rooms eine ORDER BY-Klausel keine eindeutige und vollständige Anordnung der Daten erzeugt, ist die Reihenfolge der Zeilen nicht deterministisch. Das heißt, wenn der ORDER BY-Ausdruck duplizierte Werte erzeugt (eine partielle Anordnung), kann die Rückgabereihenfolge dieser Zeilen von einer Ausführung von AWS Clean Rooms zur 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, die 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 Ausführungen 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 Rooms unterstützt zwei Arten von Fensterfunktionen: Aggregat und Rangfestlegung.

Die folgenden Aggregationsfunktionen werden unterstützt:

- [Die Fensterfunktion AVG](#)
- [Die Fensterfunktion COUNT](#)
- [CUME_DIST-Fensterfunktion](#)
- [Die Fensterfunktion DENSE_RANK](#)
- [Die Fensterfunktion FIRST_VALUE](#)
- [Die Fensterfunktion LAG](#)
- [Die Fensterfunktion LAST_VALUE](#)
- [Die Fensterfunktion LEAD](#)
- [Die Fensterfunktion LISTAGG](#)
- [Die Fensterfunktion MAX](#)
- [Die Fensterfunktion MEDIAN](#)
- [Die Fensterfunktion MIN](#)
- [Die Fensterfunktion NTH_VALUE](#)
- [Fensterfunktion PERCENTILE_CONT](#)
- [Die Fensterfunktion PERCENTILE_DISC](#)
- [Die Fensterfunktion RATIO_TO_REPORT](#)
- [Die Fensterfunktionen STDDEV_SAMP und STDDEV_POP](#) (STDDEV_SAMP und STDDEV sind Synonyme)
- [Die Fensterfunktion SUM](#)
- [Die Fensterfunktionen VAR_SAMP und VAR_POP](#) (VAR_SAMP und VARIANCE sind Synonyme)

Die folgenden Rangfestlegungsfunktionen werden unterstützt:

- [Die Fensterfunktion DENSE_RANK](#)
- [Die Fensterfunktion NTILE](#)
- [Die Fensterfunktion PERCENT_RANK](#)
- [Die Fensterfunktion RANK](#)
- [Die Fensterfunktion ROW_NUMBER](#)

Beispieltabelle mit Beispielen von Fensterfunktionen

Zu jeder Funktionsbeschreibung gehören spezifische Fensterfunktionsbeispiele. Einige der Beispiele verwenden eine Tabelle mit dem Namen WINDSALES, die 11 Zeilen enthält, wie in der folgenden Tabelle gezeigt.

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	

Die Fensterfunktion AVG

Die AVG-Fensterfunktion gibt den Durchschnitt (das arithmetische Mittel) der Eingabeausdruckwerte zurück. Die Funktion AVG ist mit numerischen Werten kompatibel und ignoriert NULL-Werte.

Syntax

```
AVG ( [ALL ] expression ) OVER
(
```

```
[ PARTITION BY expr_list ]  
[ ORDER BY order_list  
           frame_clause ]  
)
```

Argumente

Ausdruck

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

ALL

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

OVER

Gibt die Fensterklauseln für die Aggregationsfunktionen an. Die OVER-Klausel unterscheidet Fensteraggregationsfunktionen von normalen Satzaggregationsfunktionen.

PARTITION BY *expr_list*

Definiert das Fenster für die AVG-Funktion in Bezug auf mindestens einen Ausdruck.

ORDER BY *order_list*

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle.

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 innerhalb des geordneten Ergebnisses. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe [Übersicht über die Syntax von Fensterfunktionen](#).

Datentypen

Die von der Funktion AVG unterstützten Argumenttypen sind SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL und DOUBLE PRECISION.

Die von der Funktion AVG unterstützten Rückgabetyper sind:

- BIGINT für SMALLINT- oder INTEGER-Argumente
- NUMERIC für BIGINT-Argumente
- DOUBLE PRECISION für Gleitkomma-Argumente

Beispiele

Im folgenden Beispiel wird ein gleitender Durchschnitt von verkauften Mengen nach Datum berechnet und die Ergebnisse nach Datums- und Verkaufs-ID geordnet:

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

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

(11 rows)

Eine Beschreibung der Tabelle WINSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

Die Fensterfunktion COUNT

Die Fensterfunktion COUNT zählt die durch den Ausdruck definierten Zeilen.

Die Funktion COUNT hat zwei 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.

Syntax

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

Argumente

Ausdruck

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

ALL

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

OVER

Gibt die Fensterklauseln für die Aggregationsfunktionen an. Die OVER-Klausel unterscheidet Fensteraggregationsfunktionen von normalen Satzaggregationsfunktionen.

PARTITION BY *expr_list*

Definiert das Fenster für die COUNT-Funktion in Bezug auf mindestens einen Ausdruck.

ORDER BY *order_list*

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle.

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 innerhalb des geordneten Ergebnisses. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe [Übersicht über die Syntax von Fensterfunktionen](#).

Datentypen

Die Funktion COUNT unterstützt alle Argumentdatentypen.

Der von der Funktion COUNT unterstützte Rückgabebetyp ist BIGINT.

Beispiele

Das folgende Beispiel zeigt die Verkaufs-ID, die Menge und die Zahl aller Zeilen ab dem Beginn des Datenfensters:

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

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

(11 rows)

Eine Beschreibung der Tabelle WINSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

Das folgende Beispiel zeigt die Verkaufs-ID, die Menge und die Zahl der Nicht-Null-Zeilen ab dem Beginn des Datenfensters an. (In der Tabelle WINSALES enthält die Spalte QTY_SHIPPED einige NULL-Werte.)

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

```
order by salesid;
```

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

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:

$$\text{count of rows with values } \leq x \text{ / count of rows in the window or partition}$$

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](#).

Rückgabebetyp

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 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](#).

Rückgabebetyp

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 `WINDSALES` 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 `WINDSALES` finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

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 enthalten AWS Clean Rooms soll. 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](#).

Rückgabebetyp

Diese Funktionen unterstützen Ausdrücke, die primitive AWS Clean Rooms Datentypen verwenden. Der Rückgabebetyp 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, venue_name,
first_value(venue_name)
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 | venue_name | 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 Spezifikation, die angibt, dass bei der Festlegung der zu verwendenden Zeile Nullwerte überspringen AWS Clean Rooms soll. 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 bei der Bestimmung der zu verwendenden Zeile Nullwerte enthalten AWS Clean Rooms soll. 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 Fensterfunktion LAG unterstützt Ausdrücke, die einen der AWS Clean Rooms Datentypen verwenden. Der Rückgabetyt 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)

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 enthalten AWS Clean Rooms soll. 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](#).

Rückgabebetyp

Diese Funktionen unterstützen Ausdrücke, die primitive AWS Clean Rooms Datentypen verwenden. Der Rückgabebetyp 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(venue_name)
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	venue_name	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 Spezifikation, die angibt, dass bei der Festlegung der zu verwendenden Zeile Nullwerte überspringen AWS Clean Rooms soll. 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 bei der Bestimmung der zu verwendenden Zeile Nullwerte enthalten AWS Clean Rooms soll. 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 Fensterfunktion LOAD unterstützt Ausdrücke, die einen der AWS Clean Rooms Datentypen verwenden. Der Rückgabetyt 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 LISTAGG

Die Fensterfunktion „LISTAGG“ ordnet die Zeilen der Gruppe in einer Abfrage nach dem ORDER BY-Ausdruck an. Anschließend werden die Werte zu einer einzigen Zeichenfolge verkettet.

LISTAGG ist eine reine Datenverarbeitungsknoten-Funktion. Die Funktion gibt einen Fehler zurück, wenn die Abfrage nicht auf eine benutzerdefinierte Tabelle oder AWS Clean Rooms Systemtabelle verweist.

