Amazon Kinesis Data Analytics

SQL 参考

Amazon Kinesis Data Analytics: SQL 参考

Amazon Web Services 文档中描述的 Amazon Web Services 服务或功能可能因区域而异。要查看适用于中国区域的差异，请参阅中国的 Amazon Web Services 服务入门。

Table of Contents

Amazon Kinesis Data Analytics

这些区域有：Amazon Kinesis Data Analytics描述了Amazon Kinesis Data Analytics 支持的 SQL 语言元素。该语言基于 SQL:2008 标准，具有一些扩展功能以启用对流数据进行操作。

对于新项目，我们建议你使用 Kinesis Data Analytics Studio 而不是 SQL 应用程序的 Kinesis Data Analytics。Kinesis Data Analytics Studio 将易用性与高级分析功能相结合，使您能够在几分钟内构建复杂的流处理应用程序。

有关开发 Kinesis Data Analytics 应用程序的信息，请参阅Kinesis Data Analytics.

本指南涵盖以下内容：

- 流式处理 SQL 语言元素 (p. 2) – 数据类型 (p. 3), 串流 SQL 操作符 (p. 6), 函数 (p. 69).
- Standard SQL (p. 30) – CREATE CO (p. 30), SELECT (p. 35).
- 用于转换和筛选传入数据的运算符 —WHERE 子句 (p. 57),加入子句 (p. 41),GROUP BY 子句 (p. 55),WINDOW 子句 (滑动窗口) (p. 57).
- 逻辑运算符 (p. 13)— AS、AND、OR 等

流式处理 SQL 语言元素

以下主题讨论了 Amazon Kinesis Data Analytics 中构成其语法和操作基础的语言元素：

主题

标识符

所有标识符最多可包含 128 个字符。标识符可以加上引号（区分大小写），方法是将标识符用双引号 (“) 括起来，也可以不加引号（在存储和查找前都使用隐式大写）。

未加引号的标识符必须以字母或下划线开头，并且必须后跟字母、数字或下划线；字母全部转换为大写字母。

带引号的标识符也可以包含其他标点符号（实际上，除控制字符之外的任何 Unicode 字符：代码 0x0000 到 0x001F）。可以在标识符中包含双引号，方法是使用另一个双引号对其进行转义。

在以下示例中，使用未加引号的标识符创建流，在将流定义存储在目录中之前，该标识符会转换为大写字母。可以使用其大写名称进行引用，也可以使用隐式转换为大写的未加引号的标识符进行引用。

```
-- Create a stream. Stream name specified without quotes,
-- which defaults to uppercase.
CREATE OR REPLACE STREAM ExampleStream (col1 VARCHAR(4));

- example 1: OK, stream name interpreted as uppercase.
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO ExampleStream
    SELECT * FROM SOURCE_SQL_STREAM_001;

- example 2: OK, stream name interpreted as uppercase.
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO examplestream
    SELECT * FROM   customerdata;

- example 3: Ok.
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO EXAMPLESTREAM
    SELECT * FROM   customerdata;

- example 2: Not found. Quoted names are case-sensitive.
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "examplestream"
    SELECT * FROM   customerdata;
```

在 Amazon Kinesis Data Analytics 中创建对象时，它们的名称会被隐式引用，因此很容易创建包含小写字符、空格、短划线或其他标点符号的标识符。如果您在 SQL 语句中引用这些对象，则需要引用它们的名称。

保留字和关键字

某些标识符（称为关键字）如果出现在流式处理 SQL 语句中的特定位置，则具有特殊含义。这些关键字的子集被称为保留字，除非用引号将其用作对象的名称。有关更多信息，请参阅保留字和关键字 (p. 25)。

数据类型

下表汇总了Amazon Kinesis Data Analytics 支持的数据类型。

SQL 数据类型	JSON 数据类型	描述	注意
BIGINT	number	64 位有符号整数	
BINARY	BASE64 编码的字符串	二进制（非字符）数据	子字符串适用于二进制。串联在 BINARY 上不起作用。
BOOLEAN	布尔值	TRUE、FALSE 或 NUL	计算结果为 “真”、“假” 和 “未知”。
CHAR (n)	字符串	固定长度 n 的字符串。也可指定为 CHARACTER	n 必须大于 0 且小于 65535。
DATE	字符串	日期是日历日（年/月/日）。	精度就是一天。范围从最大值（大约 +229（以年为单位）到最小值 -229 不等。
DECIMAL DEC NUMERIC	number	一个固定点，最多 19 个有效数字。	可以用十进制、十进制或数字指定。
DOUBLE DOUBLE PRECISION	number	64 位浮点数	64 位近似值；-1.79E +308 到 1.79E+308。遵循 ISO DOUBLE PRECISION 数据类型，科学计数法中使用 53 位作为数字尾数，表示 15 位精度和 8 字节存储空间。
INTEGER INT	number		32 位带符号整数。范围为 -2147483648 到 2147483647 [2** (31) 到 2** (31)-1]
INTERVAL <timeunit> [TO <timeunit>]	字符串	支持日间隔，不支持年月间隔	允许在日期算术的表达式中使用，但不能用作表或流中列的数据类型。
<timeUnit>	字符串	间隔值的单位	支持的单位为年、月、日、时、分和秒
SMALLINT	number	16 位有符号整数	范围为 -32768 到 32767 [2** (15) 到 2** (15) -1]
REAL	number	32 位浮点数	在 ISO REAL 数据类型之后，24 位用于科学记数法中的数字尾数，

SQL 数据类型	JSON 数据类型	描述	注意
			表示 7 位精度和 4 字节的存储空间。最小值为 -3.40E+38；最大值为 3.40E+38。
TIME	字符串	时间是一天中的时间（小时:分钟:秒）。	其精度是毫秒；其范围是 00:00:00.000 到 23:59:59.999。由于系统时钟以 UTC 运行，因此不考虑存储在 TIME 或 TIMESTAMP 列中的值的时区。 用于存储在 TIME 或 TIMESTAMP 列中的值。
TIMESTAMP	字符串	时间戳是日期和时间的组合。	时间戳值的精度始终为 1 毫秒。它没有特定的时区。由于系统时钟以 UTC 运行，因此不考虑存储在 TIME 或 TIMESTAMP 列中的值的时区。其范围从最大值（大约 +229（以年为单位）到最小值 -229 不等。每个时间戳存储为一个有符号的 64 位整数，其中 0 表示 Unix 时代（1970 年 1 月 1 日凌晨 00:00）。这意味着最大的 TIMESTAMP 值代表 1970 年后的大约 3 亿年，最小值代表 1970 年之前的大约 3 亿年。按照 SQL 标准，时间戳值具有未定义的时区。
TINYINT	number	8 位有符号整数	范围为 -128 到 127
VARBINARY (n)	BASE64 编码的字符串	也可以指定为二进制变量	n 必须大于 0 且小于 65535。
VARCHAR (n)	字符串	也可以指定为字符变化	n 必须大于 0 且小于 65535。

注意

关于字符：

- Amazon Kinesis Data Analytics 仅支持 Java 单字节字符集。
- 不支持隐式类型转换。也就是说，当且仅当字符从相同的字符库中提取并且是 CHARACTER 或 CHARACTER VARIONING 数据类型的值时，字符才可以相互分配。

关于数字：

- 如果数字是
  NUMERIC、DECIMAL、INTEGER、BIGINT、BIGINT、SMALLINT、TINYINT、REAL、BIGINT、SMALLINT、TINGE

以下数据类型集是同义词：

- DECIMAL 和 D
- DUBLE PRECIS
- 字符和字符
- 字符变化或角色变化和 VARCHAR
- 二进制变化和 VARBINARY
- INTEGER 和 IN

- 二进制值（数据类型二进制和二进制变化）始终是相互可比的，并且可以相互分配。

关于日期、时间和时间戳：

- 不支持隐式类型转换（也就是说，只有当赋值的源和目标都是 DATE 类型或两者都是 TIME 类型，或者两者都是 TIMESTAMP 类型时，日期时间值才可以相互分配）。

- Amazon Kinesis Data Analytics 时区始终为 UTC。时间函数，包括Amazon Kinesis Data Analytics 扩展 CURRENT_ROW_TIMESTAMP，以 UTC 返回时间。

数字类型和精度

对于 DECIMAL，我们支持最多 18 位的精度和小数位数。

精度指定列中可以存储的最大十进制位数，包括小数点的右侧和左侧。您可以指定从 1 位数到 18 位数的精度，也可以使用 18 位的默认精度。

小数点右侧可以存储的最大位数。小数位数必须小于或等于精度。您可以指定从 0 位数到 18 位数的缩放比例，也可以使用 0 位的默认比例。

除法规则

假设 p1, s1 是第一个操作数的精度和小数位数，例如 DECIMAL (10,1)。

假设 p2, s2 是第二个操作数的精度和小数位数，例如 DECIMAL (10,3)。

假设 p, s 是结果的精度和小数位数。

假设 d 是结果中的整数位数。然后，结果类型为十进制，如下所示：

`d = p1 - s1 + s2`	D = 10-1 + 3 结果中的整数位数 = 6
s <= MAX (6, s1 + p2 +1)	S <= MAX (6, 1 + 10 + 1) 结果比例 = 14
p = d + s	结果精度 = 18

精度和小数位数上限为其最大值（18，其中小数位数不能大于精度）。

优先级是首先给出至少第一个参数的小数位数 (s >= s1)，然后是足够的整数以表示结果而不会溢出

乘法规则

假设 p1, s1 是第一个操作数 DECIMAL (10,1) 的精度和小数位数。

假设 p2, s2 是第二个操作数 DECIMAL (10,3) 的精度和小数位数。

假设 p, s 是结果的精度和小数位数。

然后，结果类型为十进制，如下所示：

p = p1 + p2	p = 10 + 10 结果精度 = 18
s = s1 + s2	s = 1 + 3 结果比例 = 4

总和或减法规则

类型推断策略，其中调用的结果类型是两个精确数字操作数的十进制和，其中至少有一个操作数是十进制。

假设 p1, s1 是第一个操作数 DECIMAL (10,1) 的精度和小数位数。

假设 p2, s2 是第二个操作数 DECIMAL (10,3) 的精度和小数位数。

假设 p, s 是结果的精度和小数位数，如下所示：

s = max (s1, s2)	s = max (1,3) 结果比例 = 3
p = max (p1-s1, p2-s2) + s + 1	p = max (10-1,10-3) + 3 + 1 结果精度 = 11

s 和 p 的上限为其最大值

串流 SQL 操作符

子查询运算符

在查询和子查询中使用运算符来合并或测试各种属性、属性或关系的数据。

以下主题将介绍可用的运算符，分为以下几类：

- LIKE 模式
- SIMILAR TO

- 日期、时间戳和间隔运算符 (p. 124)
- 逻辑运算符 (p. 13)
  - 三态布尔逻辑
  - 示例

IN 运算符

作为条件测试中的运算符，IN 测试标量或行值在值列表、关系表达式或子查询中的成员资格。

```
Examples:
1. --- IF column IN ('A','B','C')
2. --- IF (col1, col2) IN (
    select a, b from my_table
    )
```

如果在列表、计算关系表达式的结果中或子查询返回的行中找到正在测试的值，则返回 TRUE；否则返回 FALSE。

Note

IN 有不同的含义和用法CREATE FUNCTION (p. 31).

存在运算符

测试关系表达式是否返回任何行；如果返回任何行，则返回 TRUE，否则返回 FALSE。

标量运算符

运算符类型

标量运算符的两大类是：

- 一元法：一元运算符仅对一个操作数进行运算。一元运算符通常以以下格式与其操作数一起出现：

```
operator operand
```

- Binary: 二元运算符对两个操作数进行运算。二元运算符及其操作数以这种格式出现：

```
operand1 operator operand2
```

下面的操作数描述中特别注明了一些使用不同格式的运算符。

如果给一个运算符一个空操作数，则结果几乎总是空值（有关异常，请参阅逻辑运算符主题）。

优先顺序

流式处理 SQL 遵循运算符的通常优先级：

1. 计算带括号的子表达式。
2. 计算一元运算符（例如 + 或-、逻辑 NOT）。
3. 计算乘法和除法（* 和/）。

4. 评估加法、减法（+ 和-）和逻辑组合（AND 和 OR）。

如果其中一个操作数为 NULL，则结果也为 NULL 如果操作数属于不同但可比较的类型，则结果将是精度最高的类型。如果操作数的类型相同，则结果将与操作数的类型相同。例如 5/2 = 2，而不是 2.5，因为 5 和 2 都是整数。

算术运算符

操作符	一元/二进制	描述
+	U	求同
-	U	Negation
+	B	加
-	B	减
*	B	乘
/	B	除

这些运算符中的每一个都按照正常的算术行为工作，但需要注意以下几点：

1. 如果其中一个运算数为 NULL，则结果为 NULL
2. 如果操作数属于不同但可比较的类型，则结果将是精度最高的类型。
3. 如果操作数的类型相同，则结果将与操作数的类型相同。例如 5/2 = 2，而不是 2.5，因为 5 和 2 都是整数。

示例

运算	结果
1 + 1	2
2.0 + 2.0	4.0
3.0 + 2	5.0
5/2	2
5.0/2	2.500000000000
5*2+2	12

字符串运算符

您可以使用字符串运算符进行流式处理 SQL，包括串联和字符串模式比较，以组合和比较字符串。

操作符	一元/二进制	描述	注意
||	B	联接	也适用于二进制类型

操作符	一元/二进制	描述	注意
LIKE	B	字符串模式比较	<string> LIKE <like pattern> [ESCAPE <escape character>]
SIMILAR TO	B	字符串模式比较	<string> SIMILAR TO <similar to pattern> [ESCAPE <escape character>]

联接

此运算符用于连接一个或多个字符串，如下表所示。

运算	结果
'sql'||'stream'	SQLstream
'sql'|" ||'stream'	SQLstream
'sql'||'stream'|'已合并'	SQLstream Incorporated
<col1>||<col2>||<col3>||<col4>	<col1><col2><col3><col4>

喜欢模式

LIKE 将字符串与字符串模式进行比较。在模式中，字符 _（下划线）和%（百分比）具有特殊含义。

模式中的字符	效果
_	匹配任何单个字符
%	匹配任何子字符串，包括空字符串
<any other character>	仅匹配完全相同的字符

如果任一操作数为 NULL，则 LIKE 运算的结果为未知。

要显式匹配字符串中的特殊字符，必须使用 ESCAPE 子句指定转义字符。然后，转义字符必须位于模式中的特殊字符之前。下表列出了示例。

运算	结果
'a'就像'a'	TRUE
'a'就像'A'	FALSE
'a'就像'b'	FALSE
'ab'比如'a_'	TRUE
'ab'比如'a%'	TRUE

运算	结果
'ab'就像 'a\ _' ESCAPE '\'	FALSE
'ab'就像 'a\ %' ESCAPE '\'	FALSE
'a_'就像 'a\ _' ESCAPE '\'	TRUE
'a%' 就像 'a\ %' ESCAPE '\'	TRUE
'a'就像'a_'	FALSE
'a'就像'a%'	TRUE
'abcd'就像 'a_'	FALSE
'abcd'比如'a%'	TRUE
“喜欢”	TRUE
'1a' 就像 '_a'	TRUE
'123axyz'比如'%a%'	TRUE
'123axyz'比如'_%_a%_ '	TRUE

SIMILAR TO

SIMILAR TO 将字符串与模式进行比较。它很像 LIKE 运算符，但功能更强大，因为模式是正则表达式。

在下面的 SIMILAR TO 表中，seq 表示显式指定的字符的任何序列（如 '13aq'）。用于匹配的非字母数字字符前面必须有一个在 SIMILAR TO 语句中明确声明的转义字符，例如 '13aq\! ' SIMILAR TO '13aq\!24br\!% ESCAPE '\'（此语句为 TRUE）。

当指定范围时，例如在图案中使用短划线，则使用当前的排序顺序。典型范围为 0-9 和 a-z。PostgreSQL 提供了模式匹配的典型讨论（包括范围）。

当一行需要多次比较时，将首先匹配可以匹配的最内层模式，然后是 “下一个最内层”，依此类推。

在应用周围运算之前，会先计算括号内的表达式和匹配运算，同样按照最先的优先顺序进行计算。

分隔符	模式中的字符	效果	规则 ID
圆括号 ()	（ seq ）	为 seq 分组（用于定义模式表达式的优先顺序）	1
方括号 []	[seq]	匹配序列中的任何单个字符	2
脱字符号或抑扬符	[^seq]	匹配序列中不存在的任何单个字符	3
	[seq ^ seq]	匹配 seq 中的任意单个字符，而不匹配 seq 中的任何单个字符	4
破折号	<character1>-<character2>	指定字符 1 和字符 2 之间的字符范围	5

分隔符	模式中的字符	效果	规则 ID
		（使用一些已知序列，例如 1-9 或 a-z）	
酒吧	[seq seq]	匹配 seq 或 seq	6
星号	seq*	匹配 seq 的零或多个重复项	7
加	seq+	匹配 seq 的一个或多个重复项	8
支架	seq{<number>}	精确匹配 seq 的重复数字	9
	seq{<low number>,<high number>}	将 seq 的低次数或更多重复次数匹配到最大值	10
问号	seq?	匹配 seq 的零或一个实例	11
下划线	_	匹配任何单个字符	12
百分之	%	匹配任何子字符串，包括空字符串	13
字符	<any other character>	仅匹配完全相同的字符	14
NULL	NULL	如果任一操作数为 NULL，则 SIMILAR TO 运算的结果为 UNKNOWN。	15
非字母数字	特殊字符	要显式匹配字符串中的特殊字符， 该特殊字符前面必须有一个使用定义的转义字符 在模式末尾指定的 ESCAPE 子句。	16

下表列出了示例。

运算	结果	规则
'a'类似于'a'	TRUE	14
'a'类似于'A'	FALSE	14
'a'类似于'b'	FALSE	14
'ab'类似于'a_'	TRUE	12
'ab'类似于'a%'	TRUE	13

运算	结果	规则
'a'类似于'a_'	FALSE	12 和 14
'a'类似于'a%'	TRUE	13
'abcd'类似于'a_'	FALSE	12
'abcd'类似于'a%'	TRUE	13
“类似于”	TRUE	14
'1a'类似于'_a'	TRUE	12
'123axyz'类似于”	TRUE	14
'123axyz'类似于 '_%_a%_'	TRUE	13 和 12
'xy'类似于' (xy) '	TRUE	1
'abd' 类似于 '[ab] [bcde] d'	TRUE	2
'bdd'类似于 '[ab] [bcde] d'	TRUE	2
'abd' 类似于 '[ab] d'	FALSE	2
'cd'类似于 '[a-e] d'	TRUE	2
'cd'类似于 '[a-e^c] d'	FALSE	4
'cd'类似于 '[^ (a-e)] d'	无效	
'yd' 类似于 '[^ (a-e)] d'	无效	
'amy'类似于'amyfred'	TRUE	6
'fred'类似于'amyfred'	TRUE	6
'mike'类似于'amyfred'	FALSE	6
'acd'类似于'ab*c+d'	TRUE	7 & 8
'acccd'类似于'ab*c+d'	TRUE	7 & 8
'abd'类似于'ab*c+d'	FALSE	7 & 8
'aabc'类似于'ab*c+d'	FALSE	
'abb'类似于 'a (b {3}) '	FALSE	9
'abbb'类似于 'a (b {3}) '	TRUE	9
'abbbb'类似于 'a (b {3}) '	FALSE	9
'abbbb'类似于 'ab {3,6} '	TRUE	10
'abbbbbbb'类似于 'ab {3,6} '	FALSE	10
“类似于 'ab ? '	FALSE	11
“类似于 '(ab) ? '	TRUE	11
'a'类似于 'ab ? '	TRUE	11

运算	结果	规则
'a'类似于 '(ab) ? '	FALSE	11
'a'类似于 'a (b?) '	TRUE	11
'ab'类似于 'ab ? '	TRUE	11
'ab'类似于 'a (b?) '	TRUE	11
'abb'类似于 'ab ? '	FALSE	11
'ab'类似于 'a\ _' ESCAPE '\'	FALSE	16
'ab'类似于 'a\ %' ESCAPE '\'	FALSE	16
'a_'类似于 'a\ _' ESCAPE '\'	TRUE	16
'a%' 与 'a\ %' 类似 ESCAPE '\'	TRUE	16
'a (b {3}) '类似于 'a (b {3}) '	FALSE	16
'a (b {3}) '类似于 'a\ (b\ {3\}\) 'ESCAPE '\'	TRUE	16

逻辑运算符

逻辑运算符允许您建立条件并测试其结果。

操作符	一元/二进制	描述	运算元
NOT	U	逻辑非	布尔值
AND	B	和” 和	布尔值
或	B	或 DIST 或	布尔值
IS	B	逻辑断言	布尔值
不是未知的	U	否定未知比较： <expr> IS NOT UNKNOWN	布尔值
IS NULL	U	Null 比较： <expr> IS NULL	任何
IS NOT NULL	U	否定空比较： <expr> IS NOT NULL	任何
=	B	平等	任何
!=	B	不平等	任何
<>	B	不平等	任何
>	B	大于	有序类型（数字、字符串、日期、时间）

操作符	一元/二进制	描述	运算元
>=	B	大于或等于（不小于）	ORDER BY
<	B	小于	ORDER BY
<=	B	小于或等于（不大于）	ORDER BY
BETWEEN	三元	范围对比： col1 介于 expr1 和 expr2 之间	ORDER BY
IS DISTINCT FROM	B	区别	任何
没有区别	B	非 “或” 区别”	任何

三态布尔逻辑

SQL 布尔值有三种可能的状态，而不是通常的两种状态：TRUE、FALSE 和 UNKNOWN，最后一个等同于布尔值 NULL。TRUE 和 FALSE 操作数通常根据正常的二态布尔逻辑起作用，但是在将它们与 UNKNOWN 操作数配对时适用其他规则，如下表所示。

> **Note**
>
> UNKOWN 代表 “可能是真的，也许是错误的”，或者换句话说，“不一定是真的，也不是绝对是错误的”。这种理解可能有助于你阐明为什么表中的某些表达式会像它们一样求值。

否定（不是）

运算	结果
TRUE”	FALSE
非 FALSE	TRUE
非 UNKNE	UNKNOWN

连词（和）

运算	结果
TRUE	TRUE
TRUE 和 FAL	FALSE
TRUE” 和 “	UNKNOWN
FALSE 和 TR	FALSE
FALSE 和 FAL	FALSE
FALSE, NK	FALSE
UNKNKNOWN	UNKNOWN
UNKNKNOWN	FALSE
未知和未知	UNKNOWN

分离（OR）

运算	结果
TRUE	TRUE
TRUE 或 FAL	TRUE
TRUE 或 UNK	TRUE
FALSE 或 TR	TRUE
FALSE 或 FAL	FALSE
FALSE 或 UNK	UNKNOWN
UNKNKNOWN	TRUE
UNKNOWN 或	UNKNOWN
未知或未知	UNKNOWN

断言 (IS)

运算	结果
TRUE”	TRUE
TRUE BE,	FALSE
TRUE” 或 “	FALSE
FALSE BE,	FALSE
FALSE BE,	TRUE
FALSE” 为空	FALSE
UNKNKNOWN	FALSE
UNKNKNOWN	FALSE
未知是未知	TRUE

不是未知的

运算	结果
TRUE 不是未知的	TRUE
FALSE 不是未知数	TRUE
未知不是未知	FALSE

IS NOT UNKNOWN 本身就是一个特殊的运算符。表达式 “x 不是未知” 等同于 “（x 是真的）或（x 是假的）”，而不是 “x 是（不是未知）”。因此，在上表中替换：

x	运算	结果		在“（x 为真）或（x 为假）”中替换 x 的结果
TRUE	TRUE 不是未知的	TRUE	变成	“（真是真）或（真是假）”— 因此是真的
FALSE	FALSE 不是未知数	TRUE	变成	“（假是真）或（假是假）”— 因此是真的
UNKNOWN	未知不是未知	FALSE	变成	“（未知是真的）或（未知是假的）”— 因此是错误的， 因为未知既不是真的，也不是假的

由于 IS NOT UNKNOWN 是一个特殊运算符，因此上面的运算不能围绕 IS 这个词传递：

运算	结果
非 UNKNKNOWN	FALSE
非 UNKNKNOWN	FALSE
不是未知就是未知	TRUE

为空, 非 Null

运算	结果
UNKNKNOWN	TRUE
未知不为空	FALSE
NULL 为空	TRUE
NULL 不是 NULL	FALSE

不同于且与之没有区别

运算	结果
未知与真实截然不同	TRUE
未知与假不同	TRUE
未知与未知不同	FALSE
未知与真实没有区别	FALSE
未知与假没有区别	FALSE
未知与未知没有区别	TRUE

通常，“x IS DISTINCT FROM y”类似于“x <> y”，在 x 或 y（而非二者）为 NULL 时也为 true 的情况除外。DISTINCT FROM 与相同值相反，后者的通常含义是值（真、假或未知）与自身相同，并且与其他所有值不同。IS 和 IS NOT 运算符以一种特殊的方式处理 UNKOWN，因为它代表 “可能是真的，也许是假的”。

其他逻辑运算符

对于所有其他运算符，传递 NULL 或 UNKNOWN 操作数将导致结果为 UNKNOWN（与 NULL 相同）。

示例

运算	结果
TRUE 并强制转换（空值为布尔值）	UNKNOWN
FALSE 并强制转换（空值为布尔值）	FALSE
1 > 2	FALSE
1 < 2	TRUE
'foo' = 'bar'	FALSE
'foo' <> 'bar'	TRUE
'foo' <= 'bar'	FALSE
'foo' <= 'bar'	TRUE
介于 1 和 5 之间的 3	TRUE
介于 3 和 5 之间的 1	FALSE
3 和 5 之间的 3 和 5	TRUE
5 介于 3 和 5 之间	TRUE
1 不同于 1.0	FALSE
CAST（NULL 作为整数）与强制转换（NULL 作为整数）没有区别	TRUE

表达式和文字

值表达式

值表达式由以下语法定义：

```
value-expression := <character-expression > | <number-expression> | <datetime-expression> |
 <interval-expression> | <boolean-expression>
```

字符（字符串）表达式

字符表达式由以下语法定义：

```
character-expression := <character-literal>
                  | <character-expression> || <character-expression>
                  | <character-function> ( <parameters> )
```

```
 character-literal  := <quote> { <character> }* <quote>
 string-literal     := <quote> { <character> }* <quote>
 character-function   :=   CAST | COALESCE | CURRENT_PATH
                  | FIRST_VALUE  | INITCAP | LAST_VALUE
                  | LOWER | MAX | MIN | NULLIF
                  | OVERLAY | SUBSTRING| SYSTEM_USER
                  | TRIM | UPPER
                  | <user-defined-function>
```

请注意，Amazon Kinesis Data Analytics 直播 SQL 支持 unicode 字符文字，例如 u&'foo'。与使用正则文字一样，你可以转义这些文字中的单引号，例如 u&'can “t'。与普通文字不同，你可以使用 unicode 转义：例如，u&'\ 0009' 是一个仅由制表符组成的字符串。你可以用另一个\ 来转义\，比如 u&'back\\ slash '。Amazon Kinesis Data Analytics 还支持备用转义字符，例如 u&'！ 0009！！' uescape '！' 是一个制表符。

数字表达式

数字表达式由以下语法定义：

```
number-expression := <number-literal>
                  | <number-unary-oper> <number-expression>
                  | <number-expression> <number-operator> <number-expression>
                  | <number-function> [ ( <parameters> ) ]
 number-literal :=   <UNSIGNED_INTEGER_LITERAL> | <DECIMAL_NUMERIC_LITERAL>
                        | <APPROX_NUMERIC_LITERAL>
```

```
--Note: An <APPROX_NUMERIC_LITERAL> is a number in scientific notation, such as with an
--exponent, such as 1e2 or -1.5E-6.
number-unary-oper := + | -
 number-operator      :=   + | - | / | *

 number-function      :=   ABS | AVG | CAST | CEIL
                         | CEILING | CHAR_LENGTH
                         | CHARACTER_LENGTH | COALESCE
                         | COUNT | EXP | EXTRACT
                         | FIRST_VALUE
                         | FLOOR | LAST_VALUE
                         | LN | LOG10
                         | MAX | MIN  | MOD
                         | NULLIF
                         | POSITION | POWER
                         | SUM| <user-defined-function>
```

日期/时间表达式

日期/时间表达式由以下语法定义：

```
datetime-expression := <datetime-literal>
                         | <datetime-expression> [ + | - ] <number-expression>
                         | <datetime-function> [ ( <parameters> ) ]
 datetime-literal    :=  <left_brace> { <character-literal> } *  <right_brace>
                         |  <DATE> { <character-literal> } *
                         |  <TIME> { <character-literal> } *
                         |  <TIMESTAMP> { <character-literal> } *
 datetime-function   :=    CAST | CEIL | CEILING
                         | CURRENT_DATE | CURRENT_ROW_TIMESTAMP
                         | CURRENT_ROW_TIMESTAMP
                         | FIRST_VALUE| FLOOR
                         | LAST_VALUE | LOCALTIME
                         | LOCALTIMESTAMP | MAX | MIN
                         | NULLIF | ROWTIME
                         | <user-defined-function>
```

```
 <time unit>          :=  YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
```

