Référence SQL

AWS Clean Rooms

AWS Clean Rooms: Référence SQL

Table of Contents

Vue d'ensemble de SQL dans AWS Clean Rooms

Bienvenue dans la référence AWS Clean Rooms SQL.

AWS Clean Roomsrepose sur le langage de requête structuré (SQL) standard du secteur, un langage de requête composé de commandes et de fonctions que vous utilisez pour travailler avec des bases de données et des objets de base de données. SQL applique également des règles concernant l'utilisation des types de données, des expressions et des littéraux.

Les rubriques suivantes fournissent des informations générales sur les conventions et les règles de dénomination utilisées dans cette référence SQL.

Rubriques

- Conventions du guide de référence SQL
- Règles de dénomination SQL
- Prise en charge des types de données par le moteur SQL

Les sections suivantes fournissent des informations sur les littéraux, les types de données, les commandes SQL, les types de fonctions SQL et les conditions SQL que vous pouvez utiliser. AWS Clean Rooms

- AWS Clean Rooms SQL Spark

Pour plus d'informationsAWS Clean Rooms, consultez le guide de l'AWS Clean Roomsutilisateur et le guide de référence des AWS Clean Rooms API.

Conventions du guide de référence SQL

Cette section explique les conventions utilisées pour écrire la syntaxe des expressions, commandes et fonctions SQL.

Caractère	Description
CAPS	Les mots en lettres majuscules sont des mots clés.
[ ]	Les crochets indiquent des arguments facultatifs. Plusieurs arguments entre crochets signifient que vous

Caractère	Description
	pouvez choisir n’importe quel nombre d’arguments. En outre, les arguments entre crochets placés sur des lignes séparées indiquent que l'analyseur s'attend à ce que les arguments soient dans l'ordre où ils apparaiss ent dans la syntaxe.
{ }	Les accolades indiquent que vous devez choisir l’un des arguments proposés.
|	Les barres verticales indiquent que vous pouvez choisir entre les arguments.
italique	Les mots en italique correspondent à des espaces réservés. Vous devez insérer la valeur appropriée à la place du mot en italique.
...	Les trois points de suspension indiquent que vous pouvez répéter l’élément précédent.
'	Les mots entre apostrophes droites signifient que vous devez taper les apostrophes.

Règles de dénomination SQL

Les sections suivantes expliquent les règles de dénomination SQL dans AWS Clean Rooms.

Rubriques

Noms et colonnes d'associations de tables configurés

Les membres autorisés à effectuer des requêtes utilisent les noms d'associations de tables configurés comme noms de table dans les requêtes. Les noms des associations de tables configurées et les colonnes de table configurées peuvent être aliasés dans les requêtes.

Les règles de dénomination suivantes s'appliquent aux noms d'associations de tables configurés, aux noms de colonnes de table configurés et aux alias :

- Ils ne doivent utiliser que des caractères alphanumériques, un trait de soulignement (_) ou un trait d'union (-), mais ils ne peuvent pas commencer ou se terminer par un tiret.
  - (Règle d'analyse personnalisée uniquement) Ils peuvent utiliser le signe dollar ($) mais ne peuvent pas utiliser un modèle qui suit une constante de chaîne entre guillemets en dollars.

    Une constante de chaîne entre guillemets en dollars se compose des éléments suivants :
    - un signe du dollar ($)
    - un « tag » optionnel de zéro caractère ou plus
    - un autre signe du dollar
    - séquence arbitraire de caractères qui constitue le contenu de la chaîne
    - un signe du dollar ($)
    - le même tag qui a commencé la cotation du dollar
    - un signe du dollar

    Par exemple : `$$invalid$$`
- Ils ne peuvent pas contenir de tirets (-) consécutifs.
- Ils ne peuvent commencer par aucun des préfixes suivants :

  `padb_`, `pg_`, `stcs_`, `stl_`, `stll_`, `stv_`, `svcs_`, `svl_`, `svv_`, `sys_`, `systable_`
- Ils ne peuvent pas contenir de barres obliques inversées (\), de guillemets (') ou d'espaces qui ne sont pas entre guillemets doubles.
- S'ils commencent par un caractère non alphabétique, ils doivent être placés entre guillemets (» «).
- S'ils contiennent un trait d'union (-), ils doivent être entre guillemets (» «).
- Ils doivent comporter entre 1 et 127 caractères.
- Les mots réservés doivent être entre guillemets (» «).
- Les noms de colonnes réservés suivants ne peuvent pas être utilisés AWS Clean Rooms (même entre guillemets) :
  - oid
  - tabloïd
  - xmin
  - cmin

- xmax
- cmax
- ctid

Mots réservés

Voici une liste des mots réservés dans AWS Clean Rooms.

AES128	DELTA32KDESC	LEADING	PRIMARY
AES256ALL	DISTINCT	LEFTLIKE	RAW
ALLOWOVER WRITEANALYSE	DO	LIMIT	READRATIO
ANALYZE	DISABLE	LOCALTIME	RECOVERRE FERENCES
AND	ELSE	LOCALTIMESTAMP	REJECTLOG
ANY	EMPTYASNU LLENABLE	LUN	RESORT
ARRAY	ENCODE	LUNS	RESPECT
AS	ENCRYPT	LZO	RESTORE
ASC	ENCRYPTIONEND	LZOP	RIGHTSELECT
AUTHORIZATION	EXCEPT	MINUS	SESSION_USER
AZ64	EXPLICITFALSE	MOSTLY16	SIMILAR
BACKUPBETWEEN	FOR	MOSTLY32	SNAPSHOT
BINARY	FOREIGN	MOSTLY8NATURAL	SOME
BLANKSASN ULLBOTH	FREEZE	NEW	SYSDATESYSTEM

BYTEDICT	FROM	NOT	TABLE
BZIP2CASE	FULL	NOTNULL	TAG
CAST	GLOBALDICT256	NULL	TDES
CHECK	GLOBALDIC T64KGRANT	NULLSOFF	TEXT255
COLLATE	GROUP	OFFLINEOFFSET	TEXT32KTHEN
COLUMN	GZIPHAVING	OID	TIMESTAMP
CONSTRAINT	IDENTITY	OLD	TO
CREATE	IGNOREILIKE	ON	TOPTRAILING
CREDENTIA LSCROSS	IN	ONLY	TRUE
CURRENT_DATE	INITIALLY	OPEN	TRUNCATEC OLUMNSUNION
CURRENT_TIME	INNER	OR	UNIQUE
CURRENT_T IMESTAMP	INTERSECT	ORDER	UNNEST
CURRENT_USER	INTERVAL	OUTER	USING
CURRENT_U SER_IDDEFAULT	INTO	OVERLAPS	VERBOSE
DEFERRABLE	IS	PARALLELP ARTITION	WALLETWHEN
DEFLATE	ISNULL	PERCENT	WHERE
DEFRAG	JOIN	PERMISSIONS	WITH
DELTA	LANGUAGE	PIVOTPLACING	WITHOUT

Prise en charge des types de données par le moteur SQL

AWS Clean Rooms prend en charge plusieurs moteurs et dialectes SQL. Comprendre les systèmes de types de données dans ces implémentations est essentiel pour une collaboration et une analyse de données réussies. Les tableaux suivants présentent les types de données équivalents dans AWS Clean Rooms SQL, Snowflake SQL et Spark SQL.

Types de données numériques

Les types numériques représentent différents types de nombres, des nombres entiers précis aux valeurs approximatives à virgule flottante. Le choix du type numérique influe à la fois sur les exigences de stockage et sur la précision des calculs. Les types entiers varient en fonction de la taille des octets, tandis que les types décimaux et à virgule flottante offrent différentes options de précision et d'échelle.

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
Entier de 8 octets	BIGINT	Non pris en charge	GROS, LONG	Entiers signés compris entre -9,223,372 036 854 775 808 et 9 223 372 036 854 775 807.
Entier de 4 octets	INT	Non pris en charge	INT, INTEGER	Entiers signés compris entre -2 147 483 648 et 2 147 483 647
Entier de 2 octets	SMALLINT	Non pris en charge	SMALL INT, SHORT	Entiers signés compris entre -32 768 et 32 767

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
Entier sur 1 octet	Non pris en charge	Non pris en charge	TINYINT, OCTET	Entiers signés compris entre -128 et 127
Flotteur à double précision	DOUBLE, DOUBLE PRÉCISION	FLOAT FLOAT4, FLOAT8, DOUBLE, DOUBLE PRÉCISION, RÉEL	DOUBLE	Numéros à virgule flottante à double précision sur 8 octets
Flotteur de précision unique	RÉEL, FLOTTANT	Non pris en charge	FLOAT	Numéros à virgule flottante à précision unique sur 4 octets

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
Décimal (précision fixe)	DECIMAL	DÉCIMAL, NUMÉRIQUE, NOMBRE **Note** Snowflake aliase automatiquement les types numériques exacts de plus petite largeur (INT, BIGINT, SMALLINT, etc.) en NUMBER.	DÉCIMAL, NUMÉRIQUE,	Nombres décimaux signés avec une précision arbitraire
Décimal (avec précision)	DÉCIMAL (p)	DÉCIMAL (p), NOMBRE (p)	DÉCIMAL (p)	Nombres décimaux à précision fixe
Décimal (avec échelle)	DECIMAL(p,s)	DÉCIMAL (p, s), NOMBRE (p, s)	DECIMAL(p,s)	Nombres décimaux à précision fixe avec échelle

Types de données booléennes

Les types booléens représentent des valeurs true/false logiques simples. Ces types sont cohérents dans tous les moteurs SQL et sont couramment utilisés pour les indicateurs, les conditions et les opérations logiques.

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
Booléen	BOOLEAN	BOOLEAN	BOOLEAN	Représente true/false des valeurs

Types de données de date et d’heure

Les types de date et d'heure traitent des données temporelles, avec différents niveaux de précision et de connaissance du fuseau horaire. Ces types prennent en charge différents formats pour le stockage des dates, des heures et des horodatages, avec des options permettant d'inclure ou d'exclure les informations de fuseau horaire.

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
Date	DATE	DATE	DATE	Valeurs de date (année, mois, jour) sans fuseau horaire
Heure	TIME	Non pris en charge	Non pris en charge	Heure du jour en UTC, sans fuseau horaire
Temps passé avec TZ	TIMETZ	Non pris en charge	Non pris en charge	Heure du jour en UTC, avec fuseau horaire

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
Horodatage	TIMESTAMP	HORODATAGE, TIMESTAMP_NTZ	TIMESTAMP_NTZ	Horodatage sans fuseau horaire **Note** NTZ indique « Pas de fuseau horaire »
Horodatage avec TZ	TIMESTAMPTZ	TIMESTAMP_LTZ	HORODATAGE, TIMESTAMP_LTZ	Horodatage avec fuseau horaire local **Note** LTZ indique le « fuseau horaire local »

Types de données de caractères

Les types de caractères stockent des données textuelles, offrant à la fois des options de longueur fixe et de longueur variable. Ces types gèrent les chaînes de texte et les données binaires, avec des spécifications de longueur facultatives pour contrôler l'allocation de stockage.

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
Caractère de longueur fixe	CHAR	CHAR, CARACTÈRE	CHAR, CARACTÈRE	Chaîne de caractères de longueur fixe
Caractère de longueur fixe avec longueur	CHAR(n)	CARACTÈRE (n), CARACTÈRE (n)	CARACTÈRE (n), CARACTÈRE (n)	Chaîne de caractères de longueur fixe avec une longueur spécifiée
Caractère de longueur variable	VARCHAR	VARCHAR, CHAÎNE, TEXTE	VARCHAR, CHAÎNE	Chaîne de caractères de longueur variable
Caractère de longueur variable avec longueur	VARCHAR(n)	VARCHAR (n), CHAÎNE (n), TEXTE (n)	VARCHAR(n)	Chaîne de caractères de longueur variable avec limite de longueur
Binaire	VARBYTE	BINARY, VARBINARY	BINAIRE	Séquence d'octets binaires
Binaire avec longueur	VARBYTE(n)	Non pris en charge	Non pris en charge	Séquence d'octets binaires avec limite de longueur

Types de données structurées

Les types structurés permettent une organisation complexe des données en combinant plusieurs valeurs dans des champs uniques. Il s'agit notamment de tableaux pour les collections ordonnées, de cartes pour les paires clé-valeur et de structures pour créer des structures de données personnalisées avec des champs nommés.

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
Tableau	MATRICE <type>	TABLEAU (type)	MATRICE <type>	Séquence ordonnée d'éléments du même type **Note** Les types de tableaux doivent contenir des éléments du même type
Map	CARTE<key, value>	MAP (clé, valeur)	CARTE<key, value>	Collection de paires clé-valeur **Note** Les types

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
				de cartes doivent contenir des éléments du même type
Struct	STRUCTURE < field1 : type1, field2 : type2>	OBJET (champ1 type1, champ2 type2)	STRUCTURE < field1 : type1, field2 : type2 >	Structure avec champs nommés de types spécifiés **Note** La syntaxe des types structurés peut varier légèrement entre les implémentations

Type de données	AWS Clean Rooms SQL	Snowflake SQL	SQL Spark	Description
Super	SUPER	Non pris en charge	Non pris en charge	Type flexible supportant tous les types de données, y compris les types complexes

AWS Clean Rooms SQL Spark

AWS Clean Rooms Spark SQL applique les règles relatives à l'utilisation des types de données, des expressions et des littéraux.

Pour plus d'informations sur AWS Clean Rooms Spark SQL, consultez le guide de l'AWS Clean Rooms utilisateur et le guide de référence des AWS Clean Rooms API.

Les rubriques suivantes fournissent des informations sur les littéraux, les types de données, les commandes, les fonctions et les conditions pris en charge dans AWS Clean Rooms Spark SQL.

Rubriques

Littéraux

Un littéral ou une constante est une valeur de données fixe, composée d’une séquence de caractères ou d’une constante numérique.

AWS Clean Rooms Spark SQL prend en charge plusieurs types de littéraux, notamment :

- Les littéraux numériques pour les entiers, les décimaux et les nombres à virgule flottante.
- Les littéraux de caractères, également appelés chaînes, chaînes de caractères ou constantes de caractères, sont utilisés pour spécifier une valeur de chaîne de caractères.
- Litéraux de date, d'heure et d'horodatage, utilisés avec les types de données datetime. Pour de plus amples informations, veuillez consulter Littéraux de type date, heure et horodatage.
- Littéraux d'intervalle. Pour de plus amples informations, veuillez consulter Littéraux de type interval.
- Littéraux booléens. Pour de plus amples informations, veuillez consulter Littéraux booléens.
- Littéraux nuls, utilisés pour spécifier une valeur nulle.
- Uniquement TAB, CARRIAGE RETURN (CR), et LINE FEED (LF) Les caractères de contrôle Unicode de la catégorie générale Unicode (Cc) sont pris en charge.

AWS Clean Rooms Spark SQL ne prend pas en charge les références directes à des chaînes littérales dans la clause SELECT, mais elles peuvent être utilisées dans des fonctions telles que CAST.

+ Opérateur (concaténation)

Concatène les littéraux numériques, les littéraux de chaîne et/ou les littéraux de date/heure et d'intervalle. Ils se trouvent de chaque côté du symbole + et renvoient des types différents en fonction des entrées de chaque côté du symbole +.

Syntaxe

```
numeric + string
```

```
date + time
```

```
date + timetz
```

L'ordre des arguments peut être inversé.

Arguments

numeric literals

Les littéraux ou constantes qui représentent des nombres peuvent être des entiers ou à virgule flottante.

string literals

Chaînes, chaînes de caractères ou constantes de caractères

date

A DATE colonne ou expression qui est implicitement convertie en DATE.

time

A TIME colonne ou expression qui est implicitement convertie en TIME.

timetz

A TIMETZ colonne ou expression qui est implicitement convertie en TIMETZ.

exemple

Le tableau d'exemple suivant TIME_TEST comporte une colonne TIME_VAL (type TIME) avec trois valeurs insérées.

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

Types de données

Chaque valeur stockée ou récupérée par AWS Clean Rooms Spark SQL possède un type de données associé à un ensemble fixe de propriétés associées. Les types de données sont déclarés lorsque les tables sont créées. Un type de données contraint l'ensemble des valeurs qu'une colonne ou un argument peut contenir.

Le tableau suivant répertorie les types de données que vous pouvez utiliser dans AWS Clean Rooms Spark SQL.

Nom du type de données	Type de données	les alias ;	Description
ARRAY	the section called "Type imbriqué"	Non applicable	Type de données imbriqué dans un tableau
BIGINT	the section called "Types numériques"	Non applicable	Entier signé sur huit octets
BINAIRE	the section called "Type binaire"	Non applicable	Valeurs de séquence d'octets
BOOLEAN	the section called "Type Boolean"	BOOL	Booléen logique (true/false)
BYTE	the section called "Types numériques"	Non applicable	Nombres entiers signés sur 1 octet, compris entre -128 et 127

Nom du type de données	Type de données	les alias ;	Description
CHAR	the section called “Types caractères”	CHARACTER	Chaîne de caractères de longueur fixe
DATE	the section called “Types datetime”	Non applicable	Date calendaire (année, mois, jour)
DECIMAL	the section called “Types numériques”	NUMERIC	Valeur numérique exacte avec précision sélectionnable
FLOAT	the section called “Types numériques”	FLOAT8, DOUBLE PRÉCISION	Nombre à virgule flottante de double précision
INTEGER	the section called “Types numériques”	INT	Entier signé sur quatre octets
INTERVAL	the section called “Types datetime”	Non applicable	Durée en ordre journalier ou mensuel
LONG	the section called “Types numériques”	Non applicable	Nombres entiers signés sur 8 octets
MAP	the section called “Type imbriqué”	Non applicable	Type de données imbriquées sur la carte
REAL	the section called “Types numériques”	FLOAT4	Nombre à virgule flottante simple précision
SHORT	the section called “Types numériques”	Non applicable	Nombres entiers signés sur 2 octets.
SMALLINT	the section called “Types numériques”	Non applicable	Entier signé sur deux octets

Nom du type de données	Type de données	les alias ;	Description
STRUCT	the section called “Type imbriqué”	Non applicable	Type de données imbriqué dans la structure
TIMESTAMP_LTZ	the section called “Types datetime”	Non applicable	Heure de la journée avec fuseau horaire local
TIMESTAMP_NTZ	the section called “Types datetime”	Non applicable	Heure de la journée sans fuseau horaire
TINYINT	the section called “Types numériques”	Non applicable	Nombres entiers signés sur 1 octet, compris entre -128 et 127
VARCHAR	the section called “Types caractères”	CARACTÈRE VARIABLE	Chaîne de caractères de longueur variable avec une limite définie par l’utilisateur

> **Note**
>
> Les types de données imbriqués ARRAY, STRUCT et MAP ne sont actuellement activés que pour la règle d'analyse personnalisée. Pour de plus amples informations, veuillez consulter Type imbriqué.

Caractères multioctets

Le type de données VARCHAR prend en charge les caractères multioctets UTF-8 jusqu’à un maximum de quatre octets. Les caractères de cinq octets ou plus ne sont pas pris en charge. Pour calculer la taille d’une colonne VARCHAR qui contient des caractères multioctets, multipliez le nombre de caractères par le nombre d’octets par caractère. Par exemple, si une chaîne possède

quatre caractères chinois et que chaque caractère est long de trois octets, vous avez besoin d'une colonne VARCHAR(12) pour stocker la chaîne.

Le type de données VARCHAR ne prend pas en charge les points de code UTF-8 non valides suivants :

`0xD800 – 0xDFFF` (Séquences d'octets :`ED A0 80` – `ED BF BF`)

Le type de données CHAR ne prend pas en charge les caractères multioctets.

Types numériques

Les types de données numériques incluent les entiers, les décimaux et les nombres à virgule flottante.

Rubriques

- Types d'entier
- Type DECIMAL ou NUMERIC
- Types à virgule flottante
- Calculs avec les valeurs numériques

Types d'entier

Utilisez les types de données suivants pour stocker des nombres entiers de différentes plages. Vous ne pouvez pas stocker de valeurs en dehors de la plage autorisée pour chaque type.

Nom	Stockage	Range
SMALLINT	2 bytes	-32768 à +32767
SHORT	2 bytes	-32768 à +32767
INTEGER ou INT	4 octets	-2147483648 à +2147483647
BIGINT	8 octets	-9223372036854775808 à 9223372036854775807

Nom	Stockage	Range
LONG	8 octets	-9223372036854775808 à 9223372036854775807

Type DECIMAL ou NUMERIC

Utilisez le type de données DECIMAL ou NUMERIC pour stocker les valeurs avec une précision définie par l'utilisateur. Les mots clés DECIMAL et NUMERIC sont interchangeables. Dans ce document, decimal est le terme privilégié pour ce type de données. Le terme numeric (numérique) est utilisé de façon générique pour faire référence aux types de données integer, decimal et floating-point (entier, décimal et virgule flottante).

Stockage	Range
Variable, jusqu'à 128 bits pour les types DECIMAL non compressés.	Entiers signés 128 bits avec précision maximale de 38 chiffres.

Définissez une colonne DECIMAL dans un tableau en spécifiant a *precision* et *scale* :

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

precision

Le nombre total de chiffres significatifs dans la valeur entière : le nombre de chiffres de chaque côté de la virgule. Par exemple, le nombre `48.2891` a une précision de 6 et une échelle de 4. La précision par défaut, si elle n'est pas spécifiée, est de 18. La précision maximale est de 38.

Si le nombre de chiffres à gauche de la virgule décimale dans une valeur d'entrée dépasse la précision de la colonne moins son échelle, la valeur ne peut pas être copiée dans la colonne (ni insérée ou mise à jour). Cette règle s'applique à toute valeur qui se trouve en dehors de la plage de la définition de la colonne. Par exemple, la plage autorisée de valeurs pour une colonne `numeric(5,2)` s'étend de `-999.99` à `999.99`.

scale

Le nombre de chiffres décimaux de la partie fractionnaire de la valeur, à droite de la virgule. Les entiers possèdent une échelle égale à zéro. Dans une spécification de colonne, la valeur de l'échelle doit être inférieure ou égale à la valeur de la précision. L'échelle par défaut, si elle n'est pas spécifiée, est de 0. L'échelle maximale est de 37.

Si l'échelle d'une valeur d'entrée chargée dans une table est supérieure à l'échelle de la colonne, la valeur est arrondie à l'échelle spécifiée. Par exemple, la colonne PRICEPAID de la table SALES est une colonne DECIMAL(8,2). Si une valeur DECIMAL(8,4) est insérée dans la colonne PRICEPAID, la valeur est arrondie à une échelle de 2.

```
insert into sales
values (0, 8, 1, 1, 2000, 14, 5, 4323.8951, 11.00, null);

select pricepaid, salesid from sales where salesid=0;

pricepaid | salesid
-----------+---------
4323.90 |       0
(1 row)
```

Cependant, les résultats de conversions explicites de valeurs sélectionnées dans les tables ne sont pas arrondis.

Note

La valeur positive maximale que vous pouvez insérer dans une colonne DECIMAL(19,0) est `9223372036854775807` (2^{63} -1). La valeur négative maximale est `-9223372036854775807`. Par exemple, une tentative d'insérer la valeur 9999999999999999999 (19 fois le chiffre neuf) entraîne une erreur de dépassement de capacité. Quel que soit le placement de la virgule décimale, la plus grande chaîne qu' AWS Clean Rooms puisse représenter comme nombre DECIMAL est `9223372036854775807`. Par exemple, la plus grande valeur que vous puissiez charger dans une colonne DECIMAL(19,18) est `9.223372036854775807`.
Ces règles sont dues aux raisons suivantes :

- Les valeurs DECIMAL dont la précision est inférieure ou égale à 19 chiffres significatifs sont stockées en interne sous forme de nombres entiers de 8 octets.

- Les valeurs DECIMAL avec une précision de 20 à 38 chiffres significatifs sont stockées sous forme de nombres entiers de 16 octets.

Notes sur l'utilisation des colonnes DECIMAL ou NUMERIC 128 bits

N'attribuez pas de façon arbitraire une précision maximale aux colonnes DECIMAL, sauf si vous avez la certitude que votre application a besoin de cette précision. Les valeurs 128 bits utilisent deux fois plus d'espace disque que les valeurs 64 bits et peuvent ralentir le temps d'exécution des requêtes.

Types à virgule flottante

Utilisez les types de données REAL et DOUBLE PRECISION pour stocker les valeurs numériques avec une précision variable. Ces types sont des types inexacts, ce qui signifie que certaines valeurs sont stockées comme approximations, de telle sorte que le stockage et le retour d'une valeur spécifique peuvent se traduire par de légers écarts. Si vous avez besoin d'un stockage et d'un calcul exacts (pour des montants monétaires, par exemple), utilisez le type de données DECIMAL.

REAL représente le format à virgule flottante à précision unique, conformément à la norme IEEE 754 pour l'arithmétique à virgule flottante. Il a une précision d'environ 6 chiffres et une plage d'environ 1E-37 à 1E+37. Vous pouvez également spécifier ce type de données sous la forme FLOAT4.

DOUBLE PRECISION représente le format de virgule flottante en double précision, conformément à la norme IEEE 754 pour l'arithmétique binaire en virgule flottante. Il a une précision d'environ 15 chiffres et une plage d'environ 1E-307 à 1E+308. Vous pouvez également spécifier ce type de données sous la forme FLOAT ou FLOAT8.

Calculs avec les valeurs numériques

Dans AWS Clean Rooms, le calcul fait référence aux opérations mathématiques binaires : addition, soustraction, multiplication et division. Cette section décrit les types de retour attendus pour ces opérations, ainsi que la formule spécifique appliquée pour déterminer la précision et l'échelle lorsque les types de données DECIMAL sont impliqués.

Lorsque des valeurs numériques sont calculées pendant le traitement des requêtes, vous pouvez rencontrer des cas où le calcul est impossible et où la requête renvoie une erreur de dépassement de capacité numérique. Vous pouvez également rencontrer des cas où l'échelle des valeurs calculées varie ou est inattendue. Pour certaines opérations, vous pouvez utiliser le transtypage explicite

(promotion de type) ou les paramètres de configuration de AWS Clean Rooms afin de contourner ces problèmes.

Pour plus d'informations sur les résultats de calculs similaires avec les fonctions SQL, consultez AWS Clean Rooms Fonctions Spark SQL.

Types de retour pour les calculs

Compte tenu de l'ensemble des types de données numériques pris en charge dans AWS Clean Rooms, le tableau suivant indique les types de retour attendus pour les opérations d'addition, de soustraction, de multiplication et de division. La première colonne sur la gauche du tableau représente le premier opérande dans le calcul et la ligne du haut le second opérande.

Opérande 1	Opérande 2	Type de retour
SMALLINT ou SHORT	SMALLINT ou SHORT	SMALLINT ou SHORT
SMALLINT ou SHORT	INTEGER	INTEGER
SMALLINT ou SHORT	BIGINT	BIGINT
SMALLINT ou SHORT	DECIMAL	DECIMAL
SMALLINT ou SHORT	FLOAT4	FLOAT8
SMALLINT ou SHORT	FLOAT8	FLOAT8
INTEGER	INTEGER	INTEGER
INTEGER	BIGINT ou LONG	BIGINT ou LONG
INTEGER	DECIMAL	DECIMAL
INTEGER	FLOAT4	FLOAT8
INTEGER	FLOAT8	FLOAT8
BIGINT ou LONG	BIGINT ou LONG	BIGINT ou LONG
BIGINT ou LONG	DECIMAL	DECIMAL
BIGINT ou LONG	FLOAT4	FLOAT8

Opérande 1	Opérande 2	Type de retour
BIGINT ou LONG	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL
DECIMAL	FLOAT4	FLOAT8
DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8
FLOAT8	FLOAT8	FLOAT8

Précision et échelle des résultats DECIMAL calculés

Le tableau suivant résume les règles de calcul de la précision et de l'échelle obtenues lorsque les opérations mathématiques retournent des résultats DECIMAL. Dans ce tableau, `p1` et `s1` représentez la précision et l'échelle du premier opérande d'un calcul. `p2`et `s2` représentent la précision et l'échelle du deuxième opérande. (Quels que soient les calculs, la précision maximale du résultat est de 38 et l'échelle maximale du résultat de 38 également.)

Opération	Précision et échelle du résultat
+ ou -	Évolutivité = `max(s1,s2)` Précision = `max(p1-s1,p2-s2)+1+scale`
*	Évolutivité = `s1+s2` Précision = `p1+p2+1`
/	Évolutivité = `max(4,s1+p2-s2+1)` Précision = `p1-s1+ s2+scale`

Par exemple, les colonnes PRICEPAID et COMMISSION de la table SALES sont toutes deux des colonnes DECIMAL(8,2). Si vous divisez PRICEPAID par COMMISSION (ou inversement), la formule est appliquée comme suit :

```
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)
```

Le calcul suivant constitue la règle générale pour le calcul de la précision et de l'échelle obtenues dans le cas des opérations effectuées sur les valeurs DECIMAL avec les opérateurs définis tels que UNION, INTERSECT et EXCEPT, ou les fonctions comme COALESCE et DECODE :

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

Par exemple, une DEC1 table avec une colonne DECIMAL (7,2) est jointe à une DEC2 table avec une colonne DECIMAL (15,3) pour créer une table. DEC3 Le schéma de DEC3 montre que la colonne devient une colonne NUMERIC (15,3).

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

Dans l'exemple ci-dessus, la formule est appliquée comme suit :

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

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

Remarques sur les opérations de division

Pour les opérations de division, divide-by-zero les conditions renvoient des erreurs.

La limite d'échelle de 100 est appliquée après le calcul de la précision et de l'échelle. Si l'échelle de résultat calculée est supérieure à 100, les résultats de la division sont mis à l'échelle comme suit :

- Précision = `precision - (scale - max_scale)`
- Évolutivité = `max_scale`

Si la précision calculée est supérieure à la précision maximale (38), la précision est réduite à 38, et l'échelle devient le résultat de : `max(38 + scale - precision), min(4, 100))`

Conditions de dépassement de capacité

Le dépassement de capacité est contrôlé pour tous les calculs numériques. Les données DECIMAL avec une précision de 19 ou moins sont stockées en tant qu'entiers 64 bits. Les données DECIMAL avec une précision supérieure à 19 sont stockées sous forme d'entiers 128 bits. La précision maximale de toutes les valeurs DECIMAL est 38 et l'échelle maximale 37. Les erreurs de dépassement de capacité se produisent quand une valeur dépasse ces limites, qui s'appliquent aux jeux de résultats intermédiaires et finaux :

- Le casting explicite entraîne des erreurs de dépassement d'exécution lorsque des valeurs de données spécifiques ne correspondent pas à la précision ou à l'échelle requises spécifiées par la fonction de conversion. Par exemple, vous ne pouvez pas convertir toutes les valeurs de la colonne PRICEPAID de la table SALES (une colonne DECIMAL (8,2)) et renvoyer un résultat DECIMAL (7,3) :

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

  Cette erreur se produit car certaines des valeurs les plus élevées de la colonne PRICEPAID ne peuvent pas être converties.
- Les opérations de multiplication produisent des résultats dans lesquels l'échelle du résultat est la somme des échelles de chaque opérande. Si les deux opérandes ont une échelle de 4, par exemple, l'échelle du résultat est 8, ce qui ne laisse que 10 chiffres à gauche de la virgule. Par conséquent, il est relativement facile de se trouver en situation de dépassement de capacité lors de la multiplication de deux grands nombres ayant une échelle significative.

Calculs numériques avec les types INTEGER et DECIMAL

Lorsque l'un des opérandes d'un calcul est de type INTEGER et que l'autre est DECIMAL, l'opérande INTEGER est implicitement converti en DECIMAL.

- SMALLINT ou SHORT est converti en DECIMAL (5,0)
- INTEGER est converti en DECIMAL (10,0)
- BIGINT ou LONG est converti en DECIMAL (19,0)

Par exemple, si vous multipliez SALES.COMMISSION, colonne DECIMAL(8,2) et SALES.QTYSOLD, colonne SMALLINT, le calcul est converti comme suit :

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

Types caractères

Les types de données caractères incluent CHAR (caractère) et VARCHAR (caractère variable).

Rubriques

- CHAR ou CHARACTER
- VARCHAR ou CHARACTER VARYING
- Signification des blancs de fin

CHAR ou CHARACTER

Utilisez une colonne CHAR ou CHARACTER pour stocker les chaînes de longueur fixe. Ces chaînes étant remplies de blancs, une colonne CHAR(10) occupe toujours 10 octets de stockage.

```
char(10)
```

Une colonne CHAR sans spécification de longueur se traduit par une colonne CHAR(1).

Les types de données CHAR et VARCHAR sont définis en termes d'octets, pas de caractères. Comme une colonne CHAR ne peut contenir que des caractères d'un octet, une colonne CHAR(10) peut contenir une chaîne d'une longueur maximale de 10 octets.

Nom	Stockage	Plage (largeur de colonne)
CHAR ou CHARACTER	Longueur de la chaîne, blancs de fin inclus (le cas échéant)	4096 bytes

VARCHAR ou CHARACTER VARYING

Utilisez une colonne VARCHAR ou CHARACTER VARYING pour stocker des chaînes de longueur variable avec une limite fixe. Comme ces chaînes ne sont pas remplies avec des blancs, une colonne VARCHAR(120) se compose d'un maximum de 120 caractères codés sur un octet, de 60 caractères codés sur deux octets, de 40 caractères codés sur trois octets ou de 30 caractères codés sur quatre octets.

```
varchar(120)
```

Les types de données VARCHAR sont définis en termes d'octets et non de caractères. Une donnée VARCHAR peut contenir des caractères multioctets, jusqu'à un maximum de quatre octets par caractère. Par exemple, une colonne VARCHAR(12) peut contenir 12 caractères codés sur un octet, 6 caractères codés sur deux octets, 4 caractères codés sur trois octets ou 3 caractères codés sur quatre octets.

Nom	Stockage	Plage (largeur de colonne)
VARCHAR ou CHARACTER VARYING	4 octets + le nombre total d'octets des caractères, où chaque caractère peut être codé sur 1 à 4 octets.	65535 octets (64 K -1)

Signification des blancs de fin

Les types de données CHAR et VARCHAR stockent les chaînes de longueur maximale de n octets. Toute tentative de stockage d'une chaîne plus longue dans une colonne de ce type entraîne une erreur. Toutefois, si les caractères supplémentaires sont tous des espaces (blancs), la chaîne est tronquée à la longueur maximale. Si la chaîne est plus courte que la longueur maximale, les valeurs CHAR sont remplies de blancs, mais les valeurs VARCHAR stockent la chaîne sans blancs.

Les blancs de fin des valeurs CHAR sont toujours insignifiants sur le plan sémantique. Ils sont ignorés lorsque vous comparez deux valeurs CHAR, ne sont pas inclus dans les calculs LENGTH et sont supprimés lorsque vous convertissez une valeur CHAR en un autre type de chaîne.

Les espaces de fin des valeurs VARCHAR et CHAR sont traités comme sans importance du point de vue sémantique lorsque les valeurs sont comparées.

Les longueurs de calcul retournent la longueur des chaînes de caractères VARCHAR avec les espaces de fin inclus dans la longueur. Les blancs de fin ne comptent pas dans la longueur des chaînes de caractères de longueur fixe.

Types datetime

Les types de données Datetime incluent DATE, TIME, TIMESTAMP_LTZ et TIMESTAMP_NTZ.

Rubriques

DATE

Utilisez le type de données DATE pour stocker les dates calendaires simples sans horodatage.

Nom	Stockage	Range	Résolution
DATE	4 octets	4713 av. J.-C. à 294276 apr. J.-C.	1 jour

TIMESTAMP_LTZ

Utilisez le type de données TIMESTAMP_LTZ pour stocker les valeurs d'horodatage complètes qui incluent la date, l'heure et le fuseau horaire local.

TIMESTAMP représente des valeurs comprenant les valeurs des champs`year`,,`month`,`day`, et `hour` `minutesecond`, avec le fuseau horaire local de la session. La `timestamp` valeur représente un point absolu dans le temps.

Dans Spark, TIMESTAMP est un alias spécifié par l'utilisateur associé à l'une des variantes TIMESTAMP_LTZ et TIMESTAMP_NTZ. Vous pouvez définir le type d'horodatage par défaut comme TIMESTAMP_LTZ (valeur par défaut) ou TIMESTAMP_NTZ via la configuration. `spark.sql.timestampType`

TIMESTAMP_NTZ

Utilisez le type de données TIMESTAMP_NTZ pour stocker les valeurs d'horodatage complètes qui incluent la date et l'heure, sans le fuseau horaire local.

TIMESTAMP représente des valeurs comprenant les valeurs des champs`year`,`month`,`day`, `hourminute`, et. `second` Toutes les opérations sont effectuées sans tenir compte du fuseau horaire.

Dans Spark, TIMESTAMP est un alias spécifié par l'utilisateur associé à l'une des variantes TIMESTAMP_LTZ et TIMESTAMP_NTZ. Vous pouvez définir le type d'horodatage par défaut comme TIMESTAMP_LTZ (valeur par défaut) ou TIMESTAMP_NTZ via la configuration. `spark.sql.timestampType`

Exemples avec les types datetime

Les exemples suivants vous montrent comment utiliser les types de date/heure pris en charge par AWS Clean Rooms.

Exemples de date

Les exemples suivants insèrent des dates qui ont des formats différents et affichent le résultat.

```
select * from datetable order by 1;

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

Si vous insérez une valeur d’horodatage dans une colonne DATE, la partie temps est ignorée et seule la date est chargée.

Exemples d’heure

Les exemples suivants insèrent des valeurs TIME et TIMETZ qui n’ont pas le même format et affichent la sortie.

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

Littéraux de type date, heure et horodatage

Vous trouverez ci-dessous les règles d'utilisation des littéraux de date, d'heure et d'horodatage pris en charge par AWS Clean Rooms Spark SQL.

Dates

Le tableau suivant présente les dates d'entrée qui sont des exemples valides de valeurs de date littérales que vous pouvez charger dans AWS Clean Rooms des tables. La valeur `MDY DateStyle` par défaut est supposée être en vigueur. Ce mode signifie que la valeur month précède la valeur day dans les chaînes telles que `1999-01-08` et `01/02/00`.

> **Note**
>
> Une date ou un horodatage littéral doit être placé entre guillemets lorsque vous le chargez dans une table.

Date en entrée	Date complète
8 janvier 1999	8 janvier 1999
1999-01-08	8 janvier 1999
1/8/1999	8 janvier 1999
01/02/00	2 janvier 2000
2000-Jan-31	31 janvier 2000

Date en entrée	Date complète
Jan-31-2000	31 janvier 2000
31-Jan-2000	31 janvier 2000
20080215	15 février 2008
080215	15 février 2008
2008.366	31 décembre 2008 (la partie à trois chiffres de la date doit être comprise entre 001 et 366)

Times

Le tableau suivant indique les heures d'entrée qui sont des exemples valides de valeurs temporelles littérales que vous pouvez charger dans AWS Clean Rooms des tables.

Heures en entrée	Description (de la partie heure)
04:05:06.789	4:05 AM et 6,789 secondes
04:05:06	4:05 AM et 6 secondes
04:05	4:05 AM exactement
040506	4:05 AM et 6 secondes
04:05 AM	4:05 AM exactement ; AM est facultatif
04:05 PM	4:05 PM exactement ; la valeur d’heure doit être inférieure à 12
16h05	4:05 PM exactement

Valeurs datetime spéciales

Le tableau suivant indique les valeurs spéciales qui peuvent être utilisées comme littéraux de date/heure et comme arguments de fonctions de date. Elles requièrent des apostrophes droites et sont converties en valeurs timestamp régulières lors du traitement de la requête.

Valeur spéciale	Description
now	Correspond à l'heure de début de la transacti on actuelle et retourne un horodatage avec une précision de l'ordre de la microseconde.
today	Correspond à la date appropriée et renvoie un horodatage avec des zéros pour la partie heure.
tomorrow	Correspond à la date appropriée et renvoie un horodatage avec des zéros pour la partie heure.
yesterday	Correspond à la date appropriée et renvoie un horodatage avec des zéros pour la partie heure.

Les exemples suivants montrent comment now et comment utiliser today la fonction DATE_ADD.

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

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

select date_add('now', 1);

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

Littéraux de type interval

Vous trouverez ci-dessous les règles d'utilisation des littéraux d'intervalle pris en charge par AWS Clean Rooms Spark SQL.

Utilisez un littéral de type interval pour identifier les périodes spécifiques, comme `12 hours` ou `6 weeks`. Vous pouvez utiliser ces littéraux de type interval dans les cas et les calculs qui impliquent des expressions de type datetime.

> **Note**
>
> Vous ne pouvez pas utiliser le type de données INTERVAL pour les colonnes des AWS Clean Rooms tables.

Un intervalle est exprimé comme la combinaison du mot clé INTERVAL avec une quantité numérique et d'une partie date prise en charge ; par exemple : `INTERVAL '7 days'` ou `INTERVAL '59 minutes'`. Plusieurs quantités et unités peuvent être associées pour former un intervalle plus précis ; par exemple : `INTERVAL '7 days, 3 hours, 59 minutes'`. Les abréviations et les pluriels de chaque unité sont également pris en charge ; par exemple : `5 s`, `5 second` et `5 seconds` sont des intervalles équivalents.

Si vous ne spécifiez pas une partie date, la valeur de l'intervalle correspond à des secondes. Vous pouvez spécifier la valeur de la quantité sous forme de fraction (par exemple : `0.5 days`).

Exemples

Les exemples suivants illustrent une série de calculs avec différentes valeurs d'intervalle.

L'exemple suivant ajoute 1 seconde à la date spécifiée.

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

L'exemple suivant ajoute 1 minute à la date spécifiée.

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

L'exemple suivant ajoute 3 heures et 35 minutes à la date spécifiée.

```
select caldate + interval '3 hours, 35 minutes' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 03:35:00
(1 row)
```

L'exemple suivant ajoute 52 semaines à la date spécifiée.

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

L'exemple suivant ajoute 1 semaine, 1 heure, 1 minute et 1 seconde à la date spécifiée.

```
select caldate + interval '1w, 1h, 1m, 1s' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2009-01-07 01:01:01
(1 row)
```

L'exemple suivant ajoute 12 heures (une demi-journée) à la date spécifiée.

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

```
2008-12-31 12:00:00
(1 row)
```

L'exemple suivant soustrait 4 mois à compter du 31 mars 2023 et le résultat est le 30 novembre 2022. Le calcul prend en compte le nombre de jours dans un mois.

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

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

Types de données et littéraux interval

Vous pouvez utiliser un type de données interval pour stocker les durées dans des unités telles que `seconds`, `minutes`, `hours`, `days`, `months` et `years`. Les types de données et les littéraux interval peuvent être utilisés dans les calculs de date/heure, tels que l'ajout d'intervalles aux dates et aux horodatages, la somme des intervalles et la soustraction d'un intervalle d'une date ou d'un horodatage. Les littéraux interval peuvent être utilisés comme valeurs d'entrée pour intercaler les colonnes de type de données d'une table.

Syntaxe du type de données interval

Pour spécifier un type de données interval afin de stocker une durée en années et en mois :

```
INTERVAL year_to_month_qualifier
```

Pour spécifier un type de données interval afin de stocker une durée en jours, heures, minutes et secondes :

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

Syntaxe du littéral interval

Pour spécifier un littéral interval afin de définir une durée en années et en mois :

```
INTERVAL quoted-string year_to_month_qualifier
```

Pour spécifier un littéral interval afin de définir une durée en jours, heures, minutes et secondes :

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

Arguments

quoted-string

Spécifie une valeur numérique positive ou négative spécifiant une quantité et l'unité date/heure en tant que chaîne d'entrée. Si la chaîne entre guillemets ne contient qu'un chiffre, elle AWS Clean Rooms détermine les unités à partir du qualificatif year_to_month_qualifier ou du day_to_second_qualifier. Par exemple, `'23' MONTH` représente `1 year 11 months`, `'-2' DAY` représente `-2 days 0 hours 0 minutes 0.0 seconds`, `'1-2' MONTH` représente `1 year 2 months` et `'13 day 1 hour 1 minute 1.123 seconds' SECOND` représente `13 days 1 hour 1 minute 1.123 seconds`. Pour plus d'informations sur les formats de sortie d'un intervalle, consultez Styles d'intervalle.

year_to_month_qualifier

Spécifie la plage de l'intervalle. Si vous utilisez un qualificatif et que vous créez un intervalle dont les unités de temps sont inférieures au qualificatif, AWS Clean Rooms les plus petites parties de l'intervalle sont tronquées et supprimées. Les valeurs valides pour year_to_month_qualifier sont les suivantes :

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

day_to_second_qualifier

Spécifie la plage de l'intervalle. Si vous utilisez un qualificatif et que vous créez un intervalle dont les unités de temps sont inférieures au qualificatif, AWS Clean Rooms les plus petites parties de l'intervalle sont tronquées et supprimées. Les valeurs valides pour day_to_second_qualifier sont les suivantes :

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

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

La sortie du littéral INTERVAL est tronquée au plus petit composant INTERVAL spécifié. Par exemple, lorsque vous utilisez un qualificatif MINUTE, AWS Clean Rooms les unités de temps inférieures à MINUTE sont supprimées.

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

La valeur résultante est tronquée à `'1 day 01:01:00'`.

fractional_precision

Paramètre facultatif qui spécifie le nombre de chiffres fractionnaires autorisés dans l'intervalle. L'argument fractional_precision ne doit être spécifié que si votre intervalle contient SECOND. Par exemple, `SECOND(3)` crée un intervalle qui n'autorise que trois chiffres fractionnaires, tels que 1,234 seconde. Le nombre maximum de chiffres fractionnaires est de six.

La configuration de session `interval_forbid_composite_literals` détermine si une erreur est renvoyée lorsqu'un intervalle est spécifié avec les parties YEAR TO MONTH et DAY TO SECOND.

Arithmétique des intervalles

Vous pouvez utiliser des valeurs interval avec d'autres valeurs datetime pour effectuer des opérations arithmétiques. Les tableaux suivants décrivent les opérations disponibles et le type de données résultant de chaque opération.

> **Note**
>
> Les opérations qui peuvent produire les résultats `date` et `timestamp` le font en fonction de la plus petite unité de temps impliquée dans l'équation. Par exemple, lorsque vous ajoutez un `interval` à une `date`, le résultat est une `date`, s'il s'agit d'un intervalle YEAR TO MONTH, et un horodatage s'il s'agit d'un intervalle DAY TO SECOND.

Les opérations où le premier opérande est un `interval` produisent les résultats suivants pour le second opérande donné :

Opérateur	Date	Horodatage	Interval	Numérique
-	N/A	N/A	Interval	N/A
+	Date	Date/Horodatage.	Interval	N/A
*	N/A	N/A	N/A	Interval
/	N/A	N/A	N/A	Interval

Les opérations où le premier opérande est une `date` produisent les résultats suivants pour le second opérande donné :

Opérateur	Date	Horodatage	Interval	Numérique
-	Numérique	Interval	Date/Horodatage.	Date
+	N/A	N/A	N/A	N/A

Les opérations où le premier opérande est une `timestamp` produisent les résultats suivants pour le second opérande donné :

Opérateur	Date	Horodatage	Interval	Numérique
-	Numérique	Interval	Horodatage	Horodatage
+	N/A	N/A	N/A	N/A

Styles d'intervalle

- `postgres` : suit le style PostgreSQL. Il s'agit de l'option par défaut.
- `postgres_verbose` : suit le style détaillé PostgreSQL.

- `sql_standard` : suit le style des littéraux interval SQL standard.

La commande suivante définit le style d'intervalle sur `sql_standard`.

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

format de sortie postgres

Le format de sortie pour le style d'intervalle `postgres` est le suivant. Chaque valeur numérique peut être négative.

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

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

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

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

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

format de sortie postgres_verbose

La syntaxe postgres_verbose est similaire à postgres, mais les sorties postgres_verbose contiennent également l'unité de temps.

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

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

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

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

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

format de sortie sql_standard

Les valeurs interval year to month sont formatées comme suit. La spécification d'un signe négatif avant l'intervalle indique que l'intervalle est une valeur négative et s'applique à l'ensemble de l'intervalle.

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

Les valeurs interval day to second sont formatées comme suit.

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

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

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

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

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

Exemples de type de données interval

Les exemples suivants montrent comment utiliser les types de données INTERVAL avec des tables.

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

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

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

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

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

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

Exemples de littéraux interval

Les exemples suivants sont exécutés avec le style d'intervalle défini sur `postgres`.

L'exemple suivant montre comment créer un littéral INTERVAL de 1 an.

```
select INTERVAL '1' YEAR

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

Si vous spécifiez une quoted-string qui dépasse le qualificatif, les unités de temps restantes sont tronquées par rapport à l'intervalle. Dans l'exemple suivant, un intervalle de 13 mois devient 1 an et 1 mois, mais le mois restant est omis en raison du qualificatif YEAR.

```
select INTERVAL '13 months' YEAR

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

Si vous utilisez un qualificatif inférieur à votre chaîne d'intervalle, les unités restantes sont incluses.

```
select INTERVAL '13 months' MONTH

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

Si vous spécifiez une précision dans votre intervalle, le nombre de chiffres fractionnaires est tronqué à la précision spécifiée.

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

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

Si vous ne spécifiez aucune précision, AWS Clean Rooms utilise la précision maximale de 6.

```
select INTERVAL '1.23456789' SECOND

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

L'exemple suivant montre comment créer un intervalle par plage.

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

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

Les qualificatifs dictent les unités que vous spécifiez. Par exemple, même si l'exemple suivant utilise la même chaîne entre guillemets de « 2:2 » que l'exemple précédent, il AWS Clean Rooms reconnaît qu'il utilise des unités de temps différentes en raison du qualificatif.

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

intervald2s
-----------------------------
```

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

Les abréviations et les pluriels de chaque unité sont également pris en charge. Par exemple, `5s`, `5 second` et `5 seconds` sont des intervalles équivalents. Les unités prises en charge sont les années, les mois, les heures, les minutes et les secondes.

```
select INTERVAL '5s' SECOND

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

```
select INTERVAL '5 HOURS' HOUR

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

```
select INTERVAL '5 h' HOUR

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

Exemples de littéraux interval sans syntaxe de qualificatif

> **Note**
>
> Les exemples suivants illustrent l'utilisation d'un littéral interval sans qualificatif `YEAR TO MONTH` ou `DAY TO SECOND`. Pour plus d'informations sur l'utilisation du littéral interval recommandé avec un qualificatif, consultez Types de données et littéraux interval.

Utilisez un littéral de type interval pour identifier les périodes spécifiques, comme `12 hours` ou `6 months`. Vous pouvez utiliser ces littéraux de type interval dans les cas et les calculs qui impliquent des expressions de type datetime.

Un littéral interval est exprimé comme la combinaison du mot clé INTERVAL avec une quantité numérique et d'une partie date prise en charge ; par exemple : `INTERVAL '7 days'` ou `INTERVAL`

`'59 minutes'`. Plusieurs quantités et unités peuvent être associées pour former un intervalle plus précis ; par exemple : `INTERVAL '7 days, 3 hours, 59 minutes'`. Les abréviations et les pluriels de chaque unité sont également pris en charge ; par exemple : `5 s`, `5 second` et `5 seconds` sont des intervalles équivalents.

Si vous ne spécifiez pas une partie date, la valeur de l'intervalle correspond à des secondes. Vous pouvez spécifier la valeur de la quantité sous forme de fraction (par exemple : `0.5 days`).

Les exemples suivants illustrent une série de calculs avec différentes valeurs d'intervalle.

Ce qui suit ajoute 1 seconde à la date spécifiée.

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

Ce qui suit ajoute 1 minute à la date spécifiée.

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

Ce qui suit ajoute 3 heures et 35 minutes à la date spécifiée.

```
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)
```

Ce qui suit ajoute 52 semaines à la date spécifiée.

```
select caldate + interval '52 weeks' as dateplus from date
```

```
where caldate='12-31-2008';
dateplus
---------------------
2009-12-30 00:00:00
(1 row)
```

Ce qui suit ajoute 1 semaine, 1 heure, 1 minute, et 1 seconde à la date spécifiée.

```
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)
```

Ce qui suit ajoute 12 heures (la moitié d'une journée) à la date spécifiée.

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

Ce qui suit soustrait 4 mois à compter du 15 février 2023 et le résultat est le 15 octobre 2022.

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

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

Ce qui suit soustrait 4 mois à compter du 31 mars 2023 et le résultat est le 30 novembre 2022. Le calcul prend en compte le nombre de jours dans un mois.

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

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

Type Boolean

Utilisez le type de données BOOLEAN pour stocker les valeurs true et false dans une colonne codée sur un octet. Le tableau suivant décrit les trois états possibles pour une valeur booléenne et les valeurs littérales qui entraînent cet état. Quelle que soit la chaîne en entrée, une colonne booléenne stocke et émet « t » pour true et « f » pour false.

State	Valeurs littérales valides	Stockage
True	`TRUE 't' 'true' 'y' 'yes' '1'`	1 octet
False	`FALSE 'f' 'false' 'n' 'no' '0'`	1 octet
Je ne sais pas	`NULL`	1 octet

Vous pouvez utiliser une comparaison IS pour vérifier une valeur booléenne uniquement sous la forme d'un prédicat dans la clause WHERE. Vous ne pouvez pas utiliser la comparaison IS avec une valeur booléenne dans la liste SELECT.

Exemples

Vous pouvez utiliser une colonne BOOLEAN pour enregistrer un état « actif/inactif » pour chaque client dans une table CUSTOMER.

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

Dans cet exemple, la requête suivante sélectionne les utilisateurs du tableau USERS qui aiment le sport mais pas le théâtre :

```
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)
```

L'exemple suivant sélectionne les utilisateurs de la table USERS pour lesquels on ignore s'ils aiment la musique rock.

```
select firstname, lastname, likerock
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
----------+----------+----------
Alejandro | Rosalez  |
Carlos    | Salazar  |
Diego     | Ramirez  |
John      | Stiles   |
Kwaku     | Mensah   |
Martha    | Rivera   |
Mateo     | Jackson  |
Paulo     | Santos   |
Richard   | Roe      |
Saanvi    | Sarkar   |
(10 rows)
```

L'exemple suivant renvoie une erreur parce qu'il utilise une comparaison IS dans la liste SELECT.

```
select firstname, lastname, likerock is true as "check"
```

```
from users
order by userid limit 10;

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

L'exemple suivant réussit car il utilise une comparaison égale (=) dans la liste SELECT au lieu de la IS comparaison.

```
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    |
```

Littéraux booléens

Les règles suivantes concernent l'utilisation des littéraux booléens pris en charge par AWS Clean Rooms Spark SQL.

Utilisez un littéral booléen pour spécifier une valeur booléenne, telle que ou. `TRUE` `FALSE`

Syntaxe

```
TRUE | FALSE
```

Exemple

L'exemple suivant montre une colonne avec une valeur spécifiée de`TRUE`.

```
SELECT TRUE AS col;
```

```
+----+
| col|
+----+
|true|
+----+
```

Type binaire

Utilisez le type de données BINARY pour stocker et gérer des données binaires de longueur fixe et non interprétées, offrant ainsi des fonctionnalités de stockage et de comparaison efficaces pour des cas d'utilisation spécifiques.

Le type de données BINARY stocke un nombre fixe d'octets, quelle que soit la longueur réelle des données stockées. La longueur maximale est généralement de 255 octets.

BINARY est utilisé pour stocker des données binaires brutes non interprétées, telles que des images, des documents ou d'autres types de fichiers. Les données sont stockées exactement telles qu'elles sont fournies, sans codage de caractères ni interprétation. Les données binaires stockées dans des colonnes BINARY sont comparées et triées byte-by-byte en fonction des valeurs binaires réelles, plutôt que des règles de codage de caractères ou de classement.

