SQL 参考

AWS Clean Rooms

AWS Clean Rooms: SQL 参考

Table of Contents

中的 SQL 概述 AWS Clean Rooms

欢迎使用《AWS Clean Rooms SQL 参考》。

AWS Clean Rooms 围绕行业标准的结构化查询语言 (SQL) 构建，结构化查询语言是一种由用于处理数据库和数据库对象的命令和函数组成的查询语言。SQL 还会强制实施有关数据类型、表达式和文本使用的规则。

以下主题提供有关约定、命名规则和数据类型的一般信息：

主题

- SQL 参考惯例
- SQL 命名规则
- 数据类型

要了解可以在中使用的 SQL 命令、SQL 函数类型和 SQL 条件 AWS Clean Rooms，请查看以下主题：

- 中的 SQL 命令 AWS Clean Rooms
- 中的 SQL 函数 AWS Clean Rooms
- 中的 SQL 条件 AWS Clean Rooms

有关的更多信息 AWS Clean Rooms，请参阅《AWS Clean Rooms 用户指南》和《AWS Clean Rooms API 参考》。

SQL 参考惯例

本节介绍用于为 SQL 表达式、命令和函数编写语法的约定。

字符	描述
CAPS	采用大写字母的字样为关键字。
[ ]	方括号表示可选参数。方括号中的多个参数表示您可选择任意数量的参数。此外，不同行的括号中的参数表示分析程序需要按照参数在语法中列出的顺序获取参数。

字符	描述
{ }	大括号表示您需要选择大括号内的参数之一。
|	管道表示您可以在不同参数之间选择。
斜体	斜体字样表示占位符。必须插入适当的值以替换斜体字样。
...	省略号表示可以重复前面的元素。
'	单引号中的字样表示必须键入引号。

SQL 命名规则

以下各节说明了 AWS Clean Rooms 中的 SQL 命名规则。

配置表关联名称和列

可以查询的成员使用配置表关联名称作为查询中的表名。配置表关联名称和配置表列可以在查询中使用别名。

以下命名规则适用于配置表关联名称、配置表的列名和别名：

- 它们只能使用字母数字、下划线 (_) 或连字符 (-)，但不能以连字符开头或结尾。
  - （仅限自定义分析规则）它们可以使用美元符号 ($)，但不能使用后跟用美元括起来的字符串常量的模式。

    用美元括起来的字符串常量包括：

    - 一个美元符号 ($)
    - 零个或多个字符（可选“标签”）
    - 另一个美元符号
    - 构成字符串内容的任意字符序列
    - 一个美元符号 ($)
    - 以美元引号开头的同一个标签
    - 一个美元符号

例如：$$invalid$$

- 它们不能包含连续的连字符 (-)。
- 它们不能以以下任何前缀开头：

  padb_, pg_, stcs_, stl_, stll_, stv_, svcs_, svl_, svv_, sys_, systable_
- 它们不能包含反斜杠字符 (\)、引号 (') 或非双引号的空格。
- 如果它们以非字母字符开头，则必须位于双引号 (" ") 中。
- 如果它们包含连字符 (-)，则必须位于双引号 (" ") 内。
- 它们的长度必须在 1 到 127 个字符之间。
- 保留字必须位于双引号 (" ") 内。
- 以下保留的列名不能在 AWS Clean Rooms 中使用（即使带引号）：
  - oid
  - tableoid
  - xmin
  - cmin
  - xmax
  - cmax
  - ctid

文本

文本或常量是固定数据值，由一系列字符或数字常量组成。

下面是 AWS Clean Rooms 中的文本的命名规则：

- 支持数本、字符以及日期、时间和时间戳文本。
- 仅支持 Unicode 通用类别 (Cc) 中的 TAB、CARRIAGE RETURN (CR) 和 LINE FEED (LF) Unicode 控制字符。
- SELECT 语句不支持直接引用投影列表中的文本。

  例如：

```
SELECT 'test', consumer.first_purchase_day
FROM consumer
```

```
INNER JOIN provider2
ON consumer.hashed_email = provider2.hashedemail
```

保留字

下面是 AWS Clean Rooms 中的保留字列表。

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

数据类型

AWS Clean Rooms 存储或检索的每个值都有一个数据类型，其中包含一组固定的关联属性。数据类型是在创建表时声明的。数据类型约束了列或参数可包含的一组值。

下表列出了可在 AWS Clean Rooms 表中使用的数据类型。

数据类型	别名	描述
ARRAY	不适用	ARRAY 嵌套数据类型
BIGINT	不适用	有符号的八字节整数
BOOLEAN	BOOL	逻辑布尔值 (true/false)
CHAR	CHARACTER	固定长度字符串
DATE	不适用	日历日期（年、月、日）
DECIMAL	NUMERIC	可选精度的精确数字
DOUBLE PRECISION	FLOAT8、FLOAT	双精度浮点数
INTEGER	INT	有符号的四字节整数
MAP	不适用	MAP 嵌套数据类型
REAL	FLOAT4	单精度浮点数
SMALLINT	不适用	有符号的二字节整数
STRUCT	不适用	STRUCT 嵌套数据类型
SUPER	不适用	包含所有标量类型的超集数据类型， AWS Clean Rooms 包括复杂类型，例如 ARRAY 和 STRUCTS。
TIME	不适用	Time of day
TIMETZ	不适用	Time of day with time zone

数据类型	别名	描述
VARBYTE	VARBINARY，二进制可变	长度可变的二进制值
VARCHAR	CHARACTER VARYING	具有用户定义的限制的可变长度字符串，

> **Note**
>
> ARRAY、STRUCT 和 MAP 嵌套数据类型目前仅适用于自定义分析规则。有关更多信息，请参阅 嵌套类型。

多字节字符

VARCHAR 数据类型支持多达 4 个字节的 UTF-8 多字节字符。不支持 5 个字节或更长的字符。要计算包含多字节字符的 VARCHAR 列的大小，请用字符数乘以每个字符的字节数。例如，如果一个字符串包含四个中文字符，并且每个字符的长度为三个字节，则您需要一个 VARCHAR(12) 列才能存储该字符串。

VARCHAR 数据类型不支持下列无效的 UTF-8 代码点：

`0xD800 – 0xDFFF`（字节序列：`ED A0 80 – ED BF BF`）

CHAR 数据类型不支持多字节字符。

数字类型

主题

- 整数类型
- DECIMAL 或 NUMERIC 类型
- 有关使用 128 位 DECIMAL 或 NUMERIC 列的说明
- 浮点类型
- 数值计算

数字数据类型包括整数、小数和浮点数。

整数类型

使用 SMALLINT、INTEGER 和 BIGINT 数据类型存储各种范围的整数。您无法存储每种类型所允许范围之外的值。

名称	存储	范围
SMALLINT	2 字节	-32768 到 +32767
INTEGER 或 INT	4 字节	-2147483648 到 +2147483647
BIGINT	8 字节	-9223372036854775808 到 9223372036854775807

DECIMAL 或 NUMERIC 类型

使用 DECIMAL 或 NUMERIC 数据类型存储具有用户定义的精度 的值。DECIMAL 和 NUMERIC 关键字是可互换的。在本文档中，小数 是此数据类型的首选术语。术语数字 一般用于指整数、小数和浮点数据类型。

存储	范围
可变，对于未压缩的 DECIMAL 类型可以多达 128 位。	128 位有符号整数，精度位数可以多达 38 位。

通过指定*##* 和*####* 来定义表中的 DECIMAL 列：

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

##

整个值中有效位的总数：小数点两边的位数。例如，数字 48.2891 的精度为 6，小数位数为 4。如果未指定，默认精度为 18。最大精度为 38。

如果输入值中的小数点左侧的位数超出了列的精度减去其小数位数的差值，则此值无法复制到列中（或无法插入或更新）。此规则适用于列定义范围之外的任何值。例如，numeric(5,2) 列所允许的值范围为 -999.99 到 999.99。

####

值的小数部分中小数点右侧的小数位数。整数的小数位数为零。在列规范中，小数位数值必须小于或等于精度值。如果未指定，默认小数位数为 0。最大小数位数为 37。

如果加载到表中的输入值的小数位数大于列的小数位数，则该值将四舍五入到指定小数位数。例如，SALES 表中的 PRICEPAID 列为 DECIMAL(8,2) 列。如果将 DECIMAL(8,4) 值插入到 PRICEPAID 列中，则该值将四舍五入到小数位数 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)
```

但是，对于显示强制转换表中选定的值而得出的结果，不会进行四舍五入。

> **Note**
>
> 您可插入到 DECIMAL(19,0) 列中的最大正值为 9223372036854775807 (2^{63} -1)。最大负值为 -9223372036854775807。例如，尝试插入值 9999999999999999999（19 个 9）将导致溢出错误。无论小数点的位置如何， AWS Clean Rooms 可表示为 DECIMAL 数的最大字符串为 9223372036854775807。例如，您可加载到 DECIMAL(19,18) 列中的最大值为 9.223372036854775807。
> 制定这些规则的原因如下：
>
> - 有效精度为 19 位或更少的 DECIMAL 值在内部存储为 8 字节整数。
> - 有效精度为 20-38 的 DECIMAL 值在内部存储为 16 字节整数。

有关使用 128 位 DECIMAL 或 NUMERIC 列的说明

请勿为 DECIMAL 列随意分配最高精度，除非您确定您的应用程序需要此精度。128 位值使用的磁盘空间是 64 位值的两倍，并且可能会降低查询执行速度。

浮点类型

使用 REAL 和 DOUBLE PRECISION 数据类型可存储具有可变精度 的数值。这些类型是不精确的 类型，意味着一些值是作为估计值存储的，因此存储和返回某个特定值可能导致细微的差异。如果您需要精确的存储和计算（如货币金额），请使用 DECIMAL 数据类型。

根据 IEEE 浮点运算标准 754，REAL 表示单精度浮点数格式。它的精度约为 6 位，范围在 1E-37 到 1E+37 之间。您也可以将此数据类型指定为 FLOAT4。

根据 IEEE 二进制浮点运算标准 754，DOUBLE PRECISION 表示双精度浮点数格式。它的精度约为 15 位，范围在 1E-307 到 1E+308 之间。您也可以将此数据类型指定为 FLOAT 或 FLOAT8。

数值计算

在中 AWS Clean Rooms，计算是指二进制数学运算：加法、减法、乘法和除法。此部分介绍这些运算的预期返回类型，以及在涉及 DECIMAL 数据类型时应用于确定精度和小数位数的特定公式。

当在查询处理期间计算数值时，您可能会遇到无法计算和查询返回数字溢出错误的情况。您还可能会遇到计算值的小数位数发生变化或出乎意料的情况。对于一些运算，您可使用显式强制转换（类型提升）或 AWS Clean Rooms 配置参数来解决这些问题。

有关类似使用 SQL 函数的计算的结果的信息，请参阅中的 SQL 函数 AWS Clean Rooms。

计算的返回类型

给定中支持的一组数值数据类型 AWS Clean Rooms，下表显示了加法、减法、乘法和除法运算的预期返回类型。表左侧第一列表示计算中的第一个操作数，顶部行表示第二个操作数。

	SMALLINT	INTEGER	BIGINT	DECIMAL	FLOAT4	FLOAT8
SMALLINT	SMALLINT	INTEGER	BIGINT	DECIMAL	FLOAT8	FLOAT8
INTEGER	INTEGER	INTEGER	BIGINT	DECIMAL	FLOAT8	FLOAT8
BIGINT	BIGINT	BIGINT	BIGINT	DECIMAL	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	FLOAT8	FLOAT8

FLOAT4	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT4	FLOAT8
FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8	FLOAT8

计算的 DECIMAL 结果的精度和小数位数

下表汇总了在数学运算返回 DECIMAL 结果时用于计算生成的精度和小数位数的规则。在此表中，p1 和 s1 分别表示计算中第一个操作数的精度和小数位数，p2 和 s2 分别表示第二个操作数的精度和小数位数。（不管这些计算如何，最大的结果精度为 38，最大的结果小数位数为 38）。

运算	结果精度和小数位数
+ 或者 -	小数位数 = max(s1,s2) 精度 = max(p1-s1,p2-s2)+1+scale
*	小数位数 = s1+s2 精度 = p1+p2+1
/	小数位数 = max(4,s1+p2-s2+1) 精度 = p1-s1+ s2+scale

例如，SALES 表中的 PRICEPAID 和 COMMISSION 列均为 DECIMAL(8,2) 列。如果您用 PRICEPAID 除以 COMMISSION（或者反过来），采用的公式如下所示：

```
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)
```

以下计算是使用集合运算符（如 UNION、INTERSECT 和 EXCEPT）或 COALESCE 和 DECODE 等函数计算对 DECIMAL 值执行的运算的最终精度和小数位数的一般规则：

```
Scale = max(s1,s2)
```

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

例如，具有一个 DECIMAL(7,2) 列的 DEC1 表将与具有一个 DECIMAL(15,3) 列的 DEC2 表联接以创建 DEC3 表。DEC3 的 schema 表明该列变成了 NUMERIC(15,3) 列。

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

在上例中，采用的公式如下所示：

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

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

有关除法运算的说明

对于除法运算， divide-by-zero 条件会返回错误。

在计算精度和小数位数之后，将应用小数位数最多为 100 的限制。如果计算所得的结果小数位数大于 100，则除法运算结果的范围如下所示：

- 精度 = `precision - (scale - max_scale)`
- 小数位数 = `max_scale`

如果计算所得的精度大于最大精度 (38)，则精度将减少为 38，小数位数的结果将介于以下范围：`max(38 + scale - precision), min(4, 100))`

溢出条件

将检查所有数值计算是否存在溢出情况。精度为 19 或 19 以下的 DECIMAL 数据存储为 64 位整数。精度大于 19 的 DECIMAL 数据存储为 128 位整数。所有 DECIMAL 值的最大精度为 38，最大小数位数为 37。当值超出这些限制时将出现溢出错误，中间结果集和最终结果集都存在这种情况：

- 当特定数据值不符合强制转换函数指定的请求精度或小数位数时，显式强制转换将生成运行时溢出错误。例如，您无法强制转换 SALES 表中 PRICEPAID 列（DECIMAL(8,2) 列）的所有值并返回 DECIMAL(7,3) 结果：

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

此错误出现的原因是 PRICEPAID 列中的一些 较大的值无法强制转换。

- 乘法运算的乘积结果中的小数位数为所有操作数的小数位数之和。例如，如果两个操作数的小数位数都为 4，则结果的小数位数为 8，小数点左侧仅剩下 10 位。因此，在具有有效小数位数的两个大数相乘时，遇到溢出的情况相对容易一些。

INTEGER 和 DECIMAL 类型的数值计算

如果计算中的一个操作数具有 INTEGER 数据类型且另一个操作数为 DECIMAL，则 INTEGER 操作数将隐式强制转换为 DECIMAL：

- SMALLINT 将强制转换为 DECIMAL(5,0)
- INTEGER 将强制转换为 DECIMAL(10,0)
- BIGINT 将强制转换为 DECIMAL(19,0)

例如，如果将 SALES.COMMISSION（DECIMAL(8,2) 列）和 SALES.QTYSOLD（SMALLINT 列）相乘，则此计算将强制转换为：

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

字符类型

字符数据类型包括 CHAR（字符）和 VARCHAR（字符变体）。

存储和范围

CHAR 和 VARCHAR 数据类型是按照字节而不是字符来定义的。CHAR 列只能包含单字节字符，因此 CHAR(10) 列可包含最大长度为 10 字节的字符串。VARCHAR 可包含多字节字符，并且每个字符最多可以有 4 个字节。例如，VARCHAR(12) 列可包含 12 个单字节字符、6 个双字节字符、4 个三字节字符或 3 个四字节字符。

名称	存储	范围（列宽度）
CHAR 或 CHARACTER	字符串的长度，包括尾部空格（如有）	4096 字节
VARCHAR 或 CHARACTER VARYING	4 字节 + 字符的总字节，其中每个字符可为 1 至 4 个字节。	65535 字节 (64K -1)

CHAR 或 CHARACTER

使用 CHAR 或 CHARACTER 列存储固定长度字符串。这些字符串将使用空格填补，因此 CHAR(10) 列始终占用 10 字节的存储。

```
char(10)
```

未指定长度的 CHAR 列将生成 CHAR(1) 列。

VARCHAR 或 CHARACTER VARYING

使用 VARCHAR 或 CHARACTER VARYING 列存储具有固定限制的可变长度字符串。这些字符串不会使用空格填补，因此 VARCHAR(120) 列最多包含 120 个单字节字符、60 个双字节字符、40 个三字节字符或 30 个四字节字符。

```
varchar(120)
```

尾部空格的意义

CHAR 和 VARCHAR 数据类型存储长度最多为 n 字节的字符串。尝试将更长的字符串存储到这些类型的列中将导致错误。但是，如果额外的字符全为空格，则字符串将截断至最大长度。如果字符串短于最大长度，CHAR 值将使用空格填补，但 VARCHAR 值将存储不带空格的字符串。

CHAR 值中的尾部空格始终无语义意义。当比较两个 CHAR 值时将忽视尾部空格，而不将其包含在 LENGTH 计算中，在将 CHAR 值转换为其他字符串类型时将删除尾部空格。

VARCHAR 和 CHAR 值中的尾部空格将在比较值时视为无语义意义。

长度计算将返回 VARCHAR 字符串的包含尾部空格在内的长度。尾部空格不会计入固定长度字符串的长度中。

日期时间类型

日期时间数据类型包括 DATE、TIME、TIMETZ、TIMESTAMP 和 TIMESTAMPTZ。

主题

存储和范围

名称	存储	范围	解析
DATE	4 字节	4713 BC 到 294276 AD	1 天
TIME	8 字节	00:00:00 至 24:00:00	1 微秒
TIMETZ	8 字节	00:00:00+1459 至 00:00:00+1459	1 微秒
TIMESTAMP	8 字节	4713 BC 到 294276 AD	1 微秒
TIMESTAMPTZ	8 字节	4713 BC 到 294276 AD	1 微秒

DATE

使用 DATE 数据类型存储没有时间戳的简单日历日期。

TIME

使用 TIME 数据类型存储一天中的时间。

TIME 列存储小数秒的精度最高达到 6 位的值。

默认情况下，用户表和 AWS Clean Rooms 系统表中的 TIME 值均为协调世界时 (UTC)。

TIMETZ

使用 TIMETZ 数据类型来存储带有时区的一天中的时间。

TIMETZ 列存储小数秒的精度最高达到 6 位的值。

默认情况下，用户表和 AWS Clean Rooms 系统表中的 TIMETZ 值均为 UTC。

TIMESTAMP

使用 TIMESTAMP 数据类型存储包含日期和当日时间的完整时间戳值。

TIMESTAMP 列存储小数秒的精度最高达到 6 位的值。

如果将日期插入到 TIMESTAMP 列中，或插入具有部分时间戳值的日期，则值将隐式转换为完整时间戳值。此完整时间戳值具有缺少的小时、分钟和秒的默认值 (00)。输入字符串中的时区值被忽略。

默认情况下，用户表和 AWS Clean Rooms 系统表中的时间戳值均为 UTC。

TIMESTAMPTZ

使用 TIMESTAMPTZ 数据类型输入包含日期、当日时间和时区的完整时间戳值。当输入值包含一个时区时， AWS Clean Rooms 使用时区将值转化为协 UTC 并存储 UTC 值。

要查看支持的时区名称的列表，请执行以下命令。

```
select my_timezone_names();
```

要查看支持的时区缩写的列表，请执行以下命令。

```
select my_timezone_abbrevs();
```

您还可以在 IANA 时区数据库中找到有关时区的当前信息。

下表提供了时区格式的示例。

格式	示例
dd mon hh:mi:ss yyyy tz	17 Dec 07:37:16 1997 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 US/Pacific
yyyy-mm-dd hh: mi: ss+/-tz	1997-12-17 07:37:16-08
dd.mm.yyyy hh:mi:ss tz	17.12.1997 07:37:16.00 PST

TIMESTAMPTZ 列存储小数秒的精度最高达到 6 位的值。

如果将日期插入到 TIMESTAMPTZ 列中，或插入具有部分时间戳的日期，则值将隐式转换为完整时间戳值。此完整时间戳值具有缺少的小时、分钟和秒的默认值 (00)。

TIMESTAMPTZ 值为用户表中的 UTC。

日期时间类型的示例

以下示例向您展示如何处理 AWS Clean Rooms支持的日期时间类型。

日期示例

以下示例插入具有不同格式的日期并显示输出。

```
select * from datetable order by 1;

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

如果您将时间戳值插入 DATE 列，时间部分会被忽略，只会加载日期。

时间示例

以下示例插入具有不同格式的 TIME 和 TIMETZ 值并显示输出。

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

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

时间戳示例

如果您将日期插入到 TIMESTAMP 或 TIMESTAMPTZ 列中，时间将默认为午夜。例如，如果您插入文本 `20081231`，存储的值为 `2008-12-31 00:00:00`。

以下示例插入具有不同格式的时间戳并显示输出。

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

日期、时间和时间戳文本

以下是使用支持的日期、时间和时间戳文字的规则。 AWS Clean Rooms

日期

下表显示了输入日期，这些日期是您可以加载到 AWS Clean Rooms 表中的文字日期值的有效示例。默认 `MDY DateStyle` 模式被认为是有效的。此模式意味着在字符串中，月份值将位于日期值之前，如 `1999-01-08` 和 `01/02/00`。

> **Note**
>
> 当您将日期或时间戳文本加载到表中时，这些文本必须用引号括起来。

输入日期	完整日期
1999 年 1 月 8 日	1999 年 1 月 8 日
1999-01-08	1999 年 1 月 8 日
1/8/1999	1999 年 1 月 8 日
01/02/00	2000 年 1 月 2 日

输入日期	完整日期
2000-Jan-31	2000 年 1 月 31 日
Jan-31-2000	2000 年 1 月 31 日
31-Jan-2000	2000 年 1 月 31 日
20080215	2008 年 2 月 15 日
080215	2008 年 2 月 15 日
2008.366	2008 年 12 月 31 日（三位数的日期部分必须介于 001 和 366 之间）

Times

下表显示了输入时间，这些时间是您可以加载到 AWS Clean Rooms 表中的文字时间值的有效示例。

输入时间	描述（时间部分）
04:05:06.789	上午 4:05 过 6.789 秒
04:05:06	上午 4:05 过 6 秒
04:05	恰好上午 4:05
040506	上午 4:05 过 6 秒
04:05 AM	恰好上午 4:05；AM 为可选
04:05 PM	恰好下午 4:05；小时值必须小于 12
16:05	恰好下午 4:05

时间戳

下表显示了输入时间戳，这些时间戳是您可以加载到 AWS Clean Rooms 表中的文字时间值的有效示例。所有有效的日期文本可与下列时间文本组合。

输入时间戳（连接在一起的日期和时间）	描述（时间部分）
20080215 04:05:06.789	上午 4:05 过 6.789 秒
20080215 04:05:06	上午 4:05 过 6 秒
20080215 04:05	恰好上午 4:05
20080215 040506	上午 4:05 过 6 秒
20080215 04:05 AM	恰好上午 4:05；AM 为可选
20080215 04:05 PM	恰好下午 4:05；小时值必须小于 12
20080215 16:05	恰好下午 4:05
20080215	午夜（默认情况）

特殊日期时间值

下列显示可用作日期时间文本和日期函数参数的特殊值。它们需要单引号，并在查询处理期间转换为常规时间戳值。

特殊值	描述
now	计算结果为当前事务的开始时间并返回具有微秒精度的时间戳。
today	计算结果为相应的日期并返回时间部分为零的时间戳。
tomorrow	计算结果为相应的日期并返回时间部分为零的时间戳。
yesterday	计算结果为相应的日期并返回时间部分为零的时间戳。

以下示例演示 now 和 today 如何与 DATEADD 函数结合使用。

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

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

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

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

间隔文本

以下是用于处理 AWS Clean Rooms支持的时间间隔文本的规则。

使用间隔文本标识特定时间段（如 12 hours 或 6 weeks）。您可在涉及日期时间表达式的条件和计算中使用这些间隔文本。

> **Note**
>
> 不能对 AWS Clean Rooms 表中的列使用 INTERVAL 数据类型。

间隔用 INTERVAL 关键字与数量和支持的日期部分的组合表示；例如 INTERVAL '7 days' 或 INTERVAL '59 minutes'。您可以将许多数量和单位连接在一起以形成更精确的间隔；例如：INTERVAL '7 days, 3 hours, 59 minutes'。还支持每个单位的缩写和复数；例如：5 s、5 second 和 5 seconds 是等效的间隔。

如果您未指定日期部分，则间隔值表示秒。您可指定数量值作为小数（例如：0.5 days）。

示例

以下示例显示了具有不同间隔值的一系列计算。

以下示例向指定日期添加 1 秒。

```
select caldate + interval '1 second' as dateplus from date
where caldate='12-31-2008';
dateplus
```

```
---------------------
2008-12-31 00:00:01
(1 row)
```

以下示例向指定日期添加 1 分钟。

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

以下示例向指定日期添加 3 小时 35 分钟。

```
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)
```

以下示例向指定日期添加 52 周。

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

以下示例向指定日期添加 1 周、1 小时、1 分钟和 1 秒。

```
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)
```

以下示例向指定日期添加 12 小时（半天）。

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

以下示例将从 2023 年 2 月 15 日减去 4 个月，结果为 2022 年 10 月 15 日。

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

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

以下示例将从 2023 年 3 月 31 日减去 4 个月，结果为 2022 年 11 月 30 日。计算时会考虑一个月中的天数。

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

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

布尔值类型

使用 BOOLEAN 数据类型在单字节列中存储 true 和 false 值。下表描述了布尔值的三种可能状态以及导致这些状态的文本值。不管输入字符串如何，Boolean 列将存储和输出“t”表示 true，“f”表示 false。

州	有效的文本值	存储
True	`TRUE 't' 'true' 'y' 'yes' '1'`	1 字节
False	`FALSE 'f' 'false' 'n' 'no' '0'`	1 字节

州	有效的文本值	存储
Unknown	NULL	1 字节

您可以使用 IS 比较将布尔值仅作为 WHERE 子句中的谓词进行检查。不能将 IS 比较与 SELECT 列表中的布尔值一起使用。

示例

您可使用 BOOLEAN 列将每个客户的“活跃/非活跃”状态存储在 CUSTOMER 表中。

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

在此示例中，以下查询从 USERS 表中选择喜欢运动而不喜欢电影院的用户：

```
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)
```

以下示例从 USERS 表中选择不清楚是否喜欢摇滚音乐的用户。

```
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)
```

以下示例返回错误，因为它在 SELECT 列表中使用了 IS 比较。

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

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

以下示例成功，因为它在 SELECT 列表中使用了等于比较 (=) 而不是 IS 比较。

```
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    |
```

SUPER 类型

使用 SUPER 数据类型将半结构化数据或文档存储为值。

半结构化数据不符合 SQL 数据库中使用的关系数据模型的刚性和表格结构。SUPER 数据类型包含引用数据中不同实体的标签。SUPER 数据类型可以包含复杂的值，如数组、嵌套结构和其他与序列化格式（如 JSON）相关联的复杂结构。SUPER 数据类型是一组无架构数组和结构值，它们包含 AWS Clean Rooms的所有其他标量类型。

SUPER 数据类型最高支持 1 MB 的单个 SUPER 字段或对象的数据。

SUPER 数据类型具有以下属性：

- AWS Clean Rooms 标量值：
  - Null
  - 布尔值
  - 一个数字，如 smallint、整数、bigint、小数或浮点（如 float4 或 float8）
  - 字符串值，如 varchar 或 char
- 一个复杂的值：
  - 一个值数组，包括标量或复数
  - 一个结构，也称为元组或对象，它是属性名称和值（标量或复数）的映射

这两种类型的复数值中的任何一种都包含它们自己的标量或复数值，而对规则性没有任何限制。

SUPER 数据类型以无 schema 形式支持半结构化数据的持久性。虽然分层数据模型可以更改，但旧版本的数据可以共存在于同一个 SUPER 列中。

嵌套类型

AWS Clean Rooms 支持涉及嵌套数据类型（特别是 AWS Glue 结构、数组和映射列类型）的数据的查询。只有自定义分析规则支持嵌套数据类型。

值得注意的是，嵌套数据类型并不符合 SQL 数据库关系数据模型严格的表格结构。

嵌套数据类型包含引用数据中不同实体的标签。它们可以包含复杂的值，如数组、嵌套结构和其他与序列化格式（如 JSON）相关联的复杂结构。嵌套数据类型支持单个嵌套数据类型字段或对象最多 1 MB 的数据。

嵌套数据类型的示例

对于 `struct<given:varchar, family:varchar>` 类型，有两个属性名称：`given` 和 `family`，每个名称对应一个 `varchar` 值。

对于 `array<varchar>` 类型，数组指定为 `varchar` 的列表。

`array<struct<shipdate:timestamp, price:double>>` 类型是指具有 `struct<shipdate:timestamp, price:double>` 类型的元素列表。

`map` 数据类型的行为类似于 `structs` 的 `array`，其中数组中每个元素的属性名称用 `key` 表示并映射到 `value`。

Example

例如，`map<varchar(20), varchar(20)>` 类型被视为 `array<struct<key:varchar(20), value:varchar(20)>>`，其中 `key` 和 `value` 指的是底层数据中的映射的属性。

有关如何 AWS Clean Rooms 启用对数组和结构的导航的信息，请参见导航。

有关如何通过使用查询的 FROM 子句浏览数组来 AWS Clean Rooms 启用对数组的迭代的信息，请参见取消嵌套查询。

VARBYTE 类型

使用 VARBYTE、VARBINARY 或 BINARY VARYING 列存储具有固定限制的可变长度二进制值。

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

最大字节数（n）范围为 1 到 1,024,000。默认值为 64,000。

下面是一些你可能想使用 VARBYTE 数据类型的示例:

- 在 VARBYTE 列上联接表。
- 创建包含 VARBYTE 列的实体化视图。支持包含 VARBYTE 列的实体化视图的增量刷新。但是，除了 COUNT、MIN、MAX 和 GROUP BY 以外，VARBYTE 列上的聚合函数不支持增量刷新。

为确保所有字节都是可打印字符，请 AWS Clean Rooms 使用十六进制格式打印 VARBYTE 值。例如，以下 SQL 将十六进制字符串 6162 转换为二进制值。尽管返回值为二进制值，但结果将以十六进制形式 6162 打印。

```
select from_hex('6162');

 from_hex
----------
 6162
```

AWS Clean Rooms 支持在 VARBYTE 和以下数据类型之间进行转换：

- CHAR
- VARCHAR
- SMALLINT
- INTEGER
- BIGINT

以下 SQL 语句将 VARCHAR 字符串转换为 VARBYTE。尽管返回值为二进制值，但结果将以十六进制形式 616263 打印。

```
select 'abc'::varbyte;

 varbyte
---------
 616263
```

以下 SQL 语句将列中的 CHAR 值转换为 VARBYTE。此示例创建一个包含 CHAR(10) 列（c）的表，插入长度小于 10 的字符值。生成的转换使用空格字符（hex'20'）将结果填充到定义的列大小。尽管返回值是二进制值，但结果将以十六进制形式打印。

```
create table t (c char(10));
insert into t values ('aa'), ('abc');
select c::varbyte from t;
          c
----------------------
 61612020202020202020
 61626320202020202020
```

以下 SQL 语句将 SMALLINT 字符串转换为 VARBYTE。尽管返回值是二进制值，但结果将以十六进制形式 0005 打印，为 2 个字节或 4 个十六进制字符。

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

 varbyte
---------
 0005
```

以下 SQL 语句将 INTEGER 转换为 VARBYTE。尽管返回值是二进制值，但结果将以十六进制形式 00000005 打印，为 4 个字节或 8 个十六进制字符。

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

 varbyte
----------
 00000005
```

以下 SQL 语句将 BIGINT 转换为 VARBYTE。尽管返回值是二进制值，但结果将以十六进制形式 0000000000000005 打印，为 8 个字节或 16 个十六进制字符。

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

     varbyte
------------------
 0000000000000005
```

将 VARBYTE 数据类型与 AWS Clean Rooms一起使用时的限制

以下是将 VARBYTE 数据类型与配合 AWS Clean Rooms使用时的限制：

- AWS Clean Rooms 仅支持 Parquet 和 ORC 文件的 VARBYTE 数据类型。
- AWS Clean Rooms 查询编辑器尚未完全支持 VARBYTE 数据类型。因此，在处理 VARBYTE 表达式时，请使用不同的 SQL 客户端。

  作为使用查询编辑器的解决方法，如果您的数据长度低于 64 KB 并且内容是有效的 UTF-8，那么您可以将 VARBYTE 值转换为 VARCHAR，例如：

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

- 您不能将 VARBYTE 数据类型与 Python 或 Lambda 用户定义的函数（UDF）结合使用。
- 您不能从 VARBYTE 列创建 HLLSKETCH 列，也不能在 VARBYTE 列上使用近似去重统计。

类型兼容性和转换

以下讨论介绍了类型转换规则和数据类型兼容性如何在 AWS Clean Rooms中工作。

兼容性

在各种数据库操作期间，将会出现数据类型匹配以及文本值和常量与数据类型的匹配，包括以下情况：

- 表中的数据操控语言 (DML) 操作
- UNION、INTERSECT 和 EXCEPT 查询
- CASE 表达式
- 谓词（如 LIKE 和 IN）的计算
- 执行数据比较或提取的 SQL 函数的计算
- 数学运算符的比较

这些运算的结果取决于类型转换规则和数据类型兼容性。兼容性意味着并非总是需要 one-to-one 匹配特定值和特定数据类型。由于一些数据类型是兼容的，因此可进行隐式转换或强制转换。有关更多信息，请参阅 隐式转换类型。如果数据类型不兼容，您有时可通过使用显式转换函数将值从一种数据类型转换为另一种数据类型。

一般兼容性和转换规则

请注意下列兼容性和转换规则：

- 一般来说，同属一种类型类别的数据类型（如不同的数字数据类型）是兼容的并且可隐式转换。

  例如，通过使用隐式转换，您可以将一个小数值插入整数列。小数进位为整数。或者，您可以从日期中提取一个数字值（如 2008）并将其插入到整数列中。
- 数字数据类型会强制执行尝试插入 out-of-range 值时出现的溢出条件。例如，精度为 5 的小数值无法放入到精度定义为 4 的小数列中。整数或小数的整数部分永远不会被截断。不过，小数的小数部分可以酌情向上或向下舍入。但是，对于显示强制转换表中选定的值而得出的结果，不会进行四舍五入。

- 不同类型的字符字符串是兼容的。包含单字节数据的 VARCHAR 列字符串和 CHAR 列字符串是兼容且可隐式转换的。包含多字节数据的 VARCHAR 字符串是不可兼容的。此外，如果字符串是适当的文本值，则您可以将字符字符串转换为日期、时间、时间戳或数字值。将忽略任何前导空格或尾随空格。反过来，您也可以将日期、时间、时间戳或数字值转换为固定长度或可变长度的字符串。

  > **Note**
  >
  > 您要强制转换为数字类型的字符串必须包含数字的字符表示形式。例如，您可将字符串 `'1.0'` 或 `'5.9'` 强制转换为小数值，但无法将字符串 `'ABC'` 强制转换为任何数字类型。

- 如果将 DECIMAL 值与字符串进行比较，则会 AWS Clean Rooms 尝试将字符串转换为 DECIMAL 值。在将所有其他数值与字符串进行比较时，数值将转换为字符串。如果要强制进行相反的转换（例如，将字符串转换为正数，或者将 DECIMAL 值转换为字符串），请使用显式函数，例如 CAST 函数。
- 若要将 64 位 DECIMAL 或 NUMERIC 值转换为更高的精度，必须使用显式转换函数（如 CAST 或 CONVERT）。
- 将 DATE 或 TIMESTAMP 转换为 TIMESTAMPTZ 时，或者将 TIME 转换为 TIMETZ 时，时区设置为当前会话时区。会话时区默认为 UTC。
- 与之类似，TIMESTAMPTZ 可根据当前会话时区转化为 DATE、TIME 或 TIMESTAMP。会话时区默认为 UTC。转换后，时区信息将被删除。
- 使用当前会话时区（默认为 UTC）将代表有指定时区的时间戳的字符串转换为 TIMESTAMPTZ。使用当前会话时区（默认为 UTC）将代表有指定时区的时间的字符串转换为 TIMETZ。

隐式转换类型

有两种隐式转换类型：

- 赋值中的隐式转化，如 INSERT 或 UPDATE 命令中的设置值。
- 表达式中的隐式转化，例如在 WHERE 子句中执行比较

下表列出了在赋值或表达式中可隐式转换的数据类型。您还可使用显式转换函数执行这些转换。

源类型	目标类型
BIGINT	BOOLEAN

源类型	目标类型
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
CHAR	VARCHAR
DATE	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT
	CHAR
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
DOUBLE PRECISION (FLOAT8)	BIGINT

源类型	目标类型
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
INTEGER (INT)	BIGINT
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	REAL (FLOAT4)
	SMALLINT
	VARCHAR
REAL (FLOAT4)	BIGINT
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT)
	SMALLINT
	VARCHAR

源类型	目标类型
SMALLINT	BIGINT
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT)
	REAL (FLOAT4)
	VARCHAR
TIMESTAMP	CHAR
	DATE
	VARCHAR
	TIMESTAMPTZ
	TIME
TIMESTAMPTZ	CHAR
	DATE
	VARCHAR
	TIMESTAMP
	TIMETZ
TIME	VARCHAR
	TIMETZ

源类型	目标类型
TIMETZ	VARCHAR
	TIME

> **Note**
>
> 在 TIMESTAMPTZ、TIMESTAMP、DATE、TIME、TIMETZ 或字符串之间的隐式转换使用当前会话时区。
> 无法将 VARBYTE 数据类型隐式转换为任何其它数据类型。有关更多信息，请参阅 CAST 函数。

中的 SQL 命令 AWS Clean Rooms

中支持以下 SQL 命令 AWS Clean Rooms：

主题

- SELECT

SELECT

SELECT 命令返回表和用户定义的函数中的行。

中支持以下 SELECT SQL 命令 AWS Clean Rooms：

主题

- SELECT list
- WITH 子句
- FROM 子句
- WHERE 子句
- GROUP BY 子句
- HAVING 子句
- 集合运算符
- ORDER BY 子句
- 子查询示例
- 关联的子查询

SELECT list

SELECT list 指定希望查询返回的列、函数和表达式。列表表示查询的输出。

语法

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

参数

TOP *number*

TOP 将正整数用作其参数，用于定义返回到客户端的行数。使用 TOP 子句的行为与使用 LIMIT 子句的行为相同。返回的行数是固定的，但行集不固定。要返回一致的行集，请将 TOP 或 LIMIT 与 ORDER BY 子句结合使用。

DISTINCT

一个选项，用于根据一个或多个列中的匹配值消除结果集中的重复行。

expression

由查询引用的表中存在的一个或多个列构成的表达式。表达式可包含 SQL 函数。例如：

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

AS column_alias

在最终结果集中使用的列的临时名称。AS 关键字是可选的。例如：

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

如果您没有为不是简单列名的表达式指定别名，则结果集将对该列应用默认名称。

> **Note**
>
> 在目标列表中定义别名后，它将立即被识别。您不能在其他表达式中使用在同一目标列表中晚于该别名定义的某个别名。

使用说明

TOP 是一个 SQL 扩展。TOP 提供 LIMIT 行为的替代。不能在同一个查询中使用 TOP 和 LIMIT。

WITH 子句

WITH 子句是一个可选子句，该子句在查询中位于 SELECT 列表之前。WITH 子句定义一个或多个 common_table_expressions。每个通用表表达式 (CTE) 均定义一个临时表，它与视图定义类似。您

可以在 FROM 子句中引用这些临时表。它们仅在它们所属的查询运行时使用。WITH 子句中的每个子 CTE 均指定一个表名、一个可选的列名称列表以及一个计算结果为表的查询表达式（SELECT 语句）。

WITH 子句子查询是定义可在单个查询的执行过程中使用的表的有效方式。在所有情况下，在 SELECT 语句的主体中使用子查询可获得相同的结果，不过 WITH 子句子查询可能在编写和阅读方面更加简单。如果可能，会将已引用多次的 WITH 子句子查询优化为常用子表达式；即，可以计算 WITH 子查询一次并重用其结果。（请注意，常用子表达式不只是限于 WITH 子句中定义的子表达式。）

语法

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

其中 common_table_expression 可以是非递归的。以下是非递归形式：