Syntax

```

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

```

Argumente

DISTINCT

(Optional) Eine Klausel, die duplizierte Werte in dem angegebenen Ausdruck beseitigt, bevor die Verkettung vorgenommen wird. Leerzeichen am Ende werden ignoriert, sodass beispielsweise die Zeichenfolgen 'a' und 'a ' als duplizierte Werte behandelt werden würden. „LISTAGG“ verwendet den ersten registrierten Wert. Weitere Informationen finden Sie unter [Die Bedeutung von Leerzeichen am Ende](#).

aggregate_expression

Ein gültiger Ausdruck (beispielsweise ein Spaltenname), der die Werte bereitstellt, die aggregiert werden sollen. NULL-Werte und leere Zeichenfolgen werden ignoriert.

delimiter

(Optional) Die Zeichenfolgenkonstante, die die verketteten Werte trennt. Der Standardwert ist „NULL“.

AWS Clean Rooms unterstützt jede Menge vorangestellter oder nachgestellter Leerzeichen um ein optionales Komma oder einen Doppelpunkt sowie eine leere Zeichenfolge oder eine beliebige Anzahl von Leerzeichen.

Beispiele für gültige Werte sind:

" , "

" : "

" "

WITHIN GROUP (ORDER BY order_list)

(Optional) Eine Klausel, die die Sortierreihenfolge der aggregierten Werte angibt. Dies ist nur dann deterministisch, wenn ORDER BY eine spezifische Reihenfolge bereitstellt. Das Standardverhalten besteht darin, alle Zeilen zu aggregieren und einen einzelnen Wert zurückzugeben.

OVER

Eine Klausel, die die Fensterpartitionierung angibt. Die OVER-Klausel darf keine Spezifikation für Fensteranordnungen oder Fensterrahmen enthalten.

PARTITION BY partition_expression

(Optional) Legt den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel fest.

Rückgabewert

VARCHAR(MAX). Wenn der Ergebnissatz größer als die maximal zulässige Größe von VARCHAR ist (64.000 – 1 oder 65535), gibt LISTAGG den folgenden Fehler zurück:

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

Beispiele

Im folgenden Beispiel wird die Tabelle WINSALES verwendet. Eine Beschreibung der Tabelle WINSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

Im folgenden Beispiel wird eine Liste von Verkäufer-IDs zurückgegeben, geordnet nach Verkäufer-ID.

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

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

Im folgenden Beispiel wird eine Liste von Verkäufer-IDs für Verkäufer B zurückgegeben, geordnet nach Datum.

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

```
  seller
-----
    3233
    3233
    3233
    3233
(4 rows)
```

Im folgenden Beispiel wird eine durch Komma getrennte Liste von Verkaufsterminen für Käufer B zurückgegeben.

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

```

      dates
-----

```

```

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

```

```
(4 rows)
```

Im folgenden Beispiel wird mit „DISTINCT“ eine Liste von einzigartigen Verkaufsterminen für Käufer B zurückgegeben.

```

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

```

```

      dates
-----

```

```

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

```

```
(4 rows)
```

Im folgenden Beispiel wird eine durch Komma getrennte Liste von Verkaufs-IDs für die einzelnen Käufer-IDs zurückgegeben.

```

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

```

```

  buyerid | sales_id
-----+-----

```

```

a |10005,40001,40005
a |10005,40001,40005
a |10005,40001,40005

```

```
b |20001,30001,30004,30003
b |20001,30001,30004,30003
b |20001,30001,30004,30003
b |20001,30001,30004,30003
c |10001,20002,30007,10006
c |10001,20002,30007,10006
c |10001,20002,30007,10006
c |10001,20002,30007,10006
(11 rows)
```

Die Fensterfunktion MAX

Die Fensterfunktion MAX gibt den maximal zulässigen Wert der Eingabeausdruckswerte zurück. Die Funktion MAX ist mit numerischen Werten kompatibel und ignoriert NULL-Werte.

Syntax

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

Argumente

Ausdruck

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

ALL

Mit dem Argument ALL behält die Funktion alle duplizierten Werte aus dem Ausdruck bei. ALL ist das Standardargument. DISTINCT wird nicht unterstützt.

OVER

Eine Klausel, die die Fensterklauseln für die Aggregationsfunktionen angibt. Die OVER-Klausel unterscheidet Fensteraggregationsfunktionen von normalen Satzaggregationsfunktionen.

PARTITION BY *expr_list*

Definiert das Fenster für die MAX-Funktion in Bezug auf mindestens einen Ausdruck.

ORDER BY order_list

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle.

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 innerhalb des geordneten Ergebnisses. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe [Übersicht über die Syntax von Fensterfunktionen](#).

Datentypen

Akzeptiert alle Datentypen als Eingabe. Gibt denselben Datentyp wie expression zurück.

Beispiele

Das folgende Beispiel zeigt die Verkaufs-ID, die Menge und die maximale Menge ab dem Beginn des Datenfensters an:

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

```
salesid | qty | max  
-----+-----+-----  
10001 | 10 | 10  
10005 | 30 | 30  
10006 | 10 | 30  
20001 | 20 | 30  
20002 | 20 | 30  
30001 | 10 | 30  
30003 | 15 | 30  
30004 | 20 | 30  
30007 | 30 | 30  
40001 | 40 | 40  
40005 | 10 | 40  
(11 rows)
```

Eine Beschreibung der Tabelle WINDSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

Das folgende Beispiel zeigt die Verkaufs-ID, die Menge und die maximale Menge in einem eingeschränkten Rahmen an:

```
select salesid, qty,  
max(qty) over (order by salesid rows between 2 preceding and 1 preceding) as max  
from winsales  
order by salesid;
```

```
salesid | qty | max  
-----+-----+-----  
10001 | 10 |  
10005 | 30 | 10  
10006 | 10 | 30  
20001 | 20 | 30  
20002 | 20 | 20  
30001 | 10 | 20  
30003 | 15 | 20  
30004 | 20 | 15  
30007 | 30 | 20  
40001 | 40 | 30  
40005 | 10 | 40  
(11 rows)
```

Die Fensterfunktion MEDIAN

Berechnet den Medianwert für den Wertebereich in einem Fenster oder einer Partition. NULL-Werte im Bereich werden ignoriert.

MEDIAN ist eine Funktion für die inverse Verteilung, die ein kontinuierliches Verteilungsmodell annimmt.

MEDIAN ist eine reine Datenverarbeitungsknoten-Funktion. Die Funktion gibt einen Fehler zurück, wenn die Abfrage nicht auf eine benutzerdefinierte Tabelle oder AWS Clean Rooms Systemtabelle verweist.

Syntax

```
MEDIAN ( median_expression )
```

```
OVER ( [ PARTITION BY partition_expression ] )
```

Argumente

median_expression

Ein Ausdruck (beispielsweise ein Spaltenname), der die Werte bereitstellt, für die der Median ermittelt werden soll. Der Datentyp des Ausdrucks muss entweder numerisch oder Datum/Uhrzeit sein oder implizit in einen solchen konvertierbar sein.

OVER

Eine Klausel, die die Fensterpartitionierung angibt. Die OVER-Klausel darf keine Spezifikation für Fensteranordnungen oder Fensterrahmen enthalten.

PARTITION BY *partition_expression*

Optional. Ein Ausdruck, der den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel festlegt.

Datentypen

Der Rückgabetyt wird durch den Datentyp von *median_expression* festgelegt. Die folgende Tabelle zeigt den Rückgabetyt für jeden *median_expression*-Datentyp an.

Typ der Eingabe	Typ der Rückgabe
NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATUM	DATUM

Nutzungshinweise

Wenn das Argument *median_expression* den Datentyp DECIMAL hat und mit der maximal zulässigen Präzision von 38 Stellen definiert ist, gibt MEDIAN möglicherweise ein falsches Ergebnis oder einen Fehler zurück. Wenn der Rückgabewert der Funktion MEDIAN 38 Stellen überschreitet, wird das Ergebnis entsprechend abgekürzt. Dies führt zu einem Genauigkeitsverlust. Wenn während der Interpolierung ein Zwischenergebnis die maximal zulässige Genauigkeit überschreitet, erfolgt

ein numerischer Überlauf und die Funktion gibt einen Fehler zurück. Um diese Bedingungen zu vermeiden, werden die Verwendung eines Datentyps mit einer niedrigeren Genauigkeit oder die Umwandlung des Arguments `median_expression` in ein Argument mit niedrigerer Genauigkeit empfohlen.