L'exemple de requête suivant montre la représentation binaire de la chaîne`"abc"`. Chaque caractère de la chaîne est représenté par son code ASCII au format hexadécimal : « a » est 0x61, « b » est 0x62 et « c » est 0x63. Lorsqu'elles sont combinées, ces valeurs hexadécimales forment la représentation binaire`"616263"`.

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

Type imbriqué

AWS Clean Rooms prend en charge les requêtes impliquant des données avec des types de données imbriqués, en particulier les types de colonnes AWS Glue STRUCT, ARRAY et MAP. Seule la règle d'analyse personnalisée prend en charge les types de données imbriqués.

Les types de données imbriqués ne sont notamment pas conformes à la structure tabulaire rigide du modèle de données relationnel des bases de données SQL.

Les types de données imbriqués contiennent des balises qui font référence à des entités distinctes au sein des données. Elles peuvent contenir des valeurs complexes telles que des tableaux, des structures imbriquées et d'autres structures complexes associées à des formats de sérialisation, tels que JSON. Les types de données imbriqués prennent en charge jusqu'à 1 Mo de données pour un champ ou un objet de type de données imbriqué individuel.

Rubriques

Type de TABLEAU

Utilisez le type ARRAY pour représenter des valeurs comprenant une séquence d'éléments de type`elementType`.

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

`containsNull`À utiliser pour indiquer si les éléments d'un type ARRAY peuvent avoir des `null` valeurs.

Type de carte

Utilisez le type MAP pour représenter des valeurs comprenant un ensemble de paires clé-valeur.

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

`keyType`: le type de données des clés

`valueType`: le type de données des valeurs

Les clés ne sont pas autorisées à avoir `null` des valeurs. `valueContainsNull`À utiliser pour indiquer si les valeurs de type MAP peuvent avoir des `null` valeurs.

Type de STRUCTURE

Utilisez le type STRUCT pour représenter des valeurs avec la structure décrite par une séquence de StructFields (champs).

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

StructField(nom, DataType, nullable) : représente un champ dans un. StructType

`dataType`: le type de données est un champ

`name`: le nom d'un champ

`nullable`À utiliser pour indiquer si les valeurs de ces champs peuvent avoir des `null` valeurs.

Exemples de types de données imbriqués

Pour le `struct<given:varchar, family:varchar>` type, il existe deux noms d'attributs :`given`, et`family`, chacun correspondant à une `varchar` valeur.

Pour le `array<varchar>` type, le tableau est spécifié sous forme de liste de`varchar`.

Le `array<struct<shipdate:timestamp, price:double>>` type fait référence à une liste d'éléments de `struct<shipdate:timestamp, price:double>` type.

Le type de `map` données se comporte comme un `array` de`structs`, où le nom d'attribut de chaque élément du tableau est indiqué par `key` et correspond à un. `value`

Example

Par exemple, le `map<varchar(20), varchar(20)>` type est traité comme`array<struct<key:varchar(20), value:varchar(20)>>`, où `key` et fait `value` référence aux attributs de la carte dans les données sous-jacentes.

Pour plus d'informations sur le mode AWS Clean Rooms d'activation de la navigation dans les tableaux et les structures, consultezNavigation.

Pour plus d'informations sur la manière AWS Clean Rooms d'activer l'itération sur des tableaux en naviguant dans le tableau à l'aide de la clause FROM d'une requête, consultez. Désimbriquer des requêtes

Compatibilité et conversion de types

Les rubriques suivantes décrivent le fonctionnement des règles de conversion de type et de compatibilité des types de données dans AWS Clean Rooms Spark SQL.

Rubriques

Compatibilité

La correspondance des types de données et la correspondance des valeurs littérales et des constantes avec les types de données se produisent lors de différentes opérations de base de données, dont les suivantes :

- Opérations DML (Data Manipulation Language) sur les tables
- Requêtes UNION, INTERSECT et EXCEPT
- Expressions CASE
- Evaluation de prédicats, tels que LIKE et IN
- Evaluation de fonctions SQL qui effectuent des comparaisons ou des extractions de données
- Comparaisons avec les opérateurs mathématiques

Les résultats de ces opérations dépendent des règles de conversion de types et de la compatibilité des types de données. La compatibilité implique que la mise en one-to-one correspondance d'une certaine valeur et d'un certain type de données n'est pas toujours requise. Certains types de données étant compatibles, une conversion implicite, ou coercition, est possible. Pour de plus amples informations, veuillez consulter Types de conversion implicite. Lorsque les types de données sont incompatibles, vous pouvez parfois convertir une valeur d’un type de données en un autre à l’aide d’une fonction de conversion explicite.

Compatibilité générale et règles de conversion

Notez les règles de compatibilité et de conversion suivantes :

- En général, les types de données qui appartiennent à la même catégorie (comme les différents types de données numériques) sont compatibles et peuvent être convertis implicitement.

  Par exemple, avec une conversion implicite, vous pouvez insérer une valeur décimale dans une colonne de type entier. La partie décimale est arrondie pour produire un nombre entier. Ou vous pouvez extraire une valeur numérique, telle que `2008`, d’une date et insérer cette valeur dans une colonne de type entier.

- Les types de données numériques renforcent les conditions de débordement qui se produisent lorsque vous tentez d'insérer out-of-range des valeurs. Par exemple, une valeur décimale avec une précision de 5 ne peut contenir dans une colonne décimale dont la précision est 4. Un entier ou la partie entière d'un nombre décimal n'est jamais tronqué. Cependant, la partie fractionnaire d'une décimale peut être arrondie à la hausse ou à la baisse, selon le cas. Cependant, les résultats de conversions explicites de valeurs sélectionnées dans les tables ne sont pas arrondis.
- Différents types de chaînes de caractères sont compatibles. Les chaînes de colonne VARCHAR contenant des données à un octet et les chaînes de colonnes CHAR sont comparables et implicitement convertibles. Les chaînes VARCHAR qui contiennent des données codées sur plusieurs octets ne sont pas comparables. Vous pouvez également convertir une chaîne de caractères en date, heure, horodatage ou valeur numérique si la chaîne est une valeur littérale appropriée. Les espaces de début ou de fin sont ignorés. Inversement, vous pouvez convertir une date, une heure, un horodatage ou une valeur numérique en une chaîne de caractères de longueur fixe ou variable.

  > **Note**
  >
  > Une chaîne de caractères que vous voulez convertir en type numérique doit comporter la représentation en caractères d'un nombre. Par exemple, vous pouvez convertir les chaînes `'1.0'` ou `'5.9'` en valeurs décimales, mais vous ne pouvez pas convertir la chaîne `'ABC'` en un type numérique.

- Si vous comparez des valeurs DECIMAL à des chaînes de caractères, AWS Clean Rooms tente de convertir la chaîne de caractères en valeur DECIMAL. Lors de la comparaison de toutes les autres valeurs numériques avec des chaînes de caractères, les valeurs numériques sont converties en chaînes de caractères. Pour effectuer la conversion inverse (par exemple, convertir des chaînes de caractères en entiers ou convertir des valeurs DECIMALES en chaînes de caractères), utilisez une fonction explicite, telle que Fonction CAST.
- Pour convertir les valeurs DECIMAL ou NUMERIC 64 bits en une plus grande précision, vous devez utiliser une fonction de conversion explicite telle que les fonctions CAST ou CONVERT.

Types de conversion implicite

Il existe deux types de conversion implicite :

- Conversions implicites dans les affectations, telles que la définition de valeurs dans les commandes INSERT ou UPDATE

- Conversions implicites dans les expressions, telles que les comparaisons dans la clause WHERE

Le tableau suivant répertorie les types de données qui peuvent être convertis implicitement dans des assignations ou des expressions. Vous pouvez également utiliser une fonction de conversion explicite pour exécuter ces conversions.

Type de départ	Type d'arrivée
BIGINT	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRÉCISION (FLOAT8)
	INTEGER
	RÉEL (FLOAT4)
	SMALLINT ou SHORT
	VARCHAR
CHAR	VARCHAR
DATE	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT ou LONG
	CHAR
	DOUBLE PRÉCISION (FLOAT8)
	ENTIER (INT)

Type de départ	Type d’arrivée
	RÉEL (FLOAT4)
	SMALLINT ou SHORT
	VARCHAR
DOUBLE PRÉCISION (FLOAT8)	BIGINT ou LONG
	CHAR
	DECIMAL (NUMERIC)
	ENTIER (INT)
	RÉEL (FLOAT4)
	SMALLINT ou SHORT
	VARCHAR
ENTIER (INT)	BIGINT ou LONG
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRÉCISION (FLOAT8)
	RÉEL (FLOAT4)
	SMALLINT ou SHORT
	VARCHAR
RÉEL (FLOAT4)	BIGINT ou LONG
	CHAR

Type de départ	Type d'arrivée
	DECIMAL (NUMERIC)
	ENTIER (INT)
	SMALLINT ou SHORT
	VARCHAR
SMALLINT	BIGINT ou LONG
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRÉCISION (FLOAT8)
	ENTIER (INT)
	RÉEL (FLOAT4)
	VARCHAR
TIME	VARCHAR
	TIMETZ

Note

Les conversions implicites entre DATE, TIME, TIMESTAMP_LTZ, TIMESTAMP_NTZ ou des chaînes de caractères utilisent le fuseau horaire de la session en cours.
Le type de données VARBYTE ne peut pas être converti de façon implicite dans un autre type de données. Pour de plus amples informations, veuillez consulter Fonction CAST.

AWS Clean Rooms Commandes SQL Spark

Les commandes SQL suivantes sont prises en charge dans AWS Clean Rooms Spark SQL :

Rubriques

TABLE DE CACHE

La commande CACHE TABLE met en cache les données d'une table existante ou crée et met en cache une nouvelle table contenant les résultats des requêtes.

> **Note**
>
> Les données mises en cache sont conservées pendant toute la durée de la requête.

La syntaxe, les arguments et quelques exemples proviennent de la référence SQL d'Apache Spark.

Syntaxe

La commande CACHE TABLE prend en charge trois modèles de syntaxe :

Avec AS (sans parenthèses) : crée et met en cache une nouvelle table en fonction des résultats de la requête.

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

Avec AS et parenthèses : fonctionne de la même manière que la première syntaxe mais utilise des parenthèses pour regrouper explicitement la requête.

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

Sans AS : met en cache une table existante à l'aide de l'instruction SELECT pour filtrer les lignes à mettre en cache.

```
CACHE TABLE cache_table_identifier query;
```

Où :

- Toutes les instructions doivent se terminer par un point-virgule (;)
- `query`est généralement une instruction SELECT
- Les parenthèses autour de la requête sont facultatives avec AS
- Le mot clé AS est facultatif

Parameters

identificateur de table de cache

Nom de la table mise en cache. Peut inclure un qualificatif de nom de base de données facultatif.

EN TANT QUE

Mot-clé utilisé lors de la création et de la mise en cache d'une nouvelle table à partir des résultats d'une requête.

query

Une instruction SELECT ou une autre requête qui définit les données à mettre en cache.

Exemples

Dans les exemples suivants, la table mise en cache est conservée pendant toute la durée de la requête. Après la mise en cache, les requêtes suivantes faisant référence *`cache_table_identifier`* seront lues à partir de la version mise en cache plutôt que de recalculer ou de lire à partir de celle-ci. *`sourceTable`* Cela peut améliorer les performances des requêtes pour les données fréquemment consultées.

Création et mise en cache d'une table filtrée à partir des résultats de requête

Le premier exemple montre comment créer et mettre en cache une nouvelle table à partir des résultats d'une requête. Cette commande utilise le `AS` mot clé sans parenthèses autour de l'`SELECT`instruction. Il crée une nouvelle table nommée « `cache_table_identifier` » contenant uniquement les lignes de « `sourceTable` » dont le statut est « »`active'`. Il exécute la requête, stocke les résultats dans la nouvelle table et met en cache le contenu de la nouvelle table. Le

« sourceTable » d'origine reste inchangé, et les requêtes suivantes doivent faire référence à « cache_table_identifier » pour utiliser les données mises en cache.

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

Mettre en cache les résultats des requêtes avec des instructions SELECT entre parenthèses

Le deuxième exemple montre comment mettre en cache les résultats d'une requête sous la forme d'une nouvelle table portant le nom spécifié (cache_table_identifier), en plaçant l'SELECTinstruction entre parenthèses. Cette commande crée une nouvelle table nommée « cache_table_identifier » contenant uniquement les lignes de « sourceTable » dont le statut est « »active'. Il exécute la requête, stocke les résultats dans la nouvelle table et met en cache le contenu de la nouvelle table. Le « sourceTable » original reste inchangé. Les requêtes suivantes doivent faire référence à cache_table_identifier « » pour utiliser les données mises en cache.

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

Mettre en cache une table existante avec des conditions de filtre

Le troisième exemple montre comment mettre en cache une table existante en utilisant une syntaxe différente. Cette syntaxe, qui omet le mot clé « AS » et les parenthèses, met généralement en cache les lignes spécifiées dans une table existante nommée « cache_table_identifier » plutôt que de créer une nouvelle table. L'SELECTinstruction agit comme un filtre pour déterminer les lignes à mettre en cache.

> **Note**
>
> Le comportement exact de cette syntaxe varie selon les systèmes de base de données. Vérifiez toujours la syntaxe correcte pour votre AWS service spécifique.

```
CACHE TABLE cache_table_identifier
```

```
SELECT * FROM sourceTable
WHERE status = 'active';
```

Indicateurs

Les astuces pour les analyses SQL fournissent des directives d'optimisation qui guident les stratégies d'exécution des requêtes AWS Clean Rooms, vous permettant ainsi d'améliorer les performances des requêtes et de réduire les coûts de calcul. Des indices suggèrent comment le moteur d'analyse Spark doit générer son plan d'exécution.

Syntaxe

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

Les indices sont intégrés aux requêtes SQL à l'aide d'une syntaxe de type commentaire et doivent être placés directement après le mot clé SELECT.

Types d'indices pris en charge

AWS Clean Rooms prend en charge deux catégories d'astuces : les astuces de jointure et les astuces de partitionnement.

Rubriques

- Conseils de participation
- Conseils de partitionnement

Conseils de participation

Les conseils de jointure suggèrent des stratégies de jointure pour l'exécution des requêtes. La syntaxe, les arguments et quelques exemples proviennent de la référence SQL Apache Spark pour plus d'informations

DIFFUSER

Suggère d' AWS Clean Rooms utiliser la jointure par diffusion. La page de jointure contenant l'indice sera diffusée quel que soit le autoBroadcastJoin seuil. Si les deux côtés de la jointure ont des indices de diffusion, celui dont la taille est la plus petite (sur la base des statistiques) sera diffusé.

Alias : BROADCASTJOIN, MAPJOIN

Paramètres : identificateurs de table (facultatif)

Exemples :

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

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

MERGE

Suggère d' AWS Clean Rooms utiliser le shuffle, le tri, la fusion, la jointure.

Alias : SHUFFLE_MERGE, MERGEJOIN

Paramètres : identificateurs de table (facultatif)

Exemples :

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

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

SHUFFLE_HASH

Suggère d' AWS Clean Rooms utiliser la jointure par hachage automatique. Si les deux côtés ont des indices de hachage combinés, l'optimiseur de requêtes choisit le côté le plus petit (basé sur les statistiques) comme côté build.

Paramètres : identificateurs de table (facultatif)

Exemples :

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

SHUFFLE_REPLICATE_FR

Suggère d' AWS Clean Rooms utiliser la jointure shuffle-and-replicate par boucle imbriquée.

Paramètres : identificateurs de table (facultatif)

Exemples :

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

Conseils de résolution des problèmes dans Spark SQL

Le tableau suivant montre les scénarios courants dans lesquels les indices ne sont pas appliqués dans SparkSQL. Pour plus d'informations, consultez the section called “Considérations et restrictions”.

Cas d'utilisation	Exemple de requête
Référence de table introuvable	`SELECT /*+ BROADCAST(fake_table) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
La table ne participe pas à l'opération de jointure	`SELECT /*+ BROADCAST(s) */ * FROM students s WHERE s.age > 25;`
Référence de table dans une sous-requête imbriquée	`SELECT /*+ BROADCAST(s) */ * FROM employees e INNER JOIN (SELECT * FROM students s WHERE s.age > 20) sub ON e.eid = sub.sid;`

Cas d'utilisation	Exemple de requête
Nom de colonne au lieu de référence de table	`SELECT /*+ BROADCAST(e.eid) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Conseil sans paramètres obligatoires	`SELECT /*+ BROADCAST */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`
Nom de la table de base au lieu de l'alias de la table	`SELECT /*+ BROADCAST(employees) */ * FROM employees e INNER JOIN students s ON e.eid = s.sid;`

Conseils de partitionnement

Les conseils de partitionnement contrôlent la distribution des données entre les nœuds exécuteurs. Lorsque plusieurs conseils de partitionnement sont spécifiés, plusieurs nœuds sont insérés dans le plan logique, mais l'indicateur le plus à gauche est sélectionné par l'optimiseur.

COALESCE

Réduit le nombre de partitions au nombre de partitions spécifié.

Paramètres : valeur numérique (obligatoire) - doit être un entier positif compris entre 1 et 2147483647

Exemples :

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

RÉPARTITION

Repartitionne les données sur le nombre de partitions spécifié à l'aide des expressions de partitionnement spécifiées. Utilise la distribution circulaire.

Paramètres :

- Valeur numérique (facultatif) - nombre de partitions ; doit être un entier positif compris entre 1 et 2147483647
- Identifiants de colonne (facultatif) - colonnes à partitionner ; ces colonnes doivent exister dans le schéma d'entrée.
- Si les deux sont spécifiés, la valeur numérique doit apparaître en premier

Exemples :

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

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

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

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

RÉPARTITION PAR PLAGE

Repartitionne les données sur le nombre de partitions spécifié en utilisant le partitionnement par plage sur les colonnes spécifiées.

Paramètres :

- Valeur numérique (facultatif) - nombre de partitions ; doit être un entier positif compris entre 1 et 2147483647
- Identifiants de colonne (facultatif) - colonnes à partitionner ; ces colonnes doivent exister dans le schéma d'entrée.
- Si les deux sont spécifiés, la valeur numérique doit apparaître en premier

Exemples :

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

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

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

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

RÉÉQUILIBRER

Rééquilibre les partitions de sortie des résultats de la requête afin que chaque partition soit d'une taille raisonnable (ni trop petite ni trop grande). Il s'agit d'une opération très simple : s'il y a des biais, les partitions asymétriques AWS Clean Rooms seront divisées pour qu'elles ne soient pas trop grandes. Cette astuce est utile lorsque vous devez écrire le résultat d'une requête dans une table afin d'éviter des fichiers trop petits ou trop volumineux.

Paramètres :

- Valeur numérique (facultatif) - nombre de partitions ; doit être un entier positif compris entre 1 et 2147483647
- Identifiants de colonne (facultatif) : les colonnes doivent apparaître dans la liste de sortie SELECT
- Si les deux sont spécifiés, la valeur numérique doit apparaître en premier

Exemples :

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

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

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

Combiner plusieurs astuces

Vous pouvez spécifier plusieurs indices dans une seule requête en les séparant par des virgules :

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

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

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

Considérations et restrictions

- Les indices sont des suggestions d'optimisation et non des commandes. L'optimiseur de requêtes peut ignorer les indications basées sur des contraintes de ressources ou des conditions d'exécution.
- Les conseils sont intégrés directement dans les chaînes de requête SQL pour les deux CreateAnalysisTemplate et StartProtectedQuery APIs.
- Les indices doivent être placés directement après le mot clé SELECT.
- Les paramètres nommés ne sont pas pris en charge par des indices et déclencheront une exception.
- Les noms de colonnes dans les indications REPARTITION et REPARTITION_BY_RANGE doivent figurer dans le schéma d'entrée.
- Les noms des colonnes figurant dans les indices REBALANCE doivent apparaître dans la liste de sortie SELECT.
- Les paramètres numériques doivent être des entiers positifs compris entre 1 et 2147483647. Les notations scientifiques telles que 1e1 ne sont pas prises en charge

- Les indices ne sont pas pris en charge dans les requêtes SQL Differential Privacy.
- Les astuces pour les requêtes SQL ne sont pas prises en charge dans les PySpark jobs. Pour fournir des directives pour les plans d'exécution d'une PySpark tâche, utilisez l'API de trame de données. Consultez la documentation de DataFrame l'API Apache Spark pour plus d'informations.

SELECT

La commande SELECT renvoie des lignes provenant de tables et de fonctions définies par l'utilisateur.

Les commandes, clauses et opérateurs d'ensemble SELECT SQL suivants sont pris en charge dans AWS Clean Rooms Spark SQL :

Rubriques

- SELECT list
- Clause WITH
- Clause FROM
- Clause JOIN
- Clause WHERE
- Clause VALUES
- Clause GROUP BY
- Clause HAVING
- Définir les opérateurs
- Clause ORDER BY
- Exemples de sous-requête
- Sous-requêtes corrélées

La syntaxe, les arguments et quelques exemples proviennent de la référence SQL d'Apache Spark.

SELECT list

Les SELECT list noms des colonnes, des fonctions et des expressions que vous souhaitez renvoyer par la requête. La liste représente le résultat de la requête.

Syntaxe

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

Parameters

DISTINCT

Option qui élimine les lignes en double du jeu de résultats, en fonction de la correspondance des valeurs dans une ou plusieurs colonnes.

expression

Expression formée d'une ou de plusieurs colonnes qui existent dans les tables référencées par la requête. Une expression peut contenir des fonctions SQL. Par exemple :

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

AS column_alias

Nom temporaire de la colonne utilisé dans le jeu de résultats final. Le AS mot clé est facultatif. Par exemple :

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

Si vous ne spécifiez pas un alias pour une expression qui n'est pas un nom de colonne simple, le jeu de résultats applique un nom par défaut à cette colonne.

> **Note**
>
> L'alias est reconnu juste après sa définition dans la liste cible. Vous ne pouvez pas utiliser d'alias dans d'autres expressions définies après lui dans la même liste de cibles.

Clause WITH

Une clause WITH est une clause facultative qui précède la liste SELECT d'une requête. La clause WITH définit une ou plusieurs expressions common_table_expressions. Chaque expression de

table commune (CTE) définit une table temporaire, qui est similaire à la définition d'une vue. Vous pouvez référencer ces tables temporaires dans la clause FROM. Elles ne sont utilisées que pendant l'exécution de la requête à laquelle elles appartiennent. Chaque CTE de la clause WITH spécifie un nom de table, une liste facultative de noms de colonne et une expression de requête correspondant à une table (instruction SELECT).

Les sous-requêtes de clause WITH sont un moyen efficace de définir les tables qui peuvent être utilisées tout au long de l'exécution d'une même requête. Dans tous les cas, les mêmes résultats peuvent être obtenus à l'aide de sous-requêtes dans le corps principal de l'instruction SELECT, mais les sous-requêtes de clause WITH peuvent être plus simples à lire et à écrire. Chaque fois que possible, les sous-requêtes de clause WITH qui sont référencées plusieurs fois sont optimisées en tant que sous-expressions courantes ; autrement dit, il peut être possible d'évaluer une sous-requête WITH une fois et de réutiliser ses résultats. (Notez que les sous-expressions courantes ne sont pas limitées à celles définies dans la clause WITH).

Syntaxe

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

où common_table_expression peut être non récursive. Voici la forme non-récursive :

```
CTE_table_name AS ( query )
```

Parameters

common_table_expression

Définit une table temporaire que vous pouvez référencer dans Clause FROM et qui n'est utilisée que pendant l'exécution de la requête à laquelle elle appartient.

CTE_table_name

Nom unique d'une table temporaire qui définit les résultats d'une sous-requête de clause WITH. Vous ne pouvez pas utiliser de noms en double au sein d'une clause WITH. Chaque sous-requête doit avoir un nom de table qui peut être référencé dans la Clause FROM.

query

Toute requête SELECT qui AWS Clean Rooms prend en charge. Consultez SELECT.

Notes d’utilisation

Vous pouvez utiliser une clause WITH dans l'instruction SQL suivante :

- SÉLECTIONNER, AVEC, UNIR, UNIR TOUT, INTERSECTER, INTERSECTER TOUT, SAUF OU EXCEPTER TOUT

Si la clause FROM d’une requête qui contient une clause WITH ne fait pas référence à l’une des tables définies par la clause WITH, la clause WITH est ignorée et la requête s’exécute normalement.

Une table définie par une sous-requête de clause WITH peut être référencée uniquement dans la portée de la requête SELECT que commence la clause WITH. Par exemple, vous pouvez faire référence à une telle table dans la clause FROM d’une sous-requête de la liste SELECT, la clause WHERE ou la clause HAVING. Vous ne pouvez pas utiliser une clause WITH dans une sous-requête et faire référence à sa table dans la clause FROM de la requête principale ou d’une autre sous-requête. Ce modèle de requête entraîne un message d’erreur sous la forme `relation table_name doesn't exist` pour la table de la clause WITH.

Vous ne pouvez pas spécifier une autre clause WITH à l’intérieur d’une sous-requête de clause WITH.

Vous ne pouvez pas effectuer de références futures aux tables définies par des sous-requêtes de clause WITH. Par exemple, la requête suivante renvoie une erreur en raison de la référence future à la table W2 dans la définition de table W1 :

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

Exemples

L’exemple suivant illustre le cas le plus simple possible d’une requête contenant une clause WITH. La requête WITH nommée VENUECOPY sélectionne toutes les lignes de la table VENUE. La requête principale, à son tour, sélectionne toutes les lignes de VENUECOPY. La table VENUECOPY existe uniquement pendant la durée de cette requête.

```
with venuecopy as (select * from venue)
select * from venuecopy order by 1 limit 10;
```

```
 venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
1 | Toyota Park                | Bridgeview      | IL         |          0
2 | Columbus Crew Stadium      | Columbus        | OH         |          0
3 | RFK Stadium                | Washington      | DC         |          0
4 | CommunityAmerica Ballpark  | Kansas City     | KS         |          0
5 | Gillette Stadium           | Foxborough      | MA         |      68756
6 | New York Giants Stadium    | East Rutherford | NJ         |      80242
7 | BMO Field                  | Toronto         | ON         |          0
8 | The Home Depot Center      | Carson          | CA         |          0
9 | Dick's Sporting Goods Park | Commerce City   | CO         |          0
v     10 | Pizza Hut Park             | Frisco          | TX         |          0
(10 rows)
```

L'exemple suivant montre une clause WITH qui produit deux tables, nommées VENUE_SALES et TOP_VENUES. La deuxième table de requête WITH effectue la sélection à partir de la première. A son tour, la clause WHERE du bloc de requête principal contient une sous-requête qui restreint la table TOP_VENUES.

```
with venue_sales as
(select venuename, venuecity, sum(pricepaid) as venuename_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
group by venuename, venuecity),

top_venues as
(select venuename
from venue_sales
where venuename_sales > 800000)

select venuename, venuecity, venuestate,
sum(qtysold) as venue_qty,
sum(pricepaid) as venue_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
and venuename in(select venuename from top_venues)
group by venuename, venuecity, venuestate
order by venuename;
```

```
        venuename       |   venuecity   | venuestate | venue_qty | venue_sales
------------------------+---------------+------------+-----------+-------------
August Wilson Theatre   | New York City | NY         |      3187 |  1032156.00
```

```
Biltmore Theatre        | New York City | NY         |      2629 |   828981.00
Charles Playhouse       | Boston        | MA         |      2502 |   857031.00
Ethel Barrymore Theatre | New York City | NY         |      2828 |   891172.00
Eugene O'Neill Theatre  | New York City | NY         |      2488 |   828950.00
Greek Theatre           | Los Angeles   | CA         |      2445 |   838918.00
Helen Hayes Theatre     | New York City | NY         |      2948 |   978765.00
Hilton Theatre          | New York City | NY         |      2999 |   885686.00
Imperial Theatre        | New York City | NY         |      2702 |   877993.00
Lunt-Fontanne Theatre   | New York City | NY         |      3326 |  1115182.00
Majestic Theatre        | New York City | NY         |      2549 |   894275.00
Nederlander Theatre     | New York City | NY         |      2934 |   936312.00
Pasadena Playhouse      | Pasadena      | CA         |      2739 |   820435.00
Winter Garden Theatre   | New York City | NY         |      2838 |   939257.00
(14 rows)
```

Les deux exemples suivants illustrent les règles sur la portée des références de table dans les sous-requêtes de clause WITH. La première requête s'exécute, mais la deuxième échoue avec une erreur prévue. La première requête a une sous-requête de clause WITH à l'intérieur de la liste SELECT de la requête principale. La table définie par la clause WITH (HOLIDAYS) est référencée dans la clause FROM de la sous-requête de la liste SELECT :

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

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

La deuxième requête échoue, car elle tente de faire référence à la table HOLIDAYS de la requête principale, ainsi que dans la sous-requête de liste SELECT. Les références de requête principale sont hors de portée.

```
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
```

Clause FROM

La clause FROM d'une requête répertorie les références de table (tables, vues et sous-requêtes) à partir desquelles les données sont sélectionnées. Si plusieurs références de table sont répertoriées, les tables doivent être jointes, à l'aide de la syntaxe appropriée de la clause FROM ou de la clause WHERE. Si aucun critère de jointure n'est spécifié, le système traite la requête comme jointure croisée (produit cartésien).

Rubriques

- Syntaxe
- Parameters
- Notes d'utilisation

Syntaxe

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

où table_reference est l'une des références suivantes :

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

Parameters

with_subquery_table_name

Table définie par une sous-requête dans la Clause WITH.

table_name

Nom d’une table ou d’une vue.

alias

Nom alternatif temporaire d’une table ou d’une vue. Un alias doit être fourni pour une table dérivée d’une sous-requête. Dans les autres références de table, les alias sont facultatifs. Le AS mot clé est toujours facultatif. Les alias de table offrent un raccourci pratique pour identifier les tables dans d’autres parties d’une requête, telles que la clause WHERE.

Par exemple :

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

Si vous définissez un alias de table défini, l'alias doit être utilisé pour référencer cette table dans la requête.

Par exemple, si la requête l'est`SELECT "tbl"."col" FROM "tbl" AS "t"`, elle échouera car le nom de la table est désormais essentiellement remplacé. Dans ce cas, une requête valide serait`SELECT "t"."col" FROM "tbl" AS "t"`.

alias_colonne

Nom alternatif temporaire pour une colonne dans une table ou une vue.

sous-requête

Une expression de requête qui correspond à une table. La table existe uniquement pendant la durée de la requête et reçoit généralement un nom ou un alias. Toutefois, l’alias n’est pas obligatoire. Vous pouvez aussi définir des noms de colonnes pour les tables qui proviennent de sous-requêtes. Il est important de nommer les alias de colonne lorsque vous souhaitez joindre les résultats des sous-requêtes à d’autres tables et lorsque vous voulez sélectionner ou limiter les colonnes ailleurs dans la requête.

Une sous-requête peut contenir une clause ORDER BY, mais cette clause peut n’avoir aucun effet si une clause LIMIT ou OFFSET n’est pas également spécifiée.

NATURAL

Définit une jointure qui utilise automatiquement toutes les paires de colonnes portant le même nom dans les deux tables comme colonnes de jointure. Aucune condition de jointure explicite

n'est nécessaire. Par exemple, si les tables CATEGORY et EVENT ont toutes deux des colonnes nommées CATID, une jointure naturelle des tables est une jointure sur leurs colonnes CATID.

> **Note**
>
> Si une jointure NATURAL est spécifiée, mais qu'il n'y a aucune paire de colonnes portant le même nom dans les tables à joindre, la requête se résout par défaut en une jointure croisée.

join_type

Spécifiez l'un des types de jointure suivants :

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

Les jointures croisées sont des jointures non qualifiées ; elles renvoient le produit cartésien des deux tables.

Les jointures internes et externes sont des jointures qualifiées. Elles sont qualifiés implicitement (en jointures naturelles), avec la syntaxe ON ou USING de la clause FROM, ou avec une condition de clause WHERE.

Une jointure interne renvoie les lignes correspondantes uniquement, en fonction de la condition de jointure ou d'une liste de colonnes de jointure. Une jointure externe renvoie toutes les lignes que la jointure interne équivalente renvoyerait, plus les lignes non correspondantes de la table de « gauche », de la table de « droite » ou des deux tables. La table de gauche est la première table de la liste et la table de droite la deuxième table. Les lignes non correspondantes contiennent des valeurs NULL pour combler les écarts dans les colonnes de sortie.

ON condition_jointure

Type de spécification de jointure où les colonnes de jointure sont définies comme condition qui suit le mot-clé ON. Par exemple :

```
sales join listing
```

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

USING (colonne_jointure [, ...])

Type de spécification de jointure où les colonnes de jointure sont affichées entre parenthèses. Si plusieurs colonnes de jointure sont spécifiées, elles sont séparées par des virgules. Le mot-clé USING doit précéder la liste. Par exemple :

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

Notes d'utilisation

Les colonnes de jointure doivent avoir des types de données comparables.

Une jointure NATURAL ou USING conserve seulement l'une de chaque paire de colonnes de jointure dans le jeu de résultats intermédiaire.

Une jointure avec la syntaxe ON conserve les deux colonnes de jointure dans son jeu de résultats intermédiaire.

Consultez également Clause WITH.

Clause JOIN

Une clause SQL JOIN permet de combiner les données de deux ou plusieurs tables sur la base de champs communs. Les résultats peuvent ou non changer en fonction de la méthode de jointure spécifiée. Les jointures externes gauche et droite conservent les valeurs de l'une des tables jointes quand aucune correspondance n'est trouvée dans l'autre table.

La combinaison du type JOIN et de la condition de jointure détermine les lignes incluses dans le jeu de résultats final. Les clauses SELECT et WHERE contrôlent ensuite les colonnes renvoyées et la manière dont les lignes sont filtrées. Comprendre les différents types de JOIN et savoir comment les utiliser efficacement est une compétence cruciale en SQL, car cela vous permet de combiner les données de plusieurs tables de manière flexible et puissante.

Syntaxe

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

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

Parameters

SÉLECTIONNEZ la colonne 1, la colonne 2,..., la colonne N

Les colonnes que vous souhaitez inclure dans le jeu de résultats. Vous pouvez sélectionner des colonnes dans l'une ou l'autre des tables impliquées dans le JOIN ou dans les deux.

À PARTIR DU TABLEAU 1

La première table (à gauche) de l'opération JOIN.

[JOINTURE | JOINTURE INTÉRIEURE | JOINTURE GAUCHE [EXTÉRIEURE] | JOINTURE DROITE [EXTÉRIEURE] | JOINTURE [EXTÉRIEURE] COMPLÈTE] table2 :

Type de JOIN à exécuter. JOIN ou INNER JOIN renvoie uniquement les lignes dont les valeurs correspondent dans les deux tables.

LEFT [OUTER] JOIN renvoie toutes les lignes du tableau de gauche, avec les lignes correspondantes du tableau de droite.

RIGHT [OUTER] JOIN renvoie toutes les lignes du tableau de droite, avec les lignes correspondantes du tableau de gauche.

FULL [OUTER] JOIN renvoie toutes les lignes des deux tables, qu'il y ait une correspondance ou non.

CROSS JOIN crée un produit cartésien des lignes des deux tables.

SUR table1.column = table2.column

La condition de jointure, qui indique comment les lignes des deux tables sont mises en correspondance. La condition de jointure peut être basée sur une ou plusieurs colonnes.

État où :

Clause facultative qui peut être utilisée pour filtrer davantage le jeu de résultats, en fonction d'une condition spécifiée.

Exemple

L’exemple suivant est une jointure entre deux tables avec la clause USING. Dans ce cas, les colonnes listid et eventid sont utilisées comme colonnes de jointure. Les résultats sont limités à seulement cinq lignes.

```
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
```

Types de jointures

INNER

Il s'agit du type de jointure par défaut. Renvoie les lignes dont les valeurs correspondent dans les deux références de table.

L'INNER JOIN est le type de jointure le plus couramment utilisé en SQL. Il s'agit d'un moyen puissant de combiner les données de plusieurs tables sur la base d'une colonne ou d'un ensemble de colonnes communs.

Syntaxe :

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

La requête suivante renverra toutes les lignes où une valeur customer_id correspond entre les tables clients et commandes. Le jeu de résultats contiendra les colonnes customer_id, name, order_id et order_date.

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

La requête suivante est une jointure interne (sans le mot-clé JOIN) entre la table LISTING et la table SALES, où la valeur LISTID de la table LISTING est comprise entre 1 et 5. Cette requête met en correspondance les valeurs de la colonne LISTID dans les tables LISTING (table de gauche) et SALES (table de droite). Les résultats montrent que les valeurs LISTID 1, 4 et 5 correspondent aux critères.

```
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
```

L’exemple suivant est une jointure interne avec la clause ON. Dans ce cas, les lignes NULL ne sont pas renvoyées.

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

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

La requête suivante est une jointure interne de deux sous-requêtes de la clause FROM. La requête recherche le nombre de billets vendus et invendus pour les différentes catégories d’événements (concerts et spectacles). Les sous-requêtes de la clause FROM sont des sous-requêtes de table ; elles peuvent renvoyer plusieurs lignes et colonnes.

```
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
```

GAUCHE [EXTÉRIEUR]

Renvoie toutes les valeurs de la référence de table de gauche et les valeurs correspondantes de la référence de table de droite, ou ajoute NULL en cas d'absence de correspondance. Elle est également appelée jointure extérieure gauche.

Il renvoie toutes les lignes de la table de gauche (première) et les lignes correspondantes de la table de droite (deuxième). S'il n'y a aucune correspondance dans la bonne table, le jeu de résultats contiendra des valeurs NULL pour les colonnes de la bonne table. Le mot clé OUTER peut être omis, et la jointure peut être écrite simplement sous la forme LEFT JOIN. Le contraire d'une jointure externe gauche est une jointure externe droite, qui renvoie toutes les lignes de la table de droite et les lignes correspondantes de la table de gauche.

Syntaxe :

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

La requête suivante renverra toutes les lignes de la table des clients, ainsi que les lignes correspondantes de la table des commandes. Si un client n'a aucune commande, le jeu de résultats

inclura toujours les informations de ce client, avec des valeurs NULL pour les colonnes order_id et order_date.

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

La requête suivante est une jointure externe gauche. Les jointures externes gauche et droite conservent les valeurs de l'une des tables jointes quand aucune correspondance n'est trouvée dans l'autre table. Les tables gauche et droite sont la première et la deuxième répertoriées dans la syntaxe. Les valeurs NULL sont utilisées pour combler les « écarts » du jeu de résultats. Cette requête fait correspondre les valeurs de la colonne LISTID dans la table LISTING (la table de gauche) et la table SALES (la table de droite). Les résultats montrent que LISTIDs 2 et 3 n'ont donné lieu à aucune vente.

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

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

DROIT [EXTÉRIEUR]

Renvoie toutes les valeurs de la référence de table de droite et les valeurs correspondantes de la référence de table de gauche, ou ajoute NULL en cas d'absence de correspondance. Elle est également appelée jointure extérieure droite.

Elle renvoie toutes les lignes de la table de droite (deuxième) et les lignes correspondantes de la table de gauche (première). S'il n'y a aucune correspondance dans le tableau de gauche, le jeu de résultats contiendra des valeurs NULL pour les colonnes du tableau de gauche. Le mot clé OUTER peut être omis, et la jointure peut être écrite simplement sous la forme RIGHT JOIN. L'opposé d'une

jointure externe droite est une jointure externe gauche, qui renvoie toutes les lignes de la table de gauche et les lignes correspondantes de la table de droite.

Syntaxe :

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

La requête suivante renverra toutes les lignes de la table des clients, ainsi que les lignes correspondantes de la table des commandes. Si un client n'a aucune commande, le jeu de résultats inclura toujours les informations de ce client, avec des valeurs NULL pour les colonnes order_id et order_date.

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

La requête suivante est une jointure externe droite. Cette requête fait correspondre les valeurs de la colonne LISTID dans la table LISTING (la table de gauche) et la table SALES (la table de droite). Les résultats montrent que LISTIDs 1, 4 et 5 correspondent aux critères.

```
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
```

COMPLET [EXTÉRIEUR]

Renvoie toutes les valeurs des deux relations, en ajoutant les valeurs NULL du côté qui ne correspond pas. Elle est également appelée jointure externe complète.

Il renvoie toutes les lignes des tables de gauche et de droite, qu'il y ait une correspondance ou non. S'il n'y a aucune correspondance, le jeu de résultats contiendra des valeurs NULL pour les colonnes de la table qui n'ont pas de ligne correspondante. Le mot clé OUTER peut être omis, et la jointure peut être écrite simplement sous la forme FULL JOIN. La jointure externe complète est moins couramment utilisée que la jointure externe gauche ou la jointure externe droite, mais elle peut être utile dans certains scénarios où vous devez voir toutes les données des deux tables, même s'il n'y a aucune correspondance.

Syntaxe :

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

La requête suivante renverra toutes les lignes des tables des clients et des commandes. Si un client n'a aucune commande, le jeu de résultats inclura toujours les informations de ce client, avec des valeurs NULL pour les colonnes order_id et order_date. Si aucun client n'est associé à une commande, le jeu de résultats inclura cette commande, avec des valeurs NULL pour les colonnes customer_id et name.

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

La requête suivante est une jointure complète. Les jointures complètes conservent les valeurs des tables jointes lorsqu’aucune correspondance n’est trouvée dans l’autre table. Les tables gauche et droite sont la première et la deuxième répertoriées dans la syntaxe. Les valeurs NULL sont utilisées pour combler les « écarts » du jeu de résultats. Cette requête fait correspondre les valeurs de la colonne LISTID dans la table LISTING (la table de gauche) et la table SALES (la table de droite). Les résultats montrent que LISTIDs 2 et 3 n'ont donné lieu à aucune vente.

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

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

La requête suivante est une jointure complète. Cette requête fait correspondre les valeurs de la colonne LISTID dans la table LISTING (la table de gauche) et la table SALES (la table de droite). Seules les lignes qui ne génèrent aucune vente (LISTIDs 2 et 3) apparaissent dans les résultats.

```
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
```

[GAUCHE] SEMI

Renvoie les valeurs du côté gauche de la référence de table qui correspondent au côté droit. Elle est également appelée demi-jointure gauche.

Elle renvoie uniquement les lignes de la table de gauche (première) qui ont une ligne correspondante dans la table de droite (deuxième). Il ne renvoie aucune colonne du tableau de droite, uniquement les colonnes du tableau de gauche. Le LEFT SEMI JOIN est utile lorsque vous souhaitez rechercher les lignes d'une table qui correspondent dans une autre table, sans avoir à renvoyer les données de la seconde table. Le LEFT SEMI JOIN est une alternative plus efficace à l'utilisation d'une sous-requête avec une clause IN ou EXISTS.

Syntaxe :

```
SELECT column1, column2, ..., columnn
FROM table1
LEFT SEMI JOIN table2
```

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

La requête suivante renverra uniquement les colonnes customer_id et name de la table des clients, pour les clients qui ont au moins une commande dans la table des commandes. Le jeu de résultats n'inclura aucune colonne du tableau des commandes.

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

CROSS JOIN

Renvoie le produit cartésien de deux relations. Cela signifie que le jeu de résultats contiendra toutes les combinaisons possibles de lignes des deux tableaux, sans qu'aucune condition ni aucun filtre ne soient appliqués.

Le CROSS JOIN est utile lorsque vous devez générer toutes les combinaisons possibles de données à partir de deux tables, par exemple dans le cas de la création d'un rapport qui affiche toutes les combinaisons possibles d'informations sur les clients et les produits. Le CROSS JOIN est différent des autres types de jointure (INNER JOIN, LEFT JOIN, etc.) car il ne comporte aucune condition de jointure dans la clause ON. La condition de jointure n'est pas obligatoire pour une jointure croisée.

Syntaxe :

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

La requête suivante renverra un jeu de résultats contenant toutes les combinaisons possibles de customer_id, customer_name, product_id et product_name à partir des tables clients et produits. Si le tableau des clients comporte 10 lignes et le tableau des produits 20 lignes, le jeu de résultats du CROSS JOIN contiendra 10 x 20 = 200 lignes.

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

La requête suivante est une jointure croisée ou cartésienne de la table LISTING et de la table SALES avec un prédicat pour limiter les résultats. Cette requête correspond aux valeurs des colonnes LISTID de la table SALES et de la table LISTING pour LISTIDs 1, 2, 3, 4 et 5 dans les deux tables. Les résultats montrent que 20 lignes correspondent aux critères.

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

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

ANTI-JOINTURE

Renvoie les valeurs de la référence de table de gauche qui ne correspondent pas à la référence de table de droite. On l'appelle aussi « anti-jointure gauche ».

L'opération ANTI JOIN est une opération utile lorsque vous souhaitez rechercher les lignes d'une table qui ne correspondent pas dans une autre table.

Syntaxe :

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

La requête suivante renverra tous les clients qui n'ont pas passé de commande.

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

NATURAL

Spécifie que les lignes issues des deux relations seront implicitement mises en correspondance sur un pied d'égalité pour toutes les colonnes dont les noms sont identiques.

Il fait automatiquement correspondre les colonnes portant le même nom et le même type de données entre les deux tables. Il n'est pas nécessaire de spécifier explicitement la condition de jointure dans la clause ON. Il combine toutes les colonnes correspondantes entre les deux tables dans le jeu de résultats.

Le NATURAL JOIN est un raccourci pratique lorsque les tables que vous joignez comportent des colonnes portant le même nom et le même type de données. Cependant, il est généralement recommandé d'utiliser le plus explicite INNER JOIN... Syntaxe ON pour rendre les conditions de jointure plus explicites et plus faciles à comprendre.

Syntaxe :

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

L'exemple suivant est une jointure naturelle entre deux tables`departments`, `employees` avec les colonnes suivantes :

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

La requête suivante renverra un jeu de résultats qui inclut le prénom, le nom de famille et le nom du département pour toutes les lignes correspondantes entre les deux tables, en fonction de la `department_id` colonne.

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

L'exemple suivant est une jointure naturelle entre deux tables. Dans ce cas, les colonnes listid, sellerid, eventid et dateid présentent des noms et des types de données identiques dans les deux tables et sont donc utilisées comme colonnes de jointure. Les résultats sont limités à seulement cinq lignes.

```
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
```

Clause WHERE

La clause WHERE contient les conditions qui joignent les tables ou appliquent les prédicats aux colonnes des tables. Les tables peuvent être à jointure interne en utilisant la syntaxe appropriée dans la clause WHERE ou FROM. Les critères de jointure externe doivent être spécifiés dans la clause FROM.

Syntaxe

```
[ WHERE condition ]
```

condition

Toute condition avec un résultat Boolean, comme une condition de jointure ou un prédicat sur une colonne de table. Les exemples suivants sont des conditions de jointure valides :

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

Les exemples suivants sont des conditions valides sur les colonnes des tables :

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

Les conditions peuvent être simples ou complexes ; pour les conditions complexes, vous pouvez utiliser des parenthèses afin d'isoler des unités logiques. Dans l'exemple suivant, la condition de jointure est placée entre parenthèses.

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

Notes d'utilisation

Vous pouvez utiliser des alias dans la clause WHERE pour référencer les expressions de liste de sélection.

Vous ne pouvez pas limiter les résultats des fonctions d'agrégation dans la clause WHERE ; utilisez à cette fin la clause HAVING.

Les colonnes qui sont limités dans la clause WHERE doivent provenir de références de table de la clause FROM.

Exemple

La requête suivante utilise une combinaison de différentes restrictions de clause WHERE, y compris une condition de jointure pour les tables SALES et EVENT, un prédicat sur la colonne EVENTNAME et deux prédicats sur la colonne STARTTIME.

```
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Hannah Montana'
and date_part(quarter, starttime) in(1,2)
```

```
and date_part(year, starttime) = 2008
order by 3 desc, 4, 2, 1 limit 10;

eventname    |      starttime      |   costperticket   | qtysold
---------------+---------------------+-------------------+---------
Hannah Montana | 2008-06-07 14:00:00 |     1706.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |     1658.00000000 |       2
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       3
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       2
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       4
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       1
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       4
(10 rows)
```

Clause VALUES

La clause VALUES est utilisée pour fournir un ensemble de valeurs de ligne directement dans la requête, sans qu'il soit nécessaire de référencer une table.

La clause VALUES peut être utilisée dans les scénarios suivants :

- Vous pouvez utiliser la clause VALUES dans une instruction INSERT INTO pour spécifier les valeurs des nouvelles lignes insérées dans un tableau.
- Vous pouvez utiliser la clause VALUES seule pour créer un jeu de résultats temporaire, ou un tableau en ligne, sans qu'il soit nécessaire de référencer un tableau.
- Vous pouvez combiner la clause VALUES avec d'autres clauses SQL, telles que WHERE, ORDER BY ou LIMIT, pour filtrer, trier ou limiter les lignes du jeu de résultats.

Cette clause est particulièrement utile lorsque vous devez insérer, interroger ou manipuler un petit ensemble de données directement dans votre instruction SQL, sans avoir besoin de créer ou de référencer une table permanente. Il vous permet de définir les noms des colonnes et les valeurs correspondantes pour chaque ligne, ce qui vous permet de créer des ensembles de résultats temporaires ou d'insérer des données à la volée, sans avoir à gérer une table séparée.

Syntaxe

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

Parameters

expression

Expression qui spécifie une combinaison d'une ou de plusieurs valeurs, opérateurs et fonctions SQL aboutissant à une valeur.

alias de table

Alias qui spécifie un nom temporaire avec une liste de noms de colonne facultative.

Exemple

L'exemple suivant crée un tableau en ligne, un jeu de résultats temporaire semblable à un tableau avec deux colonnes, et. `col1` `col2` La seule ligne du jeu de résultats contient les valeurs `"one"` et1, respectivement. La `SELECT * FROM` partie de la requête extrait simplement toutes les colonnes et lignes de ce jeu de résultats temporaire. Les noms des colonnes (`col1`et`col2`) sont automatiquement générés par le système de base de données, car la clause VALUES ne spécifie pas explicitement les noms des colonnes.

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

Si vous souhaitez définir des noms de colonnes personnalisés, vous pouvez le faire en utilisant une clause AS après la clause VALUES, comme ceci :

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

Cela créerait un jeu de résultats temporaire avec les noms des colonnes `name` et`id`, au lieu de la valeur par défaut `col1` et`col2`.

Clause GROUP BY

La clause GROUP BY identifie les colonnes de regroupement de la requête. Les colonnes de regroupement doivent être déclarées lorsque la requête calcule les regroupements avec des fonctions standard telles que SUM, AVG et COUNT. Si une fonction d'agrégation est présente dans l'expression SELECT, toute colonne de l'expression SELECT qui ne figure pas dans une fonction d'agrégation doit figurer dans la clause GROUP BY.

Pour de plus amples informations, veuillez consulter AWS Clean Rooms Fonctions Spark SQL.

Syntaxe

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

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

Paramètres

expr

La liste des colonnes ou des expressions doit correspondre à la liste des expressions non agrégées de la liste de sélection de la requête. Par exemple, imaginons la requête simple suivante.

```
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)
```

Dans cette requête, la liste de sélection se compose de deux expressions d'agrégation. La première utilise la fonction SUM et la seconde la fonction COUNT. Les deux autres colonnes, LISTID et EVENTID, doivent être déclarées en tant que colonnes de regroupement.

Les expressions de la clause GROUP BY peuvent également faire référence à la liste de sélection en utilisant des nombres ordinaux. Par exemple, l'exemple précédent peut être abrégé comme suit.

```
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

Vous pouvez utiliser l'extension d'agrégation ROLLUP pour effectuer plusieurs opérations GROUP BY dans une seule instruction. Pour plus d'informations sur les extensions d'agrégation et les fonctions associées, consultez Extensions de regroupement.

Extensions de regroupement

AWS Clean Rooms prend en charge les extensions d'agrégation pour effectuer plusieurs opérations GROUP BY dans une seule instruction.

GROUPING SETS

Calcule un ou plusieurs jeux de regroupement dans une seule instruction. Un jeu de regroupement est l'ensemble d'une clause GROUP BY unique, un jeu de 0 colonne ou plus avec lequel vous

pouvez regrouper le jeu de résultats d'une requête. GROUP BY GROUPING SETS revient à exécuter une requête UNION ALL sur un jeu de résultats groupé par différentes colonnes. Par exemple, GROUP BY GROUPING SETS((a), (b)) est équivalent à GROUP BY a UNION ALL GROUP BY b.

L'exemple suivant renvoie le coût des produits de la table des commandes, regroupés par catégories de produits et type de produits vendus.

```
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

Suppose une hiérarchie dans laquelle les colonnes précédentes sont considérées comme les parents des colonnes suivantes. ROLLUP regroupe les données par colonnes fournies et renvoie des lignes de sous-totaux supplémentaires représentant les totaux à tous les niveaux de colonnes de regroupement, en plus des lignes groupées. Par exemple, vous pouvez utiliser GROUP BY ROLLUP((a), (b)) pour renvoyer un jeu de résultats regroupé d'abord par a, puis par b en supposant que b est une sous-section de a. ROLLUP renvoie également une ligne contenant le jeu des résultats sans regrouper les colonnes.

GROUP BY ROLLUP((a), (b)) équivaut à GROUP BY GROUPING SETS((a,b), (a), ()).

L'exemple suivant renvoie le coût des produits de la table des commandes, regroupés d'abord par catégorie, puis par produit, le produit étant une subdivision de la catégorie.

```
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

Regroupe les données par colonnes fournies et renvoie des lignes de sous-totaux supplémentaires représentant les totaux à tous les niveaux de colonnes de regroupement, en plus des lignes groupées. CUBE renvoie les mêmes lignes que ROLLUP, mais ajoute des lignes de sous-total supplémentaires pour chaque combinaison de colonnes de regroupement non couverte par ROLLUP. Par exemple, vous pouvez utiliserGROUP BY CUBE ((a), (b)) pour renvoyer un jeu de résultats regroupé d'abord par a, puis par b en supposant que b est une sous-section de a, puis par b uniquement. CUBE renvoie également une ligne contenant le jeu des résultats sans regrouper les colonnes.

GROUP BY CUBE((a), (b)) équivaut à GROUP BY GROUPING SETS((a, b), (a), (b), ()).

L'exemple suivant renvoie le coût des produits de la table des commandes, regroupés d'abord par catégorie, puis par produit, le produit étant une subdivision de la catégorie. Contrairement à l'exemple précédent pour ROLLUP, l'instruction renvoie des résultats pour chaque combinaison de colonnes de regroupement.

```
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)
```

Clause HAVING

La clause HAVING applique une condition à l'ensemble des résultats groupés intermédiaires que renvoie une requête.

Syntaxe

```
[ HAVING condition ]
```

Par exemple, vous pouvez limiter les résultats d'une fonction SUM :

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

La condition HAVING est appliquée après que toutes les conditions de la clause WHERE ont été appliquées et que les opérations GROUP BY sont terminées.

La condition elle-même prend la même forme que celle de toute condition de clause WHERE.

Notes d'utilisation

- Toutes les colonnes référencées dans une condition de clause HAVING doivent être une colonne de regroupement ou une colonne qui fait référence au résultat d'une fonction d'agrégation.
- Dans une clause HAVING, vous ne pouvez pas spécifier :
  - Un nombre ordinal qui fait référence à un élément de la liste de sélection. Seules les clauses GROUP BY et ORDER BY acceptent des nombres ordinaux.

Exemples

La requête suivante calcule la vente totale de billets pour tous les événements selon leur nom, puis supprime les événements où le total des ventes est inférieur à 800 000 $ US. La condition HAVING est appliquée aux résultats de la fonction d'agrégation de la liste de sélection : `sum(pricepaid)`.

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
group by 1
having sum(pricepaid) > 800000
```

```
order by 2 desc, 1;

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

La requête suivante calcule un ensemble de résultats similaire. Dans ce cas, toutefois, la condition HAVING est appliquée à un regroupement qui n'est pas spécifié dans la liste de sélection : `sum(qtysold)`. Les événements qui n'ont pas vendu plus de 2 000 billets disparaissent du résultat final.