```
CTE_table_name AS ( query )
```

参数

common_table_expression

定义一个您可以在 FROM 子句 中引用并且仅在执行其所属的查询期间使用的临时表。

CTE_table_name

临时表的唯一名称，该临时表用于定义 WITH 子句子查询的结果。不能在单个 WITH 子句中使用重复名称。必须为每个子查询提供一个可在 FROM 子句中引用的表名。

query

任何 AWS Clean Rooms 支持的 SELECT 查询。请参阅 SELECT。

使用说明

可在以下 SQL 语句中使用 WITH 子句：

- SELECT、WITH、UNION、INTERSECT 和 EXCEPT

如果包含 WITH 子句的查询的 FROM 子句未引用 WITH 子句所定义的任何表，则将忽略 WITH 子句，并且查询将正常执行。

WITH 子句子查询所定义的表只能在 WITH 子句开始的 SELECT 查询范围内引用。例如，可以在 SELECT 列表的子查询的 FROM 子句、WHERE 子句或 HAVING 子句中引用这样的表。不能在子查询中使用 WITH 子句，也不能在主查询或其他子查询的 FROM 子句中引用其表。此查询模式会为 WITH 子句表生成 `relation table_name doesn't exist` 形式的错误消息。

不能在 WITH 子句子查询中指定另一个 WITH 子句。

不能对 WITH 子句子查询定义的表进行前向引用。例如，以下查询返回一个错误，因为在表 W1 的定义中对表 W2 进行了前向引用：

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

示例

以下示例说明了包含 WITH 语句的查询的最简单示例。在名为 VENUECOPY 的 WITH 查询中，选择 VENUE 表中的所有行。主查询又选择 VENUECOPY 中的所有行。VENUECOPY 表仅在此查询的持续时间内存在。

```
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)
```

以下示例显示一个 WITH 子句，该子句生成两个分别名为 VENUE_SALES 和 TOP_VENUES 的表。第二个 WITH 查询表从第一个表中进行选择。而主查询块的 WHERE 子句又包含一个约束 TOP_VENUES 表的子查询。

```
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)
```

以下两个示例演示基于 WITH 子句子查询的表引用范围的规则。第一个查询运行，但第二个查询失败，并出现意料中的错误。在第一个查询中，主查询的 SELECT 列表中包含 WITH 子句子查询。SELECT 列表中的子查询的 FROM 子句中将引用 WITH 子句定义的表 (HOLIDAYS)：

```
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)
```

第二个查询失败，因为它尝试在主查询和 SELECT 列表子查询中引用 HOLIDAYS 表。主查询引用超出范围。

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

ERROR:  relation "holidays" does not exist
```

FROM 子句

查询中的 FROM 子句列出从中选择数据的表引用（表、视图和子查询）。如果列出多个表引用，则必须在 FROM 子句或 WHERE 子句中使用适当的语法来联接表。如果未指定联接条件，则系统将查询作为交叉联接（笛卡尔乘积）进行处理。

主题

- 语法
- 参数

- 使用说明
- JOIN 示例

语法

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

其中，table_reference 是下列项之一：

```
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
```

参数

with_subquery_table_name

WITH 子句中的子查询定义的表。

table_name

表或视图的名称。

alias

表或视图的临时备用名称。必须为派生自子查询的表提供别名。在其他表引用中，别名是可选的。AS 关键字始终是可选的。表别名提供了用于标识查询的其他部分（例如 WHERE 子句）中的表的快捷方法。

例如：

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

如果定义了表别名，则必须使用该别名在查询中引用该表。

例如，如果查询是 SELECT "tbl"."col" FROM "tbl" AS "t"，则查询将失败，因为表名现在基本上已被覆盖。在这种情况下，有效的查询是 SELECT "t"."col" FROM "tbl" AS "t"。

column_alias

表或视图中的列的临时备用名称。

subquery

一个计算结果为表的查询表达式。表仅在查询的持续时间内存在，并且通常会向表提供一个名称或别名。但别名不是必需的。您也可以为派生自子查询的表定义列名称。如果您希望将子查询的结果联接到其他表并且希望在查询中的其他位置选择或约束这些列，则指定列的别名是非常重要的。

子查询可以包含 ORDER BY 子句，但在未指定 LIMIT 或 OFFSET 子句的情况下，该子句可能没有任何作用。

NATURAL

定义一个联接，该联接自动将两个表中同名列的所有配对用作联接列。不需要显式联接条件。例如，如果 CATEGORY 和 EVENT 表都具有名为 CATID 的列，则这两个表的自然联接为基于其 CATID 列的联接。

> **Note**
>
> 如果指定 NATURAL 联接，但表中没有要联接的同名列配对，则查询默认为交叉联接。

join_type

指定下列类型的联接之一：

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

交叉联接是未限定的联接；它们返回两个表的笛卡尔乘积。

内部联接和外部联接是限定的联接。它们的限定方式包括：隐式（在自然联接中）；在 FROM 语句中使用 ON 或 USING 语法；或者使用 WHERE 子句条件。

内部联接仅基于联接条件或联接列的列表返回匹配的行。外部联接返回与内部联接相同的所有行，还返回“左侧”表和/或“右侧”表中的非匹配行。左侧表是第一个列出的表，右侧表是第二个列出的表。非匹配行包含 NULL 值以填补输出列中的空白。

ON join_condition

联接规范的类型，其中将联接列声明为紧跟 ON 关键字的条件。例如：

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

USING (join_column [, ...])

联接规范的类型，其中用圆括号将列出的联接列括起来。如果指定多个联接列，则用逗号将它们分隔开。USING 关键字必须在列表之前。例如：

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

使用说明

联接列必须具有可比较的数据类型。

NATURAL 或 USING 联接仅将每对联接列中的一个联接列保留在中间结果集中。

使用 ON 语法的联接会将两个联接列都保留在其中间结果集中。

另请参阅 WITH 子句。

JOIN 示例

SQL JOIN 子句用于根据公共字段合并两个或多个表中的数据。根据指定的联接方法，结果可能会发生变化，也可能不发生变化。有关 JOIN 子句的语法的更多信息，请参阅参数。

下面的查询是 LISTING 表和 SALES 表之间的内部联接（不带 JOIN 关键字），其中 LISTING 表中的 LISTID 介于 1 和 5 之间。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果显示 LISTID 1、4 和 5 符合条件。

```
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
```

以下查询是一个左外部联接。当在其他表中找不到匹配项时，左外部联接和右外部联接保留某个已联接表中的值。左表和右表是语法中列出的第一个表和第二个表。NULL 值用于填补结果集中的“空白”。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果表明 LISTID 2 和 3 不会生成任何销售额。

```
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
```

以下查询是一个右外部联接。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果显示 ListID 1、4 和 5 符合条件。

```
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
```

以下查询是一个完全联接。当在其他表中找不到匹配项时，完全联接保留已联接表中的值。左表和右表是语法中列出的第一个表和第二个表。NULL 值用于填补结果集中的“空白”。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果表明 LISTID 2 和 3 不会生成任何销售额。

```
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
```

以下查询是一个完全联接。此查询匹配 LISTING 表（左表）和 SALES 表（右表）中的 LISTID 列值。结果中只有不会导致任何销售额的行（ListID 2 和 3）。

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

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

以下示例是与 ON 子句的内部联接。在这种情况下，不返回 NULL 行。

```
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
```

以下查询是 LISTING 表和 SALES 表的交叉联接或笛卡尔联接，其中包含限制结果的谓词。此查询匹配 SALES 表和 LISTING 表中的 LISTID 列值，对应于这两个表中的 LISTID 1、2、3、4 和 5。结果显示 20 个行符合条件。

```
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
```

以下示例是两个表之间的自然联接。在这种情况下，列 listid、sellerid、eventid 和 dateid 在两个表中具有相同的名称和数据类型，因此用作联接列。结果限制为 5 行。

```
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
```

以下示例是使用 USING 子句在两个表之间进行的联接。在这种情况下，列 listid 和 eventid 用作联接列。结果限制为 5 行。

```
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
```

以下查询是 FROM 子句中的两个子查询的内部联接。此查询查找不同类别的活动（音乐会和演出）的已售门票数和未售门票数：这些 FROM 子句子查询是表 子查询；它们可返回多个列和行。

```
select catgroup1, sold, unsold
from
(select catgroup, sum(qtysold) as sold
from category c, event e, sales s
where c.catid = e.catid and e.eventid = s.eventid
group by catgroup) as a(catgroup1, sold)
join
(select catgroup, sum(numtickets)-sum(qtysold) as unsold
from category c, event e, sales s, listing l
```

```
where c.catid = e.catid and e.eventid = s.eventid
and s.listid = l.listid
group by catgroup) as b(catgroup2, unsold)

on a.catgroup1 = b.catgroup2
order by 1;

catgroup1 |  sold  | unsold
----------+--------+--------
Concerts  | 195444 |1067199
Shows     | 149905 | 817736
```

WHERE 子句

WHERE 子句包含用于联接表或将谓词应用于表中的列的条件。可在 WHERE 子句或 FROM 子句中使用适当的语法对表进行内部联接。外部联接条件必须在 FROM 子句中指定。

语法

```
[ WHERE condition ]
```

condition

任何具有布尔型结果的搜索条件，例如，联接条件或表列上的谓词。以下示例是有效的联接条件：

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

以下示例是表中的列上的有效条件：

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

条件可以是简单条件或复杂条件；对于复杂条件，可以使用圆括号来分隔逻辑单元。在下面的示例中，用圆括号将联接条件括起来。

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

使用说明

您可在 WHERE 子句中使用别名来引用选择列表表达式。

不能限制 WHERE 子句中的聚合函数的结果；要实现此目的，请使用 HAVING 子句。

WHERE 子句中受限制的列必须派生自 FROM 子句中的表引用。

示例

以下查询使用不同的 WHERE 子句限制的组合，包括 SALES 表和 EVENT 表的联接条件、EVENTNAME 列上的谓词以及 STARTTIME 列上的两个谓词。

```
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Hannah Montana'
and date_part(quarter, starttime) in(1,2)
and date_part(year, starttime) = 2008
order by 3 desc, 4, 2, 1 limit 10;

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

GROUP BY 子句

GROUP BY 子句标识查询的分组列。必须在查询使用标准函数（例如，SUM、AVG 和 COUNT）计算聚合时声明分组列。如果 SELECT 表达式中存在聚合函数，则 SELECT 表达式中不在聚合函数中的任何列都必须位于 GROUP BY 子句中。

有关更多信息，请参阅中的 SQL 函数 AWS Clean Rooms。

语法

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

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

参数

expr

列或表达式的列表必须匹配查询的选择列表中的非聚合表达式的列表。例如，考虑以下简单查询。

```
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)
```

在此查询中，选择列表包含两个聚合表达式。第一个聚合表达式使用 SUM 函数，第二个聚合表达式使用 COUNT 函数。必须将其余两个列（LISTID 和 EVENTID）声明为分组列。

GROUP BY 子句中的表达式也可以使用序号来引用选择列表。例如，上一个示例的缩略形式如下。

```
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

您可以使用聚合扩展 ROLLUP 在单个语句中执行多个 GROUP BY 操作。有关聚合扩展和相关函数的更多信息，请参阅聚合扩展。

聚合扩展

AWS Clean Rooms 支持聚合扩展，以便在单个语句中执行多个 GROUP BY 操作。

GROUPING SETS

在单个语句中计算一个或多个分组集。分组集是单个 GROUP BY 子句的集合，这是一组 0 列或更多列，您可以通过这些列对查询的结果集进行分组。GROUP BY GROUPING SETS 等效于对一个按不同列分组的结果集运行 UNION ALL 查询。例如，GROUP BY GROUPING SETS((a), (b)) 等效于 GROUP BY a UNION ALL GROUP BY b。

以下示例返回按产品类别和所售产品类型分组的订单表产品的成本。

```
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

假设在一个层次结构中，前面的列被视为后续列的父列。ROLLUP 按提供的列对数据进行分组，除了分组行之外，还返回额外的小计行，表示所有分组列级别的总计。例如，您可以使用 GROUP BY ROLLUP((a), (b)) 返回先按 a 分组的结果集，然后在假设 b 是 a 的一个子部分的情况下按 b 分组。ROLLUP 还会返回包含整个结果集而不包含分组列的行。

GROUP BY ROLLUP((a), (b)) 等效于 GROUP BY GROUPING SETS((a,b), (a), ())。

以下示例返回先按类别分组，然后按产品分组，且产品是类别细分项的订单表产品的成本。

```
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

按提供的列对数据进行分组，除了分组行之外，还返回额外的小计行，表示所有分组列级别的总计。CUBE 返回与 ROLLUP 相同的行，同时为 ROLLUP 未涵盖的每个分组列组合添加额外的小计行。例如，您可以使用 GROUP BY CUBE ((a), (b)) 返回先按 a 分组的结果集，然后在假设 b 是 a 的一个子部分的情况下按 b 分组，再然后是单独按 b 分组的结果集。CUBE 还会返回包含整个结果集而不包含分组列的行。

GROUP BY CUBE((a), (b)) 等效于 GROUP BY GROUPING SETS((a, b), (a), (b), ())。

以下示例返回先按类别分组，然后按产品分组，且产品是类别细分项的订单表产品的成本。与前面的 ROLLUP 示例不同，该语句返回每个分组列组合的结果。

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY CUBE(category, product) ORDER BY 1,2;

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

HAVING 子句

HAVING 子句将条件应用于查询返回的中间分组结果集。

语法

```
[ HAVING condition ]
```

例如，您可以限制 SUM 函数的结果：

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

在应用所有 WHERE 子句条件并完成 GROUP BY 操作后，应用 HAVING 条件。

条件本身采用与任何 WHERE 子句条件相同的形式。

使用说明

- HAVING 子句条件中引用的任何列必须为分组列或引用了聚合函数结果的列。
- 在 HAVING 子句中，无法指定：
  - 引用选择列表项的序号。仅 GROUP BY 和 ORDER BY 子句接受序号。

示例

以下查询按名称计算所有活动的门票总销售额，然后消除总销售额小于 $800000 的活动。HAVING 条件应用于选择列表中聚合函数的结果：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)
```

以下查询计算类似的结果集。不过，在本示例中，HAVING 条件将应用于未在选择列表中指定的聚合：sum(qtysold)。将从最终结果中消除未售出 2000 张以上的门票的活动。

```
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)
```

集合运算符

UNION、INTERSECT 和 EXCEPT 集合运算符 用于比较和合并两个单独的查询表达式的结果。例如，如果您希望知道网站的哪些用户既是买家又是卖家且其用户名存储在单独的列或表中，则可查找这两类用户的交集。如果您希望知道哪些网站用户是买家而不是卖家，则可使用 EXCEPT 运算符查找这两个用户列表的差集。如果您希望构建一个所有用户的列表（无论角色如何），则可使用 UNION 运算符。

> **Note**
>
> 不能在 UNION、UNION ALL、INTERSECT 和 EXCEPT 集合运算符合并的查询表达式中使用 ORDER BY、LIMIT、SELECT TOP 和 OFFSET 子句。

主题

语法

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

参数

query

一个查询表达式，该表达式（采用其选择列表形式）对应于紧跟 UNION、INTERSECT 或 EXCEPT 运算符的第二个查询表达式。这两个表达式必须包含数量相同并且数据类型兼容的输出

列；否则，无法比较和合并两个结果集。集合运算不允许不同类别的数据类型之间的隐式转换；有关更多信息，请参阅 类型兼容性和转换。

您可以构建包含无限数量的查询表达式并任意组合使用 UNION、INTERSECT 和 EXCEPT 运算符来将这些表达式链接起来的查询。例如，假定表 T1、T2 和 T3 包含兼容的列集，则以下查询结构是有效的：

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

联合

从两个查询表达式返回行的集合运算，无论行派生自一个查询表达式还是两个查询表达式。

INTERSECT

返回派生自两个查询表达式的行的集合运算。将丢弃未同时由两个表达式返回的行。

EXCEPT | MINUS

返回派生自两个查询表达式之一的行的集合运算。要符合结果的要求，行必须存在于第一个结果表而不存在于第二个结果表中。MINUS 和 EXCEPT 完全同义。

ALL

ALL 关键字保留由 UNION 生成的任何重复行。未使用 ALL 关键字时的默认行为是丢弃这些重复项。不支持 INTERSECT ALL、EXCEPT ALL 和 MINUS ALL。

集合运算符的计算顺序

UNION 和 EXCEPT 集合运算符是左关联的。如果未指定圆括号来影响优先顺序，则将以从左到右的顺序来计算这些集合运算符的组合。例如，在以下查询中，首先计算 T1 和 T2 的 UNION，然后对 UNION 结果执行 EXCEPT 操作：

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

在同一个查询中使用运算符组合时，INTERSECT 运算符优先于 UNION 和 EXCEPT 运算符。例如，以下查询将计算 T2 和 T3 的交集，然后计算得到的结果与 T1 的并集：

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

通过添加圆括号，可以强制实施不同的计算顺序。在以下示例中，将 T1 和 T2 的并集结果与 T3 执行交集运算，并且查询可能会生成不同的结果。

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

使用说明

- 集合运算查询结果中返回的列名是来自第一个查询表达式中的表的列名（或别名）。由于这些列名可能会造成误解（因为列中的值派生自位于集合运算符任一侧的表），您可能需要为结果集提供有意义的别名。
- 当集合运算符查询返回小数结果时，将提升对应的结果列以返回相同的精度和小数位数。例如，在以下查询中，T1.REVENUE 为 DECIMAL(10,2) 列而 T2.REVENUE 为 DECIMAL(8,4) 列，小数结果将提升为 DECIMAL(12,4)：

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

  小数位数为 4，因为这是两个列的最大小数位数。精度为 12，因为 T1.REVENUE 要求小数点左侧有 8 位数 (12 - 4 = 8)。此类提升可确保 UNION 两侧的所有值都适合结果。对于 64 位值，最大结果精度为 19，最大结果小数位数为 18。对于 128 位值，最大结果精度为 38，最大结果小数位数为 37。

  如果生成的数据类型超过 AWS Clean Rooms 精度和小数位数限制，则查询将返回错误。
- 对于集合运算，如果对于每个相应的列对，两个数据值相等 或都为 NULL，则两个行将被视为相同。例如，如果表 T1 和 T2 都包含一列和一行，并且两个表中的行都为 NULL，则对这两个表执行的 INTERSECT 运算将返回该行。

示例 UNION 查询

在以下 UNION 查询中，SALES 表中的行将与 LISTING 表中的行合并。从每个表中选择三个兼容的列；在这种情况下，对应的列具有相同的名称和数据类型。

```
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
```

以下示例说明如何将文本值添加到 UNION 查询的输出，以便您查看哪个查询表达式生成了结果集中的每一行。查询将第一个查询表达式中的行标识为“B”（针对买家），并将第二个查询表达式中的行标识为“S”（针对卖家）。

查询标识门票事务费用等于或大于 $10000 的买家和卖家。UNION 运算符的任一侧的两个查询表达式之间的唯一差异就是 SALES 表的联接列。

```
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
```

以下示例使用 UNION ALL 运算符，因为需要在结果中保留重复行（如果发现重复行）。对于一系列特定的活动 ID，查询为与每个活动关联的每个销售值返回 0 行或多个行，并为该活动的每个列表返回 0 行或 1 个行。活动 ID 对于 LISTING 和 EVENT 表中的每个行是唯一的，但对于 SALES 表中的活动和列表 ID 的相同组合，可能有多个销售值。

结果集中的第三个列标识行的来源。如果行来自 SALES 表，则在 SALESROW 列中将其标记为“Yes”。（SALESROW 是 SALES.LISTID 的别名。）如果行来自 LISTING 表，则在 SALESROW 列中将其标记为“No”。

在本示例中，结果集包含针对列表 500，活动 7787 的三个销售行。换而言之，将针对此列表和活动组合执行三个不同的事务。其他两个列表（501 和 502）不生成任何销售值，因此只有查询为这些列表 ID 生成的行来自 LISTING 表 (SALESROW = 'No')。

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

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

如果运行不带 ALL 关键字的相同查询，则结果只保留其中一个销售交易。

```
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
```

示例 UNION ALL 查询

以下示例使用 UNION ALL 运算符，因为需要在结果中保留重复行（如果发现重复行）。对于一系列特定的活动 ID，查询为与每个活动关联的每个销售值返回 0 行或多个行，并为该活动的每个列表返回 0 行或 1 个行。活动 ID 对于 LISTING 和 EVENT 表中的每个行是唯一的，但对于 SALES 表中的活动和列表 ID 的相同组合，可能有多个销售值。

结果集中的第三个列标识行的来源。如果行来自 SALES 表，则在 SALESROW 列中将其标记为“Yes”。（SALESROW 是 SALES.LISTID 的别名。）如果行来自 LISTING 表，则在 SALESROW 列中将其标记为“No”。

在本示例中，结果集包含针对列表 500，活动 7787 的三个销售行。换而言之，将针对此列表和活动组合执行三个不同的事务。其他两个列表（501 和 502）不生成任何销售值，因此只有查询为这些列表 ID 生成的行来自 LISTING 表 (SALESROW = 'No')。

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

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

如果运行不带 ALL 关键字的相同查询，则结果只保留其中一个销售交易。

```
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
```

示例 INTERSECT 查询

将以下示例与第一个 UNION 示例进行比较。这两个示例之间的唯一差异是所使用的集合运算符，但结果完全不同。仅其中一行相同：

```
235494 |    23875 |    8771
```

这是在包含 5 行的有限结果中，同时在两个表中找到的唯一行。

```
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
```

下面的查询查找 3 月份同时在纽约和洛杉矶举办的活动（已销售这些活动的门票）。这两个查询表达式之间的差异是 VENUECITY 列上的约束。

```
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
```

示例 EXCEPT 查询

数据库中的 CATEGORY 表包含以下 11 行：

```
 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)
```

假定 CATEGORY_STAGE 表（临时表）包含一个额外行：

```
 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)
```

返回两个表之间的差异。换而言之，返回 CATEGORY_STAGE 表中存在但 CATEGORY 表中不存在的行：

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

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

以下等效查询使用同义词 MINUS。

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

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

如果反转 SELECT 表达式的顺序，则查询不返回任何行。

ORDER BY 子句

ORDER BY 子句对查询的结果集进行排序。

> **Note**
>
> 最外面的 ORDER BY 表达式必须仅包含位于选择列表中的列。

主题

语法

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

参数

expression

定义查询结果排序顺序的表达式。它由选择列表中的一个或多个列组成。根据二进制 UTF-8 排序方式返回结果。您也可以指定：

- 表示选择列表条目的位置（如果不存在选择列表，则为表中列的位置）的序号
- 定义选择列表条目的别名

当 ORDER BY 子句包含多个表达式时，将根据第一个表达式对结果集进行排序，然后将第二个表达式应用于具有第一个表达式中的匹配值的行，以此类推。

ASC | DESC

一个定义表达式的排序顺序的选项，如下所示：

- ASC：升序（例如，按数值的从低到高的顺序和字符串的从 A 到 Z 的顺序）。如果未指定选项，则默认情况下将按升序对数据进行排序。
- DESC：降序（按数值的从高到低的顺序和字符串的从 Z 到 A 的顺序）。

NULLS FIRST | NULLS LAST

一个选项，指定是应将 NULL 值排在最前（位于非 null 值之前）还是排在最后（位于非 null 值之后）。默认情况下，按 ASC 顺序最后对 NULL 值进行排序和排名，按 DESC 顺序首先对 NULL 值进行排序和排名。

LIMIT number | ALL

一个选项，用于控制查询返回的排序行的数目。LIMIT 数字必须为正整数；最大值为 2147483647。

LIMIT 0 不返回任何行。可以使用此语法进行测试：检查查询运行（不显示任何行）或返回表中列的列表。如果使用 LIMIT 0 返回列的列表，则 ORDER BY 子句是多余的。默认值为 LIMIT ALL。

OFFSET start

一个选项，指定在开始返回行之前跳过 start 前的行数。OFFSET 数字必须为正整数；最大值为 2147483647。在与 LIMIT 选项结合使用时，将先跳过 OFFSET 行，然后再开始计算返回的 LIMIT 行数。如果不使用 LIMIT 选项，则结果集中的行数会减少跳过的行数。仍必须扫描 OFFSET 子句跳过的行，因此使用较大的 OFFSET 值可能会非常低效。

使用说明

请注意，使用 ORDER BY 子句时预期会发生以下行为：

- NULL 值被视为“高于”所有其他值。对于默认的升序排序顺序，NULL 值将排在最后。要更改此行为，请使用 NULLS FIRST 选项。
- 当查询不包含 ORDER BY 子句时，系统将返回具有不可预测的行顺序的结果集。同一查询执行两次可能会返回具有不同顺序的结果集。
- 可在不使用 ORDER BY 子句的情况下使用 LIMIT 和 OFFSET 选项；不过，要返回一致的行集，请将这两个选项与 ORDER BY 子句结合使用。
- 在任何并行系统中 AWS Clean Rooms，例如，当 ORDER BY 不生成唯一排序时，行的顺序是不确定的。也就是说，如果 ORDER BY 表达式生成重复的值，则这些行的返回顺序可能因其他系统而异，也可能因运行一次而异。 AWS Clean Rooms
- AWS Clean Rooms 不支持 ORDER BY 子句中的字符串文字。

使用 ORDER BY 的示例

返回 CATEGORY 表中的所有 11 行，这些行按第二列 CATGROUP 进行排序。对于具有相同 CATGROUP 值的结果，按字符串长度对 CATDESC 列值进行排序。然后，按列 CATID 和 CATNAME 排序。

```
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)
```

返回 SALES 表中的选定列（按最高的 QTYSOLD 值排序）。将结果限制为前 10 行：

```
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
```

通过使用 LIMIT 0 语法返回列的列表，但不返回行：

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

子查询示例

以下示例说明子查询适合 SELECT 查询的不同方式。有关使用子查询的另一个示例，请参阅JOIN 示例。

SELECT 列表子查询

以下示例在 SELECT 列表中包含一个子查询。此子查询是标量：它只返回一列和一个值，该值将在从外部查询返回的每个行的结果中重复。此查询将子查询计算出的 Q1SALES 值与外部查询定义的 2008 年其他两个季度（第 2 季度和第 3 季度）的销售值进行比较。

```
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)
```

WHERE 子句子查询

以下示例在 WHERE 子句中包含一个表子查询。此子查询生成多个行。在本示例中，行只包含一列，但表子查询可以包含多个列和行，就像任何其他表一样。

此查询查找门票销量排名前 10 位的卖家。前 10 位卖家的列表受子查询的限制，这将删除居住在设有售票点的城市的用户。可以使用不同的方式编写此查询；例如，可将子查询重新编写为主查询中的联接。

```
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)
```

WITH 子句子查询

请参阅 WITH 子句。

关联的子查询

以下示例将关联子查询 包含在 WHERE 子句中；此类型的子查询包含其列与由外部查询生成的列之间的一个或多个关联。在本示例中，关联为 `where s.listid=l.listid`。对于外部查询生成的每一行，将执行子查询以限定或取消限定行。

```
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)
```

不支持的关联子查询模式

查询计划程序使用名为“子查询去相关性”的查询重写方法来优化多个关联子查询模式以便在 MPP 环境中执行。有几种类型的关联子查询遵循 AWS Clean Rooms 无法取消关联且不支持的模式。包含以下关联引用的查询会返回错误：

- 跳过查询块的关联引用，也称为“跨级关联引用”。例如，在以下查询中，包含关联引用的块与跳过的块由 NOT EXISTS 谓词连接：

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

  在本示例中，跳过的块是针对 LISTING 表执行的子查询。关联引用将 EVENT 表和 SALES 表关联起来。
- 来自作为外部查询中 ON 子句的一部分的子查询的关联引用：

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

  ON 子句包含从子查询中的 SALES 到外部查询中的 EVENT 的关联引用。
- 对 AWS Clean Rooms 系统表的空敏感关联引用。例如：

  ```
  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);
  ```
- 来自包含窗口函数的子查询内部的关联引用。

  ```
  select listid, qtysold
  from sales s
  ```

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

- GROUP BY 列中对关联查询结果的引用。例如：

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

- 来自带聚合函数和 GROUP BY 子句（通过 IN 谓词连接到外部查询）的关联引用。（此限制不适用于 MIN 和 MAX 聚合函数。）例如：

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

中的 SQL 函数 AWS Clean Rooms

AWS Clean Rooms 支持以下 SQL 函数：

主题

聚合函数

AWS Clean Rooms 支持以下聚合函数：

主题

- MEDIAN 函数
- MIN 函数
- PERCENTILE_CONT 函数
- STDDEV_SAMP 和 STDDEV_POP 函数
- SUM 和 SUM DISTINCT 函数
- VAR_SAMP 和 VAR_POP 函数

ANY_VALUE 函数

ANY_VALUE 函数以非确定方式返回输入表达式值中的任何值。如果输入表达式未导致任何行被返回，则此函数可以返回 NULL。

语法

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

参数

DISTINCT | ALL

指定 DISTINCT 或 ALL 以从输入表达式值中返回任何值。DISTINCT 参数没有任何效果，将被忽略。

expression

对其执行函数的目标列或表达式。表达式为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISON
- BOOLEAN
- CHAR
- VARCHAR
- DATE

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

返回值

返回与 expression 相同的数据类型。

使用说明

如果为列指定 ANY_VALUE 函数的语句也包含第二列引用，则第二列必须出现在 GROUP BY 子句中或包含在聚合函数中。

示例

以下示例返回任dateid意 wher eventname e 的实例Eagles。

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

以下是结果。

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

以下示例返回任意 wher dateid e eventname 为Eagles或的实例Cold War Kids。

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

以下是结果。

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

```
 1878  | Eagles
```

APPROXIMATE PERCENTILE_DISC 函数

APPROXIMATE PERCENTILE_DISC 是一种假定离散分布模型的逆分布函数。该函数具有一个百分比值和一个排序规范，并返回给定集合中的元素。近似值可以大幅加快函数运行速度，相对错误率较低，约为 0.5%。

如果给定百分位数值，APPROXIMATE PERCENTILE_DISC 会使用分位数摘要算法估计 ORDER BY 子句中的表达式的离散百分位数。APPROXIMATE PERCENTILE_DISC 返回的值具有最小的积累分布值（针对同一排序规范），该值大于或等于百分位数。

APPROXIMATE PERCENTILE_DISC 是仅计算节点函数。如果查询未引用用户定义的表或 AWS Clean Rooms 系统表，则该函数会返回错误。

语法

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

参数

percentile

介于 0 和 1 之间的数字常数。计算中将忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定用于排序和计算百分比的数字或日期/时间值的子句。

返回值

与 WITHIN GROUP 子句中的 ORDER BY 表达式相同的数据类型。

使用说明

如果 APPROXIMATE PERCENTILE_DISC 语句包括 GROUP BY 子句，结果集将受限。这一限制将根据节点类型和节点数量发生变化。如果超出限制，函数将失败并会返回以下错误。

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

如果您需要评估的组数量超出了限制，请考虑使用 PERCENTILE_CONT 函数。

示例

以下示例返回销售数量、销售总额和排名前十位日期的第五十个百分位数。

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

caldate    | count | sum        | percentile_disc
-----------+-------+------------+----------------
2008-01-07 |   658 | 2081400.00 |         2020.00
2008-01-02 |   614 | 2064840.00 |         2178.00
2008-07-22 |   593 | 1994256.00 |         2214.00
2008-01-26 |   595 | 1993188.00 |         2272.00
2008-02-24 |   655 | 1975345.00 |         2070.00
2008-02-04 |   616 | 1972491.00 |         1995.00
2008-02-14 |   628 | 1971759.00 |         2184.00
2008-09-01 |   600 | 1944976.00 |         2100.00
2008-07-29 |   597 | 1944488.00 |         2106.00
2008-07-23 |   592 | 1943265.00 |         1974.00
```

AVG 函数

AVG 函数返回输入表达式值的平均值（算术均值）。AVG 函数使用数值并忽略 NULL 值。

语法

```
AVG (column)
```

Arguments

column

对其执行函数的目标列。列为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- DOUBLE

数据类型

AVG 函数支持的参数类型有 SMALLINT、INTEGER、BIGINT、DECIMAL 和 DOUBLE。

AVG 函数支持的返回类型为：

- 适用于任何整数类型参数的 BIGINT
- 适用于浮点参数的 DOUBLE
- 返回与任何其他参数类型的表达式相同的数据类型

带有 DECIMAL 参数的 AVG 函数结果的默认精度为 38。结果的小数位数与参数的小数位数相同。例如，DEC(5,2) 列的 AVG 返回 DEC(38,2) 数据类型。