间隔表达式

间隔表达式由以下语法定义：

```
interval-expression := <interval-literal>
                          | <interval-function>
 interval-literal    :=    <INTERVAL> ( <MINUS> | <PLUS> ) <QUOTED_STRING>
 <IntervalQualifier>
 IntervalQualifier  :=  <YEAR> ( <UNSIGNED_INTEGER_LITERAL> )
                          | <YEAR> ( <UNSIGNED_INTEGER_LITERAL> ) <TO> <MONTH>
                          |  <MONTH> [ ( <UNSIGNED_INTEGER_LITERAL> ) ]
                          |  <DAY> [ (  <UNSIGNED_INTEGER_LITERAL> )  ]
                          |  <DAY> [ (  <UNSIGNED_INTEGER_LITERAL> )  ] <TO>
                                   { <HOUR> | <MINUTE> | <SECOND>
 [ ( <UNSIGNED_INTEGER_LITERAL> ) ] }
                          |  <HOUR> [ (  <UNSIGNED_INTEGER_LITERAL> ) ]
                          |  <HOUR> [ (  <UNSIGNED_INTEGER_LITERAL> ) ] <TO>
                                   { <MINUTE> | <SECOND> [ <UNSIGNED_INTEGER_LITERAL> ] }
                          |  <MINUTE> [ ( <UNSIGNED_INTEGER_LITERAL> )  ]
                          |  <MINUTE> [ ( <UNSIGNED_INTEGER_LITERAL> )  ] <TO>
                                    <SECOND> [ ( <UNSIGNED_INTEGER_LITERAL> ) ]
                          |  <SECOND> [ ( <UNSIGNED_INTEGER_LITERAL> )  ]
 interval-function   :=    ABS  | CAST | FIRST_VALUE
                          | LAST_VALUE | MAX | MIN
                          | NULLIF| <user-defined-function>
```

布尔值表达式

布尔表达式由以下语法定义：

```
boolean-expression := <boolean-literal>
                  | <boolean-expression> <boolean-operator> <boolean-expression>
                  | <boolean-unary-oper> <boolean-expression>
                        | <boolean-function> ( <parameters> )
                        | ( <boolean-expression> )
 boolean-literal    :=  TRUE | FALSE
 boolean-operator   :=  AND | OR
 boolean-unary-oper :=  NOT
 boolean-function   :=    CAST | FIRST_VALUE | LAST_VALUE
                        | NULLIF | <user-defined-function>
```

单调表达式和运算符

由于 Amazon Kinesis Data Analytics 查询在无限的行流上运行，因此某些操作只有在对这些流有所了解的情况下才有可能。

例如，给定订单流，有意义的做法是要求按日和产品汇总订单的直播（因为日期在增加），但不要求按产品和配送状态汇总订单。我们永远无法完成 Widget X 到俄勒冈的摘要，因为我们从来没有看到 Widget 到俄勒冈州的 “最后” 订单。

这个由特定列或表达式对流进行排序的属性称为单调性。

一些与时间相关的定义：

- 单调。如果表达式是升序或降序的，则该表达式是单调的。等效的措辞是 “不递减或不递增”。
- 升序。如果给定行的 e 值始终大于或等于前一行中的值，则表达式 e 在流中升序。
- 降序。如果给定行的 e 值始终小于或等于前一行中的值，则表达式 e 在流中降序。

- 严格升序。如果给定行的 e 值始终大于前一行中的值，则表达式 e 在流中严格按升序排列。
- 严格降序。如果给定行的 e 值始终小于前一行中的值，则表达式 e 在流中严格降序。
- 常量。如果给定行的 e 值始终等于前一行中的值，则表达式 e 在流中是常量。

请注意，根据这个定义，常量表达式被视为单调表达式。

单调列

ROWTIME 系统列按升序排列。ROWTIME 列不是严格按升序排列的：连续行具有相同的时间戳是可以接受的。

Amazon Kinesis Data Analytics 可防止客户端在时间戳小于其写入流的前一行的时间戳的流中插入一行。Amazon Kinesis Data Analytics 还可确保，如果多个客户端在同一个流中插入行，则合并这些行，以使 ROWTIME 列按升序排列。

显然，例如，断言 orderID 列是升序的；或者断言 orderID 与排序顺序相差不超过 100 行会很有用。但是，当前版本不支持声明的排序键。

单调表达式

如果 Amazon Kinesis Data Analytics 知道其参数是单调的，则可以推断出表达式是单调的。（另请参阅单调函数 (p. 191)。）

另一个定义：

单调函数或运算符

如果将函数或运算符应用于严格递增的值序列时，它会产生单调的结果序列，则该函数或运算符是单调的。

例如，FLOOR 函数应用于升序输入 {1.5、3、5、5.8、6.3} 时，会产生 {1、3、5、5、6}。请注意，输入严格按升序排列，但输出仅为升序（包括重复值）。

推断单调性的规则

Amazon Kinesis Data Analytics 要求一个或多个分组表达式有效，流式传输 GROUP BY 语句才有效。在其他情况下，如果 Amazon Kinesis Data Analytics 知道单调性，它可能能够更高效地运行；例如，如果它知道某个特定的密钥再也不会出现在直播中，它可能能够从窗口汇总总数表中删除条目。

为了以这种方式利用单调性，Amazon Kinesis Data Analytics 使用了一组规则来推断表达式的单调性。以下是推断单调性的规则：

表达式	单调性
c	常量
FLOOR (p. 138)(m)	与 m 相同，但不严格
天花板/天花板 (p. 137)(m)	与 m 相同，但不严格
天花板/天花板 (p. 137)（m 到 TimeUnit）	与 m 相同，但不严格
FLOOR (p. 138)（m 到 TimeUnit）	与 m 相同，但不严格
SUBSTRING (p. 197)（m 表示从 0 表示 c）	与 m 相同，但不严格
+ m	和 m 一样

表达式	单调性
-m	m 反向
m + c c + m	和 m 一样
m1 + m2	如果 m1 和 m2 的方向相同，则与 m1 相同； 否则不是单调的
c-m	m 反向
m * c c * m	如果 c 为正值，则与 m 相同； m 的反向为 c 为负；常量 (0) c 为 0
c/m	如果 m 始终为正数或始终为负数，且 c 和 m 的符号相同，则与 m 相同； 如果 m 始终为正数或始终为负数，且 c 和 m 的符号不同，则与 m 相反； 否则不是单调的
	常量
LOCALTIME (p. 134) LOCALTIMESTAMP (p. 135) 当前_ROW_TIMESTAMP (p. 132) CURRENT_DATE (p. 132)	升序

在整个表中，c 是一个常量，m（也是 m1 和 m2）是一个单调表达式。

子句

引用者：

- GROUP 子句：HAVING 子句 (p. 55),WHERE 子句 (p. 57)，以及加入子句 (p. 41).（另请参阅 SELECT 图表及其SELECT 子句 (p. 37)。）
- 删除

条件是任何类型为 BOOLEAN 的值表达式，例如以下示例：

- 2<4
- TRUE
- FALSE
- expr_17 为空
- NOT expr_19 IS NULL AND expr_23 < expr>29
- expr_17 IS NULL OR (NOT expr_19 IS NULL AND expr_23 < expr>29)

临时谓词

下表以图形方式显示了标准 SQL 支持的时间谓词以及 Amazon Kinesis Data Analytics 支持的 SQL 标准扩展。它显示了每个谓词所涵盖的关系。每个关系均表示为时间间隔上限和下限，并具有组合意义 upperInterval predicate lowerInterval evaluates to TRUE。前 7 个谓词是标准 SQL。以粗体文本显示的最后 10 个谓词是 Amazon Kinesis Data Analytics SQL 标准的扩展。

谓词	承保关系
CONTAINS	
OVERLAPS	
EQUALS	
先于	
继任	
紧随其后	
立刻成功	
LEADS	
LAGS	
STRICTLY CONTAINS	
STRICTLY OVERLAPS	
STRICTLY PRECEDES	
STRICTLY SUCCEEDS	
STRICTLY LEADS	
STRICTLY LAGS	
IMMEDIATELY LEADS	
IMMEDIATELY LAGS	

为了启用简洁表达式，Amazon Kinesis Data Analytics 还支持以下扩展：

- 可选 PERIOD 关键字 — 可以省略 PERIOD 关键字。
- 紧凑链接-如果其中两个谓词背靠背出现，以 AND 分隔，则可以省略 AND，前提是第一个谓词的右间隔与第二个谓词的左间隔相同。
- TSDIFF — 此函数采用两个 TIMESTAMP 参数并返回它们的差值（以毫秒为单位）。

例如，您可以编写以下表达式：

```
  PERIOD (s1,e1) PRECEDES PERIOD(s2,e2)
  AND PERIOD(s2, e2) PRECEDES PERIOD(s3,e3)
```

简而言之，如下所示：

```
(s1,e1) PRECEDES (s2,e2) PRECEDES PERIOD(s3,e3)
```

以下简明表达式：

```
TSDIFF(s,e)
```

意思如下：

```
CAST((e - s) SECOND(10, 3) * 1000 AS BIGINT)
```

最后，标准 SQL 允许 CONTAINS 谓词将单个 TIMESTAMP 作为其右边的参数。例如，以下表达式：

```
PERIOD(s, e) CONTAINS t
```

等效于以下内容：

```
s <= t AND t < e
```

语法

时间谓词被集成到一个新的布尔值表达式中：

```
<period-expression> :=
  <left-period> <half-period-predicate> <right-period>

<half-period-predicate> :=
  <period-predicate> [ <left-period> <half-period-predicate> ]

<period-predicate> :=
   EQUALS
 | [ STRICTLY ] CONTAINS
 | [ STRICTLY ] OVERLAPS
 | [ STRICTLY | IMMEDIATELY ] PRECEDES
 | [ STRICTLY | IMMEDIATELY ] SUCCEEDS
 | [ STRICTLY | IMMEDIATELY ] LEADS
 | [ STRICTLY | IMMEDIATELY ] LAGS

<left-period> := <bounded-period>

<right-period> := <bounded-period> | <timestamp-expression>
```

```
<bounded-period> := [ PERIOD ] ( <start-time>, <end-time> )

<start-time> := <timestamp-expression>

<end-time> := <timestamp-expression>

<timestamp-expression> :=
  an expression which evaluates to a TIMESTAMP value

where <right-period> may evaluate to a <timestamp-expression> only if
the immediately preceding <period-predicate> is [ STRICTLY ] CONTAINS
```

以下内置函数支持此布尔表达式：

```
BIGINT tsdiff( startTime TIMESTAMP, endTime TIMESTAMP )
```

返回 (endTime-startTime) 的值，以毫秒为单位。

```
```

示例

以下示例代码会在空调开启时窗户打开时记录警报：

```
create or replace pump alarmPump stopped as
  insert into alarmStream( houseID, roomID, alarmTime, alarmMessage )
    select stream w.houseID, w.roomID, current_timestamp,
                  'Window open while air conditioner is on.'
    from
        windowIsOpenEvents over (range interval '1' minute preceding) w
    join
        acIsOnEvents over (range interval '1' minute preceding) h
    on w.houseID = h.houseID
    where (h.startTime, h.endTime) overlaps (w.startTime, w.endTime);
```

使用案例示例

当两个人尝试在两个不同的地点同时使用同一张信用卡时，以下查询使用时间谓词发出欺诈警报：

```
create pump creditCardFraudPump stopped as
 insert into alarmStream
  select stream
    current_timestamp, creditCardNumber, registerID1, registerID2
  from transactionsPerCreditCard
  where registerID1 <> registerID2
  and (startTime1, endTime1) overlaps (startTime2, endTime2)
;
```

前面的代码示例使用具有以下数据集的输入流：

```
(current_timestamp  TIMESTAMP,
  creditCardNumber  VARCHAR(16),
  registerID1       VARCHAR(16),
  registerID2       VARCHAR(16),
  startTime1        TIMESTAMP,
  endTime1          TIMESTAMP,
  startTime2        TIMESTAMP,
```

```
endTime2          TIMESTAMP)
```

保留字和关键字

保留字

以下是自 5.0.1 版本起 Amazon Kinesis Data Analytics 应用程序中的保留字列表。

A		
ABS	ALL	ALLOCATE
允许	ALTER	ANALYZE
AND	ANY	APPROXIMATE_ARR
ARE	ARRAY	AS
ASENSITIVE	ASYMMETRIC	AT
ATOMIC	AUTHORIZATION	AVG
B		
BEGIN	BETWEEN	BIGINT
BINARY	BIT	BLOB
BOOLEAN	BOTH	BY
C		
CALL	CALLED	CARDINALITY
CASCADED	CASE	CAST
CEIL	CEILING	CHAR
CHARACTER	CHARACTER_LENGTH	CHAR_LENGTH
CHECK	CHECKPOINT	CLOB
CLOSE	CLUSTERED	COALESCE
COLLATE	COLLECT	COLUMN
COMMIT	CONDITION	CONNECT
CONSTRAINT	CONVERT	CORR
CORRESPONDING	COUNT	COVAR_POP
COVAR_SAMP	创建	CROSS
CUBE	CUME_DIST	CURRENT
当前_目录	CURRENT_DATE	CURRENT_DEFAULT
CURRENT_PATH	CURRENT_ROLE	CURRENT_SCHEMA

CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_TRANSF
CURRENT_USER	CURSOR	CYCLE
D		
DATE	DAY	DEALLOCATE
DEC	DECIMAL	DECLARE
DEFAULT	删除	DENSE_RANK
DEREF	DESCRIBE	DETERMINISTIC
不允许	DISCONNECT	DISTINCT
DOUBLE	DROP	DYNAMIC
E		
EACH	ELEMENT	ELSE
END	END-EXEC	ESCAPE
EVERY	EXCEPT	EXEC
EXECUTE	EXISTS	EXP
EXPLAIN	EXP_AVG	EXTERNAL
EXTRACT		
F		
FALSE	FETCH	FILTER
FIRST_VALUE	FLOAT	FLOOR
FOR	FOREIGN	FREE
FROM	FULL	FUNCTION
FUSION		
G		
GET	GLOBAL	GRANT
GROUP	GROUPING	
H		
HAVING	HOLD	HOUR
I		
IDENTITY	忽略	进口
IN	INDICATOR	INITCAP
INNER	INOUT	INSENSITIVE
INSERT	INT	INTEGER

INTERSECT	INTERSECTION	INTERVAL
INTO	IS	
J		
JOIN		
L		
LANGUAGE	LARGE	LAST_VALUE
LATERAL	LEADING	LEFT
LIKE	LIMIT	LN
LOCAL	LOCALTIME	LOCALTIMESTAMP
LOWER		
M		
MATCH	MAX	MEMBER
MERGE	METHOD	MIN
MINUTE	MOD	MODIFIES
MODULE	MONTH	MULTISET
否		
NATIONAL	NATURAL	NCHAR
NCLOB	NEW	否
节点	NONE	NORMALIZE
NOT	NTH_VALUE	NULL
NULLIF	NUMERIC	
O		
OCTET_LENGTH	OF	OLD
开	ONLY	打开
或	ORDER	OUT
OUTER	OVER	OVERLAPS
OVERLAY		
P		
PARAMETER	PARTITION	PARTITION_ID
PARTITION_KEY	PERCENTILE_CONT	PERCENTILE_DISC
PERCENT_RANK	POSITION	POWER
PRECISION	PREPARE	PRIMARY

PROCEDURE		
R		
RANGE	RANK	READS
REAL	RECURSIVE	REF
REFERENCES	REFERENCING	REGR_AVGX
REGR_AVGY	REGR_COUNT	REGR_INTERCEPT
REGR_R2	REGR_SLOPE	REGR_SXX
REGR_SXY	RELEASE	尊重
RESULT	RETURN	RETURNS
REVOKE	RIGHT	ROLLBACK
ROLLUP	ROW	ROWS
ROWTIME	ROW_NUMBER	
S		
SAVEPOINT	SCOPE	SCROLL
SEARCH	SECOND	SELECT
SENSITIVE	SEQUENCE_NUMBER	SESSION_USER
SET	SHARD_ID	SIMILAR
SMALLINT	SOME	SORT
SPECIFIC	SQL	SQLEXCEPTION
SQLSTATE	SQLWARNING	SQRT
START	STATIC	STDDEV
STDDEV_POP	STDDEV_SAMP	STOP
流	SUBMULTISET	SUBSTRING
SUM	SYMMETRIC	SYSTEM
SYSTEM_USER		
T		
TABLE	TABLESAMPLE	THEN
TIME	TIMESTAMP	TIMEZONE_HOUR
TIMEZONE_MINUTE	TINYINT	TO
TRAILING	TRANSLATE	TRANSLATION
TREAT	TRIGGER	TRIM
TRUE	TRUNCATE	

U		
UESCAPE	联合	UNIQUE
UNKNOWN	UNNEST	更新
UPPER	USER	USING
V		
VALUE	VALUES	VARBINARY
VARCHAR	VARYING	VAR_POP
VAR_SAMP		
W		
WHEN	WHENEVER	WHERE
WIDTH_BUCKET	WINDOW	WITH
WITHIN	WITHOUT	
Y		
YEAR		

Standard SQL

以下主题讨论标准 SQL 运算符：

主题

CREATE CO

您可以在Amazon Kinesis Data Analytics 中使用以下 CREATE 语句：

创建流

CREATE STREAM 语句创建一个（本地）流。流的名称必须与同一 schema 中任何其他流的名称不同。最好添加流的描述。

与表一样，流也有列，您可以在 CREATE STREAM 语句中为这些列指定数据类型。它们应映射到您正在为其创建流的数据源。对于 column_name，任何有效的非保留 SQL 名称都可用。列值不能为空。

- 如果流已存在，则指定 OR REPLACE 会重新创建流，从而允许对现有对象进行定义更改，无需先使用 DRP 命令即可隐式删除该流。在已经有数据正在运行的流上使用 CREATE OR REPLACE 会终止直播并丢失所有历史记录。
- 只有在指定了 OR REPLACE 时才能指定重命名。
- 有关 type_specifical 中类型和值的完整列表，例如 TIMESTAMP、INTEGER 或 varchar (2)，请参阅 Amazon Kinesis Data Analytics SQL 参考指南中的主题 Amazon Kinesis Data Analytics 类型。
- 对于 option_value，可以使用任何字符串。

未解析日志数据的简单流

```
CREATE OR REPLACE STREAM logStream (
    source  VARCHAR(20),
    message VARCHAR(3072))
DESCRIPTION 'Head of webwatcher stream processing';
```

直播采集来自智能出行系统管道的传感器数据

```
CREATE OR REPLACE STREAM "LaneData" (
    -- ROWTIME is time at which sensor data collected
    LDS_ID  INTEGER,        -- loop-detector ID
    LNAME   VARCHAR(12),
    LNUM    VARCHAR(4),
    OCC     SMALLINT,
```

```
    VOL     SMALLINT,
    SPEED   DECIMAL(4,2)
) DESCRIPTION 'Conditioned LaneData for analysis queries';
```

从电子商务管道直播采集订单数据

```
CREATE OR REPLACE STREAM "OrderData" (
    "key_order"    BIGINT NOT NULL,
    "key_user"     BIGINT,
    "country"      SMALLINT,
    "key_product"  INTEGER,
    "quantity"     SMALLINT,
    "eur"          DECIMAL(19,5),
    "usd"          DECIMAL(19,5)
) DESCRIPTION 'conditioned order data, ready for analysis';
```

CREATE FUNCTION

Amazon Kinesis Data Analytics 提供了一函数 (p. 69)，还允许用户通过用户定义的函数（UDF）扩展其功能。Amazon Kinesis Data Analytics 仅支持在 SQL 中定义的 UDF。

用户定义的函数可以使用完全限定名称调用，也可以仅使用函数名调用。

传递给用户定义函数或转换的值（或从）返回的数据类型必须与相应的参数定义完全相同。换句话说，在向用户定义函数传递参数（或从中返回值）时不允许隐式转换。

用户定义函数 (UDF)

用户定义的函数可以实现复杂的计算，获取零个或多个标量参数并返回标量结果。UDF 的运行方式类似于内置函数，例如 FLOOR () 或 LOWER ()。对于 SQL 语句中每次出现用户定义的函数，每行调用一次 UDF，其标量参数：该行中的常量或列值。

语法

```
 CREATE FUNCTION ''<function_name>'' ( ''<parameter_list>'' )
  RETURNS ''<data type>''
  LANGUAGE SQL
  [ SPECIFIC ''<specific_function_name>''  | [NOT] DETERMINISTIC ]
  CONTAINS SQL
  [ READS SQL DATA ]
  [ MODIFIES SQL DATA ]
  [ RETURNS NULL ON NULL INPUT | CALLED ON NULL INPUT ]
  RETURN ''<SQL-defined function body>''
```

SPECIAL 分配一个在应用程序中是唯一的特定函数名称。请注意，常规函数名称不必是唯一的（两个或多个函数可以共享相同的名称，只要它们可以通过其参数列表进行区分）。

DEFERMINISTIC /NOT DEPERMINISTIC 表示函数是否始终会为一组给定的参数值返回相同的结果。您的应用程序可能会将其用于查询优化。

读取 SQL 数据和修改 SQL 数据分别表示该函数是否可能读取或修改 SQL 数据。如果函数在未指定 READS SQL DATA 的情况下尝试从表或流中读取数据，或者在未指定 MODIFIES SQL DATA 的情况下插入流或修改表，则会引发异常。

在 NULL 输入时返回 NULL 和 NULL INPUT 上调用表示该函数是否被定义为如果其任何参数为空则返回空值。如果未指定，则默认为 NULL INPUT 时调用。

SQL 定义的函数体仅由一个 RETURN 语句组成。

示例

```
CREATE FUNCTION get_fraction( degrees DOUBLE )
    RETURNS DOUBLE
    CONTAINS SQL
    RETURN degrees - FLOOR(degrees)
;
```

创建转储

泵是 Amazon Kinesis Data Analytics 存储库对象（SQL 标准的扩展），它提供持续运行的 INSERT INTO Stream SELECT... FROM 查询功能，从而允许将查询结果持续输入到命名流中。

您需要为查询和命名流指定列列表（这意味着一组源-目标对）。列列表需要在数据类型方面进行匹配，否则 SQL 验证器会拒绝它们。（它们不需要列出目标流中的所有列；您可以为一列设置泵。）

有关更多信息，请参阅 SELECT (p. 35)。

以下代码首先创建和设置架构，然后在此架构中创建两个流：

- “OrderDataWithCreateTime“它将作为泵的源流。
- “OrderData“它将作为泵的目标流。

```
CREATE OR REPLACE STREAM "OrderDataWithCreateTime" (
"key_order" VARCHAR(20),
"key_user" VARCHAR(20),
"key_billing_country" VARCHAR(20),
"key_product" VARCHAR(20),
"quantity" VARCHAR(20),
"eur" VARCHAR(20),
"usd" VARCHAR(20))
DESCRIPTION 'Creates origin stream for pump';

CREATE OR REPLACE STREAM "OrderData" (
"key_order" VARCHAR(20),
"key_user" VARCHAR(20),
"country" VARCHAR(20),
"key_product" VARCHAR(20),
"quantity" VARCHAR(20),
"eur" INTEGER,
"usd" INTEGER)
DESCRIPTION 'Creates destination stream for pump';
```

以下代码使用这两个流来创建泵。数据从” 中选择OrderDataWithCreateTime“并插入”OrderData“。

```
CREATE OR REPLACE PUMP "200-ConditionedOrdersPump" AS
INSERT INTO "OrderData" (
"key_order", "key_user", "country",
"key_product", "quantity", "eur", "usd")
//note that this list matches that of the query
SELECT STREAM
"key_order", "key_user", "key_billing_country",
"key_product", "quantity", "eur", "usd"
//note that this list matches that of the insert statement
FROM "OrderDataWithCreateTime";
```

欲了解更多详情，请参阅主题应用中流和泵在里面Amazon Kinesis Data Analytics.

语法

```
CREATE [ OR REPLACE ] PUMP <qualified-pump-name>
                     [ DESCRIPTION '<string-literal>' ] AS <streaming-insert>
```

其中 streaming-insert 是一个插入语句，例如：

```
INSERT INTO ''stream-name'' SELECT "columns" FROM <source stream>
```

INSERT

INSERT 用于在流中插入行。它也可以在泵中使用，将一个流的输出插入另一个流中。

语法

```
<insert statement> :=
  INSERT [ EXPEDITED ]
  INTO  <table-name > [ ( insert-column-specification ) ]
  <  query  >
<insert-column-specification> := < simple-identifier-list >
<simple-identifier-list> :=
   <simple-identifier> [ , < simple-identifier-list > ]
```

有关价值观的讨论，请参见SELECT (p. 35).

泵流插入

也可以将 INSERT 指定为创建转储 (p. 32)语句。

```
CREATE PUMP "HighBidsPump" AS INSERT INTO "highBids" ( "ticker", "shares", "price")
SELECT  "ticker", "shares", "price"
FROM SALES.bids
WHERE "shares"*"price">100000
```

在这里，要插入到 “HighBids” 流中的结果应来自计算结果为流的 UNION ALL 表达式。这将创建一个持续运行的直播插入。插入行的行时间将继承自 select 或 UNION ALL 输出的行的行时间。同样，如果其他插入器在此插入器之前插入的行时间晚于该插入器最初准备的行，则最初可能会删除行，因为后者会出现时间不合时宜。参见主题创建转储 (p. 32)。

查询

语法

```
<query> :=
    <select>
  | <query> <set-operator> [ ALL ] <query>
  | VALUES <row-constructor> { , <row-constructor> }...
  | '(' <query> ')'
 <set-operator> :=
    EXCEPT
  | INTERSECT
  | UNION
```

```
  <row-constructor> :=
     [ ROW ] ( <expression> { , <expression> }... )
```

选择

上表中的选择框代表任何 SELECT 命令；该命令在其自己的页面上有详细描述。

设置运算符（除外、相交、联合）

集合运算符使用集合运算合并查询生成的行：

- EXCEPT 返回第一个集合中但不在第二个集合中的所有行
- INTERSECT 返回第一组和第二组中的所有行
- UNION 返回任一集合中的所有行

在所有情况下，这两个集合的列数必须相同，并且列类型必须与赋值兼容。生成的关系的列名是第一个查询的列名。

使用 ALL 关键字后，运算符将使用数学上的多重集合的语义，这意味着不会消除重复的行。例如，如果特定行在第一个集合中出现 5 次，在第二个集合中出现 2 次，则 UNION ALL 将发出该行 3 + 2 = 5 次。

当前不支持 ALL 对 EXCEPT 或 INTERSECT。

所有运算符均为左关联运算符，INTERSECT 的优先级高于 EXCEPT 或 UNION，后者的优先级相同。要覆盖默认优先级，可以使用圆括号。例如：

```
SELECT * FROM a
UNION
SELECT * FROM b
INTERSECT
SELECT * FROM c
EXCEPT
SELECT * FROM d
EXCEPT
SELECT * FROM E
```

等效于全括号查询

```
( ( SELECT * FROM a
    UNION
    ( SELECT * FROM b
      INTERSECT
      SELECT * FROM c) )
  EXCEPT
  SELECT * FROM d )
EXCEPT
SELECT * FROM e
```

直播集操作员

UNION ALL 是唯一可以应用于流的集合运算符。运算符的两边都必须是流；如果一侧是流，而另一侧是关系，则是错误的。

例如，以下查询生成了一系列通过电话或网络接管的订单：

```
SELECT STREAM *
  FROM PhoneOrders
UNION ALL
```

```
SELECT STREAM *
  FROM WebOrders
```

行时生成。通过流式传输 UNION ALL 发出的行的行时间与输入行的时间戳相同。

行时界限：行时界限是对流future 内容的断言。它指出流中的下一行将具有不早于边界值的 ROWTIME。例如，如果行时间边界是 2018-12-0223:23:07，则这告诉系统下一行将不早于 2018-12-0223:23:07 到达。行时间边界可用于管理数据流中的间隙，例如证券交易所过夜的间隙。

Amazon Kinesis Data Analytics 通过根据时间戳合并传入的行来确保 ROWTIME 列按升序排列。如果第一组的行的时间戳为 10:00 和 10:30，而第二组仅到达 10:15，则 Kinesis Data Analytics 会暂停第一组并等待第二组到达 10:30。在这种情况下，如果第二个集合的生成者发送一个行时间边界，那将是有利的。

值运算符

VALUES 运算符在查询中表示常量关系。（另请参阅本指南中 SELECT 主题中对值的讨论。）

VALUES 可用作顶级查询，如下所示：

```
VALUES 1 + 2 > 3;
EXPR$0
======
FALSE
VALUES
    (42, 'Fred'),
    (34, 'Wilma');
EXPR$0 EXPR$1
====== ======
    42 Fred
    34 Wilma
```

请注意，系统已为匿名表达式生成了任意列名。您可以通过将 VALUES 放入子查询并使用 AS 子句来分配列名：

```
SELECT *
FROM (
    VALUES
        (42, 'Fred'),
        (34, 'Wilma')) AS t (age, name);
AGE NAME
=== =====
 42 Fred
 34 Wilma
```

SELECT

SELECT 从流中检索行。可以将 SELECT 用作顶级语句，或作为涉及集合操作的查询的一部分，也可以用作其他语句的一部分，包括（例如）作为查询传递到 UDX 时。有关示例，请参阅本指南中的主题 INSERT、IN、EXISTS 和 创建转储 (p. 32)。

主题中介绍了 SELECT 语句的子句SELECT 子句 (p. 37),GROUP BY 子句 (p. 55)，直播分组方式，ORDER BY 子句 (p. 65),HAVING 子句 (p. 55),WINDOW 子句 (滑动窗口) (p. 57)和WHERE 子句 (p. 57)。

语法

```
<select> :=
```

```
    SELECT [ STREAM] [ DISTINCT | ALL ]
    <select-clause>
    FROM <from-clause>
    [ <where-clause> ]
    [ <group-by-clause> ]
    [ <having-clause> ]
    [ <window-clause> ]
    [ <order-by-clause> ]
```

STREAM 关键字和流式处理 SQL 原理

SQL 查询语言专为查询存储的关系和生成有限的关系结果而设计。

流式处理 SQL 的基础是 STREAM 关键字，它告诉系统计算关系的时间差。关系的时间差是关系相对于时间的变化。流式查询计算关系相对于时间的变化，或者根据多个关系计算出的表达式的变化。

要在 Amazon Kinesis Data Analytics 中询问关系的时差，我们使用了 STREAM 关键字：

```
SELECT STREAM * FROM Orders
```

如果我们在 10:00 开始运行该查询，它将在 10:15 和 10:25 生成行。在 10:30 查询仍在运行，等待future 订单：

```
ROWTIME  orderId custName   product quantity
======== ======= ========== ======= ========
10:15:00     102 Ivy Black  Rice           6
10:25:00     103 John Wu    Apples         3
```