Beispielsweise gibt eine SUM-Funktion mit einem DECIMAL-Argument standardmäßig eine Präzision von 38 Stellen zurück. Die Ergebnisskala ist die gleiche wie die Skala des Arguments. Eine SUM für eine DECIMAL(5,2)-Spalte gibt also einen DECIMAL(38,2)-Datentyp zurück.

Im folgenden Beispiel wird eine SUM-Funktion im Argument `median_expression` einer MEDIAN-Funktion verwendet. Der Datentyp der Spalte PRICEPAID ist DECIMAL(8,2). Daher gibt die SUM-Funktion DECIMAL(38,2) zurück.

```
select salesid, sum(pricepaid), median(sum(pricepaid))
over() from sales where salesid < 10 group by salesid;
```

Um einen möglichen Präzisionsverlust oder Overflow-Fehler zu vermeiden, wandeln Sie das Ergebnis in einen DECIMAL-Datentyp mit einer niedrigeren Präzision um, wie im folgenden Beispiel gezeigt.

```
select salesid, sum(pricepaid), median(sum(pricepaid)::decimal(30,2))
over() from sales where salesid < 10 group by salesid;
```

Beispiele

Im folgenden Beispiel wird der Median für die Verkaufsmenge für die einzelnen Verkäufer berechnet:

```
select sellerid, qty, median(qty)
over (partition by sellerid)
from winsales
order by sellerid;
```

```
sellerid qty median
```

```
-----
```

```
1  10 10.0
1  10 10.0
1  30 10.0
2  20 20.0
2  20 20.0
3  10 17.5
3  15 17.5
```

```
3 20 17.5
3 30 17.5
4 10 25.0
4 40 25.0
```

Eine Beschreibung der Tabelle WINSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

Die Fensterfunktion MIN

Die Fensterfunktion MIN gibt den mindestens erforderlichen Wert der Eingabeausdruckswerte zurück. Die Funktion MIN ist mit numerischen Werten kompatibel und ignoriert NULL-Werte.

Syntax

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

Argumente

Ausdruck

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

ALL

Mit dem Argument ALL behält die Funktion alle duplizierten Werte aus dem Ausdruck bei. ALL ist das Standardargument. DISTINCT wird nicht unterstützt.

OVER

Gibt die Fensterklauseln für die Aggregationsfunktionen an. Die OVER-Klausel unterscheidet Fensteraggregationsfunktionen von normalen Satzaggregationsfunktionen.

PARTITION BY *expr_list*

Definiert das Fenster für die MIN-Funktion in Bezug auf mindestens einen Ausdruck.

ORDER BY *order_list*

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle.

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 innerhalb des geordneten Ergebnisses. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe [Übersicht über die Syntax von Fensterfunktionen](#).

Datentypen

Akzeptiert alle Datentypen als Eingabe. Gibt denselben Datentyp wie expression zurück.

Beispiele

Das folgende Beispiel zeigt die Verkaufs-ID, die Menge und die Mindestmenge ab dem Beginn des Datenfensters an:

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

```
salesid | qty | min  
-----+-----+-----  
10001 | 10 | 10  
10005 | 30 | 10  
10006 | 10 | 10  
20001 | 20 | 10  
20002 | 20 | 10  
30001 | 10 | 10  
30003 | 15 | 10  
30004 | 20 | 10  
30007 | 30 | 10  
40001 | 40 | 10  
40005 | 10 | 10  
(11 rows)
```

Eine Beschreibung der Tabelle WINSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

Das folgende Beispiel zeigt die Verkaufs-ID, die Menge und die Mindestmenge in einem eingeschränkten Rahmen an:

```
select salesid, qty,
min(qty) over
(order by salesid rows between 2 preceding and 1 preceding) as min
from winsales
order by salesid;
```

```
salesid | qty | min
-----+-----+-----
10001 | 10 |
10005 | 30 | 10
10006 | 10 | 10
20001 | 20 | 10
20002 | 20 | 10
30001 | 10 | 20
30003 | 15 | 10
30004 | 20 | 10
30007 | 30 | 15
40001 | 40 | 20
40005 | 10 | 30
(11 rows)
```

Die Fensterfunktion NTH_VALUE

Die Fensterfunktion NTH_VALUE gibt den Ausdruckswert der angegebenen Zeile des Fensterrahmens in Bezug auf die erste Zeile des Fensters zurück.

Syntax

```
NTH_VALUE (expr, offset)
[ IGNORE NULLS | RESPECT NULLS ]
OVER
( [ PARTITION BY window_partition ]
[ ORDER BY window_ordering
           frame_clause ] )
```

Argumente

expr

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

offset

Legt die Zeilenzahl in Bezug auf die erste Zeile in dem Fenster zurück, für das der Ausdruck zurückgegeben werden soll. Beim Offset kann es sich um eine Konstante oder einen Ausdruck handeln. Es muss sich um eine positive Ganzzahl größer als 0 handeln.

IGNORE NULLS

Eine optionale Spezifikation, die angibt, dass bei der Festlegung der zu verwendenden Zeile Nullwerte überspringen AWS Clean Rooms soll. Wenn IGNORE NULLS nicht angegeben wird, werden Null-Werte berücksichtigt.

RESPECT NULLS

Gibt an, dass bei der Bestimmung der zu verwendenden Zeile Nullwerte enthalten AWS Clean Rooms soll. Wenn Sie IGNORE NULLS nicht angeben, wird RESPECT NULLS standardmäßig unterstützt.

OVER

Gibt die Fensterpartitionierung und -anordnung sowie den Fensterrahmen an.

PARTITION BY window_partition

Legt den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel fest.

ORDER BY window_ordering

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn die ORDER BY-Klausel ausgelassen wird, besteht der Rahmen standardmäßig aus allen Zeilen in der Partition.

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](#).

Die Fensterfunktion `NTH_VALUE` unterstützt Ausdrücke, die einen der AWS Clean Rooms Datentypen verwenden. Der Rückgabetyt ist mit dem Typ von `expr` identisch.

Beispiele

Im folgenden Beispiel wird die Anzahl der Sitzplätze der drittgrößten Veranstaltungsorte in Kalifornien, Florida, und New York gezeigt, verglichen mit der Anzahl der Sitzplätze der anderen Veranstaltungsorte in diesen Bundesstaaten:

```
select venuestate, venuename, venueseats,
nth_value(venueseats, 3)
ignore nulls
over(partition by venuestate order by venueseats desc
rows between unbounded preceding and unbounded following)
as third_most_seats
from (select * from venue where venueseats > 0 and
venuestate in('CA', 'FL', 'NY'))
order by venuestate;
```

venuestate	venuename	venueseats	third_most_seats
CA	Qualcomm Stadium	70561	63026
CA	Monster Park	69843	63026
CA	McAfee Coliseum	63026	63026
CA	Dodger Stadium	56000	63026
CA	Angel Stadium of Anaheim	45050	63026
CA	PETCO Park	42445	63026
CA	AT&T Park	41503	63026
CA	Shoreline Amphitheatre	22000	63026
FL	Dolphin Stadium	74916	65647
FL	Jacksonville Municipal Stadium	73800	65647
FL	Raymond James Stadium	65647	65647
FL	Tropicana Field	36048	65647
NY	Ralph Wilson Stadium	73967	20000
NY	Yankee Stadium	52325	20000
NY	Madison Square Garden	20000	20000

(15 rows)

Die Fensterfunktion NTILE

Die Fensterfunktion NTILE teilt angeordnete Zeilen in der Partition so gleichmäßig wie möglich in die angegebene Zahl von Gruppen mit Rangfestlegung auf und gibt die Gruppe zurück, zu der eine bestimmte Zeile gehört.