示例

从 SALES 表中查找每笔交易所售的平均产品数。

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

BOOL_AND 函数

BOOL_AND 函数在单个布尔/整数列或表达式上运行。此函数将类似的逻辑应用于 BIT_AND 和 BIT_OR 函数。对于此函数，返回类型为布尔值（`true` 或 `false`）。

如果集合中的所有值为 true，则 BOOL_AND 函数返回 `true` (`t`)。如果任何值为 false，该函数返回 `false` (`f`)。

语法

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

参数

expression

对其执行函数的目标列或表达式。此表达式必须具有 BOOLEAN 或整数数据类型。该函数的返回类型为 BOOLEAN。

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算结果之前消除指定表达式的所有重复值。利用参数 ALL，该函数可保留所有重复值。ALL 是默认值。

示例

您可以对布尔表达式或整数表达式使用布尔函数。

例如，以下查询从 TICKIT 数据库中的标准 USERS 表返回结果，该表包含多个布尔列。

BOOL_AND 函数对所有 5 个行返回 `false`。并非每个州的所有用户都喜欢运动。

```
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)
```

BOOL_OR 函数

BOOL_OR 函数在单个布尔/整数列或表达式上运行。此函数将类似的逻辑应用于 BIT_AND 和 BIT_OR 函数。对于此函数，返回类型为布尔值（`true`、`false` 或 NULL）。

如果集合中的某个值为 `true`，则 BOOL_OR 函数返回 `true`（`t`）。如果集合中的某个值为 `false`，则该函数返回 `false`（`f`）。如果值未知，则可以返回 NULL。

语法

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

参数

expression

对其执行函数的目标列或表达式。此表达式必须具有 BOOLEAN 或整数数据类型。该函数的返回类型为 BOOLEAN。

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算结果之前消除指定表达式的所有重复值。利用参数 ALL，该函数可保留所有重复值。ALL 是默认值。

示例

您可以将布尔函数与布尔表达式或整数表达式结合使用。例如，以下查询从 TICKIT 数据库中的标准 USERS 表返回结果，该表包含多个布尔列。

BOOL_OR 函数对所有 5 个行返回 `true`。每个州中至少有一个用户喜欢运动。

```
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)
```

以下示例返回 NULL。

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

```
NULL
```

COUNT 和 COUNT DISTINCT 函数

该COUNT函数对表达式定义的行进行计数。该COUNT DISTINCT函数计算列或表达式中不同的非 NULL 值的数量。在进行计数之前，它会清除指定表达式中的所有重复值。

语法

```
COUNT (column)
```

```
COUNT (DISTINCT column)
```

Arguments

column

对其执行函数的目标列。

数据类型

COUNT函数和COUNT DISTINCT函数支持所有参数数据类型。

COUNT DISTINCT函数返回BIGINT。

示例

统计来自佛罗里达州的所有用户。

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

计算EVENT表格中所有唯一的场地 ID。

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

COUNT 函数

COUNT 函数对由表达式定义的行计数。

COUNT 函数具有以下变体。

- COUNT (*) 对目标表中的所有行计数，无论它们是否包含 null 值。
- COUNT (expression) 计算某个特定列或表达式中带非 NULL 值的行的数量。
- COUNT (DISTINCT expression) 计算某个列或表达式中非重复的非 NULL 值的数量。
- APPROXIMATE COUNT DISTINCT 估算某个列或表达式中非重复的非 NULL 值的数量。

语法

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

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

```
APPROXIMATE COUNT ( DISTINCT expression )
```

参数

expression

对其执行函数的目标列或表达式。COUNT 函数支持所有参数数据类型。

DISTINCT | ALL

利用参数 DISTINCT，该函数可在执行计数之前消除指定表达式中的所有重复值。利用参数 ALL，该函数可保留表达式中的所有重复值以进行计数。ALL 是默认值。

APPROXIMATE

与近似值一起使用时，COUNT DISTINCT 函数使用 HyperLogLog 算法来近似列或表达式中不同的非 NULL 值的数量。使用 APPROXIMATE 关键字的查询，运行速度会快得多，错误率相对较低，约为 2%。返回大量非重复值的查询可提供近似值，每个查询或组（如果有按子句划分的组）中有几百万或更多非重复值。对于较少数量（以千为单位）的非重复值，近似值计算可能慢于精确计数。APPROXIMATE 只能与 COUNT DISTINCT 结合使用。

返回类型

COUNT 函数返回 BIGINT。

示例

对来自佛罗里达州的所有用户计数：

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

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

对 EVENT 表中的所有事件名称计数：

```
select count(eventname) from event;

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

对 EVENT 表中的所有事件名称计数：

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

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

对 EVENT 表中的所有唯一场地 ID 计数：

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

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

计算每个卖家列出 4 张以上门票出售的批次的次数。按卖家 ID 对结果进行分组：

```
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
...
```

以下示例比较 COUNT 和 APPROXIMATE COUNT 的返回值与执行时间。

```
select  count(distinct pricepaid) from sales;

count
-------
  4528


Time: 48.048 ms


select approximate count(distinct pricepaid) from sales;

count
-------
  4553


Time: 21.728 ms
```

LISTAGG 函数

对于查询中的每个组，LISTAGG 聚合函数根据 ORDER BY 表达式对该组的行进行排序，然后将值串联成一个字符串。

LISTAGG 是仅计算节点函数。如果查询未引用用户定义的表或 AWS Clean Rooms 系统表，则该函数会返回错误。

语法

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

参数

DISTINCT

（可选）用于在串联之前消除指定表达式中重复值的子句。尾部空格将被忽略，因此会将字符串 'a' 和 'a ' 视为重复值。LISTAGG 将使用遇到的第一个值。有关更多信息，请参阅尾部空格的意义。

aggregate_expression

提供要聚合的值的任何有效表达式（如列名称）。忽略 NULL 值和空字符串。

分隔符

(可选) 用于分隔串联的值的字符串常数。默认值为 NULL。

AWS Clean Rooms 支持在可选的逗号或冒号周围使用任意数量的前导或尾随空格，以及空字符串或任意数量的空格。

有效值示例为：

", "

": "

" "

WITHIN GROUP (ORDER BY order_list)

(可选) 用于指定聚合值的排序顺序的子句。

返回值

VARCHAR(MAX)。如果结果集大于最大 VARCHAR 大小（64K – 1 或 65535），则 LISTAGG 返回以下错误：

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

使用说明

如果语句包含多个使用 WITHIN GROUP 子句的 LISTAGG 函数，则每个 WITHIN GROUP 子句必须使用相同的 ORDER BY 值。

例如，以下语句将返回错误。

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

以下语句将成功运行。

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

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

示例

以下示例聚合卖家 ID（按卖家 ID 进行排序）。

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

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

以下示例使用 DISTINCT 返回唯一卖家 ID 的列表。

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

listagg

-------------------------------------------------------------------------------------------
```

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

以下示例按日期顺序聚合卖家 ID。

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

   listagg
-------------
 31141242333
```

以下示例返回买家 B 的销售日期的竖线分隔的列表。

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

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

以下示例返回每个买家 ID 的销售 ID 的逗号分隔的列表。

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

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

MAX 函数

MAX 函数返回一组行中的最大值。可以使用 DISTINCT 或 ALL，但不会影响结果。

语法

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

参数

expression

对其执行函数的目标列或表达式。表达式为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISON
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算最大值之前消除指定表达式中的所有重复值。利用参数 ALL，该函数可保留表达式中的所有重复值以计算最大值。ALL 是默认值。

数据类型

返回与 expression 相同的数据类型。

示例

从所有销售中查找支付的最高价格：

```
select max(pricepaid) from sales;

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

从所有销售中查找每张门票的已支付最高价格：

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

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

MEDIAN 函数

计算一系列值的中值。忽略该范围中的 NULL 值。

MEDIAN 是一种假定连续分布模型的逆分布函数。

MEDIAN 是 PERCENTILE_CONT(.5) 的特殊情况。

MEDIAN 是仅计算节点函数。如果查询未引用用户定义的表或 AWS Clean Rooms 系统表，则该函数会返回错误。

语法

```
MEDIAN ( median_expression )
```

参数

median_expression

对其执行函数的目标列或表达式。

数据类型

返回类型由 median_expression 的数据类型确定。下表显示了每种 median_expression 数据类型的返回类型。

输入类型	返回类型
NUMERIC、DECIMAL	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

使用说明

如果 median_expression 参数是使用 38 位最大精度定义的 DECIMAL 数据类型，则 MEDIAN 可能将返回不准确的结果或错误。如果 MEDIAN 函数的返回值超过 38 位，则结果将截断以符合规范，这将导致精度降低。如果在插值期间，中间结果超出最大精度，则会发生数值溢出且函数会返回错误。要避免这些情况，建议使用具有较低精度的数据类型或将 median_expression 参数转换为较低精度。

如果语句包括对基于排序的聚合函数 (LISTAGG、PERCENTILE_CONT 或 MEDIAN) 的多个调用，则它们必须全都使用相同的 ORDER BY 值。请注意，MEDIAN 对表达式值应用隐式排序依据。

例如，以下语句将返回错误。

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

An error occurred when executing the SQL command:
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (pricepaid)
from sales group by salesid, pricepai...
```

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

以下语句将成功运行。

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

示例

以下示例显示，MEDIAN 生成和 PERCENTILE_CONT(0.5) 相同的结果。

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

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

MIN 函数

MIN 函数返回一组行中的最小值。可以使用 DISTINCT 或 ALL，但不会影响结果。

语法

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

参数

expression

对其执行函数的目标列或表达式。表达式为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISON
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

利用参数 DISTINCT，该函数可在计算最小值之前消除指定表达式中的所有重复值。利用参数 ALL，该函数可保留表达式中的所有重复值以计算最小值。ALL 是默认值。

数据类型

返回与 expression 相同的数据类型。

示例

从所有销售中查找支付的最低价格：

```
select min(pricepaid) from sales;
```

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

从所有销售中查找每张门票的已支付最低价格：

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

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

PERCENTILE_CONT 函数

PERCENTILE_CONT 是一种假定连续分布模型的逆分布函数。该函数具有一个百分比值和一个排序规范，并返回一个在有关排序规范的给定百分比值范围内的内插值。

PERCENTILE_CONT 在对值进行排序后计算值之间的线性内插。通过在聚合组中使用百分比值 (P) 和非 null 行数 (N)，该函数会在根据排序规范对行进行排序后计算行号。根据公式 (RN) 计算此行号 RN = (1+ (P*(N-1))。聚合函数的最终结果通过行号 CRN = CEILING(RN) 和 FRN = FLOOR(RN) 的行中的值之间的线性内插计算。

最终结果将如下所示。

如果 (CRN = FRN = RN)，则结果为 (value of expression from row at RN)

否则，结果将如下所示：

(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN).

PERCENTILE_CONT 是仅计算节点函数。如果查询未引用用户定义的表或 AWS Clean Rooms 系统表，则该函数会返回错误。

语法

```
PERCENTILE_CONT ( percentile )
```

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

参数

percentile

介于 0 和 1 之间的数字常数。计算中将忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定用于排序和计算百分比的数字或日期/时间值。

返回值

返回类型由 WITHIN GROUP 子句中的 ORDER BY 表达式的数据类型决定。下表显示了每种个 ORDER BY 表达式数据类型的返回类型。

输入类型	返回类型
SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

使用说明

如果 ORDER BY 表达式是使用 38 位最大精度定义的 DECIMAL 数据类型，则 PERCENTILE_CONT 可能将返回不准确的结果或错误。如果 PERCENTILE_CONT 函数的返回值超过 38 位，结果将截断以符合规范，这将导致精度降低。如果在插值期间，中间结果超出最大精度，则会发生数值溢出且函数会返回错误。要避免这些情况，建议使用具有较低精度的数据类型或将 ORDER BY 表达式转换为较低精度。

如果语句包括对基于排序的聚合函数 (LISTAGG、PERCENTILE_CONT 或 MEDIAN) 的多个调用，则它们必须全都使用相同的 ORDER BY 值。请注意，MEDIAN 对表达式值应用隐式排序依据。

例如，以下语句将返回错误。

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

An error occurred when executing the SQL command:
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (pricepaid)
from sales group by salesid, pricepai...

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

以下语句将成功运行。

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

示例

以下示例显示，MEDIAN 生成和 PERCENTILE_CONT(0.5) 相同的结果。

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

sellerid | qtysold | percentile_cont | median
---------+---------+-----------------+-------
       1 |       1 |             1.0 |    1.0
       2 |       3 |             3.0 |    3.0
       5 |       2 |             2.0 |    2.0
       9 |       4 |             4.0 |    4.0
      12 |       1 |             1.0 |    1.0
      16 |       1 |             1.0 |    1.0
      19 |       2 |             2.0 |    2.0
      19 |       3 |             3.0 |    3.0
```

```
      22 |       2 |            2.0 |    2.0
      25 |       2 |            2.0 |    2.0
```

STDDEV_SAMP 和 STDDEV_POP 函数

STDDEV_SAMP 和 STDDEV_POP 函数返回一组数值（整数、小数或浮点）的样本标准差和总体标准差。STDDEV_SAMP 函数的结果等于同一组值的样本方差的平方根。

STDDEV_SAMP 和 STDDEV 是同一函数的同义词。

语法

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

表达式必须具有整数、小数或浮点数据类型。无论表达式的数据类型如何，此函数的返回类型都是双精度数。

> **Note**
>
> 使用浮点算法计算标准偏差，其计算结果可能会稍微不准确。

使用说明

当计算包含一个值的表达式的样本标准差（STDDEV 或 STDDEV_SAMP）时，函数的结果为 NULL 而不是 0。

示例

以下查询返回 VENUE 表的 VENUESEATS 列中各值的平均数，后跟同一组值的样本标准差和总体标准差。VENUESEATS 是一个 INTEGER 列。结果的小数位数已减少至 2 位。

```
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)
```

以下查询返回 SALES 表中 COMMISSION 列的样本标准差。COMMISSION 是一个 DECIMAL 列。结果的小数位数已减少至 10 位。

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

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

以下查询将 COMMISSION 列的样本标准差转换为整数。

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

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

以下查询返回 COMMISSION 列的样本标准差和样本方差的平方根。这些计算的结果相同。

```
select
cast(stddev_samp(commission) as dec(18,10)) stddevsamp,
cast(sqrt(var_samp(commission)) as dec(18,10)) sqrtvarsamp
from sales;

stddevsamp   |  sqrtvarsamp
----------------+----------------
130.3912659086 | 130.3912659086
(1 row)
```

SUM 和 SUM DISTINCT 函数

SUM 函数返回输入列或表达式值的和。SUM 函数使用数值并忽略 NULL 值。

SUM DISTINCT 函数可在计算和之前消除指定表达式中的所有重复值。

语法

```
SUM (column)
```

```
SUM (DISTINCT column )
```

Arguments

column

对其执行函数的目标列。列为以下数据类型之一：

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- DOUBLE

数据类型

SUM 函数支持的参数类型有 SMALLINT、INTEGER、BIGINT、DECIMAL 和 DOUBLE。

SUM 函数支持以下返回类型：

- 适用于 BIGINT、SMALLINT 和 INTEGER 参数的 BIGINT
- 适用于浮点参数的 DOUBLE
- 返回与任何其他参数类型的表达式相同的数据类型

带有 DECIMAL 参数的 SUM 函数结果的默认精度为 38。结果的小数位数与参数的小数位数相同。例如，DEC(5,2) 列的 SUM 返回 DEC(38,2) 数据类型。

示例

从 SALES 表中查找所有已付佣金的和：

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

从 SALES 表中查找所有不同佣金的和：

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

VAR_SAMP 和 VAR_POP 函数

VAR_SAMP 和 VAR_POP 函数返回一组数值（整数、小数或浮点）的样本方差和总体方差。VAR_SAMP 函数的结果等于同一组值的样本标准差的平方。

VAR_SAMP 和 VARIANCE 是同一函数的同义词。

语法

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

表达式必须具有整数、小数或浮点数据类型。无论表达式的数据类型如何，此函数的返回类型都是双精度数。

> **Note**
>
> 这些函数的结果可能跨数据仓库集群而异，具体取决于每个案例中集群的配置。

使用说明

当计算包含一个值的表达式的样本方差（VARIANCE 或 VAR_SAMP）时，函数的结果为 NULL 而不是 0。

示例

以下查询返回 LISTING 表中 NUMTICKETS 列的已取整样本方差和总体方差。

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

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

以下查询运行相同的计算但将结果转换为小数值。

```
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)
```

数组函数

本节介绍 AWS Clean Rooms 中支持的 SQL 数组函数。

主题

数组函数

创建 SUPER 数据类型的数组。

语法

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

参数

expr1、expr2

除日期和时间类型之外的任何数据类型的表达式。参数不需要为相同的数据类型。

返回类型

数组函数返回 SUPER 数据类型。

示例

以下示例显示了一个数值数组和一个不同数据类型的数组。

```
--an array of numeric values
select array(1,50,null,100);
      array
------------------
 [1,50,null,100]
(1 row)

--an array of different data types
select array(1,'abc',true,3.14);
        array
-----------------------
 [1,"abc",true,3.14]
(1 row)
```

array_concat 函数

array_concat 函数连接两个数组来创建一个数组，该数组包含第一个数组中的所有元素，然后包含第二个数组中的所有元素。这两个参数必须是有效的数组。

语法

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

参数

super_expr1

指定要连接的两个数组中的第一个数组的值。

super_expr2

指定要连接的两个数组中的第二个数组的值。

返回类型

array_concat 函数返回一个 SUPER 数据值。

示例

以下示例显示了相同类型的两个数组的连接以及两个不同类型的数组的连接。

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

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

array_flatten 函数

将多个数组合并为 SUPER 类型的单个数组。

语法

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

参数

super_expr1、super_expr2

数组形式的有效 SUPER 表达式。

返回类型

array_flatten 函数返回一个 SUPER 数据值。

示例

以下示例显示 array_flatten 函数。

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

get_array_length 函数

返回指定数组的长度。GET_ARRAY_LENGTH 函数在给定对象或数组路径的情况下返回 SUPER 数组的长度。

语法

```
get_array_length( super_expr )
```

参数

super_expr

数组形式的有效 SUPER 表达式。

返回类型

get_array_length 函数返回 BIGINT。

示例

以下示例显示 get_array_length 函数。

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

split_to_array 函数

将分隔符用作可选参数。如果不存在分隔符，则默认值为逗号。

语法

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

参数

string

要拆分的输入字符串。

分隔符

输入字符串将在其上拆分的可选值。默认值为逗号。

返回类型

split_to_array 函数返回一个 SUPER 数据值。

示例

以下示例显示 split_to_array 函数。

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

子数组函数

操作数组以返回输入数组的子集。

语法

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

参数

super_expr

数组形式的有效 SUPER 表达式。

start_position

数组中开始提取的位置，从索引位置 0 开始。负位置从数组的末尾向后计数。

length

要提取的元素的数量（子字符串的长度）。

返回类型

子数组函数返回一个 SUPER 数据值。

示例

以下是子数组函数输出的示例。

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

条件表达式

AWS Clean Rooms 支持以下条件表达式：

主题

- CASE 条件表达式
- COALESCE 表达式
- GREATEST 和 LEAST 函数
- NVL 和 COALESCE 函数
- NVL2 函数
- NULLIF 函数

CASE 条件表达式

CASE 表达式是一种条件表达式，类似于其他语言中发现的 if/then/else 语句。CASE 用于指定存在多个条件时的结果。在 SQL 表达式有效的情况下使用 CASE，例如在 SELECT 命令中。

有两种类型的 CASE 表达式：简单和搜索。

- 在简单 CASE 表达式中，将一个表达式与一个值比较。在找到匹配项时，将应用 THEN 子句中的指定操作。如果未找到匹配项，则应用 ELSE 子句中的操作。
- 在搜索 CASE 表达式中，基于布尔表达式计算每个 CASE，而且 CASE 语句会返回第一个匹配的 CASE。如果在 WHEN 子句中未找到匹配，则返回 ELSE 子句中的操作。

语法

用于匹配条件的简单 CASE 语句：

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

用于计算每个条件的搜索 CASE 语句：

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

参数

expression

一个列名称或任何有效的表达式。

值

与该表达式比较的值，如数字常数或字符串。

result

计算表达式或布尔条件时返回的目标值或表达式。所有结果表达式的数据类型必须可转换为单一输出类型。

condition

计算结果为 true 或 false 的 Boolean 表达式。如果 condition 为 true，则 CASE 表达式的值是符合条件的结果，不处理 CASE 表达式的其余部分。如果 condition 为 false，则计算任何后续的 WHEN 子句。如果没有 WHEN 条件结果为 true，则 CASE 表达式的值是 ELSE 子句的结果。如果没有 ELSE 子句且没有条件为 true，则结果为 null。

示例

使用简单 CASE 表达式在针对 VENUE 表的查询中将 `New York City` 替换为 `Big Apple`。将所有其他城市名称替换为 `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
...
```

使用搜索 CASE 表达式来基于单个门票销售的 PRICEPAID 值分配组编号：

```
select pricepaid,
  case when pricepaid <10000 then 'group 1'
    when pricepaid >10000 then 'group 2'
    else 'group 3'
  end
from sales
order by 1 desc;
```

```
pricepaid |  case
----------+---------
12624     | group 2
10000     | group 3
10000     | group 3
9996      | group 1
9988      | group 1
...
```

COALESCE 表达式

COALESCE 表达式返回列表中的第一个不为 null 的表达式的值。如果所有表达式为 null，则结果为 null。当找到非 null 值时，将不计算该列表中的剩余表达式。

如果您要在首选值缺失或为 null 时返回某些项的备份值，则此类表达式非常有用。例如，查询可能返回三个电话号码（手机、住宅或工作，按该顺序）之一，无论首先在表（非 null）中找到哪一个号码。

Syntax（语法）

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

示例

将 COALESCE 表达式应用于两列。

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

NVL 表达式的默认列名称为 COALESCE。以下查询将返回相同的结果。

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

GREATEST 和 LEAST 函数

从包含任何数量的表达式的列表中返回最大值或最小值。

语法

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

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

参数

expression_list

表达式的逗号分隔的列表，如列名称。这些表达式都必须可转换为常见数据类型。忽略该列表中的 NULL 值。如果所有表达式的计算结果为 NULL，则结果为 NULL。

返回值

从所提供的表达式列表中返回最大值（对于 GREATEST）或最小值（对于 LEAST）。

示例

以下示例按字母顺序返回 `firstname` 或 `lastname` 的最高值。

```
select firstname, lastname, greatest(firstname,lastname) from users
where userid < 10
order by 3;

 firstname | lastname  | greatest
-----------+-----------+-----------
 Alejandro | Rosalez   | Ratliff
 Carlos    | Salazar   | Carlos
 Jane      | Doe       | Doe
 John      | Doe       | Doe
 John      | Stiles    | John
 Shirley   | Rodriguez | Rodriguez
 Terry     | Whitlock  | Terry
 Richard   | Roe       | Richard
 Xiulan    | Wang      | Wang
(9 rows)
```

NVL 和 COALESCE 函数

返回表达式系列中不为 null 的第一个表达式的值。当找到非 null 值时，将不计算该列表中的剩余表达式。

NVL 与 COALESCE 相同。它们是同义词。本主题说明了其语法，并提供这两者的示例。

语法

```
NVL( expression, expression, ... )
```

用于 COALESCE 的语法是相同的：

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

如果所有表达式为 null，则结果为 null。

如果您要在主要值缺失或为 null 时返回次要值，则这些函数非常有用。例如，一个查询可能会返回前三个可用电话号码中的第一个：手机、家庭或工作号码。函数中表达式的顺序决定了计算结果的顺序。

参数

expression

一个要针对 null 状态进行计算的表达式，如列名称。

返回类型

AWS Clean Rooms 根据输入表达式确定返回值的数据类型。如果输入表达式的数据类型不是通用类型，则会返回错误。

示例

如果列表包含整数表达式，则该函数返回一个整数。

```
SELECT COALESCE(NULL, 12, NULL);

coalesce
--------------
12
```

此示例与前面的示例相同（不同之处在于它使用 NVL），返回相同的结果。

```
SELECT NVL(NULL, 12, NULL);

coalesce
```

```
--------------
12
```

以下示例返回字符串类型。

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', NULL);

coalesce
--------------
AWS Clean Rooms
```

以下示例会导致错误，因为表达式列表中的数据类型有变化。在这种情况下，列表中既有字符串类型，也有数字类型。

```
SELECT COALESCE(NULL, 'AWS Clean Rooms', 12);
ERROR: invalid input syntax for integer: "AWS Clean Rooms"
```

NVL2 函数

根据指定表达式的计算结果是 NULL 还是 NOT NULL 返回这两个值之一。

语法

```
NVL2 ( expression, not_null_return_value, null_return_value )
```

参数

expression

一个要针对 null 状态进行计算的表达式，如列名称。

not_null_return_value

在 expression 的计算结果为 NOT NULL 时返回的值。not_null_return_value 值必须具有与 expression 相同的数据类型或可隐式转换为该数据类型。

null_return_value

在 expression 的计算结果为 NULL 时返回的值。null_return_value 值必须具有与 expression 相同的数据类型或可隐式转换为该数据类型。

返回类型

按以下方式确定 NVL2 返回类型：

- 如果 not_null_return_value 或 null_return_value 为 null，则返回非 null 表达式的数据类型。

如果 not_null_return_value 和 null_return_value 都不为 null：

- 如果 not_null_return_value 和 null_return_value 具有相同的数据类型，则返回该数据类型。
- 如果 not_null_return_value 和 null_return_value 具有不同的数字数据类型，则返回最小的可兼容数字数据类型。
- 如果 not_null_return_value 和 null_return_value 具有不同的日期时间数据类型，则返回时间戳数据类型。
- 如果 not_null_return_value 和 null_return_value 具有不同的字符数据类型，则返回 not_null_return_value 的数据类型。
- 如果 not_null_return_value 和 null_return_value 具有混合的数字和非数字数据类型，则返回 not_null_return_value 的数据类型。

> **Important**
>
> 在最后两个示例中（其中返回 not_null_return_value 的数据类型），null_return_value 将隐式转换为该数据类型。如果数据类型不兼容，则该函数将失败。

使用说明

在 NVL2 中，返回内容将具有 not_null_return_value 或 null_return_value 参数的值（不管函数选择哪一个），但将具有 not_null_return_value 的数据类型。

例如，假定 column1 为 NULL，则以下查询将返回相同的值。但是，DECODE 返回值数据类型将为 INTEGER，NVL2 返回值数据类型将为 VARCHAR。

```
select decode(column1, null, 1234, '2345');
select nvl2(column1, '2345', 1234);
```

示例

以下示例修改一些示例数据，然后计算两个字段以为用户提供相应的联系人信息：

```
update users set email = null where firstname = 'Aphrodite' and lastname = 'Acevedo';

select (firstname + ' ' + lastname) as name,
nvl2(email, email, phone) AS contact_info
from users
where state = 'WA'
and lastname  like 'A%'
order by lastname, firstname;

name        contact_info
--------------------+--------------------------------------------
Aphrodite Acevedo (555) 555-0100
Caldwell Acevedo  Nunc.sollicitudin@example.ca
Quinn Adams     vel@example.com
Kamal Aguilar   quis@example.com
Samson Alexander  hendrerit.neque@example.com
Hall Alford     ac.mattis@example.com
Lane Allen      et.netus@example.com
Xander Allison    ac.facilisis.facilisis@example.com
Amaya Alvarado    dui.nec.tempus@example.com
Vera Alvarez    at.arcu.Vestibulum@example.com
Yetta Anthony   enim.sit@example.com
Violet Arnold   ad.litora@example.comm
August Ashley   consectetuer.euismod@example.com
Karyn Austin    ipsum.primis.in@example.com
Lucas Ayers     at@example.com
```

NULLIF 函数

语法

NULLIF 表达式比较两个参数并在两个参数相等时返回 null。如果两个参数不相等，则返回第一个参数。此表达式为 NVL 或 COALESCE 表达式的反向表达式。

```
NULLIF ( expression1, expression2 )
```

参数

expression1，expression2

所比较的目标列或表达式。返回类型与第一个表达式的类型相同。NULLIF 结果的默认列名称为第一个表达式的列名称。

示例

在以下示例中，查询返回字符串 first，因为参数不相等。

```
SELECT NULLIF('first', 'second');

case
-------
first
```

在以下示例中，查询返回字符串 NULL，因为字符串文本参数相等。

```
SELECT NULLIF('first', 'first');

case
-------
NULL
```

在以下示例中，查询返回 1，因为整数参数不相等。

```
SELECT NULLIF(1, 2);

case
-------
1
```

在以下示例中，查询返回 NULL，因为整数参数相等。

```
SELECT NULLIF(1, 1);

case
-------
NULL
```

在以下示例中，查询在 LISTID 和 SALESID 值匹配时返回 null：

```
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)
```

数据类型格式设置函数

使用数据类型格式设置函数，您可以将值从一种数据类型转换为另一种数据类型。对于每个函数，第一个参数始终是要格式化的值，第二个参数包含新格式的模板。 AWS Clean Rooms 支持多种数据类型格式化函数。

主题

- CAST 函数
- CONVERT 函数
- TO_CHAR
- TO_DATE 函数
- TO_NUMBER
- 日期时间格式字符串
- 数字格式字符串
- 数字数据的 Teradata 类格式字符

CAST 函数

CAST 函数将一种数据类型转换为另一种兼容的数据类型。例如，您可以将字符串转换为日期，或将数值类型转换为字符串。CAST 执行运行时转换，这意味着转换不会更改源表中值的数据类型。仅在查询上下文中对其进行更改。

CAST 函数与 the section called “CONVERT” 非常相似，它们都将一种数据类型转换为另一种数据类型，但它们的调用方式不同。

某些数据类型需要使用 CAST 或 CONVERT 函数显式转换为其他数据类型。其他数据类型可进行隐式转换（作为另一个命令的一部分），无需使用 CAST 或 CONVERT。请参阅 类型兼容性和转换。

语法

使用以下两个等效的语法形式，将表达式从一种数据类型强制转换为另一种数据类型。

```
CAST ( expression AS type )
expression :: type
```

参数

expression

计算结果为一个或多个值的表达式，如列名称或文本。转换 null 值将返回 null。表达式不能包含空字符串或空字符串。

类型

除了 VARBYTE 数据类型、BINARY 和 BINARY VARIANCY 数据类型之外，其他数据类型均受支持。

返回类型

CAST 返回 type 参数指定的数据类型。

> **Note**
>
> AWS Clean Rooms 如果您尝试执行有问题的转换（例如会丢失精度的十进制转换），则返回错误，如下所示：
>
> ```
> select 123.456::decimal(2,1);
> ```
>
> 或导致溢出的 INTEGER 转换：
>
> ```
> select 12345678::smallint;
> ```

示例

以下两个查询是等效的。它们都将小数值转换为整数：

```
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)
```

以下内容会产生类似的结果。它不需要示例数据即可运行：

```
select cast(162.00 as integer) as pricepaid;

pricepaid
-----------
162
(1 row)
```

在此示例中，时间戳列中的值将强制转换为日期，这会导致从每个结果中删除时间：

```
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)
```

如果您没有像前一个示例中所示的那样使用 CAST，则结果将包括时间：2008-02-18 02:36:48。

以下查询将可变字符数据强制转换为日期。它不需要示例数据即可运行。

```
select cast('2008-02-18 02:36:48' as date) as mysaletime;

mysaletime
--------------------
2008-02-18
(1 row)
```

在此示例中，日期列中的值将强制转换为时间戳：

```
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)
```

在前面的示例中，您可以通过使用来获得对输出格式的额外控制TO_CHAR。

在此示例中，整数将强制转换为字符串：

```
select cast(2008 as char(4));

bpchar
--------
2008
```

在此示例中，DECIMAL(6,3) 值将强制转换为 DECIMAL(4,1) 值：

```
select cast(109.652 as decimal(4,1));

numeric
---------
109.7
```

此示例显示了一个更复杂的表达式。SALES 表中的 PRICEPAID 列（DECIMAL(8,2) 列）将转换为 DECIMAL(38,2) 列，并且值将乘以 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)
```

CONVERT 函数

与一样CAST 函数，CONVERT 函数将一种数据类型转换为另一种兼容的数据类型。例如，您可以将字符串转换为日期，或将数值类型转换为字符串。CONVERT 执行运行时转换，这意味着转换不会更改源表中值的数据类型。仅在查询上下文中对其进行更改。

某些数据类型需要使用 CONVERT 函数显式转换为其他数据类型。其他数据类型可进行隐式转换（作为另一个命令的一部分），无需使用 CAST 或 CONVERT。请参阅 类型兼容性和转换。

语法

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

参数

type

除了 VARBYTE 数据类型、BINARY 和 BINARY VARIANCY 数据类型之外，其他数据类型均受支持。

expression

计算结果为一个或多个值的表达式，如列名称或文本。转换 null 值将返回 null。表达式不能包含空字符串或空字符串。

返回类型

CONVERT 返回 type 参数指定的数据类型。

> **Note**
>
> AWS Clean Rooms 如果您尝试执行有问题的转换（例如会丢失精度的十进制转换），则返回错误，如下所示：
>
> ```
> SELECT CONVERT(decimal(2,1), 123.456);
> ```
>
> 或导致溢出的 INTEGER 转换：
>
> ```
> SELECT CONVERT(smallint, 12345678);
> ```

示例

以下查询使用 CONVERT 函数将一些小数列转换为整数

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

此示例将整数转换为字符串。

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

在此示例中，当前日期和时间转换为可变字符数据类型：

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

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

此示例将 saletime 列仅转换只包括时间，删除每行中的日期。

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

以下示例将可变字符数据转换为日期时间对象。

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

TO_CHAR

TO_CHAR 将时间戳或数值表达式转换为字符串数据格式。

语法

```
TO_CHAR (timestamp_expression | numeric_expression , 'format')
```

参数

timestamp_expression

一个表达式，用于生成 TIMESTAMP 或 TIMESTAMPTZ 类型值或可隐式强制转换为时间戳的值。

numeric_expression

一个表达式，用于生成数字数据类型值或可隐式强制转换为数字类型的值。有关更多信息，请参阅数字类型。TO_CHAR 在数字串左侧插入空格。

> **Note**
>
> TO_CHAR 不支持 128 位的十进制值。

format

新值的格式。有关有效格式，请参阅日期时间格式字符串和 数字格式字符串。

返回类型

VARCHAR

示例

以下示例将时间戳转换为一个具有日期和时间的值，格式为月份名称填充为九个字符、星期几的名称和当月的日期编号。

```
select to_char(timestamp '2009-12-31 23:15:59', 'MONTH-DY-DD-YYYY HH12:MIPM');
to_char
-------------------------
DECEMBER -THU-31-2009 11:15PM
```

以下示例将时间戳转换为具有这一年中日期编号的值。

```
select to_char(timestamp '2009-12-31 23:15:59', 'DDD');

to_char
-------------------------
365
```

以下示例将时间戳转换为这一周的 ISO 日期编号。

```
select to_char(timestamp '2022-05-16 23:15:59', 'ID');

to_char
-------------------------
1
```

以下示例从日期中提取月份名称。

```
select to_char(date '2009-12-31', 'MONTH');

to_char
-------------------------
DECEMBER
```

以下示例将 EVENT 表中的每个 STARTTIME 值转换为由小时、分钟和秒组成的字符串。

```
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)
```

以下示例将整个时间戳值转换为不同的格式。

```
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)
```

以下示例将时间戳文本转换为字符串。

```
select to_char(timestamp '2009-12-31 23:15:59','HH24:MI:SS');
to_char
----------
23:15:59
(1 row)
```

以下示例将一个数字转换为末尾带负号的字符串。

```
select to_char(-125.8, '999D99S');
to_char
---------
125.80-
(1 row)
```

以下示例将一个数字转换为带货币符号的字符串。

```
select to_char(-125.88, '$S999D99');
to_char
---------
$-125.88
(1 row)
```

以下示例将一个数字转换为用尖括号将负数括起来的字符串。

```
select to_char(-125.88, '$999D99PR');
to_char
---------
$<125.88>
(1 row)
```

以下示例将一个数字转换为罗马数字字符串。