在这里，系统显示 “在 10:15:00 我执行了 SELECT * FROM Orders 查询，发现结果中有一行在 10:14:59 .999 不存在”。它在 ROWTIME 列中生成值为 10:15:00 的行，因为那是该行出现的时候。这是直播的核心思想：一种随着时间的推移不断更新的关系。

您可以将此定义应用于更复杂的查询。例如，直播

```
SELECT STREAM * FROM Orders WHERE quantity > 5
```

在 10:15 有一行但在 10:25 没有行，因为关系

```
SELECT * FROM Orders WHERE quantity > 5
```

在订单 102 于 10:15 下达时从空白转移到了一个行，但在订单 103 于 10:25 下达时未受影响。

我们可以将相同的逻辑应用于涉及任意 SQL 运算符组合的查询。涉及 JOIN、GROUP BY、子查询、集合运算 UNION、INTERSECT、EXCEPT，甚至限定符（例如 IN 和 EXISTS）的查询在转换为流时都是明确定义的。组合流和存储关系的查询也定义得很明确。

全选并选择 “不同”

如果指定了 ALL 关键字，则查询不会删除重复行。如果既未指定 ALL DISTINCT 行为，这是 DISTINCT 行为。

如果指定了 DISTINCT 关键字，则查询将根据 SELECT 子句中的列删除重复行。

请注意，出于这些目的，NULL 值被视为等于自身而不等于任何其他值。这些语义与 GROUP BY 和 IS NOT DISTINCT FROM 运算符的语义相同。

DISTINCT

只要 SELECT 子句中存在非常量单调表达式，SELECT DISTINCT 就可以用于流式查询。（非常量单调表达式的理由与直播 GROUP BY 的基本原理相同。）Amazon Kinesis Data Analytics 一经准备就绪，就会为 SELECT DISTINCT

如果 ROWTIME 是 SELECT 子句中的一列，则出于重复删除的目的将其忽略。根据 SELECT 子句中的其他列删除重复项。

例如：

```
SELECT STREAM DISTINCT ROWTIME, prodId, FLOOR(Orders.ROWTIME TO DAY)
FROM Orders
```

显示在任何给定日期订购的独特产品集。

如果你正在做 “按楼层分组（ROWTIME TO MINUTE）”，并且在给定的分钟内有两行（比如 22:49:10 和 22:49:15），那么这些行的摘要将以 22:50:00 的时间戳出来。为什么？ 因为那是该行最早完成的时间。

注意：“按 ceil 分组（ROWTIME TO MINUTINE）” 或 “按楼层分组（行时间到分钟）-间隔 '1' 天” 会给出相同的行为。

决定行完成的不是分组表达式的值，而是该表达式更改值的时候。

如果你希望输出行的行时间是它们被发射的时间，那么在以下示例中，你需要从表单 1 改为使用表单 2：

```
(Form 1)
   select distinct floor(s.rowtime to hour), a,b,c
   from s
(Form 2)
   select min(s.rowtime) as rowtime, floor(s.rowtime to hour), a, b, c
   from s
   group by floor(s.rowtime to hour), a, b, c
```

SELECT 子句

在 <select-clause>STREAM 关键字后面使用以下项目：

```
  <select-list> :=
    <select-item> { , <select-item> }...
 <select-item> :=
    <select-expression> [ [ AS ] <simple-identifier> ]
 <simple-identifier> :=
    <identifier> | <quoted-identifier>
 <select-expression> :=
    <identifier> . *  | *  | <expression>
```

表达式

这些表达式中的每一个都可能是：

- 标量表达式
- 打电话给聚合函数 (p. 69)，如果这是一个聚合查询（参见GROUP BY 子句 (p. 55)）
- 打电话给分析函数 (p. 97)，如果这不是聚合查询
- 通配符表达式 * 扩展为 FROM 子句中所有关系的所有列
- 通配符表达式别名。* 扩展到由关系命名的别名的所有列
- 这ROWTIME (p. 67)
- 一个CASE 表达式 (p. 38)

可以使用 AS column_name 语法为每个表达式分配一个别名。这是此查询结果集中的列的名称。如果此查询位于封闭查询的 FROM 子句中，则该名称将用于引用该列。在流引用的 AS 子句中指定的列数必须与原始流中定义的列数相匹配。

Amazon Kinesis Data Analytics 有一些简单的规则可以派生出没有别名的表达式的别名。列表达式的默认别名是列的名称：例如，EMPS.DEPTNO 在默认情况下别名为 DEPTNO。其他表达式使用别名，如 EXPR $0。你不应该假设系统每次都会生成相同的别名。

在流式查询中，将列别名化为 ROWTIME 具有特殊含义：有关更多信息，请参阅 ROWTIME (p. 67)。

Note

所有流都有一个名为 ROWTIME 的隐式列。此列可能会影响您对 SQL: 2008 现在支持的 “As t (c1, c2,...)” 语法的使用。以前在 FROM 子句中你只能写

```
SELECT ... FROM r1 AS t1 JOIN r2 as t2
```

但是 t1 和 t2 的列与 r1 和 t2 的列相同。AS 语法允许您通过编写以下内容来重命名 r1 的列：

```
SELECT ... FROM r1 AS t1(a, b, c)
```

（r1 必须精确有 3 列才能使用此语法）。

如果 r1 是流，则 ROWTIME 被隐式包含，但它不算作列。因此，如果一个流有 3 列而不包括 ROWTIME，则无法通过指定 4 列来重命名 ROWTIME。例如，如果直播出价有三列，则以下代码无效。

```
SELECT STREAM * FROM Bids (a, b, c, d)
```

重命名另一列 ROWTIME 也是无效的，如以下示例所示。

```
SELECT STREAM * FROM Bids (ROWTIME, a, b)
```

因为这意味着将另一列重命名为 ROWTIME。有关表达式和文字的更多信息，请参阅表达式和文字 (p. 17).

CASE 表达式

CASE 表达式允许您为每个此类测试指定一组离散测试表达式和特定的返回值（表达式）。每个测试表达式都在 WHEN 子句中指定；每个返回值表达式在相应的 THEN 子句中指定。可以指定多个这样的 WHEN-THEN 对。

如果您指定 comparison-test-expression 在第一个 WHEN 子句之前，然后将 WHEN 子句中的每个表达式与该表达式进行比较 comparison-test-expression. 第一个匹配的 comparison-test-expression 导致返回其对应 THEN 子句的返回值。如果没有 WHEN 子句表达式匹配 comparison-test-expression，除非指定了 ELSE 子句，否则返回值为空，在这种情况下，将返回该 ELSE 子句中的返回值。

如果您未指定 comparison-test-expression 在第一个 WHEN 子句之前，WHEN 子句中的每个表达式都会被求值（从左到右），第一个表达式为真会返回其相应的 THEN 子句的返回值。如果没有 WHEN 子句表达式为真，则除非指定了 ELSE 子句，否则返回值为空，在这种情况下，将返回该 ELSE 子句中的返回值。

VALUES

VALUES 使用表达式来计算一个或多个行值，通常用于较大的命令中。创建多行时，VALUES 子句必须为每行指定相同数量的元素。生成的表列数据类型源自该列中出现的表达式的显式或推断类型。只要允许 SELECT，就允许在语法上使用值。另请参阅本指南的 “查询” 主题中关于 VALUES 作为运算符的讨论。

语法

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

```
    [ ORDER BY sort_expression [ ASC | DESC | USING operator ] [, ...] ]
```

VALUES 是一个 SQL 运算符，与 SELECT 和 UNION 相当，它支持以下类型的操作：

- 你可以写入 VALUES (1)、(2) 来返回两行，每行只有一个匿名列。
- 你可以写入 VALUES (1, 'a'), (2, 'b') 来返回两行两列。
- 您可以使用 AS 命名列，如以下示例所示：

```
SELECT * FROM (VALUES (1, 'a'), (2, 'b')) AS t(x, y)
```

VALUES 最重要的用法是在 INSERT 语句中插入一行：

```
  INSERT INTO emps (empno, name, deptno, gender)
    VALUES (107, 'Jane Costa', 22, 'F');
```

但是，您还可以插入多行：

```
   INSERT INTO Trades (ticker, price, amount)
     VALUES ('MSFT', 30.5, 1000),
            ('ORCL', 20.25, 2000);
```

在 SELECT 语句的 FROM 子句中使用 VALUES 时，整个 VALUES 子句必须用圆括号括起来，这与它作为查询而不是表表达式运行的事实一致。有关其他示例，请参阅 FROM 子句 (p. 39)。

Note

在直播中使用 INSERT 需要额外考虑行驶时间、泵和 INSERT EXCEDITED。有关更多信息，请参阅INSERT (p. 33)。

FROM 子句

FROM 子句是查询的行源。

```
 <from-clause> :=
    FROM <table-reference> { , <table-reference> }...
 <table-reference> :=
    <table-name> [ <table-name> ] [ <correlation> ]
| <joined-table>
 <table-name> :=  <identifier>
 <table-over> :=  OVER <window-specification>
 <window-specification> :=
 (    <window-name>
| <query_partition_clause>
| ORDER BY <order_by_clause>
| <windowing_clause>
    )
 <windowing-clause> :=
    { ROWS | RANGE }
    { BETWEEN
    { UNBOUNDED PRECEDING
      | CURRENT ROW
      | <value-expression> { PRECEDING | FOLLOWING }
    }
       AND
    { UNBOUNDED FOLLOWING
      | CURRENT ROW
      | <value-expression> { PRECEDING | FOLLOWING }
    }
    | { UNBOUNDED { PRECEDING | FOLLOWING }
```

```
      | CURRENT ROW
      | <value-expression> { PRECEDING | FOLLOWING }
    }
    }
```

有关窗口规格和窗口条款的图表，请参阅WINDOW 子句 (滑动窗口) (p. 57)在 Window 语句下。

```
<correlation> :=
   [ AS ] <correlation-name> [ '(' <column> { , <column> }... ')' ]
<joined-table> :=
   <table-reference> CROSS JOIN <table-reference>
 | <table-reference> NATURAL <join-type> JOIN <table-reference>
 | <table-reference> <join-type> JOIN <table-reference>
      [ USING '(' <column> { , <column>}... ')'
      | ON <condition>
      ]
<join-type> :=
   INNER
 | <outer-join-type> [ OUTER ]
<outer-join-type> :=
   LEFT
 | RIGHT
 | FULL
```

关系

FROM 子句中可以出现几种类型的关系：

- 命名关系 (表、流)
- 用圆括号括起的子查询。
- 结合两种关系的联接 (参见本指南中的 JOIN 主题) 。
- 转换表达式。

本指南的查询主题中更详细地描述了子查询。

下面是一些子查询示例。

```
// set operation as subquery
// (finds how many departments have no employees)
SELECT COUNT(*)
FROM (
  SELECT deptno FROM Dept
  EXCEPT
  SELECT deptno FROM Emp);
// table-constructor as a subquery,
// combined with a regular table in a join
SELECT *
FROM Dept AS d
  JOIN (VALUES ('Fred', 10), ('Bill', 20)) AS e (name, deptno)
  ON d.deptno = e.deptno;
```

与 SELECT 语句其他部分中的子查询不同，例如WHERE 子句 (p. 57)子句 (WHERE子句 (p. 21))，FROM 子句中的子查询不能包含关联变量。例如：

```
// Invalid query. Dept.deptno is an illegal reference to
// a column of another table in the enclosing FROM clause.
SELECT *
FROM Dept,
  (SELECT *
   FROM Emp
```

```
  WHERE Emp.deptno = Dept.Deptno)
```

具有多个关系的 FROM 子句

如果 FROM 子句包含多个以逗号分隔的关系，则查询会构造这些关系的笛卡尔乘积；也就是说，它将每个关系中的每一行与其他每个关系中的每一行合并。

因此，FROM 子句中的逗号等同于 CROSS JOIN 运算符。

关联名

FROM 子句中的每个关系都可以使用 AS 关联名分配一个相关名。此名称是一种备用名称，在整个查询的表达式中都可以通过该名称来引用关系。（尽管该关系可能是子查询或流，但通常将其称为“表别名”，以便将其与 SELECT 子句中定义的列别名区分开来。）

如果没有 AS 子句，命名关系的名称将成为其默认别名。（在流式查询中，OVER 子句不会阻止此默认赋值的发生。）

如果查询多次使用相同的命名关系，或者其中任何关系是子查询或表表达式，则必须使用别名。

例如，在以下查询中，使用了两次命名关系 EMPS；一次使用其默认别名 EMPS，另一次使用分配的别名 MANAGERS：

```
SELECT EMPS.NAME || ' is managed by ' || MANAGERS.NAME
FROM LOCALDB.Sales.EMPS,
  LOCALDB.Sales.EMPS AS MANAGERS
WHERE MANAGERS.EMPNO = EMPS.MGRNO
```

别名后面可以选择加上列列表：

```
SELECT e.empname,
FROM LOCALDB.Sales.EMPS AS e(empname, empmgrno)
```

OVER 子句

OVER 子句仅适用于串流联接。欲了解更多详情，请参阅主题加入子句 (p. 41)。

加入子句

SELECT 语句中的 JOIN 子句合并了来自一个或多个流或引用表中的列。

主题

流-流联接

Amazon Kinesis Data Analytics 支持使用 SQL 将应用程序内数据流与另一个应用程序内流相结合，从而将这一重要的传统数据库功能引入流环境中。

本节介绍了 Kinesis Data Analytics 支持的联接类型，包括基于时间和基于行的窗口联接，以及有关流式连接的详细信息。

联接类型

连接有五种类型：

内部加入（或者只是加入）	返回来自左侧和来自右侧并且联接条件的计算结果为 TRUE 的所有行对。
左外连接（或者只是左连接）	作为 INNER JOIN，但即使左侧的行不与右侧的任何行匹配，也会保留左侧的行。在右侧生成 NULL 值。
右外连接（或刚好右连接）	作为 INNER JOIN，但即使右侧的行不与左侧的任何行匹配，也会保留右侧的行。在左侧为这些行生成 NULL 值。
完全外部加入（或者只是完全加入）	作为 INNER JOIN，但即使两侧的行不与任一侧的任何行匹配，也会保留两侧的行。在另一侧为这些行生成 NULL 值。
CROSS JOIN	返回输入的笛卡尔乘积：左边的每一行与右边的每一行配对。

基于时间的窗口与基于行的窗口联接

将左侧流的整个历史记录与右侧流的整个历史记录相联接是不切实际的。因此，您必须使用 OVER 子句将至少一个流限制到一个时间窗口。OVER 子句定义了在给定时间考虑连接的行窗口。

窗口可以是基于时间的，也可以是基于行的：

- 基于时间的窗口使用 RANGE 关键字。它将窗口定义为其 ROWTIME 列落在查询的当前时间的特定时间间隔内的一组行。

  例如，以下子句指定窗口包含其 ROWTIME 在流当前时间之前的那个小时内的所有行：

```
OVER (RANGE INTERVAL '1' HOUR PRECEDING)
```

- 基于行的窗口使用 ROWS 关键字。它将窗口定义为具有当前时间戳的行之前或之后的给定行计数。

  例如，以下子句指定窗口中仅包含最新的 10 行：

```
OVER (ROWS 10 PRECEDING)
```

> **Note**
>
> 如果在联接的一侧没有指定时间窗口或基于行的窗口，则只有该侧的当前行参与联接计算。

流到流联接的示例

以下示例演示了应用程序内如何执行操作 stream-to-stream join 有效，连接结果何时返回，连接结果的行时间是多少。

主题

示例数据集

本节中的示例基于以下数据集和流定义：

Orders 数据示例

```
{
   "orderid":"101",
   "orders":"1"
}
```

Shipments 数据示例

```
{
   "orderid":"101",
   "shipments":"2"
}
```

创建 ORDERS_STREAM 应用程序内流

```
CREATE OR REPLACE STREAM "ORDERS_STREAM" ("orderid" int, "orderrowtime" timestamp);
CREATE OR REPLACE PUMP "ORDERS_STREAM_PUMP" AS INSERT INTO "ORDERS_STREAM"
SELECT STREAM "orderid", "ROWTIME"
FROM "SOURCE_SQL_STREAM_001" WHERE "orders" = 1;
```

创建 SHIPMENTS_STREAM 应用程序内流

```
CREATE OR REPLACE STREAM "SHIPMENTS_STREAM" ("orderid" int, "shipmentrowtime" timestamp);
CREATE OR REPLACE PUMP "SHIPMENTS_STREAM_PUMP" AS INSERT INTO "SHIPMENTS_STREAM"
SELECT STREAM "orderid", "ROWTIME"
FROM "SOURCE_SQL_STREAM_001" WHERE "shipments" = 2;
```

示例 1：连接一侧的时间窗口（内部连接）

此示例演示了一个查询，该查询返回在最后一分钟执行发货的所有订单。

联接查询

```
CREATE OR REPLACE STREAM "OUTPUT_STREAM" ("resultrowtime" timestamp, "orderid" int,
 "shipmenttime" timestamp, "ordertime" timestamp);
CREATE OR REPLACE PUMP "OUTPUT_STREAM_PUMP" AS
INSERT INTO "OUTPUT_STREAM"
  SELECT STREAM ROWTIME as "resultrowtime", s."orderid", s.rowtime as "shipmenttime",
 o.ROWTIME as "ordertime"
    FROM ORDERS_STREAM OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS o
      JOIN SHIPMENTS_STREAM AS s
        ON o."orderid" = s."orderid";
```

查询结果

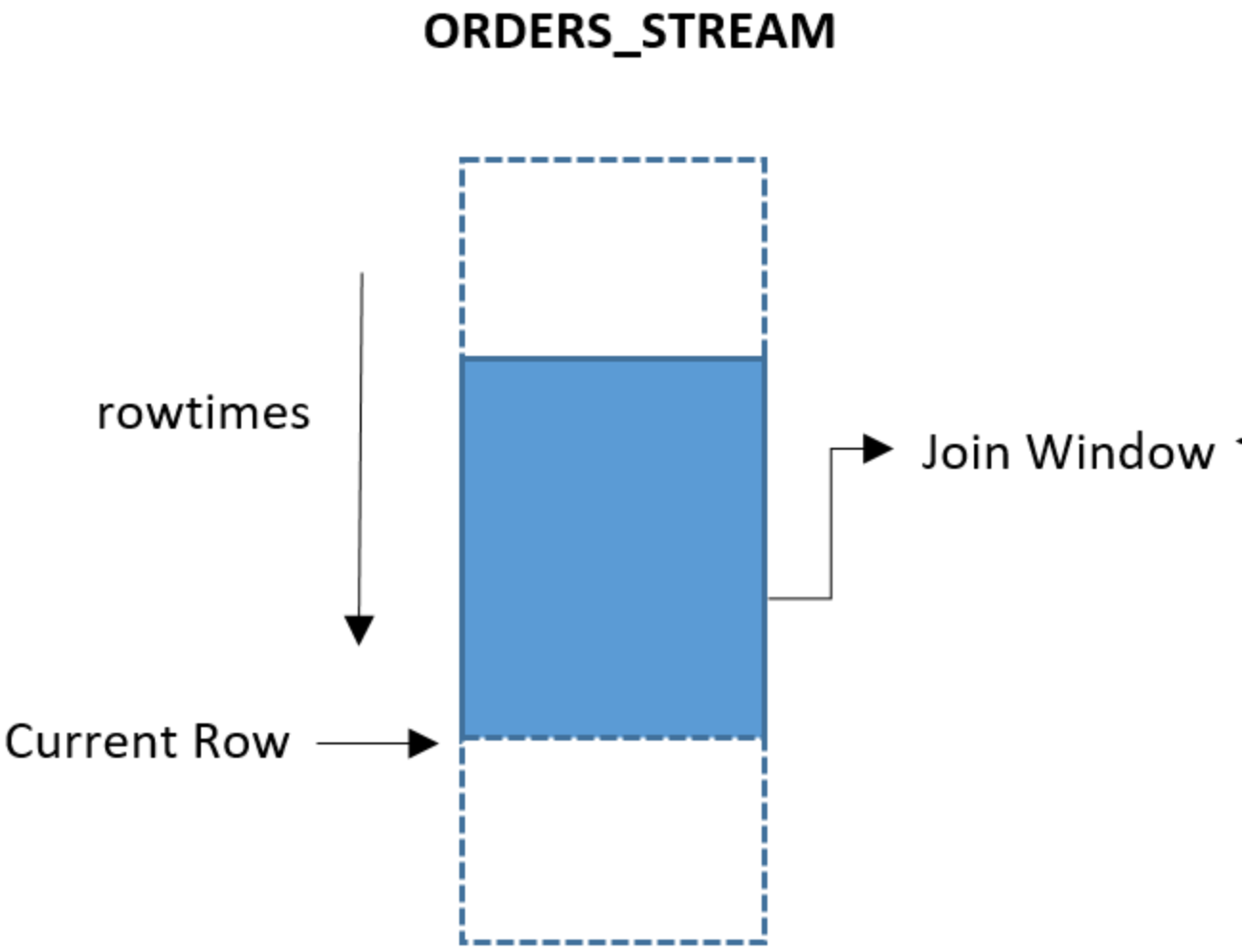
ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM			
ROWTIME	orderid	ROWTIME	orderid	resultrowtime	orderid	shipmenttime	OrderTime

ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM			
10:00:00	101	10:00:00	100				
10:00:20	102						
10:00:30	103						
10:00:40	104						
		10:00:45	104				
10:00:45	100*			10:00:45	104	10:00:45	10:00:40
		10:00:50	105				

* - orderid = 100 的记录是 Orders 流中的较迟事件。

联接的可视化表示

下图表示一个查询，该查询返回在最后一分钟执行发货的所有订单。



触发结果

下面介绍了触发查询结果的事件。

- 由于未在 Shipments 流上指定时间或行窗口，因此只有Shipments 流的当前行参与联接。
- 由于对 Orders 流的查询指定了前一分钟的窗口，因此 Orders 流中最后一分钟带有 ROWTIME 的行将参与联接。
- 当 Shipments 流中的记录对于 orderid 104 在 10:00:45到达时，将触发 JOIN 结果，因为前一分钟在 Orders 流中对于 orderid 存在匹配项。
- Orders 流中 Orderid 为 100 的记录到达较迟，因此 Shipments 流中的对应记录不是最新记录。由于未在 Shipments 流上指定任何窗口，因此只有Shipments 流的当前行参与联接。因此，JOIN 语句不会针对 orderid 100 返回任何记录。有关在 JOIN 语句中包含较迟行的信息，请参阅 示例 2 (p. 45)。
- 因为在 Shipments 流中对于 Orderid 105 没有匹配的记录，所以不会发出任何结果，并且该记录将被忽略。

结果的 ROWTIME

- 输出流中记录的 ROWTIME 是与联接匹配的行的 ROWTIME 的较迟者。

示例 2：连接两侧的时间窗口（内部连接）

此示例演示了一个查询，该查询返回最后一分钟执行的所有订单以及最后一分钟执行的发货。

联接查询

```
CREATE OR REPLACE STREAM "OUTPUT_STREAM" ("resultrowtime" timestamp, "orderid" int,
 "shipmenttime" timestamp, "ordertime" timestamp);
CREATE OR REPLACE PUMP "OUTPUT_STREAM_PUMP" AS INSERT INTO "OUTPUT_STREAM"
  SELECT STREAM ROWTIME as "resultrowtime", s."orderid", s.rowtime as "shipmenttime",
 o.ROWTIME as "ordertime"
    FROM ORDERS_STREAM OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS o
      JOIN SHIPMENTS_STREAM OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS s
        ON o."orderid" = s."orderid";
```

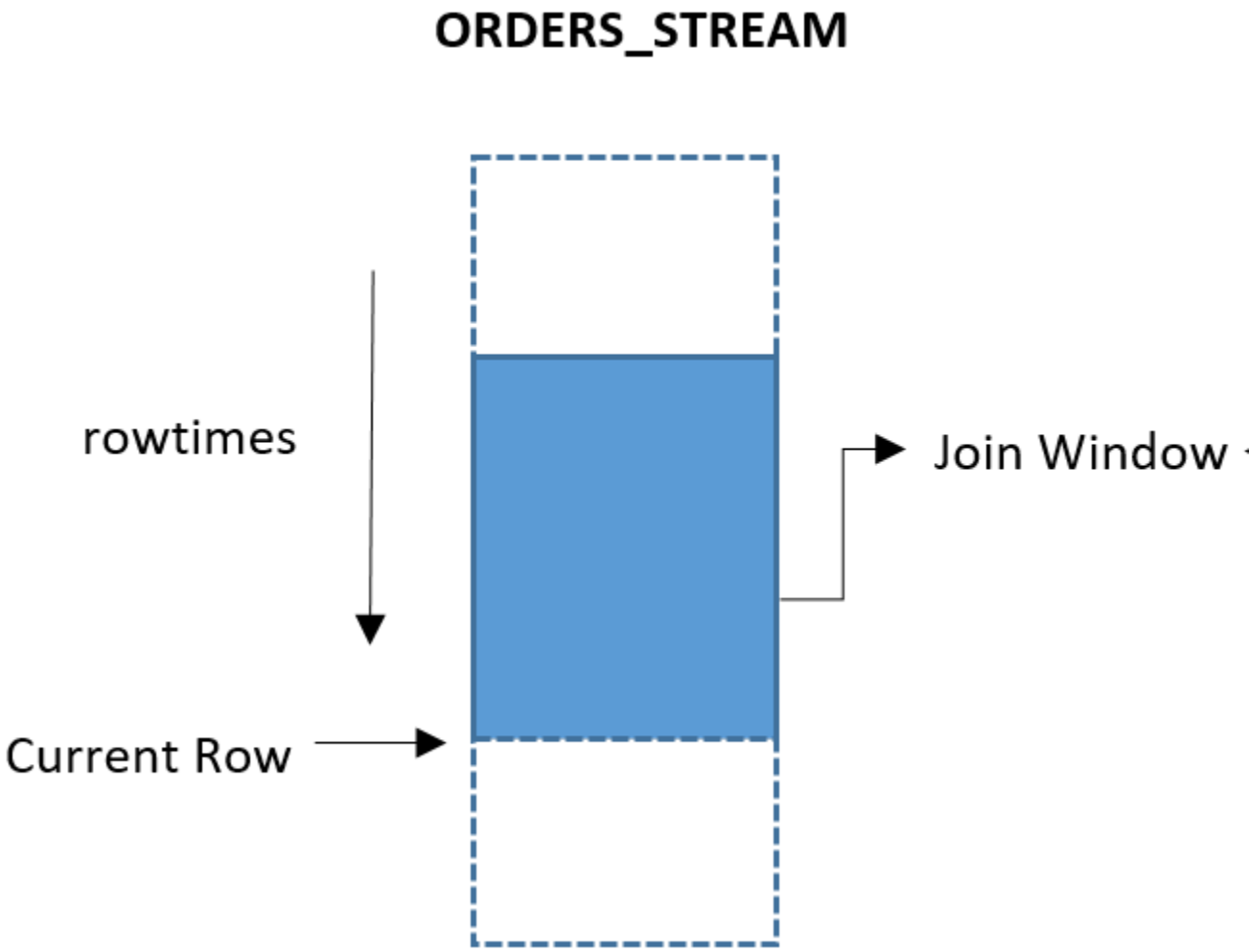
查询结果

ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM			
ROWTIME	orderid	ROWTIME	orderid	resultrowtime	orderid	shipmenttime	OrderTime
10:00:00	101	10:00:00	100				
10:00:20	102						
10:00:30	103						
10:00:40	104						
		10:00:45	104				
10:00:45	100*			10:00:45	104	10:00:45	10:00:40
				10:00:45	100	10:00:00	10:00:45
		10:00:50	105				

* - orderid = 100 的记录是 Orders 流中的较迟事件。

联接的可视化表示

下图表示一个查询，该查询返回最后一分钟执行的所有订单以及最后一分钟执行的发货。



触发结果

下面介绍了触发查询结果的事件。

- 在联接的两侧指定窗口。因此，Orders 流和 Shipments 流的当前行之前的一分钟内的所有行都参与联接。
- 当 Shipments 流中与 orderid 104 对应的记录到达时，Orders 流中的对应记录在一分钟窗口内。因此，一条记录返回到 Output 流。
- 即使 orderid 100 的订单事件在 Orders 流中到达较晚，也返回了联接结果。这是因为 Shipments 流中的窗口包括过去一分钟的订单，其中包括对应的记录。
- 在联接的两侧设置一个窗口有助于在联接的两侧包含迟到的记录；例如，如果订单或发货记录延迟收到或出现问题。

结果的 ROWTIME

- 输出流中记录的 ROWTIME 是与联接匹配的行的 ROWTIME 的较迟者。

示例 3：右连接一侧的时间窗口（右外连接）

此示例演示了一个查询，该查询返回最后一分钟执行的所有发货，无论最后一分钟是否存在对应的订单。

联接查询

```
CREATE OR REPLACE STREAM "OUTPUT_STREAM" ("resultrowtime" timestamp, "orderid" int,
 "ordertime" timestamp);
CREATE OR REPLACE PUMP "OUTPUT_STREAM_PUMP" AS INSERT INTO "OUTPUT_STREAM"
  SELECT STREAM ROWTIME as "resultrowtime", s."orderid", o.ROWTIME as "ordertime"
    FROM ORDERS_STREAM OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS o
      RIGHT JOIN SHIPMENTS_STREAM AS s
        ON o."orderid" = s."orderid";
```