Syntax

```
NTILE (expr)  
OVER (  
  [ PARTITION BY expression_list ]  
  [ ORDER BY order_list ]  
)
```

Argumente

expr

Die Anzahl der Gruppen mit Rangfestlegung; muss für jede Partition einen positiven Ganzzahlwert (größer als 0) als Ergebnis haben. Das Argument *expr* darf nicht nullwertfähig sein.

OVER

Eine Klausel, die die Fensterpartitionierung und -anordnung angibt. Die OVER-Klausel darf keine Fensterrahmenspezifikation enthalten.

PARTITION BY *window_partition*

Optional. Der Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel.

ORDER BY *window_ordering*

Optional. Ein Ausdruck, der die Zeilen innerhalb der einzelnen Partitionen sortiert. Wenn die ORDER BY-Klausel ausgelassen wird, bleibt das Rangfestlegungsverhalten gleich.

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](#).

Rückgabebetyp

BIGINT

Beispiele

Im folgenden Beispiel wird der Preis, der am 26. August 2008 für Hamlet-Tickets gezahlt wurde, in vier Rangfestlegungsgruppen angezeigt. Das Ergebnis sind 17 Zeilen, die beinahe gleichmäßig auf die Rangfestlegungsgruppen 1 bis 4 aufgeteilt sind:

```
select eventname, caldate, pricepaid, ntile(4)
over(order by pricepaid desc) from sales, event, date
where sales.eventid=event.eventid and event.dateid=date.dateid and eventname='Hamlet'
and caldate='2008-08-26'
order by 4;
```

eventname	caldate	pricepaid	ntile
Hamlet	2008-08-26	1883.00	1
Hamlet	2008-08-26	1065.00	1
Hamlet	2008-08-26	589.00	1
Hamlet	2008-08-26	530.00	1
Hamlet	2008-08-26	472.00	1
Hamlet	2008-08-26	460.00	2
Hamlet	2008-08-26	355.00	2
Hamlet	2008-08-26	334.00	2
Hamlet	2008-08-26	296.00	2
Hamlet	2008-08-26	230.00	3
Hamlet	2008-08-26	216.00	3
Hamlet	2008-08-26	212.00	3
Hamlet	2008-08-26	106.00	3
Hamlet	2008-08-26	100.00	4
Hamlet	2008-08-26	94.00	4
Hamlet	2008-08-26	53.00	4
Hamlet	2008-08-26	25.00	4

(17 rows)

Die Fensterfunktion PERCENT_RANK

Berechnet den prozentualen Rang einer bestimmten Zeile. Der prozentuale Rang wird anhand der folgenden Formel festgelegt:

$$(x - 1) / (\text{the number of rows in the window or partition} - 1)$$

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](#).

Rückgabebetyp

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](#).

Fensterfunktion PERCENTILE_CONT

PERCENTILE_CONT ist eine Funktion für die inverse Verteilung, die ein kontinuierliches Verteilungsmodell annimmt. Sie empfängt einen Perzentilwert und eine Sortierspezifikation und gibt einen interpolierten Wert zurück, der in Bezug auf die Sortierspezifikation in den angegebenen Perzentilwert fällt.

PERCENTILE_CONT berechnet eine lineare Interpolierung zwischen Werten, nachdem diese der Reihenfolge entsprechend angeordnet wurden. Mithilfe des Perzentilwerts (P) und der Anzahl der Nicht-Null-Zeilen (N) in der Aggregationsgruppe berechnet die Funktion die Anzahl der Zeilen, nachdem die Zeilen entsprechend der Sortierspezifikation angeordnet wurden. Die Anzahl von Zeilen (RN) wird mit der Formel $RN = (1 + (P * (N - 1)))$ berechnet. Das Endergebnis der Aggregationsfunktion wird durch lineare Interpolierung zwischen den Werten aus Zeilen zwischen $CRN = \text{CEILING}(RN)$ und $FRN = \text{FLOOR}(RN)$ berechnet.

Das Ergebnis wird wie folgt aussehen.

Wenn ($CRN = FRN = RN$), ist das Ergebnis (value of expression from row at RN)

Andernfalls sieht das Ergebnis wie folgt aus:

$(CRN - RN) * (\text{value of expression for row at } FRN) + (RN - FRN) * (\text{value of expression for row at } CRN)$.

Sie können in der OVER-Klausel nur die PARTITION-Klausel angeben. Wenn PARTITION angegeben ist, gibt PERCENTILE_CONT für jede Zeile den Wert zurück, der in einem Satz von Werten innerhalb einer bestimmten Partition in das angegebene Perzentil fallen würde.

PERCENTILE_CONT ist eine reine Datenverarbeitungsknoten-Funktion. Die Funktion gibt einen Fehler zurück, wenn die Abfrage nicht auf eine benutzerdefinierte Tabelle oder AWS Clean Rooms Systemtabelle verweist.

Syntax

```
PERCENTILE_CONT ( percentile )  
WITHIN GROUP (ORDER BY expr)  
OVER ( [ PARTITION BY expr_list ] )
```

Argumente

percentile

Numerische Konstante zwischen 0 und 1. Null-Werte werden bei der Berechnung ignoriert.

WITHIN GROUP (ORDER BY *expr*)

Gibt numerische oder Datum-/Zeitwerte an, nach denen das Perzentil sortiert und berechnet werden soll.

OVER

Gibt die Fensterpartitionierung an. Die OVER-Klausel darf keine Spezifikation für Fensteranordnungen oder Fensterrahmen enthalten.

PARTITION BY expr

Optionales Argument, das den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel festlegt.

Rückgabewert

Der Rückgabewert wird durch den Datentyp des ORDER BY-Ausdrucks in der WITHIN GROUP-Klausel festgelegt. Die folgende Tabelle zeigt den Rückgabewert für jeden ORDER BY-Datentyp an.

Typ der Eingabe	Typ der Rückgabe
SMALLINT, INTEGER, BIGINT, NUMERISCH, DEZIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATUM	DATUM
TIMESTAMP	TIMESTAMP

Nutzungshinweise

Wenn das Argument ORDER BY den Datentyp DECIMAL hat und mit der maximal zulässigen Präzision von 38 Stellen definiert ist, gibt PERCENTILE_CONT möglicherweise ein falsches Ergebnis oder einen Fehler zurück. Wenn der Rückgabewert der Funktion PERCENTILE_CONT 38 Stellen überschreitet, wird das Ergebnis entsprechend abgekürzt. Dies führt zu einem Genauigkeitsverlust. Wenn während der Interpolierung ein Zwischenergebnis die maximal zulässige Genauigkeit überschreitet, erfolgt ein numerischer Überlauf und die Funktion gibt einen Fehler zurück. Um diese Bedingungen zu vermeiden, werden die Verwendung eines Datentyps mit einer niedrigeren Genauigkeit oder die Umwandlung des Ausdrucks ORDER BY in einen Ausdruck mit niedrigerer Genauigkeit empfohlen.

Beispielsweise gibt eine SUM-Funktion mit einem DECIMAL-Argument standardmäßig eine Präzision von 38 Stellen zurück. Die Ergebnisskala ist die gleiche wie die Skala des Arguments. Eine SUM für eine DECIMAL(5,2)-Spalte gibt also einen DECIMAL(38,2)-Datentyp zurück.

Im folgenden Beispiel wird in der ORDER BY-Klausel einer PERCENTILE_CONT-Funktion eine SUM-Funktion verwendet. Der Datentyp der Spalte PRICEPAID ist DECIMAL(8,2). Daher gibt die SUM-Funktion DECIMAL(38,2) zurück.