```
select to_char(125, 'RN');
to_char
---------
CXXV
(1 row)
```

以下示例显示一周中的某天。

```
SELECT to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS');
               to_char
-----------------------
Wednesday, 31 09:34:26
```

以下示例显示数字的序数后缀。

```
SELECT to_char(482, '999th');
               to_char
-----------------------
 482nd
```

以下示例将销售表中支付的价格减去佣金。然后将差值四舍五入并转换为罗马数字，如to_char列中所示：

```
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)
```

以下示例将货币符号添加到to_char列中显示的差值中：

```
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)
```

以下示例列出了完成每次销售的世纪。

```
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)
```

以下示例将 EVENT 表中的每个 STARTTIME 值转换为由小时、分钟、秒和时区组成的字符串：

```
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)
```

以下示例显示了秒、毫秒和微秒的格式设置。

```
select sysdate,
to_char(sysdate, 'HH24:MI:SS') as seconds,
to_char(sysdate, 'HH24:MI:SS.MS') as milliseconds,
to_char(sysdate, 'HH24:MI:SS:US') as microseconds;

timestamp           | seconds  | milliseconds | microseconds
--------------------+----------+--------------+----------------
2015-04-10 18:45:09 | 18:45:09 | 18:45:09.325 | 18:45:09:325143
```

TO_DATE 函数

TO_DATE 会将以字符串形式表示的日期转换为 DATE 数据类型。

语法

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

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

参数

string

要转换的字符串。

format

一个字符串文本，用于定义其日期部分中的输入字符串格式。有关有效日期、月份和年份格式的列表，请参阅日期时间格式字符串。

is_strict

一个可选的布尔值，它指定在输入日期值超出范围时是否返回错误。当 is_strict 被设置为 TRUE 时，如果存在超出范围的值，则返回错误。当 is_strict 被设置为 FALSE（默认值）时，则接受溢出值。

返回类型

TO_DATE 将根据 format 值返回 DATE。

如果转换为格式 失败，则返回错误。

示例

以下 SQL 语句将日期 `02 Oct 2001` 转换为日期数据类型。

```
select to_date('02 Oct 2001', 'DD Mon YYYY');

to_date
------------
```

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

以下 SQL 语句将字符串 20010631 转换为日期。

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

结果是 2001 年 7 月 1 日，因为六月份只有 30 天。

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

以下 SQL 语句将字符串 20010631 转换为日期：

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

结果产生错误，因为 6 月只有 30 天。

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

TO_NUMBER

TO_NUMBER 将字符串转换为数字（小数）值。

语法

```
to_number(string, format)
```

参数

string

要转换的字符串。格式必须是文本值。

format

第二个参数是指示应如何分析字符串以创建数字值的格式字符串。例如，格式 '99D999' 指定要转换的字符串包含五位数，其中小数点在第三位。例如，to_number('12.345','99D999') 将 12.345 作为数字值返回。有关有效格式的列表，请参阅 数字格式字符串。

返回类型

TO_NUMBER 返回 DECIMAL 数。

如果转换为格式 失败，则返回错误。

示例

以下示例将字符串 12,454.8- 转换为数字：

```
select to_number('12,454.8-', '99G999D9S');

to_number
-----------
-12454.8
```

以下示例将字符串 $ 12,454.88 转换为数字：

```
select to_number('$ 12,454.88', 'L 99G999D99');

to_number
-----------
12454.88
```

以下示例将字符串 $ 2,012,454.88 转换为数字：

```
select to_number('$ 2,012,454.88', 'L 9,999,999.99');

to_number
-----------
2012454.88
```

日期时间格式字符串

以下日期时间格式字符串适用于 TO_CHAR 之类的函数。这些字符串可包含日期时间分隔符（如 '-'、'/' 或 ':'）以及下面的日期部分和时间部分。

有关将日期格式化为字符串的示例，请参阅TO_CHAR。

日期部分或时间部分	意义
BC 或 B.C.、AD 或 A.D.、b.c. 或 bc、ad 或 a.d。	大写和小写的纪元指示符
CC	2 位世纪数字
YYYY、YYY、YY、Y	4 位、3 位、2 位、1 位年数字
Y,YYY	带逗号的 4 位年数
IYYY、IYY、IY、I	4 位、3 位、2 位、1 位国际标准化组织 (ISO) 年数
Q	季度数（1 至 4）
MONTH、Month、month	月名称（大写、大小写混合、小写，空格填补为 9 个字符）
MON、Mon、mon	缩写的月份名称（大写、大小写混合、小写，空格填补至 3 个字符）
MM	月数 (01-12)
RM、rm	使用罗马数字的月数（I–XII，I 代表 1 月，大小写均可）
W	一个月中的周（1–5，第一周从当月的第一天开始。）
WW	一年的周数（1–53，第一周从一年的第一天开始。）
IW	一年的 ISO 周数（新的一年的第一个星期四算在第 1 周。）
DAY、Day、day	日名称（大写、大小写混合、小写，空格填补为 9 个字符）

日期部分或时间部分	意义
DY、Dy、dy	缩写的日期名称（大写、大小写混合、小写，空格填补为 3 个字符）
DDD	一年中的日 (001–366)
IDDD	ISO 8601 按周编号的年中的日期（001-371；每年的第一天是 ISO 第一周的周一）
DD	用数字表示的一个月中的日 (01–31)
D	一周中的日（1–7；星期日为 1） **Note** D 日期部分的运行方式与用于日期时间功能的 DATE_PART 和 EXTRACT 的一周中的日 (DOW) 日期部分不同。DOW 基于 0–6 的整数，其中星期日是 0。有关更多信息，请参阅 日期或时间戳函数的日期部分。
ID	ISO 8601 星期几，周一 (1) 至周日 (7)
J	儒略日（公元前 4712 年 1 月 1 日起的日期）
HH24	小时（24 小时制，00–23）
HH 或 HH12	小时（12 小时制，01–12）
MI	分钟数 (00—59)
SS	秒数 (00—59)
MS	毫秒 (.000)
US	微秒 (.000000)

日期部分或时间部分	意义
AM 或 PM；A.M. 或 P.M.；a.m. 或 p.m.；am 或 pm	大写和小写的子午线指示符（适用于 12 小时制）
TZ、tz	大小写时区缩写；仅对 TIMESTAMPTZ 有效。
OF	从 UTC 的偏移；仅对 TIMESTAMPTZ 有效。

> **Note**
>
> 您必须用单引号将 datetime 分隔符（如 '-'、'/' 或 ':'）括起，但您必须用双引号将上表中列出的 "dateparts" 和 "timeparts" 括起。

数字格式字符串

以下数字格式字符串适用于 TO_NUMBER 和 TO_CHAR 之类的函数。

- 有关将字符串格式化为数字的示例，请参阅TO_NUMBER。
- 有关将数字格式化为字符串的示例，请参阅TO_CHAR。

格式	描述
9	具有指定位数的数字值。
0	包含前导零的数字值。
.（句点），D	小数点。
,（逗号）	千位分隔符。
CC	世纪代码。例如，21 世纪从 2001-01-01 开始（仅受 TO_CHAR 支持）。
FM	填充模式。隐藏填补空格和零。
PR	尖括号中的负值。

格式	描述
S	锚定数字的符号。
L	指定位置中的货币符号。
G	组分隔符。
MI	小于 0 的数字的指定位置中的减号。
PL	大于 0 的数字的指定位置中的加号。
SG	指定位置中的加号或减号。
RN	1 到 3999 之间的罗马数字（仅受 TO_CHAR 支持）。
TH 或 th	序号后缀。不会转换小于零的分数或值。

数字数据的 Teradata 类格式字符

本主题介绍 TEXT_TO_INT_ALT 和 TEXT_TO_NUMERIC_ALT 函数如何解释输入 expression 字符串中的字符。在下表中，您还可以找到您可以在 format 短语中指定的字符列表。此外，您还可以找到有关Teradata样式格式和格式选项之间差异 AWS Clean Rooms 的描述。

格式	描述
G	不支持在输入 expression 字符串中作为组分隔符。您不能在 format 短语中指定此字符。
D	基数符号。您可以在 format 短语中指定此字符。此字符等效于. (句点)。 基数符号不能出现在包含以下任意字符的 format 短语中： • .（句点） • S（大写“s”）

格式	描述
	• V（大写“v”）
/ , : %	插入字符/（正斜杠）、逗号 (,)、:（冒号）和 %（百分号）。 您不能在 format 短语中包含这些字符。 AWS Clean Rooms 忽略输入表达式字符串中的这些字符。
.	表示基数字符的句点，即小数点。 此字符不能出现在包含以下任意字符的 format 短语中： • D（大写“d”） • S（大写“s”） • V（大写“v”）
B	不能将空格字符 (B) 包含在 format 短语中。在输入 expression 字符串中，忽略前导空格和尾随空格，并且不允许数字之间存在空格。
+ -	您不能在 format 短语中包含加号 (+) 或减号 (-)。但是，如果加号 (+) 和减号 (-) 出现在输入 expression 字符串，则将它们隐式解析为数值的一部分。
V	小数点位置指示器。 此字符不能出现在包含以下任意字符的 format 短语中： • D（大写“d”） • .（句点）

格式	描述
Z	零抑制的十进制数字。 AWS Clean Rooms 修剪前导零。Z 字符不能跟随字符 9。如果分数部分包含字符 9，则 Z 字符必须位于基数字符的左侧。
9	十进制数字。
CHAR(n)	对于此格式，您可以指定以下各项： • CHAR 由 Z 或 9 个字符组成。 AWS Clean Rooms 不支持 CHAR 值中的 +（加号）或-（减号）。 • n 是整数常量 I 或 F。对于 I，它是显示数字或整数数据的整数部分所需的字符数。对于 F，它是显示数字数据的小数部分所需的字符数。
-	连字符 (-) 字符。 您不能在 format 短语中包含此字符。 AWS Clean Rooms 忽略输入表达式字符串中的此字符。

格式	描述
S	带符号的分区十进制。S 字符必须跟随 format 短语中的最后一位十进制数字。输入 expression 字符串的最后一个字符和相应的数字转换列在带符号的分区十进制、Teradata 类数字数据格式的数据格式字符 中。 S 字符不能出现在包含以下任意字符的 format 短语中： • + (加号) • . (句点) • D (大写“d”) • Z (大写“z”) • F (大写“f”) • E (大写“e”)
E	指数表示法。输入 expression 字符串可以包含指数字符。您不能在 format 短语中将 E 指定为指数字符。
FN9	中不支持 AWS Clean Rooms。
FNE	中不支持 AWS Clean Rooms。
$、USD、US Dollars	美元符号 ($)、ISO 货币符号 (USD) 和货币名称 US Dollars。 ISO 货币符号 USD 和货币名称 US Dollars 区分大小写。 AWS Clean Rooms 仅支持美元货币。输入 expression 字符串可以在 USD 货币符号和数值之间包含空格，例如“$ 123E2”或“123E2 $”。
L	货币符号。此货币符号字符只能在 format 短语中出现一次。您不能指定重复的货币符号字符。

格式	描述
C	ISO 货币符号。此货币符号字符只能在 format 短语中出现一次。您不能指定重复的货币符号字符。
否	完整货币名称。此货币符号字符只能在 format 短语中出现一次。您不能指定重复的货币符号字符。
O	双币符号。您不能在 format 短语中指定此字符。
U	双 ISO 货币符号。您不能在 format 短语中指定此字符。
A	完整的双币名称。您不能在 format 短语中指定此字符。

带符号的分区十进制、Teradata 类数字数据格式的数据格式字符

您可以在 TEXT_TO_INT_ALT 和 TEXT_TO_NUMERIC_ALT 函数的 format 短语中将以下字符用于带符号的分区十进制值。

输入字符串的最后一个字符	数字转换
{ 或 0	n … 0
A 或 1	n … 1
B 或 2	n … 2
C 或 3	n … 3
D 或 4	n … 4
E 或 5	n … 5
F 或 6	n … 6

输入字符串的最后一个字符	数字转换
G 或 7	n … 7
H 或 8	n … 8
I 或 9	n … 9
}	-n … 0
J	-n … 1
K	-n … 2
L	-n … 3
M	-n … 4
否	-n … 5
O	-n … 6
P	-n … 7
Q	-n … 8
R	-n … 9

日期和时间函数

AWS Clean Rooms 支持以下日期和时间函数：

主题

- DATEADD 函数
- DATEDIFF 函数
- DATE_PART 函数
- DATE_TRUNC 函数
- EXTRACT 函数
- GETDATE 函数
- SYSDATE 函数
- TIMEOFDAY 函数
- TO_TIMESTAMP 函数
- 日期或时间戳函数的日期部分

日期和时间函数摘要

下表概述了 AWS Clean Rooms 中使用的日期和时间函数。

函数	语法	返回值
+ (串联) 运算符 将日期连接到 + 符号任一侧的时间，并返回 TIMESTAMP 或 TIMESTAMPTZ。	date + time	TIMESTAMP 或者 TIMESTAMPZ
ADD_MONTHS 将指定的月数添加到日期或时间戳。	ADD_MONTHS ({date|timestamp}, integer)	TIMESTAMP
CURRENT_DATE 函数 返回当前事务开始时的当前会话时区 (预设情况下为 UTC) 中的日期。	CURRENT_DATE	DATE
DATEADD 按指定的时间间隔递增日期或时间。	DATEADD (datepart, interval, {date|time|timetz|timestamp})	TIMESTAMP 、TIME 或 TIMETZ

函数	语法	返回值
DATEDIFF 返回给定日期部分（如一天或月）的两个日期或时间之间的差值。	DATEDIFF (datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp})	BIGINT
DATE_PART 从日期或时间中提取日期部分值。	DATE_PART (datepart, {date|timestamp})	DOUBLE
DATE_TRUNC 基于日期部分截断时间戳。	DATE_TRUNC ('datepart', timestamp)	TIMESTAMP
EXTRACT 从 timestamp、timestamptz、time 或 timetz 中提取日期或时间部分。	EXTRACT (datepart FROM source)	INTEGER or DOUBLE
GETDATE 函数 返回当前会话时区（预设情况下为 UTC）中的当前日期和时间。括号为必填项。	GETDATE()	TIMESTAMP
SYSDATE 返回当前事务开始的日期和时间 (UTC)。	SYSDATE	TIMESTAMP
TIMEOFDAY 以字符串值形式返回当前会话时区（预设情况下为 UTC）中的当前工作日、日期和时间。	TIMEOFDAY()	VARCHAR

函数	语法	返回值
TO_TIMESTAMP 返回指定时间戳和时区格式的一个带有时区的时间戳。	TO_TIMESTAMP ('timestamp', 'format')	TIMESTAMP TZ

> **Note**
>
> 在经过时间计算中不考虑闰秒。

事务中的日期和时间函数

当您在事务块（BEGIN … END）中运行以下函数时，该函数将返回当前事务的开始日期或时间，而不是当前语句的开始时间。

- SYSDATE
- TIMESTAMP
- CURRENT_DATE

以下函数始终返回当前语句的开始日期或时间，即使它们在事务数据块中也是如此。

- GETDATE
- TIMEOFDAY

+（串联）运算符

连接数值文本、字符串文本和/或日期时间和时间间隔文本。它们位于 + 符号的两侧，并根据 + 符号两侧的输入返回不同的类型。

语法

```
numeric + string
```

```
date + time
```

```
date + timetz
```

参数的顺序可以反转。

参数

####

表示数字的文本或常量可以是整数或浮点。

#####

字符串、字符字符串或字符常量

date

DATE 列或隐式转换为 DATE 的表达式。

time

TIME 列或隐式转换为 TIME 的表达式。

timetz

TIMETZ 列或隐式转换为 TIMETZ 的表达式。

示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

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

ADD_MONTHS 函数

ADD_MONTHS 会将指定的月数添加到日期或时间戳值或表达式中。DATEADD 函数提供了类似的功能。

语法

```
ADD_MONTHS( {date | timestamp}, integer)
```

参数

date | timestamp

日期或时间戳列，或隐式转换为日期或时间戳的表达式。如果日期是该月的最后一天，或者如果产生的月份较短，则函数在结果中返回该月的最后一天。对于其他日期，结果包含与日期表达式相同的日期编号。

integer

正整数或负整数。使用负数从日期中减去月份。

返回类型

TIMESTAMP

示例

以下查询使用 TRUNC 函数内的 ADD_MONTHS 函数。TRUNC 函数从 ADD_MONTHS 的结果中删除一天中的时间。ADD_MONTHS 函数会为 CALDATE 列中的每个值添加 12 个月。

```
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)
```

以下示例演示 ADD_MONTHS 函数在具有不同天数的月份的日期上运行时的行为。

```
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)
```

CONVERT_TIMEZONE 函数

CONVERT_TIMEZONE 将一个时区的时间戳转换为另一个时区的时间戳。该函数会自动根据夏令时调整。

语法

```
CONVERT_TIMEZONE ( ['source_timezone',] 'target_timezone', 'timestamp')
```

参数

source_timezone

（可选）当前时间戳的时区。默认值为 UTC。

target_timezone

新时间戳的时区。

timestamp

时间戳列或隐式转换为时间戳的表达式。

返回类型

TIMESTAMP

示例

以下示例将时间戳值从原定设置的 UTC 时区转换为 PST。

```
select convert_timezone('PST', '2008-08-21 07:23:54');
```

```
 convert_timezone
-----------------------
2008-08-20 23:23:54
```

以下示例将 LISTTIME 列中的时间戳值从默认 UTC 时区转换为 PST。尽管时间戳在夏令时间段内，但它会转换为标准时间，因为目标时区被指定为缩写 (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
```

以下示例将 LISTTIME 列中的时间戳从默认 UTC 时区转换为美国/太平洋时区。目标时区使用时区名称，时间戳位于夏令时间段内，因此函数返回夏令时。

```
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
```

以下示例将时间戳字符串从 EST 转换为 PST：

```
select convert_timezone('EST', 'PST', '20080305 12:25:29');

 convert_timezone
-------------------
2008-03-05 09:25:29
```

以下示例将时间戳转换为美国东部标准时间，因为目标时区使用时区名称 (America/New_York)，并且时间戳在标准时间段内。

```
select convert_timezone('America/New_York', '2013-02-01 08:00:00');

 convert_timezone
---------------------
2013-02-01 03:00:00
```

```
(1 row)
```

以下示例将时间戳转换为美国东部夏令时，因为目标时区使用时区名称 (America/New_York)，并且时间戳在夏令时时间段内。

```
select convert_timezone('America/New_York', '2013-06-01 08:00:00');

 convert_timezone
---------------------
2013-06-01 04:00:00
(1 row)
```

以下示例演示了偏移的用法。

```
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)
```

CURRENT_DATE 函数

CURRENT_DATE 以默认格式 YYYY-MM-DD 返回当前会话时区（预设情况下为 UTC）中的日期。

> **Note**
>
> CURRENT_DATE 返回当前事务的开始日期，而不是当前语句的开始日期。考虑这样的场景，即您在 2008 年 1 月 10 日 23:59 开始一个包含多个语句的事务，而包含 CURRENT_DATE 的语句在 2008 年 2 月 10 日 00:00 运行。CURRENT_DATE 返回 10/01/08，而不是 10/02/08。

语法

```
CURRENT_DATE
```

返回类型

DATE

示例

以下示例返回当前日期（在函数运行 AWS 区域 的地方）。

```
select current_date;

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

DATEADD 函数

按指定的时间间隔递增 DATE、TIME、TIMETZ 或 TIMESTAMP 值。

语法

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

参数

datepart

函数操作的日期部分（例如年、月、日或小时）。有关更多信息，请参阅日期或时间戳函数的日期部分。

interval

指定要添加到目标表达式的时间间隔（例如天数）的整数。负整数减去时间间隔。

date|time|timetz|timestamp

DATE、TIME、TIMETZ 或 TIMESTAMP 列或隐式转换为 DATE、TIME、TIMETZ 或 TIMESTAMP 的表达式。DATE、TIME、TIMETZ 或 TIMESTAMP 表达式必须包含指定的日期部分。

返回类型

TIMESTAMP 或 TIME 或 TIMETZ，具体取决于输入数据类型。

具有 DATE 列的示例

以下示例为 DATE 表中存在的 11 月中的每个日期添加 30 天。

```
select dateadd(day,30,caldate) as novplus30
from date
where month='NOV'
order by dateid;

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

以下示例将 18 个月添加到文本日期值。

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

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

DATEADD 函数的默认列名称为 DATE_ADD。日期值的默认时间戳为 00:00:00。

以下示例向未指定时间戳的日期值添加 30 分钟。

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

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

您可以用全名或缩写来命名日期部分。在此情况下，m 代表几分钟，而不是几个月。

具有 TIME 列的示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

以下示例为 TIME_TEST 表中的每个 TIME_VAL 添加 5 分钟。

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

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

以下示例为文本时间值添加 8 小时。

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

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

以下示例显示时间何时超过 24:00:00 或低于 00:00:00。

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

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

具有 TIMETZ 列的示例

这些示例中的输出值以 UTC 为默认时区。

下面的示例表 TIMETZ_TEST 具有一个列 TIMETZ_VAL（类型 TIMETZ），其中插入了三个值。

```
select timetz_val from timetz_test;
```

```
timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

下面的示例为 TIMETZ_TEST 表中的每个 TIMETZ_VAL 添加 5 分钟。

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

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

以下示例将 2 小时添加到一个文本 timetz 值。

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

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

具有 TIMESTAMP 列的示例

这些示例中的输出值以 UTC 为默认时区。

下面的示例表 TIMESTAMP_TEST 具有一个列 TIMESTAMP_VAL（类型为 TIMESTAMP），其中插入了三个值。

```
SELECT timestamp_val FROM timestamp_test;

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

以下示例仅向 TIMESTAMP_TEST 中 2000 年之前的 TIMESTAMP_VAL 值增加 20 年。

```
SELECT dateadd(year,20,timestamp_val)
```

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

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

以下示例向不带秒指示器的文本时间戳值增加 5 秒。

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

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

使用说明

DATEADD(month, ...) 和 ADD_MONTHS 函数以不同的方式处理位于月末的日期：

- ADD_MONTHS：如果添加到的日期是该月的最后一天，则无论该月有多少天，结果始终是结果月份的最后一天。例如，4 月 30 日 + 1 个月是 5 月 31 日。

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

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

- DATEADD：如果添加到的日期中的天数少于结果月份，则结果是结果月份的对应日期，而不是该月的最后一天。例如，4 月 30 日 + 1 个月是 5 月 30 日。

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

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

当使用 dateadd(month, 12,…) 或 dateadd(year, 1, …) 时，DATEADD 函数以不同方式处理闰年日期 02-29。

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

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

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

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

DATEDIFF 函数

DATEDIFF 返回两个日期或时间表达式的日期部分之间的差异。

语法

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

参数

datepart

该函数运行所依据的日期或时间值的特定部分（年、月或日、小时、分钟、秒、毫秒或微秒）。有关更多信息，请参阅日期或时间戳函数的日期部分。

具体而言，DATEDIFF 确定在两个表达式之间交叉的日期部分边界的数量。例如，假设您计算两个日期 12-31-2008 与 01-01-2009 之间的年份差异。在这种情况下，函数返回 1 年，尽管这些日期仅相隔一天。如果您发现两个时间戳 01-01-2009 8:30:00 与 01-01-2009 10:00:00 之间存在小时差，则结果为 2 小时。如果您发现两个时间戳 8:30:00 与 10:00:00 之间存在小时差，则结果为 2 小时。

date|time|timetz|timestamp

DATE、TIME、TIMETZ 或 TIMESTAMP 列或隐式转换为 DATE、TIME、TIMETZ 或 TIMESTAMP 的表达式。表达式必须同时包含指定的日期或时间部分。如果第二个日期或时间晚于第一个日期或时间，则结果为正值。如果第二个日期或时间早于第一个日期或时间，则结果为负值。

返回类型

BIGINT

具有 DATE 列的示例

以下示例查找两个文本日期值之间的差异（以周数为单位）。

```
select datediff(week,'2009-01-01','2009-12-31') as numweeks;

numweeks
----------
52
(1 row)
```

以下示例查找两个文本日期值之间的差异，以小时为单位。如果您没有为日期提供时间值，则默认为 00:00:00。

```
select datediff(hour, '2023-01-01', '2023-01-03 05:04:03');

date_diff
----------
53
(1 row)
```

以下示例查找两个文本 TIMESTAMETZ 值之间的差异，以天为单位。

```
Select datediff(days, 'Jun 1,2008  09:59:59 EST', 'Jul 4,2008  09:59:59 EST')

date_diff
----------
33
```

以下示例查找表中同一行的两个日期之间的差异，以天为单位。

```
select * from date_table;

start_date |   end_date
-----------+-----------
2009-01-01 | 2009-03-23
2023-01-04 | 2024-05-04
```

```
(2 rows)

select datediff(day, start_date, end_date) as duration from date_table;

duration
---------
      81
     486
(2 rows)
```

以下示例查找过去日期和今天日期中的文本值之间的差异（以季度数为单位）。此示例假定当前日期为 2008 年 6 月 5 日。您可以可以用全名或缩写来命名日期部分。DATEDIFF 函数的默认列名称为 DATE_DIFF。

```
select datediff(qtr, '1998-07-01', current_date);

date_diff
-----------
40
(1 row)
```

以下示例将 SALES 和 LISTING 表联接，以计算它们列出后多少天清单 1000 到 1005 的所有票证被售出。这些清单的最长销售等待时间为 15 天，最短等待时间不到一天（0 天）。

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

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

此示例计算卖家等待所有票证销售的平均小时数。

```
select avg(datediff(hours, listtime, saletime)) as avgwait
from sales, listing
where sales.listid = listing.listid;

avgwait
---------
465
(1 row)
```

具有 TIME 列的示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

以下示例查找 TIME_VAL 列与时间文本之间的小时数差异。

```
select datediff(hour, time_val, time '15:24:45') from time_test;

 date_diff
-----------
        -5
        15
        15
```

以下示例查找两个文本时间值之间的分钟数差异。

```
select datediff(minute, time '20:00:00', time '21:00:00') as nummins;

nummins
----------
60
```

具有 TIMETZ 列的示例

下面的示例表 TIMETZ_TEST 具有一个列 TIMETZ_VAL（类型 TIMETZ），其中插入了三个值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

以下示例查找 TIMETZ 文本与 timetz_val 之间的小时数差异。

```
select datediff(hours, timetz '20:00:00 PST', timetz_val) as numhours from timetz_test;

numhours
----------
0
-4
1
```

以下示例查找两个文本 TIMETZ 值之间的小时数差异。

```
select datediff(hours, timetz '20:00:00 PST', timetz '00:58:00 EST') as numhours;

numhours
----------
1
```

DATE_PART 函数

DATE_PART 从表达式中提取日期部分值。DATE_PART 是 PGDATE_PART 函数的同义词。

语法

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

参数

datepart

函数所操作的日期值的特定部分（例如年、月或日）的标识符文本或字符串。有关更多信息，请参阅日期或时间戳函数的日期部分。

{date|timestamp}

日期列、时间戳列或隐式转换为日期或时间戳的表达式。date 或 timestamp 的列或表达式必须包含 datepart 中指定的日期部分。

返回类型

DOUBLE

示例

DATE_PART 函数的原定设置列名是 pgdate_part。

以下示例从时间戳文本中查找分钟。

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

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

以下示例从时间戳文本中查找周编号。周编号计算遵循 ISO 8601 标准。有关更多信息，请参阅 Wikipedia 中的 ISO 8601。

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

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

以下示例从时间戳文本中查找月份中的某个日期。

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

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

下面的示例从时间戳文本中查找星期几信息。周编号计算遵循 ISO 8601 标准。有关更多信息，请参阅 Wikipedia 中的 ISO 8601。

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

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

以下示例从时间戳文本中查找世纪。世纪计算遵循 ISO 8601 标准。有关更多信息，请参阅 Wikipedia 中的 ISO 8601。

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

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

以下示例从时间戳文本中查找千禧年。千禧年计算遵循 ISO 8601 标准。有关更多信息，请参阅 Wikipedia 中的 ISO 8601。

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

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

以下示例从时间戳文本中查找微秒。微秒计算遵循 ISO 8601 标准。有关更多信息，请参阅 Wikipedia 中的 ISO 8601。

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

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

以下示例从日期文本中查找月份。

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

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

以下示例将 DATE_PART 函数应用于表中的列。

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


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

您可以用全名或缩写来命名日期部分；在这种情况下，w 代表星期数。

星期日期部分返回一个从 0-6 整数，从星期日开始。将 DATE_PART 与 dow (DAYOFWEEK) 结合使用以查看星期六的活动。

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

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

DATE_TRUNC 函数

DATE_TRUNC 函数根据您指定的日期部分（如小时、天或月）截断时间戳表达式或文字。

语法

```
DATE_TRUNC('datepart', timestamp)
```

参数

datepart

截断时间戳值的日期部分。输入时间戳被截断为输入 datepart 的精度。例如，month 截断至每月的第一天。有效格式如下所示：

- microsecond、microseconds
- millisecond、milliseconds
- second、seconds
- minute、minutes
- hour、hours
- day、days
- week、weeks
- month、months
- quarter、quarters
- year、years
- decade、decades
- century、centuries
- millennium、millennia

有关某些格式的缩写的更多信息，请参阅日期或时间戳函数的日期部分

timestamp

时间戳列或隐式转换为时间戳的表达式。

返回类型

TIMESTAMP

示例

将输入时间戳截断至秒。

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

将输入时间戳截断至分钟。

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

将输入时间戳截断至小时。

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

将输入时间戳截断至天。

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

将输入时间戳截断至一个月的第一天。

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

将输入时间戳截断至一个季度的第一天。

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

将输入时间戳截断至一年的第一天。

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

将输入时间戳截断至一个世纪的第一天。

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

将输入时间戳截断至某周的星期一。

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

在以下示例中，DATE_TRUNC 函数使用“周”日期部分返回每周星期一的日期。

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

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

EXTRACT 函数

EXTRACT 函数返回 TIMESTAMP、TIMESTAMPTZ、TIME 或 TIMETZ 值中的日期或时间部分。示例包括时间戳中的日、月、年、小时、分钟、秒、毫秒或微秒。

语法

```
EXTRACT(datepart FROM source)
```

参数

datepart

要提取的日期或时间的子字段，例如日、月、年、小时、分钟、毫秒或微秒。有关可能的值，请参阅日期或时间戳函数的日期部分。

源

计算结果为 TIMESTAMP、TIMESTAMPTZ、TIME 或 TIMETZ 数据类型的列或表达式。

返回类型

如果 source 值的计算结果为数据类型 TIMESTAMP、TIME 或 TIMETZ，则为 INTEGER。

如果 source 值的计算结果为数据类型 TIMESTAMPTZ，则为 DOUBLE PRECISION。

TIMESTAMP 示例

以下示例确定支付价格为 10000 美元或更高的销售周数。

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

salesid | weeknum
--------+---------
 159073 |       6
 160318 |       8
 161723 |      26
```

以下示例从文本时间戳值返回分钟值。

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

date_part
--
```

以下示例从文本时间戳值返回毫秒值。

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

date_part
-----------
101
```

TIMESTAMPZ 示例

以下示例从文本 timestamptz 值返回年份值。

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

date_part
-----------
1997
```

TIME 示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

```
select time_val from time_test;

time_val
--------------------
20:00:00
00:00:00.5550
00:58:00
```

以下示例从每个 time_val 中提取分钟数。

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
         0
         58
```

以下示例从每个 time_val 中提取小时数。

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

hours
-----------
         20
         0
         0
```

以下示例从文本值中提取毫秒。

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

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

TIMETZ 示例

下面的示例表 TIMETZ_TEST 具有一个列 TIMETZ_VAL（类型 TIMETZ），其中插入了三个值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

以下示例从每个 timetz_val 中提取小时数。

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

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

以下示例从文本值中提取毫秒。在处理提取之前，文本不会转换为 UTC。

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

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

以下示例从文本 timetz 值返回与 UTC 的时区偏移小时数。

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

```
date_part
-----------
-7
```

GETDATE 函数

GETDATE 函数返回当前会话时区（预设情况下为 UTC）中的当前日期和时间。

它返回当前语句的开始日期或时间，即使它在事务块中也是如此。

语法

```
GETDATE()
```

括号为必填项。

返回类型

TIMESTAMP

示例

以下示例使用 GETDATE 函数返回当前日期的完整时间戳。

```
select getdate();
```

SYSDATE 函数

SYSDATE 返回当前会话时区（预设情况下为 UTC）中的当前日期和时间。

> **Note**
>
> SYSDATE 返回当前事务的开始日期和时间，而不是当前语句的开始日期和时间。

语法

```
SYSDATE
```

此函数不需要任何参数。

返回类型

TIMESTAMP

示例

以下示例使用 SYSDATE 函数返回当前日期的完整时间戳。

```
select sysdate;

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

以下示例使用 TRUNC 函数内的 SYSDATE 函数来返回没有时间的当前日期。

```
select trunc(sysdate);

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

以下查询返回介于发出查询的日期和 120 天之前的任何日期之间的日期的销售信息。

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

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

TIMEOFDAY 函数

TIMEOFDAY 是一个特殊的别名，用于将工作日、日期和时间作为字符串值返回。它返回当前语句的时间字符串，即使它在事务块内也是如此。

语法

```
TIMEOFDAY()
```

返回类型

VARCHAR

示例

以下示例通过使用 TIMEOFDAY 函数返回当前日期和时间。

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

TO_TIMESTAMP 函数

TO_TIMESTAMP 将 TIMESTAMP 字符串转换为 TIMESTAMPTZ。

语法

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

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

参数

timestamp

以 format 指定的格式表示时间戳值的字符串。如果将此参数留为空，则时间戳值默认为 `0001-01-01 00:00:00`。

format

一个字符串文本，用于定义 timestamp 值的格式。包含时区的格式（**TZ**、**tz**，或者 **OF**）不支持作为输入。有关有效的时间戳格式，请参阅日期时间格式字符串。

is_strict

一个可选的布尔值，它指定在输入时间戳值超出范围时是否返回错误。当 is_strict 被设置为 TRUE 时，如果存在超出范围的值，则返回错误。当 is_strict 被设置为 FALSE（默认值）时，则接受溢出值。

返回类型

TIMESTAMPTZ

示例

以下示例演示使用 TO_TIMESTAMP 函数将 TIMESTAMP 字符串转换为 TIMESTAMPTZ。

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

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

可以传递日期的 TO_TIMESTAMP 部分。其余日期部分设置为默认值。时间包括在输出中：

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp
---------------------------
2017-01-01 00:00:00+00
```

以下 SQL 语句将字符串“2011-12-18 24:38:15”转换为 TIMESTAMPTZ。得到的结果是第二天的 TIMESTAMPTZ，因为小时数超过 24 小时：

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

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

以下 SQL 语句将字符串“2011-12-18 24:38:15”转换为 TIMESTAMPTZ。结果产生错误，因为时间戳中的时间值超过 24 小时：

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

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

日期或时间戳函数的日期部分

下表标识了作为以下函数参数接受的日期部分和时间部分的名称和缩写：

- DATEADD
- DATEDIFF
- DATE_PART
- EXTRACT

日期部分或时间部分	缩写
millennium、millennia	mil、mils
century、centuries	c、cent、cents
decade、decades	dec、decs
纪元	epoch（由 EXTRACT 提供支持）
year、years	y、yr、yrs
quarter、quarters	qtr、qtrs
month、months	mon,、mons
week、weeks	w
星期几	dayofweek、dow、dw、weekday（由 DATE_PART 和 EXTRACT 函数 提供支持） 返回 0–6 的整数（星期日是第一个数）。

日期部分或时间部分	缩写
	Note DOW 日期部分的运行方式与用于日期时间格式字符串的星期 (D) 日期部分不同。D 是基于 1–7 的整数，其中星期日是 1。有关更多信息，请参阅日期时间格式字符串。
一年中的日期	dayofyear、doy、dy、yearday（由 EXTRACT提供支持）
day、days	d
hour、hours	h、hr、hrs
minute、minutes	m、min、mins
second、seconds	s、sec、secs
millisecond、milliseconds	ms、msec、msecs、msecond、mseconds、millisec、millisecs、millisecon
microsecond、microseconds	microsec、microsecs、microsecond、usecond、useconds、us、usec、usecs
timezone、timezone_hour、timezone_minute	由 EXTRACT 支持，仅用于带有时区的时间戳 (TIMESTAMPTZ)。

秒、毫秒和微秒导致的结果差异

当不同的日期函数指定秒、毫秒或微秒作为日期部分时，查询结果会出现细微差异：

- EXTRACT 函数仅返回指定日期部分的整数，忽略较高级别和较低级别的日期部分。如果指定的日期部分为秒，则结果中不包括毫秒和微秒。如果指定的日期部分为毫秒，则不包括秒和微秒。如果指定的日期部分为微秒，则不包括秒和毫秒。
- DATE_PART 函数返回时间戳的完整秒部分，无论指定的日期部分是什么，从而根据需要返回十进制值或整数。

CENTURY、EPOCH、DECADE 和 MIL 说明

CENTURY 或 CENTURIES

AWS Clean Rooms 将世纪解释为从年份 ## #1 开始并以年份结尾：###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

EPOCH 的 AWS Clean Rooms 实现与 1970-01-01 00:00:00.00.000 无关，与集群所在的时区无关。根据集群所在的时区，您可能需要按小时差来抵消结果。

DECADE 或 DECADES

AWS Clean Rooms 根据通用日历解释 “十年” 或 “十年” 的日期部分。例如，由于公历从第一年开始，因此第一个十年（第 1 个十年）是 0001-01-01 到 0009-12-31，而第二个十年（第 2 个十年）是 0010-01-01 到 0019-12-31。例如，十年 201 为 2000-01-01 - 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 或 MILS

AWS Clean Rooms 将 MIL 解释为从年 #001 的第一天开始，到一年的最后一天结束：#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)
```