```
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)
```

Définir les opérateurs

Les opérateurs set sont utilisés pour comparer et fusionner les résultats de deux expressions de requête distinctes.

AWS Clean Rooms Spark SQL prend en charge les opérateurs d'ensemble suivants répertoriés dans le tableau suivant.

Opérateur du set
INTERSECT
TOUT CROISER
EXCEPT
SAUF TOUS
UNION
UNION TOUT

Par exemple, si vous voulez savoir quels utilisateurs d'un site web sont à la fois acheteurs et vendeurs, mais que leurs noms d'utilisateur sont stockés dans des colonnes ou tables distinctes, vous pouvez trouver l'intersection de ces deux types d'utilisateurs. Si vous voulez savoir quels utilisateurs du site web sont acheteurs mais pas vendeurs, vous pouvez utiliser l'opérateur EXCEPT pour trouver la différence entre les deux listes d'utilisateurs. Si vous souhaitez créer une liste de tous les utilisateurs, quel que soit le rôle, vous pouvez utiliser l'opérateur UNION.

> **Note**
>
> Les clauses ORDER BY, LIMIT, SELECT TOP et OFFSET ne peuvent pas être utilisées dans les expressions de requête fusionnées par les opérateurs d'ensemble UNION, UNION ALL, INTERSECT et EXCEPT.

Rubriques

- Exemple de requête EXCEPT

Syntaxe

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

Parameters

sous-requête1, sous-requête2

Expression de requête qui correspond, sous la forme de sa liste de sélection, à une deuxième expression de requête qui suit l'opérateur UNION, UNION ALL, INTERSECT, INTERSECT ALL, EXCEPT ou EXCEPT ALL. Les deux expressions doivent comporter le même nombre de colonnes de sortie avec des types de données compatibles ; sinon, les deux jeux de résultats ne peuvent pas être comparés et fusionnés. Les opérations de définition n'autorisent pas la conversion implicite entre différentes catégories de types de données. Pour de plus amples informations, veuillez consulter Compatibilité et conversion de types.

Vous pouvez créer des requêtes qui contiennent un nombre illimité d'expressions de requête et les lier avec les opérateurs UNION, INTERSECT et EXCEPT dans n'importe quelle combinaison. Par exemple, la structure de requête suivante est valide, en supposant que les tables T1, T2 et T3 contiennent des ensembles de colonnes compatibles :

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

UNION [TOUS | DISTINCTS]

Opération de définition qui renvoie les lignes de deux expressions de requête, indépendamment de savoir si les lignes proviennent de l'une ou des deux expressions.

SE CROISER [TOUS | DISTINCTS]

Opération de définition qui renvoie les lignes provenant de deux expressions de requête. Les lignes qui ne sont pas retournées par les deux expressions sont ignorées.

SAUF [TOUS | DISTINCT]

Opération de définition qui renvoie les lignes qui dérivent de l'une de deux expressions de requête. Pour être éligible pour le résultat, lignes doivent exister dans la première table de résultats, pas dans la deuxième.

EXCEPT ALL ne supprime pas les doublons des lignes de résultats.

MINUS et EXCEPT sont des synonymes exacts.

Ordre d'évaluation des opérateurs ensemblistes

Les opérateurs ensemblistes UNION et EXCEPT sont associatifs à gauche. Si les parenthèses ne sont pas spécifiées pour influer sur l'ordre de priorité, une combinaison de ces opérateurs ensemblistes est évaluée de gauche à droite. Par exemple, dans la requête suivante, l'UNION de T1 et de T2 est évaluée en premier, puis l'opération EXCEPT est effectuée sur le résultat UNION :

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

L'opérateur INTERSECT est prioritaire sur les opérateurs UNION et EXCEPT quand une combinaison d'opérateurs est utilisée dans la même requête. Par exemple, la requête suivante permet d'évaluer l'intersection de T2 et de T3, puis d'unir le résultat à T1 :

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

Par l'ajout de parenthèses, vous pouvez appliquer un ordre d'évaluation différent. Dans le cas suivant, le résultat de l'union de T1 et de T2 est croisé avec T3, et la requête est susceptible de produire un résultat différent.

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

```
(select * from t3)
```

Notes d'utilisation

- Les noms de colonne retournés dans le résultat d'une opération ensembliste sont les noms de colonne (ou alias) des tables de la première expression de requête. Comme ces noms de colonne sont potentiellement trompeurs, en ce sens que les valeurs de la colonne proviennent de tables de l'un ou de l'autre côté de l'opérateur ensembliste, il se peut que vous vouliez fournir des alias descriptifs pour le jeu de résultats.
- Lorsque les requêtes avec opérateurs ensemblistes renvoient des résultats décimaux, les colonnes de résultats correspondantes sont promues pour renvoyer les mêmes précision et échelle. Par exemple, dans la requête suivante, où T1.REVENUE est une colonne DECIMAL(10,2) et T2.REVENUE une colonne DECIMAL(8,4), le résultat décimal est promu en DECIMAL(12,4) :

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

  L'échelle est 4, parce que c'est l'échelle maximale des deux colonnes. La précision est 12 parce que T1.REVENUE nécessite 8 chiffres à gauche de la virgule (12-4 = 8). Cette promotion de type garantit que toutes les valeurs de chaque côté de l'UNION conviennent au résultat. Pour les valeurs 64 bits, la précision de résultat maximale est de 19 et l'échelle de résultat maximale de 18. Pour les valeurs 128 bits, la précision de résultat maximale est de 38 et l'échelle de résultat maximale de 37.

  Si le type de données obtenu dépasse les limites de AWS Clean Rooms précision et d'échelle, la requête renvoie une erreur.
- Pour les opérations ensemblistes, deux lignes sont traitées comme identiques si, pour chaque paire correspondante de colonnes, les deux valeurs de données sont égales ou toutes deux NULL. Par exemple, si les tables T1 et T2 contiennent une colonne et une ligne, et que la ligne a la valeur NULL dans les deux tables, une opération INTERSECT sur ces tables renvoie cette ligne.

Exemple de requêtes UNION

Dans la requête UNION suivante, les lignes de la table SALES sont fusionnées avec les lignes de la table LISTING. Trois colonnes compatibles sont sélectionnées à partir de chaque table ; dans ce cas, les colonnes correspondantes ont les mêmes noms et types de données.

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

```
union select listid, sellerid, eventid from sales


listid | sellerid | eventid
--------+----------+---------
1 |    36861 |    7872
2 |    16002 |    4806
3 |    21461 |    4256
4 |     8117 |    4337
5 |     1616 |    8647
```

L'exemple suivant montre comment vous pouvez ajouter une valeur littérale à la sortie d'une requête UNION afin que vous puissiez voir quelle expression de requête a généré chaque ligne du jeu de résultats. La requête identifie les lignes de la première expression de requête comme « B » (pour « buyers ») et les lignes de la deuxième expression de requête comme « S » (pour « sellers »).

La requête identifie les acheteurs et les vendeurs pour les transactions de billet égales ou supérieures à 10 000 $ US. La seule différence entre les deux expressions de requête de chaque côté de l'opérateur d'UNION est la colonne de jointure de la table SALES.

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

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

L’exemple suivant utilise un opérateur UNION ALL, car les lignes dupliquées, s’il en existe, doivent être conservées dans le résultat. Pour une série d'événements spécifique IDs, la requête renvoie 0 ligne ou plus pour chaque vente associée à chaque événement, et 0 ou 1 ligne pour chaque annonce de cet événement. IDs Les événements sont propres à chaque ligne des tableaux LISTING et EVENT, mais il peut y avoir plusieurs ventes pour la même combinaison d'événement et d'annonce IDs dans le tableau SALES.

La troisième colonne du jeu de résultats identifie la source de la ligne. Si la source est la table SALES, un « Yes » apparaît dans la colonne SALESROW. (SALESROW est un alias de SALES. LISTID.) Si la ligne vient de la table LISTING, un « No » apparaît dans la colonne SALESROW.

Dans ce cas, le jeu de résultats se compose de trois lignes de vente pour affichage 500, événement 7787. En d’autres termes, trois transactions différentes ont eu lieu pour cette combinaison d’affichage et d’événement. Les deux autres listes, 501 et 502, n'ont généré aucune vente. La seule ligne produite par la requête pour ces listes IDs provient donc de la table LISTING (SALESROW = « Non »).

```
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
```

Si vous exécutez la même requête sans le mot-clé ALL, le résultat ne conserve qu’une seule des transactions de vente.

```
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
```

Exemple de requête UNION ALL

L'exemple suivant utilise un opérateur UNION ALL, car les lignes dupliquées, s'il en existe, doivent être conservées dans le résultat. Pour une série d'événements spécifique IDs, la requête renvoie 0 ligne ou plus pour chaque vente associée à chaque événement, et 0 ou 1 ligne pour chaque annonce de cet événement. IDs Les événements sont propres à chaque ligne des tableaux LISTING et EVENT, mais il peut y avoir plusieurs ventes pour la même combinaison d'événement et d'annonce IDs dans le tableau SALES.

La troisième colonne du jeu de résultats identifie la source de la ligne. Si la source est la table SALES, un « Yes » apparaît dans la colonne SALESROW. (SALESROW est un alias de SALES. LISTID.) Si la ligne vient de la table LISTING, un « No » apparaît dans la colonne SALESROW.

Dans ce cas, le jeu de résultats se compose de trois lignes de vente pour affichage 500, événement 7787. En d'autres termes, trois transactions différentes ont eu lieu pour cette combinaison d'affichage et d'événement. Les deux autres listes, 501 et 502, n'ont généré aucune vente. La seule ligne produite par la requête pour ces listes IDs provient donc de la table LISTING (SALESROW = « Non »).

```
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
```

Si vous exécutez la même requête sans le mot-clé ALL, le résultat ne conserve qu'une seule des transactions de vente.

```
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
```

Exemple de requêtes INTERSECT

Comparez l'exemple suivant avec le premier exemple UNION. La seule différence entre les deux exemples est l'opérateur ensembliste qui est utilisé, mais les résultats sont très différents. Seule une des lignes est la même :

```
235494 |    23875 |    8771
```

Il s'agit de la seule ligne du résultat limité de 5 lignes qui a été trouvé dans les deux tables.

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

listid | sellerid | eventid
--------+----------+---------
235494 |    23875 |    8771
235482 |     1067 |    2667
```

```
235479 |     1589 |    7303
235476 |    15550 |     793
235475 |    22306 |    7848
```

La requête suivante détecte les événements (pour lesquels des billets ont été vendus) qui se sont déroulées dans des lieux de New York et de Los Angeles en mars. La différence entre les deux expressions de requête est la contrainte sur la colonne 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
```

Exemple de requête EXCEPT

La table CATEGORY de la base de données contient les 11 lignes suivantes :

```
 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)
```

Supposons qu'une table CATEGORY_STAGE (table intermédiaire) contienne une seule ligne supplémentaire :

```
 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)
```

renvoiez la différence entre les deux tables. En d'autres termes, renvoiez les lignes qui sont dans la table CATEGORY_STAGE, mais pas dans la table CATEGORY :

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

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

```
(1 row)
```

La requête équivalente suivante utilise le synonyme MINUS.

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

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

Si vous inversez l'ordre des expressions SELECT, la requête ne renvoie aucune ligne.

Clause ORDER BY

La clause ORDER BY trie le jeu de résultats d'une requête.

> **Note**
>
> L'expression ORDER BY la plus éloignée ne doit comporter que des colonnes figurant dans la liste de sélection.

Rubriques

- Syntaxe
- Parameters
- Notes d'utilisation
- Exemples avec ORDER BY

Syntaxe

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

Parameters

expression

Expression qui définit l'ordre de tri du résultat de la requête. Il se compose d'une ou de plusieurs colonnes dans la liste de sélection. Les résultats sont retournés en fonction du classement UTF-8 binaire. Vous pouvez aussi spécifier les éléments suivants :

- Nombres ordinaux qui représentent la position des entrées de la liste de sélection (ou position des colonnes de la table s'il n'existe aucune liste de sélection)
- Alias qui définissent les entrées de la liste de sélection

Lorsque la clause ORDER BY contient plusieurs expressions régulières, le jeu de résultats est trié selon la première expression, puis la deuxième expression est appliquée aux lignes de la première expression ayant des valeurs correspondantes, et ainsi de suite.

ASC | DESC

Option qui définit l'ordre de tri de l'expression, comme suit :

- ASC : croissant (par exemple, de faible à élevé pour les valeurs numériques et de « A » à « Z » pour les chaînes de caractères). Si aucune option n'est spécifiée, les données sont triées dans l'ordre croissant par défaut.
- DESC : descendantes (valeurs d'élevées à faibles pour les valeurs numériques ; de « Z » à « A » pour les chaînes).

NULLS FIRST | NULLS LAST

Option qui spécifie si les valeurs NULL doivent être triées en premier, avant les valeurs non null, ou en dernier, après les valeurs non null. Par défaut, les valeurs NULL sont triées et classées en dernier par ordre croissant (ASC) et triées et classées en premier par ordre décroissant (DESC).

LIMIT nombre | ALL

Option qui contrôle le nombre de lignes triées renvoyées par la requête. Le nombre LIMIT doit être un nombre entier positif ; la valeur maximale est `2147483647`.

LIMIT 0 ne renvoie aucune ligne. Vous pouvez utiliser cette syntaxe à des fins de test, pour vérifier qu'une requête s'exécute (sans afficher aucune ligne) ou pour renvoyer une liste de colonnes d'une table. Une clause ORDER BY est redondante si vous utilisez LIMIT 0 pour renvoyer une liste de colonnes. La valeur par défaut est LIMIT ALL.

OFFSET début

Option qui spécifie d'ignorer le nombre de lignes qui précèdent début avant de commencer à renvoyer les lignes. Le nombre OFFSET doit être un nombre entier positif ; la valeur maximale est 2147483647. Lorsqu'elles sont utilisées avec l'option LIMIT, les lignes OFFSET sont ignorées avant de commencer à compter les lignes LIMIT qui sont retournées. Si l'option LIMIT n'est pas utilisée, le nombre de lignes du jeu de résultats est diminué du nombre de lignes qui sont ignorées. Comme les lignes ignorées par une clause OFFSET continuent de devoir être analysées, il peut être inefficace de choisir une valeur OFFSET élevée.

Notes d'utilisation

Notez le comportement attendu suivant avec les clauses ORDER BY :

- Les valeurs NULL sont considérées comme « plus élevés » que toutes les autres valeurs. Avec l'ordre de tri croissant par défaut, les valeurs NULL sont triées à la fin. Pour modifier ce comportement, utilisez l'option NULLS FIRST.
- Lorsqu'une requête ne contient pas une clause ORDER BY, le système renvoie des jeux de résultats sans classement prévisible des lignes. La même requête exécutée deux fois peut renvoyer le même jeu de résultats dans un ordre différent.
- Les options LIMIT et OFFSET peuvent être utilisées sans clause ORDER BY ; cependant, pour renvoyer un ensemble cohérent de lignes, utilisez ces options conjointement à ORDER BY.
- Dans tout système parallèle AWS Clean Rooms, par exemple, lorsque ORDER BY ne produit pas d'ordre unique, l'ordre des lignes n'est pas déterministe. En d'autres termes, si l'expression ORDER BY produit des valeurs dupliquées, l'ordre de retour de ces lignes peut varier d'un système à l'autre ou d'une exécution AWS Clean Rooms à l'autre.
- AWS Clean Rooms ne prend pas en charge les littéraux de chaîne dans les clauses ORDER BY.

Exemples avec ORDER BY

renvoiez les 11 lignes de la table CATEGORY, triées sur la deuxième colonne, CATGROUP. Pour les résultats qui ont la même valeur CATGROUP, classez les valeurs de colonne CATDESC en fonction de la longueur de la chaîne de caractères. Triez ensuite sur les colonnes CATID et CATNAME.

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

catid | catgroup |  catname  |                 catdesc
```

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

renvoiez les colonnes sélectionnées de la table SALES, triées selon les valeurs QTYSOLD les plus élevées. Limitez les résultats aux 10 lignes supérieures :

```
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
```

renvoiez une liste de colonnes et aucune ligne à l'aide de la syntaxe LIMIT 0 :

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

Exemples de sous-requête

Les exemples suivants illustrent différentes façons par lesquelles les sous-requêtes conviennent aux requêtes SELECT. Pour obtenir un autre exemple de l'utilisation des sous-requêtes, consultez Exemple.

Sous-requête SELECT liste

L'exemple suivant contient une sous-requête dans la liste SELECT. Cette sous-requête est scalaire : elle renvoie une et une seule colonne et une seule valeur, ce qui est répété dans le résultat pour chaque ligne retournée à partir de la requête externe. La requête compare la valeur Q1SALES que la sous-requête calcule aux valeurs des ventes des deux autres trimestres (2 et 3) en 2008, comme défini par la requête externe.

```
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)
```

Sous-requête de clause WHERE

L'exemple suivant contient une sous-requête de table dans la clause WHERE. Cette sous-requête produit plusieurs lignes. Dans ce cas, les lignes ne contiennent qu'une seule colonne, mais les sous-requêtes de table peuvent contenir plusieurs colonnes et lignes, tout comme n'importe quelle autre table.

La requête recherche les 10 meilleurs vendeurs en termes de nombre maximal de billets vendus. La liste des 10 meilleurs est limitée par la sous-requête, qui supprime les utilisateurs qui résident dans les villes où il y a des lieux de vente. Cette requête peut être écrite de différentes façons ; par exemple, la sous-requête peut être réécrite comme jointure au sein de la requête principale.

```
select firstname, lastname, city, max(qtysold) as maxsold
from users join sales on users.userid=sales.sellerid
where users.city not in(select venuecity from venue)
group by firstname, lastname, city
order by maxsold desc, city desc
limit 10;

firstname | lastname  |      city      | maxsold
----------+-----------+----------------+---------
Noah      | Guerrero | Worcester      |       8
Isadora   | Moss     | Winooski       |       8
Kieran    | Harrison | Westminster    |       8
Heidi     | Davis    | Warwick        |       8
Sara      | Anthony  | Waco           |       8
Bree      | Buck     | Valdez         |       8
Evangeline | Sampson  | Trenton        |       8
Kendall   | Keith    | Stillwater     |       8
Bertha    | Bishop   | Stevens Point  |       8
Patricia  | Anderson | South Portland |       8
(10 rows)
```

Sous-requêtes de clause WITH

Consultez Clause WITH.

Sous-requêtes corrélées

L'exemple suivant contient une sous-requête corrélée dans la clause WHERE ; ce genre de sous-requête contient une ou plusieurs corrélations entre ses colonnes et les colonnes générés par la requête externe. Dans ce cas, la corrélation est `where s.listid=l.listid`. Pour chaque ligne que produit la requête externe, la sous-requête est exécutée pour qualifier ou disqualifier la ligne.

```
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)
```

Modèles de sous-requêtes corrélées non pris en charge

Le planificateur de requête utilise une méthode de réécriture de requête appelée décorrélation de sous-requête afin d'optimiser plusieurs modèles de sous-requêtes corrélées en vue de l'exécution dans un environnement MPP. Certains types de sous-requêtes corrélées suivent des modèles qui ne AWS Clean Rooms peuvent pas être décorrélés et qui ne sont pas compatibles. Les requêtes qui contiennent les références de corrélation suivantes génèrent des erreurs :

- Les références de corrélation qui ignorent un bloc de requête, également appelées « références de corrélation de niveau non hiérarchique ». Par exemple, dans la requête suivante, le bloc contenant la référence de corrélation et le bloc ignoré sont connectés par un prédicat NOT EXISTS :

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

  Le bloc ignoré dans ce cas est la sous-requête sur la table LISTING. La référence de corrélation correspond aux tables EVENT et SALES.
- Références de corrélation à partir d'une sous-requête qui fait partie d'une clause ON dans une requête externe :

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

  La clause ON contient une référence de corrélation depuis SALES dans la sous-requête jusqu'à EVENT dans la requête externe.
- Références de corrélation sensibles à la valeur nulle avec une table AWS Clean Rooms système. Par exemple :

```
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);
```

- Références de corrélation à partir d'une sous-requête contenant une fonction de fenêtrage.

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

- Références d'une colonne GROUP BY aux résultats d'une sous-requête corrélée. Par exemple :

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

- Références de corrélation à partir d'une sous-requête avec fonction d'agrégation et d'une clause GROUP BY, connectée à la requête externe par un prédicat IN. (Cette restriction ne s'applique pas aux fonctions d'agrégation MIN et MAX.) Par exemple :

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

AWS Clean Rooms Fonctions Spark SQL

AWS Clean Rooms Spark SQL prend en charge les fonctions SQL suivantes :

Rubriques

- Fonctions de formatage des types de données
- Fonctions de date et d'heure
- Fonctions de chiffrement et de déchiffrement
- Fonctions de hachage
- Fonctions Hyperloglog
- Fonctions JSON
- Fonctions mathématiques
- Fonctions scalaires
- Fonctions de chaîne
- Fonctions liées à la confidentialité
- Fonctions de fenêtrage

Fonctions d'agrégation

Les fonctions d'agrégation de AWS Clean Rooms Spark SQL sont utilisées pour effectuer des calculs ou des opérations sur un groupe de lignes et renvoyer une valeur unique. Ils sont essentiels pour les tâches d'analyse et de synthèse des données.

AWS Clean Rooms Spark SQL prend en charge les fonctions d'agrégation suivantes :

Rubriques

- Fonction ANY_VALUE
- Fonction APPROX COUNT_DISTINCT
- Fonction APPROX PERCENTILE
- Fonction AVG
- Fonction BOOL_AND
- Fonction BOOL_OR
- Fonction CARDINALITY
- Fonction COLLECT_LIST
- Fonction COLLECT_SET
- COUNTet COUNT DISTINCT fonctions
- Fonction COUNT
- Fonction MAX

- Fonction MEDIAN
- Fonction MIN
- Fonction PERCENTILE
- Fonction SKEWNESS
- Fonctions STDDEV_SAMP et STDDEV_POP
- SUMet SUM DISTINCT fonctions
- Fonctions VAR_SAMP et VAR_POP

Fonction ANY_VALUE

La fonction ANY_VALUE renvoie n'importe quelle valeur des valeurs d'expression en entrée de manière non déterministe. Cette fonction peut renvoyer la valeur NULL si l'expression en entrée n'entraîne pas de renvoi de ligne.

Syntaxe

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

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère. L'expression est l'un des types de données suivants :

isIgnoreNull

Un booléen qui détermine si la fonction doit renvoyer uniquement des valeurs non nulles.

Renvoie

Renvoie le même type de données que expression.

Notes d'utilisation

Si une instruction qui spécifie la fonction ANY_VALUE d'une colonne inclut également une deuxième référence de colonne, la deuxième colonne doit apparaître dans une clause GROUP BY ou être incluse dans une fonction d'agrégation.

Exemples

L'exemple suivant renvoie une instance de n'importe quel `dateid` endroit où se `eventname` trouve le`Eagles`.

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

Voici les résultats.

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

L'exemple suivant renvoie une instance de n'importe quel `dateid` endroit où `eventname` est `Eagles` ou`Cold War Kids`.

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

Voici les résultats.

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

Fonction APPROX COUNT_DISTINCT

APPROX COUNT_DISTINCT fournit un moyen efficace d'estimer le nombre de valeurs uniques dans une colonne ou un ensemble de données.

Syntaxe

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

Arguments

expr

Expression ou colonne pour laquelle vous souhaitez estimer le nombre de valeurs uniques.

Il peut s'agir d'une seule colonne, d'une expression complexe ou d'une combinaison de colonnes.

Membres de la famille D

Paramètre facultatif qui spécifie l'écart type relatif souhaité de l'estimation.

Il s'agit d'une valeur comprise entre 0 et 1, représentant l'erreur relative maximale acceptable de l'estimation. Une valeur RelativeSD plus faible se traduira par une estimation plus précise mais plus lente.

Si ce paramètre n'est pas fourni, une valeur par défaut (généralement autour de 0,05 ou 5 %) est utilisée.

Renvoie

Renvoie la cardinalité estimée par HyperLogLog ++. RelativeSD définit l'écart type relatif maximal autorisé.

Exemple

La requête suivante estime le nombre de valeurs uniques dans la `col1` colonne, avec un écart type relatif de 1 % (0,01).

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

La requête suivante estime que la `col1` colonne contient 3 valeurs uniques (les valeurs 1, 2 et 3).

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

Fonction APPROX PERCENTILE

APPROX PERCENTILE est utilisé pour estimer la valeur percentile d'une expression ou d'une colonne donnée sans avoir à trier l'ensemble de données dans son intégralité. Cette fonction est utile dans les scénarios dans lesquels vous devez comprendre rapidement la distribution d'un ensemble de données volumineux ou suivre des métriques basées sur des percentiles, sans les frais de calcul liés à un calcul de percentile exact. Cependant, il est important de comprendre les compromis entre vitesse et précision, et de choisir la tolérance d'erreur appropriée en fonction des exigences spécifiques de votre cas d'utilisation.

Syntaxe

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

Arguments

expr

Expression ou colonne pour laquelle vous souhaitez estimer la valeur du percentile.

Il peut s'agir d'une seule colonne, d'une expression complexe ou d'une combinaison de colonnes.

percentile

La valeur du percentile que vous souhaitez estimer, exprimée sous la forme d'une valeur comprise entre 0 et 1.

Par exemple, 0,5 correspondrait au 50e percentile (médiane).

précision

Paramètre facultatif qui spécifie la précision souhaitée de l'estimation du percentile. Il s'agit d'une valeur comprise entre 0 et 1, représentant l'erreur relative maximale acceptable de l'estimation. Une `accuracy` valeur inférieure se traduira par une estimation plus précise mais plus lente. Si ce paramètre n'est pas fourni, une valeur par défaut (généralement autour de 0,05 ou 5 %) est utilisée.

Renvoie

Renvoie le percentile approximatif de la colonne d'intervalle numérique ou ANSI col qui est la plus petite valeur parmi les valeurs de col ordonnées (triées de la plus petite à la plus grande), de telle sorte qu'un pourcentage maximum de valeurs de col ne soit inférieur à la valeur ou égal à cette valeur.

La valeur du pourcentage doit être comprise entre 0,0 et 1,0. Le paramètre de précision (par défaut : 10000) est un littéral numérique positif qui contrôle la précision des approximations au détriment de la mémoire.

Une valeur de précision plus élevée donne une meilleure précision, `1.0/accuracy` c'est-à-dire l'erreur relative de l'approximation.

Lorsque le pourcentage est un tableau, chaque valeur du tableau de pourcentage doit être comprise entre 0,0 et 1,0. Dans ce cas, renvoie le tableau de percentiles approximatif de la colonne col pour le tableau de pourcentages donné.

Exemples

La requête suivante estime le 95e percentile de la `response_time` colonne, avec une erreur relative maximale de 1 % (0,01).

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

La requête suivante estime les valeurs des 50e, 40e et 10e percentiles de la `col` colonne du tableau. `tab`

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

La requête suivante estime le 50e percentile (médiane) des valeurs de la colonne col.

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

Fonction AVG

La AVG fonction renvoie la moyenne (moyenne arithmétique) des valeurs des expressions d'entrée. La AVG fonction fonctionne avec des valeurs numériques et ignore les valeurs NULL.

Syntaxe

```
AVG (column)
```

Arguments

column

Colonne cible sur laquelle la fonction opère. La colonne est de l'un des types de données suivants :

- SMALLINT
- INTEGER

- BIGINT
- DECIMAL
- DOUBLE
- FLOAT

Types de données

Les types d'arguments pris en charge par la AVG fonction sont SMALLINT INTEGERBIGINT,DECIMAL, etDOUBLE.

Les types de retour pris en charge par la AVG fonction sont les suivants :

- BIGINTpour tout argument de type entier
- DOUBLEpour un argument à virgule flottante
- Renvoie le même type de données que l'expression pour tout autre type d'argument

La précision par défaut pour le résultat d'une AVG fonction avec un DECIMAL argument est de 38. L'échelle du résultat est identique à celle de l'argument. Par exemple, AVG une DEC(5,2) colonne renvoie un type de DEC(38,2) données.

Exemple

Trouvez la quantité moyenne vendue par transaction dans le SALES tableau.

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

Fonction BOOL_AND

La fonction BOOL_AND opère sur une seule colonne ou expression booléenne ou entière. Elle applique une logique similaire aux fonctions BIT_AND et BIT_OR. Pour cette fonction, le type de retour est une valeur booléenne (`true` ou `false`).

Si toutes les valeurs d'un ensemble sont true, la fonction BOOL_AND renvoie `true` (`t`). Si une valeur est false, la fonction renvoie `false` (`f`).

Syntaxe

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

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère. Cette expression doit comporter un type de données BOOLEAN ou nombre entier. Le type de retour de la fonction est BOOLEAN.

DISTINCT | ALL

Avec l'argument DISTINCT, la fonction supprime toutes les valeurs en double de l'expression spécifiée avant de calculer le résultat. Avec l'argument ALL, la fonction conserve toutes les valeurs en double. La valeur par défaut est ALL.

Exemples

Vous pouvez utiliser les fonctions booléennes par rapport à des expressions booléennes ou à des expressions de type nombre entier.

Par exemple, la requête suivante renvoie les résultats de la table USERS standard de la base de données TICKIT, qui comporte plusieurs colonnes booléennes.

La fonction BOOL_AND renvoie `false` pour les cinq lignes. Tous les utilisateurs de chacun de ces états n'aiment pas le 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)
```

Fonction BOOL_OR

La fonction BOOL_OR opère sur une seule colonne ou expression booléenne ou entière. Elle applique une logique similaire aux fonctions BIT_AND et BIT_OR. Pour cette fonction, le type de retour est une valeur booléenne (`true`, `false` ou NULL).

Si une valeur d'un ensemble est `true`, la fonction BOOL_OR renvoie `true` (`t`). Si une valeur d'un ensemble est `false`, la fonction renvoie `false` (`f`). La valeur NULL peut être renvoyée si la valeur est inconnue.

Syntaxe

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

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère. Cette expression doit comporter un type de données BOOLEAN ou nombre entier. Le type de retour de la fonction est BOOLEAN.

DISTINCT | ALL

Avec l'argument DISTINCT, la fonction supprime toutes les valeurs en double de l'expression spécifiée avant de calculer le résultat. Avec l'argument ALL, la fonction conserve toutes les valeurs en double. La valeur par défaut est ALL.

Exemples

Vous pouvez utiliser les fonctions booléennes avec des expressions booléennes ou des expressions de type nombre entier. Par exemple, la requête suivante renvoie les résultats de la table USERS standard de la base de données TICKIT, qui comporte plusieurs colonnes booléennes.

La fonction BOOL_OR renvoie `true` pour les cinq lignes. Au moins un utilisateur de chacun de ces états aime le 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)
```

L’exemple suivant renvoie la valeur NULL.

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

Fonction CARDINALITY

La fonction CARDINALITY renvoie la taille d'une expression ARRAY ou MAP (expr).

Cette fonction est utile pour déterminer la taille ou la longueur d'un tableau.

Syntaxe

```
cardinality(expr)
```

Arguments

expr

Expression ARRAY ou MAP.

Renvoie

Renvoie la taille d'un tableau ou d'une carte (INTEGER).

La fonction renvoie `NULL` une entrée nulle si elle `sizeOfNull` est définie sur `false` ou `enabled` est définie sur`true`.

Dans le cas contraire, la fonction renvoie `-1` une entrée nulle. Avec les paramètres par défaut, la fonction renvoie `-1` une entrée nulle.

Exemple

La requête suivante calcule la cardinalité, ou le nombre d'éléments, dans le tableau donné. Le tableau (`'b', 'd', 'c', 'a'`) comporte 4 éléments, donc le résultat de cette requête serait4.

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

Fonction COLLECT_LIST

La fonction COLLECT_LIST collecte et renvoie une liste d'éléments non uniques.

Ce type de fonction est utile lorsque vous souhaitez collecter plusieurs valeurs d'un ensemble de lignes dans une seule structure de données de type tableau ou liste.

> **Note**
>
> La fonction n'est pas déterministe car l'ordre des résultats collectés dépend de l'ordre des lignes, qui peut être non déterministe après l'exécution d'une opération de brassage.

Syntaxe

```
collect_list(expr)
```

Arguments

expr

Expression de n'importe quel type.

Renvoie

Renvoie un ARRAY du type d'argument. L'ordre des éléments du tableau n'est pas déterministe.

Les valeurs NULL sont exclues.

Si DISTINCT est spécifié, la fonction collecte uniquement des valeurs uniques et est synonyme de fonction d'`collect_set` agrégation.

Exemple

La requête suivante rassemble toutes les valeurs de la colonne col dans une liste. La `VALUES` clause est utilisée pour créer un tableau en ligne de trois lignes, où chaque ligne possède une seule colonne col avec les valeurs 1, 2 et 1 respectivement. La `collect_list()` fonction est ensuite utilisée pour agréger toutes les valeurs de la colonne col dans un seul tableau. La sortie de cette instruction SQL serait le tableau`[1,2,1]`, qui contient toutes les valeurs de la colonne col dans l'ordre dans lequel elles apparaissent dans les données d'entrée.

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

Fonction COLLECT_SET

La fonction COLLECT_SET collecte et renvoie un ensemble d'éléments uniques.

Cette fonction est utile lorsque vous souhaitez collecter toutes les valeurs distinctes d'un ensemble de lignes dans une structure de données unique, sans inclure de doublons.

> **Note**
>
> La fonction n'est pas déterministe car l'ordre des résultats collectés dépend de l'ordre des lignes, qui peut être non déterministe après l'exécution d'une opération de brassage.

Syntaxe

```
collect_set(expr)
```

Arguments

expr

Expression de n'importe quel type sauf MAP.

Renvoie

Renvoie un ARRAY du type d'argument. L'ordre des éléments du tableau n'est pas déterministe.

Les valeurs NULL sont exclues.

Exemple

La requête suivante rassemble toutes les valeurs uniques de la colonne col dans un ensemble. La `VALUES` clause est utilisée pour créer un tableau en ligne de trois lignes, où chaque ligne possède une seule colonne col avec les valeurs 1, 2 et 1 respectivement. La `collect_set()` fonction est ensuite utilisée pour agréger toutes les valeurs uniques de la colonne col en un seul ensemble. Le résultat de cette instruction SQL serait le set`[1,2]`, qui contient les valeurs uniques de la colonne col. La valeur dupliquée de 1 n'est incluse qu'une seule fois dans le résultat.

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

COUNTet COUNT DISTINCT fonctions

La COUNT fonction compte les lignes définies par l'expression. La COUNT DISTINCT fonction calcule le nombre de valeurs non nulles distinctes dans une colonne ou une expression. Il élimine toutes les valeurs dupliquées de l'expression spécifiée avant de procéder au décompte.

Syntaxe

```
COUNT (DISTINCT column)
```

Arguments

column

Colonne cible sur laquelle la fonction opère.

Types de données

La COUNT fonction et la COUNT DISTINCT fonction prennent en charge tous les types de données d'arguments.

La COUNT DISTINCT fonction revientBIGINT.

Exemples

Comptez tous les utilisateurs de l'État de Floride.

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

Comptez tous les lieux uniques IDs à partir de la EVENT table.

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

Fonction COUNT

La fonction COUNT compte les lignes définies par l’expression.

La fonction COUNT présente les variantes suivantes.

- COUNT (*) compte toutes les lignes de la table cible, qu'elles comprennent des valeurs null ou non.
- COUNT (expression) calcule le nombre de lignes avec des valeurs non NULL dans une colonne ou une expression spécifique.
- COUNT (DISTINCT expression) calcule le nombre de valeurs non NULL distinctes dans une colonne ou une expression.

Syntaxe

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

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

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère. La fonction COUNT prend en charge tous les types de données d'argument.

DISTINCT | ALL

Avec l'argument DISTINCT, la fonction supprime toutes les valeurs en double dans l'expression spécifiée avant d'effectuer le compte. Avec l'argument ALL, la fonction conserve toutes les valeurs en double de l'expression pour le compte. La valeur par défaut est ALL.

Type de retour

La fonction COUNT renvoie BIGINT.

Exemples

Pour compter tous les utilisateurs de l'état de Floride :

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

count
-------
```

```
510
```

Comptez tous les noms d'événements de la table EVENT :

```
select count(eventname) from event;

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

Comptez tous les noms d'événements de la table EVENT :

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

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

Comptez tous les lieux uniques dans le tableau IDs des ÉVÉNEMENTS :

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

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

Pour compter le nombre de fois où chaque vendeur a répertorié des lots de plus de quatre billets en vente. Pour regrouper les résultats de l'ID du vendeur :

```
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
...
```

Fonction MAX

La fonction MAX renvoie la valeur maximale d'un ensemble de lignes. La fonction DISTINCT ou ALL peut être utilisée, mais elle n'affecte pas le résultat.

Syntaxe

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

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère. L'expression est un type de données numérique quelconque.

DISTINCT | ALL

Avec l'argument DISTINCT, la fonction supprime toutes les valeurs en double dans l'expression spécifiée avant de calculer la valeur maximale. Avec l'argument ALL, la fonction conserve toutes les valeurs en double de l'expression pour calculer la valeur maximale. La valeur par défaut est ALL.

Types de données

Renvoie le même type de données que expression.

Exemples

Recherchez le prix le plus élevé payé de toutes les ventes :

```
select max(pricepaid) from sales;

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

Pour trouver le prix le plus élevé payé par billet de toutes les ventes :

```
select max(pricepaid/qtysold) as max_ticket_price
```

```
from sales;

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

Fonction MEDIAN

Syntaxe

```
MEDIAN ( median_expression )
```

Arguments

median_expression

Colonne cible ou expression sur laquelle la fonction opère.

Fonction MIN

La fonction MIN renvoie la valeur minimale d'un ensemble de lignes. La fonction DISTINCT ou ALL peut être utilisée, mais elle n'affecte pas le résultat.

Syntaxe

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

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère. L'expression est un type de données numérique quelconque.

DISTINCT | ALL

Avec l'argument DISTINCT, la fonction supprime toutes les valeurs en double dans l'expression spécifiée avant de calculer la valeur minimale. Avec l'argument ALL, la fonction conserve toutes les valeurs en double de l'expression pour calculer la valeur minimale. La valeur par défaut est ALL.

Types de données

Renvoie le même type de données que expression.

Exemples

Pour trouver le prix le plus bas payé de toutes les ventes :

```
select min(pricepaid) from sales;

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

Pour trouver le prix le plus bas payé par billet de toutes les ventes :

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

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

Fonction PERCENTILE

La fonction PERCENTILE est utilisée pour calculer la valeur percentile exacte en triant d'abord les valeurs dans la `col` colonne, puis en recherchant la valeur spécifiée. `percentage`

La fonction PERCENTILE est utile lorsque vous devez calculer la valeur exacte du percentile et que le coût de calcul est acceptable pour votre cas d'utilisation. Elle fournit des résultats plus précis que la fonction APPROX_PERCENTILE, mais elle peut être plus lente, en particulier pour les grands ensembles de données.

En revanche, la fonction APPROX_PERCENTILE est une alternative plus efficace qui peut fournir une estimation de la valeur du percentile avec une tolérance d'erreur spécifiée, ce qui la rend plus adaptée aux scénarios où la vitesse est une priorité supérieure à la précision absolue.

Syntaxe

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

Arguments

col

Expression ou colonne pour laquelle vous souhaitez calculer la valeur du percentile.

pourcentage

La valeur du percentile que vous souhaitez calculer, exprimée sous la forme d'une valeur comprise entre 0 et 1.

Par exemple, 0,5 correspondrait au 50e percentile (médiane).

fréquence

Paramètre facultatif qui spécifie la fréquence ou le poids de chaque valeur de la `col` colonne. S'il est fourni, la fonction calculera le percentile en fonction de la fréquence de chaque valeur.

Renvoie

Renvoie la valeur percentile exacte de la colonne d'intervalle numérique ou ANSI col au pourcentage donné.

La valeur du pourcentage doit être comprise entre 0,0 et 1,0.

La valeur de la fréquence doit être une intégrale positive

Exemple

La requête suivante trouve la valeur supérieure ou égale à 30 % des valeurs de la `col` colonne. Les valeurs étant 0 et 10, le 30e percentile est de 3,0, car il s'agit de la valeur supérieure ou égale à 30 % des données.

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

Fonction SKEWNESS

La fonction SKEWNESS renvoie la valeur d'asymétrie calculée à partir des valeurs d'un groupe.

L'asymétrie est une mesure statistique qui décrit l'asymétrie ou le manque de symétrie d'un ensemble de données. Il fournit des informations sur la forme de la distribution des données.

Cette fonction peut être utile pour comprendre les propriétés statistiques d'un ensemble de données et éclairer les analyses ultérieures ou la prise de décision.

Syntaxe

```
skewness(expr)
```

Arguments

expr

Expression dont l'évaluation est une valeur numérique.

Renvoie

Renvoie DOUBLE.

Si DISTINCT est spécifié, la fonction ne fonctionne que sur un ensemble unique de valeurs expr.

Exemples

La requête suivante calcule l'asymétrie des valeurs de la colonne. `col` Dans cet exemple, la `VALUES` clause est utilisée pour créer un tableau en ligne de quatre lignes, où chaque ligne possède une seule colonne `col` avec les valeurs -10, -20, 100 et 1000. La `skewness()` fonction est ensuite utilisée pour calculer l'asymétrie des valeurs de la colonne. `col` Le résultat, 1,1135657469022011, représente le degré et la direction de l'asymétrie des données. Une valeur d'asymétrie positive indique que les données sont inclinées vers la droite, la majeure partie des valeurs étant concentrée sur le côté gauche de la distribution. Une valeur d'asymétrie négative indique que les données sont inclinées vers la gauche, la majeure partie des valeurs étant concentrée sur le côté droit de la distribution.

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

La requête suivante calcule l'asymétrie des valeurs de la colonne col. Comme dans l'exemple précédent, la `VALUES` clause est utilisée pour créer un tableau en ligne de quatre lignes, où chaque ligne possède une seule colonne `col` avec les valeurs -1000, -100, 10 et 20. La `skewness()` fonction est ensuite utilisée pour calculer l'asymétrie des valeurs de la colonne. `col` Le résultat, -1,1135657469022011, représente le degré et la direction de l'asymétrie des données. Dans ce cas,

la valeur d'asymétrie négative indique que les données sont inclinées vers la gauche, la majeure partie des valeurs étant concentrée sur le côté droit de la distribution.

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

Fonctions STDDEV_SAMP et STDDEV_POP

Les fonctions STDDEV_SAMP et STDDEV_POP renvoient l'écart type entre l'échantillon et la population d'un ensemble de valeurs numériques (nombre entier, décimale ou à virgule flottante). Le résultat de la fonction STDDEV_SAMP est équivalent à la racine carré de la variance de l'échantillon du même ensemble de valeurs.

STDDEV_SAMP et STDDEV sont des synonymes de la même fonction.

Syntaxe

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

L'expression doit avoir un type de données numérique. Quel que soit le type de données de l'expression, le type de retour de cette fonction est un nombre double précision.

> **Note**
>
> L'écart type est calculé à l'aide de l'arithmétique à virgule flottante, qui peut se traduire par une légère imprécision.

Notes d'utilisation

Lorsque l'écart type de l'échantillon (STDDEV ou STDDEV_SAMP) est calculé pour une expression qui se compose d'une seule valeur, le résultat de la fonction est NULL, pas 0.

Exemples

La requête suivante renvoie la moyenne des valeurs de la colonne VENUESEATS de la table VENUE, suivie par l'écart type de l'échantillon et l'écart type de la population du même ensemble de valeurs. VENUESEATS est une colonne INTEGER. L'échelle du résultat est réduite à 2 chiffres.

```
select avg(venueseats),
```

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

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

La requête suivante renvoie l'écart type de l'échantillon pour la colonne COMMISSION de la table SALES. COMMISSION est une virgule DECIMAL. L'échelle du résultat est réduite à 10 chiffres.

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

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

La requête suivante convertit l'écart type de l'échantillon de la colonne COMMISSION en un nombre entier.

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

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

La requête suivante renvoie l'écart type de l'échantillon et la racine carré de la variance de l'échantillon pour la colonne COMMISSION. Les résultats de ces calculs sont identiques.

```
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)
```

SUMet SUM DISTINCT fonctions

La SUM fonction renvoie la somme des valeurs de colonne ou d'expression en entrée. La SUM fonction fonctionne avec des valeurs numériques et ignore NULL les valeurs.

La SUM DISTINCT fonction élimine toutes les valeurs dupliquées de l'expression spécifiée avant de calculer la somme.

Syntaxe

```
SUM (DISTINCT column )
```

Arguments

column

Colonne cible sur laquelle la fonction opère. La colonne contient tous les types de données numériques.

Exemples

Trouvez la somme de toutes les commissions payées dans le SALES tableau.

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

Trouvez la somme de toutes les commissions distinctes payées dans le SALES tableau.

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

Fonctions VAR_SAMP et VAR_POP

Les fonctions VAR_SAMP et VAR_POP renvoient la variance entre l'échantillon et la population d'un ensemble de valeurs numériques (nombre entier, décimale ou à virgule flottante). Le résultat de la fonction VAR_SAMP est équivalent au carré de l'écart type de l'échantillon du même ensemble de valeurs.

VAR_SAMP et VARIANCE sont des synonymes de la même fonction.

Syntaxe

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

L'expression doit comporter un type de données de nombre entier, décimale ou à virgule flottante. Quel que soit le type de données de l'expression, le type de retour de cette fonction est un nombre double précision.

> **Note**
>
> Les résultats de ces fonctions peuvent varier entre les clusters d'entrepôts des données, en fonction de la configuration du cluster dans chaque cas.

Notes d'utilisation

Lorsque l'écart type de l'échantillon (VARIANCE ou VAR_SAMP) est calculé pour une expression qui se compose d'une valeur unique, le résultat de la fonction est NULL pas 0.

Exemples

La requête suivante renvoie la variance arrondie entre l'échantillon et la population de la colonne NUMTICKETS dans la table LISTING.

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

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

La requête suivante exécute les mêmes calculs mais traduit les résultats en valeur décimales.

```
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)
```

Fonctions de tableau

Cette section décrit les fonctions de tableau pour SQL prises en charge dans AWS Clean Rooms.

Rubriques

- Fonction ARRAY
- Fonction ARRAY_CONTAINS
- Fonction ARRAY_DISTINCT
- Fonction ARRAY_EXCEPT
- Fonction ARRAY_INTERSECT
- Fonction ARRAY_JOIN
- Fonction ARRAY_REMOVE
- Fonction ARRAY_UNION
- Fonction EXPLODE
- Fonction FLATTEN

Fonction ARRAY

Crée un tableau avec les éléments donnés.

Syntaxe

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

Argument

expr1, expr2

Expressions de tous types de données, à l'exception des types de date et d'heure. Les arguments ne doivent pas nécessairement être du même type de données.

Type de retour

La fonction array renvoie un ARRAY contenant les éléments de l'expression.

exemple

L'exemple suivant montre un tableau de valeurs numériques et un tableau de différents types de données.

```
--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)
```

Fonction ARRAY_CONTAINS

La fonction ARRAY_CONTAINS peut être utilisée pour effectuer des vérifications d'appartenance de base sur les structures de données des tableaux. La fonction ARRAY_CONTAINS est utile lorsque vous devez vérifier si une valeur spécifique est présente dans un tableau.

Syntaxe

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

Arguments

réseau

Un ARRAY à rechercher.

valeur

Expression dont le type partage le type le moins courant avec les éléments du tableau.

Type de retour

La fonction ARRAY_CONTAINS renvoie une valeur BOOLEAN.

Si la valeur est NULL, le résultat est NULL.

Si un élément du tableau est NULL, le résultat est NULL si la valeur ne correspond à aucun autre élément.

Exemples

L'exemple suivant vérifie si le tableau `[1, 2, 3]` contient la valeur4. Comme le `[1, 2, 3` [array] ne contient pas la valeur4, la fonction array_contains revient. `false`

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

L'exemple suivant vérifie si le tableau `[1, 2, 3]` contient la valeur2. Comme le tableau `[1, 2, 3]` contient la valeur2, la fonction array_contains est renvoyée. `true`

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

Fonction ARRAY_DISTINCT

La fonction ARRAY_DISTINCT peut être utilisée pour supprimer les valeurs dupliquées d'un tableau. La fonction ARRAY_DISTINCT est utile lorsque vous devez supprimer les doublons d'un tableau et utiliser uniquement les éléments uniques. Cela peut être utile dans les scénarios où vous souhaitez effectuer des opérations ou des analyses sur un ensemble de données sans l'interférence de valeurs répétées.

Syntaxe

```
array_distinct(array)
```

Arguments

réseau

Expression ARRAY.

Type de retour

La fonction ARRAY_DISTINCT renvoie un ARRAY contenant uniquement les éléments uniques du tableau d'entrée.

Exemples

Dans cet exemple, le tableau d'entrée `[1, 2, 3, null, 3]` contient une valeur dupliquée de3. La `array_distinct` fonction supprime cette valeur dupliquée 3 et renvoie un nouveau tableau contenant les éléments uniques :`[1, 2, 3, null]`.

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

Dans cet exemple, le tableau d'entrée `[1, 2, 2, 3, 3, 3]` contient des valeurs dupliquées de 2 et3. La `array_distinct` fonction supprime ces doublons et renvoie un nouveau tableau avec les éléments uniques :. `[1, 2, 3]`

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

Fonction ARRAY_EXCEPT

La fonction ARRAY_EXCEPT prend deux tableaux comme arguments et renvoie un nouveau tableau contenant uniquement les éléments présents dans le premier tableau, mais pas dans le second tableau.

Le ARRAY_EXCEPT est utile lorsque vous devez trouver les éléments uniques d'un tableau par rapport à un autre. Cela peut être utile dans les scénarios où vous devez effectuer des opérations similaires à des ensembles sur des tableaux, telles que la recherche de la différence entre deux ensembles de données.

Syntaxe

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

Arguments

matrice1

Un ARRAY de n'importe quel type avec des éléments comparables.

tableau 2

Un TABLEAU d'éléments partageant le type le moins commun avec les éléments de array1.

Type de retour

La fonction ARRAY_EXCEPT renvoie un ARRAY de type correspondant à array1 sans doublons.

Exemples

Dans cet exemple, le premier tableau `[1, 2, 3]` contient les éléments 1, 2 et 3. Le second tableau `[2, 3, 4]` contient les éléments 2, 3 et 4. La `array_except` fonction supprime les éléments 2 et 3 du premier tableau, puisqu'ils sont également présents dans le second tableau. Le résultat obtenu est le tableau`[1]`.

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

Dans cet exemple, le premier tableau `[1, 2, 3]` contient les éléments 1, 2 et 3. Le second tableau `[1, 3, 5]` contient les éléments 1, 3 et 5. La `array_except` fonction supprime les éléments 1 et 3 du premier tableau, puisqu'ils sont également présents dans le second tableau. Le résultat obtenu est le tableau`[2]`.

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

Fonction ARRAY_INTERSECT

La fonction ARRAY_INTERSECT prend deux tableaux comme arguments et renvoie un nouveau tableau contenant les éléments présents dans les deux tableaux d'entrée. Cette fonction est utile lorsque vous devez trouver les éléments communs entre deux tableaux. Cela peut être utile dans les scénarios où vous devez effectuer des opérations similaires à des ensembles sur des tableaux, telles que la recherche de l'intersection entre deux ensembles de données.

Syntaxe

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

Arguments

matrice1

Un ARRAY de n'importe quel type avec des éléments comparables.

tableau 2

Un TABLEAU d'éléments partageant le type le moins commun avec les éléments de array1.

Type de retour

La fonction ARRAY_INTERSECT renvoie un ARRAY de type correspondant à array1 sans doublons et sans éléments contenus à la fois dans array1 et array2.

Exemples

Dans cet exemple, le premier tableau `[1, 2, 3]` contient les éléments 1, 2 et 3. Le second tableau `[1, 3, 5]` contient les éléments 1, 3 et 5. La fonction ARRAY_INTERSECT identifie les éléments communs entre les deux tableaux, à savoir 1 et 3. Le tableau de sortie résultant est`[1, 3]`.

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

Fonction ARRAY_JOIN

La fonction ARRAY_JOIN prend deux arguments : le premier argument est le tableau d'entrée qui sera joint. Le deuxième argument est la chaîne de séparation qui sera utilisée pour concaténer les éléments du tableau. Cette fonction est utile lorsque vous devez convertir un tableau de chaînes (ou tout autre type de données) en une seule chaîne concaténée. Cela peut être utile dans les scénarios où vous souhaitez présenter un tableau de valeurs sous la forme d'une seule chaîne formatée, par exemple à des fins d'affichage ou pour un traitement ultérieur.

Syntaxe

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

Arguments

réseau

N'importe quel type ARRAY, mais ses éléments sont interprétés comme des chaînes.

delimiter

Une CHAÎNE utilisée pour séparer les éléments du tableau concaténé.

Remplacement nul

Chaîne utilisée pour exprimer une valeur NULL dans le résultat.

Type de retour

La fonction ARRAY_JOIN renvoie une chaîne dans laquelle les éléments du tableau sont séparés par un délimiteur et les éléments nuls sont remplacés par des éléments nuls. `nullReplacement` Si `nullReplacement` ce paramètre est omis, `null` les éléments sont filtrés. Si un argument l'est`NULL`, le résultat est`NULL`.

Exemples

Dans cet exemple, la fonction ARRAY_JOIN prend le tableau `['hello', 'world']` et joint les éléments à l'aide du séparateur `' '` (un espace). Le résultat obtenu est la chaîne de caractères`'hello world'`.

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

Dans cet exemple, la fonction ARRAY_JOIN prend le tableau `['hello', null, 'world']` et joint les éléments à l'aide du séparateur `' '` (un espace). La `null` valeur est remplacée par la chaîne de remplacement fournie `','` (une virgule). Le résultat obtenu est la chaîne de caractères`'hello , world'`.

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

Fonction ARRAY_REMOVE

La fonction ARRAY_REMOVE accepte deux arguments : le premier argument est le tableau d'entrée dont les éléments seront supprimés. Le deuxième argument est la valeur qui sera supprimée du tableau. Cette fonction est utile lorsque vous devez supprimer des éléments spécifiques d'un tableau. Cela peut être utile dans les scénarios où vous devez effectuer un nettoyage ou un prétraitement des données sur un tableau de valeurs.

Syntaxe

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

Arguments

réseau

Un ARRAY.

élément

Expression d'un type partageant le type le moins commun avec les éléments du tableau.

Type de retour

La fonction ARRAY_REMOVE renvoie le type de résultat correspondant au type du tableau. Si l'élément à supprimer l'est`NULL`, le résultat est`NULL`.

Exemples

Dans cet exemple, la fonction ARRAY_REMOVE prend le tableau `[1, 2, 3, null, 3]` et supprime toutes les occurrences de la valeur 3. Le résultat obtenu est le tableau`[1, 2, null]`.

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

Fonction ARRAY_UNION

La fonction ARRAY_UNION prend deux tableaux comme arguments et renvoie un nouveau tableau contenant les éléments uniques des deux tableaux d'entrée. Cette fonction est utile lorsque vous devez combiner deux tableaux et éliminer les éléments dupliqués. Cela peut être utile dans les scénarios où vous devez effectuer des opérations similaires à des ensembles sur des tableaux, telles que la recherche de l'union entre deux ensembles de données.

Syntaxe

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

Arguments

matrice1

Un ARRAY.

tableau 2

Un ARRAY du même type que array1.

Type de retour

La fonction ARRAY_UNION renvoie un ARRAY du même type qu'un tableau.

exemple

Dans cet exemple, le premier tableau `[1, 2, 3]` contient les éléments 1, 2 et 3. Le second tableau `[1, 3, 5]` contient les éléments 1, 3 et 5. La fonction ARRAY_UNION combine les éléments uniques des deux tableaux pour obtenir le tableau de sortie. `[1, 2, 3, 5]` T

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

Fonction EXPLODE

La fonction EXPLODE est utilisée pour transformer une seule ligne contenant un tableau ou une colonne de carte en plusieurs lignes, chaque ligne correspondant à un seul élément du tableau ou de la carte.

Syntaxe

```
explode(expr)
```

Arguments

expr

Expression matricielle ou expression cartographique.

Type de retour

La fonction EXPLODE renvoie un ensemble de lignes, chaque ligne représentant un élément unique du tableau ou de la carte en entrée.

Le type de données des lignes de sortie dépend du type de données des éléments du tableau ou de la carte en entrée.

Exemples

L'exemple suivant prend le tableau à une seule ligne [10, 20] et le transforme en deux lignes distinctes, chacune contenant l'un des éléments du tableau (10 et 20).

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

Dans le premier exemple, le tableau d'entrée a été directement transmis en tant qu'argument à`explode()`. Dans cet exemple, le tableau d'entrée est spécifié à l'aide de la => syntaxe, où le nom de colonne (`collection`) est explicitement fourni.

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

Les deux approches sont valides et permettent d'obtenir le même résultat, mais la seconde syntaxe peut être plus utile lorsque vous devez faire exploser une colonne d'un ensemble de données plus important, plutôt qu'un simple tableau littéral.

Fonction FLATTEN

La fonction FLATTEN est utilisée pour « aplatir » une structure de tableau imbriqué en un seul tableau plat.

Syntaxe

```
flatten(arrayOfArrays)
```

Arguments

arrayOfArrays

Un tableau de tableaux.

Type de retour

La fonction FLATTEN renvoie un tableau.

exemple

Dans cet exemple, l'entrée est un tableau imbriqué avec deux tableaux internes, et la sortie est un seul tableau plat contenant tous les éléments des tableaux internes. La fonction FLATTEN prend le tableau imbriqué `[[1, 2], [3, 4]]` et combine tous les éléments en un seul tableau. `[1, 2, 3, 4]`

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

Expressions conditionnelles

En SQL, les expressions conditionnelles sont utilisées pour prendre des décisions en fonction de certaines conditions. Ils vous permettent de contrôler le flux de vos instructions SQL et de renvoyer différentes valeurs ou d'effectuer différentes actions en fonction de l'évaluation d'une ou de plusieurs conditions.

AWS Clean Rooms prend en charge les expressions conditionnelles suivantes :

Rubriques

- Expression conditionnelle CASE
- COALESCEexpression
- La plus grande et la moins grande expression
- Expression IF
- Expression IS_NULL
- Expression IS_NOT_NULL
- Fonctions NVL et COALESCE
- NVL2 fonction
- Fonction NULLIF

Expression conditionnelle CASE

L'expression CASE est une expression conditionnelle, similaire aux if/then/else instructions trouvées dans d'autres langages. L’expression CASE est utilisée pour spécifier un résultat lorsqu’il y a plusieurs conditions. Utilisez CASE là où l’utilisation d’une expression SQL est valide, par exemple dans une commande SELECT.

Il existe deux types d’expressions CASE : simple et recherchée.

- Dans les expressions CASE simples, une expression est comparée à une valeur. Lorsqu’une correspondance est trouvée, l’action spécifiée dans la clause THEN est appliquée. Si aucune correspondance n’est trouvée, l’action de la clause ELSE est appliquée.
- Dans les expressions CASE recherchées, chaque expression CASE est évaluée en fonction d’une expression booléenne, et l’instruction CASE renvoie la première expression CASE correspondante. Si aucune correspondance n’est trouvée parmi les clauses WHEN, l’action contenue dans la clause ELSE est renvoyée.

Syntaxe

Instruction CASE simple utilisée pour mettre en correspondance des conditions :

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

Instructions CASE recherchées utilisées pour évaluer chaque condition :

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

Arguments

expression

Nom de la colonne ou n’importe quelle expression valide.

valeur

Valeur à laquelle l'expression est comparée, par exemple une constante numérique ou une chaîne de caractères.

result

Valeur ou expression cible qui est renvoyée lorsqu'une expression ou une condition booléenne est évaluée. Les types de données de toutes les expressions de résultat doivent pouvoir être convertis en un seul type de sortie.

condition

Expression booléenne qui prend la valeur true ou false. Si la condition a la valeur true, la valeur de l'expression CASE est le résultat qui suit la condition, et le reste de l'expression CASE n'est pas traité. Si la condition a la valeur false, les clauses WHEN suivantes sont évaluées. Si aucune condition WHEN n'a la valeur true en résultat, la valeur de l'expression CASE est le résultat de la clause ELSE. Si la clause ELSE est omise et qu'aucune condition n'a la valeur true, le résultat est null.

Exemples

Utilisez une expression CASE simple pour remplacer `New York City` par `Big Apple` dans une requête sur la table de VENUE. Remplacer tous les autres noms de villes par `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
...
```

Utiliser une expression CASE recherchée pour affecter des numéros de groupes basés sur la valeur PRICEPAID pour les vente de billets individuelles :

```
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
...
```

COALESCEexpression

Une COALESCE expression renvoie la valeur de la première expression de la liste qui n'est pas nulle. Si toutes les expressions régulières sont null, le résultat est null. Lorsqu'une valeur non null est trouvée, les expressions restantes de la liste ne sont pas évaluées.