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid) desc) over()
from sales where salesid < 10 group by salesid;
```

Um einen möglichen Präzisionsverlust oder Overflow-Fehler zu vermeiden, wandeln Sie das Ergebnis in einen DECIMAL-Datentyp mit einer niedrigeren Präzision um, wie im folgenden Beispiel gezeigt.

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid)::decimal(30,2) desc) over()
from sales where salesid < 10 group by salesid;
```

Beispiele

Im folgenden Beispiel wird die Tabelle WINDSALES verwendet. Eine Beschreibung der Tabelle WINDSALES finden Sie unter [Beispieltable mit Beispielen von Fensterfunktionen](#).

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over() as median from winsales;
```

sellerid	qty	median
1	10	20.0
1	10	20.0
3	10	20.0
4	10	20.0
3	15	20.0
2	20	20.0
3	20	20.0
2	20	20.0
3	30	20.0
1	30	20.0


```

      4 | 40 | 20.0
(11 rows)

```

```

select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

```

```

sellerid | qty | median
-----+-----+-----
      2 | 20 | 20.0
      2 | 20 | 20.0
      4 | 10 | 25.0
      4 | 40 | 25.0
      1 | 10 | 10.0
      1 | 10 | 10.0
      1 | 30 | 10.0
      3 | 10 | 17.5
      3 | 15 | 17.5
      3 | 20 | 17.5
      3 | 30 | 17.5
(11 rows)

```

Im folgenden Beispiel wird der Wert für PERCENTILE_CONT und PERCENTILE_DISC für Ticketverkäufe von Verkäufern im Bundesstaat Washington berechnet.

```

SELECT sellerid, state, sum(qtysold*pricepaid) sales,
percentile_cont(0.6) within group (order by sum(qtysold*pricepaid)::decimal(14,2) )
desc) over(),
percentile_disc(0.6) within group (order by sum(qtysold*pricepaid)::decimal(14,2) )
desc) over()
from sales s, users u
where s.sellerid = u.userid and state = 'WA' and sellerid < 1000
group by sellerid, state;

```

```

sellerid | state | sales | percentile_cont | percentile_disc
-----+-----+-----+-----+-----
    127 | WA   | 6076.00 | 2044.20 | 1531.00
    787 | WA   | 6035.00 | 2044.20 | 1531.00
    381 | WA   | 5881.00 | 2044.20 | 1531.00
    777 | WA   | 2814.00 | 2044.20 | 1531.00
     33 | WA   | 1531.00 | 2044.20 | 1531.00
    800 | WA   | 1476.00 | 2044.20 | 1531.00
     1  | WA   | 1177.00 | 2044.20 | 1531.00

```

(7 rows)

Die Fensterfunktion PERCENTILE_DISC

PERCENTILE_DISC ist eine Funktion für die inverse Verteilung, die ein diskretes Verteilungsmodell annimmt. Sie empfängt einen Perzentilwert und eine Sortierspezifikation und gibt ein Element aus dem angegebenen Satz zurück.

PERCENTILE_DISC sortiert für den Perzentilwert P die Werte des Ausdrucks in der ORDER BY-Klausel und gibt den Wert mit dem kleinsten kumulativen Verteilungswert (in Bezug auf dieselbe Sortierspezifikation) zurück, der größer als oder gleich P ist.

Sie können in der OVER-Klausel nur die PARTITION-Klausel angeben.

PERCENTILE_DISC ist eine reine Datenverarbeitungsknoten-Funktion. Die Funktion gibt einen Fehler zurück, wenn die Abfrage nicht auf eine benutzerdefinierte Tabelle oder AWS Clean Rooms Systemtabelle verweist.

Syntax

```
PERCENTILE_DISC ( percentile )  
WITHIN GROUP (ORDER BY expr)  
OVER ( [ PARTITION BY expr_list ] )
```

Argumente

percentile

Numerische Konstante zwischen 0 und 1. Null-Werte werden bei der Berechnung ignoriert.

WITHIN GROUP (ORDER BY *expr*)

Gibt numerische oder Datum-/Zeitwerte an, nach denen das Perzentil sortiert und berechnet werden soll.

OVER

Gibt die Fensterpartitionierung an. Die OVER-Klausel darf keine Spezifikation für Fensteranordnungen oder Fensterrahmen enthalten.

PARTITION BY *expr*

Optionales Argument, das den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel festlegt.

Rückgabewert

Derselbe Datentyp wie der ORDER BY-Ausdruck in der WITHIN GROUP-Klausel.

Beispiele

Im folgenden Beispiel wird die Tabelle WINDSALES verwendet. Eine Beschreibung der Tabelle WINDSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over() as median from winsales;
```

sellerid	qty	median
1	10	20
3	10	20
1	10	20
4	10	20
3	15	20
2	20	20
2	20	20
3	20	20
1	30	20
3	30	20
4	40	20

(11 rows)

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;
```

sellerid	qty	median
2	20	20
2	20	20
4	10	10
4	40	10
1	10	10
1	10	10
1	30	10
3	10	15
3	15	15

```
      3 | 20 | 15
      3 | 30 | 15
(11 rows)
```

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 Rangfestlegungskriterien erhalten denselben Rang. AWS Clean Rooms fügt die Anzahl der gebundenen Zeilen zum gebundenen Rang hinzu, um den nächsten Rang zu berechnen, und daher sind die Ränge möglicherweise keine aufeinanderfolgenden Zahlen. 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](#).

Rückgabebetyp

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 zurückgibt. Die Zeilen mit den Verkaufs-IDs 10001 und 10006 besitzen beispielsweise identische Werte für QTY und RNK. 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 RATIO_TO_REPORT

Berechnet das Verhältnis eines Werts zur Summe der Werte in einem Fenster oder einer Partition. Der RATIO_TO_REPORT-Wert wird anhand der folgenden Formel festgelegt:

$$\text{value of ratio_expression argument for the current row} / \text{sum of ratio_expression argument for the window or partition}$$

Der folgende Datensatz zeigt die Verwendung dieser Formel:

```

Row# Value Calculation RATIO_TO_REPORT
1 2500 (2500)/(13900) 0.1798
2 2600 (2600)/(13900) 0.1870
3 2800 (2800)/(13900) 0.2014
4 2900 (2900)/(13900) 0.2086
5 3100 (3100)/(13900) 0.2230

```

Der Rückgabewertbereich ist 0 bis 1 (einschließlich). Wenn ratio_expression NULL ist, dann ist der Rückgabewert NULL.

Syntax

```
RATIO_TO_REPORT ( ratio_expression )
```

```
OVER ( [ PARTITION BY partition_expression ] )
```

Argumente

ratio_expression

Ein Ausdruck (beispielsweise ein Spaltenname), der den Wert bereitstellt, für den das Verhältnis ermittelt werden soll. Der Datentyp des Ausdrucks muss entweder numerisch sein oder implizit in einen solchen konvertierbar sein.

Sie können in ratio_expression keine anderen analytischen Funktionen verwenden.

OVER

Eine Klausel, die die Fensterpartitionierung angibt. Die OVER-Klausel darf keine Spezifikation für Fensteranordnungen oder Fensterrahmen enthalten.

PARTITION BY partition_expression

Optional. Ein Ausdruck, der den Datensatzbereich für die einzelnen Gruppen in der OVER-Klausel festlegt.

Rückgabebetyp

FLOAT8

Beispiele

Im folgenden Beispiel werden die Verhältnisse der Verkaufsmengen für die einzelnen Verkäufer berechnet:

```
select sellerid, qty, ratio_to_report(qty)
over (partition by sellerid)
from winsales;
```

```
sellerid qty  ratio_to_report
-----
```

```
2  20.12312341    0.5
2  20.08630000    0.5
4  10.12414400    0.2
4  40.23000000    0.8
1  30.37262000    0.6
1  10.64000000    0.21
```