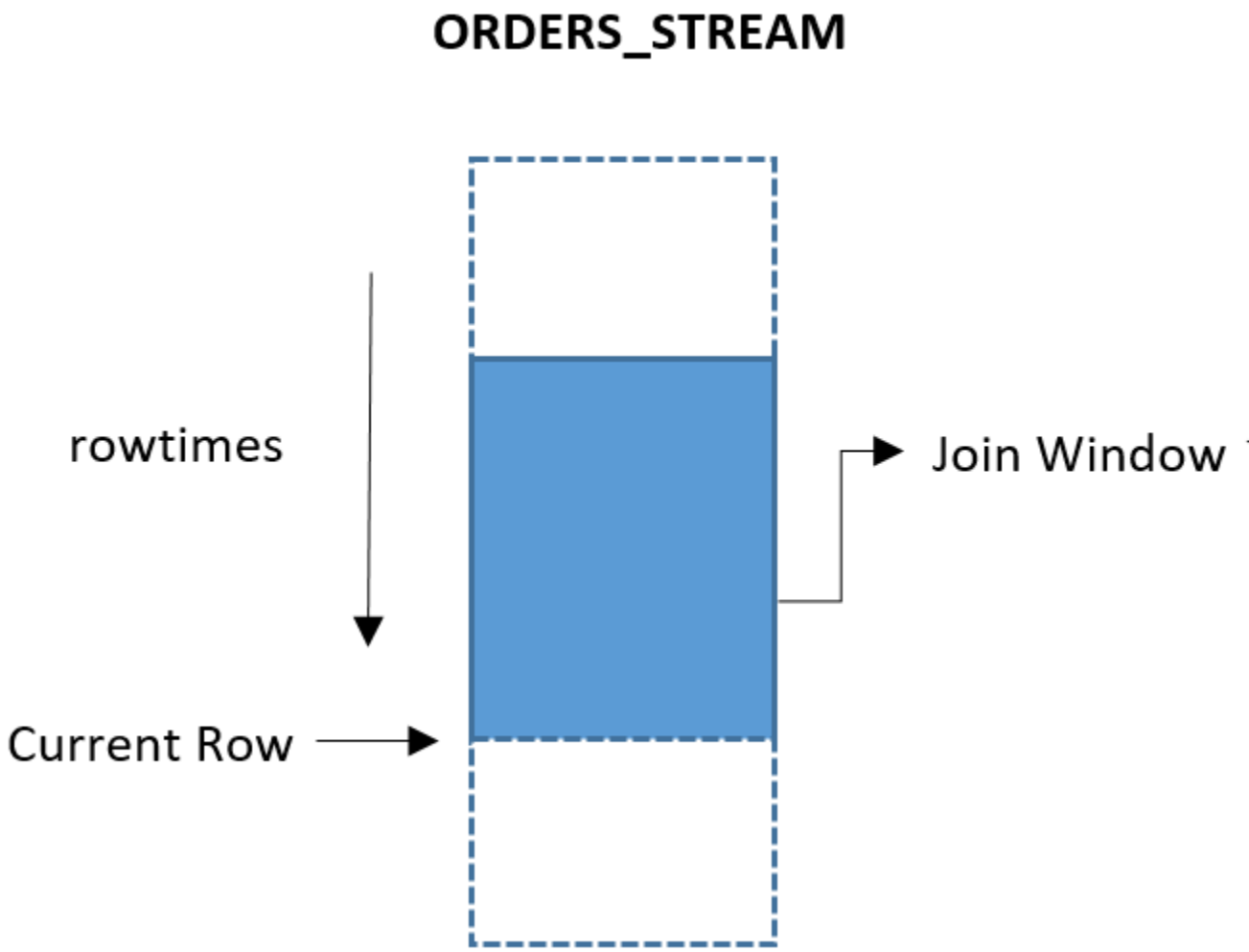
查询结果

ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM		
ROWTIME	orderid	ROWTIME	orderid	resultrowtime	orderid	OrderTime
10:00:00	101	10:00:00	100			
				10:00:00	100	null
10:00:20	102					
10:00:30	103					
10:00:40	104					
		10:00:45	104			
10:00:45	100*			10:00:45	104	10:00:40
		10:00:50	105			
				10:00:50	105	null

* - orderid = 100 的记录是 Orders 流中的较迟事件。

联接的可视化表示

下图表示一个查询，该查询返回最后一分钟执行的所有装运，无论是否在最后一分钟内存在对应的订单。



触发结果

下面介绍了触发查询结果的事件。

- 当 Shipments 流中与 orderid 104 对应的记录到达时，将在 Output 流中发出结果。
- 只要 Shipments 流中与 orderid 105 对应的记录到达，就在 Output 流中发出一条记录。但是，订单流中没有匹配的记录，所以 OrderTime 值为空。

结果的 ROWTIME

- 输出流中记录的 ROWTIME 是与联接匹配的行的 ROWTIME 的较迟者。
- 因为联接（Shipments 流）的右侧没有窗口，所以具有不匹配联接的结果的 ROWTIME 是不匹配行的 ROWTIME。

示例 4：右连接两侧的时间窗口（右外连接）

此示例演示一个查询，该查询返回最后一分钟执行的所有发货，无论它们是否具有对应的订单。

联接查询

```
CREATE OR REPLACE STREAM "OUTPUT_STREAM" ("resultrowtime" timestamp, "orderid" int,
 "shipmenttime" timestamp, "ordertime" timestamp);
CREATE OR REPLACE PUMP "OUTPUT_STREAM_PUMP" AS INSERT INTO "OUTPUT_STREAM"
  SELECT STREAM ROWTIME as "resultrowtime", s."orderid", s.ROWTIME as "shipmenttime",
 o.ROWTIME as "ordertime"
    FROM ORDERS_STREAM OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS o
      RIGHT JOIN SHIPMENTS_STREAM OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS s
        ON o."orderid" = s."orderid";
```

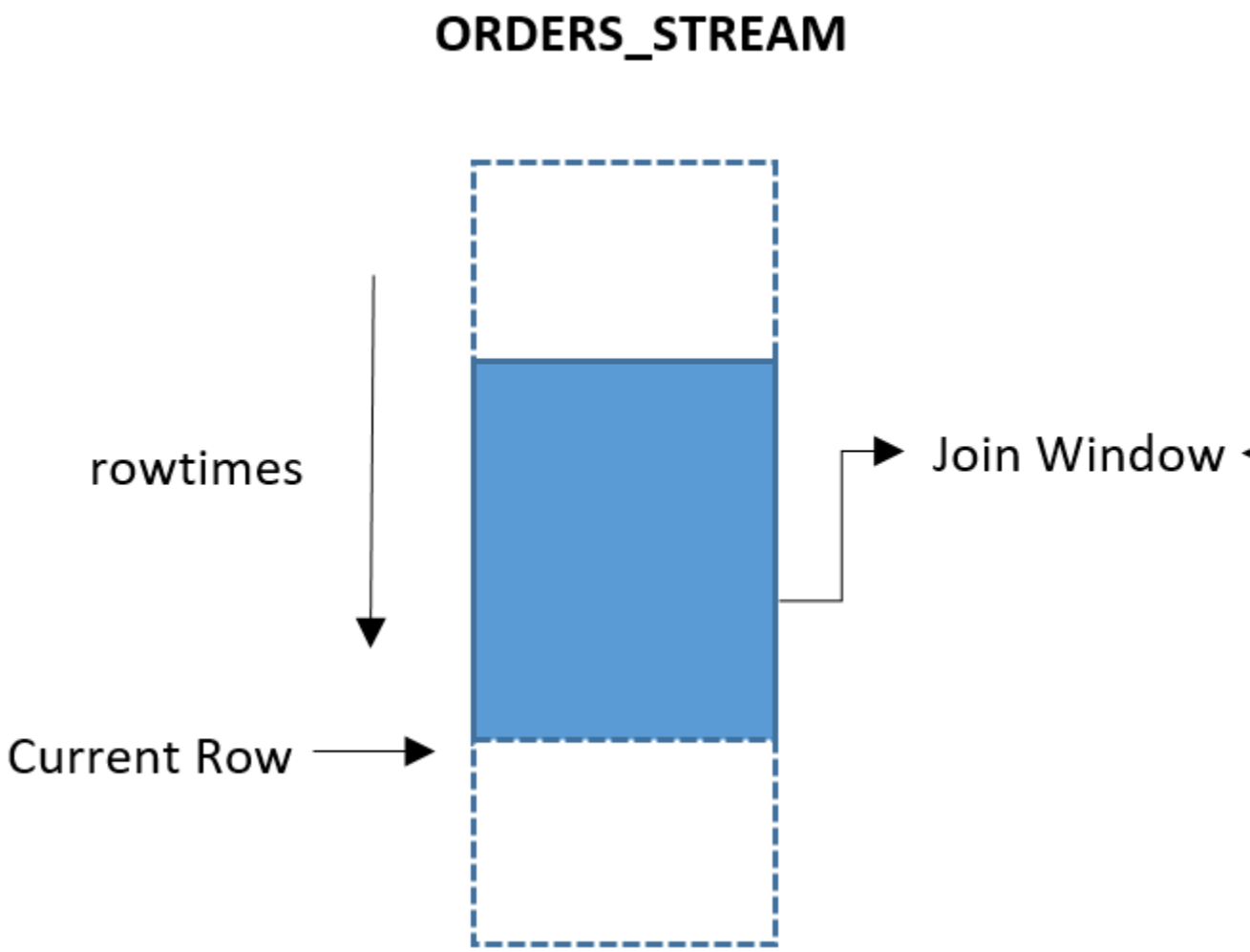
查询结果

ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM			
ROWTIME	orderid	ROWTIME	orderid	resultrowtime	orderid	shipmenttime	OrderTime
10:00:00	101	10:00:00	100				
10:00:20	102						
10:00:30	103						
10:00:40	104						
		10:00:45	104				
10:00:45	100*			10:00:45	104	10:00:40	10:00:45
				10:00:45	100	10:00:45	10:00:00
		10:00:50	105				
				10:01:50	105	10:00:50	null

* - orderid = 100 的记录是 Orders 流中的较迟事件。

联接的可视化表示

下图表示一个查询，该查询返回最后一分钟执行的所有发货，无论它们是否具有对应的订单。



触发结果

下面介绍了触发查询结果的事件。

- 当 Shipments 流中与 orderid 104 对应的记录到达时，将在 Output 流中发出结果。
- 即使 orderid 100 的订单事件在 Orders 流中到达较晚，也会返回联接结果。这是因为 Shipments 流中的窗口包括过去一分钟的订单，其中包括对应的记录。
- 对于未找到其订单的发货（对于 orderid 105），直到 Shipments 流上的一分钟窗口结束时，结果才会发送到 Output 流。

结果的 ROWTIME

- 输出流中记录的 ROWTIME 是与联接匹配的行的 ROWTIME 的较迟者。
- 对于没有匹配的订单记录的发货记录，结果的 ROWTIME 是窗口末尾的 ROWTIME。这是因为联接的右侧（来自 Shipments 流）现在是事件的一分钟窗口，并且服务正在等待窗口结束以确定是否有任何匹配的记录到达。当窗口结束且没有找到匹配的记录时，将以与窗口结束对应的 ROWTIME 发出结果。

示例 5：左联接一侧的时间窗口（左外连接）

此示例演示了一个查询，该查询返回最后一分钟执行的所有订单，无论最后一分钟是否存在对应的发货。

联接查询

```
CREATE OR REPLACE STREAM "OUTPUT_STREAM" ("resultrowtime" timestamp, "orderid" int,
 "ordertime" timestamp);
CREATE OR REPLACE PUMP "OUTPUT_STREAM_PUMP" AS INSERT INTO "OUTPUT_STREAM"
  SELECT STREAM ROWTIME as "resultrowtime", o."orderid", o.ROWTIME as "ordertime"
    FROM ORDERS_STREAM OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS o
      LEFT JOIN SHIPMENTS_STREAM AS s
        ON o."orderid" = s."orderid";
```

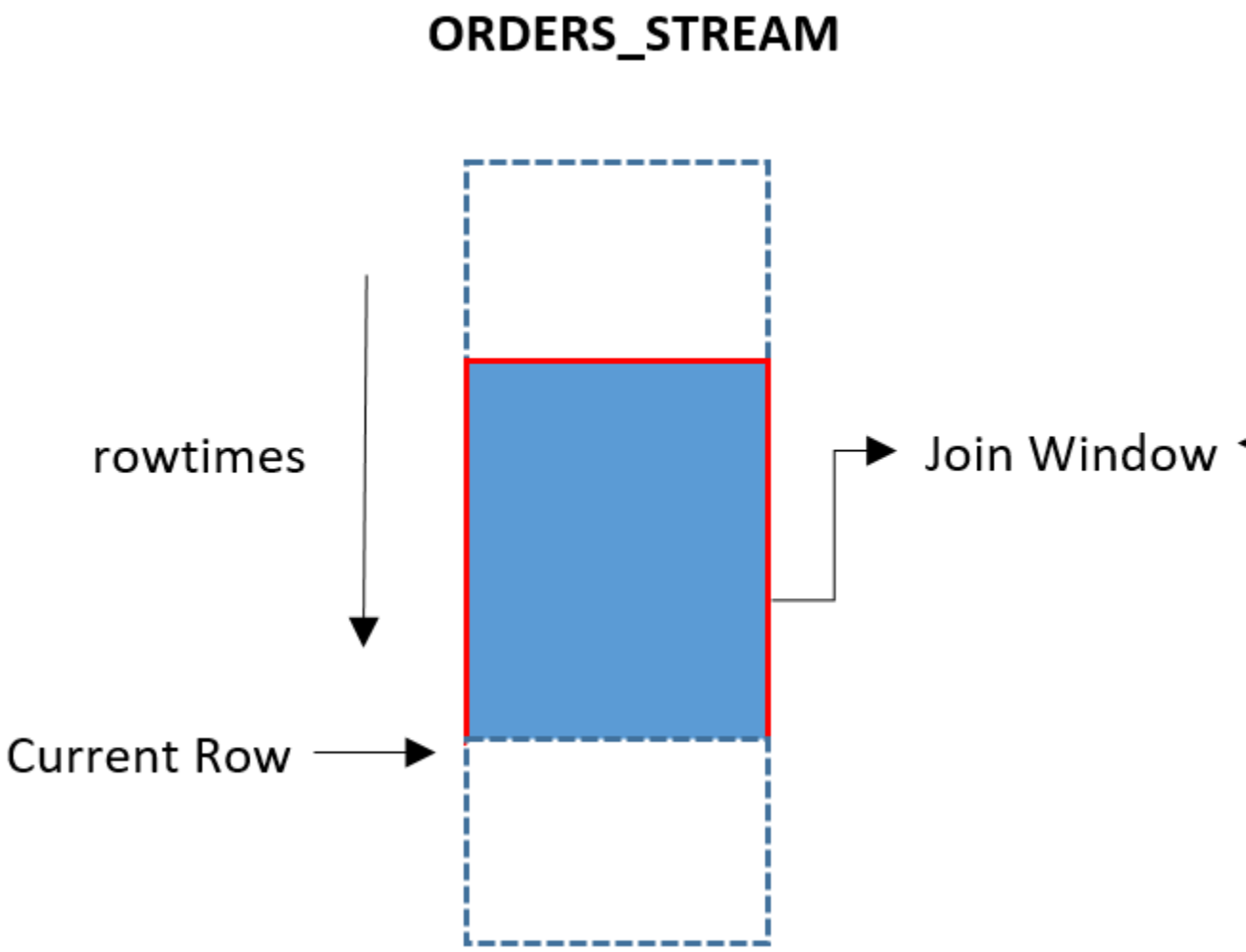
查询结果

ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM		
ROWTIME	orderid	ROWTIME	orderid	resultrowtime	orderid	OrderTime
10:00:00	101	10:00:00	100			
10:00:20	102					
10:00:30	103					
10:00:40	104					
		10:00:45	104			
10:00:45	100*			10:00:45	104	10:00:40
		10:00:50	105			
				10:01:00	101	10:00:00
				10:01:20	102	10:00:20
				10:01:30	103	10:00:30
				10:01:40	104	10:00:40
				10:01:45	100	10:00:45

* - orderid = 100 的记录是 Orders 流中的较迟事件。

联接的可视化表示

下图表示一个查询，该查询返回最后一分钟执行的所有订单，无论最后一分钟是否存在对应的发货。



触发结果

下面介绍了触发查询结果的事件。

- 当 Shipments 流中与 orderid 104 对应的记录到达时，将在 Output 流中发出结果。
- 对于位于 Orders 流中但在 Shipments 流中没有相应记录的记录，直到一分钟窗口结束时才会将记录发出到 Output 流。这是因为服务正在等待直到窗口结束以获取匹配的记录。

结果的 ROWTIME

- 输出流中记录的 ROWTIME 是与联接匹配的行的 ROWTIME 的较迟者。
- 对于位于 Orders 流中但在 Shipments 流中没有相应记录的记录，结果的 ROWTIME 是当前窗口结束的 ROWTIME。

示例 6：左联接两侧的时间窗口（左外连接）

此示例演示一个查询，该查询返回最后一分钟执行的所有订单，无论它们是否具有对应的发货。

联接查询

```
CREATE OR REPLACE STREAM "OUTPUT_STREAM" ("resultrowtime" timestamp, "orderid" int,
 "shipmenttime" timestamp, "ordertime" timestamp);
CREATE OR REPLACE PUMP "OUTPUT_STREAM_PUMP" AS INSERT INTO "OUTPUT_STREAM"
  SELECT STREAM ROWTIME as "resultrowtime", s."orderid", s.ROWTIME as "shipmenttime",
 o.ROWTIME as "ordertime"
    FROM ORDERS_STREAM OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS o
      LEFT JOIN SHIPMENTS_STREAM OVER (RANGE INTERVAL '1' MINUTE PRECEDING) AS s
        ON o."orderid" = s."orderid";
```

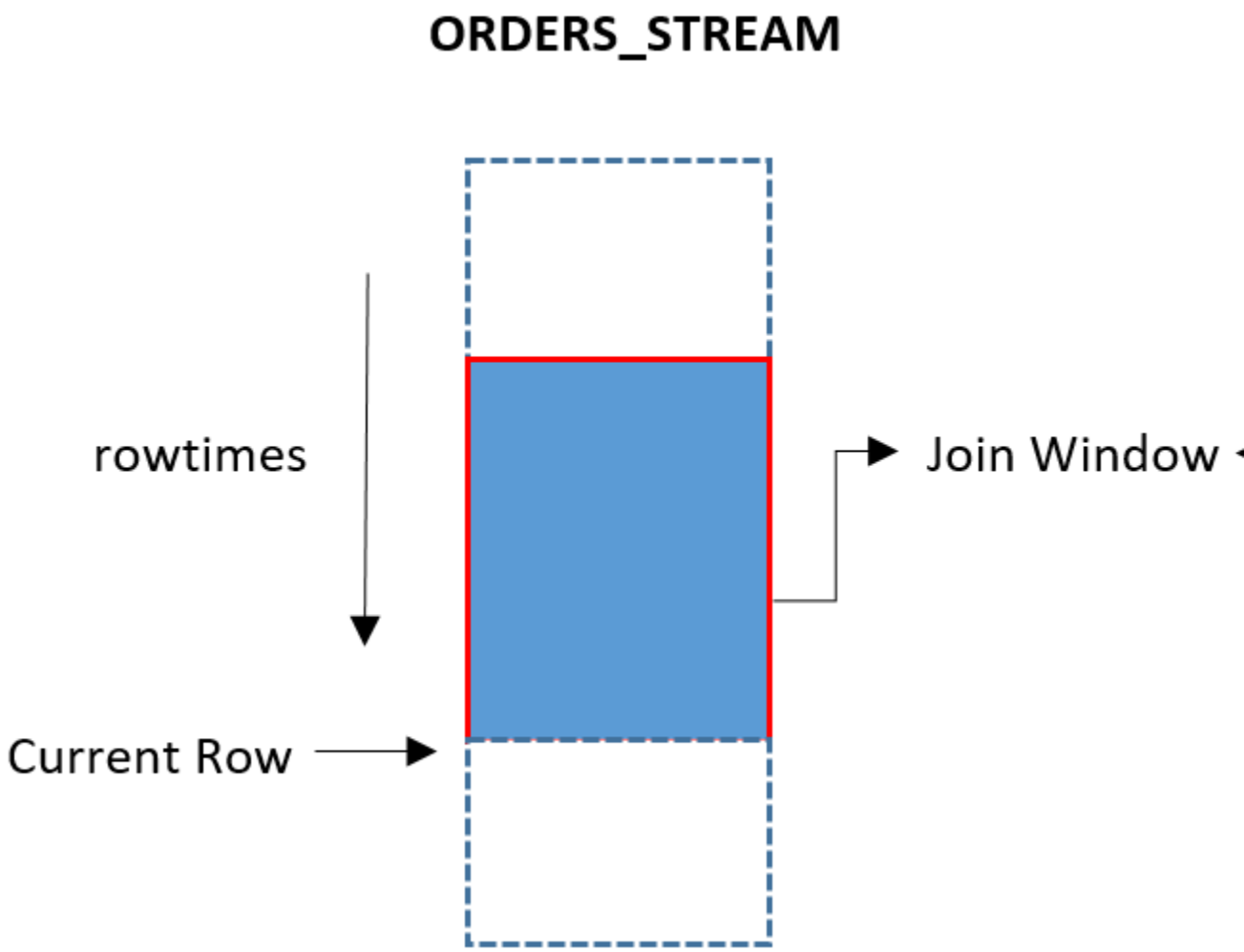
查询结果

ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM			
ROWTIME	orderid	ROWTIME	orderid	resultrowtime	orderid	shipmenttime	OrderTime
10:00:00	101	10:00:00	100				
10:00:20	102						
10:00:30	103						
10:00:40	104						
		10:00:45	104				
10:00:45	100*			10:00:45	104	10:00:40	10:00:45
		10:00:50	105	10:00:45	100	10:00:00	10:00:45
				10:01:00	101	null	10:00:00
				10:01:20	102	null	10:00:20
				10:01:30	103	null	10:00:30
				10:01:40	104	null	10:00:40
				10:01:45	100	null	10:00:45

* - orderid = 100 的记录是 Orders 流中的较迟事件。

联接的可视化表示

下图表示一个查询，该查询返回最后一分钟执行的所有订单，无论它们是否具有对应的发货。



触发结果

下面介绍了触发查询结果的事件。

- 当 Shipments 流中与 orderid 104 和 100 对应的记录到达时，将在 Output 流中发出结果。即使 Orders 流中与 orderid 100 对应的记录较迟到达，也会发生这种情况。
- 对于位于 Orders 流中但在 Shipments 流中没有相应记录的记录，将在一分钟窗口结束时在 Output 流中发出。这是因为该服务一直等到窗口结束，以获取 Shipments 流中的对应记录。

结果的 ROWTIME

- 输出流中记录的 ROWTIME 是与联接匹配的行的 ROWTIME 的较迟者。
- 对于位于 Orders 流中但在 Shipments 流中没有相应记录的记录，订单的 ROWTIME 是与窗口结束相对应的 ROWTIME。

摘要

- Kinesis Data Analytics 始终以 ROWTIME 的升序返回联接中的行。
- 对于内部联接，输出行的 ROWTIME 是两个输入行的 ROWTIME 的较迟者。这也适用于在其中发现了匹配项的输入行的外部联接。

- 对于未发现其匹配项的外部联接，输出行的 ROWTIME 是以下两个时间的较迟者：
  - 未发现其匹配项的输入行的 ROWTIME。
  - 在已发现任何可能的匹配项的时间点，另一个输入流的窗口的较迟边界。

流-表联接

如果其中一个关系是流而另一个是有限关系，则它被称为流表连接。对于流中的每一行，查询会在表中查找与连接条件相匹配的一行或多行。

例如，订单是一个直播， PriceList 是一张桌子。联接的作用是向订单添加价目表信息。

有关创建参考数据源和将流联接到参考表的信息，请参见示例：添加参考数据源在里面Amazon Kinesis Data Analytics 开发人员指南.

HAVING 子句

SELECT 中的 HAVING 子句指定了要在组或聚合中应用的条件。换句话说，HAVING 会在应用 GROUP BY 子句的聚合后筛选行。由于 HAVING 是在 GROUP BY 之后计算的，因此它只能引用由分组键构造（或可导出）的表达式、聚合表达式和常量。（这些规则与适用于 GROUP BY 查询的 SELECT 子句中的表达式的规则相同。）HAVING 条款必须出现在任何 GROUP BY 条款之后和任何 ORDER BY 条款之前。HAVING 就像WHERE 子句 (p. 57)，但适用于群组。HAVING 子句的结果表示原始行的分组或聚合，而 WHERE 子句的结果是单独的原始行。

在非流式应用程序中，如果没有 GROUP BY 子句，则假定 GROUP BY ()（但由于没有分组表达式，表达式只能由常量和聚合表达式组成）。在流式查询中，没有 GROUP BY 子句就无法使用 HAVING。

WHERE 和 HAVING 都可以出现在单个 SELECT 语句中。WHERE 从流或表中选择满足其条件（WHERE 条件）的各个行。GROUP BY 条件仅适用于由 WHERE 条件选择的行。

这样的分组，例如 “GROUP BY CustomerID”，可以通过有条件进一步限定，然后在指定分组中选择满足其条件的行的聚合。例如，“按 ClientID 分组有总和 (ShipmentValue) > 3600" 将只选择那些符合WHERE标准的各种货物的总价值也超过3,600的客户。

有关条件，请参阅 WHERE 子句语法表，该表适用于 HAVING 和 WHERE 子句。

条件必须是布尔谓词表达式。查询仅返回谓词计算为 TRUE 的行。

以下示例显示了一个流式查询，该查询显示了过去一小时内订单超过 1000 美元的产品。

```
SELECT STREAM "prodId"
FROM "Orders"
GROUP BY FLOOR("Orders".ROWTIME TO HOUR), "prodId"
HAVING SUM("quantity" * "price") > 1000;
```

GROUP BY 子句

GROUP BY 子句的语法表

（要了解该条款的适用范围，请参阅SELECT (p. 35))

例如，GROUP BY <column name-or-expression>，其中：

- 表达式可以是聚合；而且，
- GROUP BY 子句中使用的任何列名也必须在 SELECT 语句中。

此外，未在 GROUP BY 子句中命名或衍生自 GROUP BY 子句的列不能出现在 SELECT 语句中，除非出现在聚合中，例如 SUM (allOrdersValue)。

可导出的意思是，在 GROUP BY 子句中指定的列允许访问要包含在 SELECT 子句中的列。如果某列是可导出的，则即使在 GROUP BY 子句中未明确命名该列，SELECT 语句也可以指定该列。

示例：如果表的键在 GROUP BY 子句中，则该表的任何列都可以出现在选择列表中，因为给定该键，此类列被认为是可访问的。

GROUP BY 子句根据分组表达式的值对选定行进行分组，为所有列中具有相同值的每组行返回一个信息摘要行。

请注意，出于这些目的，NULL 值被视为等于自身而不等于任何其他值。这些语义与 IS NOT DISTINCT FROM 运算符的语义相同。

直播分组依据

只要其中一个分组表达式是非常量单调或基于时间的表达式，GROUP BY 就可以在流式查询中使用。要使 Amazon Kinesis Data Analytics 取得进展，这一要求是必要的，如下所述。

单调表达式是指总是朝着相同方向移动的表达式：要么 ascends-or-stays-the-同样，还是这样 descends-or-stays 一样；它不会反转方向。它不必严格按升序或严格降序排列，也就是说，每个值总是高于前一个值或每个值始终低于前一个值。常量表达式属于单调的定义范围——从技术上讲，它既是升序又是降序的——但显然不适合用于这些目的。有关单调的更多信息，请参阅单调表达式和运算符 (p. 19).

请考虑以下查询：

```
SELECT STREAM prodId, COUNT(*)
FROM Orders
GROUP BY prodId
```

该查询旨在以流形式计算每种产品的订单数量。但是，由于订单是无限流，Amazon Kinesis Data Analytics 永远无法知道它已经看到了给定产品的所有订单，永远无法完成特定行的总计，因此永远无法输出一行。Amazon Kinesis Data Analytics 验证器不允许一个永远无法发出一行的查询，而是拒绝该查询。

Grop By for more details.

```
GROUP BY <monotonic or time-based expression> ,

<column name-or-expression, ...>
```

其中 GROUP BY 子句中使用的任何列名都必须在 SELECT 语句中；表达式可以是聚合。此外，未在 GROUP BY 子句中出现的列名称不能在 SELECT 语句中出现，除非是在聚合中，或者，如果对列的访问权限可从您在 GROUP BY 子句中指定的列中创建（如上所示）。

例如，以下用于计算每小时产品计数的查询使用单调表达式 FLOOR（orders.rowTime TO HOUR）因此有效：

```
SELECT STREAM FLOOR(Orders.ROWTIME TO HOUR) AS theHour, prodId, COUNT(*)
FROM Orders
GROUP BY FLOOR(Orders.ROWTIME TO HOUR), prodId
```

GROUP BY 中的一个表达式必须是单调的或基于时间的。例如，GROUP BY FLOOR (S.ROWTIME) TO HOUR 将为前一小时的输入行生成每小时一行输出。GROUP BY 可以指定其他分区术语。例如，在 GROUP BY FLOOR (S.ROWTIME) TO HOUR 中，每个 USERID 值每小时将生成一个输出行。如果你知道一个表达式是单调的，你可以使用单调函数 (p. 191). 如果实际数据不是单调的，则由此产生的系统行为是不确定的：结果可能不符合预期或预期。

参见主题单调函数 (p. 191)有关更多信息，请参阅。

流中可能会出现重复的行时间，只要 ROWTIME 值相同，GROUP BY 操作就会继续累积行。为了发出一行，ROWTIME 值必须在某个时候发生变化。

WHERE 子句

WHERE 子句提取满足指定条件的记录。条件可以是数值或字符串比较，也可以使用 BETWEEN、LIKE 或 IN 运算符：参见串流 SQL 操作符 (p. 6). 可以使用 AND、OR 和 NOT 等逻辑运算符组合条件。

WHERE 子句就像HAVING 子句 (p. 55)子句。它适用于组，也就是说，WHERE 子句的结果是单独的原始行，而 HAVING 子句的结果表示原始行的分组或聚合。

WHERE 和 HAVING 都可以出现在单个 SELECT 语句中。WHERE 从流或表中选择满足其条件（WHERE 条件）的各个行。GROUP BY 条件仅适用于由 WHERE 条件选择的行。这样的分组，例如 “GROUP BY CustomerID”，可以通过有条件进一步限定，然后在指定分组中选择满足其条件的行的聚合。例如，“按 ClientID 分组有总和 (ShipmentValue) > 3600" 将只选择那些符合WHERE标准的各种货物的总价值也超过3,600的客户。

要查看此子句在 SELECT 语句中的位置，请参见SELECT (p. 35).