哈希函数

哈希函数是将数值输入值转换为另一个值的数学函数。AWS Clean Rooms 支持以下哈希函数：

主题

- MD5 函数
- SHA 函数
- SHA1 函数
- SHA2 函数
- MURMUR3_32_HASH

MD5 函数

使用 MD5 加密哈希函数将长度可变的字符串转换为以 128 位校验和的十六进制值的文本表示形式表示的 32 字符字符串。

语法

```
MD5(string)
```

参数

string

一个长度可变的字符串。

返回类型

MD5 函数返回以 128 位校验和的十六进制值的文本表示形式表示的 32 字符字符串。

示例

以下示例显示了字符串“AWS Clean Rooms”的 128 位值：

```
select md5('AWS Clean Rooms');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

SHA 函数

SHA1 函数的同义词。

请参阅 SHA1 函数。

SHA1 函数

SHA1 函数使用 SHA1 加密哈希函数将长度可变的字符串转换为以 160 位校验和的十六进制值的文本表示形式表示的 40 个字符的字符串。

语法

SHA1 是 SHA 函数 的同义词。

```
SHA1(string)
```

参数

string

一个长度可变的字符串。

返回类型

SHA1 函数返回以 160 位校验和的十六进制值的文本表示形式表示的 40 个字符的字符串。

示例

以下示例返回单词“AWS Clean Rooms”的 160 位值：

```
select sha1('AWS Clean Rooms');
```

SHA2 函数

SHA2 函数使用 SHA2 加密哈希函数将长度可变的字符串转换为一个字符串。该字符串是具有指定位数的校验和的十六进制值的文本表示形式。

语法

```
SHA2(string, bits)
```

参数

string

一个长度可变的字符串。

integer

哈希函数中的位数。有效值为 0（与 256 相同）、224、256、384 和 512。

返回类型

SHA2 函数返回一个字符串，该字符串是校验和的十六进制值的文本表示形式；如果位数无效，此函数将返回一个空字符串。

示例

以下示例返回单词“AWS Clean Rooms”的 256 位值：

```
select sha2('AWS Clean Rooms', 256);
```

MURMUR3_32_HASH

MURMUR3_32_HASH 函数计算包括数值和字符串类型在内的所有常见数据类型的 32 位 Murmur3A 非加密哈希。

语法

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

参数

值

要进行哈希处理的输入值。AWS Clean Rooms 对输入值的二进制表示进行哈希处理。此行为类似于 FNV_HASH，但值会转换为由 Apache Iceberg 32 位 Murmur3 哈希规范指定的二进制表示形式。

种子

哈希函数的 INT 种子。此参数是可选的。如果未给定，AWS Clean Rooms 使用默认种子 0。这样可以组合多个列的哈希，而无需任何转换或联接。

返回类型

此函数返回 INT。

示例

以下示例分别返回数字、字符串“AWS Clean Rooms”以及两者的联接的 Murmur3 哈希值。

```
select MURMUR3_32_HASH(1);

    MURMUR3_32_HASH
----------------------
```

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

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

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

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

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

使用说明

要计算具有多列的表的哈希，可以计算第一列的 Murmur3 哈希，并将其作为种子传递给第二列的哈希。然后，它将第二列的 Murmur3 哈希作为种子传递给第三列的哈希。

以下示例创建种子来对包含多列的表进行哈希处理。

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

同一个属性可用于计算字符串联接的哈希。

```
select MURMUR3_32_HASH('abcd');

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

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

   MURMUR3_32_HASH
----------------------
```

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

哈希函数使用输入的类型来确定要进行哈希处理的字节数。如有必要，使用强制转换来强制特定类型。

以下示例使用不同的输入类型来生成不同的结果。

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

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

```
select MURMUR3_32_HASH(1);

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

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

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

JSON 函数

当您需要存储相对较小的一组键值对时，您可以通过以 JSON 格式存储数据来节省空间。由于 JSON 字符串可存储在单个列中，因此使用 JSON 可能比以表格格式存储数据更高效。

Example

例如，假设您有一个稀疏表，在此表中，您需要设置多个列来完整表示所有可能的属性。但大多数列值对任何给定行或任何给定列为 NULL。通过将 JSON 用于存储，您可能能够将行的数据以键值对的形式存储在单个 JSON 字符串中并删除稀疏填充的表列。

此外，您还可以轻松修改 JSON 字符串以存储其他键值对，而无需向表添加列。

我们建议慎用 JSON。若要存储较大的数据集，JSON 不是一个好的选择，因为将分散的数据存储在单个列中后，JSON 不会利用 AWS Clean Rooms 的列存储架构。

JSON 使用 UTF-8 编码的文本字符串，因此 JSON 字符串可存储为 CHAR 或 VARCHAR 数据类型。如果字符串包含多字节字符，则使用 VARCHAR。

JSON 字符串必须是根据以下规则正确设置格式的 JSON：

- 根级别的 JSON 可以是 JSON 对象或 JSON 数组。JSON 对象是用大括号括起的一组无序的键值对（由逗号分隔）。

  例如 `{"one":1, "two":2}`
- JSON 数组是用方括号括起的一组有序值（由逗号分隔）。

  以下是示例：`["first", {"one":1}, "second", 3, null]`
- JSON 数组使用从零开始的索引；数组中的第一个元素位于位置 0。在 JSON 键:值对中，键是用双引号括起的字符串。
- JSON 值可能为以下任一值：
  - JSON 对象
  - JSON 数组
  - 用双引号括起的字符串
  - 数字 (整数和浮点)
  - 布尔值
  - Null
- 空对象和空数组是有效的 JSON 值。
- JSON 字段区分大小写。
- 将忽略 JSON 结构元素之间的空格（如 `{ }, [ ]`）。

AWS Clean Rooms JSON 函数和 AWS Clean Rooms COPY 命令使用相同的方法处理 JSON 格式的数据。

主题

- CAN_JSON_PARSE 函数
- JSON_EXTRACT_ARRAY_ELEMENT_TEXT 函数
- JSON_EXTRACT_PATH_TEXT 函数

- JSON_PARSE 函数
- JSON_SERIALIZE 函数
- JSON_SERIALIZE_TO_VARBYTE 函数

CAN_JSON_PARSE 函数

CAN_JSON_PARSE 函数以 JSON 格式解析数据，如果可以使用 JSON_PARSE 函数将结果转换为 SUPER 值，则返回 `true`。

语法

```
CAN_JSON_PARSE(json_string)
```

参数

json_string

以 VARBYTE 或 VARCHAR 形式返回序列化 JSON 的表达式。

返回类型

BOOLEAN

示例

要查看是否可以将 JSON 数组 `[10001,10002,"abc"]` 转换为 SUPER 数据类型，请使用以下示例。

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

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

JSON_EXTRACT_ARRAY_ELEMENT_TEXT 函数

JSON_EXTRACT_ARRAY_ELEMENT_TEXT 函数返回 JSON 字符串的最外侧数组中的 JSON 数组元素（使用从零开始的索引）。数组中的第一个元素位于位置 0。如果索引为负或超出界

限，JSON_EXTRACT_ARRAY_ELEMENT_TEXT 将返回空字符串。如果 null_if_invalid 参数设置为 `true` 并且 JSON 字符串无效，函数将返回 NULL 而不是返回错误。

有关更多信息，请参阅JSON 函数。

语法

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

参数

json_string

格式正确的 JSON 字符串。

pos

一个整数，表示要返回的数组元素的索引（使用从零开始的数组索引）。

null_if_invalid

一个布尔值，指定在输入 JSON 字符串无效时是否返回 NULL 而不返回错误。要在 JSON 无效时返回 NULL，请指定 `true` (t)。要在 JSON 无效时返回错误，请指定 `false` (f)。默认为 `false`。

返回类型

表示 pos 引用的 JSON 数组元素的 VARCHAR 字符串。

示例

以下示例返回位置 2 的数组元素，它是从零开始的数组索引的第三个元素：

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

json_extract_array_element_text
-------------------------------
113
```

在以下示例中，因为 JSON 无效，所以返回错误。

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

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

以下示例将 null_if_invalid 设置为 true，因此语句在 JSON 无效时返回 NULL 而不是返回错误。

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

json_extract_array_element_text
-------------------------------
```

JSON_EXTRACT_PATH_TEXT 函数

JSON_EXTRACT_PATH_TEXT 函数返回 JSON 字符串中的一系列路径元素引用的 key:value 对的值。JSON 路径最深可嵌套至 5 层。路径元素区分大小写。如果 JSON 字符串中不存在路径元素，JSON_EXTRACT_PATH_TEXT 将返回空字符串。如果 `null_if_invalid` 参数设置为 `true` 并且 JSON 字符串无效，函数将返回 NULL 而不是返回错误。

有关其他 JSON 函数的信息，请参阅 JSON 函数。

语法

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

参数

json_string

格式正确的 JSON 字符串。

path_elem

JSON 字符串中的路径元素。需要一个路径元素。可指定额外的路径元素，最深五层。

null_if_invalid

一个布尔值，指定在输入 JSON 字符串无效时是否返回 NULL 而不返回错误。要在 JSON 无效时返回 NULL，请指定 `true` (t)。要在 JSON 无效时返回错误，请指定 `false` (f)。默认为 `false`。

在 JSON 字符串中，AWS Clean Rooms 将 \n 识别为换行符，将 \t 识别为制表符。要加载反斜杠，请使用反斜杠 (\\ ) 对其进行转义。

返回类型

表示路径元素引用的 JSON 值的 VARCHAR 字符串。

示例

以下示例返回路径 'f4', 'f6' 的值。

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


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

在以下示例中，因为 JSON 无效，所以返回错误。

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

An error occurred when executing the SQL command:
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4', 'f6')
```

以下示例将 null_if_invalid 设置为 true，因此语句在 JSON 无效时返回 NULL 而不是返回错误。

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

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

以下示例返回路径 'farm', 'barn', 'color' 的值，其中检索到的值位于第三级。为更便于阅读，此示例使用 JSON lint 工具进行格式化。

```
select json_extract_path_text('{
    "farm": {
        "barn": {
```

```
            "color": "red",
            "feed stocked": true
        }
    }
}', 'farm', 'barn', 'color');

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

以下示例返回 NULL，因为 'color' 元素缺失。此示例使用 JSON lint 工具进行格式化。

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

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

如果 JSON 有效，则尝试提取缺失的元素将返回 NULL。

以下示例返回路径 'house', 'appliances', 'washing machine', 'brand' 的值。

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

```
      "dryer": {
        "brand": "Any Brand",
        "color": "white"
      }
    }
  }
}', 'house', 'appliances', 'washing machine', 'brand');

json_extract_path_text
----------------------
Any Brand
```

JSON_PARSE 函数

JSON_PARSE 函数以 JSON 格式解析数据并将其转换为 SUPER 表示形式。

要使用 INSERT 或 UPDATE 命令摄取到 SUPER 数据类型，请使用 JSON_PARSE 函数。当您使用 JSON_PARSE () 将 JSON 字符串解析为 SUPER 值时，某些限制适用。

语法

```
JSON_PARSE(json_string)
```

参数

json_string

以 varbyte 或 varchar 类型返回序列化 JSON 的表达式。

返回类型

SUPER

示例

以下示例是 JSON_PARSE 函数的示例。

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

```
 [10001,10002,"abc"]
(1 row)
```

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

JSON_SERIALIZE 函数

JSON_SERIALIZE 函数将 SUPER 表达式序列化为文本 JSON 表示形式，以遵守 RFC 8259。有关该 RFC 的更多信息，请参阅 JavaScript Object Notation (JSON) 数据交换格式。

SUPER 大小限制与数据块限制大致相同，并且 varchar 限制小于 SUPER 大小限制。因此，当 JSON 格式超出系统的 varchar 限制时，JSON_SERIALIZE 函数会返回一个错误。

语法

```
JSON_SERIALIZE(super_expression)
```

参数

super_expression

super 表达式或列。

返回类型

varchar

示例

以下示例将 SUPER 值序列化为字符串。

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

```
(1 row)
```

JSON_SERIALIZE_TO_VARBYTE 函数

JSON_SERIALIZE_TO_VARBYTE 函数将 SUPER 值转换为类似于 JSON_SERIALIZE() 的 JSON 字符串，但存储在 VARBYTE 值中。

语法

```
JSON_SERIALIZE_TO_VARBYTE(super_expression)
```

参数

super_expression

super 表达式或列。

返回类型

varbyte

示例

以下示例序列化 SUPER 值并以 VARBYTE 格式返回结果。

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

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

以下示例序列化 SUPER 值并将结果转换为 VARCHAR 格式。

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

```
  json_serialize_to_varbyte
---------------------------
```

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

数学函数

本部分描述 AWS Clean Rooms 中支持的数学运算符和函数。

主题

- ROUND 函数
- SIGN 函数
- SIN 函数
- SQRT 函数
- TRUNC 函数

数学运算符符号

下表列出了支持的数学运算符。

支持的运算符

操作符	描述	示例	结果
+	加	2 + 3	5
-	减	2 - 3	–1
*	乘	2 * 3	6
/	除	4 / 2	2
%	取模	5 % 4	1
^	幂	2.0 ^ 3.0	8
|/	平方根	| / 25.0	5
||/	立方根	|| / 27.0	3
@	绝对值	@ -5.0	5

示例

为给定交易计算支付的佣金加 2.00 美元手续费：

```
select commission, (commission + 2.00) as comm
from sales where salesid=10000;
```

```
commission | comm
-----------+-------
28.05      | 30.05
(1 row)
```

为给定交易计算销售价格的 20%：

```
select pricepaid, (pricepaid * .20) as twentypct
from sales where salesid=10000;

pricepaid | twentypct
----------+-----------
187.00    |    37.400
(1 row)
```

根据持续增长模式预测票的销售量。在此示例中，子查询将返回 2008 年销售的票数。在此后 10 年，该结果将以 5% 的连续增长率呈指数增长。

```
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)
```

查找带有大于或等于 2000 的日期 ID 的销售的总支付价格和总佣金。然后将总支付价格减去总佣金。

```
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)
```

ABS 函数

ABS 用于计算数字的绝对值，该数字可以是文本或计算结果为数字的表达式。

语法

```
ABS (number)
```

参数

number

数字或计算结果为数字的表达式。它可以是 SMALLINT、INTEGER、BIGINT、DECIMAL、FLOAT4 或 FLOAT8 类型。

返回类型

ABS 返回与其参数相同的数据类型。

示例

计算 -38 的绝对值：

```
select abs (-38);
abs
-------
38
(1 row)
```

计算 (14-76) 的绝对值：

```
select abs (14-76);
```

```
abs
-------
62
(1 row)
```

ACOS 函数

ACOS 是返回数字的反余弦的三角函数。返回值采用弧度形式且介于 0 和 PI 之间。

语法

```
ACOS(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 -1 的反余弦，请使用以下示例。

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

ASIN 函数

ASIN 是返回数字的反正弦的三角函数。返回值采用弧度形式且介于 PI/2 和 -PI/2 之间。

语法

```
ASIN(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 1 的反正弦，请使用以下示例。

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

ATAN 函数

ATAN 是返回数字的反正切的三角函数。返回值采用弧度形式且介于 -PI 和 PI 之间。

语法

```
ATAN(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 1 的反正切并将其乘以 4，请使用以下示例。

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

ATAN2 函数

ATAN2 是一个三角函数，它返回两个数值相除的结果的反正切。返回值采用弧度形式且介于 PI/2 和 -PI/2 之间。

语法

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

参数

number1

DOUBLE PRECISION 数值。

number2

DOUBLE PRECISION 数值。

返回类型

DOUBLE PRECISION

示例

要返回 2/2 的反正切并将其乘以 4，请使用以下示例。

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

CBRT 函数

CBRT 函数是计算数字的立方根的数学函数。

语法

```
CBRT (number)
```

参数

CBRT 将 DOUBLE PRECISION 数作为参数。

返回类型

CBRT 返回 DOUBLE PRECISION 数。

示例

计算为给定交易支付的佣金的立方根：

```
select cbrt(commission) from sales where salesid=10000;

cbrt
------------------
3.03839539048843
(1 row)
```

CEILING（或 CEIL）函数

CEILING 或 CEIL 函数用于将数字向上舍入到下一个整数。（FLOOR 函数将数字向下舍入到下一个整数）

语法

```
CEIL | CEILING(number)
```

参数

number

数字或计算结果为数字的表达式。它可以是 SMALLINT、INTEGER、BIGINT、DECIMAL、FLOAT4 或 FLOAT8 类型。

返回类型

CEILING 和 CEIL 返回与其参数相同的数据类型。

示例

计算为给定销售交易支付的佣金的上限：

```
select ceiling(commission) from sales
where salesid=10000;

ceiling
---------
29
(1 row)
```

COS 函数

COS 是返回数字的余弦的三角函数。返回值采用弧度形式且介于 -1 和 1 之间（含）。

语法

```
COS(double_precision)
```

参数

number

输入参数是双精度数。

返回类型

COS 函数返回双精度数。

示例

以下示例返回 0 的余弦：

```
select cos(0);
cos
-----
1
(1 row)
```

以下示例返回 PI 的余弦：

```
select cos(pi());
cos
-----
-1
(1 row)
```

COT 函数

COT 是返回数字的余切的三角函数。输入参数必须为非零。

语法

```
COT(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 1 的余切，请使用以下示例。

```
SELECT COT(1);

+--------------------+
|        cot         |
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

DEGREES 函数

将用弧度表示的角度转换用度表示。

语法

```
DEGREES(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 0.5 弧度的等效度数，请使用以下示例。

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
```

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

要将 PI 弧度转换为度数，请使用以下示例。

```
SELECT DEGREES(pi());

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

DEXP 函数

DEXP 函数返回双精度数的采用科学表示法的指数值。DEXP 函数和 EXP 函数的唯一区别在于 DEXP 的参数必须为 DOUBLE PRECISION。

语法

```
DEXP(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

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

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

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

DLOG1 函数

DLOG1 函数返回输入参数的自然对数。

DLOG1 函数是 LN 函数 的同义词。

DLOG10 函数

DLOG10 返回输入参数的以 10 为底的对数。

DLOG10 函数是 LOG 函数 的同义词。

语法

```
DLOG10(number)
```

参数

number

输入参数是双精度数。

返回类型

DLOG10 函数返回双精度数。

示例

以下示例返回数字 100 的以 10 为底的对数：

```
select dlog10(100);

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

EXP 函数

EXP 函数实施数值表达式的指数函数，即以自然对数 e 为底数，对表达式求次方。EXP 函数是 LN 函数的反函数。

语法

```
EXP (expression)
```

参数

expression

表达式必须是 INTEGER、DECIMAL 或 DOUBLE PRECISION 数据类型。

返回类型

EXP 返回 DOUBLE PRECISION 数。

示例

使用 EXP 函数根据持续增长模式预测票的销售量。在此示例中，子查询将返回 2008 年销售的票数。该结果将乘以 EXP 函数的结果（指定了在接下来 10 年保持 7% 的持续增长率）。

```
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)
```

FLOOR 函数

FLOOR 函数将数字向下舍入到下一个整数。

语法

```
FLOOR (number)
```

参数

number

数字或计算结果为数字的表达式。它可以是 SMALLINT、INTEGER、BIGINT、DECIMAL、FLOAT4 或 FLOAT8 类型。

返回类型

FLOOR 返回与其参数相同的数据类型。

示例

此示例显示在使用 FLOOR 函数之前和之后为给定的销售交易支付的佣金值。

```
select commission from sales
where salesid=10000;

floor
-------
28.05
(1 row)

select floor(commission) from sales
where salesid=10000;

floor
-------
28
(1 row)
```

LN 函数

LN 函数返回输入参数的自然对数。

LN 函数是 DLOG1 函数 的同义词。

语法

```
LN(expression)
```

参数

expression

对其执行函数的目标列或表达式。

> **Note**
>
> 如果表达式引用了 AWS Clean Rooms 用户创建的表或 AWS Clean Rooms STL 或 STV 系统表，则对于某些数据类型，此函数会返回错误。

具有以下数据类型的表达式在引用了用户创建的表或系统表时将产生错误。

- BOOLEAN
- CHAR
- DATE
- DECIMAL 或 NUMERIC
- TIMESTAMP
- VARCHAR

具有以下数据类型的表达式可在用户创建的表以及 STL 或 STV 系统表上成功运行：

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

返回类型

LN 函数返回与表达式相同的类型。

示例

以下示例返回数字 2.718281828 的自然对数（即以 e 为底的对数）：

```
select ln(2.718281828);
ln
--------------------
0.9999999998311267
(1 row)
```

请注意，结果约等于 1。

此示例返回 USERS 表的 USERID 列中的值的自然对数：

```
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)
```

LOG 函数

返回数字的以 10 为底的对数。

DLOG10 函数的同义词。

语法

```
LOG(number)
```

参数

number

输入参数是双精度数。

返回类型

LOG 函数返回双精度数。

示例

以下示例返回数字 100 的以 10 为底的对数：

```
select log(100);
dlog10
--------
2
(1 row)
```

MOD 函数

返回两个数字的余数，也称为取模 运算。将第一个参数除以第二个参数来计算结果。

语法

```
MOD(number1, number2)
```

参数

number1

第一个输入参数是 INTEGER、SMALLINT、BIGINT 或 DECIMAL 数。如果其中一个参数是 DECIMAL 类型，则另一参数也必须是 DECIMAL 类型。如果其中一个参数是 INTEGER，则另一参数可以是 INTEGER、SMALLINT 或 BIGINT。两个参数也都可以是 SMALLINT 或 BIGINT，但如果一个参数是 BIGINT，则另一个参数不能是 SMALLINT。

number2

第二个参数是 INTEGER、SMALLINT、BIGINT 或 DECIMAL 数。相同的数据类型规则与 number1 一样适用于 number2。

返回类型

有效的返回类型是 DECIMAL、INT、SMALLINT 或 BIGINT。如果两个参数属于相同的类型，MOD 函数的返回类型是与输入参数相同的数值类型。但是，如果其中一个输入参数是 INTEGER，返回类型也将是 INTEGER。

使用说明

您可以使用 % 作为取模运算符。

示例

以下示例返回一个数字除以另一个数字后的余数：

```
SELECT MOD(10, 4);

 mod
------
 2
```

以下示例返回一个小数结果：

```
SELECT MOD(10.5, 4);

 mod
------
 2.5
```

您可以转换参数值：

```
SELECT MOD(CAST(16.4 as integer), 5);

 mod
------
 1
```

通过将第一个参数除以 2 来检查该参数是否为偶数：

```
SELECT mod(5,2) = 0 as is_even;

 is_even
--------
 false
```

您可以使用 % 作为取模运算符：

```
SELECT 11 % 4 as remainder;
```

```
 remainder
-----------
 3
```

以下示例返回 CATEGORY 表中的奇数类别的信息：

```
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)
```

PI 函数

PI 函数返回 14 个小数位的 pi 值。

语法

```
PI()
```

返回类型

DOUBLE PRECISION

示例

要返回 pi 的值，请使用以下示例。

```
SELECT PI();
```

```
+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

POWER 函数

POWER 函数是让一个数值表达式自乘到另一个数值表达式的幂的指数函数。例如，2 的三次幂的计算公式为 POWER(2,3)，结果为 8。

语法

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

参数

expression1

要自乘的数值表达式。必须是 INTEGER、DECIMAL 或 FLOAT 数据类型。

expression2

让 expression1 自乘到的幂。必须是 INTEGER、DECIMAL 或 FLOAT 数据类型。

返回类型

DOUBLE PRECISION

示例

```
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 |
```

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

RADIANS 函数

RADIANS 函数将用度表示的角度转换为用弧度表示。

语法

```
RADIANS(number)
```

参数

number

输入参数是 DOUBLE PRECISION 数。

返回类型

DOUBLE PRECISION

示例

要返回 180 度的等效弧度，请使用以下示例。

```
SELECT RADIANS(180);

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

RANDOM 函数

RANDOM 函数生成介于 0.0（含）和 1.0（不含）之间的随机值。

语法

```
RANDOM()
```

返回类型

RANDOM 返回 DOUBLE PRECISION 数。

示例

1. 计算介于 0 和 99 之间的随机值。如果随机数为 0 - 1，此查询将生成 0 - 100 的随机值：

```
select cast (random() * 100 as int);

INTEGER
------
24
(1 row)
```

2. 检索 10 个项目的统一随机样本：

```
select *
from sales
order by random()
limit 10;
```

现在检索 10 个项目的随机样本，但选择与其价格成比例的项目。例如，价格是另一个两倍的项目在查询结果中出现的可能性是其两倍：

```
select *
from sales
order by log(1 - random()) / pricepaid
limit 10;
```

3. 此示例使用 SET 命令设置一个 SEED 值，以使 RANDOM 生成可预测的数字序列。

首先，返回三个 RANDOM 整数，而不先设置 SEED 值：

```
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)
```

现在，将 SEED 值设置为 .25，并返回 3 个以上的 RANDOM 数字：

```
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)
```

最后，将 SEED 值重置为 .25，并验证 RANDOM 是否返回与前三个调用相同的结果：

```
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)
```

ROUND 函数

ROUND 函数将数字舍入到最近的整数或小数。

ROUND 函数可以选择性地以整数形式包含另一个参数，指示在任意方向舍入到的小数位数。当您不提供第二个参数时，函数会舍入到最接近的整数。指定第二个参数 >n 时，函数将舍入为最接近的数字，其中精度为 n 个小数位。

语法

```
ROUND (number [ , integer ] )
```

参数

number

数字或计算结果为数字的表达式。它可以是 DECIMAL 或 FLOAT8 类型。 AWS Clean Rooms 可以根据隐式转换规则转换其他数据类型。

integer（可选）

一个整数，指示任意方向四舍五入的小数位数。

返回类型

ROUND 返回与输入参数相同的数字数据类型。

示例

将为给定交易支付的佣金舍入到最近的整数。

```
select commission, round(commission)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    28
(1 row)
```

将为给定交易支付的佣金舍入到第一个小数位。

```
select commission, round(commission, 1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |  28.1
(1 row)
```

对于同一查询，请沿相反的方向扩展精度。

```
select commission, round(commission, -1)
from sales where salesid=10000;

commission | round
-----------+-------
     28.05 |    30
(1 row)
```

SIGN 函数

SIGN 函数返回数字的符号（正或负）。SIGN 函数的结果为 1、-1 或 0，表示参数的符号。

语法

```
SIGN (number)
```

参数

number

数字或计算结果为数字的表达式。它可以是 DecimaLor FLOAT8 类型。 AWS Clean Rooms 可以根据隐式转换规则转换其他数据类型。

返回类型

SIGN 返回与输入参数相同的数字数据类型。如果输入为 DECIMAL，则输出为 DECIMAL(1,0)。

示例

要从 SALES 表中确定为给定交易支付的佣金的符号，请使用以下示例。

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

SIN 函数

SIN 是返回数字的正弦的三角函数。返回值介于 -1 与 1 之间。

语法

```
SIN(number)
```

参数

number

以弧度表示的 DOUBLE PRECISION 数值。

返回类型

DOUBLE PRECISION

示例

要返回 -PI 的正弦，请使用以下示例。

```
SELECT SIN(-PI());

+-------------------------+
|           sin           |
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

SQRT 函数

SQRT 函数返回数字值的平方根。平方根是一个乘以自身以得到给定值的数字。

语法

```
SQRT (expression)
```

参数

expression

表达式必须具有整数、小数或浮点数据类型。表达式可以包含函数。系统可能会执行隐式类型转换。

返回类型

SQRT 返回 DOUBLE PRECISION 数。

示例

以下示例返回数字的平方根。

```
select sqrt(16);

sqrt
---------------
```

```
4
```

以下示例执行隐式类型转换。

```
select sqrt('16');

sqrt
---------------
4
```

以下示例嵌套函数以执行更复杂的任务。

```
select sqrt(round(16.4));

sqrt
---------------
4
```

以下示例得出给定圆面积时的半径长度。例如，当给定以平方英寸为单位的面积时，它以英寸为单位计算半径。示例中的面积为 20。

```
select sqrt(20/pi());
```

这将返回值 5.046265044040321。

以下示例返回 SALES 表中 COMMISSION 值的平方根。COMMISSION 列是 DECIMAL 列。此示例说明如何在具有更复杂条件逻辑的查询中使用该函数。

```
select sqrt(commission)
from sales where salesid < 10 order by salesid;

sqrt
------------------
10.4498803820905
3.37638860322683
7.24568837309472
5.1234753829798
...
```

以下查询返回同一组 COMMISSION 值的平方根的舍入值。

```
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
...
```

有关中示例数据的更多信息 AWS Clean Rooms，请参阅示例数据库。

TRUNC 函数

TRUNC 函数将数字截断为前一个整数或小数。

TRUNC 函数可以选择性地以整数形式包含另一个参数，指示在任意方向舍入到的小数位数。当您不提供第二个参数时，函数会舍入到最接近的整数。当指定第二个参数 >n 时，函数将舍入为最接近的数字 >n 精度的小数位。此函数还会截断时间戳并返回日期。

语法

```
TRUNC (number [ , integer ] |
timestamp )
```

参数

number

数字或计算结果为数字的表达式。它可以是 DECIMAL 或 FLOAT8 类型。 AWS Clean Rooms 可以根据隐式转换规则转换其他数据类型。

integer（可选）

一个整数，指示精度在任意方向的小数位数。如果未提供整数，数字将作为整数截断；如果指定了整数，数字将截断到指定的小数位。

timestamp

该函数也可返回时间戳中的日期。（要返回以 00:00:00 作为时间的时间戳值，请将函数结果强制转换为时间戳。）

返回类型

TRUNC 返回与第一个输入参数的数据类型相同的数据类型。对于时间戳，TRUNC 将返回日期。

示例

截断为给定销售交易支付的佣金。

```
select commission, trunc(commission)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   111

(1 row)
```

将同一佣金值截断到第一个小数位。

```
select commission, trunc(commission,1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 | 111.1

(1 row)
```

截断第二个参数为负值的佣金；111.15 向下舍入到 110。

```
select commission, trunc(commission,-1)
from sales where salesid=784;

commission | trunc
-----------+-------
    111.15 |   110
(1 row)
```

返回 SYSDATE 函数（返回时间戳）的结果的日期部分：

```
select sysdate;
```

```
timestamp
----------------------------
2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc
------------
2011-07-21
(1 row)
```

将 TRUNC 函数应用于 TIMESTAMP 列。返回类型为日期。

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

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

字符串函数

主题

- || (串联) 运算符
- BTRIM 函数
- CHAR_LENGTH 函数
- CHARACTER_LENGTH 函数
- CHARINDEX 函数
- CONCAT 函数
- LEFT 和 RIGHT 函数
- LEN 函数
- LENGTH 函数
- LOWER 函数
- LPAD 和 RPAD 函数
- LTRIM 函数

- POSITION 函数
- REGEXP_COUNT 函数
- REGEXP_INSTR 函数
- REGEXP_REPLACE 函数
- REGEXP_SUBSTR 函数
- REPEAT 函数
- REPLACE 函数
- REPLICATE 函数
- REVERSE 函数
- RTRIM 函数
- SOUNDEX 函数
- SPLIT_PART 函数
- STRPOS 函数
- SUBSTR 函数
- SUBSTRING 函数
- TEXTLEN 函数
- TRANSLATE 函数
- TRIM 函数
- UPPER 函数

字符串函数用于处理和操作字符串或计算结果为字符串的表达式。当这些函数中的 string 参数为文本值时，该参数必须括在单引号中。支持的数据类型包括 CHAR 和 VARCHAR。

以下部分提供了支持的函数的函数名称、语法和描述。对字符串的所有偏移都从 1 开始。

|| (串联) 运算符

联接位于 || 符号的任意一侧的两个表达式并返回联接后的表达式。

串联运算符类似于 CONCAT 函数。

> **Note**
>
> 对于 CONCAT 函数和联接运算符，如果一个或多个表达式为 null，则联接的结果也为 null。

语法

```
expression1 || expression2
```

参数

expression1、expression2

两个参数都可以是长度固定或长度可变的字符串或表达式。

返回类型

|| 运算符返回字符串。字符串的类型与输入参数的类型相同。

示例

以下示例将 USERS 表中的 FIRSTNAME 和 LASTNAME 字段联接：

```
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)
```

要联接可能包含 null 值的列，请使用 NVL 和 COALESCE 函数 表达式。以下示例使用 NVL 在遇到 NULL 时返回 0。

```
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
```

BTRIM 函数

BTRIM 函数通过删除前导空格和尾随空格或删除与可选的指定字符串匹配的前导字符和尾随字符来剪裁字符串。

语法

```
BTRIM(string [, trim_chars ] )
```

参数

string

要剪裁的输入 VARCHAR 字符串。

trim_chars

该 VARCHAR 字符串包含要匹配的字符。

返回类型

BTRIM 函数返回 VARCHAR 字符串。

示例

以下示例从字符串 ' abc ' 中剪裁前导和尾随空格：

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

以下示例从字符串 'xyzaxyzbxyzcxyz' 中删除前导和尾随 'xyz' 字符串。将删除前导和尾随的 'xyz'，但不会删除字符串内部的匹配字符。

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

以下示例从字符串 'setuphistorycassettes' 中删除与 trim_chars 列表 'tes' 中的任何字符相匹配的开头和结尾部分。在输入字符串开头或结尾部分，在 trim_chars 列表中未包含的其他字符之前出现的任何 t、e 或 s 都将被删除。

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
-----------------
 uphistoryca
```

CHAR_LENGTH 函数

LEN 函数的同义词。

请参阅 LEN 函数。

CHARACTER_LENGTH 函数

LEN 函数的同义词。

请参阅 LEN 函数。

CHARINDEX 函数

返回指定子字符串在字符串中的位置。

有关类似的函数，请参阅POSITION 函数和STRPOS 函数。

语法

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

参数

substring

要在 string 中搜索的子字符串。

string

要搜索的字符串或列。

返回类型

CHARINDEX 函数返回与子字符串的位置对应的整数（从 1 开始，不从 0 开始）。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。

使用说明

如果在 `string` 中未找到子字符串，CHARINDEX 将返回 0：

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

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

示例

以下示例显示字符串 `fish` 在单词 `dogfish` 中的位置：

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

```
 charindex
----------
         4
(1 row)
```

以下示例返回 SALES 表中 COMMISSION 超过 999.00 的销售交易的数量：

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

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

CONCAT 函数

CONCAT 函数将联接两个表达式并返回生成的表达式。要联接两个以上的表达式，请使用嵌套 CONCAT 函数。在两个表达式之间使用联接运算符（||）将生成与 CONCAT 函数相同的结果。

> **Note**
>
> 对于 CONCAT 函数和联接运算符，如果一个或多个表达式为 null，则联接的结果也为 null。

语法

```
CONCAT ( expression1, expression2 )
```

参数

expression1、expression2

两个参数可以是固定长度字符串、可变长度字符串、二进制表达式或计算结果为其中一个输入的表达式。

返回类型

CONCAT 返回一个表达式。表达式的数据类型与输入参数的数据类型相同。

如果输入表达式的类型不同，则 AWS Clean Rooms 尝试对其中一个表达式进行隐式类型转换。如果值无法转换，则会返回一个错误。

示例

以下示例联接两个字符文本：

```
select concat('December 25, ', '2008');

concat
-------------------
December 25, 2008
(1 row)
```