Ce type d'expression s'avère utile lorsque vous souhaitez renvoyer une valeur de sauvegarde pour quelque chose lorsque la valeur préférée est manquante ou null. Par exemple, une requête peut renvoyer un des trois numéros de téléphone (portable, maison ou professionnel, dans l'ordre), selon ce qui est trouvé en premier dans le tableau (non null).

Syntaxe

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

Exemples

Appliquez COALESCE l'expression à deux colonnes.

```
select coalesce(start_date, end_date)
from datetable
```

```
order by 1;
```

Le nom de colonne par défaut pour une expression NVL estCOALESCE. La requête suivante renvoie les mêmes résultats.

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

La plus grande et la moins grande expression

Renvoie la valeur la plus grande ou la plus petite d'une liste d'un nombre quelconque d'expressions.

Syntaxe

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

Paramètres

expression_list

Liste d'expressions séparées par des virgules, telles que des noms de colonnes. Les expressions doivent toutes être converties dans un type de données commun. Les valeurs NULL de la liste sont ignorées. Si toutes les expressions sont évaluées à NULL, le résultat est NULL.

Renvoie

Renvoie la plus grande (pour GREATEST) ou la plus petite (pour LEAST) valeur de la liste d'expressions fournie.

exemple

L'exemple suivant renvoie la valeur la plus élevée dans l'ordre alphabétique pour `firstname` ou `lastname`.

```
select firstname, lastname, greatest(firstname,lastname) from users
where userid < 10
order by 3;

 firstname | lastname  | greatest
```

```
-----------+-----------+-----------
 Alejandro | Rosalez   | Ratliff
 Carlos    | Salazar   | Carlos
 Jane      | Doe       | Doe
 John      | Doe       | Doe
 John      | Stiles    | John
 Shirley   | Rodriguez | Rodriguez
 Terry     | Whitlock  | Terry
 Richard   | Roe       | Richard
 Xiulan    | Wang      | Wang
(9 rows)
```

Expression IF

La fonction conditionnelle IF renvoie l'une des deux valeurs en fonction d'une condition.

Cette fonction est une instruction de flux de contrôle courante utilisée dans SQL pour prendre des décisions et renvoyer différentes valeurs en fonction de l'évaluation d'une condition. C'est utile pour implémenter une logique if-else simple dans une requête.

Syntaxe

```
if(expr1, expr2, expr3)
```

Arguments

expr1

Condition ou expression évaluée. Si c'est le cas`true`, la fonction renverra la valeur de expr2. Si expr1 l'est*`false`*, la fonction renverra la valeur de expr3.

expr2

Expression évaluée et renvoyée si expr1 l'est. `true`

expr3

Expression évaluée et renvoyée si expr1 l'est. `false`

Renvoie

Si la `expr1` valeur est égale à`true`, renvoie `expr2` ; dans le cas contraire, renvoie`expr3`.

exemple

L'exemple suivant utilise la `if()` fonction pour renvoyer l'une des deux valeurs en fonction d'une condition. La condition évaluée est `1 < 2true`, c'est-à-dire que la première valeur `'a'` est renvoyée.

```
SELECT if(1 < 2, 'a', 'b');
 a]
```

Expression IS_NULL

L'expression IS_NULL conditionnelle est utilisée pour vérifier si une valeur est nulle.

Cette expression est synonyme deIS NULL.

Syntaxe

```
is_null(expr)
```

Arguments

expr

Expression de n'importe quel type.

Renvoie

L'expression IS_NULL conditionnelle renvoie une valeur booléenne. Si la valeur `expr1` est NULL`true`, renvoie, sinon renvoie`false`.

Exemples

L'exemple suivant vérifie si la valeur 1 est nulle et renvoie le résultat booléen `true` car 1 est une valeur valide et non nulle.

```
SELECT is not null(1);
 true
```

L'exemple suivant sélectionne la `id` colonne dans le `squirrels` tableau, mais uniquement pour les lignes où se trouve la colonne d'âge`null`.

```
SELECT id FROM squirrels WHERE is_null(age)
```

Expression IS_NOT_NULL

L'expression IS_NOT_NULL conditionnelle est utilisée pour vérifier si une valeur n'est pas nulle.

Cette expression est synonyme deIS NOT NULL.

Syntaxe

```
is_not_null(expr)
```

Arguments

expr

Expression de n'importe quel type.

Renvoie

L'expression IS_NOT_NULL conditionnelle renvoie une valeur booléenne. Si `expr1` ce n'est pas NULL, renvoie`true`, sinon renvoie`false`.

Exemples

L'exemple suivant vérifie si la valeur n'1est pas nulle et renvoie le résultat booléen `true` car 1 est une valeur valide et non nulle.

```
SELECT is not null(1);
 true
```

L'exemple suivant sélectionne la `id` colonne dans le `squirrels` tableau, mais uniquement pour les lignes où la colonne d'âge ne figure pas`null`.

```
SELECT id FROM squirrels WHERE is_not_null(age)
```

Fonctions NVL et COALESCE

Renvoie la valeur de la première expression qui n’est pas nulle dans une série d’expressions. Lorsqu’une valeur non nulle est trouvée, les expressions restantes de la liste ne sont pas évaluées.

NVL est identique à COALESCE. Ce sont des synonymes. Cette rubrique explique la syntaxe et contient des exemples pour les deux.

Syntaxe

```
NVL( expression, expression, ... )
```

La syntaxe de COALESCE est identique :

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

Si toutes les expressions régulières sont null, le résultat est null.

Ces fonctions sont utiles lorsque vous souhaitez renvoyer une valeur secondaire lorsqu'une valeur primaire est manquante ou nulle. Par exemple, une requête peut renvoyer le premier des trois numéros de téléphone disponibles : portable, domicile ou travail. L'ordre des expressions dans la fonction détermine l'ordre d'évaluation.

Arguments

expression

Expression, telle qu'un nom de colonne, à évaluer pour l'état null.

Type de retour

AWS Clean Rooms détermine le type de données de la valeur renvoyée en fonction des expressions d'entrée. Si les types de données des expressions d'entrée n'ont pas de type commun, une erreur est renvoyée.

Exemples

Si la liste contient des expressions entières, la fonction renvoie un entier.

```
SELECT COALESCE(NULL, 12, NULL);

coalesce
--------------
12
```

Cet exemple, qui est identique à l'exemple précédent, sauf qu'il utilise NVL, renvoie le même résultat.

```
SELECT NVL(NULL, 12, NULL);

coalesce
--------------
12
```

L'exemple suivant renvoie une chaîne de caractères.

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', NULL);

coalesce
--------------
AWS Clean Rooms
```

L'exemple suivant génère une erreur car les types de données varient dans la liste d'expressions. Dans ce cas, la liste contient à la fois un type de chaîne et un type de nombre.

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', 12);
ERROR: invalid input syntax for integer: "AWS Clean Rooms"
```

NVL2 fonction

Renvoie l'une des deux valeurs selon qu'une expression spécifiée a pour valeur NULL ou NOT NULL.

Syntaxe

```
NVL2 ( expression, not_null_return_value, null_return_value )
```

Arguments

expression

Expression, telle qu'un nom de colonne, à évaluer pour l'état null.

not_null_return_value

Valeur renvoyée si expression a une valeur NOT NULL. La valeur not_null_return_value doit avoir le même type de données que expression ou être implicitement convertie en ce type de données.

null_return_value

Valeur renvoyée si expression a une valeur NULL. La valeur null_return_value doit avoir le même type de données que expression ou être implicitement convertie en ce type de données.

Type de retour

Le type de NVL2 retour est déterminé comme suit :

- Si not_null_return_value ou null_return_value a une valeur null, le type de données de l'expression non-null est renvoyé.

Si not_null_return_value et null_return_value n'ont pas de valeur null :

- Si not_null_return_value et null_return_value ont le même type de données que le type de données renvoyé.
- Si not_null_return_value et null_return_value ont des types de données numériques distincts, le type de données numériques compatible le plus petit est renvoyé.
- Si not_null_return_value et null_return_value ont des types de données datetime distincts, un type de données d'horodatage est renvoyé.
- Si not_null_return_value et null_return_value ont des types de données de caractères distincts, le type de données not_null_return_value est renvoyé.
- Si not_null_return_value et null_return_value ont des types de données numériques et non numériques mixtes, le type de données de not_null_return_value est renvoyé.

> **Important**
>
> Dans les deux derniers cas où le type de données de not_null_return_value est renvoyé, null_return_value est converti implicitement en ce type de données. Si les types de données sont incompatibles, la fonction échoue.

Notes d'utilisation

En effet NVL2, le retour aura la valeur du paramètre not_null_return_value ou null_return_value, selon ce qui est sélectionné par la fonction, mais aura le type de données not_null_return_value.

Par exemple, en supposant que column1 a la valeur NULL, les requêtes suivantes renverront la même valeur. Cependant, le type de données de la valeur de retour DECODE sera INTEGER et le type de données de la valeur de NVL2 retour sera VARCHAR.

```
select decode(column1, null, 1234, '2345');
```

```
select nvl2(column1, '2345', 1234);
```

exemple

L'exemple suivant modifie quelques exemples de données, puis évalue les deux champs pour fournir des informations de contact aux utilisateurs :

```
update users set email = null where firstname = 'Aphrodite' and lastname = 'Acevedo';

select (firstname + ' ' + lastname) as name,
nvl2(email, email, phone) AS contact_info
from users
where state = 'WA'
and lastname  like 'A%'
order by lastname, firstname;

name        contact_info
-------------------+-----------------------------------------
Aphrodite Acevedo (555) 555-0100
Caldwell Acevedo  Nunc.sollicitudin@example.ca
Quinn Adams     vel@example.com
Kamal Aguilar   quis@example.com
Samson Alexander  hendrerit.neque@example.com
Hall Alford     ac.mattis@example.com
Lane Allen      et.netus@example.com
Xander Allison    ac.facilisis.facilisis@example.com
Amaya Alvarado    dui.nec.tempus@example.com
Vera Alvarez    at.arcu.Vestibulum@example.com
Yetta Anthony   enim.sit@example.com
Violet Arnold   ad.litora@example.comm
August Ashley   consectetuer.euismod@example.com
Karyn Austin    ipsum.primis.in@example.com
Lucas Ayers     at@example.com
```

Fonction NULLIF

Compare les deux arguments et renvoie null si les arguments sont égaux. S'ils ne sont pas égaux, le premier argument est renvoyé.

Syntaxe

L’expression NULLIF compare les deux arguments et renvoie la valeur nulle si les arguments sont égaux. S'ils ne sont pas égaux, le premier argument est renvoyé. Cette expression est l’inverse de l’expression NVL ou COALESCE.

```
NULLIF ( expression1, expression2 )
```

Arguments

expression1, expression2

Colonnes ou expressions cible qui sont comparées. Le type de retour est le identique au type de la première expression.

Exemples

Dans l’exemple suivant, la requête renvoie la chaîne `first` car les arguments ne sont pas égaux.

```
SELECT NULLIF('first', 'second');

case
-------
first
```

Dans l’exemple suivant, la requête renvoie NULL car les arguments littéraux de la chaîne sont égaux.

```
SELECT NULLIF('first', 'first');

case
-------
NULL
```

Dans l’exemple suivant, la requête renvoie 1 car les arguments entiers ne sont pas égaux.

```
SELECT NULLIF(1, 2);

case
-------
```

```
1
```

Dans l'exemple suivant, la requête renvoie NULL car les arguments entiers sont égaux.

```
SELECT NULLIF(1, 1);

case
-------
NULL
```

Dans l'exemple suivant, la requête renvoie la valeur nulle lorsque les valeurs LISTID et SALESID correspondent :

```
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)
```

Fonctions de constructeur

Une fonction de constructeur SQL est une fonction utilisée pour créer de nouvelles structures de données, telles que des tableaux ou des cartes.

Ils prennent des valeurs d'entrée et renvoient un nouvel objet de structure de données. Les fonctions de constructeur sont généralement nommées d'après le type de données qu'elles créent, tel que ARRAY ou MAP.

Les fonctions de constructeur sont différentes des fonctions scalaires ou des fonctions d'agrégation, qui opèrent sur des données existantes et renvoient une valeur unique. Les fonctions du constructeur

sont utilisées pour créer de nouvelles structures de données qui peuvent ensuite être utilisées dans le cadre d'un traitement ou d'une analyse de données ultérieurs.

AWS Clean Rooms prend en charge les fonctions de constructeur suivantes :

Rubriques

Fonction constructeur MAP

La fonction constructeur MAP crée une carte avec les paires clé/valeur données.

Les fonctions de constructeur telles que MAP sont utiles lorsque vous devez créer de nouvelles structures de données par programmation dans vos requêtes SQL. Ils vous permettent de créer des structures de données complexes qui peuvent être utilisées dans le cadre d'un traitement ou d'une analyse de données ultérieurs.

Syntaxe

```
map(key0, value0, key1, value1, ...)
```

Arguments

clé0

Expression de n'importe quel type comparable. Toutes les clés 0 doivent partager le type le moins commun.

valeur0

Expression de n'importe quel type. Toutes les valeurs N doivent partager le type le moins commun.

Renvoie

La fonction MAP renvoie une carte dont les touches sont saisies comme le type le moins courant de clé0 et les valeurs saisies comme le type le moins courant de valeur0.

Exemples

L'exemple suivant crée une nouvelle carte avec deux paires clé-valeur : La clé `1.0` est associée à la valeur. `'2'` La clé `3.0` est associée à la valeur`'4'`. La carte obtenue est ensuite renvoyée en tant que sortie de l'instruction SQL.

```
SELECT map(1.0, '2', 3.0, '4');
 {1.0:"2",3.0:"4"}
```

Fonction constructeur NAMED_STRUCT

La fonction constructeur NAMED_STRUCT crée une structure avec les noms et valeurs de champs donnés.

Les fonctions de constructeur telles que NAMED_STRUCT sont utiles lorsque vous devez créer de nouvelles structures de données par programmation dans vos requêtes SQL. Ils vous permettent de créer des structures de données complexes, telles que des structures ou des enregistrements, qui peuvent être utilisées dans le cadre d'un traitement ou d'une analyse de données ultérieurs.

Syntaxe

```
named_struct(name1, val1, name2, val2, ...)
```

Arguments

nom1

Un champ de dénomination littéral STRING 1.

val1

Expression de n'importe quel type spécifiant la valeur du champ 1.

Renvoie

La fonction NAMED_STRUCT renvoie une structure dont le champ 1 correspond au type de val1.

Exemples

L'exemple suivant crée une nouvelle structure avec trois champs nommés : La valeur `"a" 1` est attribuée au champ. La valeur `"b"` est affectée au champ `2.` Le champ `"c"` reçoit la valeur`3`. La structure résultante est ensuite renvoyée en tant que sortie de l'instruction SQL.

```
SELECT named_struct("a", 1, "b", 2, "c", 3);
 {"a":1,"b":2,"c":3}
```

Fonction constructeur STRUCT

La fonction constructeur STRUCT crée une structure avec les valeurs de champ données.

Les fonctions de constructeur telles que STRUCT sont utiles lorsque vous devez créer de nouvelles structures de données par programmation dans vos requêtes SQL. Ils vous permettent de créer des structures de données complexes, telles que des structures ou des enregistrements, qui peuvent être utilisées dans le cadre d'un traitement ou d'une analyse de données ultérieurs.

Syntaxe

```
struct(col1, col2, col3, ...)
```

Arguments

colonel 1

Nom de la colonne ou n’importe quelle expression valide.

Renvoie

La fonction STRUCT renvoie une structure dont field1 correspond au type de expr1.

Si les arguments sont des références nommées, les noms sont utilisés pour nommer le champ. Sinon, les champs sont nommés ColN, où N est la position du champ dans la structure.

Exemples

L'exemple suivant crée une nouvelle structure avec trois champs : La valeur 1 est attribuée au premier champ. La valeur 2 est attribuée au second champ. La valeur 3 est attribuée au troisième champ. Par défaut, les champs de la structure résultante sont nommés `col1` `col2col3`, et en fonction de leur position dans la liste d'arguments. La structure résultante est ensuite renvoyée en tant que sortie de l'instruction SQL.

```
SELECT struct(1, 2, 3);
```

```
{"col1":1,"col2":2,"col3":3}
```

Fonctions de formatage des types de données

À l'aide d'une fonction de formatage des types de données, vous pouvez convertir des valeurs d'un type de données à un autre. Pour chacune de ces fonctions, le premier argument est toujours la valeur à formater et le second argument contient le modèle du nouveau format.

AWS Clean Rooms Spark SQL prend en charge plusieurs fonctions de formatage des types de données.

Rubriques

BASE64 fonction

La BASE64 fonction convertit une expression en chaîne de base 64 à l'aide du codage de transfert RFC2045 Base64 pour le MIME.

Syntaxe

```
base64(expr)
```

Arguments

expr

Une expression BINAIRE ou une CHAÎNE que la fonction interprétera comme BINAIRE.

Type de retour

`STRING`

Exemple

Pour convertir l'entrée de chaîne donnée en sa représentation codée en Base64, utilisez l'exemple suivant. Le résultat est la représentation codée en Base64 de la chaîne d'entrée « Spark SQL », qui est « u3bhcmsgu1fm ».

```
SELECT base64('Spark SQL');
 U3BhcmsgU1FM
```

Fonction CAST

La fonction CAST convertit un type de données en un autre type de données compatible. Par exemple, vous pouvez convertir une chaîne en date ou un type numérique en chaîne. CAST effectue une conversion d'exécution, ce qui signifie que la conversion ne modifie pas le type de données d'une valeur dans une table source. Elle n'est modifiée que dans le contexte de la requête.

Certains types de données nécessitent une conversion explicite en d'autres types de données à l'aide de la fonction CAST. D'autres types de données peuvent être convertis implicitement, dans le cadre d'une autre commande, sans utiliser CAST. Consultez Compatibilité et conversion de types.

Syntaxe

Utilisez l'une de ces deux formes de syntaxes équivalentes pour convertir les expressions cast d'un type de données à un autre.

```
CAST ( expression AS type )
```

Arguments

expression

Expression qui correspond à une ou plusieurs valeurs, par exemple un nom de colonne ou un littéral. La conversion de valeurs null renvoie des valeurs null. L'expression ne peut pas contenir de chaînes vides ou vides.

type

L'un des types de données pris en chargeTypes de données, à l'exception des types de données BINARY et BINARY VARIING.

Type de retour

CAST renvoie le type de données spécifié par l'argument type.

> **Note**
>
> AWS Clean Rooms renvoie une erreur si vous essayez d'effectuer une conversion problématique, telle qu'une conversion DECIMAL qui perd en précision, comme suit :
>
> ```
> select 123.456::decimal(2,1);
> ```
>
> ou une conversion INTEGER qui entraîne un dépassement de capacité :
>
> ```
> select 12345678::smallint;
> ```

Exemples

Les deux requêtes suivantes sont équivalentes. Toutes deux convertissent une valeur décimale en un nombre entier :

```
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)
```

Ce qui suit produit un résultat similaire. Il ne nécessite pas d'exemples de données pour s'exécuter :

```
select cast(162.00 as integer) as pricepaid;

pricepaid
-----------
162
(1 row)
```

Dans cet exemple, les valeurs d'une colonne d'horodatage sont converties en dates, ce qui entraîne la suppression de l'horodatage de chaque résultat :

```
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)
```

Si vous n'avez pas utilisé CAST comme illustré dans l'exemple précédent, les résultats incluraient l'heure : 2008-02-18 02:36:48.

La requête suivante convertit des données de caractères variables en date. Elle ne nécessite pas d'exemples de données pour s'exécuter.

```
select cast('2008-02-18 02:36:48' as date) as mysaletime;

mysaletime
--------------------
2008-02-18
(1 row)
```

Dans cet exemple, les valeurs d'une colonne de dates sont converties en horodatages :

```
select cast(caldate as timestamp), dateid
from date order by dateid limit 10;

      caldate       | dateid
--------------------+--------
2008-01-01 00:00:00 |   1827
2008-01-02 00:00:00 |   1828
2008-01-03 00:00:00 |   1829
2008-01-04 00:00:00 |   1830
2008-01-05 00:00:00 |   1831
2008-01-06 00:00:00 |   1832
2008-01-07 00:00:00 |   1833
2008-01-08 00:00:00 |   1834
2008-01-09 00:00:00 |   1835
2008-01-10 00:00:00 |   1836

(10 rows)
```

Dans un cas comme dans l'exemple précédent, vous pouvez obtenir un contrôle supplémentaire sur le formatage de sortie en utilisantTO_CHAR.

Dans cet exemple, un nombre entier est converti en chaîne de caractères :

```
select cast(2008 as char(4));

bpchar
```

```
--------
2008
```

Dans cet exemple, une valeur DECIMAL(6,3) est convertie en valeur DECIMAL(4,1) :

```
select cast(109.652 as decimal(4,1));

numeric
---------
109.7
```

Cet exemple montre une expression plus complexe. La colonne PRICEPAID (colonne DECIMAL(8,2)) de la table SALES est convertie en une colonne DECIMAL(38,2) et les valeurs sont multipliées par 100000000000000000000 :

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


 salesid |           value
---------+----------------------------
       1 | 728000000000000000000000.00
       2 |  760000000000000000000000.00
       3 | 350000000000000000000000.00
       4 | 175000000000000000000000.00
       5 | 154000000000000000000000.00
       6 | 394000000000000000000000.00
       7 | 788000000000000000000000.00
       8 | 197000000000000000000000.00
       9 | 591000000000000000000000.00

(9 rows)
```

Fonction DECODE

La fonction DECODE est l'équivalent de la fonction ENCODE, qui est utilisée pour convertir une chaîne en format binaire à l'aide d'un codage de caractères spécifique. La fonction DECODE prend les données binaires et les reconvertit en un format de chaîne lisible en utilisant le codage de caractères spécifié.

Cette fonction est utile lorsque vous devez travailler avec des données binaires stockées dans une base de données et les présenter dans un format lisible par l'homme, ou lorsque vous devez convertir des données entre différents codages de caractères.

Syntaxe

```
decode(expr, charset)
```

Arguments

expr

Expression BINAIRE codée dans un jeu de caractères.

jeu de caractères

Expression STRING.

Encodages de jeux de caractères pris en charge (sans distinction majuscules/minuscules) : `'US-ASCII'`,`'ISO-8859-1'`, `'UTF-8''UTF-16BE'`, `'UTF-16LE'` et. `'UTF-16'`

Type de retour

La fonction DECODE renvoie une chaîne.

Exemple

L'exemple suivant contient une table appelée `messages` avec une colonne appelée `message_text` qui stocke les données des messages dans un format binaire à l'aide du codage de caractères UTF-8. La fonction DECODE reconvertit les données binaires en un format de chaîne lisible. Le résultat de cette requête est le texte lisible du message stocké dans la table des messages, avec l'ID123, converti du format binaire en chaîne à l'aide du `'utf-8'` codage.

```
SELECT decode(message_text, 'utf-8') AS message
FROM messages
WHERE message_id = 123;
```

Fonction ENCODE

La fonction ENCODE est utilisée pour convertir une chaîne en sa représentation binaire à l'aide d'un codage de caractères spécifié.

Cette fonction est utile lorsque vous devez travailler avec des données binaires ou lorsque vous devez effectuer une conversion entre différents codages de caractères. Par exemple, vous pouvez utiliser la fonction ENCODE lorsque vous stockez des données dans une base de données qui nécessite un stockage binaire ou lorsque vous devez transférer des données entre des systèmes utilisant des codages de caractères différents.

Syntaxe

```
encode(str, charset)
```

Arguments

str

Expression STRING à encoder.

jeu de caractères

Expression STRING spécifiant le codage.

Encodages de jeux de caractères pris en charge (sans distinction majuscules/minuscules) : `'US-ASCII'`,`'ISO-8859-1'`, `'UTF-8''UTF-16BE'`, `'UTF-16LE'` et. `'UTF-16'`

Type de retour

La fonction ENCODE renvoie un BINARY.

Exemple

L'exemple suivant convertit la chaîne en sa représentation binaire `'abc'` à l'aide du `'utf-8'` codage, ce qui, dans ce cas, renvoie la chaîne d'origine. Cela est dû au fait que le `'utf-8'` codage est un codage de caractères à largeur variable qui peut représenter l'ensemble du jeu de caractères ASCII (y compris les lettres `'a''b'`, et`'c'`) en utilisant un seul octet par caractère. Par conséquent, la représentation binaire de `'abc'` l'utilisation `'utf-8'` est la même que celle de la chaîne d'origine.

```
SELECT encode('abc', 'utf-8');
 abc
```

Fonction HEX

La fonction HEX convertit une valeur numérique (un entier ou un nombre à virgule flottante) en sa représentation sous forme de chaîne hexadécimale correspondante.

L'hexadécimal est un système numérique qui utilise 16 symboles distincts (0-9 et A-F) pour représenter des valeurs numériques. Il est couramment utilisé en informatique et en programmation pour représenter des données binaires dans un format plus compact et lisible par l'homme.

Syntaxe

```
hex(expr)
```

Arguments

expr

Expression BIGINT, BINARY ou STRING.

Type de retour

HEX renvoie une chaîne. La fonction renvoie la représentation hexadécimale de l'argument.

Exemple

L'exemple suivant prend la valeur entière 17 en entrée et lui applique la fonction HEX (). La sortie est11, qui est la représentation hexadécimale de la valeur d'entrée17.

```
SELECT hex(17);
 11
```

L'exemple suivant convertit la chaîne `'Spark_SQL'` en sa représentation hexadécimale.
La sortie est`537061726B2053514C`, qui est la représentation hexadécimale de la chaîne d'entrée`'Spark_SQL'`.

```
SELECT hex('Spark_SQL');
 537061726B2053514C
```

Dans cet exemple, la chaîne « Spark_SQL » est convertie comme suit :

- 'S' -> 53
- « p » -> 70
- « a » -> 61
- 'r' -> 72 '
- k' -> 6 B
- « _ » -> 20
- 'S' -> 53
- « Q » -> 51
- « L' » -> 4C

La concaténation de ces valeurs hexadécimales donne le résultat final « ». `537061726B2053514C"`

Fonction STR_TO_MAP

La fonction STR_TO_MAP est une fonction de conversion. string-to-map Il convertit une représentation sous forme de chaîne d'une carte (ou d'un dictionnaire) en une structure de données cartographique réelle.

Cette fonction est utile lorsque vous devez travailler avec des structures de données cartographiques en SQL, mais les données sont initialement stockées sous forme de chaîne. En convertissant la représentation sous forme de chaîne en une carte réelle, vous pouvez ensuite effectuer des opérations et des manipulations sur les données cartographiques.

Syntaxe

```
str_to_map(text[, pairDelim[, keyValueDelim]])
```

Arguments

texte

Expression STRING qui représente la carte.

Pair Delim

Un littéral STRING facultatif qui indique comment séparer les entrées. La valeur par défaut est une virgule ()`','`.

keyValueDelim

Un littéral STRING facultatif qui indique comment séparer chaque paire clé-valeur. La valeur par défaut est deux points (`':'`).

Type de retour

La fonction STR_TO_MAP renvoie une carte de type STRING pour les clés et les valeurs. PairDelim et PairDelim keyValueDelimsont tous deux traités comme des expressions régulières.

Exemple

L'exemple suivant prend la chaîne d'entrée et les deux arguments du séparateur, et convertit la représentation sous forme de chaîne en une structure de données cartographique réelle. Dans cet exemple spécifique, la chaîne d'entrée `'a:1,b:2,c:3'` représente une carte avec les paires clé-valeur suivantes : `'a'` est la clé et `'1'` est la valeur. `'b'`est la clé, et `'2'` c'est la valeur. `'c'`est la clé, et `'3'` c'est la valeur. Le `','` délimiteur est utilisé pour séparer les paires clé-valeur, et le `':'` délimiteur est utilisé pour séparer la clé et la valeur au sein de chaque paire. Le résultat de cette requête est : `{"a":"1","b":"2","c":"3"}` Il s'agit de la structure de données cartographiques résultante`'a'`, où les clés sont `'b''c'`, et, et les valeurs correspondantes sont `'1''2'`, et`'3'`.

```
SELECT str_to_map('a:1,b:2,c:3', ',', ':');
 {"a":"1","b":"2","c":"3"}
```

L'exemple suivant montre que la fonction STR_TO_MAP s'attend à ce que la chaîne d'entrée soit dans un format spécifique, avec les paires clé-valeur correctement délimitées. Si la chaîne d'entrée ne correspond pas au format attendu, la fonction tentera tout de même de créer une carte, mais les valeurs obtenues risquent de ne pas correspondre aux attentes.

```
SELECT str_to_map('a');
 {"a":null}
```

TO_CHAR

TO_CHAR convertit un horodatage ou une expression numérique en un format de données de chaînes de caractères.

Syntaxe

```
TO_CHAR (timestamp_expression | numeric_expression , 'format')
```

Arguments

timestamp_expression

Expression qui se traduit par une valeur de type TIMESTAMP ou TIMESTAMPTZ ou par une valeur qui peut être implicitement convertie en un horodatage.

numeric_expression

Expression qui se traduit par une valeur de type de données numérique ou par une valeur qui peut être implicitement convertie en un type numérique. Pour de plus amples informations, veuillez consulter Types numériques. TO_CHAR insère un espace à gauche de la chaîne numérique.

> **Note**
>
> TO_CHAR ne prend pas en charge les valeurs DECIMAL 128 bits.

format

Format de la nouvelle valeur. Pour connaître les formats valides, consultez Chaînes de format datetime et Chaînes de format numériques.

Type de retour

VARCHAR

Exemples

L'exemple suivant convertit un horodatage en une valeur contenant la date et l'heure dans un format comprenant le nom du mois rempli jusqu'à neuf caractères, le nom du jour de la semaine et le numéro du jour du mois.

```
select to_char(timestamp '2009-12-31 23:15:59', 'MONTH-DY-DD-YYYY HH12:MIPM');
to_char
-------------------------
```

```
DECEMBER -THU-31-2009 11:15PM
```

L'exemple suivant convertit un horodatage en une valeur avec le numéro du jour de l'année.

```
select to_char(timestamp '2009-12-31 23:15:59', 'DDD');

to_char
-------------------------
365
```

L'exemple suivant convertit un horodatage en un numéro de jour de la semaine ISO.

```
select to_char(timestamp '2022-05-16 23:15:59', 'ID');

to_char
------------------------
1
```

L'exemple suivant extrait le nom du mois d'une date.

```
select to_char(date '2009-12-31', 'MONTH');

to_char
-------------------------
DECEMBER
```

L'exemple suivant convertit chaque valeur STARTTIME de la table EVENT en une chaîne composée d'heures, de minutes et de secondes.

```
select to_char(starttime, 'HH12:MI:SS')
from event where eventid between 1 and 5
order by eventid;

to_char
----------
02:30:00
08:00:00
02:30:00
02:30:00
07:00:00
(5 rows)
```

L'exemple suivant convertit une valeur d'horodatage complète en un format différent.

```
select starttime, to_char(starttime, 'MON-DD-YYYY HH12:MIPM')
from event where eventid=1;

      starttime      |       to_char
---------------------+---------------------
 2008-01-25 14:30:00 | JAN-25-2008 02:30PM
(1 row)
```

L'exemple suivant convertit un littéral d'horodatage en une chaîne de caractères.

```
select to_char(timestamp '2009-12-31 23:15:59','HH24:MI:SS');
to_char
----------
23:15:59
(1 row)
```

L'exemple suivant convertit un nombre en chaîne de caractères avec le signe moins à la fin.

```
select to_char(-125.8, '999D99S');
to_char
---------
125.80-
(1 row)
```

L'exemple suivant convertit un nombre en chaîne de caractères avec le symbole de la devise.

```
select to_char(-125.88, '$S999D99');
to_char
---------
$-125.88
(1 row)
```

L'exemple suivant convertit un nombre en une chaîne de caractères.

```
select to_char(-125.88, '$999D99PR');
to_char
---------
$<125.88>
```

```
(1 row)
```

L'exemple suivant convertit un nombre en chaîne numérale romaine.

```
select to_char(125, 'RN');
to_char
---------
CXXV
(1 row)
```

L'exemple suivant affiche le jour de la semaine.

```
SELECT to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS');
               to_char
-----------------------
Wednesday, 31 09:34:26
```

L'exemple suivant affiche le suffixe ordinal d'un nombre.

```
SELECT to_char(482, '999th');
               to_char
-----------------------
 482nd
```

L'exemple suivant soustraie la commission du prix d'achat de la table des ventes. La différence est ensuite arrondie et convertie en chiffres romains, comme indiqué dans la `to_char` colonne :

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'rn') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;

 salesid | pricepaid | commission | difference |     to_char
---------+-----------+------------+------------+-----------------
       1 |    728.00 |     109.20 |     618.80 |           dcxix
       2 |     76.00 |      11.40 |      64.60 |             lxv
       3 |    350.00 |      52.50 |     297.50 |        ccxcviii
       4 |    175.00 |      26.25 |     148.75 |           cxlix
       5 |    154.00 |      23.10 |     130.90 |           cxxxi
       6 |    394.00 |      59.10 |     334.90 |         cccxxxv
       7 |    788.00 |     118.20 |     669.80 |           dclxx
       8 |    197.00 |      29.55 |     167.45 |          clxvii
```

```
       9 |    591.00 |      88.65 |     502.35 |             dii
      10 |     65.00 |       9.75 |      55.25 |              lv
(10 rows)
```

L'exemple suivant ajoute le symbole monétaire aux valeurs de différence indiquées dans la `to_char` colonne :

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'l99999D99') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;

salesid | pricepaid | commission | difference |  to_char
--------+-----------+------------+------------+------------
      1 |    728.00 |     109.20 |     618.80 | $   618.80
      2 |     76.00 |      11.40 |      64.60 | $    64.60
      3 |    350.00 |      52.50 |     297.50 | $   297.50
      4 |    175.00 |      26.25 |     148.75 | $   148.75
      5 |    154.00 |      23.10 |     130.90 | $   130.90
      6 |    394.00 |      59.10 |     334.90 | $   334.90
      7 |    788.00 |     118.20 |     669.80 | $   669.80
      8 |    197.00 |      29.55 |     167.45 | $   167.45
      9 |    591.00 |      88.65 |     502.35 | $   502.35
     10 |     65.00 |       9.75 |      55.25 | $    55.25
(10 rows)
```

L’exemple suivant répertorie le siècle au cours duquel chaque vente a été effectuée.

```
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)
```

L'exemple suivant convertit chaque valeur STARTTIME de la table EVENT en une chaîne qui se compose d'heures, de minutes, de secondes et d'un fuseau horaire.

```
select to_char(starttime, 'HH12:MI:SS TZ')
from event where eventid between 1 and 5
order by eventid;

to_char
----------
02:30:00 UTC
08:00:00 UTC
02:30:00 UTC
02:30:00 UTC
07:00:00 UTC
(5 rows)

(10 rows)
```

L'exemple suivant illustre la mise en forme des secondes, millisecondes et microsecondes.

```
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
```

Fonction TO_DATE

TO_DATE convertit une date représentée par une chaîne de caractères en un type de données DATE.

Syntaxe

```
TO_DATE (date_str)
```

```
TO_DATE (date_str, format)
```

Arguments

date_str

Chaîne de date ou type de données pouvant être converti en chaîne de date.

format

Chaîne littérale qui correspond aux modèles de date/heure de Spark. Pour les modèles de date/heure valides, voir Modèles de date/heure pour le formatage et l'analyse syntaxique.

Type de retour

TO_DATE renvoie une DATE, selon la valeur de format.

Si la conversion au format échoue, une erreur est renvoyée.

Exemples

L'instruction SQL suivante convertit la date `02 Oct 2001` en type de données de date.

```
select to_date('02 Oct 2001', 'dd MMM yyyy');

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

L'instruction SQL suivante convertit la chaîne `20010631` en date.

```
select to_date('20010631', 'yyyyMMdd');
```

L'instruction SQL suivante convertit la chaîne `20010631` en date :

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

Le résultat est une valeur nulle car le mois de juin ne compte que 30 jours.

```
to_date
------------
NULL
```

TO_NUMBER

TO_NUMBER convertit une chaîne en une valeur numérique (décimale).

Syntaxe

```
to_number(string, format)
```

Arguments

string

Chaîne à convertir. Le format doit être une valeur littérale.

format

Le deuxième argument est une chaîne de format qui indique comment la chaîne de caractères doit être analysée afin de créer la valeur numérique. Par exemple, le format `'99D999'` spécifie que la chaîne à convertir se compose de cinq chiffres, avec la virgule à la troisième position. Par exemple, `to_number('12.345','99D999')` renvoie `12.345` comme une valeur numérique. Pour obtenir la liste des formats valides, consultez Chaînes de format numériques.

Type de retour

TO_NUMBER renvoie un nombre DECIMAL.

Si la conversion au format échoue, une erreur est renvoyée.

Exemples

L'exemple suivant convertit la chaîne `12,454.8-` en un nombre :

```
select to_number('12,454.8-', '99G999D9S');

to_number
-----------
-12454.8
```

L'exemple suivant convertit la chaîne `$ 12,454.88` en un nombre :

```
select to_number('$ 12,454.88', 'L 99G999D99');
```

```
to_number
-----------
12454.88
```

L'exemple suivant convertit la chaîne `$ 2,012,454.88` en un nombre :

```
select to_number('$ 2,012,454.88', 'L 9,999,999.99');

to_number
-----------
2012454.88
```

UNBASE64 fonction

La UNBASE64 fonction convertit un argument d'une chaîne de base 64 en binaire.

Le codage Base64 est couramment utilisé pour représenter des données binaires (telles que des images, des fichiers ou des informations cryptées) dans un format textuel sûr pour la transmission sur différents canaux de communication (tels que le courrier électronique, les paramètres d'URL ou le stockage de base de données).

La UNBASE64 fonction permet d'inverser ce processus et de récupérer les données binaires d'origine. Ce type de fonctionnalité peut être utile dans les scénarios où vous devez travailler avec des données codées au format Base64, par exemple lors de l'intégration à des systèmes externes ou APIs lorsque vous utilisez Base64 comme mécanisme de transfert de données.

Syntaxe

```
unbase64(expr)
```

Arguments

expr

Expression STRING au format base64.

Type de retour

`BINARY`

Exemple

Dans l'exemple suivant, la chaîne codée en Base64 `'U3BhcmsgU1FM'` est reconvertie en chaîne d'origine. `'Spark SQL'`

```
SELECT unbase64('U3BhcmsgU1FM');
 Spark SQL
```

Fonction UNHEX

La fonction UNHEX reconvertit une chaîne hexadécimale en sa représentation sous forme de chaîne d'origine.

Cette fonction peut être utile dans les scénarios où vous devez travailler avec des données stockées ou transmises dans un format hexadécimal, et vous devez restaurer la représentation de chaîne d'origine pour un traitement ou un affichage ultérieurs.

La fonction UNHEX est le pendant de la fonction HEX.

Syntaxe

```
unhex(expr)
```

Arguments

expr

Expression de type STRING composée de caractères hexadécimaux.

Type de retour

UNHEX renvoie un BINARY.

Si la longueur de expr est impaire, le premier caractère est supprimé et le résultat est complété par un octet nul. Si expr contient des caractères non hexadécimaux, le résultat est NULL.

Exemple

L'exemple suivant reconvertit une chaîne hexadécimale en sa représentation sous forme de chaîne d'origine en utilisant conjointement les fonctions UNHEX () et DECODE (). La première partie de la requête utilise la fonction UNHEX () pour convertir la chaîne hexadécimale '537061726B2053514C' en sa représentation binaire. La deuxième partie de la requête utilise la fonction DECODE () pour

reconvertir les données binaires obtenues par la fonction UNHEX () en chaîne, en utilisant le codage de caractères « UTF-8 ». Le résultat de la requête est la chaîne originale 'Spark_SQL' qui a été convertie en hexadécimal puis redevenue une chaîne.

```
SELECT decode(unhex('537061726B2053514C'), 'UTF-8');
 Spark SQL
```

Chaînes de format datetime

Vous pouvez utiliser des modèles de date/heure dans les scénarios courants suivants :

- Lorsque vous travaillez avec des sources de données CSV et JSON pour analyser et formater du contenu date/heure
- Lors de la conversion entre des types de chaînes et des types de date ou d'horodatage à l'aide de fonctions telles que :
  - horodatage unix
  - date_format
  - to_unix_timestamp
  - from_unixtime
  - to_date
  - to_timestamp
  - from_utc_timestamp
  - to_utc_timestamp

Utilisez les lettres types du tableau suivant pour l'analyse et le formatage de la date et de l'horodatage.

Partie de date ou d’horodatage	Signification	Exemples
a	Matin ou PM du jour, présenté comme étant du matin au soir	PM
D	Jour de l'année, présenté sous forme de numéro à 3 chiffres	189

Partie de date ou d'horodatage	Signification	Exemples
d	Jour du mois, présenté sous forme de nombre à 2 chiffres	28
E	Jour de la semaine, présenté sous forme de texte	mardi Mardi
F	Jour de la semaine aligné dans le mois, présenté sous forme de nombre à 1 chiffre	3
G	Indicateur d'ère, présenté sous forme de texte	AD Anno Domini
h	Heure du matin ou du soir, présentée sous forme de nombre à 2 chiffres	12
H	Heure du jour, présentée sous la forme d'un nombre à deux chiffres compris entre 0 et 23	0
k	Heure du jour, présentée sous la forme d'un nombre à 2 chiffres compris entre 1 et 24	1
K	Heure du matin ou du soir, présentée sous la forme d'un nombre à 2 chiffres compris entre 0 et 11	0
m	Minute de l'heure, présentée sous forme de nombre à 2 chiffres	30

Partie de date ou d'horodatage	Signification	Exemples
M/L	Mois de l'année, présenté sous forme de mois	7 07 Juil juillet
O	Décalage de zone localisé par rapport à l'UTC	GMT+8 GMT+ 8:00 UTC- 08:00
Q/Q	Trimestre de l'année, présenté sous forme de chiffre (1 à 4) ou de texte	3 03 Q3 3e trimestre
s	Seconde de minute, présentée sous la forme d'un nombre à deux chiffres	55
S	Fraction de seconde, présentée sous forme de fraction	978
V	Identifiant de fuseau horaire, présenté sous forme d'identifiant de zone	America/Los_Angeles Z 08h30

Partie de date ou d'horodatage	Signification	Exemples
x	Décalage de zone par rapport à UTC (Offset-X)	+0000 -08 -0830 - 08h30 -083015 - 08 H 30 min 15
X	Décalage de zone par rapport à l'UTC ; où Z est égal à zéro	Z -08 -0830 - 08h30 -083015 - 08 H 30 min 15
y	Année, présentée comme une année	2020 20
z	Nom du fuseau horaire, présenté sous forme de texte	Heure normale du Pacifique PST
Z	Décalage de zone par rapport à UTC (Offset-Z)	+0000 -0800 - 08h00

Partie de date ou d'horodatage	Signification	Exemples
'	Échappement pour le texte, présenté sous forme de délimiteur	N/A
''	Citation unique, présentée sous forme littérale	'
[	Début de section facultatif	N/A
]	Fin de section facultative	N/A

Le nombre de lettres du modèle détermine le type de format :

Format du texte

- Utilisez 1 à 3 lettres pour la forme abrégée (par exemple, « Mon » pour le lundi)
- Utilisez exactement 4 lettres pour le formulaire complet (par exemple, « lundi »)
- N'utilisez pas 5 lettres ou plus, cela provoquerait une erreur

Format numérique (n)

- La valeur n représente le nombre maximum de lettres autorisées
- Pour les modèles à lettre unique :
  - La sortie utilise un minimum de chiffres sans remplissage
- Pour plusieurs modèles de lettres :
  - La sortie est complétée par des zéros pour correspondre à la largeur du nombre de lettres
- Lors de l'analyse, l'entrée doit contenir le nombre exact de chiffres

Format numérique/texte

- Pour 3 lettres ou plus, suivez les règles du format de texte
- Pour réduire le nombre de lettres, suivez les règles relatives au format numérique

Format de fraction

- Utilisez 1 à 9 caractères « S » (par exemple, SSSSSS)
- Pour l'analyse syntaxique :
  - Acceptez les fractions comprises entre 1 et le nombre de caractères S
- Pour le formatage :
  - Bloc avec des zéros correspondant au nombre de caractères S
- Supporte jusqu'à 6 chiffres pour une précision de l'ordre de la microseconde
- Peut analyser des nanosecondes mais tronque des chiffres supplémentaires

Format de l'année

- Le nombre de lettres définit la largeur de champ minimale pour le rembourrage
- Pour deux lettres :
  - Imprime les deux derniers chiffres
  - Analyse les années entre 2000 et 2099
- Pour moins de quatre lettres (sauf deux) :
  - Affiche le signe uniquement pour les années négatives
- N'utilisez pas 7 lettres ou plus, cela provoquerait une erreur

Format du mois

- Utilisez « M » pour le formulaire standard ou « L » pour le formulaire autonome
- « M » ou « L » simple :
  - Affiche les numéros des mois 1 à 12 sans rembourrage
- « MM » ou « LL » :
  - Affiche les numéros des mois 01-12 avec rembourrage
- « MAMAN » :
  - Affiche le nom abrégé du mois sous forme standard
  - Doit faire partie d'un schéma de date complet
- « TOUT LE MONDE » :
  - Affiche le nom abrégé du mois sous forme autonome

  - À utiliser pour le formatage mensuel uniquement
- « MMMM » :
  - Affiche le nom complet du mois sous forme standard
  - À utiliser pour les dates et les horodatages
- « TOUT LE MONDE » :
  - Affiche le nom complet du mois sous forme autonome
  - À utiliser pour le formatage mensuel uniquement

Formats de fuseau horaire

- Du matin au soir : utilisez une seule lettre
- ID de zone (V) : utilisez 2 lettres uniquement
- Noms de zone (z) :
  - 1 à 3 lettres : affiche le nom court
  - 4 lettres : affiche le nom complet
  - N'utilisez pas 5 lettres ou plus

Formats de décalage

- X et x :
  - 1 lettre : affiche l'heure (+01) ou l'heure/minute (+0130)
  - 2 lettres : affiche les heures et les minutes sans deux points (+0130)
  - 3 lettres : affiche les heures et les minutes avec deux points (+ 01:30)
  - 4 lettres : Afficher hour-minute-second sans deux points (+013015)
  - 5 lettres : Afficher hour-minute-second avec deux points (+ 01:30:15)
  - X utilise « Z » pour un décalage nul
  - x utilise « +00 », « +0000 » ou « + 00:00 » pour un décalage nul
- O:
  - 1 lettre : affiche la forme abrégée (GMT+8)
  - 4 lettres : affiche le formulaire complet (GMT+ 08:00)
- Z :
  - 1 à 3 lettres : affiche les heures et les minutes sans deux points (+0130)

- 4 lettres : affiche le formulaire localisé complet
- 5 lettres : Indique hour-minute-second avec deux points

Sections facultatives

- Utilisez des crochets [] pour marquer le contenu facultatif
- Vous pouvez imbriquer des sections facultatives
- Toutes les données valides apparaissent dans la sortie
- La saisie peut omettre des sections facultatives entières

> **Note**
>
> Les symboles « E », « F », « q » et « Q » ne fonctionnent que pour le formatage de la date et de l'heure (comme date_format). Ne les utilisez pas pour l'analyse de la date et de l'heure (comme to_timestamp).

Chaînes de format numériques

Les chaînes de format numérique suivantes s'appliquent à des fonctions telles que TO_NUMBER et TO_CHAR.

- Pour des exemples de formatage de chaînes sous forme de nombres, consultez TO_NUMBER.
- Pour des exemples de formatage de nombres sous forme de chaînes, consultez TO_CHAR.

Format	Description
9	Valeur numérique avec le nombre spécifié de chiffres.
0	Valeur numérique commençant par des zéros.
. (point), D	Virgule.
, (comma)	Séparateur de milliers.

Format	Description
CC	Code de siècle. Par exemple, le 21e siècle a commencé le 2001-01-01 (pris en charge pour TO_CHAR uniquement).
FM	Mode de remplissage. Permet de supprimer les zéros et les vides de remplissage.
PR	Valeur négative entre crochets.
S	Signe fixé à un nombre.
L	Symbole de devise à la position spécifiée.
G	Séparateur de groupe.
MI	Signe moins à la position spécifiée pour les numéros inférieurs à 0.
PL	Signe plus à la position spécifiée pour les numéros supérieurs à 0.
SG	Signe plus ou moins à la position spécifiée.
RN	Chiffre romains compris entre 1 et 3 999 (pris en charge pour TO_CHAR uniquement).
TH ou th	Suffixe de nombre ordinal. Ne convertit pas les nombres ou les valeurs fractionnaires inférieurs à zéro.

Fonctions de date et d’heure

Les fonctions de date et d'heure vous permettent d'effectuer un large éventail d'opérations sur les données de date et d'heure, telles que l'extraction de parties d'une date, le calcul des dates, le formatage des dates et des heures et l'utilisation de la date et de l'heure actuelles. Ces fonctions sont essentielles pour les tâches telles que l'analyse des données, le reporting et la manipulation de données impliquant des données temporelles.

AWS Clean Rooms prend en charge les fonctions de date et d'heure suivantes :

Rubriques

Fonction ADD_MONTHS

ADD_MONTHS ajoute le nombre de mois spécifié à une date, à une valeur d’horodatage ou à une expression. La fonction DATE_ADD fournit une fonctionnalité similaire.

Syntaxe

```
ADD_MONTHS( {date | timestamp}, integer)
```

Arguments

date | timestamp

Colonne date ou timestamp ou expression qui convertit implicitement en un horodatage ou une date. Si la date est le dernier jour du mois, ou si le mois résultant est plus court, la fonction renvoie le dernier jour du mois dans le résultat. Pour les autres dates, le résultat contient le même nombre de jours que l'expression de date.

integer

Nombre entier positif ou négatif. Utilisez un nombre négatif pour soustraire des mois à partir de dates.