条件必须是布尔谓词表达式。该查询仅返回谓词计算结果为 TRUE 的行；如果条件的计算结果为 NULL，则不发送该行。

WHERE 子句中的条件不能包含窗口聚合表达式，因为如果 where 子句条件导致行被删除，它将改变窗口的内容。

主题中还讨论了哪里加入子句 (p. 41)和HAVING 子句 (p. 55)。

WINDOW 子句 (滑动窗口)

这些区域有：WINDOW滑动窗口查询的子句指定了相对于当前行在一组行中计算分析函数的行。这些聚合函数生成一个输出行，该行由每个输入行的一列或多列中的键聚合。这些区域有：WINDOW查询中的子句指定按时间范围间隔或行数分区的流中的记录，以及由PARTITION BY子句。您可以定义可用于分析函数和流式处理的命名或内联窗口规范JOIN子句。有关分析函数的更多信息，请参阅分析函数 (p. 97)。

滑动窗口查询中的聚合函数是在滑动窗口中指定的每列上执行的OVER子句。这些区域有：OVER子句可以引用已命名的窗口规范，也可以作为SELECT泵的声明。以下示例说明如何使用 OVER 子句引用指定的窗口规范并在 SELECT 语句中内联使用。

语法

```
[WINDOW window_name AS
(
 {PARTITION BY partition_name
  RANGE INTERVAL 'interval' {SECOND | MINUTE | HOUR} PRECEDING |
  ROWS number PRECEDING
, …}
)
```

OVER 子句

以下示例说明如何使用 OVER 子句引用指定的窗口规范。

示例 1：OVER 引用命名窗口规范

以下示例显示了一个聚合函数，该函数引用了名为 W1 的窗口规范。在此示例中，平均价格是根据指定的一组记录计算得出的W1窗口规格。要了解有关如何将 OVER 子句与窗口规范结合使用的更多信息，请参阅示例 (p. 58)。

```
AVG(price) OVER W1 AS avg_price
```

示例 2：OVER 引用内联窗口规范

以下示例显示了一个引用内联窗口规范的聚合函数。在此示例中，使用行内窗口规格计算每个输入行的平均价格。要了解有关如何将 OVER 子句与窗口规范结合使用的更多信息，请参阅示例 (p. 58)。

```
AVG(price) OVER (
    PARTITION BY ticker_symbol
    RANGE INTERVAL '1' HOUR PRECEDING) AS avg_price
```

有关聚合函数和 OVER 子句的更多信息，请参阅聚合函数 (p. 69)。

参数

window-name

指定可从 OVER 子句或后续窗口定义中引用的唯一名称。该名称用于分析函数和流式传输JOIN子句。有关分析函数的更多信息，请参阅分析函数 (p. 97)。

AS

为 WINDOW 子句定义指定的窗口规范。

PARTITION BY partition-name

将行分成共享相同值的组。在对行进行划分后，窗口函数将计算当前行所在划分中的所有行。

RANGE INTERVAL 'interval' {SECOND | MINUTE | HOUR} PRECEDING

从时间范围间隔开始指定窗口边界。窗口函数将计算具有与当前行相同的时间间隔的所有行。

ROWS number PRECEDING

根据行数指定窗口边界。窗口函数将计算在相同行数内的所有行。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是其中的一部分开始使用在Amazon Kinesis Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics App 和配置示例股票行情输入流，请参阅开始使用在Amazon Kinesis Analytics. 有关其他示例，请参阅滑动窗口。

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
```

```
change          REAL,
price           REAL)
```

示例 1：基于时间的滑动窗口，引用了命名窗口规范

此示例定义了一个命名窗口规范，其分区边界在当前行之前一分钟。数据泵的 OVER 语句的 SELECT 子句引用指定的窗口规范。

```
WINDOW W1 AS (
    PARTITION BY ticker_symbol
    RANGE INTERVAL '1' MINUTE PRECEDING);
```

要运行此示例，请创建股票示例应用程序，并运行和保存以下 SQL 代码。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    min_price     DOUBLE,
    max_price     DOUBLE,
    avg_price     DOUBLE);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM ticker_symbol,
        MIN(price) OVER W1 AS min_price,
        MAX(price) OVER W1 AS max_price,
        AVG(price) OVER W1 AS avg_price
    FROM "SOURCE_SQL_STREAM_001"
    WINDOW W1 AS (
        PARTITION BY ticker_symbol
        RANGE INTERVAL '1' MINUTE PRECEDING);
```

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL	MIN_PRICE
2017-01-31 23:30:11.661	QXZ	215.0399932861328
2017-01-31 23:30:16.673	IOP	118.41999816894531
2017-01-31 23:30:16.673	AMZN	727.469970703125
2017-01-31 23:30:16.673	AMZN	713.0900268554688
2017-01-31 23:30:16.673	TBV	164.00999450683594

示例 2：基于行的滑动窗口，引用了命名窗口规范

此示例定义了一个命名窗口规范，其分区边界为当前行前两行，当前行前十行。数据泵的 OVER 语句的 SELECT 子句引用指定的窗口规范。

```
 WINDOW
    last2rows AS (PARTITION BY ticker_symbol ROWS 2 PRECEDING),
```

```
    last10rows AS (PARTITION BY ticker_symbol ROWS 10 PRECEDING);
```

要运行此示例，请创建股票示例应用程序，并运行和保存以下 SQL 代码。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol      VARCHAR(4),
    price              DOUBLE,
    avg_last2rows      DOUBLE,
    avg_Last10rows     DOUBLE);
CREATE OR REPLACE PUMP "myPump" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM ticker_symbol,
    price,
    AVG(price) OVER last2rows,
    AVG(price) OVER last10rows
FROM SOURCE_SQL_STREAM_001
WINDOW
    last2rows AS (PARTITION BY ticker_symbol ROWS 2 PRECEDING),
    last10rows AS (PARTITION BY ticker_symbol ROWS 10 PRECEDING);
```

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL	PRICE	AVERA
2017-01-31 23:43:06.414	QXZ	75.33999633789062	114.71
2017-01-31 23:43:06.414	SLW	75.44000244140625	77.423
2017-01-31 23:43:06.414	SAC	41.709999084472656	42.950
2017-01-31 23:43:06.414	QWE	223.1999969482422	221.07
2017-01-31 23:43:06.414	WAS	14.039999961853027	13.993

示例 3：带内嵌窗口规范的基于时间的滑动窗规范

此示例定义了一个行内窗口规范，其分区边界在当前行之前一分钟。数据泵的 `OVER` 语句的 `SELECT` 子句使用内联窗口规范。

要运行此示例，请创建股票示例应用程序，并运行和保存以下 SQL 代码。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    price         DOUBLE,
    avg_price     DOUBLE);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM ticker_symbol, price,
        AVG(Price) OVER (
            PARTITION BY ticker_symbol
            RANGE INTERVAL '1' HOUR PRECEDING) AS avg_price
    FROM "SOURCE_SQL_STREAM_001"
```

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-02 19:41:21.621	TBV
2017-02-02 19:41:21.621	JKL
2017-02-02 19:41:21.621	JKL
2017-02-02 19:41:21.621	QXZ
2017-02-02 19:41:21.621	WSB

使用说明

对于 WINDOW 子句和终端节点，Amazon Kinesis Analytics SQL 在某个范围内遵循窗口的 SQL-2008 标准。

要包括 1 小时的终端节点，您可以使用以下窗口语法。

```
WINDOW HOUR AS (RANGE INTERVAL '1' HOUR PRECEDING)
```

要排除上一个小时的终端节点，您可以使用以下窗口语法。

```
WINDOW HOUR AS (RANGE INTERVAL '59:59.999' MINUTE TO SECOND(3) PRECEDING);
```

有关更多信息，请参阅 允许和禁止的窗口规范 (p. 61)。

相关主题

- 《Kinesis 开发人员指南》中的滑动窗口
- 聚合函数 (p. 69)
- SELECT (p. 35)
- 创建流 (p. 30) statement
- 创建转储 (p. 32) statement

允许和禁止的窗口规范

Amazon Kinesis Data Analytics 几乎支持所有以当前行结尾的窗口。

不能定义无限窗口、负大小的窗口，也不能在窗口规范中使用负整数。目前不支持偏移窗口。

- 无限窗口是没有界限的窗口。通常，这些指向future，对于直播来说，未来是无限的。例如，不支持 “当前行与未绑定关注之间的行”，因为在流式上下文中，这样的查询不会产生结果，因为随着新数据的到来，流会不断扩展。不支持 UNBOUNDED FOLLING 的所有用法。
- 逆向窗口。例如，“前 0 和前 4 之间的行” 是一个大小为负的窗口，因此是非法的。您可以改为使用：在本例中为 “前 4 行和前 0 之间的行”。

- 偏移窗口是指不以 “当前行” 结尾的窗口。当前版本不支持这些。例如，不支持 “前面无界限和后面 4 行之间的行”。（窗口跨越当前行，而不是从该行开始或结束。）
- 使用负整数定义的窗口。例如，“前 -4 行与当前行之间的行” 无效，因为不允许使用负整数。

此外，... 前面的 0（以及... 0 FOLLING）不能用于窗口聚合；相反，可以使用同义词 CURRENT ROW。

对于窗口聚合，允许使用分区窗口，但不得存在 ORDER BY。

对于窗口联接，不允许使用分区窗口，但如果 ORDER BY 按其中一个输入的 ROWTIME 列排序，则可以显示 ORDER BY。

窗口示例

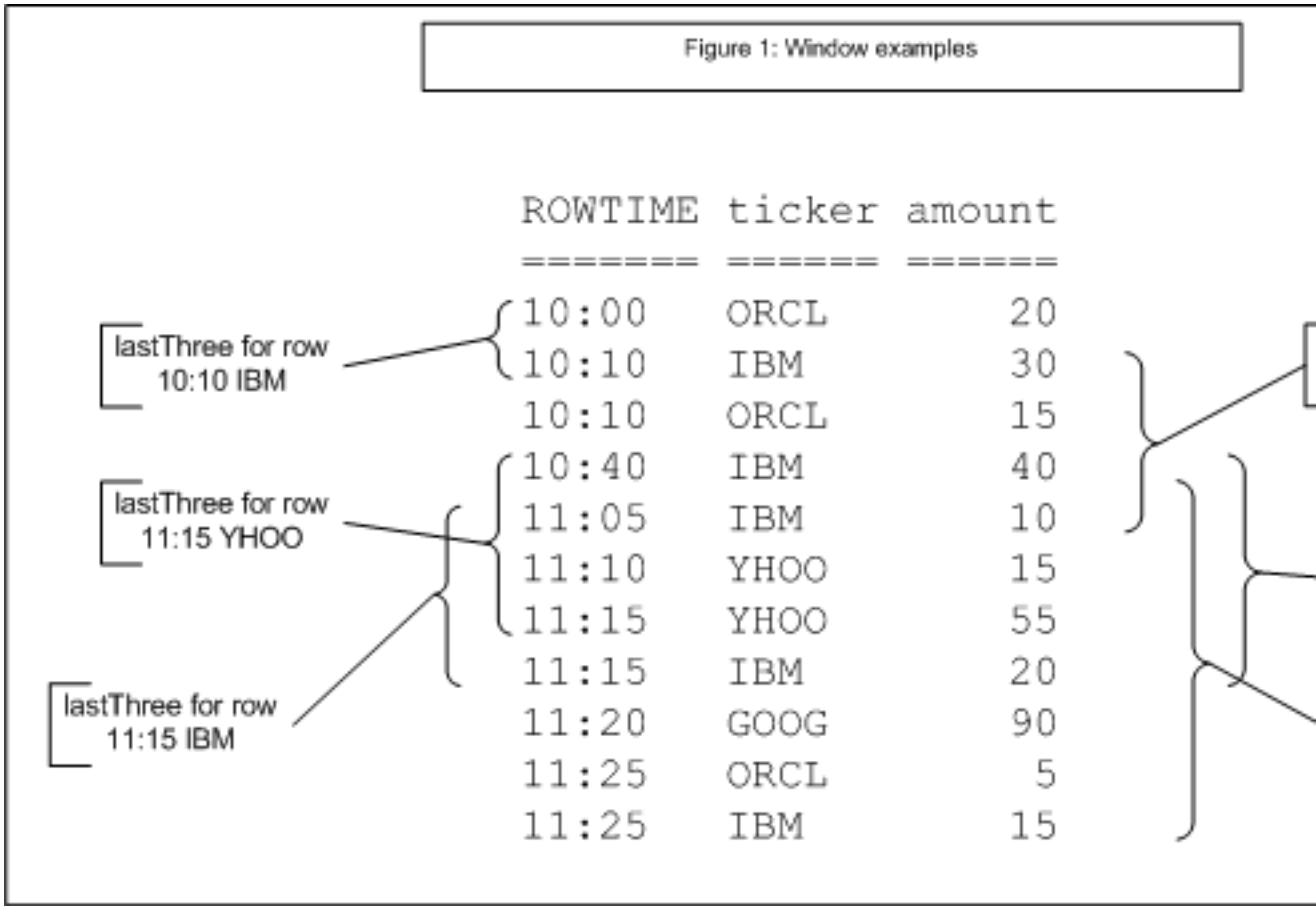
以下示例显示了一个示例输入数据集、多个窗口的定义以及这些窗口在 10:00 之后的不同时间（本示例开始到达的时间数据）的内容。

窗口的定义如下：

```
SELECT STREAM
  ticker,
  sum(amount) OVER lastHour,
  count(*) OVER lastHour
  sum(amount) OVER lastThree
FROM Trades
WINDOW
  lastHour AS (RANGE INTERVAL '1' HOUR PRECEDING),
  lastThree AS (ROWS 3 PRECEDING),
  lastZeroRows AS (ROWS CURRENT ROW),
  lastZeroSeconds AS (RANGE CURRENT ROW),
  lastTwoSameTicker AS (PARTITION BY ticker ROWS 2 PRECEDING),
  lastHourSameTicker AS (PARTITION BY ticker RANGE INTERVAL '1' HOUR PRECEDING)
```

第一个示例：基于时间的窗口与基于行的窗口

如下图右侧所示，基于时间的 LastHour 窗口包含不同数量的行，因为窗口成员资格是由时间范围定义的。



包含行的窗口示例

基于行的 lastThree 窗口通常包含四行：前三行和当前行。但是，对于第 10:10 行 IBM，它只包含两行，因为 10:00 之前没有数据。

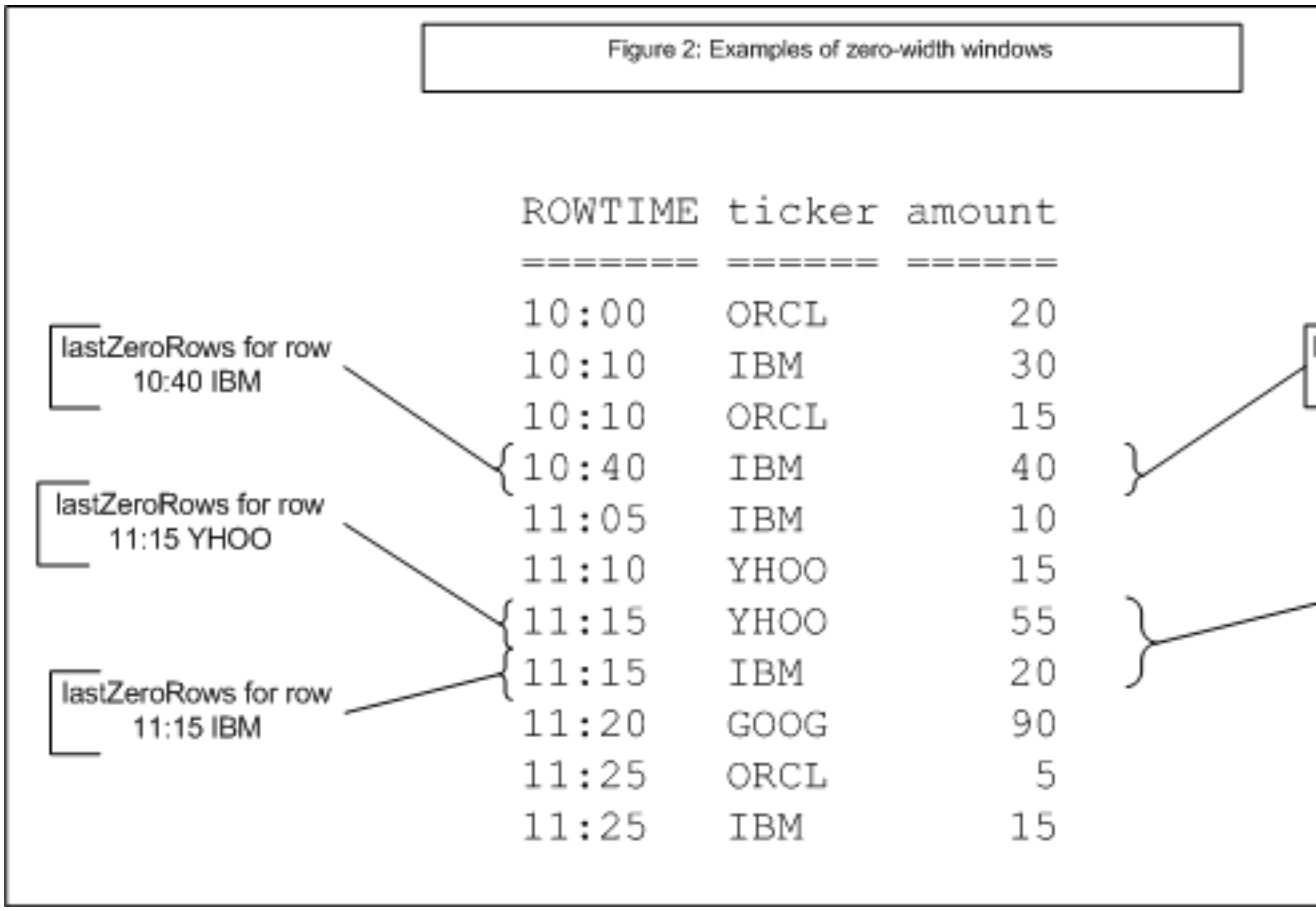
基于行的窗口可以包含多行，这些行的 ROWTIME 值相同，尽管它们到达的时间（挂钟时间）不同。此类行在基于行的窗口中的顺序取决于其到达时间；实际上，该行的到达时间可以决定哪个窗口包含该行。

例如，图 1 中中间的 LastThree 窗口显示了 ROWTIME 11:15 的 YHOO 交易的到来（以及之前的最后三笔交易）。但是，这个窗口不包括IBM的下一笔交易，它的ROWTIME也是 11:15，但肯定比YHOO交易晚了。这笔 11:15 的 IBM 交易包含在 “下一步” 窗口中，其前身 11:15 YHOO 交易也是如此。

第二个示例：零宽度窗口，基于行和基于时间

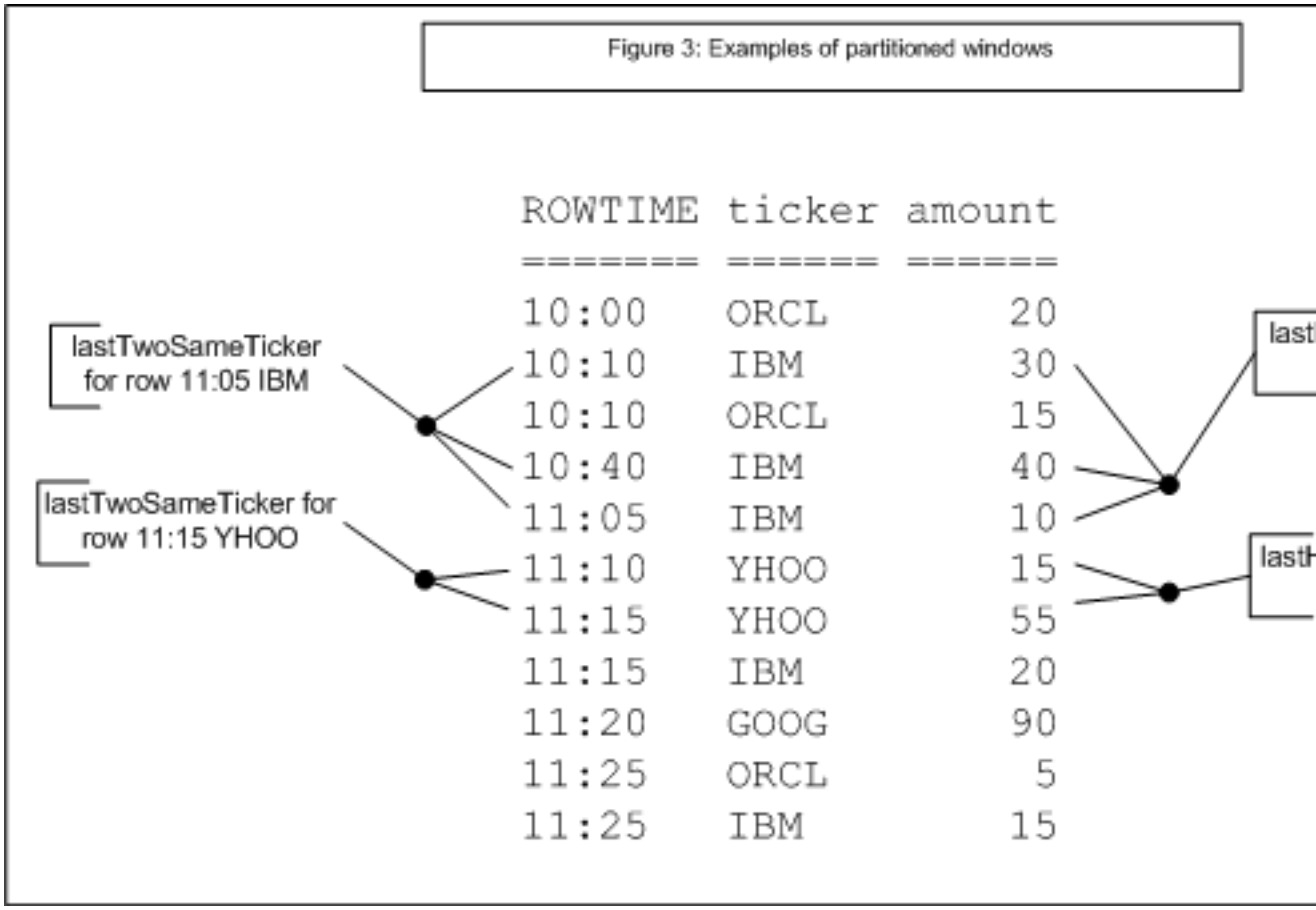
图 2：零宽度窗口的示例显示了宽度为零的基于行和基于时间的窗口。基于行的窗口 lastZeroRows 仅包含当前行，因此始终精确地包含一行。请注意，ROWS CURRENT ROW 等同于前面的行 0。

基于时间的窗口 lastZeroSeconds 包含所有具有相同时间戳的行，其中可能有几行。请注意，RANGE CURRENT ROW 等同于之前的范围间隔 '0' 秒。



第三个例子：分区应用于基于行和基于时间的窗口

图 3 显示了与图 1 中的窗口相似但带有 PARTITION BY 子句的窗口。适用于基于时间的时间窗口 lastTwoSame股票行号和基于行的窗口 lastHourSameTicker，该窗口包含符合窗口标准且与股票行情列值相同的行。注意：分区在窗口之前进行评估。



ORDER BY 子句

如果流式查询的前导表达式是基于时间且单调的，则该流式查询可以使用 ORDER BY。例如，前导表达式基于 ROWTIME 列的流式查询可以使用 ORDER BY 执行以下操作：

- 对直播分组的结果进行排序。
- 对在流的固定时间窗口内到达的一批行进行排序。
- 在 windowed-joins 上执行直播 ORDER BY。

对前导表达式的 “基于时间和单调” 的要求意味着查询

```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM DISTINCT ticker FROM trades ORDER BY ticker
```

会失败，但是查询

```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM DISTINCT rowtime, ticker FROM trades ORDER BY ROWTIME, ticker
```

会成功。

> **Note**
>
> 前面的示例使用 `DISTINCT` 子句从结果集中删除同一股票代码的重复实例，以便结果是单调的。

Streaming ORDER BY 使用符合 SQL-2008 标准的 ORDER BY 子句语法对行进行排序。它可以与 UNION ALL 语句组合使用，并且可以对表达式进行排序，例如：

```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM x, y FROM t1
UNION ALL
SELECT STREAM a, b FROM t2 ORDER BY ROWTIME, MOD(x, 5)
```

ORDER BY 子句可以指定升序或降序排序顺序，可以使用列序号和序数指定（引用）项目在选择列表中的位置。

Note

上述查询中的 UNION 语句从两个单独的流中收集记录以进行排序。

按 SQL 声明排列直播顺序

串流 ORDER BY 子句包括以下功能属性：

- 收集行，直到流式处理 ORDER BY 子句中的单调表达式没有改变。
- 不需要在同一语句中流式传输 GROUP BY 子句。
- 可以使用基本 SQL 数据类型为 TIMESTAMP、DATE、DECIMAL、INTEGER、FLOAT、CHAR、VARCHAR 的任何列。
- 不要求 ORDER BY 子句中的列/表达式出现在语句的 SELECT 列表中。
- 应用 ORDER BY 子句的所有标准 SQL 验证规则。

以下查询是串流 ORDER BY 的示例：

```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM state, city, SUM(amount)
FROM orders
GROUP BY FLOOR(ROWTIME TO HOUR), state, city
ORDER BY FLOOR(ROWTIME TO HOUR), state, SUM(amount)
```

T 排序流输入

Amazon Kinesis Data Analytics 实时分析使用了这样一个事实，即到达的数据由 ROWTIME 排序。但是，有时来自多个来源的数据可能不会在时间上同步。

虽然 Amazon Kinesis Data Analytics 可以对来自已独立插入到 Amazon Kinesis Data Analytics 应用程序原生流中的单个数据源的数据进行排序，但在某些情况下，此类数据可能已经从多个来源合并在一起（例如为了在早期阶段提高使用效率）处理）。在其他时候，高容量数据源可能会导致无法直接插入。

此外，不可靠的数据源可能会迫使 Amazon Kinesis Data Analytics 应用程序无限期等待，直到所有连接的数据源都交付后才能继续，从而阻碍进度。在这种情况下，来自该源的数据可能不同步。

您可以使用 ORDER BY 子句来解决这些问题。Amazon Kinesis Data Analytics 使用基于滑动时间的传入行窗口，按 ROWTIME 对这些行进行重新排序。

语法

您可以使用以下语法指定用于排序的基于时间的参数和对流式行进行时间排序的基于时间的窗口：

```
ORDER BY <timestamp_expr> WITHIN
    <interval_literal>
```

Restrictions

T 排序有以下限制：

- ORDER BY 表达式的数据类型必须是时间戳。
- 部分排序的表达式 <timestamp_expr> 必须存在于别名为 ROWTIME 的查询的选择列表中。
- ORDER BY 子句的前导表达式不得包含 ROWTIME 函数，也不得使用 DESC 关键字。
- ROWTIME 列需要完全合格。例如：

- `ORDER BY FLOOR(ROWTIME TO MINUTE), ...`失败。
- `ORDER BY FLOOR(s.ROWTIME TO MINUTE), ...`作品。

如果未满足其中任何要求，该语句将失败并出现错误。

附加说明：

- 您不能使用传入的行时界限。这些被系统忽略。
- 如果 <timestamp_expr> 的计算结果为 NULL，对应的行将被舍弃。

ROWTIME

ROWTIME 是一个运算符和系统列，它返回创建流中特定行的时间。

它以四种不同的方式使用：

- 作为操作员
- 作为直播的系统列
- 作为列别名，用于覆盖当前行上的时间戳
- 作为表中的普通列

有关更多详细信息，请参阅主题时间戳、ROWTIME 和当前_ROW_TIMESTAMP (p. 132)。

行时运算符

在流式查询的 SELECT 子句中使用时，无需由前面的 “别名” 限定。，ROWTIME 是一个运算符，其计算结果为即将生成的行的时间戳。

它的类型始终是时间戳不是 NULL。

ROWTIME 系统列

每个直播都有一个 ROWTIME 列。要从查询中引用此列，请使用流名称（或别名）对其进行限定。例如，以下联接查询返回三个时间戳列：其输入流的系统列和生成行的时间戳。

```
SELECT STREAM
  o.ROWTIME AS leftRowtime,
  s.ROWTIME AS rightRowtime,
  ROWTIME AS joinRowtime
FROM Orders AS o
  JOIN Shipments OVER (RANGE INTERVAL '1' HOUR FOLLOWING) AS s
  ON o.orderId = s.orderId

leftRowtime         rightRowtime        joinRowtime
=================== =================== ===================
2008-02-20 10:15:00 2008-02-20 10:30:00 2008-02-20 10:15:00
```

```
2008-02-20 10:25:00 2008-02-20 11:15:00 2008-02-20 10:25:00
2008-02-20 10:25:30 2008-02-20 11:05:00 2008-02-20 10:25:30
```

碰巧的是，LeftRowTime 始终等于 joinRowTime，因为指定连接时输出行的时间戳始终等于 Orders 流中的 ROWTIME 列。

因此，每个流式查询都有一个 ROWTIME 列。但是，除非您在 SELECT 子句中明确包含 ROWTIME 列，否则不会从顶级 JDBC 查询中返回 ROWTIME 列。例如：

```
CREATE STREAM Orders(
  "orderId" INTEGER NOT NULL,
  "custId" INTEGER NOT NULL);
SELECT columnName
FROM ALL_STREAMS;

columnName
==========
orderId
custId

SELECT STREAM *
FROM Orders;

orderId custId
======= ======
    100    501
    101     22
    102    699

SELECT STREAM ROWTIME, *
FROM Orders;

ROWTIME             orderId custId
=================== ======= ======
2008-02-20 10:15:00     100    501
2008-02-20 10:25:00     101     22
2008-02-20 10:25:30     102    699
```

这主要是为了确保与 JDBC 的兼容性：流订单声明了两列，因此 “SELECT STREAM *” 应该返回两列是有道理的。

函数

本节主题介绍流式处理 SQL 支持的函数。

主题

聚合函数

聚合函数不是返回从单行计算得出的结果，而是返回根据有限行集合中包含的聚合数据或有关有限行集的信息计算得出的结果。聚合函数可能出现在以下任何内容中：

- SELECT 子句 (p. 37)的 <selection list> 部分
- ORDER BY 子句 (p. 65)
- HAVING 子句 (p. 55)

聚合函数不同于分析函数 (p. 97)，它们总是相对于必须指定的窗口进行求值，因此它们不能出现在 HAVING 子句中。本主题后面的表格中描述了其他差异。

聚合函数在表聚合查询中的操作方式与在流的聚合查询中使用聚合函数时的操作方式略有不同，如下所示。如果对表的聚合查询包含 GROUP BY 子句，则聚合函数在输入行集合中为每组返回一个结果。缺少显式 GROUP BY 子句等同于 GROUP BY ()，并且只为整组输入行返回一个结果。

在流上，聚合查询必须包含基于行时间的单调表达式上的显式 GROUP BY 子句。如果没有一个，唯一的组就是整个直播，它永远不会结束，无法报告任何结果。添加基于单调表达式的 GROUP BY 子句将流分解为时间连续的有限行集，然后可以聚合和报告每个这样的集合。

每当到达的行改变了单调分组表达式的值时，就会启动一个新组并认为前一组已完成。然后，Amazon Kinesis Data Analytics 应用程序输出聚合函数的值。请注意，GROUP BY 子句还可能包括其他非单调表达式，在这种情况下，每组行可能会产生多个结果。

对流执行聚合查询通常被称为流式聚合，这与中讨论的窗口化聚合不同分析函数 (p. 97)和直播窗口聚合 (p. 73). 有关更多信息 stream-to-stream 联接，请参阅加入子句 (p. 41).