```
1 10.00000000 0.2
3 10.03500000 0.13
3 15.14660000 0.2
3 30.54790000 0.4
3 20.74630000 0.27
```

Eine Beschreibung der Tabelle WINSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

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 Zeilennummern.

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](#).

Rückgabebetyp

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](#).

Die Fensterfunktionen STDDEV_SAMP und STDDEV_POP

Die Fensterfunktionen STDDEV_SAMP und STDDEV_POP geben die Stichproben- und Populationsstandardabweichungen eines Satzes numerischer Werte (integer, decimal oder floating-point) zurück. Weitere Informationen finden Sie auch unter [Die Funktionen STDDEV_SAMP und STDDEV_POP](#).

STDDEV_SAMP und STDDEV sind Synonyme für dieselbe Funktion.

Syntax

```
STDDEV_SAMP | STDDEV | STDDEV_POP  
( [ ALL ] expression ) OVER  
(  
[ PARTITION BY expr_list ]  
[ ORDER BY order_list  
                frame_clause ]  
)
```

Argumente

Ausdruck

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

ALL

Mit dem Argument ALL behält die Funktion alle duplizierten Werte aus dem Ausdruck bei. ALL ist das Standardargument. DISTINCT wird nicht unterstützt.

OVER

Gibt die Fensterklauseln für die Aggregationsfunktionen an. Die OVER-Klausel unterscheidet Fensteraggregationsfunktionen von normalen Satzaggregationsfunktionen.

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 PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle.

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 innerhalb des geordneten Ergebnisses. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe [Übersicht über die Syntax von Fensterfunktionen](#).

Datentypen

Die von den STDDEV-Funktion SUM unterstützten Argumenttypen sind SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL und DOUBLE PRECISION.

Unabhängig vom Datentyp des Ausdrucks ist der Rückgabewert einer STDDEV-Funktion eine DOUBLE PRECISION-Zahl.

Beispiele

Das folgende Beispiel zeigt, wie die Funktionen STDDEV_POP und VAR_POP als Fensterfunktionen verwendet werden. Die Abfrage berechnet Populationsvarianz und Populationsstandardabweichung für PRICEPAID-Werte in der Tabelle SALES.

```
select salesid, dateid, pricepaid,
round(stddev_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as stddevpop,
round(var_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as varpop
from sales
order by 2,1;
```

salesid	dateid	pricepaid	stddevpop	varpop
33095	1827	234.00	0	0
65082	1827	472.00	119	14161
88268	1827	836.00	248	61283
97197	1827	708.00	230	53019
110328	1827	347.00	223	49845
110917	1827	337.00	215	46159
150314	1827	688.00	211	44414
157751	1827	1730.00	447	199679
165890	1827	4192.00	1185	1403323

...

Die Funktionen für Stichprobenstandardabweichung und -varianz können auf die gleiche Weise verwendet werden.

Die Fensterfunktion SUM

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

Syntax

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

Argumente

Ausdruck

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

ALL

Mit dem Argument ALL behält die Funktion alle duplizierten Werte aus dem Ausdruck bei. ALL ist das Standardargument. DISTINCT wird nicht unterstützt.

OVER

Gibt die Fensterklauseln für die Aggregationsfunktionen an. Die OVER-Klausel unterscheidet Fensteraggregationsfunktionen von normalen Satzaggregationsfunktionen.

PARTITION BY *expr_list*

Definiert das Fenster für die SUM-Funktion in Bezug auf mindestens einen Ausdruck.

ORDER BY *order_list*

Sortiert die Zeilen innerhalb der einzelnen Partitionen. Wenn PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle.

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 innerhalb des geordneten Ergebnisses. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe [Übersicht über die Syntax von Fensterfunktionen](#).

Datentypen

Die von der Funktion SUM unterstützten Argumenttypen sind SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL und DOUBLE PRECISION.

Die von der Funktion SUM unterstützten Rückgabetypen sind:

- BIGINT für SMALLINT- oder INTEGER-Argumente
- NUMERIC für BIGINT-Argumente
- DOUBLE PRECISION für Gleitkomma-Argumente

Beispiele

Im folgenden Beispiel wird eine kumulative (gleitende) Summe von Verkaufsmengen nach Datum und Verkaufs-ID erstellt:

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

salesid	dateid	sellerid	qty	sum
30001	2003-08-02	3	10	10
10001	2003-12-24	1	10	20
10005	2003-12-24	1	30	50
40001	2004-01-09	4	40	90
10006	2004-01-18	1	10	100
20001	2004-02-12	2	20	120
40005	2004-02-12	4	10	130
20002	2004-02-16	2	20	150
30003	2004-04-18	3	15	165

```
30004 | 2004-04-18 |      3 | 20 | 185
30007 | 2004-09-07 |      3 | 30 | 215
(11 rows)
```

Eine Beschreibung der Tabelle WINSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

Im folgenden Beispiel wird eine kumulative (gleitende) Summe von Verkaufsmengen nach Datum erstellt, die Ergebnisse nach Verkäufer-ID partitioniert und die Ergebnisse innerhalb der Partition nach Datum und Verkaufs-ID geordnet:

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

```
salesid | dateid | sellerid | qty | sum
-----+-----+-----+----+----
30001 | 2003-08-02 |      3 | 10 | 10
10001 | 2003-12-24 |      1 | 10 | 10
10005 | 2003-12-24 |      1 | 30 | 40
40001 | 2004-01-09 |      4 | 40 | 40
10006 | 2004-01-18 |      1 | 10 | 50
20001 | 2004-02-12 |      2 | 20 | 20
40005 | 2004-02-12 |      4 | 10 | 50
20002 | 2004-02-16 |      2 | 20 | 40
30003 | 2004-04-18 |      3 | 15 | 25
30004 | 2004-04-18 |      3 | 20 | 45
30007 | 2004-09-07 |      3 | 30 | 75
(11 rows)
```

Im folgenden Beispiel werden alle Zeilen im Ergebnissatz sequenziell nummeriert, geordnet nach den Spalten SELLERID und SALESID:

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

```
salesid | sellerid | qty | rownum
-----+-----+----+-----
```

```

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

```

Eine Beschreibung der Tabelle WINSALES finden Sie unter [Beispieltabelle mit Beispielen von Fensterfunktionen](#).

Im folgenden Beispiel werden alle Zeilen im Ergebnissatz sequenziell nummeriert, die Ergebnisse nach SELLERID partitioniert und die Ergebnisse innerhalb der Partition nach SELLERID und SALESID geordnet:

```

select salesid, sellerid, qty,
sum(1) over (partition by sellerid
order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

```

```

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

```


Die Fensterfunktionen VAR_SAMP und VAR_POP

Die Fensterfunktionen VAR_SAMP und VAR_POP geben die Stichproben- und Populationsabweichung eines Satzes numerischer Werte (integer, decimal oder floating-point) zurück. Weitere Informationen finden Sie auch unter [Die Funktionen VAR_SAMP und VAR_POP](#).

VAR_SAMP und VARIANCE sind Synonyme für dieselbe Funktion.

Syntax