Type de retour

TIMESTAMP

Exemple

La requête suivante utilise la fonction ADD_MONTHS à l'intérieur d'une fonction TRUNC. La fonction TRUNC supprime l'heure du jour des résultats de ADD_MONTHS. La fonction ADD_MONTHS ajoute 12 mois à chaque valeur de la colonne CALDATE.

```
select distinct trunc(add_months(caldate, 12)) as calplus12,
trunc(caldate) as cal
from date
order by 1 asc;

 calplus12  |    cal
------------+------------
 2009-01-01 | 2008-01-01
 2009-01-02 | 2008-01-02
 2009-01-03 | 2008-01-03
...
(365 rows)
```

Les exemples suivants illustrent le comportement lorsque la fonction ADD_MONTHS opère sur des dates comportant des mois avec un nombre de jours différent.

```
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)
```

Fonction CONVERT_TIMEZONE

CONVERT_TIMEZONE convertit un horodatage d'un fuseau horaire à un autre. La fonction s'ajuste automatiquement pour l'heure d'été.

Syntaxe

```
CONVERT_TIMEZONE ( ['source_timezone',] 'target_timezone', 'timestamp')
```

Arguments

source_timezone

(Facultatif) Fuseau horaire de l'horodatage actuel. La valeur par défaut est UTC.

target_timezone

Fuseau horaire du nouvel horodatage.

timestamp

Colonne timestamp ou expression qui convertit implicitement en un horodatage.

Type de retour

TIMESTAMP

Exemples

L'exemple suivant convertit la valeur d'horodatage du fuseau horaire UTC par défaut en HNP.

```
select convert_timezone('PST', '2008-08-21 07:23:54');

 convert_timezone
------------------------
2008-08-20 23:23:54
```

L'exemple suivant convertit la valeur d'horodatage dans la colonne LISTTIME du fuseau horaire UTC par défaut en HNP. Même si l'horodatage est à l'heure d'été, il est converti en heure normale, car le fuseau horaire cible est spécifié comme abréviation (PST).

```
select listtime, convert_timezone('PST', listtime) from listing
where listid = 16;

     listtime       |   convert_timezone
--------------------+-------------------
2008-08-24 09:36:12     2008-08-24 01:36:12
```

L'exemple suivant convertit une colonne d'horodatage LISTTIME du fuseau horaire UTC par défaut en fuseau horaire. US/Pacific Le fuseau horaire cible utilise un nom de fuseau horaire, et l'horodatage se situe pendant la période l'heure d'été, donc la fonction renvoie l'heure.

```
select listtime, convert_timezone('US/Pacific', listtime) from listing
where listid = 16;

     listtime       |   convert_timezone
--------------------+--------------------
2008-08-24 09:36:12 | 2008-08-24 02:36:12
```

L'exemple suivant convertit une chaîne d'horodatage de l'EST à PST :

```
select convert_timezone('EST', 'PST', '20080305 12:25:29');

 convert_timezone
```

```
-------------------
2008-03-05 09:25:29
```

L'exemple suivant convertit un horodatage à l'heure normale de l'est des États-Unis, car le fuseau horaire cible utilise un nom de fuseau horaire (Amérique/New_York) et que l'horodatage est à l'heure normale.

```
select convert_timezone('America/New_York', '2013-02-01 08:00:00');

 convert_timezone
---------------------
2013-02-01 03:00:00
(1 row)
```

L'exemple suivant convertit un horodatage à l'heure d'été de l'est des États-Unis, car le fuseau horaire cible utilise un nom de fuseau horaire (Amérique/New_York) et que l'horodatage est à l'heure d'été.

```
select convert_timezone('America/New_York', '2013-06-01 08:00:00');

 convert_timezone
---------------------
2013-06-01 04:00:00
(1 row)
```

L'exemple suivant illustre l'utilisation des décalages.

```
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)
```

Fonction CURRENT_DATE

CURRENT_DATE renvoie une date dans le fuseau horaire de la session en cours (UTC par défaut) au format par défaut :. YYYY-MM-DD

> **Note**
>
> CURRENT_DATE renvoie la date de début de la transaction en cours, pas le début de l'instruction en cours. Imaginez un scénario où vous démarrez une transaction contenant plusieurs instructions le 10/01/08 à 23h59, et où l'instruction contenant CURRENT_DATE s'exécute le 10/02/08 à 00h00. CURRENT_DATE renvoie `10/01/08`, et non `10/02/08`.

Syntaxe

```
CURRENT_DATE
```

Type de retour

DATE

Exemple

L'exemple suivant renvoie la date actuelle (dans Région AWS laquelle la fonction s'exécute).

```
select current_date;

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

Fonction CURRENT_TIMESTAMP

CURRENT_TIMESTAMP renvoie la date et l'heure actuelles, y compris la date, l'heure et (éventuellement) les millisecondes ou microsecondes.

Cette fonction est utile lorsque vous devez obtenir la date et l'heure actuelles, par exemple pour enregistrer l'horodatage d'un événement, pour effectuer des calculs basés sur le temps ou pour remplir des colonnes. date/time

Syntaxe

```
current_timestamp()
```

Type de retour

La fonction CURRENT_TIMESTAMP renvoie une DATE.

Exemple

L'exemple suivant renvoie la date et l'heure actuelles au moment où la requête est exécutée, soit le 25 avril 2020 à 15:49:11.914 (15:49:11.914 PM).

```
SELECT current_timestamp();
 2020-04-25 15:49:11.914
```

L'exemple suivant extrait la date et l'heure actuelles pour chaque ligne du `squirrels` tableau.

```
SELECT current_timestamp() FROM squirrels
```

Fonction DATE_ADD

Renvoie la date qui est postérieure de num_jours à la date de début.

Syntaxe

```
date_add(start_date, num_days)
```

Arguments

date_de début

La valeur de la date de début.

num_days

Le nombre de jours à ajouter (entier). Un nombre positif ajoute des jours, un nombre négatif soustrait des jours.

Type de retour

DATE

Exemples

L'exemple suivant ajoute un jour à une date :

```
SELECT date_add('2016-07-30', 1);

Result:
2016-07-31
```

L'exemple suivant ajoute plusieurs jours.

```
SELECT date_add('2016-07-30', 5);

Result:
2016-08-04
```

Notes d’utilisation

Cette documentation concerne la fonction DATE_ADD de Spark SQL, qui fournit une interface plus simple pour ajouter des jours aux dates par rapport à d'autres variantes SQL. Pour ajouter d'autres intervalles tels que des mois ou des années, différentes fonctions peuvent être nécessaires.

Fonction DATE_DIFF

DATE_DIFF renvoie la différence entre les parties datées de deux expressions de date ou d'heure.

Syntaxe

```
date_diff(endDate, startDate)
```

Arguments

endDate

Expression DATE.

startDate

Expression DATE.

Type de retour

BIGINT

Exemples avec une colonne DATE

L'exemple suivant met en évidence la différence, en nombre de semaines, entre deux valeurs de date littérales.

```
select date_diff(week,'2009-01-01','2009-12-31') as numweeks;

numweeks
----------
52
(1 row)
```

L'exemple suivant permet de trouver la différence, en heures, entre deux valeurs littérales de date. Si vous n'indiquez pas la valeur temporelle d'une date, celle-ci est fixée par défaut à 00:00:00.

```
select date_diff(hour, '2023-01-01', '2023-01-03 05:04:03');

date_diff
----------
53
(1 row)
```

L'exemple suivant permet de trouver la différence, en jours, entre deux valeurs TIMESTAMETZ littérales.

```
Select date_diff(days, 'Jun 1,2008  09:59:59 EST', 'Jul 4,2008  09:59:59 EST')

date_diff
----------
33
```

L'exemple suivant permet de trouver la différence, en jours, entre deux dates figurant sur la même ligne d'une table.

```
select * from date_table;

start_date |   end_date
-----------+-----------
2009-01-01 | 2009-03-23
2023-01-04 | 2024-05-04
(2 rows)
```

```
select date_diff(day, start_date, end_date) as duration from date_table;

duration
---------
      81
     486
(2 rows)
```

L'exemple suivant met en évidence la différence, dans le nombre de trimestres, entre une valeur littérale dans le passé et la date du jour. Cet exemple suppose que la date du jour est le 5 juin 2008. Vous pouvez nommer les parties de date intégralement ou les abréger. Le nom de colonne par défaut de la fonction DATE_DIFF est DATE_DIFF.

```
select date_diff(qtr, '1998-07-01', current_date);

date_diff
-----------
40
(1 row)
```

L'exemple suivant joint les tables SALES et LISTING pour calculer combien de jours après leur mise en vente des billets ont été vendus pour les listes 1000 à 1005. L'attente la plus longue pour les ventes de ces listes a été 15 jours, et la plus courte a été de moins d'une journée (0 jour).

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

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

Cet exemple calcule la moyenne du nombre d'heures que les vendeurs ont attendu pour toutes les ventes de billets.

```
select avg(date_diff(hours, listtime, saletime)) as avgwait
from sales, listing
where sales.listid = listing.listid;

avgwait
---------
465
(1 row)
```

Exemples avec une colonne TIME

L'exemple de table TIME_TEST suivant comporte une colonne TIME_VAL (type TIME) avec trois valeurs insérées.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

L'exemple suivant montre comment trouver la différence en nombre d'heures entre la colonne TIME_VAL et une valeur de temps littérale.

```
select date_diff(hour, time_val, time '15:24:45') from time_test;

 date_diff
-----------
        -5
        15
        15
```

L'exemple suivant montre comment trouver la différence en nombre de minutes entre deux valeurs de temps littérales.

```
select date_diff(minute, time '20:00:00', time '21:00:00') as nummins;

nummins
```

```
----------
60
```

Exemples avec une colonne TIMETZ

L'exemple de table TIMETZ_TEST suivant comporte une colonne TIMETZ_VAL (type TIMETZ) avec trois valeurs insérées.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

L’exemple suivant montre comment trouver la différence en nombre d’heures entre une valeur TIMETZ littérale et une valeur timez_val.

```
select date_diff(hours, timetz '20:00:00 PST', timetz_val) as numhours from
 timetz_test;

numhours
----------
0
-4
1
```

L’exemple suivant montre comment trouver la différence en nombre d’heures entre deux valeurs TIMETZ littérales.

```
select date_diff(hours, timetz '20:00:00 PST', timetz '00:58:00 EST') as numhours;

numhours
----------
1
```

Fonction DATE_PART

DATE_PART extrait des valeurs date part d’une expression. DATE_PART est un synonyme de la fonction PGDATE_PART.

Syntaxe

```
datepart(field, source)
```

Arguments

champ

La partie de la source qui doit être extraite et les valeurs de chaîne prises en charge sont les mêmes que les champs de la fonction équivalente EXTRACT.

source

Colonne DATE ou INTERVAL à partir de laquelle le champ doit être extrait.

Type de retour

Si le champ est « DEUXIÈME », une décimale (8, 6). Dans tous les autres cas, un INTEGER.

Exemple

L'exemple suivant extrait le jour de l'année (DOY) à partir d'une valeur de date. La sortie indique que le jour de l'année pour la date « 2019-08-12 » est. 224 Cela signifie que le 12 août 2019 est le 224e jour de l'année 2019.

```
SELECT datepart('doy', DATE'2019-08-12');
 224
```

Fonction DATE_TRUNC

La fonction DATE_TRUNC tronque une expression d'horodatage ou littérale en fonction de la partie de date que vous spécifiez, telle que l'heure, le jour ou le mois.

Syntaxe

```
date_trunc(format, datetime)
```

Arguments

format

Format représentant l'unité à tronquer. Les formats valides sont les suivants :

- « ANNÉE », « YYYY », « YY » - tronqués à la première date de l'année où le ts tombe, la partie temporelle sera nulle
- « TRIMESTRE » : tronquez à la première date du trimestre où le ts tombe, la partie temporelle sera nulle
- « MONTH », « MM », « MON » - tronquez à la première date du mois où le ts tombe, la partie temporelle sera nulle
- « SEMAINE » : tronquez au lundi de la semaine où le ts tombe, le temps sera nul
- « JOUR », « DD » - mettez à zéro la partie horaire
- « HEURE » : zéro la minute et la seconde avec une fraction
- « MINUTE » : zéro la seconde avec une fraction
- « DEUXIÈME » - met à zéro la deuxième fraction
- « MILLISECOND » - mettez à zéro les microsecondes
- « MICROSECONDE » - tout reste

ts

Une valeur date/heure

Type de retour

Renvoie l'horodatage ts tronqué à l'unité spécifiée par le modèle de format

Exemples

L'exemple suivant tronque une valeur de date au début de l'année. Le résultat indique que la date « 2015-03-05" a été tronquée en « 2015-01-01", ce qui correspond au début de l'année 2015.

```
SELECT date_trunc('YEAR', '2015-03-05');

 date_trunc
-----------
2015-01-01
```

Fonction DAY

La fonction DAY renvoie le jour du mois correspondant à la date/à l'horodatage.

Les fonctions d'extraction de date sont utiles lorsque vous devez travailler avec des composants spécifiques d'une date ou d'un horodatage, par exemple lorsque vous effectuez des calculs basés sur la date, que vous filtrez des données ou que vous formatez des valeurs de date.

Syntaxe

```
day(date)
```

Arguments

date

Expression DATE ou TIMESTAMP.

Renvoie

La fonction DAY renvoie un INTEGER.

Exemples

L'exemple suivant extrait le jour du mois (`30`) de la date saisie`'2009-07-30'`.

```
SELECT day('2009-07-30');
 30
```

L'exemple suivant extrait le jour du mois de la `birthday` colonne du `squirrels` tableau et renvoie les résultats sous forme de sortie de l'instruction SELECT. Le résultat de cette requête sera une liste de valeurs de jour, une pour chaque ligne du `squirrels` tableau, représentant le jour du mois pour l'anniversaire de chaque écureuil.

```
SELECT day(birthday) FROM squirrels
```

Fonction DAYOFMONTH

La fonction DAYOFMONTH renvoie le jour du mois du date/timestamp (une valeur comprise entre 1 et 31, selon le mois et l'année).

La fonction DAYOFMONTH est similaire à la fonction DAY, mais leur nom et leur comportement sont légèrement différents. La fonction DAY est plus couramment utilisée, mais la fonction DAYOFMONTH

peut être utilisée comme alternative. Ce type de requête peut être utile lorsque vous devez effectuer une analyse basée sur la date ou un filtrage sur une table contenant des données de date ou d'horodatage, par exemple pour extraire des composants spécifiques d'une date à des fins de traitement ou de génération de rapports ultérieurs.

Syntaxe

```
dayofmonth(date)
```

Arguments

date

Expression DATE ou TIMESTAMP.

Renvoie

La fonction DAYOFMONTH renvoie un INTEGER.

Exemple

L'exemple suivant extrait le jour du mois (`30`) de la date saisie `'2009-07-30'`.

```
SELECT dayofmonth('2009-07-30');
 30
```

L'exemple suivant applique la fonction DAYOFMONTH à la `birthday` colonne du `squirrels` tableau. Pour chaque ligne du `squirrels` tableau, le jour du mois indiqué dans la `birthday` colonne sera extrait et renvoyé en sortie de l'instruction SELECT. Le résultat de cette requête sera une liste de valeurs de jour, une pour chaque ligne du `squirrels` tableau, représentant le jour du mois pour l'anniversaire de chaque écureuil.

```
SELECT dayofmonth(birthday) FROM squirrels
```

Fonction DAYOFWEEK

La fonction DAYOFWEEK saisit une date ou un horodatage et renvoie le jour de la semaine sous forme de chiffre (1 pour le dimanche, 2 pour le lundi,..., 7 pour le samedi).

Cette fonction d'extraction de date est utile lorsque vous devez travailler avec des composants spécifiques d'une date ou d'un horodatage, par exemple lorsque vous effectuez des calculs basés sur la date, que vous filtrez des données ou que vous formatez des valeurs de date.

Syntaxe

```
dayofweek(date)
```

Arguments

date

Expression DATE ou TIMESTAMP.

Renvoie

La fonction DAYOFWEEK renvoie un INTEGER où

1 = dimanche

2 = lundi

3 = mardi

4 = mercredi

5 = jeudi

6 = Vendredi

7 = samedi

Exemples

L'exemple suivant extrait le jour de la semaine à partir de cette date, qui est 5 (représentant le jeudi).

```
SELECT dayofweek('2009-07-30');
 5
```

L'exemple suivant extrait le jour de la semaine de la `birthday` colonne du `squirrels` tableau et renvoie les résultats sous forme de sortie de l'instruction SELECT. Le résultat de cette requête sera une liste de valeurs du jour de la semaine, une pour chaque ligne du `squirrels` tableau, représentant le jour de la semaine pour l'anniversaire de chaque écureuil.

```
SELECT dayofweek(birthday) FROM squirrels
```

Fonction DAYOFYEAR

La fonction DAYOFYEAR est une fonction d'extraction de date qui prend une date ou un horodatage comme entrée et renvoie le jour de l'année (une valeur comprise entre 1 et 366, selon l'année et selon qu'il s'agit d'une année bissextile).

Cette fonction est utile lorsque vous devez travailler avec des composants spécifiques d'une date ou d'un horodatage, par exemple lorsque vous effectuez des calculs basés sur la date, lorsque vous filtrez des données ou que vous formatez des valeurs de date.

Syntaxe

```
dayofyear(date)
```

Arguments

date

Expression DATE ou TIMESTAMP.

Renvoie

La fonction DAYOFYEAR renvoie un INTEGER (compris entre 1 et 366, selon l'année et selon qu'il s'agit d'une année bissextile).

Exemples

L'exemple suivant extrait le jour de l'année (`100`) de la date d'entrée`'2016-04-09'`.

```
SELECT dayofyear('2016-04-09');
 100
```

L'exemple suivant extrait le jour de l'année de la `birthday` colonne du `squirrels` tableau et renvoie les résultats sous forme de sortie de l'instruction SELECT.

```
SELECT dayofyear(birthday) FROM squirrels
```

Fonction EXTRACT

La fonction EXTRACT renvoie une partie de date ou d'heure à partir d'une valeur TIMESTAMP, TIMESTAMPTZ, TIME ou TIMETZ. Les exemples incluent le jour, le mois, l'année, l'heure, la minute, la seconde, la milliseconde ou la microseconde d'un horodatage.

Syntaxe

```
EXTRACT(datepart FROM source)
```

Arguments

datepart

Sous-champ d'une date ou d'une heure à extraire, tel que le jour, le mois, l'année, l'heure, la minute, la seconde, la milliseconde ou la microseconde. Pour les valeurs possibles, consultez Parties de date pour les fonctions de date ou d'horodatage.

source

Une colonne ou une expression qui évalue un type de données TIMESTAMP, TIMESTAMPTZ, TIME ou TIMETZ.

Type de retour

INTEGER si la valeur source est de type TIMESTAMP, TIME ou TIMETZ.

DOUBLE PRECISION si la valeur source est de type TIMESTAMPTZ.

Exemples avec TIME

L'exemple de table TIME_TEST suivant comporte une colonne TIME_VAL (type TIME) avec trois valeurs insérées.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

L’exemple suivant extrait les minutes de chaque time_val.

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

L’exemple suivant extrait les heures de chaque time_val.

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

hours
-----------
         20
         0
         0
```

Fonction FROM_UTC_TIMESTAMP

La fonction FROM_UTC_TIMESTAMP convertit la date d'entrée de l'UTC (temps universel coordonné) vers le fuseau horaire spécifié.

Cette fonction est utile lorsque vous devez convertir des valeurs de date et d'heure de l'UTC vers un fuseau horaire spécifique. Cela peut être important lorsque vous travaillez avec des données provenant de différentes régions du monde et qui doivent être présentées à l'heure locale appropriée.

Syntaxe

```
from_utc_timestamp(timestamp, timezone
```

Arguments

timestamp

Une expression TIMESTAMP avec un horodatage UTC.

timezone

Expression STRING correspondant à un fuseau horaire valide dans lequel la date ou l'horodatage en entrée doivent être convertis.

Renvoie

La fonction FROM_UTC_TIMESTAMP renvoie un TIMESTAMP.

Exemple

L'exemple suivant convertit la date d'entrée de l'heure UTC vers le fuseau horaire spécifié (`'Asia/Seoul'`), qui dans ce cas est 9 heures avant l'heure UTC. Le résultat obtenu est la date et l'heure dans le fuseau horaire de Séoul, qui est`2016-08-31 09:00:00`.

```
SELECT from_utc_timestamp('2016-08-31', 'Asia/Seoul');
 2016-08-31 09:00:00
```

Fonction HOUR

La fonction HOUR est une fonction d'extraction de l'heure qui prend une heure ou un horodatage en entrée et renvoie le composant horaire (une valeur comprise entre 0 et 23).

Cette fonction d'extraction temporelle est utile lorsque vous devez travailler avec des composants spécifiques d'une heure ou d'un horodatage, par exemple lorsque vous effectuez des calculs temporels, que vous filtrez des données ou que vous formatez des valeurs temporelles.

Syntaxe

```
hour(timestamp)
```

Arguments

timestamp

Expression TIMESTAMP.

Renvoie

La fonction HOUR renvoie un INTEGER.

Exemple

L'exemple suivant extrait le composant horaire (12) de l'horodatage `'2009-07-30 12:58:59'` d'entrée.

```
SELECT hour('2009-07-30 12:58:59');
 12
```

Fonction MINUTE

La fonction MINUTE est une fonction d'extraction de l'heure qui prend une heure ou un horodatage comme entrée et renvoie le composant minute (une valeur comprise entre 0 et 60).

Syntaxe

```
minute(timestamp)
```

Arguments

timestamp

Une expression TIMESTAMP ou une CHAÎNE d'un format d'horodatage valide.

Renvoie

La fonction MINUTE renvoie un INTEGER.

Exemple

L'exemple suivant extrait le composant minute (58) de l'horodatage `'2009-07-30 12:58:59'` d'entrée.

```
SELECT minute('2009-07-30 12:58:59');
 58
```

Fonction MONTH

La fonction MONTH est une fonction d'extraction de l'heure qui prend une heure ou un horodatage comme entrée et renvoie le composant du mois (une valeur comprise entre 0 et 12).

Syntaxe

```
month(date)
```

Arguments

date

Une expression TIMESTAMP ou une CHAÎNE d'un format d'horodatage valide.

Renvoie

La fonction MONTH renvoie un INTEGER.

Exemple

L'exemple suivant extrait le composant du mois (7) de l'horodatage `'2016-07-30'` d'entrée.

```
SELECT month('2016-07-30');
 7
```

DEUXIÈME fonction

La fonction SECOND est une fonction d'extraction temporelle qui prend une heure ou un horodatage en entrée et renvoie le second composant (une valeur comprise entre 0 et 60).

Syntaxe

```
second(timestamp)
```

Arguments

timestamp

Expression TIMESTAMP.

Renvoie

La fonction SECOND renvoie un INTEGER.

Exemple

L'exemple suivant extrait le deuxième composant (59) de l'horodatage `'2009-07-30 12:58:59'` d'entrée.

```
SELECT second('2009-07-30 12:58:59');
```

```
 59
```

Fonction TIMESTAMP

La fonction TIMESTAMP prend une valeur (généralement un nombre) et la convertit en un type de données d'horodatage.

Cette fonction est utile lorsque vous devez convertir une valeur numérique représentant une heure ou une date en un type de données d'horodatage. Cela peut être utile lorsque vous travaillez avec des données stockées dans un format numérique, comme les horodatages Unix ou l'heure d'époque.

Syntaxe

```
timestamp(expr)
```

Arguments

expr

Toute expression pouvant être convertie en TIMESTAMP.

Renvoie

La fonction TIMESTAMP renvoie un TIMESTAMP.

Exemple

L'exemple suivant convertit un timestamp numérique Unix (`1632416400`) en son type de données d'horodatage correspondant : 22 septembre 2021 à 12 h 00 UTC.

```
SELECT timestamp(1632416400);
 2021-09-22 12:00:00 UTC
```

Fonction TO_TIMESTAMP

TO_TIMESTAMP convertit une chaîne TIMESTAMP en TIMESTAMPTZ.

Syntaxe

```
to_timestamp (timestamp)
```

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

Arguments

timestamp

Chaîne d'horodatage ou type de données pouvant être converti en chaîne d'horodatage.

format

Chaîne littérale qui correspond aux modèles de date/heure de Spark. Pour les modèles de date/heure valides, voir Modèles de date/heure pour le formatage et l'analyse syntaxique.

Type de retour

TIMESTAMP

Exemples

L'exemple suivant montre comment utiliser la fonction TO_TIMESTAMP pour convertir une chaîne TIMESTAMP en TIMESTAMP.

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

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

Il est possible de transmettre la partie TO_TIMESTAMP d'une date. Les autres parties de la date sont définies sur des valeurs par défaut. L'heure est incluse dans la sortie :

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp
---------------------------
2017-01-01 00:00:00+00
```

L'instruction SQL suivante convertit la chaîne « 2011-12-18 24:38:15 » en TIMESTAMP. Le résultat est un horodatage qui tombe le jour suivant car le nombre d'heures est supérieur à 24 heures :

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

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

Fonction YEAR

La fonction YEAR est une fonction d'extraction de date qui prend une date ou un horodatage en entrée et renvoie le composant annuel (un nombre à quatre chiffres).

Syntaxe

```
year(date)
```

Arguments

date

Expression DATE ou TIMESTAMP.

Renvoie

La fonction YEAR renvoie un INTEGER.

Exemple

L'exemple suivant extrait le composant annuel (`2016`) de la date d'entrée `'2016-07-30'`.

```
SELECT year('2016-07-30');
 2016
```

L'exemple suivant extrait le composant annuel de la `birthday` colonne du `squirrels` tableau et renvoie les résultats sous forme de sortie de l'instruction SELECT. Le résultat de cette requête sera une liste de valeurs annuelles, une pour chaque ligne du `squirrels` tableau, représentant l'année de naissance de chaque écureuil.

```
SELECT year(birthday) FROM squirrels
```

Parties de date pour les fonctions de date ou d'horodatage

Le tableau suivant identifie les noms de partie de date et d’horodatage et les abréviations qui sont acceptées comme arguments pour les fonctions suivantes :

- DATE_ADD
- DATE_DIFF
- DATE_PART
- EXTRACT

Partie de date ou de temps	Abréviations
millénaire, millénaires	mil
siècle, siècles	s, siècle, siècles
décennie, décennies	déc
époque	époque (prise en charge par la EXTRACT)
année, années	an, ans
trimestre, trimestres	trim
mois	mois
semaine, semaines	s, sem
jour de la semaine	jdls (pris en charge par les fonctions DATE_PART et Fonction EXTRACT) Renvoie un nombre entier compris entre 0 et 6, en commençant par le dimanche. **Note** La partie de date DOW se comporte différemment de la partie de date jour de la semaine (D) utilisée pour les chaînes au format datetime. D s’appuie sur des nombres

Partie de date ou de temps	Abréviations
	entiers compris entre 1 et 7, où le dimanche est 1. Pour de plus amples informations, veuillez consulter Chaînes de format datetime.
jour de l'année	dayofyear, doy, dy, yearday (prise en charge par la EXTRACT)
jour, jours	d
heure, heures	h
minute, minutes	m, min
seconde, secondes	s
milliseconde, millisecondes	ms
microseconde, microsecondes	µs
timezone, timezone_hour, timezone_minute	Pris en charge par la EXTRACT pour l'horodatage avec fuseau horaire (TIMESTAMPTZ) uniquement.

Variations de résultats avec les secondes, les millisecondes et les microsecondes

Des différences mineures dans les résultats de la requête se produisent lorsque d'autres fonctions de date spécifient les secondes, les millisecondes ou les microsecondes comme des parties de date :

- La fonction EXTRACT renvoie des nombres entiers pour la partie de date spécifiée uniquement, sans tenir compte des parties de date de niveau supérieur et inférieur. Si la partie de date spécifiée est les secondes, les millisecondes et les microsecondes ne figurent pas dans le résultat. Si la partie de date spécifiée est les millisecondes, les secondes et les microsecondes ne sont pas incluses. Si la partie de date spécifiée est les microsecondes, les secondes et les millisecondes ne sont pas incluses.
- La fonction DATE_PART renvoie la seconde partie complète de l'horodatage, quelle que soit la partie de date spécifiée, en renvoyant une valeur décimale ou un nombre entier comme requis.

Remarques sur CENTURY, EPOCH, DECADE et MIL

CENTURY ou CENTURIES

AWS Clean Rooms interprète un CENTURY comme commençant par l'année ## #1 et se terminant par l'année : ###0

```
select extract (century from timestamp '2000-12-16 12:21:13');
date_part
-----------
20
(1 row)

select extract (century from timestamp '2001-12-16 12:21:13');
date_part
-----------
21
(1 row)
```

EPOCH

L' AWS Clean Rooms implémentation d'EPOCH est relative au 1970-01-01 00:00:00.000 quel que soit le fuseau horaire dans lequel réside le cluster. Vous devrez peut-être décaler les résultats de la différence en heures selon le fuseau horaire sur lequel se trouve le cluster.

DECADE ou DECADES

AWS Clean Rooms interprète le DECADE ou DECADES DATEPART en fonction du calendrier commun. Par exemple, si le calendrier commun commence à partir de l'année 1, la première décennie (décennie 1) est 0001-01-01 jusqu'au 0009-12-31, et la deuxième décennie (décennie 2) du 0010-01-01 au 0019-12-31. Par exemple, la décennie 201 s'étend du 2000-01-01 au 2009-12-31 :

```
select extract(decade from timestamp '1999-02-16 20:38:40');
date_part
-----------
200
(1 row)

select extract(decade from timestamp '2000-02-16 20:38:40');
date_part
-----------
```

```
201
(1 row)

select extract(decade from timestamp '2010-02-16 20:38:40');
date_part
-----------
202
(1 row)
```

MIL ou MILS

AWS Clean Rooms interprète un MIL comme commençant par le premier jour de l'année #001 et se terminant par le dernier jour de l'année #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)
```

Fonctions de chiffrement et de déchiffrement

Les fonctions de chiffrement et de déchiffrement aident les développeurs SQL à protéger les données sensibles contre tout accès non autorisé ou toute utilisation abusive en les convertissant entre une forme lisible en texte clair et une forme chiffrée illisible.

AWS Clean Rooms Spark SQL prend en charge les fonctions de chiffrement et de déchiffrement suivantes :

Rubriques

- Fonction AES_ENCRYPT
- Fonction AES_DECRYPT

Fonction AES_ENCRYPT

La fonction AES_ENCRYPT est utilisée pour chiffrer les données à l'aide de l'algorithme AES (Advanced Encryption Standard).

Syntaxe

```
aes_encrypt(expr, key[, mode[, padding[, iv[, aad]]]])
```

Arguments

expr

La valeur binaire à chiffrer.

key

Phrase secrète à utiliser pour chiffrer les données.

Les longueurs de clé de 16, 24 et 32 bits sont prises en charge.

mode

Spécifie le mode de chiffrement par blocs à utiliser pour chiffrer les messages.

Modes valides : ECB (électronique CodeBook), GCM (mode Galois/Counter), CBC (Cipher-Block Chaining).

rembourrage

Spécifie comment ajouter des messages dont la longueur n'est pas un multiple de la taille du bloc.

Valeurs valides : PKCS, NONE, DEFAULT.

Le remplissage DEFAULT signifie PKCS (Public Key Cryptography Standards) pour ECB, NONE pour GCM et PKCS pour CBC.

Les combinaisons prises en charge de (mode, rembourrage) sont (« ECB », « PKCS »), (« GCM », « NONE ») et (« CBC », « PKCS »).

iv

Vecteur d'initialisation facultatif (IV). Compatible uniquement avec les modes CBC et GCM.

Valeurs valides : 12 octets pour le GCM et 16 octets pour le CBC.

aad

Données authentifiées supplémentaires (AAD) facultatives. Compatible uniquement avec le mode GCM. Il peut s'agir de n'importe quelle entrée libre et doit être fournie à la fois pour le chiffrement et le déchiffrement.

Type de retour

La fonction AES_ENCRYPT renvoie une valeur cryptée de expr en utilisant AES dans un mode donné avec le rembourrage spécifié.

Exemples

L'exemple suivant montre comment utiliser la fonction Spark SQL AES_ENCRYPT pour chiffrer de manière sécurisée une chaîne de données (dans ce cas, le mot « Spark ») à l'aide d'une clé de chiffrement spécifiée. Le texte chiffré obtenu est ensuite codé en Base64 pour faciliter son stockage ou sa transmission.

```
SELECT base64(aes_encrypt('Spark', 'abcdefghijklmnop'));
  4A5jOAh9FNGwoMeuJukfllrLdHEZxA2DyuSQAWz77dfn
```

L'exemple suivant montre comment utiliser la fonction Spark SQL AES_ENCRYPT pour chiffrer de manière sécurisée une chaîne de données (dans ce cas, le mot « Spark ») à l'aide d'une clé de chiffrement spécifiée. Le texte chiffré obtenu est ensuite représenté au format hexadécimal, ce qui peut être utile pour des tâches telles que le stockage, la transmission ou le débogage de données.

```
SELECT hex(aes_encrypt('Spark', '0000111122223333'));
 83F16B2AA704794132802D248E6BFD4E380078182D1544813898AC97E709B28A94
```

L'exemple suivant montre comment utiliser la fonction Spark SQL AES_ENCRYPT pour chiffrer de manière sécurisée une chaîne de données (dans ce cas, « Spark SQL ») à l'aide d'une clé de chiffrement, d'un mode de chiffrement et d'un mode de remplissage spécifiés. Le texte chiffré obtenu est ensuite codé en Base64 pour faciliter son stockage ou sa transmission.

```
SELECT base64(aes_encrypt('Spark SQL', '1234567890abcdef', 'ECB', 'PKCS'));
 3lmwu+Mw0H3fi5NDvcu9lg==
```

Fonction AES_DECRYPT

La fonction AES_DECRYPT est utilisée pour déchiffrer les données à l'aide de l'algorithme AES (Advanced Encryption Standard).

Syntaxe

```
aes_decrypt(expr, key[, mode[, padding[, aad]]])
```

Arguments

expr

La valeur binaire à déchiffrer.

key

Phrase secrète à utiliser pour déchiffrer les données.

La phrase secrète doit correspondre à la clé utilisée à l'origine pour produire la valeur cryptée et avoir une longueur de 16, 24 ou 32 octets.

mode

Spécifie le mode de chiffrement par blocs à utiliser pour déchiffrer les messages.

Modes valides : ECB, GCM, CBC.

rembourrage

Spécifie comment ajouter des messages dont la longueur n'est pas un multiple de la taille du bloc.

Valeurs valides : PKCS, NONE, DEFAULT.

Le rembourrage DEFAULT signifie PKCS pour ECB, NONE pour GCM et PKCS pour CBC.

aad

Données authentifiées supplémentaires (AAD) facultatives. Compatible uniquement avec le mode GCM. Il peut s'agir de n'importe quelle entrée libre et doit être fournie à la fois pour le chiffrement et le déchiffrement.

Type de retour

Renvoie une valeur déchiffrée de expr en utilisant AES en mode avec rembourrage.

Exemples

L'exemple suivant montre comment utiliser la fonction Spark SQL AES_ENCRYPT pour chiffrer de manière sécurisée une chaîne de données (dans ce cas, le mot « Spark ») à l'aide d'une clé de chiffrement spécifiée. Le texte chiffré obtenu est ensuite codé en Base64 pour faciliter son stockage ou sa transmission.

```
SELECT base64(aes_encrypt('Spark', 'abcdefghijklmnop'));
  4A5jOAh9FNGwoMeuJukfllrLdHEZxA2DyuSQAWz77dfn
```

L'exemple suivant montre comment utiliser la fonction Spark SQL AES_DECRYPT pour déchiffrer des données précédemment chiffrées et codées en Base64. Le processus de déchiffrement nécessite la clé de chiffrement et les paramètres appropriés (mode de chiffrement et mode de remplissage) pour récupérer correctement les données en texte brut d'origine.

```
SELECT aes_decrypt(unbase64('3lmwu+Mw0H3fi5NDvcu9lg=='), '1234567890abcdef', 'ECB',
 'PKCS');
 Spark SQL
```

Fonctions de hachage

Une fonction de hachage est une fonction mathématique qui convertit une valeur d'entrée numérique en une autre valeur.

AWS Clean Rooms Spark SQL prend en charge les fonctions de hachage suivantes :

Rubriques

- MD5 fonction
- Fonction SHA
- SHA1 fonction
- SHA2 fonction
- HASH64 fonction xx

MD5 fonction

Utilise la fonction de hachage MD5 cryptographique pour convertir une chaîne de longueur variable en une chaîne de 32 caractères qui est une représentation textuelle de la valeur hexadécimale d'une somme de contrôle de 128 bits.

Syntaxe

```
MD5(string)
```

Arguments

string

Chaîne de longueur variable.

Type de retour

La MD5 fonction renvoie une chaîne de 32 caractères qui est une représentation textuelle de la valeur hexadécimale d'une somme de contrôle de 128 bits.

Exemples

L'exemple suivant illustre la valeur de 128 bits de la chaîne « AWS Clean Rooms » :

```
select md5('AWS Clean Rooms');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

Fonction SHA

Synonyme de SHA1 fonction.

Consultez SHA1 fonction.

SHA1 fonction

La SHA1 fonction utilise la fonction de hachage SHA1 cryptographique pour convertir une chaîne de longueur variable en une chaîne de 40 caractères qui est une représentation textuelle de la valeur hexadécimale d'une somme de contrôle de 160 bits.

Syntaxe

SHA1 est un synonyme de. Fonction SHA

```
SHA1(string)
```

Arguments

string

Chaîne de longueur variable.

Type de retour

La SHA1 fonction renvoie une chaîne de 40 caractères qui est une représentation textuelle de la valeur hexadécimale d'une somme de contrôle de 160 bits.

exemple

L'exemple suivant renvoie la valeur de 160 bits du mot « AWS Clean Rooms » :

```
select sha1('AWS Clean Rooms');
```

SHA2 fonction

La SHA2 fonction utilise la fonction de hachage SHA2 cryptographique pour convertir une chaîne de longueur variable en chaîne de caractères. La chaîne de caractères est une représentation textuelle de la valeur hexadécimale du total de contrôle avec le nombre de bits spécifié.

Syntaxe

```
SHA2(string, bits)
```

Arguments

string

Chaîne de longueur variable.

integer

Nombre de bits dans les fonctions de hachage. Les valeurs valides sont 0 (identique à 256), 224, 256, 384 et 512.

Type de retour

La SHA2 fonction renvoie une chaîne de caractères qui est une représentation textuelle de la valeur hexadécimale de la somme de contrôle ou une chaîne vide si le nombre de bits n'est pas valide.

exemple

L'exemple suivant renvoie la valeur de 256 bits du mot « AWS Clean Rooms » :

```
select sha2('AWS Clean Rooms', 256);
```

HASH64 fonction xx

La fonction xxhash64 renvoie une valeur de hachage des arguments sur 64 bits.

La fonction xxhash64 () est une fonction de hachage non cryptographique conçue pour être rapide et efficace. Il est souvent utilisé dans les applications de traitement et de stockage de données, où un identifiant unique est nécessaire pour une donnée, mais le contenu exact des données n'a pas besoin d'être gardé secret.

Dans le contexte d'une requête SQL, la fonction xxhash64 () peut être utilisée à diverses fins, par exemple :

- Génération d'un identifiant unique pour une ligne d'un tableau
- Partitionnement des données en fonction d'une valeur de hachage
- Mise en œuvre de stratégies d'indexation ou de distribution de données personnalisées

Le cas d'utilisation spécifique dépendra des exigences de l'application et des données traitées.

Syntaxe

```
xxhash64(expr1, expr2, ...)
```

Arguments

expr1

Expression de n'importe quel type.

expr2

Expression de n'importe quel type.

Renvoie

Renvoie une valeur de hachage des arguments sur 64 bits (BIGINT). La graine de haschisch est de 42.

exemple

L'exemple suivant génère une valeur de hachage de 64 bits (5602566077635097486) en fonction de l'entrée fournie. Le premier argument est une valeur de chaîne, dans ce cas, le mot « Spark ». Le deuxième argument est un tableau contenant la seule valeur entière 123. Le troisième argument est une valeur entière représentant le point de départ de la fonction de hachage.

```
SELECT xxhash64('Spark', array(123), 2);
 5602566077635097486
```

Fonctions Hyperloglog

Les fonctions HyperLogLog (HLL) de SQL permettent d'estimer efficacement le nombre d'éléments uniques (cardinalité) dans un ensemble de données volumineux, même lorsque l'ensemble réel d'éléments uniques n'est pas stocké.

Les principaux avantages de l'utilisation des fonctions HLL sont les suivants :

- Efficacité de la mémoire : les esquisses HLL nécessitent beaucoup moins de mémoire que le stockage de l'ensemble complet d'éléments uniques, ce qui les rend adaptées à de grands ensembles de données.
- Informatique distribuée : les esquisses HLL peuvent être combinées entre plusieurs sources de données ou nœuds de traitement, ce qui permet une estimation distribuée efficace du nombre unique.
- Résultats approximatifs : HLL fournit une estimation du nombre unique approximative, avec un compromis ajustable entre précision et utilisation de la mémoire (via le paramètre de précision).

Ces fonctions sont particulièrement utiles dans les scénarios où vous devez estimer le nombre d'éléments uniques, tels que dans les applications d'analyse, d'entreposage de données et de traitement de flux en temps réel.

AWS Clean Rooms prend en charge les fonctions HLL suivantes.

Rubriques

- Fonction HLL_SKETCH_AGG
- Fonction HLL_SKETCH_ESTIMATE
- Fonction HLL_UNION
- Fonction HLL_UNION_AGG

Fonction HLL_SKETCH_AGG

La fonction d'agrégation HLL_SKETCH_AGG crée une esquisse HLL à partir des valeurs de la colonne spécifiée. Elle renvoie un type de données HLLSKETCH qui encapsule les valeurs des expressions d'entrée.

La fonction d'agrégation HLL_SKETCH_AGG fonctionne avec tous les types de données et ignore les valeurs NULL.

Lorsqu'il n'y a pas de lignes dans une table ou que toutes les lignes sont NULL, le schéma résultant n'a pas de paires index-valeur telles que `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}`.

Syntaxe

```
HLL_SKETCH_AGG (aggregate_expression[, lgConfigK ] )
```

Argument

aggregate_expression

Toute expression de type INT, BIGINT, STRING ou BINARY par rapport à laquelle un comptage unique sera effectué. Toutes NULL les valeurs sont ignorées.

LG Configk

Une constante INT optionnelle comprise entre 4 et 21 inclus avec 12 par défaut. Log-base-2 de K, où K est le nombre de compartiments ou de fentes pour l'esquisse.

Type de retour

La fonction HLL_SKETCH_AGG renvoie un tampon BINAIRE non NULL contenant l' HyperLogLog esquisse calculée en raison de la consommation et de l'agrégation de toutes les valeurs d'entrée du groupe d'agrégation.

Exemples

Les exemples suivants utilisent l'algorithme HyperLogLog (HLL) pour estimer le nombre distinct de valeurs dans la `col` colonne. La `hll_sketch_agg(col, 12)` fonction agrège les valeurs de la colonne col pour créer une esquisse HLL avec une précision de 12. La `hll_sketch_estimate()` fonction est ensuite utilisée pour estimer le nombre distinct de valeurs sur la base de l'esquisse HLL générée. Le résultat final de la requête est 3, ce qui représente le nombre distinct estimé de valeurs dans la `col` colonne. Dans ce cas, les valeurs distinctes sont 1, 2 et 3.

```
SELECT hll_sketch_estimate(hll_sketch_agg(col, 12))
    FROM VALUES (1), (1), (2), (2), (3) tab(col);
  3
```

L'exemple suivant utilise également l'algorithme HLL pour estimer le nombre distinct de valeurs dans la `col` colonne, mais il ne spécifie pas de valeur de précision pour l'esquisse HLL. Dans ce cas, il utilise la précision par défaut de 14. La `hll_sketch_agg(col)` fonction prend les valeurs de la `col` colonne et crée une esquisse HyperLogLog (HLL), qui est une structure de données compacte qui peut être utilisée pour estimer le nombre distinct d'éléments. La `hll_sketch_estimate(hll_sketch_agg(col))` fonction prend l'esquisse HLL créée à l'étape précédente et calcule une estimation du nombre distinct de valeurs dans la `col` colonne. Le résultat final de la requête est 3, ce qui représente le nombre distinct estimé de valeurs dans la `col` colonne. Dans ce cas, les valeurs distinctes sont 1, 2 et 3.

```
SELECT hll_sketch_estimate(hll_sketch_agg(col))
FROM VALUES (1), (1), (2), (2), (3) tab(col);
3
```

Fonction HLL_SKETCH_ESTIMATE

La fonction HLL_SKETCH_ESTIMATE prend une esquisse HLL et estime le nombre d'éléments uniques représentés par l'esquisse. Il utilise l'algorithme HyperLogLog (HLL) pour compter une approximation probabiliste du nombre de valeurs uniques dans une colonne donnée, en consommant une représentation binaire connue sous le nom de tampon d'esquisse précédemment générée par la fonction HLL_SKETCH_AGG et en renvoyant le résultat sous la forme d'un grand entier.

L'algorithme d'esquisse HLL fournit un moyen efficace d'estimer le nombre d'éléments uniques, même pour de grands ensembles de données, sans avoir à stocker l'ensemble complet des valeurs uniques.

Les `hll_union_agg` fonctions `hll_union` et peuvent également combiner des esquisses en consommant et en fusionnant ces tampons en tant qu'entrées.

Syntaxe

```
HLL_SKETCH_ESTIMATE (hllsketch_expression)
```

Argument

hllsketch_expression

`BINARY`Expression contenant une esquisse générée par HLL_SKETCH_AGG

Type de retour

La fonction HLL_SKETCH_ESTIMATE renvoie une valeur BIGINT correspondant au nombre distinct approximatif représenté par l'esquisse en entrée.

Exemples

Les exemples suivants utilisent l'algorithme d'esquisse HyperLogLog (HLL) pour estimer la cardinalité (nombre unique) des valeurs de la colonne. `col` La `hll_sketch_agg(col, 12)` fonction prend la `col` colonne et crée une esquisse HLL avec une précision de 12 bits. L'esquisse HLL est une structure de données approximative qui permet d'estimer efficacement le nombre d'éléments uniques dans un ensemble. La `hll_sketch_estimate()` fonction prend l'esquisse HLL créée par `hll_sketch_agg` et estime la cardinalité (nombre unique) des valeurs représentées par l'esquisse. `FROM VALUES (1), (1), (2), (2), (3) tab(col);`Génère un ensemble de données de test de 5 lignes, où la `col` colonne contient les valeurs 1, 1, 2, 2 et 3. Le résultat de cette requête est le nombre unique estimé des valeurs de la `col` colonne, qui est de 3.

```
SELECT hll_sketch_estimate(hll_sketch_agg(col, 12))
    FROM VALUES (1), (1), (2), (2), (3) tab(col);
  3
```

La différence entre l'exemple suivant et le précédent est que le paramètre de précision (12 bits) n'est pas spécifié dans l'appel de `hll_sketch_agg` fonction. Dans ce cas, la précision par défaut de 14 bits est utilisée, ce qui peut fournir une estimation plus précise du nombre unique par rapport à l'exemple précédent qui utilisait 12 bits de précision.

```
SELECT hll_sketch_estimate(hll_sketch_agg(col))
FROM VALUES (1), (1), (2), (2), (3) tab(col);
3
```

Fonction HLL_UNION

La fonction HLL_UNION combine deux esquisses HLL en une seule esquisse unifiée. Il utilise l'algorithme HyperLogLog (HLL) pour combiner deux esquisses en une seule. Les requêtes peuvent utiliser les tampons obtenus pour calculer des nombres uniques approximatifs sous forme de longs entiers avec la `hll_sketch_estimate` fonction.

Syntaxe

```
HLL_UNION (( expr1, expr2 [, allowDifferentLgConfigK ] ))
```

Argument

ExPRN

`BINARY`Expression contenant une esquisse générée par HLL_SKETCH_AGG.

allowDifferentLgConfiguration K

Expression BOOLEAN facultative contrôlant s'il faut autoriser la fusion de deux esquisses avec des valeurs LGConfigk différentes. La valeur par défaut est `false`.

Type de retour

La fonction HLL_UNION renvoie une mémoire tampon BINAIRE contenant l' HyperLogLog esquisse calculée à la suite de la combinaison des expressions d'entrée. Lorsque le `allowDifferentLgConfigK` paramètre est défini`true`, l'esquisse du résultat utilise la plus petite des deux `lgConfigK` valeurs fournies.

Exemples

Les exemples suivants utilisent l'algorithme d'esquisse HyperLogLog (HLL) pour estimer le nombre unique de valeurs sur deux colonnes `col1` et `col2` dans un ensemble de données.

La `hll_sketch_agg(col1)` fonction crée une esquisse HLL pour les valeurs uniques de la `col1` colonne.

La `hll_sketch_agg(col2)` fonction crée une esquisse HLL pour les valeurs uniques de la colonne col2.

La `hll_union(...)` fonction combine les deux esquisses HLL créées aux étapes 1 et 2 en une seule esquisse HLL unifiée.

La `hll_sketch_estimate(...)` fonction prend l'esquisse HLL combinée et estime le nombre unique de valeurs entre les deux `col1` et`col2`.

La `FROM VALUES` clause génère un ensemble de données de test de 5 lignes, `col1` contenant les valeurs 1, 1, 2, 2 et 3, et `col2` contenant les valeurs 4, 4, 5, 5 et 6.

Le résultat de cette requête est le nombre unique estimé de valeurs entre les deux `col1` et`col2`, qui est de 6. L'algorithme d'esquisse HLL fournit un moyen efficace d'estimer le nombre d'éléments uniques, même pour de grands ensembles de données, sans avoir à stocker l'ensemble complet des valeurs uniques. Dans cet exemple, la `hll_union` fonction est utilisée pour combiner les esquisses HLL des deux colonnes, ce qui permet d'estimer le nombre unique pour l'ensemble de données, plutôt que pour chaque colonne individuellement.

```
SELECT hll_sketch_estimate(
  hll_union(
    hll_sketch_agg(col1),
    hll_sketch_agg(col2)))
  FROM VALUES
    (1, 4),
    (1, 4),
    (2, 5),
    (2, 5),
    (3, 6) AS tab(col1, col2);
  6
```

La différence entre l'exemple suivant et le précédent est que le paramètre de précision (12 bits) n'est pas spécifié dans l'appel de `hll_sketch_agg` fonction. Dans ce cas, la précision par défaut de 14 bits est utilisée, ce qui peut fournir une estimation plus précise du nombre unique par rapport à l'exemple précédent qui utilisait 12 bits de précision.

```
SELECT hll_sketch_estimate(
  hll_union(
    hll_sketch_agg(col1, 14),
    hll_sketch_agg(col2, 14)))
```

```
  FROM VALUES
    (1, 4),
    (1, 4),
    (2, 5),
    (2, 5),
    (3, 6) AS tab(col1, col2);
```

Fonction HLL_UNION_AGG

La fonction HLL_UNION_AGG combine plusieurs esquisses HLL en une seule esquisse unifiée. Il utilise l'algorithme HyperLogLog (HLL) pour combiner un groupe de croquis en un seul. Les requêtes peuvent utiliser les tampons obtenus pour calculer des nombres uniques approximatifs à l'aide de la `hll_sketch_estimate` fonction.

Syntaxe

```
HLL_UNION_AGG ( expr [, allowDifferentLgConfigK ] )
```

Argument

expr

`BINARY`Expression contenant une esquisse générée par HLL_SKETCH_AGG.

allowDifferentLgConfiguration K

Expression BOOLEAN facultative contrôlant s'il faut autoriser la fusion de deux esquisses avec des valeurs LGConfigk différentes. La valeur par défaut est `false`.

Type de retour

La fonction HLL_UNION_AGG renvoie une mémoire tampon BINAIRE contenant l' HyperLogLog esquisse calculée en combinant les expressions d'entrée du même groupe. Lorsque le `allowDifferentLgConfigK` paramètre est défini`true`, l'esquisse du résultat utilise la plus petite des deux `lgConfigK` valeurs fournies.

Exemples

Les exemples suivants utilisent l'algorithme d'esquisse HyperLogLog (HLL) pour estimer le nombre unique de valeurs dans plusieurs esquisses HLL.

Le premier exemple estime le nombre unique de valeurs dans un ensemble de données.

```
SELECT hll_sketch_estimate(hll_union_agg(sketch, true))
    FROM (SELECT hll_sketch_agg(col) as sketch
            FROM VALUES (1) AS tab(col)
          UNION ALL
          SELECT hll_sketch_agg(col, 20) as sketch
            FROM VALUES (1) AS tab(col));
  1
```

La requête interne crée deux esquisses HLL :

- La première instruction SELECT crée une esquisse à partir d'une valeur unique de 1.
- La deuxième instruction SELECT crée une esquisse à partir d'une autre valeur unique de 1, mais avec une précision de 20.

La requête externe utilise la fonction HLL_UNION_AGG pour combiner les deux esquisses en une seule esquisse. Il applique ensuite la fonction HLL_SKETCH_ESTIMATE à cette esquisse combinée pour estimer le nombre unique de valeurs.

Le résultat de cette requête est le nombre unique estimé des valeurs de la `col` colonne, qui est1. Cela signifie que les deux valeurs d'entrée de 1 sont considérées comme uniques, même si elles ont la même valeur.

Le deuxième exemple inclut un paramètre de précision différent pour la fonction HLL_UNION_AGG. Dans ce cas, les deux esquisses HLL sont créées avec une précision de 14 bits, ce qui permet de les combiner avec succès à l'aide `hll_union_agg` du `true` paramètre.

```
SELECT hll_sketch_estimate(hll_union_agg(sketch, true))
    FROM (SELECT hll_sketch_agg(col, 14) as sketch
            FROM VALUES (1) AS tab(col)
          UNION ALL
          SELECT hll_sketch_agg(col, 14) as sketch
            FROM VALUES (1) AS tab(col));
  1
```

Le résultat final de la requête est le nombre unique estimé, qui dans ce cas l'est également1. Cela signifie que les deux valeurs d'entrée de 1 sont considérées comme uniques, même si elles ont la même valeur.

Fonctions JSON

Lorsque vous avez besoin de stocker un ensemble relativement petit de paires clé-valeur, vous pouvez économiser de l'espace en stockant les données au format JSON. Étant donné que les chaînes au format JSON peuvent être stockées dans une seule colonne, l'utilisation de JSON peut être plus efficace que de stocker vos données sous forme de table.

Example

Supposons, par exemple, que vous disposiez d'un tableau clairsemé, dans lequel vous devez disposer de nombreuses colonnes pour représenter pleinement tous les attributs possibles. Cependant, la plupart des valeurs de colonne sont NULL pour une ligne ou une colonne donnée. En utilisant le JSON pour le stockage, vous pouvez peut-être stocker les données d'une ligne sous forme de paires clé-valeur dans une seule chaîne JSON et éliminer les colonnes de table peu remplies.

En outre, vous pouvez facilement modifier les chaînes au format JSON pour stocker des paires clé:valeur supplémentaires sans avoir besoin d'ajouter des colonnes à une table.

Nous vous conseillons d'utiliser JSON avec modération. Le JSON n'est pas un bon choix pour stocker des ensembles de données plus volumineux car, en stockant des données disparates dans une seule colonne, le JSON n'utilise pas l'architecture du magasin de AWS Clean Rooms colonnes.

JSON utilise des chaînes de texte codées UTF-8, les chaînes JSON peuvent donc être stockées sous forme de types de données CHAR ou VARCHAR. Utilisez VARCHAR si les chaînes incluent des caractères de plusieurs octets.

Les chaînes JSON doivent être au bon format JSON, selon les règles suivantes :

- Le JSON de niveau racine peut être un objet JSON ou un tableau JSON. Un objet JSON est un ensemble non trié de paires clé:valeur séparées par des virgules délimitées par des accolades.

  Par exemple, `{"one":1, "two":2}`
- Un tableau JSON est un ensemble ordonné de valeurs séparées par des virgules délimitées par des crochets.

  Voici un exemple : `["first", {"one":1}, "second", 3, null]` .
- Les tableaux JSON utilisent un index de base zéro ; le premier élément d'un tableau se trouve à la position 0. Dans une paire clé:valeur JSON, la clé est une chaîne entre guillemets doubles.

- Une valeur JSON peut être l'une des suivantes :
  - Objet JSON
  - un tableau JSON
  - Chaîne entre guillemets
  - Nombre (entier et à virgule flottante)
  - Booléen
  - Null
- Les objets vides et les tableaux vides sont des valeurs JSON valides.
- Les champs JSON sont sensibles à la casse.
- Les espace vides entre les éléments structurels JSON (tel que `{ }, [ ]`) sont ignorés.

Rubriques

Fonction GET_JSON_OBJECT

La fonction GET_JSON_OBJECT extrait un objet json à partir de. `path`

Syntaxe

```
get_json_object(json_txt, path)
```

Arguments

json_txt

Expression STRING contenant du JSON bien formé.

chemin

Un littéral STRING avec une expression de chemin JSON bien formée.

Renvoie

Renvoie une chaîne.

Une valeur NULL est renvoyée si l'objet est introuvable.

exemple

L'exemple suivant extrait une valeur d'un objet JSON. Le premier argument est une chaîne JSON qui représente un objet simple avec une seule paire clé-valeur. Le deuxième argument est une expression de chemin JSON. Le $ symbole représente la racine de l'objet JSON, et la `.a` partie indique que nous voulons extraire la valeur associée à la clé a « ». La sortie de la fonction est « b », qui est la valeur associée à la touche « a » dans l'objet JSON d'entrée.