如果输入行包含null在用作数据分析函数输入的列中，数据分析函数忽略该行（COUNT 除外）。

聚合函数和分析函数之间的区别

函数类型	输出	使用的行数或窗口	注意
聚合函数	每组输入行一个输出行。	所有输出列都是在同一个窗口或同一组行上计算的。	串流聚合中不允许计数不同。不允许使用以下类型的语句： SELECT COUNT(DISTINCT x) ... FROM ... GROUP BY ...
分析函数 (p. 97)	每个输入行对应一个输出行。	可以使用不同的窗口或分区来计算每个输出列。	COUNT DISTINCT 不能用作分析函数 (p. 97)或者在窗口聚合中。

流式聚合和行时界限

通常，当一行到达时，聚合查询会生成一个结果，该结果会改变 GROUP BY 中单调表达式的值。例如，如果查询按 FLOOR（行时间到分钟）分组，而当前行的行时间为 9:59 .30，则行时间为 10:00 .00 的新行将触发结果。

或者，可以使用行时界限来推进单调表达式并使查询返回结果。例如，如果查询按 FLOOR（行时间到分钟）分组，而当前行的行时间为 9:59 .30，则传入的行时间界限为 10:00 .00，则查询将返回结果。

聚合函数列表

Amazon Kinesis Data Analytics 支持以下聚合函数：

- AVG (p. 77)
- COUNT (p. 80)
- COUNT_DISCT_ITEMS_TUMBLING 函数 (p. 83)
- EXP_AVG (p. 85)
- FIRST_VALUE (p. 85)
- LAST_VALUE (p. 86)
- MAX (p. 86)
- MIN (p. 89)
- SUM (p. 92)
- TOP_K_ITEMS_TUMBLING 函数 (p. 95)

以下 SQL 使用 AVG 聚合函数作为查询的一部分来查找所有员工的平均年龄：

```
SELECT
    AVG(AGE) AS AVERAGE_AGE
FROM SALES.EMPS;
```

结果：

AVERAGE_AGE
38

要计算每个部门员工的平均年龄，我们可以在查询中添加明确的 GROUP BY 子句：

```
SELECT
    DEPTNO,
    AVG(AGE) AS AVERAGE_AGE
FROM SALES.EMPS
GROUP BY DEPTNO;
```

返回值:

DEPTNO	AVERAGE_AGE
10	30
20	25
30	40
40	57

流上聚合查询（流式聚合）的示例

在此示例中，假设下表中的数据正在流经名为 WEATHERSTREAM 的流。

ROWTIME	CITY	TEMP
2018-11-00	丹佛	29
2018-11-00	安克雷奇	2
2018-11-00	迈阿密	65
2018-11-00	丹佛	32
2018-11-0	安克雷奇	9
2018-11-0	丹佛	50
2018-11-00	安克雷奇	10
2018-11-0	迈阿密	71
2018-11-0	丹佛	43
2018-11-00	安克雷奇	4
2018-11-00	丹佛	39
2018-11-00	丹佛	46
2018-11-00	安克雷奇	3

ROWTIME	CITY	TEMP
2018-11-00	丹佛	56
2018-11-00	安克雷奇	2
2018-11-00	丹佛	50
2018-11-00	丹佛	36
2018-11-00	安克雷奇	1

如果要查找每天任何地方（无论城市如何，全球）记录的最低和最高温度，可以分别使用聚合函数 MIN 和 MAX 来计算最低和最高温度。要表示我们每天需要此类信息（以及要提供单调表达式作为 GROUP BY 子句的参数），我们使用 FLOOR 函数将每个行的行时间向下取整为最近一天：

```
SELECT STREAM

    FLOOR(WEATHERSTREAM.ROWTIME to DAY) AS FLOOR_DAY,
    MIN(TEMP) AS MIN_TEMP,
    MAX(TEMP) AS MAX_TEMP
FROM WEATHERSTREAM

GROUP BY FLOOR(WEATHERSTREAM.ROWTIME TO DAY);
```

聚合查询的结果见下表。

FLOOR_DAY	MIN_TEMP	MAX_TEMP
2018-11-01 00:00:00 .0	2	71
2018-11-00	2	56

尽管示例数据确实包含当天的温度测量值，但2018-11-03没有行。这是因为在已知当天的所有行都已到达之前，无法对2018-11-03的行进行聚合，并且只有当行时间为2018-11-04 00:00:00 .0（或更晚）的行到达时，才会发生这种情况。如果两者何时到达，下一个结果将如下表所示。

FLOOR_DAY	MIN_TEMP	MAX_TEMP
2018-11-03 00:00:00 .0	1	36

假设我们想要找到每个城市每天的最低、最高和平均温度，而不是寻找全球每天的最低和最高气温。为此，我们使用 SUM 和 COUNT 聚合函数来计算平均值，并将 CITY 添加到 GROUP BY 子句中，如下所示：

```
SELECT STREAM
    FLOOR(WEATHERSTREAM.ROWTIME TO DAY) AS FLOOR_DAY,

    CITY,
    MIN(TEMP) AS MIN_TEMP,
    MAX(TEMP) AS MAX_TEMP,
    SUM(TEMP)/COUNT(TEMP) AS AVG_TEMP
FROM WEATHERSTREAM
GROUP BY FLOOR(WEATHERSTREAM.ROWTIME TO DAY), CITY;
```

聚合查询的结果见下表。

FLOOR_DAY	CITY	MIN_TEMP	MAX_TEMP	AVG_TEMP
2018-11-01 00:00:00 .0	安克雷奇	2	10	7
2018-11-01 00:00:00 .0	丹佛	29	50	38
2018-11-01 00:00:00 .0	迈阿密	65	71	68
2018-11-00	安克雷奇	2	4	3
2018-11-00	丹佛	39	56	47

在这种情况下，新一天温度测量值的行的到来会触发前一天数据的聚合，按城市分组，然后生成当天测量中包含的每个城市一行。

同样，在下表显示了2018-11-04的任何实际测量结果之前，可以使用2018-11-04 00:00:00 .0 来提示2018-11-03的结果。

FLOOR_DAY	CITY	MIN_TEMP	MAX_TEMP	AVG_TEMP
2018-11-03 00:00:00 .0	安克雷奇	1	1	1
2018-11-03 00:00:00 .0	丹佛	36	36	36

直播窗口聚合

为了说明窗口聚合在 Amazon Kinesis 数据流上的工作原理，假设下表中的数据正在流经名为 WEATHERSTREAM 的数据流。

ROWTIME	CITY	TEMP
2018-11-01	丹佛	29
2018-11-01	安克雷奇	2
2018-11-01	迈阿密	65
2018-11-01	丹佛	32
2018-11-01	安克雷奇	9
2018-11-01	丹佛	50
2018-11-01	安克雷奇	10

ROWTIME	CITY	TEMP
2018-11-01	迈阿密	71
2018-11-01	丹佛	43
2018-11-01	安克雷奇	4
2018-11-01	丹佛	39
2018-11-01	丹佛	46
2018-11-01	安克雷奇	3
2018-11-01	丹佛	56
2018-11-01	安克雷奇	2
2018-11-01	丹佛	50
2018-11-01	丹佛	36
2018-11-01	安克雷奇	1

假设您要查找 24 小时期间内记录的全球（无论是哪个城市）的最低和最高温度（在任意给定读数之前）。为此，您需要定义 `RANGE INTERVAL '1' DAY PRECEDING` 的一个窗口，并在 `MIN` 和 `MAX` 分析函数的 `OVER` 子句中使用它：

```
SELECT STREAM
       ROWTIME,
       MIN(TEMP) OVER W1 AS WMIN_TEMP,
       MAX(TEMP) OVER W1 AS WMAX_TEMP
FROM WEATHERSTREAM
WINDOW W1 AS (
   RANGE INTERVAL '1' DAY PRECEDING
);
```

结果

ROWTIME	WMIN_TEMP	WMAX_TEMP
2018-11-01	29	29
2018-11-01	2	29
2018-11-01	2	65
2018-11-01	2	65
2018-11-01	2	65
2018-11-01	2	65
2018-11-01	2	65
2018-11-01	2	71
2018-11-01	2	71

ROWTIME	WMIN_TEMP	WMAX_TEMP
2018-11-01	2	71
2018-11-01	2	71
2018-11-01	4	71
2018-11-01	3	71
2018-11-01	3	71
2018-11-01	2	71
2018-11-01	2	56
2018-11-01	2	56
2018-11-01	1	56

现在，假定您要查找在 24 小时期间内记录的在任意给定读数之前的最低、最高和平均温度（按城市划分）。为此，您可以向窗口规范添加针对 `CITY` 的 `PARTITION BY` 子句，并向选择列表添加针对同一个窗口的 `AVG` 分析函数：

```
SELECT STREAM
       ROWTIME,
       CITY,
       MIN(TEMP) over W1 AS WMIN_TEMP,
       MAX(TEMP) over W1 AS WMAX_TEMP,
       AVG(TEMP) over W1 AS WAVG_TEMP
FROM AGGTEST.WEATHERSTREAM
WINDOW W1 AS (
       PARTITION BY CITY
       RANGE INTERVAL '1' DAY PRECEDING
);
```

结果

ROWTIME	CITY	WMIN_TEMP	WMAX_TEMP	WAVG_TEMP
2018-11-01	丹佛	29	29	29
2018-11-01	安克雷奇	2	2	2
2018-11-01	迈阿密	65	65	65
2018-11-01	丹佛	29	32	30
2018-11-01	安克雷奇	2	9	5
2018-11-01	丹佛	29	50	37
2018-11-01	安克雷奇	2	10	7
2018-11-01	迈阿密	65	71	68
2018-11-01	丹佛	29	50	38
2018-11-01	安克雷奇	2	10	6

ROWTIME	CITY	WMIN_TEMP	WMAX_TEMP	WAVG_TEMP
2018-11-01	丹佛	29	50	38
2018-11-01	丹佛	32	50	42
2018-11-01	安克雷奇	3	10	6
2018-11-01	丹佛	39	56	46
2018-11-01	安克雷奇	2	10	4
2018-11-01	丹佛	39	56	46
2018-11-01	丹佛	36	56	45
2018-11-01	安克雷奇	1	4	2

行时界限和窗口聚合示例

这是窗口聚合查询的示例：

```
SELECT STREAM ROWTIME, ticker, amount, SUM(amount)
   OVER (
       PARTITION BY ticker
       RANGE INTERVAL '1' HOUR PRECEDING)
AS hourlyVolume
FROM Trades
```

由于这是对流的查询，因此行一进入就会从该查询中弹出。例如，给定以下输入：

```
Trades: IBM 10 10 10:00:00
Trades: ORCL 20 10:10:00
Trades.bound: 10:15:00
Trades: ORCL 15 10:25:00
Trades: IBM 30 11:05:00
Trades.bound: 11:10:00
```

在此示例中，输出如下所示：

```
Trades: IBM 10 10 10:00:00
Trades: ORCL 20 20 10:10:00
Trades.bound: 10:15:00
Trades: ORCL 15 35 10:25:00
Trades: IBM 30 30 11:05:00
Trades.bound: 11:10:00
```

这些行仍然在后台逗留一个小时，因此第二个 ORCL 行输出的总计为 35；但原始 IBM 交易落在“上一个小时”窗口外，因此它未包含在 IBM 总计中。

示例

有些业务问题似乎需要对直播的整个历史记录进行总结，但这通常不切实际地进行计算。但是，此类业务问题通常可以通过查看最后一天、最后一小时或最后 N 条记录来解决。此类记录的集合称为窗口化聚合。

它们易于在流数据库中计算，可以用 ANSI (SQL: 2008) 标准 SQL 表示，如下所示：

```
SELECT STREAM ticker,
```

```
    avg(price) OVER lastHour AS avgPrice,
    max(price) OVER lastHour AS maxPrice
FROM Bids
WINDOW lastHour AS  (
    PARTITION BY ticker
    RANGE INTERVAL '1' HOUR PRECEDING)
```

Note

`Interval_clause` 必须属于以下合适的类型之一：

- 带有 ROWS 的整数文字
- 数值列上的 RANGE 的数值
- 一个范围在日期/时间/时间戳上的间隔

AVG

从窗口式查询返回一组值的平均值。窗口查询是根据时间或行数定义的。有关窗口式查询的信息，请参阅窗口式查询。要返回在指定时间窗口内选择的值表达式的流的指数加权平均值，请参阅 EXP_AVG (p. 85)。

在使用 AVG 时，请注意以下事项：

- 如果你不使用OVER子句，AVG以聚合函数的形式计算。在这种情况下，聚合查询必须包含GROUP BY 子句 (p. 55)在基于以下条件的单调表达式上ROWTIME它将流分组为有限的行。否则，该组是无限流，查询将永远不会完成，也不会发出任何行。有关更多信息，请参阅 聚合函数 (p. 69)。
- 使用 GROUP BY 子句的窗口式查询处理滚动窗口中的行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您将OVER子句，AVG以分析函数的形式计算。有关更多信息，请参阅 分析函数 (p. 97)。
- 使用 OVER 子句的窗口式查询处理滑动窗口中的行。有关更多信息，请参阅滑动窗口

语法

滚动窗口式查询

```
AVG(number-expression) ... GROUP BY monotonic-expression | time-based-expression
```

滑动窗口式查询

```
AVG([DISTINCT | ALL] number-expression) OVER window-specification
```

参数

DISTINCT

仅对值的每个唯一实例执行聚合函数。

ALL

对所有值执行聚合函数。ALL 为默认值。

number-expression

指定针对聚合中的每一行计算的值表达式。

OVER window-specification

将流中的记录除以时间范围间隔或行数分区。窗口规范定义流中记录的划分方式 (按时间范围间隔或行数)。

GROUP B单调表达式|time-based-expression

基于分组表达式的值为记录分组，从而针对在所有列中具有相同值的每组行返回一个摘要行。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是入门练习在里面Amazon Kinesis Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics（Analytics），请参阅开始使用在里面Amazon Kinesis Analytics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：使用 GROUP BY 子句返回值的平均值

在此示例中，聚合查询有一个GROUP BY子句 on onROWTIME它将流分组为有限的行。随后，从 AVG 子句返回的行计算 GROUP BY 函数。

使用 STEP（推荐）

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    avg_price     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        ticker_symbol,
        AVG(price) AS avg_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol,
        STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

使用 FLOOR

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    avg_price     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        ticker_symbol,
        AVG(price) AS avg_price
```

```
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol,
        FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE);
```

结果

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-16 21:46:00.0	NFS
2017-02-16 21:47:00.0	WAS
2017-02-16 21:47:00.0	PPL
2017-02-16 21:47:00.0	ALY

示例 2：使用 OVER 子句返回值的平均值

在此示例中，OVER子句将流中的记录除以前 '1' 小时的时间范围间隔。随后，从 AVG 子句返回的行计算 OVER 函数。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    avg_price     DOUBLE);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM ticker_symbol,
        AVG(price) OVER (
            PARTITION BY ticker_symbol
            RANGE INTERVAL '1' HOUR PRECEDING) AS avg_price
    FROM "SOURCE_SQL_STREAM_001"
```

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-16 21:57:25.509	AAPL
2017-02-16 21:57:25.509	TGT
2017-02-16 21:57:25.509	RFV
2017-02-16 21:57:25.509	SAC

使用说明

Amazon Kinesis AnalyticsAVG应用于间隔类型。此功能偏离了 SQL:2008 标准。

当用作分析函数时，AVG如果正在评估的窗口不包含任何行，或者如果所有行都包含空值，则返回 null。有关更多信息，请参阅 分析函数 (p. 97)。对于 a，AVG 也会返回空值PARTITION BY子句，窗口内与输入行匹配的分区不包含任何行或所有行均为空。有关 PARTITION BY 的更多信息，请参阅 WINDOW 子句 (滑动窗口) (p. 57)。

AVG忽略一组值或数值表达式中的空值。例如，以下各项返回值 2：

- AVG(1, 2, 3) = 2
- AVG(1,null, 2, null, 3, null) = 2

相关主题

- 窗口式查询
- EXP_AVG (p. 85)
- 聚合函数 (p. 69)
- GROUP BY 子句 (p. 55)
- 分析函数 (p. 97)
- 入门练习
- WINDOW 子句 (滑动窗口) (p. 57)

COUNT

返回窗口查询中一组值的限定行数。窗口查询是根据时间或行数定义的。有关窗口式查询的信息，请参阅窗口式查询。

在使用 COUNT 时，请注意以下事项：

- 如果你不使用OVER子句，COUNT以聚合函数的形式计算。在这种情况下，聚合查询必须包含GROUP BY 子句 (p. 55)在基于以下条件的单调表达式上ROWTIME它将流分组为有限的行。否则，该组是无限流，查询将永远不会完成，也不会发出任何行。有关更多信息，请参阅 聚合函数 (p. 69)。
- 使用 GROUP BY 子句的窗口式查询处理滚动窗口中的行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您将OVER子句，COUNT以分析函数的形式计算。有关更多信息，请参阅 分析函数 (p. 97)。
- 使用 OVER 子句的窗口式查询处理滑动窗口中的行。有关更多信息，请参阅滑动窗口

语法

滚动窗口式查询

```
COUNT(number-expression) ... GROUP BY monotonic-expression | time-based-expression
```

滑动窗口式查询

```
COUNT(* | ALL number-expression) OVER window-specification
```

参数

*

对所有行进行计数。

ALL

计算所有行。ALL 是默认值。

number-expression

指定针对聚合中的每一行计算的值表达式。

OVER window-specification

将流中的记录除以时间范围间隔或行数分区。窗口规范定义流中记录的划分方式 (按时间范围间隔或行数)。

GROUP B单调表达式|time-based-expression

基于分组表达式的值为记录分组，从而针对在所有列中具有相同值的每组行返回一个摘要行。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是其中的一部分开始使用在里面Amazon Kinesis Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 创建Analytics开始使用在里面Amazon Kinesis Analytics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：使用 GROUP BY 子句返回值的数量

在此示例中，聚合查询有一个GROUP BY子句ROWTIME它将流分组为有限的行。随后，从 COUNT 子句返回的行计算 GROUP BY 函数。

使用 STEP（推荐）

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    count_price     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        ticker_symbol,
        COUNT(Price) AS count_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

使用 FLOOR

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    count_price     DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
```

```
    SELECT STREAM
        ticker_symbol,
        COUNT(Price) AS count_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE);
```

结果

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-16 23:30:00.0	AAPL
2017-02-16 23:31:00.0	WSB
2017-02-16 23:31:00.0	AAPL
2017-02-16 23:31:00.0	UHN

示例 2：使用 OVER 子句返回值的数量

在此示例中，OVER子句将流中的记录除以前 '1' 小时的时间范围间隔。随后，从 COUNT 子句返回的行计算 OVER 函数。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    count_price     DOUBLE);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM ticker_symbol,
        COUNT(price) OVER (
            PARTITION BY ticker_symbol
            RANGE INTERVAL '1' HOUR PRECEDING) AS count_price
    FROM "SOURCE_SQL_STREAM_001"
```

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-16 23:36:16.729	WMT
2017-02-16 23:36:16.729	DFG
2017-02-16 23:36:16.729	RFV
2017-02-16 23:36:16.729	TGH

使用说明

Amazon Kinesis Analytics 不支持FILTER子句COUNT函数或使用COUNT DISTINCT在聚合函数或分析函数中。有关聚合函数和分析函数的更多信息，请参阅聚合函数 (p. 69)和分析函数 (p. 97). 此功能偏离了 SQL:2008 标准。

当用作分析函数时，COUNT如果被评估的窗口不包含任何行，则返回零。有关更多信息，请参阅 分析函数 (p. 97)。COUNT 还会为 a 返回零PARTITION BY子句，窗口内与输入行匹配的分区不包含任何行。有关 PARTITION BY 的更多信息，请参阅 WINDOW 子句 (滑动窗口) (p. 57)。

COUNT忽略一组值或数值表达式中的空值。例如，以下各项返回值 3：

- COUNT(1, 2, 3) = 3
- COUNT(1,null, 2, null, 3, null) = 3

相关主题

- 窗口式查询
- 聚合函数 (p. 69)
- GROUP BY 子句 (p. 55)
- 分析函数 (p. 97)
- 入门练习
- WINDOW 子句 (滑动窗口) (p. 57)

COUNT_DISCT_ITEMS_TUMBLING 函数

返回滚动窗口中指定的应用程序内流列中不同项目的数量的计数。生成的计数是近似值；该函数使用 HyperLogLog 算法。

在使用 COUNT_DISTINCT_ITEMS_TUMBLING 时，请注意以下事项：

- 当窗口中的项目数小于或等于 10000 时，此函数将返回准确计数。
- 精确计算不同项目的数量可能效率低下且成本高昂。因此，此函数近似计数。例如，如果有 100,000 个不同的项目，则该算法可能返回 99,700 个。如果不考虑成本和效率，您可以自行编写 SELECT 语句以获取准确计数。

  以下示例演示了如何在五秒钟的翻滚窗口中获取每个股票代码的不同行的精确数量。SELECT 语句使用所有列（ROWTIME 除外）来确定唯一性。

```
CREATE OR REPLACE STREAM output_stream (ticker_symbol VARCHAR(4), unique_count BIGINT);


CREATE OR REPLACE PUMP stream_pump AS
INSERT INTO output_stream
SELECT STREAM TICKER_SYMBOL, COUNT(distinct_stream.price) AS unique_count
  FROM (
    SELECT STREAM DISTINCT rowtime as window_time,

      TICKER_SYMBOL,
      CHANGE,
      PRICE,
      STEP((SOURCE_SQL_STREAM_001.rowtime) BY INTERVAL '5' SECOND)

      FROM SOURCE_SQL_STREAM_001) as distinct_stream
  GROUP BY TICKER_SYMBOL,
      STEP((distinct_stream.window_time) BY INTERVAL '5' SECOND);
```

此函数在一个滚动窗口内运行。您可以将翻滚窗口的大小指定为参数。

语法

```
COUNT_DISTINCT_ITEMS_TUMBLING (

      in-application-streamPointer,
      'columnName',
      windowSize
   )
```

参数

以下各节介绍参数。

in-application-stream指针

使用此参数，您可以提供指向应用程序内部流的指针。您可以使用以下命令来设置指针CURSOR函数。例如，以下语句将指针设置为InputStream.

```
CURSOR(SELECT STREAM * FROM InputStream)
```

columnName

应用程序内流中您希望函数使用该列来计算不同值的列名。请注意有关Columname

- 必须出现在单引号 (') 中。例如，'column1'。

窗口大小

翻滚窗口的大小，以秒为单位。大小应至少为 1 秒且不应超过 1 小时 = 3600 秒。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是其中的一部分开始使用在里面Amazon Kinesis Analitics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Analitics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：近似列中的独特值的数量

以下示例演示如何使用COUNT_DISTINCT_ITEMS_TUMBLING函数用于近似不同数量的函数TICKER_SYMBOL应用程序内部流的当前滚动时间的值。有关滚动窗口的更多信息，请参阅滚动窗口。

```
CREATE OR REPLACE STREAM DESTINATION_SQL_STREAM (
    NUMBER_OF_DISTINCT_ITEMS BIGINT);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
   INSERT INTO "DESTINATION_SQL_STREAM"
```

```
    SELECT STREAM *
    FROM TABLE(COUNT_DISTINCT_ITEMS_TUMBLING(
        CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"),  -- pointer to the data
 stream
          'TICKER_SYMBOL',                                     -- name of column in
 single quotes
          60                                                   -- tumbling window size in
 seconds
    )
);
```

上一示例输出的流与以下内容类似：

Filter by column name

ROWTIME	NUMBER_OF_DISTINCT_ITEMS
2017-03-16 21:22:57.49	47
2017-03-16 21:23:57.49	47
2017-03-16 21:24:57.49	47
2017-03-16 21:25:57.49	47

EXP_AVG

```
EXP_AVG ( expression, <time-interval> )
```

EXP_AVG 返回在指定时间窗口内选择的值表达式的流的指数加权的平均值。EXP_AVG 基于 <time-interval> 的值将指定窗口分成若干个时间间隔。对于最近的时间间隔，指定表达式的值的加权最大，而对于较早的时间间隔，则加权指数级降低。

示例

此示例创建了 30 秒窗口内每个股票代码价格的指数加权平均值，这样最近 10 秒子窗口中的价格（该股票代码）的权重是中间 10 秒子窗口中价格权重的两倍，是中间 10 秒子窗口中价格权重的四倍最古老的 10 秒子窗口。

```
select stream t.rowtime, ticker, price,
exp_avg(price, INTERVAL '10' SECOND) over w as avgPrice
from t
window w as (partition by ticker range interval '30' second preceding);
```

在此示例中，10 秒是衰减函数的半衰期，也就是说，应用于平均价格的权重在此期间减少了两倍。换句话说，较旧的重量将是新版本的一半。在调用 EXP_AVG 时，它被指定为 time_interval，间隔 “10” 秒。

FIRST_VALUE

```
FIRST_VALUE( <value-expression>) <null treatment> OVER <window-specification>
```

FIRST_VALUE 从有资格聚合的第一行中返回 <value expression> 的评估。FIRST_VALUE 需要 OVER 子句，被视为分析函数 (p. 97). FIRST_VALUE 在下表中定义了一个空处理选项。

Null 治疗选项	效果
FIRST_VALUE(x) IGNORE NULLS OVER <window-specification>	返回 <window-specification> 中 x 的首个非 null 值
FIRST_VALUE(x) RESPECT NULLS OVER <window-specification>	返回第一个值，包括 <window-specification> 中 x 的 null 值
FIRST_VALUE(x) OVER <window-specification>	返回第一个值，包括 <window-specification> 中 x 的 null 值

LAST_VALUE

```
LAST_VALUE ( <value-expression> )  OVER <window-specification>
```

LAST_VALUE 从有资格聚合的最后一行中返回 <value expression> 的计算。

Null 治疗选项	效果
LAST_VALUE(x) IGNORE NULLS OVER <window-specification>	返回 <window-specification> 中 x 的最后一个非 null 值
LAST_VALUE(x) RESPECT NULLS OVER <window-specification>	返回最后一个值，包括 <window-specification> 中 x 的 null 值
LAST_VALUE(x) OVER <window-specification>	返回最后一个值，包括 <window-specification> 中 x 的 null 值

MAX

返回窗口查询中一组值的最大值。窗口查询是根据时间或行数定义的。有关窗口查询的信息，请参阅窗口式查询。

在使用 MAX 时，请注意以下事项：

- 如果你不使用OVER子句，MAX以聚合函数的形式计算。在这种情况下，聚合查询必须包含GROUP BY 子句 (p. 55)在基于以下条件的单调表达式上ROWTIME它将流分组为有限的行。否则，该组是无限流，查询将永远不会完成，也不会发出任何行。有关更多信息，请参阅 聚合函数 (p. 69)。
- 使用 GROUP BY 子句的窗口式查询处理滚动窗口中的行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您将OVER子句，MAX以分析函数的形式计算。有关更多信息，请参阅 分析函数 (p. 97)。
- 使用 OVER 子句的窗口式查询处理滑动窗口中的行。有关更多信息，请参阅滑动窗口

语法

滚动窗口式查询

```
```

```
MAX(number-expression) ... GROUP BY monotonic-expression | time-based-expression
```

滑动窗口式查询

```
MAX(number-expression) OVER window-specification
```

参数

number-expression

指定针对聚合中的每一行计算的值表达式。

OVER window-specification

将流中的记录除以时间范围间隔或行数分区。窗口规范定义流中记录的划分方式 (按时间范围间隔或行数)。

GROUP B单调表达式|time-based-expression

基于分组表达式的值为记录分组，从而针对在所有列中具有相同值的每组行返回一个摘要行。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是其中的一部分开始使用在里面Amazon Kinesis Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics App 和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Analytics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：使用 GROUP BY 子句返回最大值

在此示例中，聚合查询有一个GROUP BY子句ROWTIME它将流分组为有限的行。随后，从 MAX 子句返回的行计算 GROUP BY 函数。

使用 STEP（推荐）

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    max_price     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
```