以下查询（使用 || 运算符而不是 CONCAT）将生成相同的结果：

```
select 'December 25, '||'2008';

concat
-------------------
December 25, 2008
(1 row)
```

以下示例使用两个 CONCAT 函数联接三个字符串：

```
select concat('Thursday, ', concat('December 25, ', '2008'));

concat
----------------------------
Thursday, December 25, 2008
(1 row)
```

要联接可能包含 null 值的列，请使用 NVL 和 COALESCE 函数。以下示例使用 NVL 在遇到 NULL 时返回 0。

```
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)
```

以下查询联接 VENUE 表中的 CITY 和 STATE 值：

```
select concat(venuecity, venuestate)
from venue
where venueseats > 75000
order by venueseats;

concat
-------------------
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

以下查询使用嵌套 CONCAT 函数。该查询将联接 VENUE 表中的 CITY 和 STATE 值，但会使用逗号和空格分隔生成的字符串：

```
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)
```

LEFT 和 RIGHT 函数

这些函数返回指定数量的位于字符串最左侧或最右侧的字符。

该数量基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。

语法

```
LEFT ( string,  integer )

RIGHT ( string,  integer )
```

参数

string

任何字符串或计算结果为字符串的任何表达式。

integer

一个正整数。

返回类型

LEFT 和 RIGHT 返回 VARCHAR 字符串。

示例

以下示例返回 ID 在 1000 和 1005 之间的事件的名称的最左侧和最右侧的 5 个字符：

```
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)
```

LEN 函数

以字符数形式返回指定字符串的长度。

语法

LEN 是 LENGTH 函数、CHAR_LENGTH 函数、CHARACTER_LENGTH 函数和 TEXTLEN 函数的同义词。

```
LEN(expression)
```

参数

expression

输入参数是 CHAR 或 VARCHAR 或其中一个有效输入类型的别名。

返回类型

LEN 函数返回一个整数，表示输入字符串中的字符的数量。

如果输入的是字符串，LEN 函数将返回多字节字符串中的字符的实际数量，而不是字节的数量。例如，存储 3 个 4 字节中文字符需要 VARCHAR(12) 列。LEN 函数将对同一字符串返回 3。

使用说明

长度计算对长度固定的字符串不计尾随空格，但对长度可变的字符串相反。

示例

以下示例将返回字符串 `français` 中的字节数和字符数。

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

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

以下示例返回没有尾随空格的字符串 cat 中的字符数以及有三个尾随空格的 cat 中的字符数：

```
select len('cat'), len('cat   ');
 len | len
-----+-----
   3 |   6
```

以下示例返回 VENUE 表中的 10 个最长的 VENUENAME 条目：

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

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

LENGTH 函数

LEN 函数的同义词。

请参阅 LEN 函数。

LOWER 函数

将字符串转换为小写。LOWER 支持 UTF-8 多字节字符，并且每个字符最多可以有 4 个字节。

语法

```
LOWER(string)
```

参数

string

输入参数是 VARCHAR 字符串（或任何其他可隐式转换为 VARCHAR 的数据类型，如 CHAR）。

返回类型

LOWER 函数返回与输入字符串具有相同数据类型的字符串。

示例

以下示例将 CATNAME 字段转换为小写：

```
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)
```

LPAD 和 RPAD 函数

这些函数根据指定长度在字符串前面或后面追加字符。

语法

```
LPAD (string1, length, [ string2 ])
```

```
RPAD (string1, length, [ string2 ])
```

参数

string1

一个字符串或计算结果为字符串的表达式，如字符列的名称。

length

一个用于定义函数结果的长度的整数。字符串的长度基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。如果 string1 的长度超过指定长度，它将被截断（在右侧）。如果 length 为负数，函数的结果将为空字符串。

string2

追加到 string1 前面或后面的一个或多个字符。此参数是可选的；如果未指定它，则使用空格。

返回类型

这些函数返回 VARCHAR 数据类型。

示例

将指定的一组事件名称截断到 20 个字符并在短于此长度的名称前面追加空格：

```
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)
```

将指定的一组事件名称截断到 20 个字符但在短于此长度的名称后面追加 0123456789。

```
select rpad(eventname,20,'0123456789') from event
where eventid between 1 and 5 order by 1;

 rpad
```

```
--------------------
Boris Godunov0123456
Gotterdammerung01234
Il Trovatore01234567
La Cenerentola (Cind
Salome01234567890123
(5 rows)
```

LTRIM 函数

从字符串的开头剪裁几个字符。删除只包含剪裁字符列表中的字符的最长字符串。当输入字符串中没有了剪裁字符时，剪裁即告完成。

语法

```
LTRIM( string [, trim_chars] )
```

参数

string

要剪裁的字符串列、表达式或字符串文本。

trim_chars

表示要从 string 的开头剪裁的字符的字符串列、表达式或字符串文本。如果未指定，则使用空格作为剪裁字符。

返回类型

LTRIM 函数返回与输入字符串（CHAR 或 VARCHAR）具有相同数据类型的字符串。

示例

以下示例从 `listime` 列中剪裁掉年份。字符串文本中的剪裁字符 `'2008-'` 表示要从左侧剪裁的字符。如果您使用剪裁字符 `'028-'`，则会获得相同的结果。

```
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
```

当 trim_chars 中的任意字符出现在 string 的开头时，LTRIM 都会予以删除。以下示例从 VENUENAME（VARCHAR 列）的开头剪裁字符“C”、“D”和“G”。

```
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
```

以下示例使用从 venueid 列中检索到的剪裁字符 2。

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

以下示例不剪裁任何字符，因为在 '0' 剪裁字符之前找到了 2。

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

以下示例使用默认的空格剪裁字符，从字符串的开头剪裁掉两个空格。

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

POSITION 函数

返回指定子字符串在字符串中的位置。

有关类似的函数，请参阅CHARINDEX 函数和STRPOS 函数。

语法

```
POSITION(substring IN string )
```

参数

substring

要在 string 中搜索的子字符串。

string

要搜索的字符串或列。

返回类型

POSITION 函数返回与子字符串的位置对应的整数（从 1 开始，而不是从 0 开始）。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。

使用说明

如果在字符串中未找到子字符串，POSITION 将返回 0：

```
select position('dog' in 'fish');

position
----------
 0
(1 row)
```

示例

以下示例显示字符串 fish 在单词 dogfish 中的位置：

```
select position('fish' in 'dogfish');

position
----------
 4
(1 row)
```

以下示例返回 SALES 表中 COMMISSION 超过 999.00 的销售交易的数量：

```
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)
```

REGEXP_COUNT 函数

在字符串中搜索正则表达式模式并返回指示该模式在字符串中出现的次数的整数。如果未找到匹配项，此函数将返回 0。

语法

```
REGEXP_COUNT ( source_string, pattern [, position [, parameters ] ] )
```

参数

source_string

要搜索的字符串表达式（如列名称）。

pattern

表示正则表达式模式的字符串文本。

position

指示在 source_string 中开始搜索的位置的正整数。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。默认值为 1。如果 position 小于 1，则搜索从 source_string 的第一个字符开始。如果 position 大于 source_string 中字符的数量，则结果为 0。

参数

一个或多个字符串，指示函数与模式的匹配方式。可能的值包括：

- c – 执行区分大小写的匹配。默认情况下，使用区分大小写的匹配。
- i – 执行不区分大小写的匹配。
- p – 使用 Perl 兼容正则表达式 (PCRE) 方言解释模式。

返回类型

整数

示例

以下示例计算三个字母序列出现的次数。

```
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');

 regexp_count
 --------------
            8
```

以下示例计算顶级域名为 org 或 edu 的次数。

```
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
```

下面的示例计算字符串 FOX 的出现次数，使用不区分大小写的匹配。

```
SELECT regexp_count('the fox', 'FOX', 1, 'i');

 regexp_count
 --------------
            1
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例使用区分大小写的匹配计算此类单词的出现次数。

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

 regexp_count
 --------------
            2
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的含义。此示例计算此类单词的出现次数，但与前面的示例不同，因为它使用了不区分大小写的匹配。

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');

 regexp_count
 --------------
            3
```

REGEXP_INSTR 函数

在字符串中搜索正则表达式模式并返回指示匹配子字符串的开始位置的整数。如果未找到匹配项，此函数将返回 0。REGEXP_INSTR 与 函数相似，只不过前者可让您在字符串中搜索正则表达式模式。

语法

```
REGEXP_INSTR ( source_string, pattern [, position [, occurrence] [, option
 [, parameters ] ] ] ] )
```

参数

source_string

要搜索的字符串表达式（如列名称）。

pattern

表示正则表达式模式的字符串文本。

position

指示在 source_string 中开始搜索的位置的正整数。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。默认值为 1。如果 position 小于 1，则搜索从 source_string 的第一个字符开始。如果 position 大于 source_string 中字符的数量，则结果为 0。

出现

一个正整数，指示要使用的模式的匹配项。REGEXP_INSTR 会跳过第一个 occurrence -1 匹配项。默认值为 1。如果 occurrence 小于 1 或大于 source_string 中的字符串，则会忽略搜索，并且结果为 0。

option

一个值，指示是否返回匹配项的第一个字符的位置 (0) ，或匹配项结尾后第一个字符的位置 (1)。非零值与 1 相同。默认值是 0。

参数

一个或多个字符串，指示函数与模式的匹配方式。可能的值包括：

- c – 执行区分大小写的匹配。默认情况下，使用区分大小写的匹配。
- i – 执行不区分大小写的匹配。
- e – 使用子表达式提取子字符串。

  如果 pattern 包含一个子表达式，REGEXP_INSTR 会使用 pattern 中的第一个子表达式来匹配子字符串。REGEXP_INSTR 仅考虑第一个子表达式；其他子表达式会被忽略。如果模式没有子表达式，REGEXP_INSTR 会忽略“e”参数。

- p – 使用 Perl 兼容正则表达式 (PCRE) 方言解释模式。

返回类型

整数

示例

以下示例搜索作为域名的开头的 @ 字符并返回第一个匹配项的开始位置。

```
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
```

以下示例搜索单词 Center 的变体并返回第一个匹配项的开始位置。

```
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
```

以下示例使用不区分大小写的匹配逻辑找到字符串 FOX 第一次出现的起始位置。

```
SELECT regexp_instr('the fox', 'FOX', 1, 1, 0, 'i');

 regexp_instr
 --------------
```

```
            5
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例查找第二个此类单词的起始位置。

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

 regexp_instr
 --------------
           21
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。本示例查找第二个此类单词的起始位置，但与前面的示例不同，因为它使用了不区分大小写的匹配。

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');

 regexp_instr
 --------------
           15
```

REGEXP_REPLACE 函数

在字符串中搜索正则表达式模式并将该模式的每个匹配项替换为指定字符串。REGEXP_REPLACE 与 REPLACE 函数相似，只不过前者可让您在字符串中搜索正则表达式模式。

REGEXP_REPLACE 与 TRANSLATE 函数和 REPLACE 函数相似，只不过 TRANSLATE 进行多次单字符替换，REPLACE 一次性将整个字符串替换为其他字符串，而 REGEXP_REPLACE 可让您在字符串中搜索正则表达式模式。

语法

```
REGEXP_REPLACE ( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

参数

source_string

要搜索的字符串表达式（如列名称）。

pattern

表示正则表达式模式的字符串文本。

replace_string

将替换模式的每个匹配项的字符串表达式（如列名称）。默认值是空字符串 ("")。

position

指示在 source_string 中开始搜索的位置的正整数。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。默认值为 1。如果 position 小于 1，则搜索从 source_string 的第一个字符开始。如果 position 大于 source_string 中的字符数量，则结果为 source_string。

参数

一个或多个字符串，指示函数与模式的匹配方式。可能的值包括：

- c – 执行区分大小写的匹配。默认情况下，使用区分大小写的匹配。
- i – 执行不区分大小写的匹配。
- p – 使用 Perl 兼容正则表达式 (PCRE) 方言解释模式。

返回类型

VARCHAR

如果 pattern 或 replace_string 为 NULL，则返回 NULL。

示例

以下示例删除电子邮件地址中的 @ 和域名。

```
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
```

以下示例将使用此值替换电子邮件地址的域名：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
```

下面的示例使用不区分大小写的匹配替换值 quick brown fox 内的字符串 FOX 的所有出现次数。

```
SELECT regexp_replace('the fox', 'FOX', 'quick brown fox', 1, 'i');

   regexp_replace
---------------------
 the quick brown fox
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例将此单词的每次出现替换为值 [hidden]。

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

        regexp_replace
-------------------------------
 [hidden] plain A1234 [hidden]
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例将此单词的每次出现替换为值 [hidden]，但与前面的示例不同，它使用不区分大小写的匹配。

```
SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

          regexp_replace
---------------------------------
 [hidden] plain [hidden] [hidden]
```

REGEXP_SUBSTR 函数

通过在字符串中搜索正则表达式模式，返回字符串中的字符。REGEXP_SUBSTR 与 SUBSTRING 函数 函数相似，只不过前者可让您在字符串中搜索正则表达式模式。如果函数无法将正则表达式与字符串中的任何字符匹配，则返回一个空字符串。

语法

```
REGEXP_SUBSTR ( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

参数

source_string

要搜索的字符串表达式。

pattern

表示正则表达式模式的字符串文本。

position

指示在 source_string 中开始搜索的位置的正整数。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。默认值为 1。如果 position 小于 1，则搜索从 source_string 的第一个字符开始。如果 position 大于 source_string 中的字符数量，则结果为空字符串 ("")。

出现

一个正整数，指示要使用的模式的匹配项。REGEXP_SUBSTR 会跳过第一个 occurrence -1 匹配项。默认值为 1。如果 occurrence 小于 1 或大于 source_string 中的字符串，则会忽略搜索，并且结果为 NULL。

参数

一个或多个字符串，指示函数与模式的匹配方式。可能的值包括：

- c – 执行区分大小写的匹配。默认情况下，使用区分大小写的匹配。
- i – 执行不区分大小写的匹配。
- e – 使用子表达式提取子字符串。

  如果 pattern 包含一个子表达式，REGEXP_SUBSTR 会使用 pattern 中的第一个子表达式来匹配子字符串。子表达式是模式中用括号括起的表达式。例如，模式 `'This is a (\\w+)'` 将第一个表达式与字符串 `'This is a '` 后接一个字词进行匹配。此时不返回模式，带 e 参数的 REGEXP_SUBSTR 仅返回子表达式内的字符串。

  REGEXP_SUBSTR 仅考虑第一个子表达式；其他子表达式会被忽略。如果模式没有子表达式，REGEXP_SUBSTR 会忽略“e”参数。
- p – 使用 Perl 兼容正则表达式 (PCRE) 方言解释模式。

返回类型

VARCHAR

示例

以下示例返回电子邮件地址中 @ 字符和域扩展名之间的部分。

```
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
```

以下示例使用不区分大小写的匹配返回与字符串 FOX 的第一次出现相对应的输入部分。

```
SELECT regexp_substr('the fox', 'FOX', 1, 1, 'i');

 regexp_substr
---------------
 fox
```

以下示例返回以小写字母开头的输入的第一部分。这在功能上与不带 c 参数的同一 SELECT 语句相同。

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');

 regexp_substr
---------------
 abc
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例返回与第二个此类单词相对应的输入部分。

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

以下示例使用用 PCRE 方言编写的模式来定位至少包含一个数字和一个小写字母的单词。它使用 ?= 运算符，它在 PCRE 中具有特定的前瞻含义。此示例返回与第二个此类单词相对应的输入部分，但与前面的示例不同，因为它使用了不区分大小写的匹配。

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
---------------
 A1234
```

以下示例使用子表达式，通过不区分大小写的匹配来查找与模式 'this is a (\\w+)' 匹配的第二个字符串。它返回括号内的子表达式。

```
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
```

REPEAT 函数

将字符串重复指定的次数。如果输入参数为数字，REPEAT 会将其视为字符串。

REPLICATE 函数的同义词。

语法

```
REPEAT(string, integer)
```

参数

string

第一个输入参数是要重复的字符串。

integer

第二个参数是指示字符串重复次数的整数。

返回类型

REPEAT 函数返回字符串。

示例

以下示例将重复 CATEGORY 表中 CATID 列的值三次：

```
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)
```

REPLACE 函数

将现有字符串中一组字符的所有匹配项替换为其他指定字符。

REPLACE 与 TRANSLATE 函数和 REGEXP_REPLACE 函数相似，只不过 TRANSLATE 进行多次单字符替换，REGEXP_REPLACE 可让您在字符串中搜索正则表达式模式，而 REPLACE 一次性将整个字符串替换为其他字符串。

语法

```
REPLACE(string1, old_chars, new_chars)
```

参数

string

要搜索的 CHAR 或 VARCHAR 字符串

old_chars

要替换的 CHAR 或 VARCHAR 字符串。

new_chars

用于替换 old_string 的新 CHAR 或 VARCHAR 字符串。

返回类型

VARCHAR

如果 old_chars 或 new_chars 为 NULL，则将返回 NULL。

示例

以下示例将 CATGROUP 字段中的字符串 Shows 转换为 Theatre：

```
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)
```

REPLICATE 函数

REPEAT 函数的同义词。

请参阅 REPEAT 函数。

REVERSE 函数

REVERSE 函数对字符串运行并以反向顺序返回字符。例如，`reverse('abcde')` 将返回 edcba。此函数适用于数字和日期数据类型以及字符数据类型；但在大多数情况下，它对于字符串具有实用价值。

语法

```
REVERSE ( expression )
```

参数

expression

一个表达式，带有表示字符反转目标的字符、日期、时间戳或数字数据类型。所有表达式均可隐式转换为可变长度的字符串。将忽略定宽字符串中的尾随空格。

返回类型

REVERSE 返回 VARCHAR。

示例

从 USERS 表中选择 5 个不同的城市名称及其对应的反转名称：

```
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)
```

选择 5 个销售 ID 及其对应的反转 ID（已强制转换为字符串）：

```
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)
```

RTRIM 函数

RTRIM 函数从字符串的末尾剪裁指定的一组字符。删除只包含剪裁字符列表中的字符的最长字符串。当输入字符串中没有了剪裁字符时，剪裁即告完成。

语法

```
RTRIM( string, trim_chars )
```

参数

string

要剪裁的字符串列、表达式或字符串文本。

trim_chars

表示要从 string 的结尾剪裁的字符的字符串列、表达式或字符串文本。如果未指定，则使用空格作为剪裁字符。

返回类型

与 string 参数具有相同的数据类型的字符串。

示例

以下示例从字符串 ' abc ' 中剪裁前导和尾随空格：

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

以下示例从字符串 'xyzaxyzbxyzcxyz' 中删除尾随字符串 'xyz'。将删除尾随的 'xyz'，但不会删除字符串内部的匹配字符。

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

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

以下示例从字符串 'setuphistorycassettes' 中删除与 trim_chars 列表 'tes' 中的任何字符相匹配的结尾部分。在输入字符串结尾部分，在 trim_chars 列表中未包含的其他字符之前出现的任何 t、e 或 s 都将被删除。

```
SELECT rtrim('setuphistorycassettes', 'tes');
```

```
     rtrim
-----------------
 setuphistoryca
```

以下示例从 VENUENAME 的末尾剪裁字符“Park”（如果有）：

```
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
```

请注意，当字符 P、a、r 或 k 中的任意一个出现在 VENUENAME 的末尾时，RTRIM 都会予以删除。

SOUNDEX 函数

SOUNDEX 函数返回美国 Soundex 值，其中包括第一个字母，后跟一个 3 位数字的声音编码，该编码表示您指定的字符串的英语发音。

语法

```
SOUNDEX(string)
```

参数

string

您可以指定要转换为美国 Soundex 代码值的 CHAR 或 VARCHAR 字符串。

返回类型

SOUNDEX 函数返回一个 VARCHAR(4) 字符串，其中包括一个大写字母，后跟代表英语发音的三位数字声音编码。

使用说明

SOUNDEX 函数仅转换英文字母小写或大写 ASCII 字符，包括 a–z 和 A–Z。SOUNDEX 将忽略其他字符。对于由空格分隔的多个单词组成的字符串，SOUNDEX 返回单个 Soundex 值。

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

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

如果输入字符串不包含任何英文字母，SOUNDEX 将返回一个空字符串。

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

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

```

示例

以下示例将返回单词 Amazon 的 Soundex A525。

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

```
 soundex
---------
 A525
```

SPLIT_PART 函数

用指定的分隔符拆分字符串，并返回指定位置的部分内容。

语法

```
SPLIT_PART(string, delimiter, position)
```

参数

string

要拆分的字符串列、表达式或字符串文本。字符串可以是 CHAR 或 VARCHAR。

分隔符

分隔符字符串指示输入 string 的部分。

如果 delimiter 是文本，则将其括在单引号中。

position

要返回的 string 部分的位置（从 1 算起）。必须是大于 0 的整数。如果 position 大于字符串部分的数量，SPLIT_PART 将返回空字符串。如果在字符串中未找到分隔符，则返回的值包含指定部分的内容，它可能是整个字符串或一个空值。

返回类型

CHAR 或 VARCHAR 字符串，与 string 参数相同。

示例

以下示例使用 $ 分隔符，将字符串文本拆分为多个部分，并返回第二部分。

```
select split_part('abc$def$ghi','$',2)

split_part
----------
def
```

以下示例使用 $ 分隔符，将字符串文本拆分为多个部分。它返回一个空字符串，因为找不到部分 4。

```
select split_part('abc$def$ghi','$',4)

split_part
----------
```

以下示例使用 # 分隔符，将字符串文本拆分为多个部分。它返回整个字符串，也就是第一部分，因为找不到分隔符。

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

以下示例将时间戳字段 LISTTIME 拆分为年、月和日组成部分。

```
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
```

以下示例选择 LISTTIME 时间戳字段并在 '-' 字符处拆分它以获取月（LISTTIME 字符串的第二部分），然后计算每个月的条目数：

```
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
```

STRPOS 函数

返回子字符串在指定字符串中的位置。

有关类似的函数，请参阅CHARINDEX 函数和POSITION 函数。

语法

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

参数

string

第一个输入参数是要在其中进行搜索的字符串。

substring

第二个参数是要在 string 中搜索的子字符串。

返回类型

STRPOS 函数返回与子字符串的位置对应的整数（从 1 开始，而不是从 0 开始）。此位置基于字符数而不是字节数，这是为了将多字节字符作为单字符计数。

使用说明

如果在 string 中未找到 substring，STRPOS 将返回 0：

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

示例

以下示例显示字符串 fish 在单词 dogfish 中的位置：

```
select strpos('dogfish', 'fish');
strpos
--------
4
(1 row)
```

以下示例返回 SALES 表中 COMMISSION 超过 999.00 的销售交易的数量：

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

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

SUBSTR 函数

SUBSTRING 函数的同义词。

请参阅 SUBSTRING 函数。

SUBSTRING 函数

按指定的开始位置返回子字符串子集。

如果输入的是字符串，字符的开始位置和数量基于字符数而不是字节数，这是为了将多字节字符作为单个字符计数。如果输入的是二进制表达式，则开始位置和提取的子字符串基于字节。您无法指定负长度，但可指定负开始位置。

语法

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

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

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

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

参数

character_string

要搜索的字符串。非字符数据类型将视为字符串。

start_position

字符串中开始提取的位置，从 1 开始。start_position 基于字符数而不是字节数，这是为了将多字节字符作为单个字符计数。此数字可以为负。

number_characters

要提取的字符的数量（子字符串的长度）。number_characters 基于字符数而不是字节数，这是为了将多字节字符作为单个字符计数。此数字不能为负。

start_byte

二进制表达式中开始提取的位置，从 1 开始。此数字可以为负。

number_bytes

要提取的字节的数量（子字符串的长度）。此数字不能为负。

返回类型

VARCHAR

字符串的使用说明

以下示例返回以第 6 个字符开头的 4 字符字符串。

```
select substring('caterpillar',6,4);
substring
-----------
pill
(1 row)
```

如果 start_position + number_characters 超过 string 的长度，SUBSTRING 将返回从 start_position 开始到此字符串末尾的子字符串。例如：

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

如果 start_position 为负或 0，SUBSTRING 函数将返回从长度为 start_position + number_characters -1 的字符串的第一个字符开始的子字符串。例如：

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

如果 start_position + number_characters -1 小于或等于零，SUBSTRING 将返回空字符串。例如：

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

示例

以下示例返回 LISTING 表的 LISTTIME 字符串中的月份：

```
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)
```

以下示例与上述示例相同，但使用 FROM...FOR 选项：

```
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)
```

您无法使用 SUBSTRING 以可预测的方式提取可能包含多字节字符的字符串的前缀，因为您需要根据字节数（而不是字符数）指定多字节字符串的长度。要基于以字节为单位的长度提取字符串的开始部分，您可将字符串强制转换为 VARCHAR(byte_length) 以截断字符串，其中 byte_length 是必需长度。以下示例提取字符串 'Fourscore and seven' 的前 5 个字节。

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
Fours
```

以下示例返回出现在输入字符串 Silva, Ana 中最后一个空格之后的名字 Ana。

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))

 reverse
-----------
 Ana
```

TEXTLEN 函数

LEN 函数的同义词。

请参阅 LEN 函数。

TRANSLATE 函数

对于给定表达式，将指定字符的所有匹配项替换为指定替代项。现有字符将按其在 characters_to_replace 和 characters_to_substitute 参数中的位置映射到替换字符。如果在 characters_to_replace 参数中指定的字符多于在 characters_to_substitute 参数中指定的字符，返回值中将省略 characters_to_replace 参数中的额外字符。

TRANSLATE 与 REPLACE 函数和 REGEXP_REPLACE 函数相似，只不过 REPLACE 将整个字符串替换为其他字符串，REGEXP_REPLACE 可让您在字符串中搜索正则表达式模式，而 TRANSLATE 进行多次单字符替换。

如果任何参数为空，则返回 NULL。

语法

```
TRANSLATE ( expression, characters_to_replace, characters_to_substitute )
```

参数

expression

要转换的表达式。

characters_to_replace

一个包含要替换的字符的字符串。

characters_to_substitute

一个字符串，其中包含要替换其他字符的字符。

返回类型

VARCHAR

示例

以下示例将替换字符串中的多个字符：

```
select translate('mint tea', 'inea', 'osin');

translate
-----------
most tin
```

以下示例将列中的所有值的 at (@) 符号替换为句点：

```
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
```

以下示例将空格替换为下划线并去掉列中的所有值的句点：

```
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
```

TRIM 函数

通过删除前导空格和尾随空格或删除与可选的指定字符串匹配的前导字符和尾随字符来剪裁字符串。

语法

```
TRIM( [ BOTH ] [ trim_chars FROM ] string
```

参数

trim_chars

（可选）要从字符串剪裁的字符数。如果忽略此参数，则剪裁空白区域。

string

要剪裁的字符串。

返回类型

TRIM 函数返回 VARCHAR 或 CHAR 字符串。如果您将 TRIM 函数与 SQL 命令一起使用，则会将结果 AWS Clean Rooms 隐式转换为 VARCHAR。如果您将 SELECT 列表中的 TRIM 函数用于 SQL 函数，则 AWS Clean Rooms 不会隐式转换结果，并且可能需要执行显式转换以避免出现数据类型不匹配错误。有关显式转换的信息，请参阅 CAST 函数和 CONVERT 函数 函数。

示例

以下示例从字符串 ' abc ' 中剪裁前导和尾随空格：

```
select '     abc    ' as untrim, trim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    | abc
```

以下示例将删除将字符串 "dog" 括起的双引号：

```
select trim('"' FROM '"dog"');

btrim
-------
dog
```

当 trim_chars 中的任意字符出现在 string 的开头时，TRIM 都会予以删除。以下示例从 VENUENAME（VARCHAR 列）的开头剪裁字符“C”、“D”和“G”。

```
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
```

UPPER 函数

将字符串转换为大写。UPPER 支持 UTF-8 多字节字符，并且每个字符最多可以有 4 个字节。

语法

```
UPPER(string)
```

参数

string

输入参数是 VARCHAR 字符串（或任何其他可隐式转换为 VARCHAR 的数据类型，如 CHAR）。

返回类型

UPPER 函数返回与输入字符串具有相同数据类型的字符串。

示例

以下示例将 CATNAME 字段转换为大写：

```
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)
```

SUPER 类型信息函数

本节介绍 SQL 用于从 AWS Clean Rooms 支持的 SUPER 数据类型输入中派生动态信息的信息函数。

主题

DECIMAL_PRECISION 函数

检查要存储的最大小数位数总数的精度。此数字包括小数点的左侧和右侧数字。精度范围为 1 到 38，默认值为 38。

语法

```
DECIMAL_PRECISION(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

INTEGER

示例

要将 DECIMAL_PRECISION 函数应用于表 t，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_PRECISION(s) FROM t;

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

DECIMAL_SCALE 函数

检查要存储在小数点右侧的小数位数。小数位数范围从 0 到精度点，默认值为 0。

语法

```
DECIMAL_SCALE(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

INTEGER

示例

要将 DECIMAL_SCALE 函数应用于表 t，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_SCALE(s) FROM t;

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

IS_ARRAY 函数

检查变量是否为数组。如果变量是数组，则函数返回 `true`。该函数还包括空数组。否则，对于所有其他值，包括 null，函数返回 `false`。

语法

```
IS_ARRAY(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_ARRAY 函数检查 `[1,2]` 是否为数组，请使用以下示例。

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

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

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

IS_BIGINT 函数

检查某个值是否为 BIGINT。对于 64 位范围内的小数位数 0 的数量，IS_BIGINT 函数将返回 `true`。否则，对于所有其他值，包括 null 和浮点数，该函数将返回 `false`。

IS_BIGINT 函数是 IS_INTEGER 的超集。

语法

```
IS_BIGINT(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_BIGINT 函数检查 5 是否为 BIGINT，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_BIGINT(s) FROM t;

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

IS_CHAR 函数

检查某个值是否为 CHAR。对于仅包含 ASCII 字符的字符串，IS_CHAR 函数返回 true，因为 CHAR 类型只能存储 ASCII 格式的字符。对于任何其他值，该函数返回 false。

语法

```
IS_CHAR(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_CHAR 函数检查 t 是否为 CHAR，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('t');

SELECT s, IS_CHAR(s) FROM t;

+-----+---------+
|  s  | is_char |
+-----+---------+
| "t" | true    |
+-----+---------+
```

IS_DECIMAL 函数

检查某个值是否为 DECIMAL。对于非浮点的数值，IS_DECIMAL 函数返回 true。对于任何其他值，包括 null，该函数返回 false。

IS_DECIMAL 函数是 IS_BIGINT 的超集。

语法

```
IS_DECIMAL(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_DECIMAL 函数检查 1.22 是否为 DECIMAL，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (1.22);

SELECT s, IS_DECIMAL(s) FROM t;

+------+------------+
|  s   | is_decimal |
+------+------------+
| 1.22 | true       |
+------+------------+
```

IS_FLOAT 函数

检查值是否为浮点数。对于浮点数（FLOAT4 和 FLOAT8），IS_FLOAT 函数返回 true。对于任何其他值，该函数返回 false。

IS_DECIMAL 集和 IS_FLOAT 集是不相交的。

语法

```
IS_FLOAT(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_FLOAT 函数检查 2.22::FLOAT 是否为 FLOAT，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES(2.22::FLOAT);

SELECT s, IS_FLOAT(s) FROM t;

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

IS_INTEGER 函数

对于 32 位范围内的小数位数 0 的数量，返回 true；对于其他任何值（包括 null 和浮点数），则返回 false。

IS_INTEGER 函数是 IS_SMALLINT 函数的超集。

语法

```
IS_INTEGER(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_INTEGER 函数检查 5 是否为 INTEGER，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_INTEGER(s) FROM t;

+---+------------+
| s | is_integer |
+---+------------+
| 5 | true       |
+---+------------+
```

IS_OBJECT 函数

检查变量是否为对象。对于包括空对象在内的对象，IS_OBJECT 函数返回 `true`。对于任何其他值，包括 null，该函数返回 `false`。

语法

```
IS_OBJECT(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_OBJECT 函数检查 {"name": "Joe"} 是否为对象，请使用以下示例。

```
CREATE TABLE t(s super);

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

SELECT s, IS_OBJECT(s) FROM t;

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

IS_SCALAR 函数

检查变量是否为标量。对于非数组或对象的任何值，IS_SCALAR 函数返回 `true`。对于任何其他值，包括 null，该函数返回 `false`。

IS_ARRAY、IS_OBJECT 和 IS_SCALAR 的集合覆盖除 null 之外的所有值。

语法

```
IS_SCALAR(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_SCALAR 函数检查 {"name": "Joe"} 是否为标量，请使用以下示例。

```
CREATE TABLE t(s SUPER);

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

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

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

IS_SMALLINT 函数

检查变量是否为 SMALLINT。对于 16 位范围内的小数位数 0 的数量，IS_SMALLINT 函数返回 `true`。对于任何其他值，包括 null 和浮点数，该函数返回 `false`。

语法

```
IS_SMALLINT(super_expression)
```

参数

super_expression

SUPER 表达式或列。

Return

BOOLEAN

示例

要使用 IS_SMALLINT 函数检查 5 是否为 SMALLINT，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

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

```
+---+-------------+
| s | is_smallint |
+---+-------------+
| 5 | true        |
+---+-------------+
```

IS_VARCHAR 函数

检查变量是否为 VARCHAR。对于所有字符串，IS_VARCHAR 函数返回 `true`。对于任何其他值，该函数返回 `false`。

IS_VARCHAR 函数是 IS_CHAR 函数的超集。

语法

```
IS_VARCHAR(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

BOOLEAN

示例

要使用 IS_VARCHAR 函数检查 abc 是否为 VARCHAR，请使用以下示例。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('abc');

SELECT s, IS_VARCHAR(s) FROM t;

+-------+------------+
|   s   | is_varchar |
```

```
+-------+------------+
| "abc" | true       |
+-------+------------+
```

JSON_TYPEOF 函数

根据 SUPER 值的动态类型，JSON_TYPEOF 标量函数返回具有布尔值、数值、字符串、对象、数组或 null 的 VARCHAR。

语法

```
JSON_TYPEOF(super_expression)
```

参数

super_expression

SUPER 表达式或列。

返回类型

VARCHAR

示例

要使用 JSON_TYPEOF 函数检查数组 [1,2] 的 JSON 类型，请使用以下示例。

```
SELECT JSON_TYPEOF(ARRAY(1,2));

+-------------+
| json_typeof |
+-------------+
| array       |
+-------------+
```

VARBYTE 函数

AWS Clean Rooms 支持以下 VARBYTE 函数。

主题

FROM_HEX 函数

FROM_HEX 将十六进制转换为二进制值。

语法

```
FROM_HEX(hex_string)
```

参数

hex_string

要转换的数据类型为 VARCHAR 或 TEXT 的十六进制字符串。格式必须是文本值。

返回类型

VARBYTE

示例

要将 '6162' 的十六进制表示形式转换为二进制值，请使用以下示例。结果会自动显示为二进制值的十六进制表示形式。

```
SELECT FROM_HEX('6162');

+----------+
| from_hex |
+----------+
|     6162 |
+----------+
```

FROM_VARBYTE 函数

FROM_VARBYTE 将二进制值转换为指定格式的字符串。

语法

```
FROM_VARBYTE(binary_value, format)
```

参数

binary_value

数据类型为 VARBYTE 的二进制值。

格式的日期和时间。

返回的字符串格式。不区分大小写的有效值包括 hex、binary、utf-8 和 utf8。

返回类型

VARCHAR

示例

要将二进制值 'ab' 转换为十六进制，请使用以下示例。

```
SELECT FROM_VARBYTE('ab', 'hex');

+--------------+
| from_varbyte |
+--------------+
|         6162 |
+--------------+
```

TO_HEX 函数

TO_HEX 将数字或二进制值转换为十六进制表示形式。

语法

```
TO_HEX(value)
```

参数

值

要转换的数值或二进制值（VARBYTE）。

返回类型

VARCHAR

示例

要将一个数值转换为其十六进制表示形式，请使用以下示例。

```
SELECT TO_HEX(2147676847);

+----------+
|  to_hex  |
+----------+
| 8002f2af |
+----------+To create a table, insert the VARBYTE representation of 'abc' to a
 hexadecimal number, and select the column with the value, use the following example.
```