```
VAR_SAMP | VARIANCE | VAR_POP  
( [ ALL ] expression ) OVER  
(  
  [ PARTITION BY expr_list ]  
  [ ORDER BY order_list  
                                frame_clause ]  
)
```

Argumente

Ausdruck

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

ALL

Mit dem Argument ALL behält die Funktion alle duplizierten Werte aus dem Ausdruck bei. ALL ist das Standardargument. DISTINCT wird nicht unterstützt.

OVER

Gibt die Fensterklauseln für die Aggregationsfunktionen an. Die OVER-Klausel unterscheidet Fensteraggregationsfunktionen von normalen Satzaggregationsfunktionen.

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 PARTITION BY nicht angegeben ist, verwendet ORDER BY die gesamte Tabelle.

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 innerhalb des geordneten Ergebnisses. Die Rahmenklausel besteht aus dem Schlüsselwort ROWS und verknüpften Spezifikatoren. Siehe [Übersicht über die Syntax von Fensterfunktionen](#).

Datentypen

Die von den VARIANCE-Funktion SUM unterstützten Argumenttypen sind SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL und DOUBLE PRECISION.

Unabhängig vom Datentyp des Ausdrucks ist der Rückgabewert einer VARIANCE-Funktion eine DOUBLE PRECISION-Zahl.

SQL-Bedingungen in AWS Clean Rooms

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.

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 unterstützt AWS Clean Rooms.

Themen

- [Vergleichsbedingungen](#)
- [Logische Bedingungen](#)
- [Patternmatching-Bedingungen](#)
- [BETWEEN-Bereichsbedingung](#)
- [„Null“-Bedingung](#)
- [EXISTS-Bedingung](#)
- [IN-Bedingung](#)
- [Syntax](#)

Vergleichsbedingungen

Vergleichsbedingungen machen eine Aussage bezüglich der logischen Beziehungen zwischen zwei Werten. Alle Vergleichsbedingungen sind binäre Operatoren mit einem booleschen Rückgabetyp. AWS Clean Rooms unterstützt die in der folgenden Tabelle beschriebenen Vergleichsoperatoren.

Operator	Syntax	Beschreibung
<	a < b	Wertaist kleiner als Wertb.
>	a > b	Wertaist größer als Wertb.
<=	a <= b	Wertaist kleiner als oder gleich dem Wertb.
>=	a >= b	Wertaist größer als oder gleich dem Wertb.
=	a = b	Wertaentspricht dem Wertb.
<> oder !=	a <> b or a != b	Wertaist nicht gleich Wertb.
a = TRUE	a IS TRUE	Wertaist BooleanTRUE.

Nutzungshinweise

= ANY | SOME

Die Schlüsselwörter ANY und SOME sind gleichbedeutend mit IN Zustand. Die Schlüsselwörter ANY und SOME geben True zurück, wenn der Vergleich für mindestens einen Wert zutrifft, der von einer Unterabfrage zurückgegeben wird, die einen oder mehrere Werte zurückgibt. AWS Clean Rooms unterstützt nur die Bedingung = (entspricht) für ANY und SOME. Ungleichheitsbedingungen werden nicht unterstützt.

Note

Das Prädikat ALL wird nicht unterstützt.

<> ALL

Das Schlüsselwort ALL ist synonym mit NOT IN (vgl. [IN-Bedingung](#)-Bedingung) und gibt „wahr“ zurück, wenn der Ausdruck nicht in den Ergebnissen der Unterabfrage enthalten ist. AWS Clean Rooms unterstützt nur die <> oder != (nicht gleich)-Bedingung für ALL. Andere Vergleichsbedingungen werden nicht unterstützt.

IS TRUE/FALSE/UNKNOWN

Werte ungleich 0 werden zu TRUE ausgewertet, 0 zu FALSE, und „Null“ zu UNKNOWN. Siehe Datentyp [Typ BOOLEAN](#).

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 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 Zeitlitterale 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`-Litterale 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	

E1	E2	E1 AND E2	E1 OR E2	NOT E2
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 | TEY680EB
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 Bedingungs Ausdruck angegebenen Muster und gibt TRUE oder FALSE zurück, je nachdem, ob er eine Übereinstimmung findet. AWS Clean Rooms 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. Bei LIKE erfolgt diese Suche unter Berücksichtigung der Groß-/Kleinschreibung, bei ILIKE wird die Groß-/Kleinschreibung ignoriert.

- Reguläre Ausdrücke mit SIMILAR TO

Der Operator SIMILAR TO führt das Patternmatching unter Verwendung von regulären Ausdrücken entsprechend dem Standardformat für SQL durch. Als Metazeichen werden dabei beiden Zeichen unterstützt, die auch der Operator LIKE unterstützt. Bei SIMILAR TO erfolgt die Suche über die gesamte Zeichenfolge und unter Berücksichtigung der Groß-/Kleinschreibung.

Themen

- [LIKE](#)
- [SIMILAR TO](#)

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.

Bei LIKE wird die Groß-/Kleinschreibung berücksichtigt, bei ILIKE nicht.

Syntax

```
expression [ NOT ] LIKE | ILIKE pattern [ ESCAPE 'escape_char' ]
```

Argumente

expression

Ein gültiger UTF8-Zeichenfolgenausdruck (beispielsweise ein Spaltenname).

LIKE | ILIKE

Bei LIKE wird beim Patternmatching die Groß-/Kleinschreibung berücksichtigt. Bei ILIKE wird beim Patternmatching die Groß-/Kleinschreibung nicht berücksichtigt, wenn die Zeichenfolge aus UTF-8 (ASCII)-Zeichen besteht. 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- und ILIKE-Prädikate nicht implizit nachfolgende Leerzeichen. Um nachfolgende Leerzeichen zu ignorieren, verwenden Sie RTRIM, oder konvertieren Sie eine CHAR-Spalte explizit zu VARCHAR.

Der Operator ~~ entspricht LIKE und ~~* entspricht ILIKE. Die Operatoren !~~ und !~~* entsprechen außerdem NOT LIKE und NOT ILIKE.

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%'	Wahr

Ausdruck	Rückgabewert
'abc' LIKE '_B_'	Falsch
'abc' ILIKE '_B_'	Wahr
'abc' LIKE 'c%'	Falsch

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 listet alle Städte auf, deren dritter und vierter Buchstabe die Folge „ea“ ist: In dem Befehl wird ILIKE verwendet, um zu zeigen, dass bei ILIKE die Groß-/Kleinschreibung nicht berücksichtigt wird:

```
select distinct city from users where city ilike '__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)
```

Im folgenden Beispiel wird der Operator `~~*` verwendet, um eine Suche ohne Berücksichtigung der Groß- und Kleinschreibung (ILIKE) nach Städten durchzuführen, die mit „Ag“ beginnen.

```
select distinct city from users where city ~~* 'Ag%' order by city;
```

```
city
-----
Agat
Agawam
Agoura Hills
Aguadilla
```

SIMILAR TO

Der SIMILAR TO-Operator vergleicht einen Zeichenfolgenausdruck (beispielsweise einen Spaltennamen) mit einem regulären Ausdruck entsprechend dem Standardformat für SQL. Ein regulärer Ausdruck entsprechend dem Standardformat für SQL kann verschiedene Metazeichen zum Patternmatching enthalten, darunter die beiden Zeichen, die auch der Operator [LIKE](#) unterstützt.

Der Operator SIMILAR TO gibt „wahr“ genau dann zurück, wenn das Muster der gesamten Zeichenfolge entspricht. Bei regulären Ausdrücken entsprechend dem POSIX-Standard hingegen kann das Muster einer beliebigen Teilzeichenfolge entsprechen.

Bei SIMILAR TO wird beim Patternmatching die Groß-/Kleinschreibung berücksichtigt.

Note

Patternmatching mit regulären Ausdrücken unter Verwendung von SIMILAR TO erfordert einen hohen Rechenaufwand. Wir empfehlen, nach Möglichkeit LIKE zu verwenden, insbesondere, wenn eine große Anzahl an Zeilen verarbeitet werden muss. Ein Beispiel: Die folgenden Abfragen sind funktional identisch, aber die Abfrage mit LIKE wird 7-mal schneller ausgeführt als die Abfrage mit einem regulären Ausdruck:

```
select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die%';
```


Syntax