```
        ticker_symbol,
        MAX(Price) AS max_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

使用 FLOOR

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    max_price     DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        ticker_symbol,
        MAX(Price) AS max_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE);
```

结果

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-17 00:34:00.0	ASD
2017-02-17 00:35:00.0	BNM
2017-02-17 00:35:00.0	PPL
2017-02-17 00:35:00.0	QXZ

示例 2：使用 OVER 子句返回最大值

在此示例中，OVER子句将流中的记录除以前 '1' 小时的时间范围间隔。随后，从 MAX 子句返回的行计算 OVER 函数。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    max_price     DOUBLE);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM ticker_symbol,
        MAX(price) OVER (
            PARTITION BY ticker_symbol
            RANGE INTERVAL '1' HOUR PRECEDING) AS max_price
    FROM "SOURCE_SQL_STREAM_001"
```

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-17 00:36:44.854	QAZ
2017-02-17 00:36:50.796	QXZ
2017-02-17 00:36:50.796	MJN
2017-02-17 00:36:50.796	WSB

使用说明

对于字符串值，MAX 通过排序序列中的最后一个字符串来确定。

如果将 MAX 用作分析函数，并且正在评估的窗口不包含任何行，则 MAX 将返回 null。有关更多信息，请参阅 分析函数 (p. 97)。

相关主题

- 窗口式查询
- 聚合函数 (p. 69)
- GROUP BY 子句 (p. 55)
- 分析函数 (p. 97)
- 入门练习
- WINDOW 子句 (滑动窗口) (p. 57)

MIN

返回窗口查询中一组值的最小值。窗口查询是根据时间或行数定义的。有关窗口式查询的信息，请参阅窗口式查询。

在使用 MIN 时，请注意以下事项：

- 如果你不使用OVER子句，MIN以聚合函数的形式计算。在这种情况下，聚合查询必须包含GROUP BY 子句 (p. 55)在基于以下条件的单调表达式上ROWTIME它将流分组为有限的行。否则，该组是无限流，查询将永远不会完成，也不会发出任何行。有关更多信息，请参阅 聚合函数 (p. 69)。
- 使用 GROUP BY 子句的窗口式查询处理滚动窗口中的行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您将OVER子句，MIN以分析函数的形式计算。有关更多信息，请参阅 分析函数 (p. 97)。
- 使用 OVER 子句的窗口式查询处理滑动窗口中的行。有关更多信息，请参阅滑动窗口

语法

滚动窗口式查询

```
MIN(number-expression) ... GROUP BY monotonic-expression | time-based-expression
```

滑动窗口式查询

```
MIN(number-expression) OVER window-specification
```

参数

number-expression

指定针对聚合中的每一行计算的值表达式。

OVER window-specification

将流中的记录除以时间范围间隔或行数分区。窗口规范定义流中记录的划分方式 (按时间范围间隔或行数)。

GROUP B单调表达式|time-based-expression

基于分组表达式的值为记录分组，从而针对在所有列中具有相同值的每组行返回一个摘要行。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是开始使用在里面Amazon Kinesis Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics App 和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Analytics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：使用 GROUP BY 子句，返回最小值

在此示例中，聚合查询有一个GROUP BY子句ROWTIME它将流分组为有限的行。随后，从 MIN 子句返回的行计算 GROUP BY 函数。

使用 STEP（推荐）

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    min_price     DOUBLE);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        ticker_symbol,
        MIN(Price) AS min_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

使用 FLOOR

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    min_price     DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        ticker_symbol,
        MIN(Price) AS min_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE);
```

结果

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-17 22:45:00.0	QXZ
2017-02-17 22:46:00.0	WMT
2017-02-17 22:46:00.0	QWE
2017-02-17 22:46:00.0	CRM

示例 2：使用 OVER 子句返回最小值

在此示例中，OVER子句将流中的记录除以前 '1' 小时的时间范围间隔。随后，从 MIN 子句返回的行计算 OVER 函数。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    min_price     DOUBLE);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM ticker_symbol,
        MIN(price) OVER (
            PARTITION BY ticker_symbol
            RANGE INTERVAL '1' HOUR PRECEDING) AS min_price
    FROM "SOURCE_SQL_STREAM_001"
```

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-17 22:49:51.147	NFS
2017-02-17 22:49:59.058	NFLX
2017-02-17 22:49:59.058	ASD
2017-02-17 22:49:59.058	DFG

使用说明

对于字符串值，MIN 通过排序序列中的最后一个字符串来确定。

如果将 MIN 用作分析函数，并且正在计算的窗口不包含任何行，则 MIN 将返回 null。有关更多信息，请参阅分析函数 (p. 97)。

相关主题

- 窗口式查询
- 聚合函数 (p. 69)
- GROUP BY 子句 (p. 55)
- 分析函数 (p. 97)
- 入门练习
- WINDOW 子句 (滑动窗口) (p. 57)

SUM

返回窗口查询中一组值的总和。窗口查询是根据时间或行数定义的。有关窗口式查询的信息，请参阅窗口式查询。

在使用 SUM 时，请注意以下事项：

- 如果你不使用OVER子句，SUM以聚合函数的形式计算。在这种情况下，聚合查询必须包含GROUP BY 子句 (p. 55)在基于以下条件的单调表达式上ROWTIME它将流分组为有限的行。否则，该组是无限流，查询将永远不会完成，也不会发出任何行。有关更多信息，请参阅 聚合函数 (p. 69)。
- 使用 GROUP BY 子句的窗口式查询处理滚动窗口中的行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您将OVER子句，SUM以分析函数的形式计算。有关更多信息，请参阅 分析函数 (p. 97)。
- 使用 OVER 子句的窗口式查询处理滑动窗口中的行。有关更多信息，请参阅滑动窗口

语法

滚动窗口式查询

```
SUM(number-expression) ... GROUP BY monotonic-expression | time-based-expression
```

滑动窗口式查询

```
SUM([DISTINCT | ALL] number-expression) OVER window-specification
```

参数

DISTINCT

仅对唯一值进行计数。

ALL

计算所有行。ALL 是默认值。

number-expression

指定针对聚合中的每一行计算的值表达式。

OVER window-specification

将流中的记录除以时间范围间隔或行数分区。窗口规范定义流中记录的划分方式 (按时间范围间隔或行数)。

GROP BY单调表达式|time-based-expression

基于分组表达式的值为记录分组，从而针对在所有列中具有相同值的每组行返回一个摘要行。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是开始练习在里面Amazon Kinesis Aletics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Aletics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：使用 GROUP BY 子句返回值的总和

在此示例中，聚合查询有一个GROUP BY子句ROWTIME它将流分组为有限的行。随后，从 SUM 子句返回的行计算 GROUP BY 函数。

使用 STEP（推荐）

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    sum_price     DOUBLE);
```

```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        ticker_symbol,
        SUM(price) AS sum_price
    FROM "SOURCE_SQL_STREAM_001"
     GROUP BY ticker_symbol, STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

使用 FLOOR

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    sum_price     DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        ticker_symbol,
        SUM(price) AS sum_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE);
```

结果

上一示例输出的流与以下内容类似。

ROWTIME	TICKER_SYMBOL
2017-02-18 00:28:00.0	KIN
2017-02-18 00:29:00.0	VVS
2017-02-18 00:29:00.0	HJK
2017-02-18 00:29:00.0	KFU

使用说明

Amazon Kinesis Analytics`SUM`应用于间隔类型。此功能偏离了 SQL:2008 标准。

`SUM`忽略一组值或数值表达式中的空值。例如，以下各项返回值 6：

- AVG(1, 2, 3) = 6
- SUM(1,null, 2, null, 3, null) = 6

相关主题

- 窗口式查询
- 聚合函数 (p. 69)
- GROUP BY 子句 (p. 55)
- 分析函数 (p. 97)
- 入门练习

- WINDOW 子句 (滑动窗口) (p. 57)

TOP_K_ITEMS_TUMBLING 函数

返回滚动窗口中指定的应用程序内流列中最常出现的值。这可用于在指定列中查找趋势（最受欢迎）值。

例如，入门练习使用演示流，提供持续的股价更新（ticker_symbol、价格、变动和其他列）。假设你想在每1分钟的暴跌窗口中找到三只最常交易的股票。您可以使用此功能来查找这些股票代码。

在使用 `TOP_K_ITEMS_TUMBLING` 时，请注意以下事项：

- 计算流媒体源上的每条传入记录效率不高，因此该函数会近似最常出现的值。例如，在搜索交易量最大的三只股票时，该函数可能会返回五只交易量最大的股票中的三只。

此函数在一个滚动窗口内运行。您可以将窗口大小指定为参数。

对于示例应用程序，请使用 step-by-step 说明，请参阅最常出现的值.

语法

```
TOP_K_ITEMS_TUMBLING (
      in-application-streamPointer,
      'columnName',
      K,
      windowSize,
   )
```

参数

以下各节介绍了这些参数。

in-application-stream指针

指向应用程序内部流的指针。您可以使用 CURSOR 函数设置指针。例如，以下语句将指针设置为 InputStream.

```
CURSOR(SELECT STREAM * FROM InputStream)
```

columnName

应用程序内流中要用来计算 TopK 值的列名。请注意有关Columnn的以下内容：

> Note
>
> 列名必须用单引号 (') 出现。例如，`'column1'`。

K

使用此参数，您可以指定要返回特定列中最常出现的值的数量。值 K 必须大于或等于 1 且不能超过 100000。

窗口大小

翻滚窗口的大小，以秒为单位。大小必须大于或等于 1 秒，且不得超过 3600 秒 (1 小时)。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是其中的一部分开始使用在里面Amazon Kinesis Analytics 开发人员指南. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建Analytics应用程序和配置示例股票代码输入流，请参阅开始使用在里面Amazon Kinesis Analytics 开发人员指南.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：返回最常出现的值

以下示例在入门教程中创建的示例流中检索最常出现的值。

```
CREATE OR REPLACE STREAM DESTINATION_SQL_STREAM (
  "TICKER_SYMBOL" VARCHAR(4),
  "MOST_FREQUENT_VALUES" BIGINT
);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM *
        FROM TABLE (TOP_K_ITEMS_TUMBLING(
            CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"),
            'TICKER_SYMBOL',         -- name of column in single quotes
            5,                       -- number of the most frequently occurring values
            60                       -- tumbling window size in seconds
            )
        );
```

上一示例输出的流与以下内容类似。

Filter by column name

ROWTIME	TICKER_SYMBOL	MOST_FREQUENT_
2017-04-12 17:14:45.305	QXZ	17
2017-04-12 17:15:45.305	QXZ	21
2017-04-12 17:15:45.305	AAPL	11
2017-04-12 17:15:45.305	DFT	8

分析函数

分析函数是返回根据一组有限行中（或大约）的数据计算得出的结果的函数，该行集由行标识SELECT 子句 (p. 37)或者在ORDER BY 子句 (p. 65).

SELECT 主题解释了排序依据条款，显示了排序依据图表以及窗口条款（和窗口规格表）。要查看在 Select 语句中使用排序依据子句的位置，请参阅本指南 SELECT 主题中的选择图表。

1. 分析函数必须指定一个窗口。由于对窗口规格有一些限制，并且为窗口聚合指定窗口和窗口联接指定窗口之间也存在一些区别，请参阅允许和禁止的窗口规范 (p. 61)说明。
2. 分析函数只能出现在 SELECT 子句的 <selection list> 部分或 ORDER BY 子句中。

本主题后面的表格中说明了其他差异。

使用分析函数执行查询通常被称为窗口聚合（如下所述），不同于聚合函数 (p. 69).

由于存在窗口规范，使用分析函数的查询生成结果的方式与聚合查询不同。对于输入集中的每一行，窗口规范标识了一组不同的行，分析函数将在这些行上运行。如果窗口规范还包含 PARTITION BY 子句，则在生成结果时，窗口中唯一要考虑的行是那些与输入行共享相同分区的行。

如果输入行在用作分析函数输入的列中包含 null，则分析函数会忽略该行，但 COUNT 除外，它会计算具有空值的行。如果窗口（或者 PARTITION BY，则为窗口内的分区）不包含任何行，则分析函数将返回 null。唯一的例外是 COUNT，它返回零。

聚合函数和分析函数之间的区别

函数类型	输出	使用的行数或窗口	注意
聚合函数 (p. 69)	每组输入行一个输出行。	所有输出列都是在同一个窗口或同一组行上计算的。	DISTINCT DISTINCT聚合函数 (p. 69). 不允许使用以下类型的语句： SELECT COUNT(DISTINCT x) ... FROM ... GROUP BY ...
分析函数	每个输入行对应一个输出行。	可以使用不同的窗口或分区来计算每个输出列。	COUNT DISTINCT 不能用作分析函数或窗口聚合。

相关主题

- 直播窗口聚合 (p. 73)
- SELECT (p. 35)
- SELECT 子句 (p. 37)

布尔函数

本节中的主题介绍了 Amazon Kinesis Data Analytics 流式处理 SQL 的布尔函数。

主题

- ANY (p. 98)

- EVERY (p. 98)

ANY

```
ANY ( <boolean_expression> )
```

如果在任何选定行中提供的布尔表达式为真，则 ANY 将返回 true。如果所提供的 boolean_expression 在所有选定行中均为真，则返回 false。

示例

如果交易流中任何股票的价格低于 1，则以下 SQL 代码段返回 “true”。如果数据流中的每个价格都大于或等于 1，则返回 “false”。

```
 SELECT STREAM ANY (price < 1) FROM trades
  GROUP BY (FLOOR trades.rowtime to hour)
```

EVERY

```
EVERY ( <boolean_expression> )
```

如果提供的 boolean_expression 在所有选定行中均为真，则 EVERY 返回真。如果在任何选定行中提供的布尔表达式为假，则返回 false。

示例

如果交易流中每个股票的价格都低于 1，则以下 SQL 代码段返回 “true”。如果任何价格等于或大于 1，则返回 “false”。

```
 SELECT STREAM EVERY (price < 1) FROM trades
  GROUP BY (FLOOR trades.rowtime to hour)
```

转换函数

本节中的主题描述了 Amazon Kinesis Data Analytics 直播 SQL 的转换函数。

对于在日期时间和时间戳值之间进行转换的函数，请参阅 日期时间转换函数 (p. 112)。

主题

- CAST (p. 98)

CAST

CAST 允许您将一个值表达式或数据类型转换为另一种值表达式或数据类型。

```
CAST ( <cast-operand> AS <cast-target> )
 <cast-operand> := <value-expression>
 <cast-target>  := <data-type>
```

有效转换

将 CAST 与下面第一列中列出的类型的源操作数一起使用，可以不受限制地创建第二列中列出的转换目标类型。不支持其他目标类型。

源操作数类型	目标操作数类型
任何数值类型 （DECIMAL、DECIMAL、DECIMAL、DUBLE）	VARCHAR、CHAR 或任何数值类型（参见注释 A）
VARCHAR，	以上所有内容，加上 DATE、TIME、TIMESTAMP、DAY-TIME 间隔、布尔值
DATE	日期、VARCHAR、CHAR、时间戳
TIME	时间、VARCHAR、CHAR、时间戳
TIMESTAMP	时间、VARCHAR、CHAR、时间戳、日期
DATIME 间隔	日间间 隔、BIGINT、DECIMAL、CHAR、VARCHAR
BOOLEAN	VARCHAR、CHAR、布尔值
二进制、DATIMAL	二进制、DATIMAL

示例

2.1 到 CHAR/VARCHAR 约会

```
+-------------+
|   EXPR$0    |
+-------------+
| 2008-08-23  |
+-------------+
1 row selected
```

（请注意，如果提供的输出规格不足，则不选择任何行：

```
values(cast(date'2008-08-23' as varchar(9)));
'EXPR$0'
No rows selected
```

（因为日期文字需要 10 个字符）

在下一个例子中，右边的日期用空白填充（由于 CHAR 数据类型的语义）：

```
+----------------------------+
|           EXPR$0           |
+----------------------------+
| 2008-08-23                 |
+----------------------------+
1 row selected
```

实数到整数

实数（数字或十进制）由转换结果四舍五入：

```
+----------+
| EXPR$0   |
+----------+
| -2       |
+----------+
1 row selected
```

字符串转换为时间戳

有以下两种方法可将字符串转换为时间戳。第一个使用 CAST，如下一个主题所示。其他用途字符转时间戳 (Sys) (p. 113).

使用 CAST 将字符串转换为时间戳

以下示例说明了这种转换方法：

```
'EXPR$0'
'2007-02-19 21:23:45'
1 row selected
```

如果输入字符串缺少六个字段（年、月、日、小时、分钟、秒）中的任何一个，或者使用了与上面显示的分隔符不同的分隔符，CAST 将不会返回值。（不允许使用小数秒。）

因此，如果输入字符串的格式不适合 CAST，则要将字符串转换为时间戳，必须使用 CHAR_TO_TIMESTAMP 方法。

使用 CHAR_TO_TIMESTAMP 将字符串转换为时间戳

当输入字符串的格式不适合 CAST 时，可以使用 CHAR_TO_TIMESTAMP 方法。它的另一个好处是，您可以指定要在后续处理中使用时间戳字符串的哪些部分，并创建仅包含这些部分的 TIMESTAMP 值。为此，您需要指定一个模板来标识您想要的部分，例如 “YYYY-mm”，仅使用年份和月份。

这些区域有： input-date-time string-to-be-converted 可以包含完整时间戳的全部或任何部分，即任何或全部标准元素的值（'yyyy-mm-dd hh: mm: ss'）。如果所有这些元素都存在于您的输入字符串中，并且'yyyy-mm-dd hh: mm: ss'是您提供的模板，则输入字符串元素将按该顺序解释为年、月、日、时、分和秒，例如 '2009-09-16 03:15:24 '。yyyy 不能是大写字母；hh 可以是大写字母，表示使用 24 小时时钟。有关许多有效说明符示例，请参阅查看本主题后面的表格和示例。有关所有有效说明符的信息，请参见类 SimpleDateFormat在甲骨文网站上。

CHAR_TO_TIMESTAMP 在函数调用中使用您指定的模板作为参数。该模板使 TIMESTAMP 结果仅使用 TIMESTAMP 的部分内容 input-date-time 您在模板中指定的值。然后，生成的 TIMESTAMP 中的这些字段将包含从你的 TIMESTAMP 中提取的相应数据 input-date-time 字符串；未在模板中指定的字段将使用默认值（见下文）。CHAR_TO_TIMESTAMP 使用的模板格式定义为类 SimpleDateFormat，该链接列出了所有说明符，其中一些带有示例。有关更多信息，请参阅 日期和时间模式 (p. 129)。

函数调用语法如下所示：

```
  CHAR_TO_TIMESTAMP('<format_string>','<input_date_time_string>')
```

其中，<format_ string> 是您为所需的 <date_time_string> 部分指定的模板，<input_date_time_string> 是将转换为 TIMESTAMP 结果的原始字符串。

每个字符串必须用单引号引起来，并且 <input_date_time_string> 的每个元素必须位于其在模板中的相应元素的范围内。否则，不返回任何结果。

示例 1

- 这些区域有： input-string-element 其位置与 MM 对应的必须是介于 1 到 12 之间的整数，因为其他任何值都不代表有效的月份。

- 这些区域有： input-string-element 其位置与 dd 对应的必须是介于 1 到 31 之间的整数，因为其他任何值都不代表有效的日期。
- 但是，如果 MM 为 2，则 dd 不能为 30 或 31，因为 2 月从来没有这样的日子。

但是，对于月或天，替换省略部分的默认起始值为 01。

例如，使用 '2009-09-16 03:15:24 '作为输入字符串，您可以通过指定来获得仅包含日期的时间戳，其他字段（如小时、分钟或秒）为零

```
CHAR_TO_TIMESTAMP('yyyy-MM-dd','2009-09-16 03:15:24').
```

结果将是时间戳 2009-09-16 00:00:00。

示例 2

- 如果调用在模板中保留了小时和分钟，而省略了月、日和秒，如以下调用所示 — CHAR_TO_TIMESTAMP ('yyyy-hh-mm', '2009-09-16 03:15:24') — 那么生成的时间戳将是 2009-01-01 03:15:00。

模板	输入字符串	输出时间戳	注意
`'yyyy-MM-dd hh:mm:ss'`	`'2009-09-16 03:15:24'`	`'2009-09-16 03:15:24'`	输入字符串必须使用“yyyy-mm-dd hh: mm: ss” 或其子集或重新排序的形式；使用诸如“2009 年 9 月 16 日星期三 03:15:24” 之类的输入字符串不起作用，这意味着不会产生任何输出。
`'yyyy-mm'`	`'2012-02-08 07:23:19'`	`'2012-01-01 00:02:00'`	上面的模板仅指定了第一年和第二分钟，因此输入字符串 (“02”) 中的第二个元素用作分钟。 默认值用于月和日 (“01”) 以及小时和秒 (“00”)。
`'yyyy-ss-mm'`	`'2012-02-08 07:23:19'`	`'2012-01-01 00:08:02'`	上面的模板仅按该顺序指定年、秒和分钟，因此输入字符串中的第二个元素（“02”）用作秒，第三个元素用作分钟（“08”）。默认值用于月和日（“01”）和小时（“00”）。
`'MMM dd, yyyy'`	`'March 7, 2010'`	`'2010-03-07 00:00:00'`	嗯以上模板中的MMM与“March” 匹配；模板的“逗号空格” 与输入字符串匹配。 — — 如果模板缺少逗号，则输入字符串也必须缺少逗号，否则没有输出；

模板	输入字符串	输出时间戳	注意
			— — 如果输入字符串缺少逗号，则模板也必须缺少逗号。
'MMM dd,'	'March 7, 2010'	'1970-03-07 00:00:00'	请注意，上面的模板不使用年份说明符，导致输出 TIMESTAMP 使用该纪元中最早的年份，即 1970 年。
'MMM dd,y'	'March 7, 2010'	'2010-03-07 00:00:00'	使用上面的模板，如果输入字符串是 “3月7日10日”，则输出时间戳将为 “0010-03-07 00:00:00”。
'M-d'	'2-8'	'1970-02-08 00:00:00'	如上所述，模板中缺少 yyyy 说明符，则使用这个时代（1970 年）的最早年份。 输入字符串 “2-8-2012” 会得到相同的结果；使用 “2012-2-8” 不会给出任何结果，因为 2012 不是有效的月份。
'MM-dd-yyyy'	'06-23-2012 10:11:12'	'2012-06-23 00:00:00'	如果模板和输入在相同位置使用破折号作为分隔符（如上所述）就可以了。由于模板省略了小时、分钟和秒，因此在输出 TIMESTAMP 中使用零。
'dd-MM-yy hh:mm:ss'	'23-06-11 10:11:12'	'2011-06-23 10:11:12'	您可以按任意顺序设置说明符，只要该顺序与您提供的输入字符串的含义相匹配，如上所述。以下下一个示例的模板和输入字符串与本示例具有相同的含义（以及相同的输出 TIMESTAMP），但它们指定了几天前的几个月和小时前的秒数。
'MM-dd-yy ss:hh:mm'	'06-23-11 12:10:11'	'2011-06-23 10:11:12'	在上面使用的模板中，月份和日说明符的顺序与上面的示例相反，秒的说明符在小时之前而不是分钟之后；但是由于输入字符串还把月份放在几天之前，秒放在小时之前，所以含义（和输出时间戳）是与上面的示例相同。

模板	输入字符串	输出时间戳	注意
'yy-dd-MM ss:hh:mm'	'06-23-11 12:10:11'	'2006-11-23 10:11:12'	上面使用的模板反向使用年份和月份说明符（与上面的示例相比），而输入字符串保持不变。在这种情况下，输出 TIMESTAMP 使用输入字符串的第一个元素作为年份，第二个元素作为日期，第三个元素作为月份。
'dd-MM-yy hh:mm'	'23-06-11 10:11:12'	'2011-06-23 10:11:00'	如上所述，在模板中省略秒数的情况下，输出 TIMESTAMP 使用 00 秒。任意数量的 y 说明符都会产生相同的结果；但是如果输入字符串无意中使用了 1 而不是 11 作为年份，如 '23-06-1 10:11:12'，则输出时间戳变为 '0001-06-23 10:11:00'。
'MM/dd/yy hh:mm:ss'	'12/19/11 10:11:12' '12/19/11 12:11:10'	'2011-12-19 10:11:12' '2011-12-19 00:11:10'	如果模板和输入在相同位置使用斜杠作为分隔符就可以了，如上所述；否则，不输出。 使用说明符 hh，12:11:10 和 00:11:10 的输入时间与早晨时间的含义相同。

模板	输入字符串	输出时间戳	注意

'MM/dd/yy HH:mm:ss'	'12/19/11 12:59:59' '12/19/11 21:08:07'	'2011-12-19 12:59:59' '2011-12-19 21:08:07'	使用此模板，输入字符串值 “2011-12-19 00:11:12” 或 “2011-12-19 12:11:12” 将失败，因为 “2011” 不是一个月，正如模板字符串 “mm/dd/YY HH: mm: ss” 所要求/预期的那样。
			
			但是，更改模板会提供有用的输出：
			
			`values(cast(CHAR_TO_TIMESTAMP('y/MM/dd HH:mm:ss', '2011/12/19 00:11:12') as varchar(19)));`
			`'EXPR$0'`
			`'2011-12-19 00:11:12'`
			`1 row selected`
			
			'12/19/11 00:11:12' 将因上述模板 ('y/MM/dd') 而失败，因为 19 不是有效月份；提供 '12/11/19 00:11:12' 将起作用。'2011-12-19 12:11:12' 作为输入时将失败，因为短划线与模板中的斜杠不符，而 '2011/12/19 12:11:12' 有效。
			
			请注意，对于中午 12 点之后的时间，即下午和晚上的时间，小时说明符必须是 HH 而不是 hh，并且输入字符串必须以 24 小时制时间指定下午或晚上，小时从 00 到 23 不等。
			
			— — 使用说明符 HH，12:11:10 和 00:11:10 的输入时间有不同的含义，第一个是下午的时间，第二个是早上的时间。
			
			— — 使用说明符 hh，从 12:00 到 11:59:59 的时间是早晨：

模板	输入字符串	输出时间戳	注意
			— — 给定说明符 hh: mm: ss，输出时间戳将在早上为输入字符串 '12:09:08' 和输入字符串 '00:09:08' 包含 '00:09:08'； — — 而 — — 给定说明符 HH: mm: ss，早上输入字符串 '00:09:08' 的输出时间戳将包含 '00:09:08' — — 下午输入字符串 '12:09:08' 的输出时间戳将包含 '12:09:08'。

更多示例

以下示例说明了在 CHAR_TO_TIMESTAMP 中使用各种模板，包括一些常见的误解。

```
 values (CHAR_TO_TIMESTAMP('yyyy-hh-mm','2009-09-16 03:15:24'));
'EXPR$0'
'2009-01-01 09:16:00'
1 row selected
```

请注意，上面输入字符串中的字段是按照模板中说明符给出的顺序使用的，如 dashes-as-delimiters 在模板和输入字符串中：先是年份，然后是小时，然后是分钟。由于模板中不存在月和日的说明符，因此它们在输入字符串中的值被忽略，输出 TIMESTAMP 中的两个值都替换了 01。模板将小时和分钟指定为第二个和第三个输入值，因此 09 变成了小时，16 变成了分钟。几秒钟内没有指定符，因此使用了 00。

年份说明符可以单独使用，也可以在与输入字符串匹配的分隔符后显示年末说明符，并使用 hours: minutes: seconds 说明符之一：

```
values (CHAR_TO_TIMESTAMP('yyyy','2009-09-16 03:15:24') );
'EXPR$0'
'2009-01-01 00:00:00'
1 row selected
values (CHAR_TO_TIMESTAMP('yyyy hh','2009-09-16 03:15:24') );
'EXPR$0'
No rows selected
```

上面的模板失败了，因为它有一个 space-as-delimiter 在 “hh” 之前，而不是输入字符串的日期说明中使用的短划线分隔符之前；

而下面的四个模板之所以起作用，是因为它们使用相同的分隔符将年份说明符与下一个说明符分开，这与输入字符串的日期规范中使用的分隔符相同（第一种情况为短划线，第二种为空格，第三种为斜杠，第四种为破折号）。

```
values (CHAR_TO_TIMESTAMP('yyyy-hh','2009-09-16 03:15:24') );
'EXPR$0'
'2009-01-01 09:00:00'
1 row selected
values (CHAR_TO_TIMESTAMP('yyyy hh','2009 09 16 03:15:24') );
'EXPR$0'
```

```
'2009-01-01 09:00:00'
1 row selected
values (CHAR_TO_TIMESTAMP('yyyy/hh','2009/09/16 03:15:24') );
'EXPR$0'
'2009-01-01 09:00:00'
1 row selected
values (CHAR_TO_TIMESTAMP('yyyy-mm','2009-09-16 03:15:24') );
'EXPR$0'
'2009-01-01 00:09:00'
1 row selected
```

但是，如果模板指定了月 (MM)，则除非还指定了天，否则无法指定小时、分钟或秒：

模板仅指定年份和月份，因此在生成的时间戳中省略了天/小时/分钟/秒：

```
values (CHAR_TO_TIMESTAMP('yyyy-MM','2009-09-16 03:15:24') );
'EXPR$0'
'2009-09-01 00:00:00'
1 row selected
```

接下来的两个模板失败，缺少 “天” 说明符：

```
values (CHAR_TO_TIMESTAMP('yyyy-MM hh','2009-09-16 03:15:24') );
'EXPR$0'
No rows selected
values (CHAR_TO_TIMESTAMP('yyyy-MM hh:','2009-09-16 03:15:24') );
'EXPR$0'
No rows selected
```

接下来的三个成功了，使用 “天” 说明符：

```
 values (CHAR_TO_TIMESTAMP('yyyy-MM-dd hh','2009-09-16 03:15:24') );
'EXPR$0'
'2009-09-16 03:00:00'
1 row selected
```

上面的模板 “yyyy-mm-dd hh” 仅指定了小时 (hh)，没有分钟或秒。由于 hh 是模板的第 4 个标记/元素，因此其值将取自输入字符串 “2009-09-16 03:15:24” 的第 4 个标记/元素；第 4 个元素是 03，然后用作小时的输出值。由于既未指定 mm 也未指定 ss，因此使用定义为 mm 和 ss 起点的默认值或初始值，即零。