```
SELECT get_json_object('{"a":"b"}', '$.a');
 b
```

Fonction TO_JSON

La fonction TO_JSON convertit une expression d'entrée en une représentation sous forme de chaîne JSON. La fonction gère la conversion de différents types de données (tels que des nombres, des chaînes et des booléens) en leurs représentations JSON correspondantes.

La fonction TO_JSON est utile lorsque vous devez convertir des données structurées (telles que des lignes de base de données ou des objets JSON) dans un format plus portable et autodescriptif tel que JSON. Cela peut être particulièrement utile lorsque vous devez interagir avec d'autres systèmes ou services qui attendent des données au format JSON.

Syntaxe

```
to_json(expr[, options])
```

Arguments

expr

Expression d'entrée que vous souhaitez convertir en chaîne JSON. Il peut s'agir d'une valeur, d'une colonne ou de toute autre expression SQL valide.

options

Ensemble facultatif d'options de configuration qui peuvent être utilisées pour personnaliser le processus de conversion JSON. Ces options peuvent inclure des éléments tels que la gestion des valeurs nulles, la représentation de valeurs numériques et le traitement des caractères spéciaux.

Renvoie

Renvoie une chaîne JSON avec une valeur de structure donnée

Exemples

L'exemple suivant convertit une structure nommée (un type de données structurées) en chaîne JSON. Le premier argument(`named_struct('a', 1, 'b', 2)`) est l'expression d'entrée transmise à la `to_json()` fonction. Il crée une structure nommée avec deux champs : « a » avec une valeur de 1, et « b » avec une valeur de 2. La fonction to_json () prend la structure nommée comme argument et la convertit en une représentation sous forme de chaîne JSON. La sortie est`{"a":1,"b":2}`, qui est une chaîne JSON valide qui représente la structure nommée.

```
SELECT to_json(named_struct('a', 1, 'b', 2));
 {"a":1,"b":2}
```

L'exemple suivant convertit une structure nommée contenant une valeur d'horodatage en une chaîne JSON, avec un format d'horodatage personnalisé. Le premier argument (`named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd'))`) crée une structure nommée avec un seul champ « time » contenant la valeur d'horodatage. Le deuxième argument (`map('timestampFormat', 'dd/MM/yyyy')`) crée une carte (dictionnaire clé-valeur) avec une seule paire clé-valeur, où la clé est « TimestampFormat » et la valeur est « ». dd/MM/yyyy'. This map is used to specify the desired format for the timestamp value when converting it to JSON. The to_json() function converts the named struct into a JSON string. The second argument, the map, is used to customize the timestamp format to 'dd/MM/yyyy La sortie est `{"time":"26/08/2015"}` une chaîne JSON avec un seul champ « heure » contenant la valeur d'horodatage au format « dd/MM/yyyy » souhaité.

```
SELECT to_json(named_struct('time', to_timestamp('2015-08-26', 'yyyy-MM-dd')),
 map('timestampFormat', 'dd/MM/yyyy'));
 {"time":"26/08/2015"}
```

Fonctions mathématiques

Cette section décrit les opérateurs mathématiques et les fonctions pris en charge dans AWS Clean Rooms Spark SQL.

Rubriques

- Symboles d’opérateurs mathématiques

- Fonction ABS
- Fonction ACOS
- Fonction ASIN
- Fonction ATAN
- ATAN2 fonction
- Fonction CBRT
- Fonction CEILING (ou CEIL)
- Fonction COS
- Fonction COT
- Fonction DEGREES
- Fonction DIV
- Fonction EXP
- Fonction FLOOR
- Fonction LN
- Fonction LOG
- Fonction MOD
- Fonction PI
- Fonction POWER
- Fonction RADIANS
- Fonction RAND
- Fonction RANDOM
- Fonction ROUND
- Fonction SIGN
- Fonction SIN
- Fonction SQRT
- Fonction TRUNC

Symboles d’opérateurs mathématiques

Le tableau suivant répertorie les opérateurs mathématiques pris en charge.

Opérateurs pris en charge

Opérateur	Description	Exemple	Résultat
+	addition	2 + 3	5
-	soustraction	2-3	-1
*	multiplication	2 * 3	6
/	division	4 / 2	2
%	modulo	5 % 4	1
^	puissance	2.0 ^ 3.0	8

Exemples

Calculez la commission payée plus des frais de gestion de 2$ pour une transaction donnée :

```
select commission, (commission + 2.00) as comm
from sales where salesid=10000;

commission | comm
-----------+-------
28.05      | 30.05
(1 row)
```

Calculer 20 % du prix de vente pour une transaction donnée :

```
select pricepaid, (pricepaid * .20) as twentypct
from sales where salesid=10000;

pricepaid | twentypct
----------+-----------
187.00    |    37.400
(1 row)
```

Prévoyez le nombre de billets vendus en fonction d'un modèle de croissance continue. Dans cet exemple, la sous-requête renvoie le nombre de billets vendus en 2008. Ce résultat est multiplié de façon exponentielle par un taux de croissance continu de 5 % sur 10 ans.

```
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)
```

Trouvez le prix total payé et les commissions pour les ventes dont le numéro de date est supérieur ou égal à 2 000. Puis soustrayez la commission totale du prix total payé.

```
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)
```

Fonction ABS

ABS calcule la valeur absolue d’un nombre, où ce nombre peut être littéral ou une expression qui a pour valeur un nombre.

Syntaxe

```
ABS (number)
```

Arguments

number

Nombre ou expression ayant pour valeur un nombre. Il peut s'agir de SMALLINT, INTEGER, BIGINT, DECIMAL ou de type. FLOAT4 FLOAT8

Type de retour

ABS renvoie le même type de données que sont argument.

Exemples

Calculez la valeur absolue de -38 :

```
select abs (-38);
abs
-------
38
(1 row)
```

Calculez la valeur absolue de (14-76) :

```
select abs (14-76);
abs
-------
62
(1 row)
```

Fonction ACOS

ACOS est une fonction trigonométrique qui renvoie l’arc cosinus d’un nombre. La valeur de retour est exprimée en radians et se situe entre `0` et `PI`.

Syntaxe

```
ACOS(number)
```

Arguments

number

Le paramètre d'entrée est un nombre DOUBLE PRECISION.

Type de retour

DOUBLE PRECISION

Exemples

Pour renvoyer l'arc cosinus de -1, utilisez l'exemple suivant.

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Fonction ASIN

ASIN est une fonction trigonométrique qui renvoie l'arc sinus d'un nombre. La valeur de retour est exprimée en radians et se situe entre PI/2 et -PI/2.

Syntaxe

```
ASIN(number)
```

Arguments

number

Le paramètre d'entrée est un nombre DOUBLE PRECISION.

Type de retour

DOUBLE PRECISION

Exemples

Pour renvoyer l'arc sinus de 1, utilisez l'exemple suivant.

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

Fonction ATAN

ATAN est une fonction trigonométrique qui renvoie l'arc tangente d'un nombre. La valeur de retour est exprimée en radians et se situe entre `-PI` et `PI`.

Syntaxe

```
ATAN(number)
```

Arguments

number

Le paramètre d'entrée est un nombre `DOUBLE PRECISION`.

Type de retour

`DOUBLE PRECISION`

Exemples

Pour renvoyer l'arc tangente de 1 et le multiplier par 4, utilisez l'exemple suivant.

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
```

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

ATAN2 fonction

ATAN2 est une fonction trigonométrique qui renvoie l'arc tangente d'un nombre divisé par un autre nombre. La valeur de retour est exprimée en radians et se situe entre `PI/2` et `-PI/2`.

Syntaxe

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

Arguments

number1

Nombre `DOUBLE PRECISION`.

number2

Nombre `DOUBLE PRECISION`.

Type de retour

`DOUBLE PRECISION`

Exemples

Pour renvoyer l’arc tangente de `2/2` et le multiplier par 4, utilisez l’exemple suivant.

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Fonction CBRT

La fonction CBRT est une fonction mathématique qui calcule la racine cubique d'un nombre.

Syntaxe

```
CBRT (number)
```

Argument

CBRT prend un certain nombre DOUBLE PRECISION en tant qu'argument.

Type de retour

CBRT renvoie un nombre DOUBLE PRECISION.

Exemples

Calculez la racine cubique de la commission payée pour une transaction donnée :

```
select cbrt(commission) from sales where salesid=10000;

cbrt
------------------
3.03839539048843
(1 row)
```

Fonction CEILING (ou CEIL)

La fonction CEILING (ou CEIL) permet d'arrondir un nombre jusqu'au nombre entier supérieur suivant. (Le Fonction FLOOR arrondit un nombre au nombre entier inférieur suivant.)

Syntaxe

```
CEIL | CEILING(number)
```

Arguments

number

Nombre ou expression ayant pour valeur un nombre. Il peut s'agir de SMALLINT, INTEGER, BIGINT, DECIMAL ou de type. FLOAT4 FLOAT8

Type de retour

CEILING et CEIL renvoient le même type de données que leur argument.

Exemple

Calculez le plafond de la commission payée pour une transaction de vente donnée :

```
select ceiling(commission) from sales
where salesid=10000;

ceiling
---------
29
(1 row)
```

Fonction COS

COS est une fonction trigonométrique qui renvoie le cosinus d'un nombre. La valeur de retour est exprimée en radians et se situe entre -1 et 1, inclus.

Syntaxe

```
COS(double_precision)
```

Argument

number

Le paramètre d'entrée est un nombre double précision.

Type de retour

La fonction COS renvoie un nombre double précision.

Exemples

L'exemple suivant renvoie le cosinus de 0 :

```
select cos(0);
cos
-----
1
(1 row)
```

L'exemple suivant renvoie le cosinus de PI :

```
select cos(pi());
cos
-----
-1
(1 row)
```

Fonction COT

COT est une fonction trigonométrique qui renvoie la cotangente d'un nombre. Le paramètre d'entrée doit être différent de zéro.

Syntaxe

```
COT(number)
```

Argument

number

Le paramètre d'entrée est un nombre `DOUBLE PRECISION`.

Type de retour

`DOUBLE PRECISION`

Exemples

Pour renvoyer la cotangente de 1, utilisez l'exemple suivant.

```
SELECT COT(1);

+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

Fonction DEGREES

Convertit un angle en radians en son équivalent en degrés.

Syntaxe

```
DEGREES(number)
```

Argument

number

Le paramètre d'entrée est un nombre DOUBLE PRECISION.

Type de retour

DOUBLE PRECISION

Exemple

Pour renvoyer l'équivalent en degrés de 0,5 radian, utilisez l'exemple suivant.

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
+-------------------+
```

Pour convertir PI radians en degrés, utilisez l'exemple suivant.

```
SELECT DEGREES(pi());

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

Fonction DIV

L'opérateur DIV renvoie la partie intégrante de la division du dividende par diviseur.

Syntaxe

```
dividend div divisor
```

Arguments

dividende

Expression dont l'évaluation correspond à un chiffre ou à un intervalle.

divisor

Un intervalle correspondant de type if `dividend` est un intervalle, un intervalle numérique dans le cas contraire.

Type de retour

`BIGINT`

Exemples

L'exemple suivant sélectionne deux colonnes dans la table des écureuils : la `id` colonne, qui contient l'identifiant unique de chaque écureuil, et une `calculated` colonne`age div 2`, qui représente la division entière de la colonne d'âge par 2. Le `age div 2` calcul effectue une division entière sur la `age` colonne, arrondissant ainsi l'âge au nombre entier pair le plus proche. Par exemple, si la `age` colonne contient des valeurs telles que 3, 5, 7 et 10, elle contiendra les valeurs 1, 2, 3 et 5, respectivement. `age div 2`

```
SELECT id, age div 2 FROM squirrels
```

Cette requête peut être utile dans les scénarios où vous devez regrouper ou analyser des données en fonction de tranches d'âge, et vous souhaitez simplifier les valeurs d'âge en les arrondissant à l'entier pair le plus proche. Le résultat obtenu fournirait l'âge `id` et l'âge divisés par 2 pour chaque écureuil du `squirrels` tableau.

Fonction EXP

La fonction EXP implémente la fonction exponentielle pour une expression numérique, ou la base du logarithme naturel, e, élevée à la puissance de l'expression. La fonction EXP est l’inverse de Fonction LN.

Syntaxe

```
EXP (expression)
```

Argument

expression

L'expression doit être un type de données INTEGER, DECIMAL ou DOUBLE PRECISION.

Type de retour

EXP renvoie un nombre DOUBLE PRECISION.

Exemple

Utilisez la fonction EXP de planifier des ventes de billets selon un modèle de croissance continue. Dans cet exemple, la sous-requête renvoie le nombre de billets vendus en 2008. Ce résultat est multiplié par le résultat de la fonction EXP, qui spécifie une croissance continue de 7 % sur 10 ans.

```
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)
```

Fonction FLOOR

La fonction FLOOR arrondit un nombre au nombre entier inférieur suivant.

Syntaxe

```
FLOOR (number)
```

Argument

number

Nombre ou expression ayant pour valeur un nombre. Il peut s'agir de SMALLINT, INTEGER, BIGINT, DECIMAL ou de type. FLOAT4 FLOAT8

Type de retour

FLOOR renvoie le même type de données que sont argument.

Exemple

L'exemple montre la valeur de la commission payée pour une transaction de vente donnée avant et après l'utilisation de la fonction FLOOR.

```
select commission from sales
where salesid=10000;

floor
-------
28.05
(1 row)

select floor(commission) from sales
where salesid=10000;

floor
-------
28
(1 row)
```

Fonction LN

La fonction LN renvoie le logarithme naturel du paramètre d'entrée.

Syntaxe

```
LN(expression)
```

Argument

expression

Colonne cible ou expression sur laquelle la fonction opère.

> **Note**
>
> Cette fonction renvoie une erreur pour certains types de données si l'expression fait référence à une table AWS Clean Rooms créée par l'utilisateur ou à une table AWS Clean Rooms système STL ou STV.

Les expressions régulières avec les types de données suivants génèrent une erreur si elles font référence à une table créée par l'utilisateur ou à une table système.

- BOOLEAN
- CHAR
- DATE
- DECIMAL ou NUMERIC
- TIMESTAMP
- VARCHAR

Les expressions régulières avec les types de données suivants s'exécutent avec succès sur des tables créées par l'utilisateur ou des tables système STL ou STV :

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

Type de retour

La fonction LN renvoie le même type que l'expression.

Exemple

L'exemple suivant renvoie le logarithme naturel, ou logarithme de base e, du nombre 2,718281828 :

```
select ln(2.718281828);
ln
--------------------
0.9999999998311267
(1 row)
```

Notez que la réponse est presque égale à 1.

Cet exemple renvoie le logarithme naturel des valeurs de la colonne USERID de la table USERS :

```
select username, ln(userid) from users order by userid limit 10;

 username |        ln
----------+-------------------
 JSG99FHE |                 0
 PGL08LJI | 0.693147180559945
 IFT66TXU |  1.09861228866811
 XDZ38RDD |  1.38629436111989
 AEB55QTM |   1.6094379124341
 NDQ15VBM |  1.79175946922805
 OWY35QYB |  1.94591014905531
 AZG78YIP |  2.07944154167984
 MSD36KVR |  2.19722457733622
 WKW41AIW |  2.30258509299405
(10 rows)
```

Fonction LOG

Renvoie le logarithme de `expr` with`base`.

Syntaxe

```
LOG(base, expr)
```

Argument

expr

L’expression doit comporter un type de données de nombre entier, décimale ou à virgule flottante.

base

Base pour le calcul du logarithme. Doit être un nombre positif (différent de 1) de type de données à double précision.

Type de retour

La fonction LOG renvoie un nombre double précision.

Exemple

L'exemple suivant renvoie le logarithme de base 10 du chiffre 100 :

```
select log(10, 100);
--------
2
(1 row)
```

Fonction MOD

Renvoie le reste de deux nombres, autrement dit une opération modulo. Pour calculer le résultat, le premier paramètre est divisé par le second.

Syntaxe

```
MOD(number1, number2)
```

Arguments

number1

Le premier paramètre d'entrée est un nombre INTEGER, SMALLINT, BIGINT ou DECIMAL. Si un paramètre est de type DECIMAL, l'autre paramètre doit également être un type DECIMAL. Si un paramètre est un INTEGER, l'autre paramètre peut être un INTEGER, SMALLINT ou BIGINT. Les deux paramètres peuvent également être SMALLINT ou BIGINT, mais un paramètre ne peut pas être un SMALLINT si l'autre est un BIGINT.

number2

Le second paramètre est un nombre INTEGER, SMALLINT, BIGINT ou DECIMAL. Les mêmes règles de type de données s’appliquent à number2 en ce qui concerne number1.

Type de retour

Les types de retour valides sont DECIMAL, INT, SMALLINT et BIGINT. Le type de retour de la fonction MOD est le même type numérique que les paramètres d'entrée, si les deux paramètres d'entrée sont de même type. Si un paramètre d'entrée est un INTEGER, toutefois, le type de retour sera également un INTEGER.

Notes d'utilisation

Vous pouvez utiliser % comme opérateur modulo.

Exemples

L'exemple suivant renvoie le reste lorsqu'un nombre est divisé par un autre :

```
SELECT MOD(10, 4);

 mod
------
 2
```

L'exemple suivant renvoie un résultat décimal :

```
SELECT MOD(10.5, 4);

 mod
------
 2.5
```

Vous pouvez projeter les valeurs des paramètres :

```
SELECT MOD(CAST(16.4 as integer), 5);

 mod
------
 1
```

Vérifiez si le premier paramètre est pair en le divisant par 2 :

```
SELECT mod(5,2) = 0 as is_even;
```

```
 is_even
--------
 false
```

Vous pouvez utiliser le % comme opérateur modulo :

```
SELECT 11 % 4 as remainder;

 remainder
-----------
 3
```

L'exemple suivant renvoie des informations pour les catégories impaires dans la table CATEGORY :

```
select catid, catname
from category
where mod(catid,2)=1
order by 1,2;

 catid |  catname
-------+-----------
     1 | MLB
     3 | NFL
     5 | MLS
     7 | Plays
     9 | Pop
    11 | Classical

(6 rows)
```

Fonction PI

La fonction PI renvoie la valeur de pi à 14 décimales.

Syntaxe

```
PI()
```

Type de retour

`DOUBLE PRECISION`

Exemples

Pour renvoyer la valeur de pi, utilisez l'exemple suivant.

```
SELECT PI();

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

Fonction POWER

La fonction POWER est une fonction exponentielle qui élève une expression numérique à la puissance d'une seconde expression numérique. Par exemple, 2 à la puissance 3 est calculé sous la forme `POWER(2,3)`, avec un résultat de 8.

Syntaxe

```
{POWER(expression1, expression2)
```

Arguments

expression1

Expression numérique à élever. Doit avoir le type de données `INTEGER`, `DECIMAL` ou `FLOAT`.

expression2

Puissance à laquelle élever expression1. Doit avoir le type de données `INTEGER`, `DECIMAL` ou `FLOAT`.

Type de retour

`DOUBLE PRECISION`

Exemple

```
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 |
+-------------------+
```

Fonction RADIANS

La fonction RADIANS convertit un angle en degrés en son équivalent en radians.

Syntaxe

```
RADIANS(number)
```

Argument

number

Le paramètre d'entrée est un nombre `DOUBLE PRECISION`.

Type de retour

`DOUBLE PRECISION`

Exemple

Pour renvoyer l'équivalent en radians de 180 degrés, utilisez l'exemple suivant.

```
SELECT RADIANS(180);

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

Fonction RAND

La fonction RAND génère un nombre aléatoire à virgule flottante compris entre 0 et 1. La fonction RAND génère un nouveau nombre aléatoire à chaque fois qu'elle est appelée.

Syntaxe

```
RAND()
```

Type de retour

RANDOM renvoie un DOUBLE.

Exemple

L'exemple suivant génère une colonne de nombres aléatoires à virgule flottante compris entre 0 et 1 pour chaque ligne du tableau. `squirrels` Le résultat obtenu serait une colonne unique contenant une liste de valeurs décimales aléatoires, avec une valeur pour chaque ligne de la table des écureuils.

```
SELECT rand() FROM squirrels
```

Ce type de requête est utile lorsque vous devez générer des nombres aléatoires, par exemple pour simuler des événements aléatoires ou pour introduire le caractère aléatoire dans votre analyse de données. Dans le contexte du `squirrels` tableau, il peut être utilisé pour attribuer des valeurs aléatoires à chaque écureuil, qui pourraient ensuite être utilisées pour un traitement ou une analyse plus poussés.

Fonction RANDOM

La fonction RANDOM génère une valeur aléatoire compris entre 0,0 (inclus) et 1,0 (exclusif).

Syntaxe

```
RANDOM()
```

Type de retour

RANDOM renvoie un nombre DOUBLE PRECISION.

Exemples

1. Calculez une valeur aléatoire comprise entre 0 et 99. Si le nombre aléatoire est de 0 à 1, cette requête génère un nombre aléatoire de 0 à 100 :

```
select cast (random() * 100 as int);

INTEGER
------
24
(1 row)
```

2. Récupère un échantillon aléatoire uniforme de 10 éléments :

```
select *
from sales
order by random()
limit 10;
```

   Maintenant, récupérez un échantillon aléatoire de 10 éléments, mais choisissez les éléments en fonction de leur prix. Par exemple, un élément dont le prix est le double d'un autre a deux fois plus de chance d'apparaître dans les résultats de la requête :

```
select *
from sales
order by log(1 - random()) / pricepaid
limit 10;
```

3. Cet exemple utilise la commande SET pour définir une valeur SEED afin que RANDOM génère une séquence de nombres prévisible.

   D’abord, renvoyez trois entiers RANDOM sans définir au préalable la valeur SEED :

```
select cast (random() * 100 as int);
INTEGER
------
6
(1 row)

select cast (random() * 100 as int);
INTEGER
------
68
(1 row)

select cast (random() * 100 as int);
```

```
INTEGER
------
56
(1 row)
```

A présent, définissez la valeur SEED sur `.25` et renvoyez trois nombres RANDOM supplémentaires :

```
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)
```

Enfin, réinitialisez la valeur SEED sur `.25` et vérifiez que RANDOM renvoie les mêmes résultats que les trois appels précédents :

```
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)
```

Fonction ROUND

La fonction ROUND arrondit des nombres à l'entier ou à la décimale la plus proche.

La fonction ROUND peut éventuellement inclure un second argument sous forme de nombre entier permettant d'indiquer le nombre de décimales de l'arrondi, dans les deux sens. Lorsque vous ne fournissez pas le second argument, la fonction arrondit au nombre entier le plus proche. Lorsque le second argument >n est spécifié, la fonction arrondit au nombre le plus proche avec une précision de n décimales.

Syntaxe

```
ROUND (number [ , integer ] )
```

Argument

number

Nombre ou expression ayant pour valeur un nombre. Il peut s'agir du DECIMAL ou du FLOAT8 type. AWS Clean Rooms peut convertir d'autres types de données selon les règles de conversion implicites.

integer (facultatif)

Nombre entier qui indique le nombre de décimales pour l'arrondi dans les deux sens.

Type de retour

ROUND renvoie le même type de données numériques en tant qu'argument(s) d'entrée.

Exemples

Arrondit la commission payée pour une transaction donnée au nombre entier le plus proche.

```
select commission, round(commission)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    28
(1 row)
```

Arrondit la commission payée pour une transaction donnée à la première décimale.

```
select commission, round(commission, 1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |  28.1
(1 row)
```

Pour la même requête, étendez la précision dans l'autre sens.

```
select commission, round(commission, -1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    30
(1 row)
```

Fonction SIGN

La fonction SIGN renvoie le signe (positif ou négatif) d’un nombre. Le résultat de la fonction SIGN est `1`, `-1` ou `0`, ce qui indique le signe de l'argument.

Syntaxe

```
SIGN (number)
```

Argument

number

Nombre ou expression ayant pour valeur un nombre. C'est peut-être le DECIMALor FLOAT8 genre. AWS Clean Rooms peut convertir d'autres types de données selon les règles de conversion implicites.

Type de retour

SIGN renvoie le même type de données numériques en tant qu'argument(s) d'entrée. Si l'entrée est DECIMAL, la sortie est DECIMAL(1,0).

Exemples

Pour déterminer le signe de la commission payée pour une transaction donnée à partir de la table SALES, utilisez l'exemple suivant.

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

Fonction SIN

SIN est une fonction trigonométrique qui renvoie le sinus d'un nombre. La valeur renvoyée est comprise entre -1 et 1.

Syntaxe

```
SIN(number)
```

Argument

number

Nombre DOUBLE PRECISION en radians.

Type de retour

DOUBLE PRECISION

Exemple

Pour renvoyer le sinus de -PI, utilisez l'exemple suivant.

```
SELECT SIN(-PI());

+-------------------------+
|           sin           |
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

Fonction SQRT

La fonction SQRT renvoie la racine carrée d'une valeur numérique. La racine carrée est un nombre multiplié par lui-même pour obtenir la valeur donnée.

Syntaxe

```
SQRT (expression)
```

Argument

expression

L'expression doit comporter un type de données de nombre entier, décimale ou à virgule flottante. L'expression peut inclure des fonctions. Le système peut effectuer des conversions de type implicites.

Type de retour

SQRT renvoie un nombre DOUBLE PRECISION.

Exemples

L'exemple suivant renvoie la racine carrée d'un nombre.

```
select sqrt(16);
```

```
sqrt
---------------
4
```

L'exemple suivant effectue une conversion de type implicite.

```
select sqrt('16');

sqrt
---------------
4
```

L'exemple suivant imbrique des fonctions pour effectuer une tâche plus complexe.

```
select sqrt(round(16.4));

sqrt
---------------
4
```

L'exemple suivant donne la longueur du rayon lorsque l'aire du cercle est indiquée. Il calcule le rayon en pouces, par exemple, lorsque la surface est indiquée en pouces carrés. Dans l'exemple, l'aire est de 20.

```
select sqrt(20/pi());
```

La valeur renvoyée est 5.046265044040321.

L'exemple suivant renvoie la racine carrée des valeurs de COMMISSION de la table SALES. La colonne COMMISSION est une colonne DECIMAL. Cet exemple montre comment utiliser la fonction dans une requête ayant une logique conditionnelle plus complexe.

```
select sqrt(commission)
from sales where salesid < 10 order by salesid;

sqrt
------------------
10.4498803820905
3.37638860322683
7.24568837309472
```

```
5.1234753829798
...
```

La requête suivante renvoie la racine carré arrondie du même ensemble de valeurs COMMISSION.

```
select salesid, commission, round(sqrt(commission))
from sales where salesid < 10 order by salesid;

salesid | commission | round
--------+------------+-------
      1 |     109.20 |    10
      2 |      11.40 |     3
      3 |      52.50 |     7
      4 |      26.25 |     5
...
```

Pour plus d'informations sur les exemples de données dans AWS Clean Rooms, consultez la section Exemple de base de données.

Fonction TRUNC

La fonction TRUNC tronque les nombres à l'entier ou à la décimale précédente.

La fonction TRUNC peut éventuellement inclure un second argument : un nombre entier permettant d'indiquer le nombre de décimales de l'arrondi, dans les deux sens. Lorsque vous ne fournissez pas le second argument, la fonction arrondit au nombre entier le plus proche. Lorsque le second argument >n est spécifié, la fonction arrondit au nombre le plus proche avec une précision de n décimales. Cette fonction tronque également un horodatage et renvoie une date.

Syntaxe

```
TRUNC (number [ , integer ] |
timestamp )
```

Arguments

number

Nombre ou expression ayant pour valeur un nombre. Il peut s'agir du DECIMAL ou du FLOAT8 type. AWS Clean Rooms peut convertir d'autres types de données selon les règles de conversion implicites.

integer (facultatif)

Nombre entier qui indique le nombre de décimales de précision, dans les deux sens. Si aucun nombre entier n'est fourni, le nombre est tronqué en tant que nombre entier ; si un nombre entier est spécifié, le nombre est tronqué à la décimale spécifiée.

timestamp

La fonction peut également renvoyer la date à partir d'un horodatage. (Pour renvoyer une valeur d'horodatage avec `00:00:00` comme heure, envoyez le résultat de la fonction à un horodatage.)

Type de retour

TRUNC renvoie le même type de données que le premier argument d'entrée. Pour les horodatages, TRUNC renvoie une date.

Exemples

Tronque la commission payée pour une transaction de vente donnée.

```
select commission, trunc(commission)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   111

(1 row)
```

Tronque la même valeur de commission que la première décimale.

```
select commission, trunc(commission,1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 | 111.1

(1 row)
```

Tronque la commission avec une valeur négative pour le second argument ; `111.15` est arrondi à `110`.

```
select commission, trunc(commission,-1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   110
(1 row)
```

Renvoyez la partie de date du résultat de la fonction SYSDATE (qui renvoie un horodatage) :

```
select sysdate;

timestamp
----------------------------
2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc
------------
2011-07-21
(1 row)
```

Appliquez la fonction TRUNC à une colonne TIMESTAMP. Le type de retour est une date.

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

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

Fonctions scalaires

Cette section décrit les fonctions scalaires prises en charge dans AWS Clean Rooms Spark SQL. Une fonction scalaire est une fonction qui prend une ou plusieurs valeurs en entrée et renvoie une seule valeur en sortie. Les fonctions scalaires fonctionnent sur des lignes ou des éléments individuels et produisent un résultat unique pour chaque entrée.

Les fonctions scalaires, telles que SIZE, sont différentes des autres types de fonctions SQL, telles que les fonctions d'agrégation (count, sum, avg) et les fonctions de génération de tables (explode, aplten). Ces autres types de fonctions fonctionnent sur plusieurs lignes ou génèrent plusieurs lignes, tandis que les fonctions scalaires fonctionnent sur des lignes ou des éléments individuels.

Rubriques

fonction SIZE

La fonction SIZE prend un tableau, une carte ou une chaîne existant comme argument et renvoie une valeur unique représentant la taille ou la longueur de cette structure de données. Cela ne crée pas de nouvelle structure de données. Il est utilisé pour interroger et analyser les propriétés des structures de données existantes, plutôt que pour en créer de nouvelles.

Cette fonction est utile pour déterminer le nombre d'éléments d'un tableau ou la longueur d'une chaîne. Cela peut être particulièrement utile lorsque vous travaillez avec des tableaux et d'autres structures de données en SQL, car cela vous permet d'obtenir des informations sur la taille ou la cardinalité des données.

Syntaxe

```
size(expr)
```

Arguments

expr

Expression ARRAY, MAP ou STRING.

Type de retour

La fonction SIZE renvoie un INTEGER.

exemple

Dans cet exemple, la fonction SIZE est appliquée au tableau `['b', 'd', 'c', 'a']` et renvoie la valeur4, qui est le nombre d'éléments du tableau.

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

Dans cet exemple, la fonction SIZE est appliquée à la carte `{'a': 1, 'b': 2}` et renvoie la valeur2, qui est le nombre de paires clé-valeur sur la carte.

```
SELECT size(map('a', 1, 'b', 2));
 2
```

Dans cet exemple, la fonction SIZE est appliquée à la chaîne `'hello world'` et renvoie la valeur11, qui est le nombre de caractères de la chaîne.

```
SELECT size('hello world');
11
```

Fonctions de chaîne

Fonctions de chaîne qui traitent et manipulent des chaînes de caractères ou des expressions qui correspondent à des chaînes de caractères. Lorsque l'argument string de ces fonctions est une valeur littérale, il doit être entre guillemets simples. Les types de données pris en charge sont CHAR et VARCHAR.

La section suivante fournit les noms de fonctions, la syntaxe et les descriptions des fonctions prises en charge. Tous les décalages en chaînes sont basés sur un.

Rubriques

- Fonction POSITION
- Fonction REGEXP_COUNT
- Fonction REGEXP_INSTR
- Fonction REGEXP_REPLACE
- Fonction REGEXP_SUBSTR
- Fonction REPEAT
- Fonction REPLACE
- Fonction REVERSE
- Fonction RTRIM
- Fonction SPLIT
- Fonction SPLIT_PART
- Fonction SUBSTRING
- Fonction TRANSLATE
- Fonction TRIM
- Fonction UPPER
- Fonction UUID

Opérateur (concaténation) ||

Concatène deux expressions de chaque côté du symbole || et renvoie l'expression concaténée.

L'opérateur de concatentation est similaire à. Fonction CONCAT

> **Note**
>
> Pour la fonction CONCAT et l'opérateur de concaténation, si une expression ou les deux ont la valeur null, le résultat de la concaténation est null.

Syntaxe

```
expression1 || expression2
```

Arguments

expression1, expression2

Les deux arguments peuvent être des chaînes de caractères de longueur fixe ou de longueur variable ou des expressions.

Type de retour

L'opérateur || renvoie une chaîne. Le type de chaîne est identique à celui des arguments d'entrée.

exemple

L'exemple suivant concatène les champs FIRSTNAME et LASTNAME de la table USERS :

```
select firstname || ' ' || lastname
from users
order by 1
limit 10;

concat
-----------------
Aaron Banks
Aaron Booth
Aaron Browning
Aaron Burnett
Aaron Casey
Aaron Cash
Aaron Castro
Aaron Dickerson
Aaron Dixon
Aaron Dotson
(10 rows)
```

Pour concaténer des colonnes susceptibles de contenir des valeurs nulles, utilisez l'expression Fonctions NVL et COALESCE. L'exemple suivant utilise NVL pour renvoyer un 0 chaque fois que la valeur NULL est rencontrée.

```
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
```

Fonction BTRIM

La fonction BTRIM tronque une chaîne en supprimant les espaces de début et de fin ou en supprimant les caractères de début et de fin qui correspondent à une chaîne spécifiée de manière facultative.

Syntaxe

```
BTRIM(string [, trim_chars ] )
```

Arguments

string

Chaîne VARCHAR d'entrée à tronquer.

trim_chars

Chaîne VARCHAR contenant les caractères à mettre en correspondance.

Type de retour

La fonction BTRIM renvoie une chaîne VARCHAR.

Exemples

L'exemple suivant tronque les espaces de début et de fin de la chaîne `' abc '` :

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

L'exemple suivant supprime les chaînes `'xyz'` de début et de fin de la chaîne `'xyzaxyzbxyzcxyz'`. Les occurrences de début et de fin de `'xyz'` sont supprimées, mais celles qui se trouvent à l'intérieur de la chaîne sont conservées.

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

L'exemple suivant supprime les parties de début et de fin de la chaîne `'setuphistorycassettes'` qui correspondent à l'un des caractères de la liste trim_chars `'tes'`. Tout caractère `t`, `e` ou `s` précédant un autre caractère qui ne figure pas dans la liste trim_chars au début ou à la fin de la chaîne d'entrée est supprimé.

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
----------------
 uphistoryca
```

Fonction CONCAT

La fonction CONCAT concatène deux expressions et renvoie l'expression résultante. Pour concaténer plus de deux expressions, utilisez les fonction CONCAT imbriquées. L'opérateur de concaténation (`||`) entre deux expressions donne les mêmes résultats que la fonction CONCAT.

> **Note**
>
> Pour la fonction CONCAT et l'opérateur de concaténation, si une expression ou les deux ont la valeur null, le résultat de la concaténation est null.

Syntaxe

```
CONCAT ( expression1, expression2 )
```

Arguments

expression1, expression2

Les deux arguments peuvent être une chaîne de caractères de longueur fixe, une chaîne de caractères de longueur variable, une expression binaire ou une expression qui a pour résultat l'une de ces entrées.

Type de retour

CONCAT renvoie une expression. Le type de données de l'expression est le même que celui des arguments d'entrée.

Si les expressions d'entrée sont de types différents, AWS Clean Rooms essaie de convertir implicitement l'une des expressions. Si des valeurs ne peuvent pas être converties, une erreur est renvoyée.

Exemples

L'exemple suivant concatène deux littéraux caractères :

```
select concat('December 25, ', '2008');

concat
------------------
December 25, 2008
(1 row)
```

La requête suivante, utilisant l'opérateur `||` au lieu de CONCAT, produit le même résultat :

```
select 'December 25, '||'2008';

concat
------------------
December 25, 2008
(1 row)
```

L'exemple suivant illustre l'utilisation des fonctions CONCAT pour concaténer trois chaînes de caractères :

```
select concat('Thursday, ', concat('December 25, ', '2008'));

concat
----------------------------
Thursday, December 25, 2008
(1 row)
```

Pour concaténer des colonnes susceptibles de contenir des valeurs nulles, utilisez la fonction Fonctions NVL et COALESCE. L'exemple suivant utilise NVL pour renvoyer un 0 chaque fois que la valeur NULL est rencontrée.

```
select concat(venuename, concat(' seats ', nvl(venueseats, 0))) as seating
from venue where venuestate = 'NV' or venuestate = 'NC'
order by 1
limit 5;

seating
-----------------------------------
Ballys Hotel seats 0
Bank of America Stadium seats 73298
Bellagio Hotel seats 0
Caesars Palace seats 0
Harrahs Hotel seats 0
(5 rows)
```

La requête suivante concatène les valeurs CITY et STATE de la table VENUE :

```
select concat(venuecity, venuestate)
from venue
where venueseats > 75000
order by venueseats;

concat
-------------------
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
```

```
(4 rows)
```

La requête suivante utilise des fonctions CONCAT imbriquées. La requête concatène les valeurs CITY et STATE de la table VENUE, mais délimite la chaîne qui en résulte par une virgule et un espace :

```
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)
```

Fonction FORMAT_STRING

La fonction FORMAT_STRING crée une chaîne formatée en remplaçant les espaces réservés dans une chaîne modèle par les arguments fournis. Elle renvoie une chaîne formatée à partir de chaînes de format de style printf.

La fonction FORMAT_STRING fonctionne en remplaçant les espaces réservés dans la chaîne du modèle par les valeurs correspondantes passées en arguments. Ce type de formatage de chaîne peut être utile lorsque vous devez créer dynamiquement des chaînes qui incluent un mélange de texte statique et de données dynamiques, par exemple lors de la génération de messages de sortie, de rapports ou d'autres types de texte informatif. La fonction FORMAT_STRING fournit un moyen concis et lisible de créer ces types de chaînes formatées, ce qui facilite la maintenance et la mise à jour du code qui génère le résultat.

Syntaxe

```
format_string(strfmt, obj, ...)
```

Arguments

strfmt

Expression STRING.

obj

Une chaîne ou une expression numérique.

Type de retour

FORMAT_STRING renvoie une chaîne.

exemple

L'exemple suivant contient un modèle de chaîne contenant deux espaces réservés : `%d` pour une valeur décimale (entier) et `%s` pour une valeur de chaîne. L'`%d`espace réservé est remplacé par la valeur décimale (entier) (`100`), et l'espace réservé %s est remplacé par la valeur de chaîne (). `"days"` La sortie est une chaîne modèle dont les espaces réservés sont remplacés par les arguments fournis :`"Hello World 100 days"`.

```
SELECT format_string("Hello World %d %s", 100, "days");
 Hello World 100 days
```

Fonctions LEFT et RIGHT

Ces fonctions renvoient le nombre de caractères spécifié le plus à gauche ou le plus à droite dans une chaîne de caractères.

Le chiffre est basé sur le nombre de caractères, pas d'octets, de sorte que les caractères à plusieurs octets soient comptés comme des caractères seuls.

Syntaxe

```
LEFT ( string,  integer )

RIGHT ( string,  integer )
```

Arguments

string

Chaîne de caractères ou expression qui a pour valeur une chaîne de caractères.

integer

Nombre entier positif.

Type de retour

LEFT et RIGHT renvoient une chaîne VARCHAR.

exemple

L'exemple suivant renvoie les 5 caractères les plus à gauche et les 5 caractères les plus à droite à partir de noms d'événements IDs compris entre 1 000 et 1 005 :

```
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)
```

Fonction LENGTH

Fonction LOWER

Convertit une chaîne en minuscules. LOWER prend en charge les caractères à plusieurs octets UTF-8, à concurrence de quatre octets au maximum par caractère.

Syntaxe

```
LOWER(string)
```

Argument

string

Le paramètre d'entrée est une chaîne VARCHAR (ou tout autre type de données, tel que CHAR, qui peut être implicitement converti en VARCHAR).

Type de retour

La fonction LOWER renvoie une chaîne de caractères qui est du même type que la chaîne d'entrée.

Exemples

L'exemple suivant convertit le champ CATNAME en minuscules :

```
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)
```

Fonctions LPAD et RPAD

Ces fonctions ajoutent des caractères en préfixe ou en suffixe à une chaîne, en fonction d’une longueur spécifiée.

Syntaxe

```
LPAD (string1, length, [ string2 ])
```

```
RPAD (string1, length, [ string2 ])
```

Arguments

string1

Chaîne de caractères ou expression qui a pour valeur une chaîne de caractères, comme le nom d'une colonne de caractères.

longueur

Nombre entier qui définit la longueur du résultat de la fonction. La longueur d’une chaîne est basée sur le nombre de caractères, pas d’octets, afin que les caractères à plusieurs octets soient comptés comme des caractères seuls. Si string1 dépasse la longueur spécifiée, il est tronqué (à droite). Si length est un nombre négatif, le résultat de la fonction est une chaîne vide.

string2

Un ou plusieurs caractères ajoutés en préfixe ou en suffixe à string1. Cet argument est facultatif. S'il n'est pas spécifié, les espaces sont utilisés.

Type de retour

Ces fonctions renvoient un type de données VARCHAR.

Exemples

Tronquez un ensemble spécifié de noms d'événements à 20 caractères et ajoutez des espaces comme préfixes aux noms plus courts :

```
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)
```

Tronquez le même ensemble de noms d'événements à 20 caractères, mais ajoutez `0123456789` comme suffixe aux noms plus courts.

```
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)
```

Fonction LTRIM

Supprime les caractères du début d'une chaîne de caractères. Supprime la chaîne la plus longue ne contenant que des caractères de la liste des caractères supprimés. Le découpage est terminé lorsqu'aucun caractère de découpage n'apparaît dans la chaîne d'entrée.

Syntaxe

```
LTRIM( string [, trim_chars] )
```

Arguments

string

Une colonne de chaîne, une expression ou un littéral de chaîne à supprimer.

trim_chars

Une colonne, une expression ou un littéral de chaîne qui représente les caractères à supprimer au début de la chaîne. Si la valeur n'est pas spécifiée, un espace est utilisé comme caractère de séparation.

Type de retour

La fonction LTRIM renvoie une chaîne de caractères qui est du même type que la chaîne d'entrée (CHAR ou VARCHAR).

Exemples

L'exemple suivant supprime l'année de la colonne `listime`. Les caractères supprimés dans la chaîne littérale `'2008-'` indiquent les caractères à supprimer à partir de la gauche. Si vous utilisez les caractères de suppression `'028-'`, vous obtiendrez le même résultat.

```
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 supprime les caractères de trim_chars lorsqu'ils apparaissent au début de la chaîne. L'exemple suivant supprime les caractères C, D et G lorsqu'ils figurent au début de VENUENAME, qui est une colonne VARCHAR.

```
select venueid, venuename, ltrim(venuename, 'CDG')
from venue
where venuename like '%Park'
order by 2
limit 7;

venueid | venuename                  | btrim
--------+----------------------------+--------------------------
    121 | ATT Park                   | ATT Park
    109 | Citizens Bank Park         | itizens Bank Park
```

```
    102 | Comerica Park              | omerica Park
      9 | Dick's Sporting Goods Park | ick's Sporting Goods Park
     97 | Fenway Park                | Fenway Park
    112 | Great American Ball Park   | reat American Ball Park
    114 | Miller Park                | Miller Park
```

L'exemple suivant utilise le caractère de suppression 2 qui est extrait de la colonne `venueid`.

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

L'exemple suivant ne supprime aucun caractère car 2 est trouvé avant le caractère de suppression `'0'`.

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

L'exemple suivant utilise le caractère de suppression d'espace par défaut et supprime les deux espaces du début de la chaîne.

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

Fonction POSITION

Renvoie l'emplacement de la sous-chaîne spécifiée dans une chaîne.

Syntaxe

```
POSITION(substring IN string )
```

Arguments

substring

Sous-chaîne à rechercher dans la chaîne.

string

Chaîne ou colonne à rechercher.

Type de retour

La fonction POSITION renvoie un nombre entier correspondant à la position de la sous-chaîne (base 1, pas base 0). La position est basée sur le nombre de caractères, pas d'octets, de sorte que les caractères à plusieurs octets soient comptés comme des caractères seuls.

Notes d'utilisation

POSITION renvoie 0 si la sous-chaîne n'est pas trouvée dans la chaîne POSITION :

```
select position('dog' in 'fish');

position
----------
 0
(1 row)
```

Exemples

L'exemple suivant montre la position de la chaîne `fish` dans le mot `dogfish` :

```
select position('fish' in 'dogfish');

position
----------
 4
(1 row)
```

L'exemple suivant renvoie le nombre de transactions commerciales avec une COMMISSION de plus de 999,00 dans la table SALES :

```
select distinct position('.' in commission), count (position('.' in commission))
```

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

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

Fonction REGEXP_COUNT

Recherche un modèle d'expression régulière dans une chaîne et renvoie un nombre entier indiquant le nombre de fois où le modèle est présent dans la chaîne. Si aucune correspondance n'est trouvée, la fonction renvoie 0.

Syntaxe

```
REGEXP_COUNT ( source_string, pattern [, position [, parameters ] ] )
```

Arguments

source_string

Expression de chaîne, comme un nom de colonne, à rechercher.

pattern

Chaîne littérale qui représente un modèle d'expression régulière.

position

Nombre entier positif qui indique à quel endroit de source_string commencer la recherche. La position est basée sur le nombre de caractères, pas d'octets, de sorte que les caractères à plusieurs octets soient comptés comme des caractères seuls. La valeur par défaut est 1. Si position est inférieur à 1, la recherche commence au premier caractère de source_string. Si position est supérieur au nombre de caractères de source_string, le résultat est 0.

parameters

Un ou plusieurs littéraux de chaîne qui indiquent comment la fonction correspond au modèle. Les valeurs possibles sont les suivantes :

- c : réaliser une correspondance avec respect de la casse. Par défaut, la correspondance avec respect de la casse est utilisée.

- i : réaliser une correspondance avec non-respect de la casse.
- p – Interpréter le modèle avec le type d'expression PCRE (Perl Compatible Regular Expression).

Type de retour

Entier

exemple

L'exemple suivant compte le nombre de fois que se produit une séquence de trois lettres.

```
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');

 regexp_count
 --------------
            8
```

L'exemple suivant compte le nombre de fois que le nom de domaine de niveau supérieur est `org` ou `edu`.

```
SELECT email, regexp_count(email,'@[^.]*\\.(org|edu)')FROM users
ORDER BY userid LIMIT 4;

                      email                      | regexp_count
-------------------------------------------------+--------------
 Etiam.laoreet.libero@sodalesMaurisblandit.edu |            1
 Suspendisse.tristique@nonnisiAenean.edu       |            1
 amet.faucibus.ut@condimentumegetvolutpat.ca   |            0
 sed@lacusUtnec.ca                             |            0
```

L'exemple suivant compte les occurrences de la chaîne `FOX`, en utilisant une correspondance avec respect de la casse.

```
SELECT regexp_count('the fox', 'FOX', 1, 'i');

 regexp_count
 --------------
            1
```

L'exemple suivant utilise un modèle écrit dans le type PCRE pour localiser des mots contenant au moins un chiffre et une lettre minuscule. Il utilise l'opérateur ?=, qui a une connotation « anticipée » spécifique au type PCRE. Cet exemple compte le nombre d'occurrences de ces mots, avec une correspondance avec respect de la casse.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

 regexp_count
 --------------
            2
```

L'exemple suivant utilise un modèle écrit dans le type PCRE pour localiser des mots contenant au moins un chiffre et une lettre minuscule. Il utilise l'opérateur ?=, qui a une connotation spécifique au type PCRE. Cet exemple compte le nombre d'occurrences de ces mots, mais diffère de l'exemple précédent car il utilise une correspondance avec non-respect de la casse.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');

 regexp_count
 --------------
            3
```

Fonction REGEXP_INSTR

Recherche un modèle d’expression régulière dans une chaîne et renvoie un nombre entier qui indique la position de début de la sous-chaîne correspondante. Si aucune correspondance n'est trouvée, la fonction renvoie 0. REGEXP_SUBSTR est similaire à la fonction POSITION, mais vous permet de rechercher un modèle d’expression régulière dans une chaîne.

Syntaxe

```
REGEXP_INSTR ( source_string, pattern [, position [, occurrence] [, option
 [, parameters ] ] ] ] )
```

Arguments

source_string

Expression de chaîne, comme un nom de colonne, à rechercher.

pattern

Chaîne littérale qui représente un modèle d'expression régulière.

position

Nombre entier positif qui indique à quel endroit de source_string commencer la recherche. La position est basée sur le nombre de caractères, pas d'octets, de sorte que les caractères à plusieurs octets soient comptés comme des caractères seuls. La valeur par défaut est 1. Si position est inférieur à 1, la recherche commence au premier caractère de source_string. Si position est supérieur au nombre de caractères de source_string, le résultat est 0.

occurrence

Nombre entier positif qui indique quelle occurrence du modèle utiliser. REGEXP_INSTR ignore les occurrence -1 premières correspondances. La valeur par défaut est 1. Si occurrence est inférieur à 1 ou supérieur au nombre de caractères de la chaîne source_string, la recherche est ignorée et le résultat est 0.

option

Valeur qui indique s'il faut renvoyer la position du premier caractère de la correspondance (`0`) ou celle du premier caractère après la fin de la correspondance (1). Toute valeur de chaîne autre que zéro est similaire à la valeur 1. La valeur par défaut est 0.

parameters

Un ou plusieurs littéraux de chaîne qui indiquent comment la fonction correspond au modèle. Les valeurs possibles sont les suivantes :

- c : réaliser une correspondance avec respect de la casse. Par défaut, la correspondance avec respect de la casse est utilisée.
- i : réaliser une correspondance avec non-respect de la casse.
- e : extraire une sous-chaîne à l'aide d'une sous-expression.

  Si pattern inclut une sous-expression, REGEXP_INSTR met en correspondance une sous-chaîne à l'aide de la première sous-expression incluse dans pattern. REGEXP_INSTR considère uniquement la première sous-expression ; les autres sous-expressions sont ignorées. Si le modèle n'inclut pas de sous-expression, REGEXP_INSTR ignore le paramètre « e ».
- p – Interpréter le modèle avec le type d'expression PCRE (Perl Compatible Regular Expression).

Type de retour

Entier

exemple

L'exemple suivant recherche le caractère @ qui commence par un nom de domaine et renvoie la position de début de la première correspondance.

```
SELECT email, regexp_instr(email, '@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

                     email                      | regexp_instr
------------------------------------------------+--------------
 Etiam.laoreet.libero@example.com |           21
 Suspendisse.tristique@nonnisiAenean.edu       |           22
 amet.faucibus.ut@condimentumegetvolutpat.ca   |           17
 sed@lacusUtnec.ca                             |            4
```

L'exemple suivant recherche des variantes du mot `Center` et renvoie la position du début de la première correspondance.

```
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
```

L'exemple suivant recherche la position de départ de la première occurrence de la chaîne FOX, à l'aide d'une logique de correspondance avec respect de la casse.

```
SELECT regexp_instr('the fox', 'FOX', 1, 1, 0, 'i');

 regexp_instr
```

```
--------------
           5
```

L'exemple suivant utilise un modèle écrit en PCRE pour localiser des mots contenant au moins un chiffre et une lettre minuscule. Il utilise l'opérateur ?=, qui a une connotation « anticipée » spécifique au type PCRE. Cet exemple montre comment trouver la position de départ du deuxième mot de ce type.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

 regexp_instr
 --------------
           21
```

L'exemple suivant utilise un modèle écrit en PCRE pour localiser des mots contenant au moins un chiffre et une lettre minuscule. Il utilise l'opérateur ?=, qui a une connotation « anticipée » spécifique au type PCRE. Cet exemple recherche la position de départ du deuxième mot de ce type, mais diffère de l'exemple précédent car il utilise une correspondance avec non-respect de la casse.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');

 regexp_instr
 --------------
           15
```

Fonction REGEXP_REPLACE

Recherche un modèle d'expression régulière dans une chaîne et remplace chaque occurrence du modèle par la chaîne spécifiée. REGEXP_REPLACE est similaire à la Fonction REPLACE, mais vous permet de rechercher un modèle d'expression régulière dans une chaîne.

REGEXP_REPLACE est similaire à la Fonction TRANSLATE et la Fonction REPLACE, sauf que TRANSLATE fait plusieurs remplacements de caractère unique et REPLACE remplace une chaîne entière par une autre chaîne, tandis que REGEXP_REPLACE vous permet de rechercher un modèle d'expression régulière dans une chaîne.

Syntaxe

```
REGEXP_REPLACE ( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

Arguments

source_string

Expression de chaîne, comme un nom de colonne, à rechercher.

pattern

Chaîne littérale qui représente un modèle d'expression régulière.

replace_string

Expression de chaîne, comme un nom de colonne, qui va remplacer chaque occurrence de modèle. La valeur par défaut est une chaîne vide ("").

position

Nombre entier positif qui indique à quel endroit de source_string commencer la recherche. La position est basée sur le nombre de caractères, pas d'octets, de sorte que les caractères à plusieurs octets soient comptés comme des caractères seuls. La valeur par défaut est 1. Si position est inférieur à 1, la recherche commence au premier caractère de source_string. Si position est supérieure au nombre de caractères de source_string, le résultat est source_string.

parameters

Un ou plusieurs littéraux de chaîne qui indiquent comment la fonction correspond au modèle. Les valeurs possibles sont les suivantes :

- c : réaliser une correspondance avec respect de la casse. Par défaut, la correspondance avec respect de la casse est utilisée.
- i : réaliser une correspondance avec non-respect de la casse.
- p – Interpréter le modèle avec le type d'expression PCRE (Perl Compatible Regular Expression).

Type de retour

VARCHAR

Si pattern ou replace_string a la valeur NULL, le retour est NULL.

exemple

L'exemple suivant supprime le caractère @ et le nom de domaine des adresses e-mail.