```
expression [ NOT ] SIMILAR TO pattern [ ESCAPE 'escape_char' ]
```

Argumente

expression

Ein gültiger UTF8-Zeichenfolgenausdruck (beispielsweise ein Spaltenname).

SIMILAR TO

Bei SIMILAR TO erfolgt die Suche über die gesamte Zeichenfolge in dem Ausdruck *expression* und unter Berücksichtigung der Groß-/Kleinschreibung.

pattern

Ein gültiger UTF8-Zeichenfolgenausdruck, der einen regulären Ausdruck entsprechend dem SQL-Standard darstellt.

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, wird das Muster als die Zeichenfolge selbst interpretiert.

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*.

SIMILAR TO unterstützt die folgenden Metazeichen in Mustern:

Operator	Beschreibung
%	Entspricht einer Folge von 0 oder mehr Zeichen.
–	Entspricht einem beliebigen Zeichen.
	Alternativen (eines der beiden angegebenen Teilmuster).

Operator	Beschreibung
*	Wiederholt das vorangehende Element 0 oder mehr Male.
+	Wiederholt das vorangehende Element 1 oder mehr Male.
?	Wiederholt das vorangehende Element 0 oder 1 Mal.
{m}	Wiederholt das vorangehende Element genau m Male.
{m, }	Wiederholt das vorangehende Element m oder mehr Male.
{m, n}	Wiederholt das vorangehende Element m bis n Male.
()	Klammern fassen Elemente zu einem logischen Element zusammen.
[...]	Ausdrücke in eckigen Klammern geben eine Zeichenklasse an, wie bei regulären Ausdrücken entsprechend dem POSIX-Standard.

Beispiele

In der folgenden Tabelle werden Beispiele für Patternmatching mit SIMILAR TO dargestellt.

Ausdruck	Rückgabewert
'abc' SIMILAR TO 'abc'	Wahr
'abc' SIMILAR TO '_b_'	Wahr
'abc' SIMILAR TO '_A_'	Falsch
'abc' SIMILAR TO '%(b d)%'	Wahr
'abc' SIMILAR TO '(b c)%'	Falsch
'AbcAbcdefgfg12efgfg12' SIMILAR TO '((Ab)?c)+d((efg)+(12))+'	Wahr
'aaaaaab11111xy' SIMILAR TO 'a{6}_[0-9]{5}(x y){2}'	Wahr

Ausdruck	Rückgabewert
'\$0.87' SIMILAR TO '\$[0-9]+(.[0-9][0-9])?'	Wahr

Das folgende Beispiel listet Städte auf, deren Name ein „E“ oder ein „H“ enthält:

```
SELECT DISTINCT city FROM users
WHERE city SIMILAR TO '%E|%H%' ORDER BY city LIMIT 5;
```

```

      city
-----
Agoura Hills
Auburn Hills
Benton Harbor
Beverly Hills
Chicago Heights
```

In dem folgenden Beispiel wird die Standard-Escape-Zeichenfolge (\\) verwendet, um nach Zeichenfolgen zu suchen, die den Unterstrich (_) enthalten.

```
SELECT tablename, "column" FROM my_table_def
WHERE "column" SIMILAR TO '%start\\_%'

ORDER BY tablename, "column" LIMIT 5;
```

```

      tablename      |      column
-----+-----
my_abort_idle      | idle_start_time
my_abort_idle      | txn_start_time
my_analyze_compression | start_time
my_auto_worker_levels | start_level
my_auto_worker_levels | start_wlm_occupancy
```

In dem folgenden Beispiel wird als Escape-Zeichenfolge ^ verwendet, und dann wird nach Zeichenfolgen gesucht, die den Unterstrich (_) enthalten.

```
SELECT tablename, "column" FROM my_table_def

WHERE "column" SIMILAR TO '%start^_%' ESCAPE '^'
```

```
ORDER BY tablename, "column" LIMIT 5;
```

tablename	column
stcs_abort_idle	idle_start_time
stcs_abort_idle	txn_start_time
stcs_analyze_compression	start_time
stcs_auto_worker_levels	start_level
stcs_auto_worker_levels	start_wlm_occupancy

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 (inklusive 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, venuesseats from venue
where venuesseats between 20000 and 50000
order by venuesseats desc;

venueid |          venuename          | venuesseats
-----+-----+-----
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

Der NULLKonditionstests 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 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

EinINCondition testet einen Wert auf Mitgliedschaft in einer Gruppe von Werten oder in 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)
```

Syntax

```
comparison_condition  
| logical_condition  
| range_condition  
| pattern_matching_condition  
| null_condition  
| EXISTS_condition  
| IN_condition
```

Verschachtelte Daten abfragen

AWS Clean Rooms bietet SQL-kompatiblen Zugriff auf relationale und verschachtelte Daten.

AWS Clean Rooms verwendet die gepunktete Notation und den Array-Index für die Pfadnavigation, wenn auf verschachtelte Daten zugegriffen wird. Es ermöglicht auch FROM-Aufheben der Verschachtelung, die Array-Überheben der Verschachtelung, Unnest und Verschachtelung, Unnest. Die folgenden Themen beschreiben die verschiedenen Abfragemuster, die die Verwendung des Array-/struct-/map-Datentyps mit Pfad- und Array-Navigation, Aufheben der Verschachtelung, Aufheben der Verschachtelung, Aufheben der Verschachtelung, Aufheben der Verschachtelung, Aufheben der Verschachtelung,

Themen

- [Navigation](#)
- [Aufheben der Verschachtelung von Abfragen](#)
- [Lax-Semantik](#)
- [Arten der Introspektion](#)

Navigation

AWS Clean Rooms ermöglicht die Navigation in Arrays und Strukturen mithilfe von [. . .] Notation in Klammern bzw. Punkten. Darüber hinaus können Sie die Navigation mithilfe von Punktschreibweise und Arrays mithilfe der Klammernotation in Strukturen mischen.

Example

Die folgende Beispielabfrage geht beispielsweise davon aus, dass `c_orders` Array-Datenspalte ist ein Array mit einer Struktur und einem Attribut, das benannt `orderkey`.

```
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

Um Abfragen zu entwirren, AWS Clean Room ermöglicht die Iteration über Arrays. 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`. Ähnlich verhält es sich mit der Syntax für das Aufheben von Verschachtelungen mithilfe des FROM-Klausellements `x (AS) y` bedeutet, dass `y` iteriert über jeden Wert im Array-Ausdruck `x`. In diesem Fall ist ein 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` ist die Iteration über `customer_order_lineitem` Basistabelle
- `c.c_orders` 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, wenn die Verwendung des Array-Index mit `AT` Schlüsselwort. Die Klausel `x AS y AT z` iteriert über ein Array `x` und generiert das Feld `z`, das ist der Array-Index.

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. Der `f.multi_level_array` ASArray iteriert über `multi_level_array`. Die Verschachtelung `ASElement` ist die Iteration über die `Array-Array-Array-Array-Abschnitt` `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 werden Navigationsvorgänge mit verschachtelten Datenwerten den Wert Null, anstatt einen Fehler, wenn die Navigation ungültig ist. Die Objektnavigation ist ungültig, wenn der verschachtelte Datenwert kein Objekt oder wenn der verschachtelte Datenwert nicht den in der Abfrage verwendeten Attributnamen, der in der Abfrage verwendet wurde.

Example

Die folgende Abfrage greift beispielsweise auf einen ungültigen Attributnamen in der verschachtelten Datenspalte `c_orders`:

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

Die Array-Navigation gibt Null zurück, wenn der verschachtelte Datenwert kein Array oder der Array-Index außerhalb der Grenzen liegt.

Example

Die folgende Abfrage gibt Null zurück, weil `c_orders[1][1]` ist außerhalb der Grenzen.

```
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, verwendet AWS Clean Rooms die `JSON_TYPEOF`-Funktion, die den Typ (die oberste Ebene) des verschachtelten Datenwerts (die oberste Ebene) des verschachtelten Datenwerts (die oberste Ebene) des verschachtelten Datenwerts, die im folgenden Beispiel gezeigt wird:

```
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
```


Dokumentverlauf 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
SQL-Befehle und SQL-Funktionen – Update	Beispiele für die JOIN-Klausel, den EXCEPT-Satzoperator, den bedingten CASE-Ausdruck und die folgenden Funktionen wurden 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, ATAN2, COT, DEXP, PI, POW, RADIANS und SIN. Die folgenden JSON-Funktionen werden jetzt unterstützt: CAN_JSON_	06. Oktober 2023

PARSE, JSON_PARSE und
JSON_SERIALIZE.

[Unterstützung für verschachtelte Datentypen](#)

AWS Clean Rooms unterstützt jetzt verschachtelte Datentypen.

30. August 2023

[Regeln für die SQL-Benennung – Aktualisierung](#)

Änderung nur für die 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.