TO_VARBYTE 函数

TO_VARBYTE 将指定格式的字符串转换为二进制值。

语法

```
TO_VARBYTE(string, format)
```

参数

string

CHAR 或 VARCHAR 字符串。

格式的日期和时间。

输入字符串的格式。不区分大小写的有效值包括 hex、binary、utf-8 和 utf8。

返回类型

VARBYTE

示例

要将十六进制 6162 转换为二进制值，请使用以下示例。结果会自动显示为二进制值的十六进制表示形式。

```
SELECT TO_VARBYTE('6162', 'hex');

+------------+
| to_varbyte |
+------------+
|       6162 |
+------------+
```

窗口函数

通过使用窗口函数，您可以更高效地创建分析业务查询。窗口函数运行于分区或结果集的“窗口”上，并为该窗口中的每个行返回一个值。相比之下，非窗口函数执行与结果集中的每个行相关的计算。与聚合结果行的分组函数不同，窗口函数在表的表达式中的保留所有行。

使用该窗口中的行集中的值计算返回的值。对于表中的每一行，窗口定义一组用于计算其他属性的行。窗口使用窗口规范（OVER 子句）进行定义并基于以下三个主要概念：

- 窗口分区，构成了行组（PARTITION 子句）
- 窗口排序，定义了每个分区中行的顺序或序列（ORDER BY 子句）
- 窗口框架，相对于每个行进行定义以进一步限制行集（ROWS 规范）

窗口函数是在查询中执行的最后一组操作（最后的 ORDER BY 子句除外）。所有联接和所有 WHERE、GROUP BY 和 HAVING 子句均在处理窗口函数前完成。因此，窗口函数只能显示在选择列表或 ORDER BY 子句中。您可以在一个具有不同框架子句的查询中使用多个窗口函数。您还可以在其他标量表达式（如 CASE）中使用窗口函数。

窗口函数语法摘要

窗口函数遵循标准语法，如下所示。

```
function (expression) OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list [ frame_clause ] ] )
```

其中，function 是本部分介绍的函数之一。

expr_list 如下所示。

```
expression | column_name [, expr_list ]
```

order_list 如下所示。

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

frame_clause 如下所示。

```
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 }}
```

参数

函数

窗口函数。有关详细信息，请参阅各个函数描述。

OVER

定义窗口规范的子句。OVER 子句是窗口函数必需的，并可区分窗口函数与其他 SQL 函数。

PARTITION BY expr_list

（可选）PARTITION BY 子句将结果集细分为分区，与 GROUP BY 子句很类似。如果存在分区子句，则为每个分区中的行计算该函数。如果未指定任何分区子句，则一个分区包含整个表，并为整个表计算该函数。

排名函数 DENSE_RANK、NTILE、RANK 和 ROW_NUMBER 需要全局比较结果集中的所有行。使用 PARTITION BY 子句时，查询优化程序可通过根据分区跨多个切片分布工作负载来并行运行每个聚合。如果不存在 PARTITION BY 子句，则必须在一个切片上按顺序运行聚合步骤，这可能对性能产生显著的负面影响，特别是对于大型集群。

AWS Clean Rooms 不支持 PARTITION BY 子句中的字符串文字。

ORDER BY order_list

（可选）窗口函数将应用于每个分区中根据 ORDER BY 中的顺序规范排序的行。此 ORDER BY 子句与 frame_clause 中的 ORDER BY 子句不同且完全不相关。ORDER BY 子句可在没有 PARTITION BY 子句的情况下使用。

对于排名函数，ORDER BY 子句确定排名值的度量。对于聚合函数，分区的行必须在为每个框架计算聚合函数之前进行排序。有关窗口函数的更多信息，请参阅 窗口函数。

顺序列表中需要列标识符或计算结果为列标识符的表达式。常数和常数表达式都不可用作列名称的替代。

NULLS 值将被视为其自己的组，并根据 NULLS FIRST 或 NULLS LAST 选项进行排序和排名。默认情况下，按 ASC 顺序最后对 NULL 值进行排序和排名，按 DESC 顺序首先对 NULL 值进行排序和排名。

AWS Clean Rooms 不支持 ORDER BY 子句中的字符串文字。

如果省略 ORDER BY 子句，则行的顺序是不确定的。

> **Note**
>
> 在任何并行系统中 AWS Clean Rooms，例如，当 ORDER BY 子句不生成数据的唯一和总体顺序时，行的顺序是不确定的。也就是说，如果 ORDER BY 表达式生成重复的值（部分排序），则这些行的返回顺序可能因运行而异。 AWS Clean Rooms 反过来，窗口函数可能返回意外的或不一致的结果。有关更多信息，请参阅窗口函数的唯一数据排序。

column_name

执行分区或排序操作所依据的列的名称。

ASC | DESC

一个定义表达式的排序顺序的选项，如下所示：

- ASC：升序（例如，按数值的从低到高的顺序和字符串的从 A 到 Z 的顺序）。如果未指定选项，则默认情况下将按升序对数据进行排序。
- DESC：降序（按数值的从高到低的顺序和字符串的从 Z 到 A 的顺序）。

NULLS FIRST | NULLS LAST

指定是应首先对 NULL 值进行排序（非 null 值之前）还是最后对 NULL 值进行排序（非 null 值之后）的选项。默认情况下，按 ASC 顺序最后对 NULLS 进行排序和排名，按 DESC 顺序首先对 NULLS 进行排序和排名。

frame_clause

对于聚合函数，框架子句在使用 ORDER BY 时进一步优化函数窗口中的行集。它使您可以包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。

frame 子句不适用于排名函数。同时，在聚合函数的 OVER 子句中未使用 ORDER BY 子句时不需要框架子句。如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。

未指定 ORDER BY 子句时，隐式框架是无界的：等同于 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING。

ROWS

此子句通过从当前行中指定物理偏移来定义窗口框架。

此子句指定当前行中的值将并入的当前窗口或分区中的行。它使用指定行位置的参数，行位置可位于当前行之前或之后。所有窗口框架的参考点为当前行。当窗口框架向前滑向分区中时，每个行会依次成为当前行。

框架可以是一组超过并包括当前行的行。

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

或者可以是两个边界之间的一组行。

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING 指示窗口从分区的第一行开始；offset PRECEDING 指示窗口开始于等同于当前行之前的偏移值的行数。UNBOUNDED PRECEDING 是默认值。

CURRENT ROW 指示窗口在当前行开始或结束。

UNBOUNDED FOLLOWING 指示窗口在分区的最后一行结束；offset FOLLOWING 指示窗口结束于等同于当前行之后的偏移值的行数。

offset 标识当前行之前或之后的实际行数。在这种情况下，offset 必须为计算结果为正数值的常数。例如，5 FOLLOWING 将在当前行之后的第 5 行结束框架。

其中，未指定 BETWEEN，框架受当前行隐式限制。例如，ROWS 5 PRECEDING 等于 ROWS BETWEEN 5 PRECEDING AND CURRENT ROW。同时，ROWS UNBOUNDED FOLLOWING 等于 ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING。

> **Note**
>
> 您无法指定起始边界大于结束边界的框架。例如，您无法指定以下任一框架。
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

窗口函数的唯一数据排序

如果窗口函数的 ORDER BY 子句不生成数据的唯一排序和总排序，则行的顺序是不确定的。如果 ORDER BY 表达式生成重复的值（部分排序），则这些行的返回顺序可能会在多次运行中有所不同。在这种情况下，窗口函数还可能返回意外的或不一致的结果。

例如，以下查询在多次运行中返回了不同的结果。出现这些不同的结果是因为 order by dateid 未生成 SUM 窗口函数的数据的唯一排序。

```
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
...
```

在这种情况下，向该窗口函数添加另一个 ORDER BY 列可解决此问题。

```
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
...
```

支持的函数

AWS Clean Rooms 支持两种类型的窗口函数：聚合和排名。

以下是支持的聚合函数：

- AVG 窗口函数
- COUNT 窗口函数
- CUME_DIST 开窗函数
- DENSE_RANK 窗口函数
- FIRST_VALUE 窗口函数
- LAG 窗口函数

- LAST_VALUE 窗口函数
- LEAD 窗口函数
- LISTAGG 窗口函数
- MAX 窗口函数
- MEDIAN 开窗函数
- MIN 窗口函数
- NTH_VALUE 窗口函数
- PERCENTILE_CONT 开窗函数
- PERCENTILE_DISC 开窗函数
- RATIO_TO_REPORT 开窗函数
- STDDEV_SAMP 和 STDDEV_POP 窗口函数（STDDEV_SAMP 和 STDDEV 是同义词）
- SUM 窗口函数
- VAR_SAMP 和 VAR_POP 窗口函数（VAR_SAMP 和 VARIANCE 是同义词）

以下是支持的排名函数：

- DENSE_RANK 窗口函数
- NTILE 窗口函数
- PERCENT_RANK 开窗函数
- RANK 窗口函数
- ROW_NUMBER 窗口函数

窗口函数示例的示例表

您可以通过每个函数描述找到特定的窗口函数示例。其中一些示例使用一个名为 WINSALES 的表，该表包含 11 行，如下表所示。

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
30001	8/2/2003	3	B	10	10

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
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	

AVG 窗口函数

AVG 窗口函数返回输入表达式值的平均值（算术平均值）。AVG 函数使用数值并忽略 NULL 值。

语法

```
AVG ( [ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值以进行计数。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 AVG 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

AVG 函数支持的参数类型为 SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 和 DOUBLE PRECISION。

AVG 函数支持的返回类型为：

- 适用于 SMALLINT 或 INTEGER 参数的 BIGINT
- 适用于 BIGINT 参数的 NUMERIC
- 适用于浮点参数的 DOUBLE PRECISION

示例

以下示例按日期计算销量的移动平均数；按日期 ID 和销售 ID 对结果进行排序：

```
select salesid, dateid, sellerid, qty,
avg(qty) over
(order by dateid, salesid rows unbounded preceding) as avg
```

```
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | avg
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
10005 | 2003-12-24 |        1 |  30 |  16
40001 | 2004-01-09 |        4 |  40 |  22
10006 | 2004-01-18 |        1 |  10 |  20
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  18
20002 | 2004-02-16 |        2 |  20 |  18
30003 | 2004-04-18 |        3 |  15 |  18
30004 | 2004-04-18 |        3 |  20 |  18
30007 | 2004-09-07 |        3 |  30 |  19
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

COUNT 窗口函数

COUNT 窗口函数对由表达式定义的行计数。

COUNT 函数具有两个变体。COUNT(*) 对目标表中的所有行计数，无论它们是否包含 null 值。COUNT(expression) 计算某个特定列或表达式中带非 NULL 值的行的数量。

语法

```
COUNT ( * | [ ALL ] expression) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值以进行计数。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 COUNT 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

COUNT 函数支持所有参数数据类型。

COUNT 函数支持的返回类型是 BIGINT。

示例

以下示例从数据窗口的开头显示销售 ID、数量和所有行的计数：

```
select salesid, qty,
count(*) over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | count
---------+-----+-----
10001 |  10 |   1
10005 |  30 |   2
10006 |  10 |   3
20001 |  20 |   4
20002 |  20 |   5
```

```
30001 |  10 |   6
30003 |  15 |   7
30004 |  20 |   8
30007 |  30 |   9
40001 |  40 |   10
40005 |  10 |   11
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例从数据窗口的开头显示销售 ID、数量和非 null 行的计数。（在 WINSALES 表中，QTY_SHIPPED 列包含一些 NULL 值。）

```
select salesid, qty, qty_shipped,
count(qty_shipped)
over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | qty_shipped | count
---------+-----+-------------+-------
10001 |  10 |          10 |   1
10005 |  30 |             |   1
10006 |  10 |             |   1
20001 |  20 |          20 |   2
20002 |  20 |          20 |   3
30001 |  10 |          10 |   4
30003 |  15 |             |   4
30004 |  20 |             |   4
30007 |  30 |             |   4
40001 |  40 |             |   4
40005 |  10 |          10 |   5
(11 rows)
```

CUME_DIST 开窗函数

计算某个窗口或分区中某个值的累积分布。假定升序排序，则使用以下公式确定累积分布：

```
count of rows with values <= x / count of rows in the window or partition
```

其中，x 等于 ORDER BY 子句中指定的列的当前行中的值。以下数据集说明了此公式的使用：

```
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
```

返回值范围介于 0 和 1（含 1）之间。

语法

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

参数

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口框架规范。

PARTITION BY partition_expression

可选。一个设置 OVER 子句中每个组的记录范围的表达式。

ORDER BY order_list

用于计算累积分布的表达式。该表达式必须具有数字数据类型或可隐式转换为 1。如果省略 ORDER BY，则所有行的返回值为 1。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅窗口函数的唯一数据排序。

返回类型

FLOAT8

示例

以下示例计算每个卖家的销量的累积分布：

```
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
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

DENSE_RANK 窗口函数

DENSE_RANK 窗口函数基于 OVER 子句中的 ORDER BY 表达式确定一组值中的一个值的排名。如果存在可选的 PARTITION BY 子句，则为每个行组重置排名。带符合排名标准的相同值的行接收相同的排名。DENSE_RANK 函数与 RANK 存在以下一点不同：如果两个或两个以上的行结合，则一系列排名的值之间没有间隔。例如，如果两个行的排名为 1，则下一个排名则为 2。

您可以在同一查询中包含带有不同的 PARTITION BY 和 ORDER BY 子句的排名函数。

语法

```
DENSE_RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

参数

( )

该函数没有参数，但需要空括号。

OVER

适用于 DENSE_RANK 函数的窗口子句。

PARTITION BY expr_list

可选。一个或多个定义窗口的表达式。

ORDER BY order_list

可选。排名值基于的表达式。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。如果省略 ORDER BY，则所有行的返回值为 1。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅窗口函数的唯一数据排序。

返回类型

INTEGER

示例

以下示例按销量对表进行排序（按降序顺序），并将紧密排名和常规排名分配给每个行。在应用窗口函数结果后，对结果进行排序。

```
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)
```

在同一查询中一起使用 DENSE_RANK 和 RANK 函数时，记下已分配给同一组行的排名的差异。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例按 SELLERID 对表进行分区，按数量对每个分区进行排序（按降序顺序），并为每个行分配紧密排名。在应用窗口函数结果后，对结果进行排序。

```
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)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

FIRST_VALUE 窗口函数

在提供一组已排序行的情况下，FIRST_VALUE 返回有关窗口框架中的第一行的指定表达式的值。

有关选择框架中最后一行的信息，请参阅 LAST_VALUE 窗口函数。

语法

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
```

```
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

IGNORE NULLS

将此选项与 FIRST_VALUE 结合使用时，该函数返回不为 NULL 的框架中的第一个值（如果所有值为 NULL，则返回 NULL）。

RESPECT NULLS

表示在确定 AWS Clean Rooms 要使用哪一行时应包含空值。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

引入函数的窗口子句。

PARTITION BY expr_list

依据一个或多个表达式定义函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY 子句，则 ORDER BY 对整个表进行排序。如果指定 ORDER BY 子句，则还必须指定 frame_clause。

FIRST_VALUE 函数的结果取决于数据的排序。在以下情况下，结果是不确定的：

- 当未指定 ORDER BY 子句且一个分区包含一个表达式的两个不同的值时
- 当表达式的计算结果为对应于 ORDER BY 列表中同一值的不同值时。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

返回类型

这些函数支持使用原始 AWS Clean Rooms 数据类型的表达式。返回类型与 expression 的数据类型相同。

示例

以下示例返回 VENUE 表中每个场地的座位数，同时按容量对结果进行排序（从高到低）。FIRST_VALUE 函数用于选择与框架中的第一行对应的场地的名称：在这种情况下，为座位数最多的行。按州对结果进行分区，以便当 VENUESTATE 值发生更改时，会选择一个新的第一个值。窗口框架是无界的，因此为每个分区中的每个行选择相同的第一个值。

对于加利福利亚，Qualcomm Stadium 具有最大座位数 (70561)，此名称是 CA 分区中所有行的第一个值。

```
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
...
```

LAG 窗口函数

LAG 窗口函数返回位于分区中当前行的上方（之前）的某个给定偏移量位置的行的值。

语法

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

参数

value_expr

对其执行函数的目标列或表达式。

offset

一个可选参数，该参数指定要返回其值的当前行前面的行数。偏移量可以是常量整数或计算结果为整数的表达式。如果未指定偏移量，则 AWS Clean Rooms 使用1作为默认值。偏移量为 0 表示当前行。

IGNORE NULLS

一种可选规范，用于指示在确定要使用哪一行时 AWS Clean Rooms 应跳过空值。如果未列出 IGNORE NULLS，则包含 Null 值。

> **Note**
>
> 您可以使用 NVL 或 COALESCE 表达式将 null 值替换为另一个值。

RESPECT NULLS

表示在确定 AWS Clean Rooms 要使用哪一行时应包含空值。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

指定窗口分区和排序。OVER 子句不能包含窗口框架规范。

PARTITION BY window_partition

一个可选参数，该参数设置 OVER 子句中每个组的记录范围。

ORDER BY window_ordering

对每个分区中的行进行排序。

LAG 窗口函数支持使用任何 AWS Clean Rooms 数据类型的表达式。返回类型与 value_expr 的类型相同。

示例

以下示例显示已售给买家 ID 为 3 的买家的票数以及买家 3 的购票时间。要将每个销售与买家 3 的上一销售进行比较，查询要返回每个销售的上一销量。由于 1/16/2008 之前未进行购买，则第一个上一销量值为 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)
```

LAST_VALUE 窗口函数

在提供一组已排序行的情况下，LAST_VALUE 函数返回有关框架中最后一行的表达式的值。

有关选择框架中第一行的信息，请参阅 FIRST_VALUE 窗口函数。

语法

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
```

```
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

IGNORE NULLS

该函数返回不为 NULL 的框架中的最后一个值（如果所有值为 NULL，则返回 NULL）。

RESPECT NULLS

表示在确定 AWS Clean Rooms 要使用哪一行时应包含空值。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

引入函数的窗口子句。

PARTITION BY expr_list

依据一个或多个表达式定义函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY 子句，则 ORDER BY 对整个表进行排序。如果指定 ORDER BY 子句，则还必须指定 frame_clause。

结果取决于数据的排序。在以下情况下，结果是不确定的：

- 当未指定 ORDER BY 子句且一个分区包含一个表达式的两个不同的值时
- 当表达式的计算结果为对应于 ORDER BY 列表中同一值的不同值时。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

返回类型

这些函数支持使用原始 AWS Clean Rooms 数据类型的表达式。返回类型与 expression 的数据类型相同。

示例

以下示例返回 VENUE 表中每个场地的座位数，同时按容量对结果进行排序（从高到低）。LAST_VALUE 函数用于选择与框架中的最后一行对应的场地的名称：在本例中，为座位数最少的行。按州对结果进行分区，以便当 VENUESTATE 值发生更改时，会选择一个新的最后一个值。窗口框架是无界的，因此为每个分区中的每个行选择相同的最后一个值。

对于加利福利亚，为该分区中的每个行返回 Shoreline Amphitheatre，因为它具有最小座位数 (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
...
```

LEAD 窗口函数

LEAD 窗口函数返回位于分区中当前行的下方（之后）的某个给定偏移量位置的行的值。

语法

```
LEAD (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

参数

value_expr

对其执行函数的目标列或表达式。

offset

一个可选参数，该参数指定要返回其值的当前行后面的行数。偏移量可以是常量整数或计算结果为整数的表达式。如果未指定偏移量，则 AWS Clean Rooms 使用1作为默认值。偏移量为 0 表示当前行。

IGNORE NULLS

一种可选规范，用于指示在确定要使用哪一行时 AWS Clean Rooms 应跳过空值。如果未列出 IGNORE NULLS，则包含 Null 值。

> **Note**
>
> 您可以使用 NVL 或 COALESCE 表达式将 null 值替换为另一个值。

RESPECT NULLS

表示在确定 AWS Clean Rooms 要使用哪一行时应包含空值。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

指定窗口分区和排序。OVER 子句不能包含窗口框架规范。

PARTITION BY window_partition

一个可选参数，该参数设置 OVER 子句中每个组的记录范围。

ORDER BY window_ordering

对每个分区中的行进行排序。

LEAD 窗口函数支持使用任何 AWS Clean Rooms 数据类型的表达式。返回类型与 value_expr 的类型相同。

示例

以下示例提供了 SALES 表中于 2008 年 1 月 1 日与 1 月 2 日已售票的事件的佣金以及为后续销售中售票所付的佣金。

```
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)
```

LISTAGG 窗口函数

对于查询中的每个组，LISTAGG 窗口函数根据 ORDER BY 表达式对该组的行进行排序，然后将值串联成一个字符串。

LISTAGG 是仅计算节点函数。如果查询未引用用户定义的表或 AWS Clean Rooms 系统表，则该函数将返回错误。

语法

```
LISTAGG( [DISTINCT] expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
OVER ( [PARTITION BY partition_expression] )
```

参数

DISTINCT

（可选）用于在串联之前消除指定表达式中重复值的子句。尾部空格将被忽略，因此会将字符串 `'a'` 和 `'a '` 视为重复值。LISTAGG 将使用遇到的第一个值。有关更多信息，请参阅尾部空格的意义。

aggregate_expression

提供要聚合的值的任何有效表达式（如列名称）。忽略 NULL 值和空字符串。

分隔符

(可选) 用于分隔串联的值的字符串常数。默认值为 NULL。

AWS Clean Rooms 支持在可选的逗号或冒号周围使用任意数量的前导或尾随空格，以及空字符串或任意数量的空格。

有效值示例为：

`", "`

`": "`

`" "`

WITHIN GROUP (ORDER BY order_list)

（可选）用于指定聚合值的排序顺序的子句。仅在 ORDER BY 提供唯一排序时是确定性的。默认为聚合所有行并返回一个值。

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY partition_expression

（可选）设置 OVER 子句中每个组的记录范围。

返回值

VARCHAR(MAX)。如果结果集大于最大 VARCHAR 大小（64K – 1 或 65535），则 LISTAGG 返回以下错误：

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

示例

以下示例使用 WINSALES 表。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例返回卖家 ID 的列表（按卖家 ID 排序）。

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

  listagg
------------
 11122333344
...
...
 11122333344
 11122333344
   (11 rows)
```

以下示例返回买家 B 的卖家 ID 的列表（按日期排序）。

```
select listagg(sellerid)
within group (order by dateid)
over () as seller
from winsales
where buyerid = 'b' ;
```

```
  seller
---------
    3233
    3233
    3233
    3233

(4 rows)
```

以下示例返回买家 B 的销售日期的逗号分隔的列表。

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

             dates
-------------------------------------------
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12

(4 rows)
```

以下示例使用 DISTINCT 返回买家 B 的唯一销售日期的列表。

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

           dates
-------------------------------
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12

(4 rows)
```

以下示例返回每个买家 ID 的销售 ID 的逗号分隔的列表。

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

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

MAX 窗口函数

MAX 窗口函数返回最大输入表达式值。MAX 函数使用数值并忽略 NULL 值。

语法

```
MAX ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

一个指定聚合函数的窗口子句的子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 MAX 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

接受任何数据类型作为输入。返回与 expression 相同的数据类型。

示例

以下示例从数据窗口的开头显示销售 ID、数量和最大数量：

```
select salesid, qty,
max(qty) over (order by salesid rows unbounded preceding) as max
from winsales
order by salesid;

salesid | qty | max
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  30
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  30
30001 |  10 |  30
30003 |  15 |  30
```

```
30004 |  20 |  30
30007 |  30 |  30
40001 |  40 |  40
40005 |  10 |  40
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例在受限制的框架中显示销售 ID、数量和最大数量：

```
select salesid, qty,
max(qty) over (order by salesid rows between 2 preceding and 1 preceding) as max
from winsales
order by salesid;

salesid | qty | max
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  20
30001 |  10 |  20
30003 |  15 |  20
30004 |  20 |  15
30007 |  30 |  20
40001 |  40 |  30
40005 |  10 |  40
(11 rows)
```

MEDIAN 开窗函数

计算某个窗口或分区中值的范围的中间值。忽略该范围中的 NULL 值。

MEDIAN 是一种假定连续分布模型的逆分布函数。

MEDIAN 是仅计算节点函数。如果查询未引用用户定义的表或 AWS Clean Rooms 系统表，则该函数将返回错误。

语法

```
MEDIAN ( median_expression )
```

```
OVER ( [ PARTITION BY partition_expression ] )
```

参数

median_expression

一个提供要为其确定中间值的值的表达式（例如列名）。该表达式必须具有数字数据类型或日期时间数据类型或可隐式转换为 1。

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY partition_expression

可选。一个设置 OVER 子句中每个组的记录范围的表达式。

数据类型

返回类型由 median_expression 的数据类型确定。下表显示了每种 median_expression 数据类型的返回类型。

输入类型	返回类型
NUMERIC、DECIMAL	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE

使用说明

如果 median_expression 参数是使用 38 位最大精度定义的 DECIMAL 数据类型，则 MEDIAN 可能将返回不准确的结果或错误。如果 MEDIAN 函数的返回值超过 38 位，则结果将截断以符合规范，这将导致精度降低。如果在插值期间，中间结果超出最大精度，则会发生数值溢出且函数会返回错误。要避免这些情况，建议使用具有较低精度的数据类型或将 median_expression 参数转换为较低精度。

例如，带 DECIMAL 参数的 SUM 函数返回 38 位的默认精度。结果的小数位数与参数的小数位数相同。因此，例如，DECIMAL(5,2) 列的 SUM 返回 DECIMAL(38,2) 数据类型。

以下示例在 MEDIAN 函数的 median_expression 参数中使用 SUM 函数。PRICEPAID 列的数据类型是 DECIMAL (8,2)，因此 SUM 函数返回 DECIMAL(38,2)。

```
select salesid, sum(pricepaid), median(sum(pricepaid))
over() from sales where salesid < 10 group by salesid;
```

要避免潜在的精度降低或溢出错误，请将结果转换为具有较低精度的 DECIMAL 数据类型，如以下示例所示。

```
select salesid, sum(pricepaid), median(sum(pricepaid)::decimal(30,2))
over() from sales where salesid < 10 group by salesid;
```

示例

以下示例计算每个卖家的平均销售数量：

```
select sellerid, qty, median(qty)
over (partition by sellerid)
from winsales
order by sellerid;


sellerid qty median
---------------------------
1  10 10.0
1  10 10.0
1  30 10.0
2  20 20.0
2  20 20.0
3  10 17.5
3  15 17.5
3  20 17.5
3  30 17.5
4  10 25.0
4  40 25.0
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

MIN 窗口函数

MIN 窗口函数返回最小输入表达式值。MIN 函数使用数值并忽略 NULL 值。

语法

```
MIN ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 MIN 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

接受任何数据类型作为输入。返回与 expression 相同的数据类型。

示例

以下示例从数据窗口的开头显示销售 ID、数量和最小数量：

```
select salesid, qty,
min(qty) over
(order by salesid rows unbounded preceding)
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  10
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  10
40001 |  40 |  10
40005 |  10 |  10
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例在受限制的框架中显示销售 ID、数量和最小数量：

```
select salesid, qty,
min(qty) over
(order by salesid rows between 2 preceding and 1 preceding) as min
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  20
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  15
40001 |  40 |  20
```

```
40005 |  10 |  30
(11 rows)
```

NTH_VALUE 窗口函数

NTH_VALUE 窗口函数返回相对于窗口的第一行的窗口框架的指定行的表达式值。

语法

```
NTH_VALUE (expr, offset)
[ IGNORE NULLS | RESPECT NULLS ]
OVER
( [ PARTITION BY window_partition ]
[ ORDER BY window_ordering
                        frame_clause ] )
```

参数

expr

对其执行函数的目标列或表达式。

offset

确定相对于要为其返回表达式的窗口中的第一行的行号。offset 可以是常数或表达式，且必须为大于 0 的正整数。

IGNORE NULLS

一种可选规范，用于指示在确定要使用哪一行时 AWS Clean Rooms 应跳过空值。如果未列出 IGNORE NULLS，则包含 Null 值。

RESPECT NULLS

表示在确定 AWS Clean Rooms 要使用哪一行时应包含空值。如果您未指定 IGNORE NULLS，则默认情况下不支持 RESPECT NULLS。

OVER

指定窗口分区、排序和窗口框架。

PARTITION BY window_partition

设置 OVER 子句中每个组的记录范围。

ORDER BY window_ordering

对每个分区中的行进行排序。如果忽略 ORDER BY，则默认框架将包含分区中的所有行。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

NTH_VALUE 窗口函数支持使用任何数据类型的表达式。 AWS Clean Rooms 返回类型与 expr 的类型相同。

示例

以下示例显示了加利福利亚、佛罗里达和纽约的第三大场地的座位数与这些州的其他场地的座位数的比较情况：

```
select venuestate, venuename, venueseats,
nth_value(venueseats, 3)
ignore nulls
over(partition by venuestate order by venueseats desc
rows between unbounded preceding and unbounded following)
as third_most_seats
from (select * from venue where venueseats > 0 and
venuestate in('CA', 'FL', 'NY'))
order by venuestate;

venuestate |           venuename            | venueseats | third_most_seats
-----------+--------------------------------+------------+------------------
CA         | Qualcomm Stadium               |      70561 |            63026
CA         | Monster Park                   |      69843 |            63026
CA         | McAfee Coliseum                |      63026 |            63026
CA         | Dodger Stadium                 |      56000 |            63026
CA         | Angel Stadium of Anaheim       |      45050 |            63026
CA         | PETCO Park                     |      42445 |            63026
CA         | AT&T Park                      |      41503 |            63026
CA         | Shoreline Amphitheatre         |      22000 |            63026
FL         | Dolphin Stadium                |      74916 |            65647
FL         | Jacksonville Municipal Stadium |      73800 |            65647
FL         | Raymond James Stadium          |      65647 |            65647
FL         | Tropicana Field                |      36048 |            65647
NY         | Ralph Wilson Stadium           |      73967 |            20000
```

```
NY         | Yankee Stadium                |      52325 |            20000
NY         | Madison Square Garden         |      20000 |            20000
(15 rows)
```

NTILE 窗口函数

NTILE 窗口函数将分区中已排序的行划分为大小尽可能相等的指定数量的已排名组，并返回给定行所在的组。

语法

```
NTILE (expr)
OVER (
[ PARTITION BY expression_list ]
[ ORDER BY order_list ]
)
```

参数

expr

排名组的数目，并且必须为每个分区生成一个正整数值（大于零）。expr 参数不得可为 null。

OVER

一个指定窗口分区和排序的子句。OVER 子句不能包含窗口框架规范。

PARTITION BY window_partition

可选。OVER 子句中每个组的记录范围。

ORDER BY window_ordering

可选。一个对每个分区中的行进行排序的表达式。如果忽略 ORDER BY 子句，则排名行为相同。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅窗口函数的唯一数据排序。

返回类型

BIGINT

示例

以下示例将于 2008 年 8 月 26 日购买 Hamlet 门票所付价格划分到四个排名组中。结果集为 17 个行，几乎均匀地划分到排名 1 到 4 中：

```
select eventname, caldate, pricepaid, ntile(4)
over(order by pricepaid desc) from sales, event, date
where sales.eventid=event.eventid and event.dateid=date.dateid and eventname='Hamlet'
and caldate='2008-08-26'
order by 4;

eventname |  caldate   | pricepaid | ntile
----------+------------+-----------+-------
Hamlet    | 2008-08-26 |   1883.00 |     1
Hamlet    | 2008-08-26 |   1065.00 |     1
Hamlet    | 2008-08-26 |    589.00 |     1
Hamlet    | 2008-08-26 |    530.00 |     1
Hamlet    | 2008-08-26 |    472.00 |     1
Hamlet    | 2008-08-26 |    460.00 |     2
Hamlet    | 2008-08-26 |    355.00 |     2
Hamlet    | 2008-08-26 |    334.00 |     2
Hamlet    | 2008-08-26 |    296.00 |     2
Hamlet    | 2008-08-26 |    230.00 |     3
Hamlet    | 2008-08-26 |    216.00 |     3
Hamlet    | 2008-08-26 |    212.00 |     3
Hamlet    | 2008-08-26 |    106.00 |     3
Hamlet    | 2008-08-26 |    100.00 |     4
Hamlet    | 2008-08-26 |     94.00 |     4
Hamlet    | 2008-08-26 |     53.00 |     4
Hamlet    | 2008-08-26 |     25.00 |     4
(17 rows)
```

PERCENT_RANK 开窗函数

计算给定行的百分比排名。使用以下公式确定百分比排名：

```
(x - 1) / (the number of rows in the window or partition - 1)
```

其中，x 为当前行的排名。以下数据集说明了此公式的使用：

```
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
```

返回值范围介于 0 和 1（含 1）之间。任何集合中的第一行的 PERCENT_RANK 均为 0。

语法

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

参数

( )

该函数没有参数，但需要空括号。

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口框架规范。

PARTITION BY partition_expression

可选。一个设置 OVER 子句中每个组的记录范围的表达式。

ORDER BY order_list

可选。用于计算百分比排名的表达式。该表达式必须具有数字数据类型或可隐式转换为 1。如果省略 ORDER BY，则所有行的返回值为 0。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅窗口函数的唯一数据排序。

返回类型

FLOAT8

示例

以下示例计算每个卖家的销售数量的百分比排名：

```
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
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

PERCENTILE_CONT 开窗函数

PERCENTILE_CONT 是一种假定连续分布模型的逆分布函数。该函数具有一个百分比值和一个排序规范，并返回一个在有关排序规范的给定百分比值范围内的内插值。

PERCENTILE_CONT 在对值进行排序后计算值之间的线性内插。通过在聚合组中使用百分比值 (P) 和非 null 行数 (N)，该函数会在根据排序规范对行进行排序后计算行号。根据公式 (RN) 计算此行号 RN = (1+ (P*(N-1))。聚合函数的最终结果通过行号 CRN = CEILING(RN) 和 FRN = FLOOR(RN) 的行中的值之间的线性内插计算。

最终结果将如下所示。

如果 (CRN = FRN = RN)，则结果为 (value of expression from row at RN)

否则，结果将如下所示：

(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN).

您在 OVER 子句中只能指定 PARTITION 子句。如果为每个行指定 PARTITION，则 PERCENTILE_CONT 会返回位于给定分区内一组值中的指定百分比范围内的值。

PERCENTILE_CONT 是仅计算节点函数。如果查询未引用用户定义的表或 AWS Clean Rooms 系统表，则该函数将返回错误。

语法

```
PERCENTILE_CONT ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

参数

percentile

介于 0 和 1 之间的数字常数。计算中将忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定用于排序和计算百分比的数字或日期/时间值。

OVER

指定窗口分区。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY expr

设置 OVER 子句中每个组的记录范围的可选参数。

返回值

返回类型由 WITHIN GROUP 子句中的 ORDER BY 表达式的数据类型决定。下表显示了每种个 ORDER BY 表达式数据类型的返回类型。

输入类型	返回类型
SMALLINTINTEGERBIGINTNUMERIC、DECIMAL	DECIMAL
FLOAT、DOUBLE	DOUBLE

输入类型	返回类型
DATE	DATE
TIMESTAMP	TIMESTAMP

使用说明

如果 ORDER BY 表达式是使用 38 位最大精度定义的 DECIMAL 数据类型，则 PERCENTILE_CONT 可能将返回不准确的结果或错误。如果 PERCENTILE_CONT 函数的返回值超过 38 位，则结果将截断以符合规范，这将导致精度降低。如果在插值期间，中间结果超出最大精度，则会发生数值溢出且函数会返回错误。要避免这些情况，建议使用具有较低精度的数据类型或将 ORDER BY 表达式转换为较低精度。

例如，带 DECIMAL 参数的 SUM 函数返回 38 位的默认精度。结果的小数位数与参数的小数位数相同。因此，例如，DECIMAL(5,2) 列的 SUM 返回 DECIMAL(38,2) 数据类型。

以下示例在 PERCENTILE_CONT 函数的 ORDER BY 子句中使用 SUM 函数。PRICEPAID 列的数据类型是 DECIMAL (8,2)，因此 SUM 函数返回 DECIMAL(38,2)。

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid) desc) over()
from sales where salesid < 10 group by salesid;
```

要避免潜在的精度降低或溢出错误，请将结果转换为具有较低精度的 DECIMAL 数据类型，如以下示例所示。

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid)::decimal(30,2) desc) over()
from sales where salesid < 10 group by salesid;
```