```
SELECT email, regexp_replace(email, '@.*\\.(org|gov|com|edu|ca)$')
FROM users
ORDER BY userid LIMIT 4;

              email                              | regexp_replace
-------------------------------------------------+----------------
 Etiam.laoreet.libero@sodalesMaurisblandit.edu | Etiam.laoreet.libero
 Suspendisse.tristique@nonnisiAenean.edu       | Suspendisse.tristique
 amet.faucibus.ut@condimentumegetvolutpat.ca   | amet.faucibus.ut
 sed@lacusUtnec.ca                             | sed
```

L'exemple suivant remplace les noms de domaine des adresses e-mail par cette valeur : `internal.company.com`.

```
SELECT email, regexp_replace(email, '@.*\\.[[:alpha:]]{2,3}',
'@internal.company.com') FROM users
ORDER BY userid LIMIT 4;

                     email                     |               regexp_replace
-----------------------------------------------
+-----------------------------------------
 Etiam.laoreet.libero@sodalesMaurisblandit.edu |
 Etiam.laoreet.libero@internal.company.com
 Suspendisse.tristique@nonnisiAenean.edu       |
 Suspendisse.tristique@internal.company.com
 amet.faucibus.ut@condimentumegetvolutpat.ca   | amet.faucibus.ut@internal.company.com
 sed@lacusUtnec.ca                             | sed@internal.company.com
```

L'exemple suivant remplace toutes les occurrences de la chaîne FOX dans la valeur `quick brown fox`, à l'aide d'une correspondance avec respect de la casse.

```
SELECT regexp_replace('the fox', 'FOX', 'quick brown fox', 1, 'i');

   regexp_replace
---------------------
```

```
 the quick brown fox
```

L'exemple suivant utilise un modèle écrit dans le type PCRE pour localiser des mots contenant au moins un chiffre et une lettre minuscule. Il utilise l'opérateur ?=, qui a une connotation « anticipée » spécifique au type PCRE. Cet exemple remplace chaque occurrence de mot de ce type par la valeur `[hidden]`.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

       regexp_replace
------------------------------
 [hidden] plain A1234 [hidden]
```

L'exemple suivant utilise un modèle écrit dans le type PCRE pour localiser des mots contenant au moins un chiffre et une lettre minuscule. Il utilise l'opérateur ?=, qui a une connotation « anticipée » spécifique au type PCRE. Cet exemple remplace chaque occurrence de mot de ce type par la valeur `[hidden]`, mais diffère de l'exemple précédent car il utilise une correspondance avec non-respect de la casse.

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

         regexp_replace
--------------------------------
 [hidden] plain [hidden] [hidden]
```

Fonction REGEXP_SUBSTR

Renvoie les caractères d'une chaîne en y recherchant un modèle d'expression régulière. REGEXP_SUBSTR est similaire à la fonction Fonction SUBSTRING, mais vous permet de rechercher un modèle d'expression régulière dans une chaîne. Si la fonction ne trouve pas correspondance entre l'expression régulière et aucun caractère de la chaîne, elle renvoie une chaîne vide.

Syntaxe

```
REGEXP_SUBSTR ( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

Arguments

source_string

Expression de chaîne à rechercher.

pattern

Chaîne littérale qui représente un modèle d'expression régulière.

position

Nombre entier positif qui indique à quel endroit de source_string commencer la recherche. La position est basée sur le nombre de caractères, pas d'octets, de sorte que les caractères à plusieurs octets soient comptés comme des caractères seuls. La valeur par défaut est 1. Si position est inférieur à 1, la recherche commence au premier caractère de source_string. Si position est supérieure au nombre de caractères de source_string, le résultat est une chaîne vide ("").

occurrence

Nombre entier positif qui indique quelle occurrence du modèle utiliser. REGEXP_SUBSTR ignore les occurrence -1 premières correspondances. La valeur par défaut est 1. Si occurrence est inférieur à 1 ou supérieur au nombre de caractères de la chaîne source_string, la recherche est ignorée et le résultat est NULL.

parameters

Un ou plusieurs littéraux de chaîne qui indiquent comment la fonction correspond au modèle. Les valeurs possibles sont les suivantes :

- c : réaliser une correspondance avec respect de la casse. Par défaut, la correspondance avec respect de la casse est utilisée.
- i : réaliser une correspondance avec non-respect de la casse.
- e : extraire une sous-chaîne à l'aide d'une sous-expression.

  Si pattern inclut une sous-expression, REGEXP_SUBSTR met en correspondance une sous-chaîne à l'aide de la première sous-expression incluse dans pattern. Une sous-expression est une expression dans le modèle qui est mise entre parenthèses. Par exemple, le modèle `'This is a (\\w+)'` met en correspondance la première expression avec la chaîne `'This is a '` suivie d'un mot. Au lieu de renvoyer le modèle, REGEXP_SUBSTR avec le paramètre e renvoie uniquement la chaîne contenue dans la sous-expression.

REGEXP_SUBSTR considère uniquement la première sous-expression ; les autres sous-expressions sont ignorées. Si le modèle n'inclut pas de sous-expression, REGEXP_SUBSTR ignore le paramètre « e ».

- p – Interpréter le modèle avec le type d'expression PCRE (Perl Compatible Regular Expression).

Type de retour

VARCHAR

exemple

L'exemple suivant renvoie la partie d'une adresse e-mail comprise entre le caractère @ et l'extension du domaine.

```
SELECT email, regexp_substr(email,'@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

                     email                      |      regexp_substr
------------------------------------------------+--------------------------
 Etiam.laoreet.libero@sodalesMaurisblandit.edu | @sodalesMaurisblandit
 Suspendisse.tristique@nonnisiAenean.edu       | @nonnisiAenean
 amet.faucibus.ut@condimentumegetvolutpat.ca   | @condimentumegetvolutpat
 sed@lacusUtnec.ca                             | @lacusUtnec
```

L'exemple suivant renvoie la partie de l'entrée correspondant à la première occurrence de la chaîne FOX, à l'aide d'une correspondance avec respect de la casse.

```
SELECT regexp_substr('the fox', 'FOX', 1, 1, 'i');

 regexp_substr
---------------
 fox
```

L'exemple suivant renvoie la première partie de l'entrée qui commence par des lettres minuscules. Il est fonctionnellement identique à la même instruction SELECT sans le paramètre c.

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');
```

```
 regexp_substr
---------------
 abc
```

L'exemple suivant utilise un modèle écrit dans le type PCRE pour localiser des mots contenant au moins un chiffre et une lettre minuscule. Il utilise l'opérateur ?=, qui a une connotation « anticipée » spécifique au type PCRE. Cet exemple renvoie la partie de l'entrée correspondant au deuxième mot de ce type.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

L'exemple suivant utilise un modèle écrit dans le type PCRE pour localiser des mots contenant au moins un chiffre et une lettre minuscule. Il utilise l'opérateur ?=, qui a une connotation « anticipée » spécifique au type PCRE. Cet exemple renvoie la partie de l'entrée correspondant au deuxième mot de ce type, mais diffère de l'exemple précédent car il utilise une correspondance avec non-respect de la casse.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
---------------
 A1234
```

L'exemple suivant utilise une sous-expression pour rechercher la deuxième chaîne correspondant au modèle `'this is a (\\w+)'` à l'aide d'une correspondance avec respect de la casse. Il renvoie la sous-expression entre parenthèses.

```
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
```

Fonction REPEAT

Répète une chaîne le nombre de fois spécifié. Si le paramètre d'entrée est numérique, REPEAT le traite sous forme de chaîne.

Syntaxe

```
REPEAT(string, integer)
```

Arguments

string

Le premier paramètre d'entrée est la chaîne à répéter.

integer

Le deuxième paramètre est un nombre entier indiquant combien de fois répéter la chaîne.

Type de retour

La fonction REPEAT renvoie une chaîne.

Exemples

L'exemple suivant répète la valeur de la colonne CATID dans la table CATEGORY à trois reprises :

```
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)
```

Fonction REPLACE

Remplace toutes les occurrences d'un jeu de caractères au sein d'une chaîne existante par d'autres caractères spécifiés.

REPLACE est similaire à la Fonction TRANSLATE et la Fonction REGEXP_REPLACE, sauf que TRANSLATE fait plusieurs remplacements de caractère unique et REGEXP_REPLACE vous permet de rechercher un modèle d'expression régulière dans une chaîne, tandis que REPLACE remplace une chaîne entière par une autre chaîne.

Syntaxe

```
REPLACE(string1, old_chars, new_chars)
```

Arguments

string

Chaîne CHAR ou VARCHAR à rechercher

old_chars

Chaîne CHAR ou VARCHAR à remplacer.

new_chars

Nouvelle chaîne CHAR ou VARCHAR remplaçant l'ancienne chaîne old_string.

Type de retour

VARCHAR

Si old_chars ou new_chars a la valeur NULL, le retour est NULL.

Exemples

L'exemple suivant convertit la chaîne `Shows` en `Theatre` dans le champ CATGROUP :

```
select catid, catgroup,
replace(catgroup, 'Shows', 'Theatre')
from category
order by 1,2,3;

 catid | catgroup | replace
-------+----------+----------
     1 | Sports   | Sports
     2 | Sports   | Sports
     3 | Sports   | Sports
     4 | Sports   | Sports
     5 | Sports   | Sports
     6 | Shows    | Theatre
     7 | Shows    | Theatre
     8 | Shows    | Theatre
     9 | Concerts | Concerts
    10 | Concerts | Concerts
    11 | Concerts | Concerts
(11 rows)
```

Fonction REVERSE

La fonction REVERSE s'applique à une chaîne et renvoie les caractères dans l'ordre inverse. Par exemple, `reverse('abcde')` renvoie `edcba`. Cette fonction s'applique aux types de données numérique et de date, ainsi qu'aux types de données de caractère. Toutefois, dans la plupart des cas, elle a une valeur pratique pour les chaînes de caractères.

Syntaxe

```
REVERSE ( expression )
```

Argument

expression

Expression avec un type de données de caractère, date, horodatage ou numérique qui représente la cible de l'inversion de caractères. Toutes les expressions régulières sont implicitement converties en chaînes de caractères de longueur variable. Les espaces de fin des chaînes de caractères à largeur fixe sont ignorés.

Type de retour

REVERSE renvoie un VARCHAR.

Exemples

Sélectionnez cinq noms de ville distincts et leur noms inversés correspondants à partir de la table USERS :

```
select distinct city as cityname, reverse(cityname)
from users order by city limit 5;

cityname | reverse
---------+----------
Aberdeen | needrebA
Abilene  | enelibA
Ada      | adA
Agat     | tagA
Agawam   | mawagA
(5 rows)
```

Sélectionnez cinq ventes IDs et leur transformation inversée correspondante IDs sous forme de chaînes de caractères :

```
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)
```

Fonction RTRIM

La fonction RTRIM supprime un ensemble spécifié de caractères à partir de la fin d’une chaîne. Supprime la chaîne la plus longue ne contenant que des caractères de la liste des caractères supprimés. Le découpage est terminé lorsqu'aucun caractère de découpage n'apparaît dans la chaîne d'entrée.

Syntaxe

```
RTRIM( string, trim_chars )
```

Arguments

string

Une colonne de chaîne, une expression ou un littéral de chaîne à supprimer.

trim_chars

Colonne de chaîne, expression ou littéral de chaîne représentant les caractères à supprimer à la fin de la chaîne. Si la valeur n'est pas spécifiée, un espace est utilisé comme caractère de séparation.

Type de retour

Chaîne qui a le même type de données que l'argument string.

exemple

L'exemple suivant tronque les espaces de début et de fin de la chaîne `' abc '` :

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

L'exemple suivant supprime les chaînes `'xyz'` de fin de la chaîne `'xyzaxyzbxyzcxyz'`. Les occurrences de fin de `'xyz'` sont supprimées, mais celles qui se trouvent à l'intérieur de la chaîne sont conservées.

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

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

L'exemple suivant supprime les parties de fin de la chaîne `'setuphistorycassettes'` qui correspondent à l'un des caractères de la liste trim_chars `'tes'`. Tout caractère `t`, `e` ou `s` précédant un autre caractère qui ne figure pas dans la liste trim_chars à la fin de la chaîne d'entrée est supprimé.

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

L'exemple suivant tronque les caractères « Park » à la fin de VENUENAME le cas échéant :

```
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
```

Notez que RTRIM supprime les caractères `P`, `a`, `r` ou `k` lorsqu'ils apparaissent à la fin d'un VENUENAME.

Fonction SPLIT

La fonction SPLIT vous permet d'extraire des sous-chaînes d'une chaîne plus grande et de les utiliser sous forme de tableau. La fonction SPLIT est utile lorsque vous devez décomposer une chaîne en composants individuels en fonction d'un délimiteur ou d'un modèle spécifique.

Syntaxe

```
split(str, regex, limit)
```

Arguments

str

Expression de chaîne à diviser.

regex

Chaîne représentant une expression régulière. La chaîne regex doit être une expression régulière Java.

limit

Expression entière qui contrôle le nombre de fois que l'expression régulière est appliquée.

- limit > 0 : la longueur du tableau résultant ne sera pas supérieure à la limite, et la dernière entrée du tableau résultant contiendra toutes les entrées au-delà de la dernière regex correspondante.
- limit <= 0 : regex sera appliquée autant de fois que possible, et le tableau résultant peut être de n'importe quelle taille.

Type de retour

La fonction SPLIT renvoie un ARRAY<STRING>.

Si `limit > 0` : La longueur du tableau résultant ne sera pas supérieure à la limite, et la dernière entrée du tableau résultant contiendra toutes les entrées au-delà de la dernière expression régulière correspondante.

if `limit <= 0` : regex sera appliquée autant de fois que possible, et le tableau résultant peut être de n'importe quelle taille.

exemple

Dans cet exemple, la fonction SPLIT divise la chaîne d'entrée `'oneAtwoBthreeC'` là où elle rencontre les caractères `'A''B'`, ou `'C'` (comme spécifié par le modèle d'expression régulière`'[ABC]'`). Le résultat obtenu est un tableau de quatre éléments : `"one""two"`,`"three"`, et une chaîne vide`""`.

```
SELECT split('oneAtwoBthreeC', '[ABC]');
 ["one","two","three",""]
```

Fonction SPLIT_PART

Divise une chaîne sur le délimiteur spécifié et renvoie la partie à la position spécifiée.

Syntaxe

```
SPLIT_PART(string, delimiter, position)
```

Arguments

string

Colonne de chaîne, expression ou littéral de chaîne à fractionner. La chaîne peut être CHAR ou VARCHAR.

delimiter

Chaîne de délimiteur indiquant les sections de la chaîne d'entrée.

Si delimiter est un littéral, mettez-le entre guillemets simples.

position

Position de la partie de chaîne à renvoyer (à partir de 1). Doit être un nombre entier supérieur à 0. Si la valeur de position est supérieure au nombre de parties de chaîne, SPLIT_PART renvoie une chaîne vide. Si délimiteur est introuvable dans chaîne, alors la valeur renvoyée contient le contenu de la partie spécifiée, qui pourrait être la chaîne entière ou une valeur vide.

Type de retour

Chaîne CHAR ou VARCHAR, la même que le paramètre string.

Exemples

L'exemple suivant fractionne un littéral de chaîne en différentes parties en utilisant le délimiteur $ et renvoie la seconde partie.

```
select split_part('abc$def$ghi','$',2)
```

```
split_part
----------
def
```

L'exemple suivant fractionne un littéral de chaîne en différentes parties en utilisant le délimiteur $. Il renvoie une chaîne vide, car la partie 4 est introuvable.

```
select split_part('abc$def$ghi','$',4)

split_part
----------

```

L'exemple suivant fractionne un littéral de chaîne en différentes parties en utilisant le délimiteur #. Il renvoie la chaîne entière, qui correspond à la première partie, car le délimiteur est introuvable.

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

L'exemple suivant divise le champ d'horodatage LISTTIME en composants d'année, de mois et de date.

```
select listtime, split_part(listtime,'-',1) as year,
split_part(listtime,'-',2) as month,
split_part(split_part(listtime,'-',3),' ',1) as day
from listing limit 5;

      listtime       | year | month | day
---------------------+------+-------+------
 2008-03-05 12:25:29 | 2008 | 03    | 05
 2008-09-09 08:03:36 | 2008 | 09    | 09
 2008-09-26 05:43:12 | 2008 | 09    | 26
 2008-10-04 02:00:30 | 2008 | 10    | 04
 2008-01-06 08:33:11 | 2008 | 01    | 06
```

L'exemple suivant sélectionne le champ d'horodatage LISTTIME et le divise sur le caractère `'-'` pour obtenir le mois (la deuxième partie de la chaîne LISTTIME), puis compte le nombre d'entrées de chaque mois :

```
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
```

Fonction SUBSTRING

Renvoie le sous-ensemble d'une chaîne sur la base de la position de départ spécifiée.

Si l'entrée est une chaîne de caractères, la position de départ et le nombre de caractères extraits sont basés sur les caractères, pas les octets, afin que les caractères à plusieurs octets soient comptés comme des caractères uniques. Si l'entrée est une expression binaire, la position de départ et la sous-chaîne extraite sont basées sur des octets. Vous ne pouvez pas spécifier de longueur négative, mais vous pouvez spécifier une position de début négative.

Syntaxe

```
SUBSTRING(charactestring FROM start_position [ FOR numbecharacters ] )
```

```
SUBSTRING(charactestring, start_position, numbecharacters )
```

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

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

Arguments

chaîne de caractères

Chaîne à rechercher. Les types de données non-caractères sont traités comme une chaîne.

start_position

Position au sein de la chaîne à laquelle commencer l'extraction, à partir de 1. La position de début start_position est basée sur le nombre de caractères, pas d'octets, de sorte que les caractères à plusieurs octets soient comptés comme des caractères seuls. Ce numéro peut être négatif.

caractères numériques

Nombre de caractères à extraire (longueur de la sous-chaîne). Le nombre de caractères est basé sur le nombre de caractères, et non sur le nombre d'octets, de sorte que les caractères multi-octets sont considérés comme des caractères uniques. Ce numéro ne peut pas être négatif.

start_byte

Position au sein de l'expression binaire à laquelle commencer l'extraction, à partir de 1. Ce numéro peut être négatif.

nombre d'octets

Nombre d'octets à extraire, c'est-à-dire la longueur de la sous-chaîne. Ce numéro ne peut pas être négatif.

Type de retour

VARCHAR

Notes d'utilisation pour les chaînes de caractères

L'exemple suivant renvoie une chaîne de quatre caractères commençant par le sixième caractère.

```
select substring('caterpillar',6,4);
substring
-----------
pill
(1 row)
```

Si le nombre de caractères start_position + dépasse la longueur de la chaîne, SUBSTRING renvoie une sous-chaîne commençant par la position de départ jusqu'à la fin de la chaîne. Par exemple :

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

Si `start_position` est négatif ou égal à 0, la fonction SUBSTRING renvoie une sous-chaîne commençant au premier caractère de la chaîne d'une longueur de `start_position` + numbe`characters` -1. Par exemple :

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

Si `start_position` + numbe`characters` -1 est inférieur ou égal à zéro, SUBSTRING renvoie une chaîne vide. Par exemple :

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

Exemples

L'exemple suivant renvoie le mois de la chaîne LISTTIME dans la table LISTING :

```
select listid, listtime,
substring(listtime, 6, 2) as month
from listing
order by 1, 2, 3
limit 10;

 listid |      listtime       | month
--------+---------------------+-------
      1 | 2008-01-24 06:43:29 | 01
      2 | 2008-03-05 12:25:29 | 03
      3 | 2008-11-01 07:35:33 | 11
      4 | 2008-05-24 01:18:37 | 05
```

```
      5 | 2008-05-17 02:29:11 | 05
      6 | 2008-08-15 02:08:13 | 08
      7 | 2008-11-15 09:38:15 | 11
      8 | 2008-11-09 05:07:30 | 11
      9 | 2008-09-09 08:03:36 | 09
     10 | 2008-06-17 09:44:54 | 06
(10 rows)
```

L'exemple suivant est le même que ci-dessus, mais utilise l'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)
```

Vous ne pouvez pas utiliser SUBSTRING pour extraire de manière prévisible le préfixe d'une chaîne pouvant contenir des caractères à plusieurs octets, car vous devez spécifier la longueur d'une chaîne de plusieurs octets basée sur le nombre d'octets, pas sur le nombre de caractères. Pour extraire le segment de début d'une chaîne en fonction de la longueur en octets, vous pouvez utiliser la fonction CAST sur la chaîne au format VARCHAR(byte_length) pour tronquer la chaîne, où byte_length est la longueur requise. L'exemple suivant extrait les 5 premiers octets de la chaîne `'Fourscore and seven'`.

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
```

```
Fours
```

L'exemple suivant renvoie le prénom Ana qui apparaît après le dernier espace de la chaîne d'entrée `Silva, Ana`.

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))

 reverse
-----------
 Ana
```

Fonction TRANSLATE

Pour une expression données, remplace toutes les occurrences de caractères spécifiés par des produits de remplacement spécifiés. Les caractères existants sont mappés à des caractères de remplacement en fonction de leurs positions dans les arguments characters_to_replace et characters_to_substitute. Si le nombre de caractères spécifiés dans l'argument characters_to_replace est supérieur à celui de l'argument characters_to_substitute, les caractères supplémentaires depuis l'argument characters_to_replace sont omis dans la valeur de retour.

TRANSLATE est similaire à la Fonction REPLACE et la Fonction REGEXP_REPLACE, sauf que REPLACE remplace une chaîne entière par une autre chaîne et que REGEXP_REPLACE vous permet de rechercher un modèle d'expression régulière dans une chaîne, tandis que TRANSLATE fait plusieurs remplacements de caractère unique.

Si un argument a la valeur null, le retour est NULL.

Syntaxe

```
TRANSLATE ( expression, characters_to_replace, characters_to_substitute )
```

Arguments

expression

Expression à traduire.

characters_to_replace

Chaîne contenant les caractères à remplacer.

characters_to_substitute

Chaîne contenant les caractères à remplacer.

Type de retour

VARCHAR

Exemples

L'exemple suivant remplace plusieurs caractères dans une chaîne :

```
select translate('mint tea', 'inea', 'osin');

translate
-----------
most tin
```

L'exemple suivant remplace le signe (@) par un point dans toutes les valeurs d'une colonne :

```
select email, translate(email, '@', '.') as obfuscated_email
from users limit 10;

email                                          obfuscated_email
-------------------------------------------------------------------------------------------
Etiam.laoreet.libero@sodalesMaurisblandit.edu
 Etiam.laoreet.libero.sodalesMaurisblandit.edu
amet.faucibus.ut@condimentumegetvolutpat.ca
 amet.faucibus.ut.condimentumegetvolutpat.ca
turpis@accumsanlaoreet.org                 turpis.accumsanlaoreet.org
ullamcorper.nisl@Cras.edu                 ullamcorper.nisl.Cras.edu
arcu.Curabitur@senectusetnetus.com              arcu.Curabitur.senectusetnetus.com
ac@velit.ca                                 ac.velit.ca
Aliquam.vulputate.ullamcorper@amalesuada.org
 Aliquam.vulputate.ullamcorper.amalesuada.org
vel.est@velitegestas.edu                        vel.est.velitegestas.edu
dolor.nonummy@ipsumdolorsit.ca                  dolor.nonummy.ipsumdolorsit.ca
et@Nunclaoreet.ca                               et.Nunclaoreet.ca
```

L'exemple suivant remplace des espaces par des traits de soulignement et supprime les périodes de toutes les valeurs d'une colonne :

```
select city, translate(city, ' .', '_') from users
where city like 'Sain%' or city like 'St%'
group by city
order by city;

city            translate
--------------+------------------
Saint Albans     Saint_Albans
Saint Cloud      Saint_Cloud
Saint Joseph     Saint_Joseph
Saint Louis      Saint_Louis
Saint Paul       Saint_Paul
St. George       St_George
St. Marys        St_Marys
St. Petersburg   St_Petersburg
Stafford         Stafford
Stamford         Stamford
Stanton          Stanton
Starkville       Starkville
Statesboro       Statesboro
Staunton         Staunton
Steubenville     Steubenville
Stevens Point    Stevens_Point
Stillwater       Stillwater
Stockton         Stockton
Sturgis          Sturgis
```

Fonction TRIM

Tronque une chaîne en supprimant les espaces de début et de fin ou en supprimant les caractères de début et de fin qui correspondent à une chaîne spécifiée de manière facultative.

Syntaxe

```
TRIM( [ BOTH ] [ trim_chars FROM ] string
```

Arguments

trim_chars

(Facultatif) Caractères à tronquer à partir de la chaîne. Si ce paramètre est oublié, les blancs sont tronqués.

string

Chaîne à tronquer.

Type de retour

La fonction TRIM renvoie une chaîne VARCHAR ou CHAR. Si vous utilisez la fonction TRIM avec une commande SQL, les résultats sont AWS Clean Rooms implicitement convertis en VARCHAR. Si vous utilisez la fonction TRIM dans la liste SELECT pour une fonction SQL, AWS Clean Rooms elle ne convertit pas implicitement les résultats et vous devrez peut-être effectuer une conversion explicite pour éviter une erreur de non-concordance des types de données. Consultez la Fonction CAST fonction pour plus d'informations sur les conversions explicites.

exemple

L'exemple suivant tronque les espaces de début et de fin de la chaîne `'  abc  '` :

```
select '     abc    ' as untrim, trim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

L'exemple suivant supprime les guillemets qui entourent de la chaîne `"dog"` :

```
select trim('"' FROM '"dog"');

btrim
-------
dog
```

TRIM supprime les caractères de trim_chars qui apparaissent au début de la chaîne. L'exemple suivant supprime les caractères C, D et G lorsqu'ils figurent au début de VENUENAME, qui est une colonne VARCHAR.

```
select venueid, venuename, trim(venuename, 'CDG')
from venue
where venuename like '%Park'
order by 2
limit 7;
```

```
venueid | venuename                  | btrim
--------+----------------------------+-------------------------
    121 | ATT Park                   | ATT Park
    109 | Citizens Bank Park         | itizens Bank Park
    102 | Comerica Park              | omerica Park
      9 | Dick's Sporting Goods Park | ick's Sporting Goods Park
     97 | Fenway Park                | Fenway Park
    112 | Great American Ball Park   | reat American Ball Park
    114 | Miller Park                | Miller Park
```

Fonction UPPER

Convertit la valeur en majuscules. UPPER prend en charge les caractères à plusieurs octets UTF-8, à concurrence de quatre octets au maximum par caractère.

Syntaxe

```
UPPER(string)
```

Arguments

string

Le paramètre d'entrée est une chaîne VARCHAR (ou tout autre type de données, tel que CHAR, qui peut être implicitement converti en VARCHAR).

Type de retour

La fonction UPPER renvoie une chaîne de caractères qui est du même type que la chaîne d'entrée.

Exemples

L'exemple suivant convertit le champ CATNAME en majuscules :

```
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)
```

Fonction UUID

La fonction UUID génère un identifiant unique universel (UUID).

UUIDs sont des identifiants uniques à l'échelle mondiale qui sont couramment utilisés pour fournir des identifiants uniques à diverses fins, telles que :

- Identification des enregistrements de base de données ou d'autres entités de données.
- Génération de noms ou de clés uniques pour des fichiers, des répertoires ou d'autres ressources.
- Suivi et corrélation des données entre les systèmes distribués.
- Fournir des identifiants uniques pour les paquets réseau, les composants logiciels ou d'autres actifs numériques.

La fonction UUID génère une valeur UUID unique avec une probabilité très élevée, même sur des systèmes distribués et sur de longues périodes. UUIDs sont généralement générés à l'aide d'une combinaison de l'horodatage actuel, de l'adresse réseau de l'ordinateur et d'autres données aléatoires ou pseudo-aléatoires, ce qui garantit qu'il est très peu probable que chaque UUID généré entre en conflit avec un autre UUID.

Dans le contexte d'une requête SQL, la fonction UUID peut être utilisée pour générer des identifiants uniques pour les nouveaux enregistrements insérés dans une base de données, ou pour fournir des clés uniques pour le partitionnement des données, l'indexation ou à d'autres fins nécessitant un identifiant unique.

> **Note**
>
> La fonction UUID n'est pas déterministe.

Syntaxe

```
uuid()
```

Arguments

La fonction UUID ne prend aucun argument.

Type de retour

UUID renvoie une chaîne d'identifiant unique universel (UUID). La valeur est renvoyée sous la forme d'une chaîne UUID canonique de 36 caractères.

exemple

L'exemple suivant génère un identifiant unique universel (UUID). La sortie est une chaîne de 36 caractères représentant un identifiant unique universel.

```
SELECT uuid();
 46707d92-02f4-4817-8116-a4c3b23e6266
```

Fonctions liées à la confidentialité

AWS Clean Rooms fournit des fonctions qui vous aident à respecter les spécifications suivantes en matière de respect de la vie privée.

- Global Privacy Platform (GPP) — Spécification de l'Interactive Advertising Bureau (IAB) qui établit un cadre mondial normalisé pour la confidentialité en ligne et l'utilisation des données. Pour plus d'informations sur les spécifications techniques du GPP, consultez la documentation de la Global Privacy Platform sur GitHub.
- Cadre de transparence et de consentement (TCF) — Élément clé du GPP, lancé en 2020, qui fournit un cadre technique standardisé pour aider les entreprises à se conformer aux réglementations en matière de confidentialité telles que le règlement général sur la protection des données (RGPD) de l'UE. Le TCF permet aux clients d'accorder ou de refuser leur consentement à la collecte et au traitement des données. Pour plus d'informations sur les spécifications techniques du TCF, consultez la documentation du TCF sur. GitHub

Rubriques

- Fonction consent_gpp_v1_decode

- Fonction consent_tcf_v2_decode

Fonction consent_gpp_v1_decode

La `consent_gpp_v1_decode` fonction est utilisée pour décoder les données de consentement de la Global Privacy Platform (GPP) v1. Il prend la chaîne de consentement codée en entrée et renvoie les données de consentement décodées, qui incluent des informations sur les préférences de confidentialité et les choix de consentement de l'utilisateur. Cette fonction est utile lorsque vous travaillez avec des données qui incluent des informations de consentement GPP v1, car elle vous permet d'accéder aux données de consentement et de les analyser dans un format structuré.

Syntaxe

```
consent_gpp_v1_decode(gpp_string)
```

Arguments

gpp_string

La chaîne de consentement GPP v1 codée.

Renvoie

Le dictionnaire renvoyé inclut les paires clé-valeur suivantes :

- `version`: version de la spécification GPP utilisée (actuellement 1).
- `cmpId`: ID de la plateforme de gestion du consentement (CMP) qui a codé la chaîne de consentement.
- `cmpVersion`: version du CMP qui a codé la chaîne de consentement.
- `consentScreen`: ID de l'écran dans l'interface utilisateur CMP où l'utilisateur a donné son consentement.
- `consentLanguage`: Le code de langue des informations de consentement.
- `vendorListVersion`: version de la liste des fournisseurs utilisée.
- `publisherCountryCode`: le code du pays de l'éditeur.
- `purposeConsent`: liste d'entiers représentant les objectifs auxquels l'utilisateur a consenti.
- `purposeLegitimateInterest`: Une liste des objectifs IDs pour lesquels l'intérêt légitime de l'utilisateur a été communiqué de manière transparente.

- `specialFeatureOptIns`: liste d'entiers représentant les fonctionnalités spéciales que l'utilisateur a choisies.
- `vendorConsent`: liste des fournisseurs IDs auxquels l'utilisateur a donné son accord.
- `vendorLegitimateInterest`: Liste des fournisseurs IDs pour lesquels l'intérêt légitime de l'utilisateur a été communiqué de manière transparente.

exemple

L'exemple suivant prend un seul argument, qui est la chaîne de consentement codée. Il renvoie un dictionnaire contenant les données de consentement décodées, y compris des informations sur les préférences de confidentialité de l'utilisateur, les choix de consentement et d'autres métadonnées.

```
SELECT * FROM consent_gpp_v1_decode('ABCDEFGHIJK');
```

La structure de base des données de consentement renvoyées comprend des informations sur la version de la chaîne de consentement, les détails de la CMP (Consent Management Platform), le consentement de l'utilisateur et les choix d'intérêts légitimes pour différents objectifs et fournisseurs, ainsi que d'autres métadonnées.

```
{
    "version": 1,
    "cmpId": 12,
    "cmpVersion": 34,
    "consentScreen": 5,
    "consentLanguage": "en",
    "vendorListVersion": 89,
    "publisherCountryCode": "US",
    "purposeConsent": [1],
    "purposeLegitimateInterests": [1],
    "specialFeatureOptins": [1],
    "vendorConsent": [1],
    "vendorLegitimateInterests": [1]}
}
```

Fonction consent_tcf_v2_decode

La `consent_tcf_v2_decode` fonction est utilisée pour décoder les données de consentement du Transparency and Consent Framework (TCF) v2. Il prend la chaîne de consentement codée en entrée et renvoie les données de consentement décodées, qui incluent des informations sur les

préférences de confidentialité et les choix de consentement de l'utilisateur. Cette fonction est utile lorsque vous travaillez avec des données qui incluent des informations de consentement TCF v2, car elle vous permet d'accéder aux données de consentement et de les analyser dans un format structuré.

Syntaxe

```
consent_tcf_v2_decode(tcf_string)
```

Arguments

tcf_string

La chaîne de consentement TCF v2 codée.

Renvoie

La `consent_tcf_v2_decode` fonction renvoie un dictionnaire contenant les données de consentement décodées à partir d'une chaîne de consentement TCF (Transparency and Consent Framework) v2.

Le dictionnaire renvoyé inclut les paires clé-valeur suivantes :

Segment principal

- `version`: version de la spécification TCF utilisée (actuellement 2).
- `created`: date et heure de création de la chaîne de consentement.
- `lastUpdated`: date et heure de dernière mise à jour de la chaîne de consentement.
- `cmpId`: ID de la plateforme de gestion du consentement (CMP) qui a codé la chaîne de consentement.
- `cmpVersion`: version du CMP qui a codé la chaîne de consentement.
- `consentScreen`: ID de l'écran dans l'interface utilisateur CMP où l'utilisateur a donné son consentement.
- `consentLanguage`: Le code de langue des informations de consentement.
- `vendorListVersion`: version de la liste des fournisseurs utilisée.
- `tcfPolicyVersion`: version de la politique TCF sur laquelle est basée la chaîne de consentement.

- `isServiceSpecific`: valeur booléenne indiquant si le consentement est spécifique à un service en particulier ou s'applique à tous les services.
- `useNonStandardStacks`: valeur booléenne indiquant si des piles non standard sont utilisées.
- `specialFeatureOptIns`: liste d'entiers représentant les fonctionnalités spéciales que l'utilisateur a choisies.
- `purposeConsent`: liste d'entiers représentant les objectifs auxquels l'utilisateur a consenti.
- `purposesLITransparency`: Une liste d'entiers représentant les objectifs pour lesquels l'utilisateur a exprimé un intérêt légitime en termes de transparence.
- `purposeOneTreatment`: valeur booléenne indiquant si l'utilisateur a demandé le « traitement dans un seul but » (c'est-à-dire que tous les objectifs sont traités de la même manière).
- `publisherCountryCode`: le code du pays de l'éditeur.
- `vendorConsent`: liste des fournisseurs IDs auxquels l'utilisateur a donné son accord.
- `vendorLegitimateInterest`: Liste des fournisseurs IDs pour lesquels l'intérêt légitime de l'utilisateur a été communiqué de manière transparente.
- `pubRestrictionEntry`: liste des restrictions imposées aux éditeurs. Ce champ contient l'ID d'objectif, le type de restriction et la liste des fournisseurs concernés IDs par cette restriction d'objectif.

Segment de fournisseurs divulgué

- `disclosedVendors`: liste d'entiers représentant les fournisseurs qui ont été divulgués à l'utilisateur.

Segment destiné aux éditeurs

- `pubPurposesConsent`: une liste d'entiers représentant les objectifs spécifiques à l'éditeur pour lesquels l'utilisateur a donné son consentement.
- `pubPurposesLITransparency`: une liste de nombres entiers représentant les objectifs spécifiques à l'éditeur pour lesquels l'utilisateur a fait preuve de transparence en matière d'intérêts légitimes.
- `customPurposesConsent`: liste d'entiers représentant les objectifs personnalisés pour lesquels l'utilisateur a donné son consentement.
- `customPurposesLITransparency`: Une liste d'entiers représentant les objectifs personnalisés pour lesquels l'utilisateur a accordé la transparence à ses intérêts légitimes.

Ces données de consentement détaillées peuvent être utilisées pour comprendre et respecter les préférences de confidentialité de l'utilisateur lorsqu'il travaille avec des données personnelles.

exemple

L'exemple suivant prend un seul argument, qui est la chaîne de consentement codée. Il renvoie un dictionnaire contenant les données de consentement décodées, y compris des informations sur les préférences de confidentialité de l'utilisateur, les choix de consentement et d'autres métadonnées.

```
from aws_clean_rooms.functions import consent_tcf_v2_decode

consent_string = "CO1234567890abcdef"
consent_data = consent_tcf_v2_decode(consent_string)

print(consent_data)
```

La structure de base des données de consentement renvoyées comprend des informations sur la version de la chaîne de consentement, les détails de la CMP (Consent Management Platform), le consentement de l'utilisateur et les choix d'intérêts légitimes pour différents objectifs et fournisseurs, ainsi que d'autres métadonnées.

```
    /** core segment **/
    version: 2,
    created: "2023-10-01T12:00:00Z",
    lastUpdated: "2023-10-01T12:00:00Z",
    cmpId: 1234,
    cmpVersion: 5,
    consentScreen: 1,
    consentLanguage: "en",
    vendorListVersion: 2,
    tcfPolicyVersion: 2,
    isServiceSpecific: false,
    useNonStandardStacks: false,
    specialFeatureOptIns: [1, 2, 3],
    purposeConsent: [1, 2, 3],
    purposesLITransparency: [1, 2, 3],
    purposeOneTreatment: true,
    publisherCountryCode: "US",
    vendorConsent: [1, 2, 3],
    vendorLegitimateInterest: [1, 2, 3],
    pubRestrictionEntry: [
```

```
        { purpose: 1, restrictionType: 2, restrictionDescription: "Example
 restriction" },
    ],

    /** disclosed vendor segment **/
    disclosedVendors: [1, 2, 3],

    /** publisher purposes  segment **/
    pubPurposesConsent: [1, 2, 3],
    pubPurposesLITransparency: [1, 2, 3],
    customPurposesConsent: [1, 2, 3],
    customPurposesLITransparency: [1, 2, 3],
};
```

Fonctions de fenêtrage

En utilisant les fonctions de fenêtrage, vous pouvez créer des requêtes d'analyse commerciale plus efficacement. Les fonctions de fenêtrage fonctionnent sur une partition ou « fenêtre » d'un ensemble de résultats et renvoient une valeur pour chaque ligne de cette fenêtre. En revanche, les fonctions non fenêtrées effectuent leurs calculs sur chaque ligne du jeu de résultats. Contrairement aux fonctions de groupe qui regroupent les lignes de résultats, les fonctions de fenêtrage conservent toutes les lignes de l'expression de table.

Les valeurs renvoyées sont calculées en utilisant les valeurs des ensembles de lignes de cette fenêtre. Pour chaque ligne de la table, la fenêtre définit un ensemble de lignes qui est utilisé pour calculer des attributs supplémentaires. Une fenêtre est définie à l'aide d'une spécification de fenêtrage (clause OVER) et s'appuie sur trois concepts principaux :

- Le partitionnement de fenêtrage qui constitue des groupes de lignes (clause PARTITION)
- L'ordonnancement de fenêtrage, qui définit un ordre ou une séquence de lignes dans chaque partition (clause ORDER BY)
- Les cadres de fenêtrage, qui sont définis par rapport à chaque ligne afin de limiter davantage l'ensemble de lignes (spécification ROWS)

Les fonctions de fenêtrage constituent le dernier ensemble d'opérations effectuées dans une requête à l'exception de la clause ORDER BY finale. Toutes les jointures et toutes les clauses WHERE, GROUP BY et HAVING doivent être terminées avant que les fonctions de fenêtrage soient traitées. Par conséquent, les fonctions de fenêtrage peuvent s'afficher uniquement dans la liste de sélection ou la clause ORDER BY. Vous pouvez utiliser plusieurs fonctions de fenêtrage dans une

seule requête avec différentes clauses de cadre. Vous pouvez également utiliser des fonctions de fenêtrage dans d'autres expressions scalaires, telles que CASE.

Récapitulatif de la syntaxe de la fonction de fenêtrage

Les fonctions de fenêtre suivent la syntaxe standard suivante.

```
function (expression) OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list [ frame_clause ] ] )
```

Ici, function est l'une des fonctions décrites dans cette section.

L'expr_list se présente comme suit.

```
expression | column_name [, expr_list ]
```

L'order_list se présente comme suit.

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

La frame_clause se présente comme suit.

```
ROWS
{ UNBOUNDED PRECEDING | unsigned_value PRECEDING | CURRENT ROW } |

{ BETWEEN
{ UNBOUNDED PRECEDING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW}
AND
{ UNBOUNDED FOLLOWING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW }}
```

Arguments

fonction

La fonction de fenêtrage. Pour plus d'informations, consultez les descriptions de chaque fonction.

OVER

La clause qui définit la spécification du fenêtrage. La clause OVER est obligatoire pour les fonctions de fenêtrage et différencie les fonctions de fenêtrage d'autres fonctions SQL.

PARTITION BY expr_list

(Facultatif) La clause PARTITION BY subdivise le jeu de résultats en partitions, comme la clause GROUP BY. Si une clause de partition est présente, la fonction est calculée pour les lignes de chaque partition. Si aucune clause de partition n'est spécifiée, une seule partition contient la totalité de la table et la fonction est calculée pour cette table complète.

Les fonctions de rang DENSE_RANK, NTILE, RANK et ROW_NUMBER, nécessitent une comparaison globale de toutes les lignes du jeu de résultats. Lorsqu'une clause PARTITION BY est utilisée, l'optimiseur de requête peut exécuter chaque agrégation en parallèle en répartissant la charge de travail sur plusieurs tranches selon les partitions. Si la clause PARTITION BY n'est pas présente, l'étape d'agrégation doit être exécutée en série sur une seule tranche, ce qui peut avoir une incidence négative importante sur les performances, surtout pour des clusters de grande taille.

AWS Clean Roomsne prend pas en charge les littéraux de chaîne dans les clauses PARTITION BY.

ORDER BY order_list

(Facultatif) La fonction de fenêtrage est appliquée aux lignes de chaque partition triées selon la spécification d'ordre de ORDER BY. Cette clause ORDER BY est distincte et sans aucun lien avec une clause ORDER BY dans la frame_clause. La clause ORDER BY peut être utilisée sans la clause PARTITION BY.

Pour les fonctions de rang, la clause ORDER BY identifie les mesures des valeurs de rang. Pour les fonctions d'agrégation, les lignes partitionnées doivent être ordonnées avant que la fonction d'agrégation soit calculée pour chaque cadre. Pour en savoir plus sur les types de fonction de fenêtrage, consultez Fonctions de fenêtrage.

Les identificateurs de colonnes ou les expressions qui correspondent aux identificateurs de colonnes sont requis dans la liste d'ordre. Ni les constantes, ni les expressions constantes ne peuvent être utilisées pour remplacer les noms de colonnes.

Les valeurs NULLS sont traitées comme leur propre groupe, triées et classées selon l'option NULLS FIRST ou NULLS LAST. Par défaut, les valeurs NULL sont triées et classées en dernier par ordre croissant (ASC) et triées et classées en premier par ordre décroissant (DESC).

AWS Clean Roomsne prend pas en charge les littéraux de chaîne dans les clauses ORDER BY.

Si la clause ORDER BY est omise, l'ordre des lignes est non déterministe.

> **Note**
>
> Dans tout système parallèleAWS Clean Rooms, par exemple lorsqu'une clause ORDER BY ne produit pas un ordre unique et total des données, l'ordre des lignes n'est pas déterministe. En d'autres termes, si l'expression ORDER BY produit des valeurs dupliquées (ordre partiel), l'ordre de retour de ces lignes peut varier d'une exécution AWS Clean Rooms à l'autre. De leur côté, les fonctions de fenêtrage peuvent renvoyer des résultats inattendus ou incohérents. Pour de plus amples informations, veuillez consulter Ordonnancement unique des données pour les fonctions de fenêtrage.

column_name

Nom d'une colonne à partitionner ou à ordonner.

ASC | DESC

Option qui définit l'ordre de tri de l'expression, comme suit :

- ASC : croissant (par exemple, de faible à élevé pour les valeurs numériques et de « A » à « Z » pour les chaînes de caractères). Si aucune option n'est spécifiée, les données sont triées dans l'ordre croissant par défaut.
- DESC : descendantes (valeurs d'élevées à faibles pour les valeurs numériques ; de « Z » à « A » pour les chaînes).

NULLS FIRST | NULLS LAST

Option qui spécifie si les valeurs NULLS devraient être classés en premier, avant les valeurs non NULL, ou en dernier, après les valeurs non NULL. Par défaut, les valeurs NULLS sont triées et classées en dernier par ordre croissant (ASC) et triées et classées en premier par ordre décroissant (DESC).

frame_clause

Pour les fonctions d'agrégation, la clause de cadre affine l'ensemble de lignes dans la fenêtre d'une fonction lorsque vous utilisez ORDER BY. Elle vous permet d'inclure ou d'exclure des ensembles de lignes dans le résultat ordonné. La clause de cadre se compose du mot-clé ROWS et des spécificateurs associés.

La clause frame ne s'applique pas aux fonctions de classement. En outre, la clause de cadre n'est pas requise lorsqu'aucune clause ORDER BY n'est utilisée dans la clause OVER pour une

fonction d'agrégation. Si une clause ORDER BY est utilisée pour une fonction d'agrégation, une clause de cadre explicite est requise.

Si aucune clause ORDER BY n'est spécifiée, le cadre implicite est sans limite : équivalent à ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

ROWS

Cette clause définit le cadre de fenêtrage en spécifiant un décalage physique de la ligne actuelle.

Cette clause spécifie les lignes de la fenêtre ou de la partition actuelle auxquelles la valeur de la ligne actuelle doit être associée. Elle utilise des arguments qui spécifient la position de la ligne, qui peut être avant ou après la ligne actuelle. Le point de référence de tous les cadres de fenêtrage est la ligne actuelle. Chaque ligne devient la ligne actuelle à son tour à mesure que le cadre de fenêtrage avance dans la partition.

Le cadre peut être un simple ensemble de lignes allant jusqu'à et incluant la ligne actuelle.

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

Ou il peut s'agir d'un ensemble de lignes situées entre les deux limites.

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING indique que la fenêtre commence à la première ligne de la partition ; offset PRECEDING indique que la fenêtre commence un certain nombre de lignes équivalant à la valeur de décalage avant la ligne actuelle. UNBOUNDED PRECEDING est la valeur par défaut.

CURRENT ROW indique que la fenêtre commence ou se termine à la ligne actuelle.

UNBOUNDED FOLLOWING indique que la fenêtre se termine à la dernière ligne de la partition ; offset FOLLOWING indique que la fenêtre se termine un certain nombre de lignes équivalant à la valeur de décalage après la ligne actuelle.

offset identifie un nombre physique de lignes avant ou après la ligne actuelle. Dans ce cas, offset doit être une constante ayant une valeur numérique positive. Par exemple, 5 FOLLOWING arrête les 5 lignes du cadre après la ligne actuelle.

Là où BETWEEN n'est pas spécifié, le cadre est implicitement délimité par la ligne actuelle. Par exemple, `ROWS 5 PRECEDING` est égal à `ROWS BETWEEN 5 PRECEDING AND CURRENT ROW`. En outre, `ROWS UNBOUNDED FOLLOWING` est égal à `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`.

> **Note**
>
> Vous ne pouvez pas spécifier un cadre dans lequel la limite de début est supérieure à la limite de fin. Par exemple, vous ne pouvez pas spécifier l'un des cadres suivants.
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

Ordonnancement unique des données pour les fonctions de fenêtrage

Si une clause ORDER BY pour une fonction de fenêtrage ne génère pas d'ordonnancement unique et total des données, l'ordre des lignes est non déterministe. Si l'expression ORDER BY génère des valeurs en double (ordonnancement partiel), l'ordre de ces lignes qui est renvoyé peut varier lors de plusieurs exécutions. Dans ce cas, les fonctions de fenêtrage peuvent également renvoyer des résultats inattendus ou incohérents.

Par exemple, la requête suivante renvoie des résultats différents sur plusieurs exécutions. Ces différents résultats se produisent parce que `order by dateid` ne produit pas d'ordonnancement unique des données pour la fonction de fenêtrage SUM.

```
select dateid, pricepaid,
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid |   sumpaid
--------+-----------+-------------
1827 |   1730.00 |     1730.00
1827 |    708.00 |     2438.00
1827 |    234.00 |     2672.00
...

select dateid, pricepaid,
```

```
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid |   sumpaid
--------+-----------+-------------
1827 |    234.00 |      234.00
1827 |    472.00 |      706.00
1827 |    347.00 |     1053.00
...
```

Dans ce cas, l'ajout d'une seconde colonne ORDER BY à la fonction de fenêtrage peut permettre de résoudre le problème.

```
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
...
```

Fonctions prises en charge

AWS Clean RoomsSpark SQL prend en charge deux types de fonctions de fenêtre : l'agrégation et le classement.

Vous trouverez ci-dessous les fonctions d'agrégation prises en charge :

- Fonction de fenêtrage CUME_DIST
- Fonction de fenêtrage DENSE_RANK
- Fonction de fenêtre FIRST
- Fonction de fenêtrage FIRST_VALUE
- Fonction de fenêtrage LAG
- Fonction LAST window
- Fonction de fenêtrage LAST_VALUE

- Fonction de fenêtrage LEAD

Vous trouverez ci-dessous les fonctions de classement prises en charge :

- Fonction de fenêtrage DENSE_RANK
- Fonction de fenêtrage PERCENT_RANK
- Fonction de fenêtrage RANK
- Fonction de fenêtrage ROW_NUMBER

Exemple de tableau contenant des exemples de fonctions de fenêtrage

Vous trouverez des exemples de fonctions de fenêtrage spécifiques avec la description de chaque fonction. Certains exemples utilisent une table nommée WINSALES, qui contient 11 lignes, comme indiqué dans le tableau suivant.

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	

Fonction de fenêtrage CUME_DIST

Calcule la distribution cumulée d'une valeur au sein d'une fenêtre ou une partition. En supposant que l'ordre est croissant, la distribution cumulée est déterminée à l'aide de la formule suivante :

```
count of rows with values <= x / count of rows in the window or partition
```

où x est égal à la valeur de la ligne actuelle de la colonne spécifiée dans la clause ORDER BY. Le jeu de données suivant illustre l'utilisation de cette formule :

```
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
```

La plage de valeur de retour est comprise entre >0 et 1, inclus.

Syntaxe

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Arguments

OVER

Clause qui spécifie le partitionnement de fenêtrage. La clause OVER ne peut pas contenir de spécification de cadre de fenêtrage.

PARTITION BY partition_expression

Facultatif. Expression qui définit la plage d'enregistrements de chaque groupe dans la clause OVER.

ORDER BY order_list

Expression permettant de calculer la distribution cumulée. L'expression doit disposer d'un type de données numériques ou être convertible implicitement en une. Si ORDER BY n'est pas spécifié, la valeur de retour est 1 pour toutes les lignes.

Si ORDER BY ne génère pas d'ordonnancement unique, l'ordre des lignes est non déterministe. Pour de plus amples informations, veuillez consulter Ordonnancement unique des données pour les fonctions de fenêtrage.

Type de retour

FLOAT8

Exemples

L'exemple suivant calcule la distribution cumulée de la quantité par vendeur :

```
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
```

Pour obtenir une description de la table WINSALES, consultez Exemple de tableau contenant des exemples de fonctions de fenêtrage.

Fonction de fenêtrage DENSE_RANK

La fonction de fenêtrage DENSE_RANK détermine le rang d'une valeur dans un groupe de valeurs, en fonction de l'expression ORDER BY dans la clause OVER. Si la clause PARTITION BY facultative est présente, les rangs sont réinitialisés pour chaque groupe de lignes. Les lignes avec des valeurs égales pour les critères de rang reçoivent le même rang. La fonction DENSE_RANK diffère de RANK sur un point : si deux lignes ou plus sont à égalité, il n'y a pas d'écart dans la séquence des valeurs classées. Par exemple, si deux lignes sont classées 1, le prochain rang est 2.

Vous pouvez avoir des fonctions de rang avec différentes clauses PARTITION BY et ORDER BY dans la même requête.

Syntaxe

```
DENSE_RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Arguments

( )

La fonction ne prend pas d'arguments, mais les parenthèses vides sont obligatoires.

OVER

Clauses de fenêtrage pour la fonction DENSE_RANK.

PARTITION BY expr_list

Facultatif. Une ou plusieurs expressions qui définissent le fenêtrage.

ORDER BY order_list

Facultatif. Expression sur laquelle sont basées les valeurs de rang. Si aucune clause PARTITION BY n'est spécifiée, ORDER BY utilise toute la table. Si ORDER BY n'est pas spécifié, la valeur de retour est 1 pour toutes les lignes.

Si ORDER BY ne génère pas d'ordonnancement unique, l'ordre des lignes est non déterministe. Pour de plus amples informations, veuillez consulter Ordonnancement unique des données pour les fonctions de fenêtrage.

Type de retour

INTEGER

Exemples

L'exemple suivant montre le classement de la table en fonction de la quantité vendue (par ordre décroissant) et l'affectation d'un rang dense et d'un rang standard à chaque ligne. Les résultats sont triés une fois que les résultats de la fonction de fenêtrage sont appliqués.

```
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)
```

Notez la différence entre les rangs affectés au même ensemble de lignes lorsque les fonctions DENSE_RANK et RANK sont utilisées côte à côte dans la même requête. Pour obtenir une description de la table WINSALES, consultez Exemple de tableau contenant des exemples de fonctions de fenêtrage.

L'exemple suivant montre le partitionnement de la table en fonction de chaque SELLERID, le classement de chaque partition selon la quantité (par ordre décroissant) et l'affectation d'un rang dense à chaque ligne. Les résultats sont triés une fois que les résultats de la fonction de fenêtrage sont appliqués.

```
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)
```

Pour obtenir une description de la table WINSALES, consultez Exemple de tableau contenant des exemples de fonctions de fenêtrage.

Fonction de fenêtre FIRST

À partir d'un ensemble ordonné de lignes, FIRST renvoie la valeur de l'expression spécifiée par rapport à la première ligne du cadre de fenêtre.

Pour savoir comment sélectionner la dernière ligne du cadre, consultez Fonction LAST window.

Syntaxe

```
FIRST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère.

IGNORE NULLS

Lorsque cette option est utilisée avec FIRST, la fonction renvoie la première valeur du cadre qui n'est pas NULL (ou NULL si toutes les valeurs sont NULL).

RESPECT NULLS

Indique que les valeurs nulles AWS Clean Rooms doivent être incluses dans la détermination de la ligne à utiliser. La clause RESPECT NULLS est prise en charge par défaut, si vous ne spécifiez pas IGNORE NULLS.

OVER

Présente les clauses de fenêtrage de la fonction.

PARTITION BY expr_list

Définit la fenêtre de la fonction en termes d'une ou de plusieurs expressions.