示例

以下示例使用 WINSALES 表。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over() as median from winsales;
```

```
 sellerid | qty | median
----------+-----+--------
        1 |  10 |   20.0
        1 |  10 |   20.0
        3 |  10 |   20.0
        4 |  10 |   20.0
        3 |  15 |   20.0
        2 |  20 |   20.0
        3 |  20 |   20.0
        2 |  20 |   20.0
        3 |  30 |   20.0
        1 |  30 |   20.0
        4 |  40 |   20.0
(11 rows)
```

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

 sellerid | qty | median
----------+-----+--------
        2 |  20 |   20.0
        2 |  20 |   20.0
        4 |  10 |   25.0
        4 |  40 |   25.0
        1 |  10 |   10.0
        1 |  10 |   10.0
        1 |  30 |   10.0
        3 |  10 |   17.5
        3 |  15 |   17.5
        3 |  20 |   17.5
        3 |  30 |   17.5
(11 rows)
```

以下示例计算华盛顿州的卖家的门票销售的 PERCENTILE_CONT 和 PERCENTILE_DISC。

```
SELECT sellerid, state, sum(qtysold*pricepaid) sales,
percentile_cont(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over(),
percentile_disc(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over()
from sales s, users u
```

```
where s.sellerid = u.userid and state = 'WA' and sellerid < 1000
group by sellerid, state;

 sellerid | state |  sales  | percentile_cont | percentile_disc
----------+-------+---------+-----------------+-----------------
      127 | WA    | 6076.00 |         2044.20 |         1531.00
      787 | WA    | 6035.00 |         2044.20 |         1531.00
      381 | WA    | 5881.00 |         2044.20 |         1531.00
      777 | WA    | 2814.00 |         2044.20 |         1531.00
       33 | WA    | 1531.00 |         2044.20 |         1531.00
      800 | WA    | 1476.00 |         2044.20 |         1531.00
        1 | WA    | 1177.00 |         2044.20 |         1531.00
(7 rows)
```

PERCENTILE_DISC 开窗函数

PERCENTILE_DISC 是一种假定离散分布模型的逆分布函数。该函数具有一个百分比值和一个排序规范，并返回给定集合中的元素。

对于给定的百分比值 P，PERCENTILE_DISC 在 ORDER BY 子句中对表达式的值进行排序，并返回带有大于或等于 P 的最小累积分布值（相对于同一排序规范）的值。

您在 OVER 子句中只能指定 PARTITION 子句。

PERCENTILE_DISC 是仅计算节点函数。如果查询未引用用户定义的表或 AWS Clean Rooms 系统表，则该函数将返回错误。

语法

```
PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

参数

percentile

介于 0 和 1 之间的数字常数。计算中将忽略 Null。

WITHIN GROUP (ORDER BY expr)

指定用于排序和计算百分比的数字或日期/时间值。

OVER

指定窗口分区。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY expr

设置 OVER 子句中每个组的记录范围的可选参数。

返回值

与 WITHIN GROUP 子句中的 ORDER BY 表达式相同的数据类型。

示例

以下示例使用 WINSALES 表。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over() as median from winsales;

sellerid | qty | median
----------+-----+--------
        1 |  10 |     20
        3 |  10 |     20
        1 |  10 |     20
        4 |  10 |     20
        3 |  15 |     20
        2 |  20 |     20
        2 |  20 |     20
        3 |  20 |     20
        1 |  30 |     20
        3 |  30 |     20
        4 |  40 |     20
(11 rows)
```

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

sellerid | qty | median
----------+-----+--------
        2 |  20 |     20
        2 |  20 |     20
```

```
        4 |  10 |     10
        4 |  40 |     10
        1 |  10 |     10
        1 |  10 |     10
        1 |  30 |     10
        3 |  10 |     15
        3 |  15 |     15
        3 |  20 |     15
        3 |  30 |     15
(11 rows)
```

RANK 窗口函数

RANK 窗口函数基于 OVER 子句中的 ORDER BY 表达式确定一组值中的一个值的排名。如果存在可选的 PARTITION BY 子句，则为每个行组重置排名。排名标准值相等的行将获得相同的排名。 AWS Clean Rooms 将并列的行数与并列的排名相加以计算下一个等级，因此排名可能不是连续的数字。例如，如果两个行的排名为 1，则下一个排名则为 3。

RANK 与 DENSE_RANK 窗口函数存在以下一点不同：对于 DENSE_RANK 来说，如果两个或两个以上的行结合，则一系列排名的值之间没有间隔。例如，如果两个行的排名为 1，则下一个排名则为 2。

您可以在同一查询中包含带有不同的 PARTITION BY 和 ORDER BY 子句的排名函数。

语法

```
RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

参数

( )

该函数没有参数，但需要空括号。

OVER

适用于 RANK 函数的窗口子句。

PARTITION BY expr_list

可选。一个或多个定义窗口的表达式。

ORDER BY order_list

可选。定义排名值基于的列。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。如果省略 ORDER BY，则所有行的返回值为 1。

如果 ORDER BY 未生成唯一顺序，则行的顺序是不确定的。有关更多信息，请参阅窗口函数的唯一数据排序。

返回类型

INTEGER

示例

以下示例按销量对表进行排序（预设情况下按升序顺序），并为每个行分配一个排名。排名值 1 为排名最高的值。在应用窗口函数结果后，对结果进行排序：

```
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)
```

请注意，此示例中的外部 ORDER BY 子句包括第 2 列和第 1 列，以确保每次运行此查询时 AWS Clean Rooms 返回排序一致的结果。例如，销售 ID 为 10001 和 10006 的行具有相同的 QTY 和 RNK 值。按列 1 对最后的结果集进行排序可确保行 10001 始终在 10006 之前。有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

在下面的示例中，将窗口函数的顺序倒转 (order by qty desc)。现在，最高排名值将应用于最大的 QTY 值。

```
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)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例按 SELLERID 对表进行分区，按数量对每个分区进行排序（按降序顺序），并为每个行分配排名。在应用窗口函数结果后，对结果进行排序。

```
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)
```

RATIO_TO_REPORT 开窗函数

计算某个窗口或分区中的某个值与所有值的和的比率。使用以下公式确定报表值的比率：

`value of` ratio_expression `argument for the current row / sum of` ratio_expression `argument for the window or partition`

以下数据集说明了此公式的使用：

```
Row# Value Calculation RATIO_TO_REPORT
1 2500 (2500)/(13900) 0.1798
2 2600 (2600)/(13900) 0.1870
3 2800 (2800)/(13900) 0.2014
4 2900 (2900)/(13900) 0.2086
5 3100 (3100)/(13900) 0.2230
```

返回值范围介于 0 和 1（含 1）之间。如果 ratio_expression 为 NULL，则返回值为 NULL。

语法

```
RATIO_TO_REPORT ( ratio_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

参数

ratio_expression

一个提供要为其确定比率的值的表达式（例如列名）。该表达式必须具有数字数据类型或可隐式转换为 1。

您无法在 ratio_expression 中使用任何其他分析函数。

OVER

一个指定窗口分区的子句。OVER 子句不能包含窗口排序或窗口框架规范。

PARTITION BY partition_expression

可选。一个设置 OVER 子句中每个组的记录范围的表达式。

返回类型

FLOAT8

示例

以下示例计算每个卖家的销售数量的比率：

```
select sellerid, qty, ratio_to_report(qty)
over (partition by sellerid)
from winsales;

sellerid qty  ratio_to_report
----------------------------------------
2  20.12312341      0.5
2  20.08630000      0.5
4  10.12414400      0.2
4  40.23000000      0.8
1  30.37262000      0.6
1  10.64000000      0.21
1  10.00000000      0.2
3  10.03500000      0.13
3  15.14660000      0.2
3  30.54790000      0.4
3  20.74630000      0.27
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

ROW_NUMBER 窗口函数

基于 OVER 子句中的 ORDER BY 表达式确定一组行中当前行的序号（从 1 开始计数）。如果存在可选的 PARTITION BY 子句，则为每组行重置序号。ORDER BY 表达式中具有相同值的行以非确定性的方式接收不同的行号。

语法

```
ROW_NUMBER () OVER
```

```
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

参数

( )

该函数没有参数，但需要空括号。

OVER

适用于 ROW_NUMBER 函数的窗口子句。

PARTITION BY expr_list

可选。一个或多个定义 ROW_NUMBER 函数的表达式。

ORDER BY order_list

可选。定义行数基于的列的表达式。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

如果 ORDER BY 未生成唯一顺序或被省略，则行的顺序是不确定的。有关更多信息，请参阅窗口函数的唯一数据排序。

返回类型

BIGINT

示例

以下示例按 SELLERID 对表进行分区并按 QTY 对每个分区进行排序（按升序顺序），然后为每个行分配一个行号。在应用窗口函数结果后，对结果进行排序。

```
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)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

STDDEV_SAMP 和 STDDEV_POP 窗口函数

STDDEV_SAMP 和 STDDEV_POP 窗口函数返回一组数值（整数、小数或浮点）的样本标准差和总体标准差。另请参阅 STDDEV_SAMP 和 STDDEV_POP 函数。

STDDEV_SAMP 和 STDDEV 是同一函数的同义词。

语法

```
STDDEV_SAMP | STDDEV | STDDEV_POP
( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

STDDEV 函数支持的参数类型包括 SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 和 DOUBLE PRECISION。

无论表达式的数据类型如何，STDDEV 函数的返回类型都是双精度数。

示例

以下示例说明如何使用 STDDEV_POP 和 VAR_POP 作为窗口函数。查询计算 SALES 表中 PRICEPAID 值的总体方差和总体标准差。

```
select salesid, dateid, pricepaid,
round(stddev_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as stddevpop,
round(var_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as varpop
from sales
order by 2,1;

salesid | dateid | pricepaid | stddevpop | varpop
--------+--------+-----------+-----------+---------
  33095 |   1827 |    234.00 |         0 |       0
  65082 |   1827 |    472.00 |       119 |   14161
  88268 |   1827 |    836.00 |       248 |   61283
```

```
  97197 |   1827 |    708.00 |       230 |   53019
 110328 |   1827 |    347.00 |       223 |   49845
 110917 |   1827 |    337.00 |       215 |   46159
 150314 |   1827 |    688.00 |       211 |   44414
 157751 |   1827 |   1730.00 |       447 |  199679
 165890 |   1827 |   4192.00 |      1185 | 1403323
...
```

样本标准差和样本标准方差函数可通过同一方式使用。

SUM 窗口函数

SUM 窗口函数返回输入列值或表达式值的和。SUM 函数使用数值并忽略 NULL 值。

语法

```
SUM ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义 SUM 函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

SUM 函数支持的参数类型为 SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 和 DOUBLE PRECISION。

SUM 函数支持的返回类型为：

- 适用于 SMALLINT 或 INTEGER 参数的 BIGINT
- 适用于 BIGINT 参数的 NUMERIC
- 适用于浮点参数的 DOUBLE PRECISION

示例

以下示例创建按日期和销售 ID 排序的销售数量的累积（移动）和：

```
select salesid, dateid, sellerid, qty,
sum(qty) over (order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  20
10005 | 2003-12-24 |        1 |  30 |  50
40001 | 2004-01-09 |        4 |  40 |  90
10006 | 2004-01-18 |        1 |  10 | 100
20001 | 2004-02-12 |        2 |  20 | 120
40005 | 2004-02-12 |        4 |  10 | 130
20002 | 2004-02-16 |        2 |  20 | 150
30003 | 2004-04-18 |        3 |  15 | 165
30004 | 2004-04-18 |        3 |  20 | 185
30007 | 2004-09-07 |        3 |  30 | 215
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例按日期创建销售数量的累积（移动）和，按卖家 ID 对结果进行分区，并按日期和销售 ID 对该分区中的结果进行排序：

```
select salesid, dateid, sellerid, qty,
sum(qty) over (partition by sellerid
order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
10005 | 2003-12-24 |        1 |  30 |  40
40001 | 2004-01-09 |        4 |  40 |  40
10006 | 2004-01-18 |        1 |  10 |  50
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  50
20002 | 2004-02-16 |        2 |  20 |  40
30003 | 2004-04-18 |        3 |  15 |  25
30004 | 2004-04-18 |        3 |  20 |  45
30007 | 2004-09-07 |        3 |  30 |  75
(11 rows)
```

对结果集中的所有行进行编号（按 SELLERID 和 SALESID 列进行排序）：

```
select salesid, sellerid, qty,
sum(1) over (order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid |  qty | rownum
--------+----------+------+--------
10001 |        1 |   10 |      1
10005 |        1 |   30 |      2
10006 |        1 |   10 |      3
20001 |        2 |   20 |      4
20002 |        2 |   20 |      5
30001 |        3 |   10 |      6
30003 |        3 |   15 |      7
30004 |        3 |   20 |      8
```

```
30007 |        3 |   30 |     9
40001 |        4 |   40 |    10
40005 |        4 |   10 |    11
(11 rows)
```

有关 WINSALES 表的说明，请参阅窗口函数示例的示例表。

以下示例对结果集中的所有行按顺序进行编号，按 SELLERID 对结果进行分区，并按 SELLERID 和 SALESID 对该分区中的结果进行排序：

```
select salesid, sellerid, qty,
sum(1) over (partition by sellerid
order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid | qty | rownum
---------+----------+-----+--------
10001 |        1 |  10 |      1
10005 |        1 |  30 |      2
10006 |        1 |  10 |      3
20001 |        2 |  20 |      1
20002 |        2 |  20 |      2
30001 |        3 |  10 |      1
30003 |        3 |  15 |      2
30004 |        3 |  20 |      3
30007 |        3 |  30 |      4
40001 |        4 |  40 |      1
40005 |        4 |  10 |      2
(11 rows)
```

VAR_SAMP 和 VAR_POP 窗口函数

VAR_SAMP 和 VAR_POP 窗口函数返回一组数值（整数、小数或浮点）的样本方差和总体方差。另请参阅 VAR_SAMP 和 VAR_POP 函数。

VAR_SAMP 和 VARIANCE 是同一函数的同义词。

语法

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

```
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

参数

expression

对其执行函数的目标列或表达式。

ALL

利用参数 ALL，该函数可保留表达式中的所有重复值。ALL 是默认值。DISTINCT 不受支持。

OVER

指定聚合函数的窗口子句。OVER 子句将窗口聚合函数与普通集合聚合函数区分开来。

PARTITION BY expr_list

依据一个或多个表达式定义函数的窗口。

ORDER BY order_list

对每个分区中的行进行排序。如果未指定 PARTITION BY，则 ORDER BY 使用整个表。

frame_clause

如果 ORDER BY 子句用于聚合函数，则需要显式框架子句。框架子句优化函数窗口中的行集，包含或排除已排序结果中的行集。框架子句包括 ROWS 关键字和关联的说明符。请参阅 窗口函数语法摘要。

数据类型

VARIANCE 函数支持的参数类型包括
SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL 和 DOUBLE PRECISION。

无论表达式的数据类型如何，VARIANCE 函数的返回类型都是双精度数。

中的 SQL 条件 AWS Clean Rooms

条件是包含一个或多个表达式和逻辑运算符的语句，计算结果为 true、false 或 unknown。条件有时也称为“谓词”。

> **Note**
>
> 所有字符串比较和 LIKE 模式匹配项均区分大小写。例如，“A”和“a”不匹配。但是，您可通过使用 ILIKE 谓词执行不区分大小写的模式匹配。

中支持以下 SQL 条件 AWS Clean Rooms。

主题

比较条件

比较条件阐明两个值之间的逻辑关系。所有比较条件都是具有布尔值返回类型的二进制运算符。AWS Clean Rooms 支持下表中描述的比较运算符。

操作符	语法	描述
<	a < b	值 a 小于值 b。
>	a > b	值 a 大于值 b。
<=	a <= b	值 a 小于或等于值 b。

操作符	语法	描述
>=	a >= b	值 a 大于或等于值 b。
=	a = b	值 a 等于值 b。
<> 或 !=	a <> b or a != b	值 a 不等于值 b。
a = TRUE	a IS TRUE	值 a 为布尔值 TRUE。

使用说明

= ANY | SOME

ANY 和 SOME 关键字与 IN 条件同义。当比较操作相对于可返回一个或多个值的子查询所返回的至少一个值为 true 时，ANY 和 SOME 关键字将返回 true。对于 ANY 和 SOME，AWS Clean Rooms 仅支持 =（等于）条件。不支持不相等条件。

> **Note**
>
> 不支持 ALL 谓词。

<> ALL

ALL 关键字与 NOT IN（请参阅 IN 条件 条件）同义并在表达式未包含在子查询的结果中时返回 true。对于 ALL，AWS Clean Rooms 仅支持 <> 或 != （不等于）条件。不支持其他比较条件。

IS TRUE/FALSE/UNKNOWN

非零值等于 TRUE，0 等于 FALSE，null 等于 UNKNOWN。请参阅布尔值类型数据类型。

示例

下面是比较条件的一些简单示例：

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

以下查询返回 VENUE 表中座位数超过 1 万的场地：

```
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)
```

此示例从 USERS 表中选择喜欢摇滚音乐的用户 (USERID)：

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
13
16
(5 rows)
```

此示例从 USERS 表中选择不清楚是否喜欢摇滚音乐的用户 (USERID)：

```
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
```

具有 TIME 列的示例

下面的示例表 TIME_TEST 具有一个列 TIME_VAL（类型 TIME），其中插入了三个值。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

以下示例从每个 timetz_val 中提取小时数。

```
select time_val from time_test where time_val < '3:00';
   time_val
---------------
 00:00:00.5550
 00:58:00
```

以下示例比较两种时间文本。

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
 t
```

具有 TIMETZ 列的示例

下面的示例表 TIMETZ_TEST 具有一个列 TIMETZ_VAL（类型 TIMETZ），其中插入了三个值。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

下面的示例仅选择小于 3:00:00 UTC 的 TIMETZ 值。将值转换为 UTC 后进行比较。

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

以下示例比较两种 TIMETZ 文本。比较时忽略时区。

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

 ?column?
----------
 t
```

逻辑条件

逻辑条件组合两个条件的结果以生成一个结果。所有逻辑条件都是具有布尔值返回类型的二进制运算符。

语法

```
expression
{ AND | OR }
expression
NOT expression
```

逻辑条件使用具有三个值的布尔逻辑，其中 null 值表示未知关系。下表描述逻辑条件的结果，其中 E1 和 E2 表示表达式：

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	

NOT 运算符先于 AND 计算，而 AND 运算符先于 OR 运算符计算。使用的任何圆括号可优先于此默认计算顺序。

示例

以下示例将返回 USERS 表中用户同时喜欢拉斯维加斯和运动的 USERID 和 USERNAME：

```
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)
```

下一个示例将返回 USERS 表中用户喜欢拉斯维加斯或运动或同时喜欢这二者的 USERID 和 USERNAME。此查询将返回上例中的所有输出以及只喜欢拉斯维加斯或运动的用户。

```
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)
```

以下查询使用圆括号将 OR 条件括起来以查找纽约或加利福尼亚演出过 Macbeth 的场地：

```
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
...
```

删除此示例中的圆括号将更改逻辑和查询的结果。

以下示例使用 NOT 运算符：

```
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
...
```

以下示例使用一个 NOT 条件并后跟一个 AND 条件：

```
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)
```

模式匹配条件

模式匹配运算符针对条件表达式中指定的模式搜索字符串，然后根据是否找到匹配项来返回 true 或 false。AWS Clean Rooms 使用以下方法进行模式匹配：

- LIKE 表达式

  LIKE 运算符将字符串表达式（如列名称）与使用通配符 %（百分比）和 _（下划线）的模式进行比较。LIKE 模式匹配始终涵盖整个字符串。LIKE 执行区分大小写的匹配，而 ILIKE 执行不区分大小写的匹配。

- SIMILAR TO 正则表达式

  SIMILAR TO 运算符使用 SQL 标准正则表达式模式来匹配字符串表达式，该模式可包含一组模式匹配元字符，其中包括 LIKE 运算符支持的两个元字符。SIMILAR TO 匹配整个字符串并且执行区分大小写的匹配。

主题

- LIKE
- SIMILAR TO

LIKE

LIKE 运算符将字符串表达式（如列名称）与使用通配符 %（百分比）和 _（下划线）的模式进行比较。LIKE 模式匹配始终涵盖整个字符串。若要匹配字符串中任意位置的序列，模式必须以百分比符号开始和结尾。

LIKE 区分大小写；ILIKE 不区分大小写。

语法

```
expression [ NOT ] LIKE | ILIKE pattern [ ESCAPE 'escape_char' ]
```

参数

expression

有效的 UTF-8 字符表达式（如列名称）。

LIKE | ILIKE

LIKE 执行区分大小写的模式匹配。ILIKE 对单字节 UTF-8 (ASCII) 字符执行不区分大小写的模式匹配。要为多字节字符执行不区分大小写的模式匹配，请将 expression 上的 LOWER 函数和带有 LIKE 函数的 pattern 一起使用。

与比较谓词（例如 = 和 <>）相反，LIKE 和 ILIKE 谓词并不会隐式忽略尾随空格。要忽略尾随空格，请使用 RTRIM 或者将 CHAR 列显式强制转换为 VARCHAR。

~~ 运算符等同于 LIKE，而 ~~* 等同于 ILIKE。此外，!~~ 和 !~~* 运算符等同于 NOT LIKE 和 NOT ILIKE。

pattern

具有要匹配的模式的有效的 UTF-8 字符表达式。

escape_char

将对模式中的元字符进行转义的字符表达式。默认为两个反斜杠 ('\\')。

如果 pattern 不包含元字符，则模式仅表示字符串本身；在此情况下，LIKE 的行为与等于运算符相同。

其中一个字符表达式可以是 CHAR 或 VARCHAR 数据类型。如果它们不同，AWS Clean Rooms 会将 pattern 转换为 expression 的数据类型。

LIKE 支持下列模式匹配元字符：

操作符	描述
%	匹配任意序列的零个或多个字符。
_	匹配任何单个字符。

示例

下表显示使用 LIKE 的模式匹配的示例：

表达式	返回值
`'abc' LIKE 'abc'`	True
`'abc' LIKE 'a%'`	True
`'abc' LIKE '_B_'`	False

表达式	返回值
`'abc' ILIKE '_B_'`	True
`'abc' LIKE 'c%'`	False

以下示例查找名称以“E”开头的所有城市：

```
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
...
```

以下示例查找姓中包含“ten”的用户：

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
-------------
Christensen
Wooten
...
```

以下示例查找第三和第四个字符为“ea”的城市。此命令使用 ILIKE 来演示不区分大小写的匹配：

```
select distinct city from users where city ilike '__EA%' order by city;
city
-------------
Brea
Clearwater
Great Falls
Ocean City
```

```
Olean
Wheaton
(6 rows)
```

以下示例使用原定设置转义字符串（\\）搜索包含“start_”（文本 start 后跟下划线 _)的字符串：

```
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)
```

以下示例指定“^”作为转义字符，然后使用该转义字符搜索包含“start_”（文本 start 后跟下划线 _)的字符串：

```
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)
```

以下示例使用 ~~* 运算符对以“Ag”开头的城市进行不区分大小写（ILIKE）的搜索。

```
select distinct city from users where city ~~* 'Ag%' order by city;
```

```
city
------------
Agat
Agawam
Agoura Hills
Aguadilla
```

SIMILAR TO

SIMILAR TO 运算符使用 SQL 标准正则表达式模式来匹配字符串表达式（如列名称）。SQL 正则表达式可包含一组模式匹配元字符，包括 LIKE 运算符支持的两个元字符。

仅当模式与整个字符串匹配时，SIMILAR TO 运算符才会返回 true，这与 POSIX 正则表达式的行为不同（其中的模式可与字符串的任何部分匹配）。

SIMILAR TO 执行区分大小写的匹配。

> **Note**
>
> 使用 SIMILAR TO 的正则表达式匹配的计算成本高昂。我们建议尽可能使用 LIKE，尤其是在处理非常多的行时。例如，下列查询的功能相同，但使用 LIKE 的查询相比于使用正则表达式的查询的运行速度快若干倍：
>
> ```
> select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
> select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die
> %';
> ```

语法

```
expression [ NOT ] SIMILAR TO pattern [ ESCAPE 'escape_char' ]
```

参数

expression

有效的 UTF-8 字符表达式（如列名称）。

SIMILAR TO

SIMILAR TO 对 expression 中的整个字符串执行区分大小写的模式匹配。

pattern

有效的 UTF-8 字符表达式，表示 SQL 标准正则表达式模式。

escape_char

将对模式中的元字符进行转义的字符表达式。默认为两个反斜杠 ('\\')。

如果 pattern 不包含元字符，则模式仅表示字符串本身。

其中一个字符表达式可以是 CHAR 或 VARCHAR 数据类型。如果它们不同，AWS Clean Rooms 会将 pattern 转换为 expression 的数据类型。

SIMILAR TO 支持下列模式匹配元字符：

操作符	描述
%	匹配任意序列的零个或多个字符。
_	匹配任何单个字符。
|	表示替换（两个替换中的一个）。
*	重复上一项目零次或更多次。
+	重复上一项目一次或更多次。
?	重复上一项目零次或一次。
{m}	重复上一项目正好 m 次。
{m,}	重复上一项目 m 次或更多次。
{m,n}	重复上一项目至少 m 次且不超过 n 次。
()	圆括号将项分组为单个逻辑项。
[...]	括号表达式指定一个字符类，正如在 POSIX 正则表达式中一样。

示例

下表显示了使用 SIMILAR TO 的模式匹配的示例：

表达式	返回值
'abc' SIMILAR TO 'abc'	True
'abc' SIMILAR TO '_b_'	True
'abc' SIMILAR TO '_A_'	False
'abc' SIMILAR TO '%(b|d)%'	True
'abc' SIMILAR TO '(b|c)%'	False
'AbcAbcdefgefg12efgefg12' SIMILAR TO '((Ab)?c)+d((efg)+(12))+'	True
'aaaaaab11111xy' SIMILAR TO 'a{6}_[0-9]{5}(x|y){2}'	True
'$0.87' SIMILAR TO '$[0-9]+(.[0-9][0-9])?'	True

以下示例查找名称包含“E”或“H”的城市：

```
SELECT DISTINCT city FROM users
WHERE city SIMILAR TO '%E%|%H%' ORDER BY city LIMIT 5;

      city
-----------------
 Agoura Hills
 Auburn Hills
 Benton Harbor
 Beverly Hills
 Chicago Heights
```

以下示例使用默认转义字符串（“\\”）搜索包含“_”的字符串：

```
SELECT tablename, "column" FROM my_table_def
WHERE "column" SIMILAR TO '%start\\_%'

ORDER BY tablename, "column" LIMIT 5;
```

```
        tablename         |       column
--------------------------+---------------------
 my_abort_idle            | idle_start_time
 my_abort_idle            | txn_start_time
 my_analyze_compression   | start_time
 my_auto_worker_levels    | start_level
 my_auto_worker_levels    | start_wlm_occupancy
```

以下示例指定“^”作为转义字符串，然后使用此转义字符串搜索包含“_”的字符串：

```
SELECT tablename, "column" FROM my_table_def

WHERE "column" SIMILAR TO '%start^_%' ESCAPE '^'
ORDER BY tablename, "column" LIMIT 5;

        tablename         |       column
--------------------------+---------------------
 stcs_abort_idle          | idle_start_time
 stcs_abort_idle          | txn_start_time
 stcs_analyze_compression | start_time
 stcs_auto_worker_levels  | start_level
 stcs_auto_worker_levels  | start_wlm_occupancy
```

BETWEEN 范围条件

BETWEEN 条件使用关键字 BETWEEN 和 AND 测试表达式是否包含在某个值范围中。

语法

```
expression [ NOT ] BETWEEN expression AND expression
```

表达式可以是数字、字符或日期时间数据类型，但它们必须是可兼容的。此范围包含起始值。

示例

第一个示例计算有多少个事务登记了 2、3 或 4 票证的销售：

```
select count(*) from sales
```

```
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

范围条件包含开始和结束值。

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
1900 | 1910
```

范围条件中的第一个表达式必须是较小的值，第二个表达式必须是较大的值。在以下示例中，由于表达式的值，将始终返回零行：

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

但是，应用 NOT 修饰符将反转逻辑并生成所有行的计数：

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
172456
(1 row)
```

以下查询将返回拥有 20000 到 50000 个座位的场馆的列表：

```
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)
```

以下示例演示了如何为日期值使用 BETWEEN：

```
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
```

请注意，尽管 BETWEEN 的范围包括在内，但日期默认具有 00:00:00 的时间值。示例查询中唯一有效的 1 月 3 日行是 saletime 为 `1/3/2008 00:00:00` 的行。

Null 条件

在缺少值或值未知时，NULL 条件测试是否存在 null。

语法

```
expression IS [ NOT ] NULL
```

参数

expression

任何表达式（如列）。

IS NULL

当表达式的值为 null 时为 true；当表达式具有一个值时，为 false。

IS NOT NULL

当表达式的值为 null 时为 false；当表达式具有一个值时，为 true。

示例

此示例指示 SALES 表的 QTYSOLD 字段中包含 null 的次数：

```
select count(*) from sales
where qtysold is null;
count
-------
0
(1 row)
```

EXISTS 条件

EXISTS 条件测试子查询中是否存在行，并在子查询返回至少一个行时返回 true。如果指定 NOT，此条件将在子查询未返回任何行时返回 true。

语法

```
[ NOT ] EXISTS (table_subquery)
```

Arguments

EXISTS

当 table_subquery 返回至少一行时，为 true。

NOT EXISTS

当 table_subquery 未返回任何行时，为 true。

table_subquery

计算结果为包含一个或多个列和一个或多个行的表的子查询。

示例

此示例针对具有任何类型的销售的日期返回所有日期标识符，一次返回一个日期：

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
--------
1827
1828
1829
...
```

IN 条件

IN 条件测试一组值或一个子查询中的成员身份值。

语法

```
expression [ NOT ] IN (expr_list | table_subquery)
```

参数

expression

数字、字符或日期时间表达式，针对 expr_list 或 table_subquery 进行计算，必须是与列表或子查询的数据类型兼容的。

expr_list

一个或多个逗号分隔的表达式，或一组或多组逗号分隔的表达式（用括号限定）。

table_subquery

一个子查询，计算结果为具有一行或多行的表，但在其选择列表中限制为一列。

IN | NOT IN

如果表达式是表达式列表或查询的成员，则 IN 将返回 true。如果表达式不是成员，NOT IN 将返回 true。在下列情况下，IN 和 NOT IN 将返回 NULL 并且不会返回任何行：如果 expression 生成 null；或者，如果没有匹配的 expr_list 或 table_subquery 值并且至少一个比较行生成 null。

示例

下列条件仅对列出的值有效：

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

优化大型 IN 列表

为了优化查询性能，包含 10 个以上的值的 IN 列表将在内部作为标量数组计算。少于 10 个值的 IN 列表将作为一系列 OR 谓词计算。SMALLINT、INTEGER、BIGINT、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP 和 TIMESTAMPTZ 数据类型均支持此优化。

查看查询的 EXPLAIN 输出以查看此优化的效果。例如：

```
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)
```

语法

```
comparison_condition
```

```
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

查询嵌套数据

AWS Clean Rooms 提供对关系数据和嵌套数据的 SQL 兼容访问。

在访问嵌套数据时，AWS Clean Rooms 使用点记法和数组下标进行路径导航。它还使 FROM 子句项能够对数组进行迭代并用于非嵌套操作。以下主题介绍了将数组/结构/映射数据类型的使用与路径和数组导航、取消嵌套和联接相结合的不同查询模式。

主题

导航

AWS Clean Rooms 通过 `[...]` 括号和点符号来支持对数组和结构的导航。此外，您还可以使用点记法将导航混合到结构中，使用括号符号将数组混合到结构中。

Example

例如，以下示例查询假定 `c_orders` 数组数据列是一个具有结构的数组，并且属性名为 `o_orderkey`。

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

您可以在所有类型的查询中使用点和括号符号，例如筛选、联接和聚合。您可以在通常存在列引用的查询中使用这些符号。

Example

以下示例使用筛选结果的 SELECT 语句。

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0].o_orderkey IS NOT NULL;
```

Example

以下示例在 GROUP BY 和 ORDER BY 子句中使用括号和点导航：

```
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;
```

取消嵌套查询

为了取消嵌套查询，AWS Clean Rooms 支持对数组进行迭代。它通过使用查询的 FROM 子句导航数组来实现这一点。

Example

使用前面的示例，以下示例对 c_orders 的属性值进行迭代。

```
SELECT o FROM customer_orders_lineitem c, c.c_orders o;
```

取消嵌套语法是 FROM 子句的扩展。在标准 SQL 中，FROM 子句 x （AS） y 表示 y 迭代关系 x 中的每个元组。在这种情况下，x 指的是关系，而 y 指的是关系 x 的别名。同样，使用 FROM 子句项 x （AS） y 进行取消嵌套的语法表示 y 迭代数组表达式 x 中的每个值。在这种情况下，x 是一个数组表达式，而 y 是 x 的别名。

左侧操作数也可以使用点和括号表示法进行常规导航。

Example

在上一个示例中：

- customer_orders_lineitem c 是对 customer_order_lineitem 基表的迭代
- c.c_orders o 是对 c.c_orders array 的迭代

要迭代作为数组中的数组的 o_lineitems 属性，必须添加多个子句。

```
SELECT o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

AWS Clean Rooms 还在使用 AT 关键字迭代数组时支持数组索引。子句 x AS y AT z 迭代数组 x 并生成字段 z，即数组索引。

Example

以下示例演示数组索引的工作原理：

```
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

以下示例对标量数组进行迭代：

```
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

以下示例对多个级别的数组进行迭代。该示例使用多个 unnest 子句来迭代到最内层的数组。f.multi_level_array AS 数组迭代 multi_level_array。数组 AS 元素是对 multi_level_array 中的数组的迭代。

```
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)
```

宽松语义

预设情况下，在导航无效时，嵌套数据值的导航操作返回 null，而不是返回错误。如果嵌套数据值不是对象，或者嵌套数据值是一个对象，但不包含查询中使用的属性名称，则对象导航无效。

Example

例如，以下查询访问嵌套数据列 `c_orders` 中的无效属性名称：

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

如果嵌套数据值不是数组或数组索引超出界限，则数组导航返回 null。

Example

以下查询返回 null，因为 `c_orders[1][1]` 超出了界限。

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

自检类型

嵌套数据列支持返回有关该值的类型和其他类型信息的检查函数。AWS Clean Rooms 支持以下用于嵌套数据列的布尔函数：

- 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

如果输入值为 null，所有这些函数都返回 false。IS_SCALAR、IS_OBJECT 和 IS_ARRAY 是相互排斥的，涵盖除 null 之外的所有可能的值。要推理与数据对应的类型，AWS Clean Rooms 使用 JSON_TYPEOF 函数，该函数返回嵌套数据值的类型（顶级），如以下示例所示：

```
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
```

AWS Clean Rooms SQL 参考的文档历史记录

下表介绍了《 AWS Clean Rooms SQL 参考》的文档版本。

如需对此文档更新的通知，您可以订阅 RSS 源。要订阅 RSS 更新，您必须为当前使用的浏览器启用 RSS 插件。

变更	说明	日期
SQL 命令和 SQL 函数-更新	添加了 JOIN 子句的示例，但集合运算符、CASE 条件表达式和以下函数除外：ANY_VALUE、NVL 和 COALESCE、NULLIF、CAST、CONVERT_TIMEZONE、EXTRACT、MOD、SIGN、CONCAT、FIRST_VALUE 和 LAST_VALUE。	2024年2月28日
SQL 函数 — 更新	AWS Clean Rooms 现在支持以下 SQL 函数：数组、SUPER 和 VARBYTE。现在支持以下数学函数：ACOS、ASIN、ATAN、ATAN2、COT、DEXP、PI、POW、RADIANS 和 SIN。现在支持以下 JSON 函数：CAN_JSON_PARSE、JSON_PARSE 和 JSON_SERIALIZE。	2023 年 10 月 6 日
支持嵌套数据类型	AWS Clean Rooms 现在支持嵌套数据类型。	2023 年 8 月 30 日
SQL 命名规则 — 更新	仅限文档的更改，以明确保留的列名。	2023 年 8 月 16 日

正式发布	《AWS Clean Rooms SQL 参考》现已正式发布。	2023 年 7 月 31 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。