ORDER BY order_list

Trie les lignes dans chaque partition. Si aucune clause PARTITION BY n'est spécifiée, ORDER BY trie toute la table. Si vous spécifiez une clause ORDER BY, vous devez également spécifier une frame_clause.

Les résultats de la fonction FIRST dépendent de l'ordre des données. Les résultats sont non déterministes dans les cas suivants :

- Quand aucune clause ORDER BY n'est spécifiée et qu'une partition contient deux valeurs différentes pour une expression
- Lorsque l'expression a des valeurs différentes qui correspondent à la même valeur dans la liste ORDER BY.

frame_clause

Si une clause ORDER BY est utilisée pour une fonction d'agrégation, une clause de cadre explicite est requise. La clause de cadre affine l'ensemble de lignes dans la fenêtre d'une fonction, en incluant ou en excluant des ensembles de lignes du résultat ordonné. La clause de cadre se compose du mot-clé ROWS et des spécificateurs associés. Consultez Récapitulatif de la syntaxe de la fonction de fenêtrage.

Type de retour

Ces fonctions prennent en charge les expressions qui utilisent des types de AWS Clean Rooms données primitifs. Le type de retour est identique au type de données de l'expression.

Exemples

L'exemple suivant renvoie le nombre de places de chaque site dans la table VENUE, avec les résultats classés par capacité (d'élevée à faible). La fonction FIRST permet de sélectionner le nom du lieu qui correspond à la première ligne du cadre : dans ce cas, la rangée avec le plus grand nombre de places. Les résultats sont partitionnés par État, lorsque la valeur VENUESTATE change, une nouvelle première valeur est donc sélectionnée. Le cadre de fenêtrage est illimité. La même première valeur est donc sélectionnée pour chaque ligne de chaque partition.

Pour la Californie, `Qualcomm Stadium` possède le plus grand nombre de places (`70561`), ce nom est donc la première valeur de toutes les lignes dans la partition `CA`.

```
select venuestate, venueseats, venuename,
first(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |          venuename            |         first
-----------+------------+-------------------------------
+------------------------------
CA         |      70561 | Qualcomm Stadium              | Qualcomm Stadium
CA         |      69843 | Monster Park                  | Qualcomm Stadium
CA         |      63026 | McAfee Coliseum               | Qualcomm Stadium
CA         |      56000 | Dodger Stadium                | Qualcomm Stadium
CA         |      45050 | Angel Stadium of Anaheim      | Qualcomm Stadium
CA         |      42445 | PETCO Park                    | Qualcomm Stadium
CA         |      41503 | AT&T Park                     | Qualcomm Stadium
CA         |      22000 | Shoreline Amphitheatre        | Qualcomm Stadium
CO         |      76125 | INVESCO Field                 | INVESCO Field
CO         |      50445 | Coors Field                   | INVESCO Field
DC         |      41888 | Nationals Park                | Nationals Park
FL         |      74916 | Dolphin Stadium               | Dolphin Stadium
FL         |      73800 | Jacksonville Municipal Stadium | Dolphin Stadium
FL         |      65647 | Raymond James Stadium         | Dolphin Stadium
FL         |      36048 | Tropicana Field               | Dolphin Stadium
...
```

Fonction de fenêtrage FIRST_VALUE

Étant donné un ensemble de lignes ordonné, FIRST_VALUE renvoie la valeur de l'expression spécifiée concernant la première ligne du cadre de fenêtrage d'un ensemble de lignes ordonné.

Pour savoir comment sélectionner la dernière ligne du cadre, consultez Fonction de fenêtrage LAST_VALUE.

Syntaxe

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
```

```
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère.

IGNORE NULLS

Lorsque cette option est utilisée avec FIRST_VALUE, la fonction renvoie la première valeur du cadre qui n'est pas NULL (ou NULL si toutes les valeurs sont NULL).

RESPECT NULLS

Indique que les valeurs nulles AWS Clean Rooms doivent être incluses dans la détermination de la ligne à utiliser. La clause RESPECT NULLS est prise en charge par défaut, si vous ne spécifiez pas IGNORE NULLS.

OVER

Présente les clauses de fenêtrage de la fonction.

PARTITION BY expr_list

Définit la fenêtre de la fonction en termes d'une ou de plusieurs expressions.

ORDER BY order_list

Trie les lignes dans chaque partition. Si aucune clause PARTITION BY n'est spécifiée, ORDER BY trie toute la table. Si vous spécifiez une clause ORDER BY, vous devez également spécifier une frame_clause.

Les résultats de la fonction FIRST_VALUE dépendent de l'ordre des données. Les résultats sont non déterministes dans les cas suivants :

- Quand aucune clause ORDER BY n'est spécifiée et qu'une partition contient deux valeurs différentes pour une expression
- Lorsque l'expression a des valeurs différentes qui correspondent à la même valeur dans la liste ORDER BY.

frame_clause

Si une clause ORDER BY est utilisée pour une fonction d'agrégation, une clause de cadre explicite est requise. La clause de cadre affine l'ensemble de lignes dans la fenêtre d'une fonction, en incluant ou en excluant des ensembles de lignes du résultat ordonné. La clause de cadre se compose du mot-clé ROWS et des spécificateurs associés. Consultez Récapitulatif de la syntaxe de la fonction de fenêtrage.

Type de retour

Ces fonctions prennent en charge les expressions qui utilisent des types de AWS Clean Rooms données primitifs. Le type de retour est identique au type de données de l'expression.

Exemples

L'exemple suivant renvoie le nombre de places de chaque site dans la table VENUE, avec les résultats classés par capacité (d'élevée à faible). La fonction FIRST_VALUE permet de sélectionner le nom du lieu qui correspond à la première ligne du cadre : dans le cas présent, la ligne comportant le plus grand nombre de places. Les résultats sont partitionnés par État, lorsque la valeur VENUESTATE change, une nouvelle première valeur est donc sélectionnée. Le cadre de fenêtrage est illimité. La même première valeur est donc sélectionnée pour chaque ligne de chaque partition.

Pour la Californie, `Qualcomm Stadium` possède le plus grand nombre de places (`70561`), ce nom est donc la première valeur de toutes les lignes dans la partition `CA`.

```
select venuestate, venueseats, venuename,
first_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |          venuename            |         first_value
-----------+------------+-------------------------------
+-----------------------------
CA         |      70561 | Qualcomm Stadium              | Qualcomm Stadium
CA         |      69843 | Monster Park                  | Qualcomm Stadium
CA         |      63026 | McAfee Coliseum               | Qualcomm Stadium
CA         |      56000 | Dodger Stadium                | Qualcomm Stadium
```

```
CA         |      45050 | Angel Stadium of Anaheim       | Qualcomm Stadium
CA         |      42445 | PETCO Park                     | Qualcomm Stadium
CA         |      41503 | AT&T Park                      | Qualcomm Stadium
CA         |      22000 | Shoreline Amphitheatre         | Qualcomm Stadium
CO         |      76125 | INVESCO Field                  | INVESCO Field
CO         |      50445 | Coors Field                    | INVESCO Field
DC         |      41888 | Nationals Park                 | Nationals Park
FL         |      74916 | Dolphin Stadium                | Dolphin Stadium
FL         |      73800 | Jacksonville Municipal Stadium | Dolphin Stadium
FL         |      65647 | Raymond James Stadium          | Dolphin Stadium
FL         |      36048 | Tropicana Field                | Dolphin Stadium
...
```

Fonction de fenêtrage LAG

La fonction de fenêtrage LAG renvoie les valeurs pour une ligne avec un décalage donné au-dessus (avant) de la ligne actuelle dans la partition.

Syntaxe

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Arguments

value_expr

Colonne cible ou expression sur laquelle la fonction opère.

offset

Paramètre facultatif qui spécifie le nombre de lignes avant la ligne actuelle pour lesquelles renvoyer des valeurs. Le décalage peut être un nombre entier constant ou une expression qui a pour valeur un nombre entier. Si vous ne spécifiez pas de décalage, AWS Clean Rooms utilisez 1 comme valeur par défaut. Un décalage de `0` indique la ligne actuelle.

IGNORE NULLS

Spécification facultative qui indique que les valeurs nulles AWS Clean Rooms doivent être ignorées lors de la détermination de la ligne à utiliser. Les valeurs NULL sont incluses si IGNORE NULLS n'est pas répertorié.

> **Note**
>
> Vous pouvez utiliser une expression NVL ou COALESCE pour remplacer les valeurs NULL par une autre valeur.

RESPECT NULLS

Indique que les valeurs nulles AWS Clean Rooms doivent être incluses dans la détermination de la ligne à utiliser. La clause RESPECT NULLS est prise en charge par défaut, si vous ne spécifiez pas IGNORE NULLS.

OVER

Spécifie le partitionnement de fenêtrage et d'ordonnancement. La clause OVER ne peut pas contenir de spécification de cadre de fenêtrage.

PARTITION BY window_partition

Argument facultatif qui définit la plage d'enregistrements de chaque groupe de la clause OVER.

ORDER BY window_ordering

Trie les lignes dans chaque partition.

La fonction de fenêtre LAG prend en charge les expressions qui utilisent n'importe quel type de AWS Clean Rooms données. Le type de retour est identique au type value_expr.

Exemples

L'exemple suivant présente la quantité de billets vendus à l'acheteur ayant l'ID d'acheteur 3 et l'heure à laquelle l'acheteur 3 a acheté les billets. Pour comparer chaque vente à la vente précédente de l'acheteur 3, la requête renvoie la quantité précédente vendue pour chaque vente. Dans la mesure où il n'y a aucun achat avant le 16/01/2008, la première quantité précédente vendue a la valeur 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)
```

Fonction LAST window

À partir d'un ensemble ordonné de lignes, la fonction LAST renvoie la valeur de l'expression par rapport à la dernière ligne du cadre.

Pour savoir comment sélectionner la première ligne du cadre, consultez Fonction de fenêtre FIRST.

Syntaxe

```
LAST( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère.

IGNORE NULLS

La fonction renvoie la dernière valeur du cadre qui n'est pas NULL (ou NULL si toutes les valeurs sont NULL).

RESPECT NULLS

Indique que les valeurs nulles AWS Clean Rooms doivent être incluses dans la détermination de la ligne à utiliser. La clause RESPECT NULLS est prise en charge par défaut, si vous ne spécifiez pas IGNORE NULLS.

OVER

Présente les clauses de fenêtrage de la fonction.

PARTITION BY expr_list

Définit la fenêtre de la fonction en termes d'une ou de plusieurs expressions.

ORDER BY order_list

Trie les lignes dans chaque partition. Si aucune clause PARTITION BY n'est spécifiée, ORDER BY trie toute la table. Si vous spécifiez une clause ORDER BY, vous devez également spécifier une frame_clause.

Les résultats dépendent de l'ordre des données. Les résultats sont non déterministes dans les cas suivants :

- Quand aucune clause ORDER BY n'est spécifiée et qu'une partition contient deux valeurs différentes pour une expression
- Lorsque l'expression a des valeurs différentes qui correspondent à la même valeur dans la liste ORDER BY.

frame_clause

Si une clause ORDER BY est utilisée pour une fonction d'agrégation, une clause de cadre explicite est requise. La clause de cadre affine l'ensemble de lignes dans la fenêtre d'une fonction, en incluant ou en excluant des ensembles de lignes du résultat ordonné. La clause de cadre se compose du mot-clé ROWS et des spécificateurs associés. Consultez Récapitulatif de la syntaxe de la fonction de fenêtrage.

Type de retour

Ces fonctions prennent en charge les expressions qui utilisent des types de AWS Clean Rooms données primitifs. Le type de retour est identique au type de données de l'expression.

Exemples

L'exemple suivant renvoie le nombre de places de chaque site dans la table VENUE, avec les résultats classés par capacité (d'élevée à faible). La fonction LAST permet de sélectionner le nom de la salle correspondant à la dernière ligne du cadre : dans ce cas, la rangée comportant le moins de places. Les résultats étant partitionnés par État, lorsque la valeur de VENUESTATE change, une nouvelle dernière valeur est sélectionnée. Comme le cadre de fenêtrage est illimité, la même dernière valeur est sélectionnée pour chaque ligne de chaque partition.

Pour la Californie, `Shoreline Amphitheatre` est renvoyé pour chaque ligne de la partition, car il possède le plus petit nombre de places (`22000`).

```
select venuestate, venueseats, venuename,
last(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |          venuename            |          last
-----------+------------+-------------------------------
+------------------------------
CA         |      70561 | Qualcomm Stadium              | Shoreline Amphitheatre
CA         |      69843 | Monster Park                  | Shoreline Amphitheatre
CA         |      63026 | McAfee Coliseum               | Shoreline Amphitheatre
CA         |      56000 | Dodger Stadium                | Shoreline Amphitheatre
CA         |      45050 | Angel Stadium of Anaheim      | Shoreline Amphitheatre
CA         |      42445 | PETCO Park                    | Shoreline Amphitheatre
CA         |      41503 | AT&T Park                     | Shoreline Amphitheatre
CA         |      22000 | Shoreline Amphitheatre        | Shoreline Amphitheatre
CO         |      76125 | INVESCO Field                 | Coors Field
CO         |      50445 | Coors Field                   | Coors Field
DC         |      41888 | Nationals Park                | Nationals Park
FL         |      74916 | Dolphin Stadium               | Tropicana Field
FL         |      73800 | Jacksonville Municipal Stadium | Tropicana Field
FL         |      65647 | Raymond James Stadium         | Tropicana Field
FL         |      36048 | Tropicana Field               | Tropicana Field
...
```

Fonction de fenêtrage LAST_VALUE

Pour un ensemble de lignes ordonnées, la fonction LAST_VALUE renvoie la valeur de l'expression par rapport à la dernière ligne du cadre.

Pour savoir comment sélectionner la première ligne du cadre, consultez Fonction de fenêtrage FIRST_VALUE.

Syntaxe

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
```

```
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

Arguments

expression

Colonne cible ou expression sur laquelle la fonction opère.

IGNORE NULLS

La fonction renvoie la dernière valeur du cadre qui n'est pas NULL (ou NULL si toutes les valeurs sont NULL).

RESPECT NULLS

Indique que les valeurs nulles AWS Clean Rooms doivent être incluses dans la détermination de la ligne à utiliser. La clause RESPECT NULLS est prise en charge par défaut, si vous ne spécifiez pas IGNORE NULLS.

OVER

Présente les clauses de fenêtrage de la fonction.

PARTITION BY expr_list

Définit la fenêtre de la fonction en termes d'une ou de plusieurs expressions.

ORDER BY order_list

Trie les lignes dans chaque partition. Si aucune clause PARTITION BY n'est spécifiée, ORDER BY trie toute la table. Si vous spécifiez une clause ORDER BY, vous devez également spécifier une frame_clause.

Les résultats dépendent de l'ordre des données. Les résultats sont non déterministes dans les cas suivants :

- Quand aucune clause ORDER BY n'est spécifiée et qu'une partition contient deux valeurs différentes pour une expression
- Lorsque l'expression a des valeurs différentes qui correspondent à la même valeur dans la liste ORDER BY.

frame_clause

Si une clause ORDER BY est utilisée pour une fonction d'agrégation, une clause de cadre explicite est requise. La clause de cadre affine l'ensemble de lignes dans la fenêtre d'une fonction, en incluant ou en excluant des ensembles de lignes du résultat ordonné. La clause de cadre se compose du mot-clé ROWS et des spécificateurs associés. Consultez Récapitulatif de la syntaxe de la fonction de fenêtrage.

Type de retour

Ces fonctions prennent en charge les expressions qui utilisent des types de AWS Clean Rooms données primitifs. Le type de retour est identique au type de données de l'expression.

Exemples

L'exemple suivant renvoie le nombre de places de chaque site dans la table VENUE, avec les résultats classés par capacité (d'élevée à faible). La fonction LAST_VALUE permet de sélectionner le nom du lieu qui correspond à la dernière ligne du cadre : dans le cas présent, il s'agit de la ligne présentant le plus petit nombre de places. Les résultats étant partitionnés par État, lorsque la valeur de VENUESTATE change, une nouvelle dernière valeur est sélectionnée. Comme le cadre de fenêtrage est illimité, la même dernière valeur est sélectionnée pour chaque ligne de chaque partition.

Pour la Californie, `Shoreline Amphitheatre` est renvoyé pour chaque ligne de la partition, car il possède le plus petit nombre de places (`22000`).

```
select venuestate, venueseats, venuename,
last_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |          venuename            |          last_value
-----------+------------+-------------------------------
+-----------------------------
CA         |      70561 | Qualcomm Stadium              | Shoreline Amphitheatre
CA         |      69843 | Monster Park                  | Shoreline Amphitheatre
CA         |      63026 | McAfee Coliseum               | Shoreline Amphitheatre
CA         |      56000 | Dodger Stadium                | Shoreline Amphitheatre
```

```
CA         |      45050 | Angel Stadium of Anaheim       | Shoreline Amphitheatre
CA         |      42445 | PETCO Park                     | Shoreline Amphitheatre
CA         |      41503 | AT&T Park                      | Shoreline Amphitheatre
CA         |      22000 | Shoreline Amphitheatre         | Shoreline Amphitheatre
CO         |      76125 | INVESCO Field                  | Coors Field
CO         |      50445 | Coors Field                    | Coors Field
DC         |      41888 | Nationals Park                 | Nationals Park
FL         |      74916 | Dolphin Stadium                | Tropicana Field
FL         |      73800 | Jacksonville Municipal Stadium | Tropicana Field
FL         |      65647 | Raymond James Stadium          | Tropicana Field
FL         |      36048 | Tropicana Field                | Tropicana Field
...
```

Fonction de fenêtrage LEAD

La fonction de fenêtrage LEAD renvoie les valeurs pour une ligne avec un décalage donné au-dessous (après) de la ligne actuelle dans la partition.

Syntaxe

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

Arguments

value_expr

Colonne cible ou expression sur laquelle la fonction opère.

offset

Paramètre facultatif qui spécifie le nombre de lignes sous la ligne actuelle pour lesquelles renvoyer des valeurs. Le décalage peut être un nombre entier constant ou une expression qui a pour valeur un nombre entier. Si vous ne spécifiez pas de décalage, AWS Clean Rooms utilisez 1 comme valeur par défaut. Un décalage de `0` indique la ligne actuelle.

IGNORE NULLS

Spécification facultative qui indique que les valeurs nulles AWS Clean Rooms doivent être ignorées lors de la détermination de la ligne à utiliser. Les valeurs NULL sont incluses si IGNORE NULLS n'est pas répertorié.

> **Note**
>
> Vous pouvez utiliser une expression NVL ou COALESCE pour remplacer les valeurs NULL par une autre valeur.

RESPECT NULLS

Indique que les valeurs nulles AWS Clean Rooms doivent être incluses dans la détermination de la ligne à utiliser. La clause RESPECT NULLS est prise en charge par défaut, si vous ne spécifiez pas IGNORE NULLS.

OVER

Spécifie le partitionnement de fenêtrage et d'ordonnancement. La clause OVER ne peut pas contenir de spécification de cadre de fenêtrage.

PARTITION BY window_partition

Argument facultatif qui définit la plage d'enregistrements de chaque groupe de la clause OVER.

ORDER BY window_ordering

Trie les lignes dans chaque partition.

La fonction de fenêtre LEAD prend en charge les expressions qui utilisent n'importe quel type de AWS Clean Rooms données. Le type de retour est identique au type value_expr.

Exemples

L'exemple suivant fournit la commission pour les événements de la table SALES pour les billets ont été vendus sur le 1er janvier 2008 et le 2 janvier 2008 et la commission payée pour la vente des billets de la vente suivante.

```
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)
```

Fonction de fenêtrage PERCENT_RANK

Calcule le rang en pourcentage d'une ligne donnée. Le rang en pourcentage est déterminé à l'aide de la formule suivante :

```
(x - 1) / (the number of rows in the window or partition - 1)
```

où x est le rang de la ligne actuelle. Le jeu de données suivant illustre l'utilisation de cette formule :

```
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
```

La plage de valeur de retour est comprise entre 0 et 1, inclus. La première ligne de n'importe quel jeu dispose d'une fonction PERCENT_RANK spécifiée sur 0.

Syntaxe

```
PERCENT_RANK ()
```

```
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

Arguments

( )

La fonction ne prend pas d'arguments, mais les parenthèses vides sont obligatoires.

OVER

Clause qui spécifie le partitionnement de fenêtrage. La clause OVER ne peut pas contenir de spécification de cadre de fenêtrage.

PARTITION BY partition_expression

Facultatif. Expression qui définit la plage d'enregistrements de chaque groupe dans la clause OVER.

ORDER BY order_list

Facultatif. Expression permettant de calculer le rang en pourcentage. L'expression doit disposer d'un type de données numériques ou être convertible implicitement en une. Si ORDER BY n'est pas spécifié, la valeur de retour est 0 pour toutes les lignes.

Si ORDER BY ne génère pas d'ordonnancement unique, l'ordre des lignes est non déterministe. Pour de plus amples informations, veuillez consulter Ordonnancement unique des données pour les fonctions de fenêtrage.

Type de retour

FLOAT8

Exemples

L'exemple suivant calcule le rang en pourcentage des volumes de ventes de chaque vendeur :

```
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
```

Pour obtenir une description de la table WINSALES, consultez Exemple de tableau contenant des exemples de fonctions de fenêtrage.

Fonction de fenêtrage RANK

La fonction de fenêtrage RANK détermine le rang d'une valeur dans un groupe de valeurs, en fonction de l'expression ORDER BY dans la clause OVER. Si la clause PARTITION BY facultative est présente, les rangs sont réinitialisés pour chaque groupe de lignes. Les lignes présentant des valeurs égales pour les critères de classement reçoivent le même classement. AWS Clean Roomsajoute le nombre de lignes égales au rang égal pour calculer le rang suivant. Les rangs peuvent donc ne pas être des nombres consécutifs. Par exemple, si deux lignes sont classées 1, le prochain rang est 3.

La fonction RANK diffère de Fonction de fenêtrage DENSE_RANK sur un point : pour DENSE_RANK, si deux lignes ou plus sont à égalité, il n'y a aucun écart dans la séquence des valeurs classées. Par exemple, si deux lignes sont classées 1, le prochain rang est 2.

Vous pouvez avoir des fonctions de rang avec différentes clauses PARTITION BY et ORDER BY dans la même requête.

Syntaxe

```
RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Arguments

( )

La fonction ne prend pas d'arguments, mais les parenthèses vides sont obligatoires.

OVER

Clauses de fenêtrage de la fonction RANK.

PARTITION BY expr_list

Facultatif. Une ou plusieurs expressions qui définissent le fenêtrage.

ORDER BY order_list

Facultatif. Définit les colonnes sur lesquelles les valeurs de rang sont basées. Si aucune clause PARTITION BY n'est spécifiée, ORDER BY utilise toute la table. Si ORDER BY n'est pas spécifié, la valeur de retour est 1 pour toutes les lignes.

Si ORDER BY ne génère pas d'ordonnancement unique, l'ordre des lignes est non déterministe. Pour de plus amples informations, veuillez consulter Ordonnancement unique des données pour les fonctions de fenêtrage.

Type de retour

INTEGER

Exemples

L'exemple suivant montre le classement de la table selon la quantité vendue (croissant par défaut) et l'affectation d'un rang à chaque ligne. 1 est la valeur classée la plus élevée. Les résultats sont triés une fois que les résultats de la fonction de fenêtrage sont appliqués:

```
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)
```

Notez que la clause externe ORDER BY de cet exemple inclut les colonnes 2 et 1 pour garantir que les résultats AWS Clean Rooms sont systématiquement triés chaque fois que cette requête est exécutée. Par exemple, les lignes avec les ventes IDs 10001 et 10006 ont des valeurs QTY et RNK identiques. L'ordonnancement du résultat final défini par la colonne 1 garantit que la ligne 10001 précède toujours 10006. Pour obtenir une description de la table WINSALES, consultez Exemple de tableau contenant des exemples de fonctions de fenêtrage.

Dans l'exemple suivant, l'ordonnancement est inversé pour la fonction de fenêtrage (`order by qty desc`). A présent, la valeur de rang la plus élevée s'applique à la valeur QTY la plus élevée.

```
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)
```

Pour obtenir une description de la table WINSALES, consultez Exemple de tableau contenant des exemples de fonctions de fenêtrage.

L'exemple suivant montre le partitionnement de la table en fonction de chaque SELLERID, le classement de chaque partition selon la quantité (par ordre décroissant) et l'affectation d'un rang à chaque ligne. Les résultats sont triés une fois que les résultats de la fonction de fenêtrage sont appliqués.

```
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)
```

Fonction de fenêtrage ROW_NUMBER

Détermine le nombre ordinal de la ligne actuelle au sein d'un groupe de lignes, à partir de 1, en fonction de l'expression ORDER BY de la clause OVER. Si la clause PARTITION BY facultative est présente, les nombres ordinaux sont réinitialisés pour chaque groupe de lignes. Les lignes avec des valeurs égales pour les expressions ORDER BY reçoivent des numéros de lignes différentes de manière non déterministe.

Syntaxe

```
ROW_NUMBER () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

Arguments

( )

La fonction ne prend pas d'arguments, mais les parenthèses vides sont obligatoires.

OVER

Clauses de fenêtrage pour la fonction ROW_NUMBER.

PARTITION BY expr_list

Facultatif. Une ou plusieurs expressions qui définissent la fonction ROW_NUMBER.

ORDER BY order_list

Facultatif. Expression qui définit les colonnes sur lesquelles sont basées les numéros de lignes. Si aucune clause PARTITION BY n'est spécifiée, ORDER BY utilise toute la table.

Si ORDER BY ne génère pas d'ordonnancement unique ou n'est pas spécifiée, l'ordre des lignes est non déterministe. Pour de plus amples informations, veuillez consulter Ordonnancement unique des données pour les fonctions de fenêtrage.

Type de retour

BIGINT

Exemples

L'exemple suivant présente la partition de la table par SELLERID et classe chaque partition par QTY (en ordre croissant), puis affecte un numéro de ligne à chaque ligne. Les résultats sont triés une fois que les résultats de la fonction de fenêtrage sont appliqués.

```
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)
```

Pour obtenir une description de la table WINSALES, consultez Exemple de tableau contenant des exemples de fonctions de fenêtrage.

AWS Clean Rooms Conditions SQL de Spark

Les conditions sont des déclarations d'une ou plusieurs expressions et opérateurs logiques dont la valeur est vraie, fausse ou inconnue. Les conditions sont également appelées parfois prédicats.

Syntaxe

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

> **Note**
>
> Toutes les comparaisons de chaîne et correspondances du modèle LIKE sont sensibles à la casse. Par exemple, « A » et « a » ne correspondent pas. Cependant, vous pouvez effectuer une correspondance de modèle non sensible à la casse à l'aide du prédicat ILIKE.

Les conditions SQL suivantes sont prises en charge dans AWS Clean Rooms Spark SQL.

Rubriques

- Opérateurs de comparaison

- Conditions logiques
- Conditions de correspondance de modèles
- Condition de plage BETWEEN
- Condition null
- Condition EXISTS
- Condition IN

Opérateurs de comparaison

Les conditions de comparaison établissent des relations logiques entre deux valeurs. Toutes les conditions de comparaison sont des opérateurs binaires avec un type de retour booléen.

AWS Clean Rooms Spark SQL prend en charge les opérateurs de comparaison décrits dans le tableau suivant.

Opérateur	Syntaxe	Description
!	`!expression`	L'`NOT`opérateur logique. Utilisé pour annuler une expressio n booléenne, c'est-à-dire qu'elle renvoie le contraire de la valeur de l'expression. Le ! L'opérateur peut également être combiné avec d'autres opérateurs logiques, tels que AND et OR, pour créer des expressions booléennes plus complexes.
<	`a < b`	L'opérateur inférieur à la comparaison. Utilisé pour comparer deux valeurs et déterminer si la valeur de gauche est inférieure à la valeur de droite.

Opérateur	Syntaxe	Description
>	`a > b`	L'opérateur supérieur à la comparaison. Permet de comparer deux valeurs et de déterminer si la valeur de gauche est supérieure à celle de droite.
<=	`a <= b`	Opérateur de comparaison inférieur ou égal à. Utilisé pour comparer deux valeurs et renvoie `true` si la valeur de gauche est inférieure ou égale à la valeur de droite, et dans le `false` cas contraire.
>=	`a >= b`	Opérateur de comparaison supérieur ou égal à. Utilisé pour comparer deux valeurs et déterminer si la valeur de gauche est supérieure ou égale à la valeur de droite.
=	`a = b`	L'opérateur de comparaison d'égalité, qui compare deux valeurs et renvoie `true` si elles sont égales, et `false` sinon.
<> ou **!=**	`a <> b` ou `a != b`	L'opérateur de comparaison non égal à, qui compare deux valeurs et renvoie `true` si elles ne sont pas égales, et dans le `false` cas contraire.

Opérateur	Syntaxe	Description
==	`a == b`	L'opérateur de comparaison d'égalité standard, qui compare deux valeurs et renvoie `true` si elles sont égales, et `false` sinon. **Note** L'opérateur == distingue les majuscules et minuscules lors de la comparaison de valeurs de chaînes. Si vous devez effectuer une comparaison sans distinction majuscules/minuscules, vous pouvez utiliser des fonctions telles que UPPER () ou LOWER () pour convertir les valeurs au même majuscule avant la comparaison.

Exemples

Voici quelques exemples simples de conditions de comparaison :

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

La requête suivante renvoie les valeurs d'identification de tous les écureuils qui ne cherchent pas de nourriture actuellement.

```
SELECT id FROM squirrels
WHERE !is_foraging
```

La requête suivante renvoie les sites de plus de 10 000 places dans le tableau VENUE :

```
select venueid, venuename, venueseats from venue
where venueseats > 10000
order by venueseats desc;

venueid |           venuename            | venueseats
---------+--------------------------------+------------
83 | FedExField                     |      91704
 6 | New York Giants Stadium        |      80242
79 | Arrowhead Stadium              |      79451
78 | INVESCO Field                  |      76125
69 | Dolphin Stadium                |      74916
67 | Ralph Wilson Stadium           |      73967
76 | Jacksonville Municipal Stadium |      73800
89 | Bank of America Stadium        |      73298
72 | Cleveland Browns Stadium       |      73200
86 | Lambeau Field                  |      72922
...
(57 rows)
```

Cet exemple sélectionne les utilisateurs (USERID) de la table USERS qui aiment la musique rock :

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
13
16
(5 rows)
```

Cet exemple sélectionne les utilisateurs (USERID) de la table USERS pour lesquels on ignore s’ils aiment la musique rock :

```
select firstname, lastname, likerock
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
----------+----------+----------
Rafael    | Taylor   |
Vladimir  | Humphrey |
Barry     | Roy      |
Tamekah   | Juarez   |
Mufutau   | Watkins  |
Naida     | Calderon |
Anika     | Huff     |
Bruce     | Beck     |
Mallory   | Farrell  |
Scarlett  | Mayer    |
(10 rows
```

Exemples avec une colonne TIME

La table d'exemple TIME_TEST suivante comporte une colonne TIME_VAL (type TIME) dans laquelle trois valeurs ont été insérées.

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

L'exemple suivant extrait les heures de chaque timetz_val.

```
select time_val from time_test where time_val < '3:00';
   time_val
---------------
 00:00:00.5550
 00:58:00
```

L’exemple suivant compare deux littéraux de type heure.

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
 t
```

Exemples avec une colonne TIMETZ

L'exemple de tableau TIMETZ_TEST suivant comporte une colonne TIMETZ_VAL (type TIMETZ) dans laquelle trois valeurs ont été insérées.

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

L'exemple suivant sélectionne uniquement les valeurs TIMETZ inférieures à `3:00:00 UTC`. La comparaison est effectuée après la conversion de la valeur en UTC.

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

L'exemple suivant compare deux littéraux TIMETZ. Le fuseau horaire n'est pas pris en compte pour la comparaison.

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

 ?column?
----------
 t
```

Conditions logiques

Les conditions logiques combinent le résultat de deux conditions pour produire un résultat unique. Toutes les conditions logiques sont des opérateurs binaires avec un type de retour booléen.

Syntaxe

```
expression
{ AND | OR }
expression
NOT expression
```

Les conditions logiques utilisent une logique booléenne à trois valeurs où la valeur nulle représente une relation inconnue. Le tableau suivant décrit les résultats des conditions logiques, où E1 et E2 représentent des expressions :

E1	E2	E1 AND E2	E1 OR E2	NOT E2
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

L'opérateur NOT est analysé avant AND et l'opérateur AND est évalué avant l'opérateur OR. Les parenthèses utilisées peuvent remplacer cet ordre d'évaluation par défaut.

Exemples

L'exemple suivant retourne USERID et USERNAME de la table USERS où l'utilisateur aime à la fois Las Vegas et les sports :

```
select userid, username from users
```

```
where likevegas = 1 and likesports = 1
order by userid;

userid | username
--------+----------
1 | JSG99FHE
67 | TWU10MZT
87 | DUF19VXU
92 | HYP36WEQ
109 | FPL38HZK
120 | DMJ24GUZ
123 | QZR22XGQ
130 | ZQC82ALK
133 | LBN45WCH
144 | UCX04JKN
165 | TEY68OEB
169 | AYQ83HGO
184 | TVX65AZX
...
(2128 rows)
```

L'exemple suivant retourne USERID et USERNAME de la table USERS où l'utilisateur aime Las Vegas, ou les sports, ou les deux. Cette requête renvoie toutes les données de sortie de l'exemple précédent, plus les utilisateurs qui aiment uniquement Las Vegas ou le sport.

```
select userid, username from users
where likevegas = 1 or likesports = 1
order by userid;

userid | username
--------+----------
1 | JSG99FHE
2 | PGL08LJI
3 | IFT66TXU
5 | AEB55QTM
6 | NDQ15VBM
9 | MSD36KVR
10 | WKW41AIW
13 | QTF33MCG
15 | OWU78MTR
16 | ZMG93CDD
22 | RHT62AGI
27 | KOY02CVE
```

```
29 | HUH27PKK
...
(18968 rows)
```

La requête suivante utilise des parenthèses autour de la condition OR pour trouver les salles de New York ou de Californie où Macbeth a été joué :

```
select distinct venuename, venuecity
from venue join event on venue.venueid=event.venueid
where (venuestate = 'NY' or venuestate = 'CA') and eventname='Macbeth'
order by 2,1;

venuename                |   venuecity
-----------------------------------------+---------------
Geffen Playhouse                         | Los Angeles
Greek Theatre                            | Los Angeles
Royce Hall                               | Los Angeles
American Airlines Theatre                | New York City
August Wilson Theatre                    | New York City
Belasco Theatre                          | New York City
Bernard B. Jacobs Theatre                | New York City
...
```

La suppression des parenthèses de cet exemple modifie la logique et les résultats de la requête.

Les exemples suivants utilisent l'opérateur NOT :

```
select * from category
where not catid=1
order by 1;

catid | catgroup |  catname  |                 catdesc
-------+----------+-----------+--------------------------------------------
2 | Sports   | NHL       | National Hockey League
3 | Sports   | NFL       | National Football League
4 | Sports   | NBA       | National Basketball Association
5 | Sports   | MLS       | Major League Soccer
...
```

L'exemple suivant utilise une condition NOT suivie d'une condition AND :

```
select * from category
```

```
where (not catid=1) and catgroup='Sports'
order by catid;

catid | catgroup | catname |            catdesc
-------+----------+---------+--------------------------------
2 | Sports   | NHL     | National Hockey League
3 | Sports   | NFL     | National Football League
4 | Sports   | NBA     | National Basketball Association
5 | Sports   | MLS     | Major League Soccer
(4 rows)
```

Conditions de correspondance de modèles

Un opérateur de correspondance de modèles recherche dans une chaîne un modèle spécifié dans l'expression conditionnelle et renvoie vrai ou faux selon qu'il trouve une correspondance. AWS Clean Rooms Spark SQL utilise les méthodes suivantes pour faire correspondre les modèles :

- Expressions LIKE

  L’opérateur LIKE compare une expression de chaîne, comme un nom de colonne, avec un modèle qui utilise les caractères génériques % (pourcentage) et _ (soulignement). La correspondance de modèle LIKE couvre toute la chaîne. LIKE effectue une correspondance en distinguant majuscules et minuscules.

Rubriques

- LIKE
- RLIKE

LIKE

L’opérateur LIKE compare une expression de chaîne, comme un nom de colonne, avec un modèle qui utilise les caractères génériques % (pourcentage) et _ (soulignement). La correspondance de modèle LIKE couvre toute la chaîne. Pour faire correspondre une séquence à n’importe quel emplacement au sein d’une chaîne, le modèle doit commencer et finir par un signe %.

LIKE fait la distinction majuscules/minuscules.

Syntaxe

```
expression [ NOT ] LIKE | pattern [ ESCAPE 'escape_char' ]
```

Arguments

expression

Expression de caractère UTF-8 valide, comme un nom de colonne.

LIKE

LIKE effectue une correspondance sensible à la casse. Pour effectuer une correspondance de modèle non sensible à la casse pour les caractères codés sur plusieurs octets, utilisez la fonction LOWER sur expression et pattern avec une condition LIKE.

Contrairement aux prédicats de comparaison, tels que = et <>, les prédicats LIKE n'ignorent pas implicitement les espaces de fin. Pour ignorer les espaces de fin, utilisez RTRIM ou convertissez explicitement une colonne CHAR en VARCHAR.

L'~~opérateur est équivalent à LIKE. De plus, l'!~~opérateur est équivalent à NOT LIKE.

pattern

Expression de caractère UTF-8 valide avec le modèle à mettre en correspondance.

escape_char

Expression de caractère qui utilise une séquence d'échappement pour les méta-caractères du modèle. La valeur par défaut est deux barres obliques inverses ('\\ »).

Si pattern ne contient pas de méta-caractères, le modèle représente uniquement la chaîne elle-même ; dans ce cas, LIKE agit de même que l'opérateur d'égalité.

Les expressions de caractère peuvent avoir CHAR ou VARCHAR comme type de données. En cas de différence, AWS Clean Rooms convertit pattern en type de données expression.

LIKE prend en charge les méta-caractères de correspondance de modèle suivants :

Opérateur	Description
%	Met en correspondance une séquence de zéro ou plusieurs caractères.

Opérateur	Description
_	Met en correspondance un seul caractère.

Exemples

Le tableau suivant montre des exemples de correspondance de modèle avec LIKE :

Expression	Renvoie
`'abc' LIKE 'abc'`	True
`'abc' LIKE 'a%'`	True
`'abc' LIKE '_B_'`	False
`'abc' LIKE 'c%'`	False

L'exemple suivant recherche toutes les villes dont le nom commence par « E » :

```
select distinct city from users
where city like 'E%' order by city;
city
---------------
East Hartford
East Lansing
East Rutherford
East St. Louis
Easthampton
Easton
Eatontown
Eau Claire
...
```

L'exemple suivant recherche les utilisateurs dont le nom contient « ten » :

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
```

```
-------------
Christensen
Wooten
...
```

L'exemple suivant trouve des villes dont les troisième et quatrième caractères sont « ea » . :

```
select distinct city from users where city like '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

L’exemple suivant utilise la chaîne d’échappement par défaut (\\) pour rechercher les chaînes qui incluent « start_» (texte `start` suivi d’un trait de soulignement _) :

```
select tablename, "column" from my_table_def


where "column" like '%start\\_%'
limit 5;

     tablename     |    column
-------------------+---------------
 my_s3client       | start_time
 my_tr_conflict    | xact_start_ts
 my_undone         | undo_start_ts
 my_unload_log     | start_time
 my_vacuum_detail | start_row
(5 rows)
```

L’exemple suivant spécifie « ^ » comme caractère d’échappement, puis utilise ce dernier pour rechercher des chaînes qui incluent « start_ » (texte `start` suivi d’un trait de soulignement _) :

```
select tablename, "column" from my_table_def

where "column" like '%start^_%' escape '^'
```

```
limit 5;

     tablename     |    column
-------------------+---------------
 my_s3client       | start_time
 my_tr_conflict    | xact_start_ts
 my_undone         | undo_start_ts
 my_unload_log     | start_time
 my_vacuum_detail | start_row
(5 rows)
```

RLIKE

L'opérateur RLIKE vous permet de vérifier si une chaîne correspond à un modèle d'expression régulière spécifié.

Renvoie `true` si str correspond`regexp`, ou `false` non.

Syntaxe

```
rlike(str, regexp)
```

Arguments

str

Une expression sous forme de chaîne

expression régulière

Expression sous forme de chaîne. La chaîne regex doit être une expression régulière Java.

Les littéraux de chaîne (y compris les modèles de regex) ne sont pas échappés dans notre analyseur SQL. Par exemple, pour correspondre à « \ abc », une expression régulière pour regexp peut être « ^ \ abc$ ».

Exemples

L'exemple suivant définit la valeur du paramètre `spark.sql.parser.escapedStringLiterals` de configuration sur`true`. Ce paramètre est spécifique au moteur Spark SQL. Le `spark.sql.parser.escapedStringLiterals` paramètre de Spark SQL contrôle la façon

dont l'analyseur SQL gère les chaînes littérales échappées. Lorsqu'il est défini sur`true`, l'analyseur interprète les barres obliques inversées (\) dans les chaînes littérales comme des caractères d'échappement, ce qui vous permet d'inclure des caractères spéciaux tels que des nouvelles lignes, des tabulations et des guillemets dans les valeurs de vos chaînes.

```
SET spark.sql.parser.escapedStringLiterals=true;
spark.sql.parser.escapedStringLiterals  true
```

Par exemple, avec`spark.sql.parser.escapedStringLiterals=true`, vous pouvez utiliser la chaîne littérale suivante dans votre requête SQL :

```
SELECT 'Hello, world!\n'
```

Le caractère de nouvelle ligne \n serait interprété comme un caractère de nouvelle ligne littéral dans la sortie.

L'exemple suivant effectue une correspondance avec un modèle d'expression régulière. Le premier argument est passé à l'opérateur RLIKE. Il s'agit d'une chaîne qui représente le chemin d'un fichier, où le nom d'utilisateur réel est remplacé par le modèle '****'. Le deuxième argument est le modèle d'expression régulière utilisé pour la correspondance. La sortie (`true`) indique que la première chaîne (`'%SystemDrive%\Users\****'`) correspond au modèle d'expression régulière (`'%SystemDrive%\\Users.*'`).

```
SELECT rlike('%SystemDrive%\Users\John', '%SystemDrive%\Users.*');
true
```

Condition de plage BETWEEN

Une condition `BETWEEN` teste les expressions pour l’inclusion dans une plage de valeurs, à l’aide des mots-clés `BETWEEN` et `AND`.

Syntaxe

```
expression [ NOT ] BETWEEN expression AND expression
```

Les expressions peuvent être de type de données numeric, character ou datetime, mais elles doivent être compatibles. La plage est inclusive.

Exemples

Le premier exemple comptabilise le nombre de transactions ayant enregistré des ventes de 2, 3 ou 4 billets :

```
select count(*) from sales
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

La condition de la plage comprend les valeurs de début et de fin.

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

La première expression d'une condition de plage doit être la valeur inférieure et la deuxième expression la valeur supérieure. L'exemple suivant retourne toujours zéro ligne en raison des valeurs des expressions :

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

Cependant, l'application du modificateur NOT inverse la logique et génère le nombre de toutes les lignes :

```
select count(*) from sales
where qtysold not between 4 and 2;
```

```
count
--------
172456
(1 row)
```

La requête suivante retourne une liste des salles avec 20 000 à 50 000 places :

```
select venueid, venuename, venueseats from venue
where venueseats between 20000 and 50000
order by venueseats desc;

venueid |       venuename               | venueseats
---------+-------------------------------+------------
116 | Busch Stadium                 |      49660
106 | Rangers BallPark in Arlington |      49115
96 | Oriole Park at Camden Yards   |      48876
...
(22 rows)
```

L'exemple suivant illustre l'utilisation de BETWEEN pour les valeurs de date :

```
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
```

Notez que même si la plage de BETWEEN est inclusive, les dates ont par défaut une valeur horaire de 00:00:00. La seule ligne valide du 3 janvier pour l'exemple de requête serait une ligne dont saletime est `1/3/2008 00:00:00`.

Condition null

Le NULL tests de condition pour détecter les valeurs nulles, lorsqu'une valeur est manquante ou inconnue.

Syntaxe

```
expression IS [ NOT ] NULL
```

Arguments

expression

N’importe quelle expression telle qu’une colonne.

IS NULL

Condition vraie lorsque la valeur de l’expression est null et fausse quand elle a une valeur.

IS NOT NULL

Condition fausse lorsque la valeur de l’expression est null et vraie quand elle a une valeur.

exemple

Cet exemple indique le nombre de fois où la table SALES contient null dans le champ QTYSOLD :

```
select count(*) from sales
where qtysold is null;
count
-------
0
(1 row)
```

Condition EXISTS

Les conditions EXIST testent l’existence de lignes dans une sous-requête et retournent la valeur true si la requête renvoie au moins une ligne. Si NOT n’est pas spécifié, la condition retourne true si une sous-requête ne renvoie aucune ligne.

Syntaxe

```
[ NOT ] EXISTS (table_subquery)
```

Arguments

EXISTS

Est vraie lorsque la table_subquery retourne au moins une ligne.

NOT EXISTS

Est vraie lorsque la table_subquery ne retourne pas de lignes.

table_subquery

Une sous-requête qui analyse une table avec une ou plusieurs colonnes et une ou plusieurs lignes.

exemple

Cet exemple retourne tous les identificateurs de date, une fois chacun, pour chaque date où une vente a eu lieu :

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
1827
1828
1829
...
```

Condition IN

Une IN condition teste l'appartenance d'une valeur à un ensemble de valeurs ou à une sous-requête.

Syntaxe

```
expression [ NOT ] IN (expr_list | table_subquery)
```

Arguments

expression

Une expression de type numeric, character ou datetime évaluée par rapport à expr_list ou table_subquery et qui doit être compatible avec le type de données de cette liste ou sous-requête.

expr_list

Une ou plusieurs expressions délimitées par des virgules, ou un ou plusieurs ensembles d'expressions délimitées par des virgules et entourées par des parenthèses.

table_subquery

Une sous-requête qui correspond à une table avec une ou plusieurs lignes, mais est limitée à une seule colonne dans la liste de sélection.

IN | NOT IN

IN retourne la valeur true si l'expression est membre de la liste d'expressions ou de la requête. NOT IN retourne la valeur true si l'expression n'est pas membre. IN et NOT IN retournent la valeur NULL et aucune ligne n'est retournée dans les cas suivants : si expression génère null, s'il n'y a aucune expr_list correspondante ou si les valeurs de table_subquery et au moins l'une de ces lignes de comparaison entraînent une valeur null.

Exemples

Les conditions suivantes sont vraies uniquement pour les valeurs répertoriées :

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

Optimisation pour les grandes listes IN

Afin d'optimiser les performances des requêtes, une liste IN qui inclut plus de 10 valeurs est analysée en interne comme un ensemble (array) scalaire. Les listes IN avec moins de 10 valeurs sont évaluées

comme une série de prédicats OR. Cette optimisation est prise en charge pour les types de données SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP et TIMESTAMPTZ.

Examinez la sortie EXPLAIN de la requête pour voir l'effet de cette optimisation. Par exemple :

```
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)
```

Interrogation de données imbriquées

AWS Clean Rooms offre un accès compatible avec SQL aux données relationnelles et imbriquées.

AWS Clean Rooms utilise la notation en pointillés et un indice de tableau pour la navigation par chemin lors de l'accès à des données imbriquées. Il permet également de FROM éléments de clause à itérer sur des tableaux et à utiliser pour les opérations de désimbrication. Les rubriques suivantes décrivent les différents modèles de requête qui associent l'utilisation du type de array/struct/map données à la navigation par chemin et par tableau, à la dénidification et aux jointures.

Rubriques

Navigation

AWS Clean Rooms permet de naviguer dans des tableaux et des structures en utilisant respectivement la notation `[...]` entre crochets et points. En outre, vous pouvez mélanger la navigation dans des structures en utilisant notation par points avec la navigation dans des tableaux en utilisant la notation entre crochets.

Example

Par exemple, l'exemple de requête suivant suppose que la colonne de données du `c_orders` tableau est un tableau doté d'une structure et d'un attribut nommé`o_orderkey`.

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

Vous pouvez utiliser la notation par points et crochets dans tous les types de requêtes, comme le filtrage, la jointure et l’agrégation. Vous pouvez utiliser ces notations dans une requête dans laquelle il y a normalement des références de colonne.

Example

L’exemple suivant utilise une instruction SELECT qui filtre les résultats.

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0].o_orderkey IS NOT NULL;
```

Example

L'exemple suivant utilise la notation par points et crochets dans les clauses GROUP BY et ORDER BY :

```
SELECT c_orders[0].o_orderdate,
       c_orders[0].o_orderstatus,
       count(*)
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderkey IS NOT NULL
GROUP BY c_orders[0].o_orderstatus,
         c_orders[0].o_orderdate
ORDER BY c_orders[0].o_orderdate;
```

Désimbriquer des requêtes

Pour désimbriquer les requêtes, AWS Clean Rooms active l'itération sur des tableaux. Pour ce faire, il navigue dans le tableau à l'aide de la clause FROM d'une requête.

Example

En utilisant l'exemple précédent, le suivant itère sur les valeurs de l'attribut pour `c_orders`.

```
SELECT o FROM customer_orders_lineitem c, c.c_orders o;
```

La syntaxe de désimbrication est une extension de la clause FROM. Dans SQL standard, la clause FROM `x (AS) y` signifie que `y` itère sur chaque tuple dans la relation `x`. Dans ce cas, `x` fait référence à une relation et `y` fait référence à un alias pour la relation `x`. De même, la syntaxe de désimbrication à l'aide de l'élément de clause FROM `x (AS) y` implique une `y` itération sur chaque valeur d'une expression matricielle. `x` Dans ce cas, `x` est une expression de tableau et `y` un alias pour`x`.

L'opérande de gauche peut également utiliser la notation par points et crochets pour la navigation régulière.

Example

Dans l'exemple précédent :

- `customer_orders_lineitem c` est l'itération sur la table de `customer_order_lineitem` base
- `c.c_orders o` est l'itération sur le `c.c_orders array`

Pour itérer sur l'attribut `o_lineitems`, qui est un tableau dans un tableau, vous ajoutez plusieurs clauses.

```
SELECT o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

AWS Clean Rooms prend également en charge un index de tableau lors de l'itération sur le tableau à l'aide du AT mot clé. La clause `x AS y AT z` itère sur le tableau `x` et génère le champ`z`, qui est l'index du tableau.

Example

L'exemple suivant illustre le fonctionnement d'un index de tableau.

```
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

L'exemple suivant itère sur un tableau scalaire

```
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

L'exemple suivant itère sur un tableau de plusieurs niveaux. L'exemple utilise plusieurs clauses de désimbrication (unnest) pour effectuer une itération dans les tableaux les plus intérieurs. La `f.multi_level_array` AS le tableau itère. `multi_level_array` La matrice AS élément est l'itération sur les tableaux qu'il contient. `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)
```

Sémantique laxiste

Par défaut, les opérations de navigation sur des valeurs de données imbriquées renvoient une valeur nulle au lieu de renvoyer une erreur lorsque la navigation n'est pas valide. La navigation par objet n'est pas valide si la valeur de données imbriquée n'est pas un objet ou si la valeur de données imbriquée est un objet mais ne contient pas le nom d'attribut utilisé dans la requête.

Example

Par exemple, la requête suivante accède à un nom d'attribut non valide dans la colonne de données imbriquée : `c_orders`

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

La navigation dans le tableau renvoie la valeur nulle si la valeur des données imbriquées n'est pas un tableau ou si l'index du tableau est hors limites.

Example

La requête suivante renvoie la valeur null car elle `c_orders[1][1]` est hors limites.

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

Types d’introspection

Les colonnes de type de données imbriquées prennent en charge les fonctions d'inspection qui renvoient le type et d'autres informations de type concernant la valeur. AWS Clean Rooms prend en charge les fonctions booléennes suivantes pour les colonnes de données imbriquées :

- 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

Toutes ces fonctions renvoient false si la valeur d’entrée est nulle. IS_SCALAR, IS_OBJECT et IS_ARRAY s’excluent mutuellement et couvrent toutes les valeurs possibles à l’exception de null. Pour déduire les types correspondant aux données, AWS Clean Rooms utilisez la fonction JSON_TYPEOF qui renvoie le type (le niveau supérieur de) la valeur de données imbriquée, comme indiqué dans l'exemple suivant :

```
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
```

Historique du document pour la référence AWS Clean Rooms SQL

Le tableau suivant décrit les versions de documentation de la référence AWS Clean Rooms SQL.

Pour recevoir les notifications sur les mises à jour de cette documentation, vous pouvez vous abonner au Flux RSS. Pour vous abonner aux mises à jour RSS, votre navigateur doit disposer d'un plug-in RSS activé.

Modification	Description	Date
Spark SQL prend en charge les astuces	AWS Clean Rooms Spark SQL prend en charge les indications de requête afin d'optimiser les performances des requêtes et de réduire les coûts de calcul.	20 janvier 2026
Spark SQL prend en charge la table de cache	AWS Clean Rooms Spark SQL prend en charge la commande CACHE TABLE, qui permet aux clients de mettre en cache des tables existantes ou de créer et de mettre en cache de nouvelles tables à partir des résultats de requêtes pour améliorer les performances des requêtes.	22 octobre 2025
Spark SQL prend en charge les fonctions FIRST et LAST Window	AWS Clean Rooms Spark SQL prend en charge les fonctions Windows suivantes : FIRST et LAST.	12 juin 2025
Mises à jour de la documentation sur les fonctions SQL	Mise à jour uniquement destinée à la documentation afin de refléter avec précision	20 mai 2025

	les fonctions Spark SQL prises en charge. Suppressi on de la documentation concernant 25 fonctions non prises en charge, notamment <=> operator, SIMILAR TO, LISTAGG et ARRAY_INSERT. Les noms de fonctions ont été corrigés, de DATEADD à DATE_ADD, DATEDIFF à DATE_DIFF, ISNULL à IS_NULL et ISNOTNULL à IS_NOT_NULL. Correction d'une faute de frappe dans les exemples DATE_PART.	
AWS Clean Rooms SQL Spark	Les clients peuvent désormais exécuter des requêtes à l'aide de certaines condition s, fonctions, commandes et conventions SQL prises en charge par le moteur d'analyse Spark SQL.	29 octobre 2024
Commandes SQL et fonctions SQL — mise à jour	Des exemples ont été ajoutés pour la clause JOIN, l'opérateur EXCEPT set, l'expression condition nelle CASE et les fonctions suivantes : ANY_VALUE , NVL et COALESCE, NULLIF, CAST, CONVERT, CONVERT_TIMEZONE, EXTRACT, MOD, SIGN, CONCAT, FIRST_VALUE et LAST_VALUE.	28 février 2024

Fonctions SQL - mise à jour	AWS Clean Rooms prend désormais en charge les fonctions SQL suivantes : Array, SUPER et VARBYTE. Les fonctions mathémati ques suivantes sont désormais prises en charge : ACOS, ASIN, ATAN, COT ATAN2, DEXP, PI, POW, RADIANS et SIN. Les fonctions JSON suivantes sont désormais prises en charge : CAN_JSON_PARSE, JSON_PARSE et JSON_SERI ALIZE.	6 octobre 2023
Support des types de données imbriqués	AWS Clean Rooms prend désormais en charge les types de données imbriqués.	30 août 2023
Règles de dénomination SQL - mise à jour	Modification concernant uniquement la documenta tion pour clarifier les noms de colonnes réservées.	16 août 2023
Disponibilité générale	La référence AWS Clean Rooms SQL est désormais disponible pour tous.	31 juillet 2023

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.