```
values (CHAR_TO_TIMESTAMP('yyyy-MM-dd ss','2009-09-16 03:15:24') );
'EXPR$0'
'2009-09-16 00:00:03'
1 row selected
```

上面的模板 “yyyy-mm-dd ss” 指定输入字符串的第 4 个标记/元素将用作秒 (ss)。输入字符串 “2009-09-16 03:15:24” 的第 4 个元素是 03，它成为模板中指定的秒数输出值；并且由于模板中既没有指定 hh 也没有指定 mm，因此使用它们的默认值或初始值，即零。

```
values (CHAR_TO_TIMESTAMP('yyyy-MM-dd mm','2009-09-16 03:15:24') );
'EXPR$0'
'2009-09-16 00:03:00'
1 row selected
```

上面的模板 “yyyy-mm-dd mm” 指定将输入字符串的第 4 个标记/元素用作分钟 (mm)。输入字符串 “2009-09-16 03:15:24” 的第 4 个元素是 03，它成为模板中指定的分钟数输出值；并且由于模板中既没有指定 hh 也没有指定 ss，因此使用它们的默认值或初始值，即零。

更多失败，缺少 “天” 说明符：

```
values (CHAR_TO_TIMESTAMP('yyyy-MM- mm','2009-09-16 03:15:24') );
'EXPR$0'
No rows selected
values (CHAR_TO_TIMESTAMP('yyyy-MM   mm','2009-09-16 03:15:24') );
'EXPR$0'
No rows selected
values (CHAR_TO_TIMESTAMP('yyyy-MM   hh','2009-09-16 03:15:24') );
'EXPR$0'
No rows selected
```

关于分隔符和值

模板中的分隔符必须与输入字符串中的分隔符相匹配；输入字符串中的值对于它们所对应的模板说明符必须是可接受的。

按照一般惯例，冒号用于将小时与分钟分开，分与秒分开。同样，一般惯例是使用破折号或斜线将年与月分开，将月与日分开。任何parallel 用法似乎都有效，以下示例说明了这一点。

```
values (CHAR_TO_TIMESTAMP('MM/dd/yy hh:mm:ss','2009/09/16 03:15:24') );
'EXPR$0'
No rows selected
```

上面的示例失败了，因为 2009 不是一个月的可接受值，而月数是模板中的第一个说明符 (MM)。

```
values (CHAR_TO_TIMESTAMP('MM/dd/yy hh:mm:ss','09/16/11 03:15:24') );
'EXPR$0'
'2011-09-16 03:15:24'
1 row selected
```

上面的示例之所以成功，是因为分隔符是parallel 的（斜杠对斜杠，冒号对冒号），并且每个值对于相应的说明符都是可接受的。

```
values (CHAR_TO_TIMESTAMP('MM/dd/yy hh/mm/ss','09/16/11 03/15/24') );
'EXPR$0'
'2011-09-16 03:15:24'
1 row selected
```

上面的示例之所以成功，是因为分隔符是parallel 的（所有斜杠），并且每个值对于相应的说明符都是可接受的。

```
values (CHAR_TO_TIMESTAMP('MM/dd/yy hh-mm-ss','09/16/11 03-15-24') );
'EXPR$0'
'2011-09-16 03:15:24'
1 row selected
values (CHAR_TO_TIMESTAMP('yyyy|MM|dd hh|mm|ss','2009|09|16 03|15|24') );
'EXPR$0'
'2009-09-16 03:15:24'
1 row selected
values (CHAR_TO_TIMESTAMP('yyyy@MM@dd hh@mm@ss','2009@09@16 03@15@24') );
'EXPR$0'
'2009-09-16 03:15:24'
1 row selected
```

上面的示例之所以成功，是因为分隔符是parallel 的，每个说明符的值都是可接受的。

在以下示例中，请注意，提供的字符串中的遗漏会导致模板值 “yyyy” 产生合乎逻辑但意外或意想不到的结果。生成的 TIMESTAMP 值中以年份形式给出的值直接派生自所提供字符串中的第一个元素。

```
 VALUES(CHAR_TO_TIMESTAMP('yyyy','09-16 03:15'));
'EXPR$0'
'0009-01-01 00:00:00'
1 row selected
VALUES(CHAR_TO_TIMESTAMP('yyyy','16 03:15'));
'EXPR$0'
'0016-01-01 00:00:00'
1 row selected
```

时间戳转换为字符串

```
 values(cast( TIMESTAMP '2007-02-19 21:25:35' AS VARCHAR(25)));
'EXPR$0'
'2007-02-19 21:25:35'
1 row selected
```

请注意，CAST 需要 TimeStamp-Literal 才能真正具有 '的完整格式yyyy-mm-dd hh: mm: s'。如果该完整格式的任何部分缺失，则该文字将被视为非法而被拒绝，如下所示：

```
 values( TIMESTAMP '2007-02-19 21:25');
Error: Illegal TIMESTAMP literal '2007-02-19 21:25':
                                  not in format 'yyyy-MM-dd HH:mm:ss' (state=,code=0)
 values( TIMESTAMP '2007-02-19 21:25:00');
'EXPR$0'
'2007-02-19 21:25:00'
1 row selected
```

此外，如果提供的输出规格不足，则不选择任何行：

```
values(cast( TIMESTAMP '2007-02-19 21:25:35' AS VARCHAR(18)));
'EXPR$0'
No rows selected
(Because the timestamp literal requires 19 characters)
```

这些限制同样适用于转换为时间或日期类型。

字符串到时间

```
 values(cast(' 21:23:45.0' AS TIME));
'EXPR$0'
'21:23:45'
1 row selected
```

有关更多信息，请参阅 Note A。

DATE

```
 values(cast('2007-02-19' AS DATE));
'EXPR$0'
'2007-02-19'
1 row selected
```

备注 A

请注意，字符串的 CAST 要求用于转换为 TIME 或 DATE 的字符串操作数具有分别表示时间或日期所需的精确格式。

如下所示，在以下情况下，投射失败：

- 字符串操作数包含与目标类型无关的数据，或
- 间隔操作数（'日小时:分钟:seconds.milliseconds'）不包括必要的数据，或
- 指定的输出字段太小，无法保存转换结果。

```
 values(cast('2007-02-19 21:23:45.0' AS TIME));
'EXPR$0'
No rows selected
```

失败，因为它包含不允许作为时间使用的日期信息。

```
 values(cast('2007-02-19 21:23:45.0' AS DATE));
'EXPR$0'
No rows selected
```

失败，因为它包含不允许作为日期的时间信息。

```
 values(cast('2007-02-19 21' AS DATE));
'EXPR$0'
No rows selected
```

失败，因为它包含不允许作为日期的时间信息。

```
 values(cast('2009-02-28' AS DATE));
'EXPR$0'
'2009-02-28'
1 row selected
```

成功是因为它包含了日期字符串的正确表示形式。

```
 values(CAST (cast('2007-02-19 21:23:45.0' AS TIMESTAMP) AS DATE));
'EXPR$0'
'2007-02-19'
1 row selected
```

成功是因为它在将字符串转换为 DATE 之前正确地将字符串转换为 TIMESTAMP。

```
 values(cast('21:23' AS TIME));
'EXPR$0'
No rows selected
```

失败，因为它缺少时间所需的时间信息（秒）。

（允许指定小数秒，但不是必需的。）

```
 values(cast('21:23:34:11' AS TIME));
'EXPR$0'
No rows selected
```

失败，因为它包含了错误的小数秒表示。

```
 values(cast('21:23:34.11' AS TIME));
'EXPR$0'
```

```
'21:23:34'
1 row selected
```

之所以成功，是因为它包含了小数秒的正确表示。

```
 values(cast('21:23:34' AS TIME));
'EXPR$0'
'21:23:34'
1 row selected
```

此示例之所以成功，是因为它包含了秒数的正确表示形式，但没有分数秒。

精确数值的间隔

间隔的 CAST 要求间隔操作数中只有一个字段，例如 MINUTE、HOUR、SECOND。

如果间隔操作数有多个字段，例如 MINUTE TO SECOND，则转换失败，如下所示：

```
 values ( cast (INTERVAL '120' MINUTE(3) as decimal(4,2)));
+----------+
| EXPR$0   |
+----------+
+----------+
No rows selected

 values ( cast (INTERVAL '120' MINUTE(3) as decimal(4)));
+----------+
| EXPR$0   |
+----------+
| 120      |
+----------+
1 row selected

 values ( cast (INTERVAL '120' MINUTE(3) as decimal(3)));
+----------+
| EXPR$0   |
+----------+
| 120      |
+----------+
1 row selected

 values ( cast (INTERVAL '120' MINUTE(3) as decimal(2)));
+----------+
| EXPR$0   |
+----------+
+----------+
No rows selected

 values cast(interval '1.1' second(1,1) as decimal(2,1));
+----------+
| EXPR$0   |
+----------+
| 1.1      |
+----------+
1 row selected

 values cast(interval '1.1' second(1,1) as decimal(1,1));
+----------+
| EXPR$0   |
+----------+
+----------+
No rows selected
```

对于年份，不允许将十进制分数作为输入和输出。

```
values cast(interval '1.1' year (1,1) as decimal(2,1));
Error: org.eigenbase.sql.parser.SqlParseException: Encountered "," at line 1, column 35.
Was expecting:
    ")" ... (state=,code=0)
values cast(interval '1.1' year (1) as decimal(2,1));
Error: From line 1, column 13 to line 1, column 35:
              Illegal interval literal format '1.1' for INTERVAL YEAR(1) (state=,code=0)
values cast(interval '1.' year (1) as decimal(2,1));
Error: From line 1, column 13 to line 1, column 34:
              Illegal interval literal format '1.' for INTERVAL YEAR(1) (state=,code=0)
values cast(interval '1' year (1) as decimal(2,1));
+----------+
| EXPR$0   |
+----------+
| 1.0      |
+----------+
1 row selected
```

有关其他示例，请参阅 SQL 运算符：更多示例。

限制

Amazon Kinesis Data Analytics 不支持直接将数值转换为间隔值。这与 SQL: 2008 标准背道而驰。将数值转换为间隔的推荐方法是将数值与特定的间隔值相乘。例如，要将整数 time_in_millis 转换为日间间隔：

```
time_in_millis * INTERVAL '0 00:00:00.001' DAY TO SECOND
```

例如：

```
 values cast( 5000 * (INTERVAL '0 00:00:00.001' DAY TO SECOND) as varchar(11));
'EXPR$0'
'5000'
1 row selected
```

日期和时间函数

以下内置函数与日期和时间有关。

主题

其中，SQL 扩展 CURRENT_ROW_TIMESTAMP 对于流式上下文最有用，因为它为您提供了有关流数据出现时间的信息，而不仅仅是查询的运行时间。这是流式查询和传统 RDMS 查询之间的关键区别：流式查询保持 “开放” 状态，生成更多数据，因此运行查询时的时间戳无法提供良好的信息。

LOCALTIMESTAMP、LOCALTIME、CURRENT_DATE 和 CURRENT_TIMESTAMP 都会生成在首次执行查询时设置为值的结果。只有 CURRENT_ROW_TIMESTAMP 会为每行生成一行具有唯一时间戳（日期和时间）的行。

以 LOCALTIMESTAMP（或 CURRENT_TIMESTAMP 或 CURRENT_TIME）作为其中一列运行的查询在首次运行查询时放入所有输出行中。如果该列改为包含 CURRENT_ROW_TIMESTAMP，则每个输出行都会获得一个新计算的 TIME 值，表示该行的输出时间。

要从日期时间值返回组成某个组成部分（例如，月份中的某天），请使用 EXTRACT (p. 133)

时区

Amazon Kinesis Data Analytics 在 UTC 因此，所有时间函数都以 UTC 返回时间。

日期时间转换函数

您可以使用带图案的字母指定日期和时间格式。日期和时间模式字符串使用从 “A” 到 “Z” 以及从 “a” 到 “z” 的无引号字母，每个字母代表一个格式化元素。

有关更多信息，请参阅 。类 SimpleDateFormatOracle 网站上。

> Note
>
> 如果您包含其他字符，则它们将在格式化期间合并到输出字符串中，或者在解析期间与输入字符串进行比较。

定义了下表中的模式字母（保留从 'A' 到 'Z' 以及从 'a' 到 'z' 的所有其他字符）。

信	日期或时间组件	演示文稿	示例
y	年份	年份	yyyy; yy 2018; 18
Y	星期年	年份	YYYY; YY 2009; 09
M	一年中的月	月份	嗯; 嗯; MM 七月; 七月; 07
w	一年中的周	数字	ww; 27
W	monthonth	数字	W 2
D	一年中的日期	数字	DDD 321
d	month 中的日期	数字	dd 10
F	month 中的日期	数字	F 2
E	周中的日名	文本	星期二；星期二
u	星期数（1 = 星期一，...，7 = 星期日）	数字	1
a	上午/下午标记	文本	下午
H	一天中的小时 (0-23)	数字	0
k	一天中的小时 (1-24)	数字	24

信	日期或时间组件	演示文稿	示例
K	上午/下午的时间 (0-11)	数字	0
h	上午/下午小时 (1-12)	数字	12
m	一小时中的分钟	数字	30
s	分钟内秒	数字	55
S	毫秒	数字	978
z	时区	一般时区	太平洋标准时间； 太平洋标准时间； GMT-08:00
Z	时区	RFC 822 时区	-0800
X	时区	ISO 8601 时区	-08; -0800;-08:00

您可以按照 YYYY 的行重复图案字母来确定确切的表示形式。

文本

如果重复模式字母的数量为 4 个或更多，则使用完整格式；否则，使用简短或缩写形式（如果有）。解析时，两种形式都可接受，与模式字母的数量无关。

数字

格式化时，模式字母的数量是最小位数，较短的数字用零填充到这个数值。在解析时，除非需要分隔两个相邻的字段，否则会忽略模式字母的数量。

年份

如果格式化程序的日历是公历，则应用以下规则。

- 格式化时，如果模式字母的数量为 2，则将年份截断为 2 位数；否则将解释为数字。
- 为了进行解析，如果模式字母的数量大于 2，则无论数字多少，都按字面解释年份。因此，使用 “mm/dd/yyyy” 模式，“01/11/12” 解析到公元 1 月 11 日 12 日

要使用缩写的年份模式（“y” 或 “yy”）进行解析， SimpleDateFormat 必须解释相对于某个世纪的缩写年份。它通过将日期调整为在 80 年之前 80 年之内和之后 20 年来做到这一点 SimpleDateFormat 实例已创建。例如，使用 “mm/dd/YY” 模式和 SimpleDateFormat 创建于 2018 年 1 月 1 日的实例，字符串 “01/11/12” 将被解释为 2012 年 1 月 11 日，而字符串 “05/04/64” 将被解释为 1964 年 5 月 4 日。在解析过程中，只有由 character.isDigit (char) 定义的正好由两位数字组成的字符串才会被解析为默认世纪。任何其他数字字符串，例如一位数的字符串、三位或更多位数的字符串或不是全数字的两位数字符串（例如 “-1”），都是按字面解释的。因此，“01/02/3” 或 “01/02/003” 的解析模式与公元 1 月 2 日、3 日相同。同样，“01/02/-3” 被解析为公元前 4 年 1 月 2 日。

否则，将应用日历系统特定的表单。对于格式化和解析，如果模式字母的数量为 4 或更多，则使用日历特定的长格式。否则，将使用日历特定的简短或缩写形式。

字符转时间戳 (Sys)

Char to Timestamp 函数是最常用的系统函数之一，因为它允许你从任何格式正确的输入字符串中创建时间戳。使用此函数，您可以指定要在后续处理中使用时间戳字符串的哪些部分，并创建仅包含这些部分的 TIMESTAMP 值。为此，您需要指定一个标识所需时间戳部分的模板。例如，要仅使用年和月，应指定 'yyyy-mm '。

输入的日期时间字符串可以包含完整时间戳的任何部分（'yyyy-mm-dd hh: mm: ss'）。如果所有这些元素都存在于您的输入字符串中，并且'yyyy-mm-dd hh: mm: ss'是您提供的模板，则输入字符串元素将按该顺序解释为年、月、日、时、分和秒，例如 '2009-09-16 03:15:24 '。yyyy 不能是大写字母；hh 可以是大写字母，表示使用 24 小时时钟。

有关所有有效说明符的信息，请参见类 SimpleDateFormat在甲骨文网站上。

CHAR_TO_TIMESTAMP 在函数调用中使用您指定的模板作为参数。该模板使得 TIMESTAMP 结果仅使用 TIMESTAMP 的部分内容 input-date-time 您在模板中指定的值。生成的 TIMESTAMP 中的那些字段包含从你的 TIMESTAMP 中提取的相应数据 input-date-time 字符串。模板中未指定的字段将使用默认值（见下文）。CHAR_TO_TIMESTAMP 使用的模板格式由类 SimpleDateFormat在甲骨文网站上。有关更多信息，请参阅 日期和时间模式 (p. 129)。

函数调用语法如下所示：

```
CHAR_TO_TIMESTAMP('<format_string>','<input_date_time_string>')
```

其中，<format_ string> 是您为所需的 <date_time_string> 部分指定的模板，<input_date_time_string> 是将转换为 TIMESTAMP 结果的原始字符串。

请注意，每个字符串必须用单引号引起来，并且 <input_date_time_string> 的每个元素必须位于其在模板中的相应元素的范围内，否则不会返回任何结果。

例如，的 input-string-element 其位置与 MM 对应的必须是介于 1 到 12 之间的整数，因为其他任何值都不代表有效的月份。同样， input-string-element 其位置与 dd 对应的必须是介于 1 到 31 之间的整数，因为其他任何值都不代表有效的日期。（但是，如果 MM 为 2，则 dd 不能为 30 或 31，因为 2 月从来没有这样的日子。）

对于小时、分钟或秒，默认起始值为零，因此，当模板中省略这些说明符时，将替换为零。对于月或天，替换省略部分的默认起始值为 01。

例如，使用 “2009-09-16 03:15:24” 作为输入字符串，您可以获得仅包含日期的时间戳，其他字段（如小时、分钟或秒）为零。

```
 CHAR_TO_TIMESTAMP('yyyy-MM-dd','2009-09-16 03:15:24').
```

结果将是 TIMESTAMP 2009-09-16 00:00:00。

如果呼叫在模板中保留了小时和分钟，而省略了月、天和秒，如以下调用所示。

```
--- --- CHAR_TO_TIMESTAMP('yyyy-hh-mm','2009-09-16 03:15:24')
```

然后，生成的时间戳将是 2009-01-01 03:15:00。

用于创建特定输出时间戳的模板字符串 (p. 116)显示了用于创建指定输出 TimeStamps 的模板和输入字符串的更多说明性示例。

> **Note**
>
> 输入字符串必须使用 “yyyy-mm-dd hh: mm: ss” 的形式或其子集或重新排序。因此，使用诸如 “2009 年 9 月 16 日星期三 03:15:24” 之类的输入字符串将不起作用，这意味着不会产生任何输出。

关于分隔符和值

模板中的分隔符必须与输入字符串中的分隔符相匹配，并且输入字符串中的值对于它们所对应的模板说明符必须是可接受的。

按照一般惯例，冒号用于将小时与分钟分开，分与秒分开。同样，一般惯例是使用破折号或斜线将年与月分开，将月与日分开。

例如，以下模板的值与输入字符串正确对齐。

```
values (CHAR_TO_TIMESTAMP('MM/dd/yy hh:mm:ss','09/16/11 03:15:24') );
'EXPR$0'
'2011-09-16 03:15:24'
1 row selected
```

如果输入字符串中的值对应的模板说明符不可接受，则结果将失败，如以下示例所示。

```
values (CHAR_TO_TIMESTAMP('MM/dd/yy hh:mm:ss','2009/09/16 03:15:24') );
'EXPR$0'
No rows selected
```

此示例不返回任何行，因为 2009 不是一个月的可接受值，这是模板中的第一个说明符 (MM)。

提供的字符串中的遗漏会导致模板值 “yyyy” 产生合乎逻辑但意外或意想不到的结果。以下示例均返回错误的年份，但该年份直接派生自所提供字符串中的第一个元素。

```
 VALUES(CHAR_TO_TIMESTAMP('yyyy','09-16 03:15'));
'EXPR$0'
'0009-01-01 00:00:00'
1 row selected
VALUES(CHAR_TO_TIMESTAMP('yyyy','16 03:15'));
'EXPR$0'
'0016-01-01 00:00:00'
1 row selected
```

使用模板创建时间戳的示例

模板的顺序必须与输入字符串相匹配。这意味着你不能在 “yyyy” 之后指定 “hh”，也不能期望该方法自动找到时间。例如，以下模板先指定年份，然后指定小时，然后指定分钟，并返回错误结果。

```
 values (CHAR_TO_TIMESTAMP('yyyy-hh-mm','2009-09-16 03:15:24'));
'EXPR$0'
'2009-01-01 09:16:00'
1 row selected
```

由于模板中不存在月和日的说明符，因此它们在输入字符串中的值被忽略，输出 TIMESTAMP 中的两个值都替换了 01。模板将小时和分钟指定为第二个和第三个输入值，因此 09 变成了小时，16 变成了分钟。几秒钟内没有指定符，因此使用了 00。

年份说明符可以单独使用，也可以在与输入字符串匹配的分隔符之后显示年末说明符，其中一个 hours: minutes: seconds 说明符。

```
values (CHAR_TO_TIMESTAMP('yyyy','2009-09-16 03:15:24') );
'EXPR$0'
'2009-01-01 00:00:00'
1 row selected
```

相比之下，下面的模板失败是因为它有一个 space-as-delimiter 在 “hh” 之前，而不是输入字符串的日期说明中使用的短划线分隔符之前。

```
  values (CHAR_TO_TIMESTAMP('yyyy hh','2009-09-16 03:15:24') );
  'EXPR$0'
  No rows selected
```

以下四个模板之所以起作用，是因为它们使用相同的分隔符将年份说明符与下一个说明符分开，这与输入字符串的日期指定中使用的分隔符相同（第一种情况为短划线，第二种为空格，第三种为斜杠，第四种为破折号）。

```
values (CHAR_TO_TIMESTAMP('yyyy-hh','2009-09-16 03:15:24') );
'EXPR$0'
'2009-01-01 09:00:00'
1 row selected
values (CHAR_TO_TIMESTAMP('yyyy hh','2009 09 16 03:15:24') );
'EXPR$0'
'2009-01-01 09:00:00'
1 row selected
values (CHAR_TO_TIMESTAMP('yyyy/hh','2009/09/16 03:15:24') );
'EXPR$0'
'2009-01-01 09:00:00'
1 row selected
values (CHAR_TO_TIMESTAMP('yyyy-mm','2009-09-16 03:15:24') );
'EXPR$0'
'2009-01-01 00:09:00'
1 row selected
```

但是，如果模板指定了月 (MM)，则除非还指定了天，否则无法指定小时、分钟或秒。

用于创建特定输出时间戳的模板字符串

模板	输入字符串	输出TIMESTA	注意
`'yyyy-MM-dd hh:mm:ss'`	`'2009-09-16 03:15:24'`	`'2009-09-16 03:15:24'`	
`'yyyy-mm'`	`'2011-02-08 07:23:19'`	`'2011-01-01 00:02:00'`	上面的模板仅指定了第一年和第二分钟，因此输入字符串 (“02”) 中的第二个元素用作分钟。默认值用于月和日 (“01”) 以及小时和秒 (“00”)。
`'MMM dd, yyyy'`	`'March 7, 2010'`	`'2010-03-07 00:00:00'`	嗯，上面模板中的 “March” 匹配；模板的 “逗号空格” 与输入字符串匹配。 如果模板缺少逗号，则输入字符串也必须缺少逗号，否则没有输出； 如果输入字符串缺少逗号，则模板也必须缺少逗号。
`'MMM dd,'`	`'March 7, 2010'`	`'1970-03-07 00:00:00'`	请注意，上面的模板不使用年份说明符，导致输出 TIMESTAMP 使用该纪元中最早的年份，即 1970 年。
`'MMM dd,y'`	`'March 7, 2010'`	`'2010-03-07 00:00:00'`	使用上面的模板，如果输入字符串是 “3月7日10

模板	输入字符串	输出TIMESTA	注意
			日”，则输出时间戳将为“0010-03-07 00:00:00”。
'M-d'	'2-8'	'1970-02-08 00:00:00'	如上所述，模板中缺少 yyyy 说明符，则使用这个时代（1970 年）的最早年份。 输入字符串 '2—8−2011' 会得到相同的结果；使用 '2011—2−8' 不会给出任何结果，因为 2011 年不是有效的月份。
'MM-dd-yyyy'	'06-23-2011 10:11:12'	'2011-06-23 00:00:00'	如果模板和输入在相同位置使用破折号作为分隔符（如上所述）就可以了。由于模板省略了小时、分钟和秒，因此在输出 TIMESTAMP 中使用零。
'dd-MM-yy hh:mm:ss'	'23-06-11 10:11:12'	'2011-06-23 10:11:12'	您可以按任意顺序使用说明符，只要该顺序与您提供的输入字符串的含义相匹配。以下下一个示例的模板和输入字符串与本示例具有相同的含义（以及相同的输出 TIMESTAMP），但它们指定了几天前的几个月和小时前的秒数。
'MM-dd-yy ss:hh:mm'	'06-23-11 12:10:11'	'2011-06-23 10:11:12'	在上面使用的模板中，月份和日说明符的顺序与上面的示例相反，秒的说明符在小时之前而不是分钟之后；但是由于输入字符串还将月份放在天前，秒放在小时之前，因此（和输出时间戳）是与上面的示例相同。
'yy-dd-MM ss:hh:mm'	'06-23-11 12:10:11'	'2006-11-23 10:11:12'	上面使用的模板反向使用年份和月份说明符（与上面的示例相比），而输入字符串保持不变。在这种情况下，输出 TIMESTAMP 使用输入字符串的第一个元素作为年份，第二个元素作为日期，第三个元素作为月份。

模板	输入字符串	输出TIMESTA	注意
`'dd-MM-yy hh:mm'`	`'23-06-11 10:11:12'`	`'2011-06-23 10:11:00'`	如上所述，在模板中省略秒数的情况下，输出 TIMESTAMP 使用 00 秒。任意数量的 y 说明符都会产生相同的结果；但是如果输入字符串无意中使用了 1 而不是 11 作为年份，如 '23-06-1 10:11:12'，则输出时间戳变为 '0001-06-23 10:11:00'。

`'MM/dd/yy hh:mm:ss'`	`'12/19/11 10:11:12'`
	
	`'12/19/11 12:11:12'`
		`'2011-12-19 10:11:12'`
		
		`'12/19/11 00:11:12'`
			如果模板和输入在相同位置使用斜杠作为分隔符就可以了，如上所述。使用说明符 hh，12:11:10 和 00:11:10 的输入时间与早晨时间的含义相同。

模板	输入字符串	输出TIMESTA	注意

'MM/dd/yy HH:mm:ss'	'12/19/11 12:59:59'
	
	'12/19/11 21:08:07'
	
	'2011-12-19 00:11:12'
	
	'2011-12-19 12:11:12'
		'2011-12-19 12:59:59'
		
		'2011-12-19 21:08:07'
			输入字符串值'2011-12-19 00:11:12'要么'2011-12-19 12:11:12'这个模板会失败，因为'2011'不是一个月，正如 template-string 要求/预期的那样'MM/dd/yy HH:mm:ss'.
			
			但是，更改模板会提供有用的输出：
			
			`values(cast(CHAR_TO_TIMESTAMP('y/MM/dd HH:mm:ss', '2011/12/19 00:11:12') as varchar(19)));`
			`'EXPR$0'`
			`'2011-12-19 00:11:12'`
			
			已选择 1 行
			
			'12/19/11 00:11:12'上面的模板会失败 ('y/MM/dd')，因为 19 不是有效的月份；提供'12/11/19 00:11:12'作品。
			
			'2011-12-19 12:11:12'作为输入会失败，因为破折号与模板中的斜杠不匹配，'2011/12/19 12:11:12'作品。
			
			请注意，对于中午 12 点之后的时间（即下午和晚上的时间），小时说明符必须是 HH 而不是 hh，并且输入字符串必须以 24 小时制时间指定下午或晚上的时间，小时从 00 到 23 不等。
			
			使用说明符 HH，12:11:10 和 00:11:10 的输入时间具有不同的含义，第一个是下午的时间，第二个是早上的时间。

模板	输入字符串	输出TIMESTA	注意
			使用说明符 hh，从 12:00 到 11:59:59 的时间是早晨： • 给定说明符 hh: mm: ss，输出时间戳将包括`'00:09:08'`早上输入字符串 '12:09:08' 和输入字符串`'00:09:08'`；而， • 给定说明符 hh: mm: ss，输入字符串的输出时间戳`'00:09:08'`早上将包括`'00:09:08'` 以及输入字符串的输出时间戳`'12:09:08'`下午将包括`'12:09:08'`.

CHAR_TO_DATE

根据指定的格式字符串将字符串转换为日期。

```
CHAR_TO_DATE(format,dateString);
```

CHAR_TO_TIME

根据指定的格式字符串将字符串转换为日期

```
CHAR_TO_TIME(format,dateString);
```

DATE_TO_CHAR

DATE_TO_CHAR 将日期转换为字符串。

```
DATE_TO_CHAR(format,d);
```

其中 d 是将转换为字符串的日期。

TIME_TO_CHAR

使用格式化字符串格式化时间。以字符串形式返回格式化的时间或时间的一部分。

```
TIME_TO_CHAR(format,time);
```

TIMESTAMP_TO_CHAR

使用格式化字符串将时间戳格式化为字符。以字符串形式返回时间戳。

```
TIMESTAMP_TO_CHAR(format,ts);
```

其中 ts 是时间戳。

Note

如果输入为 `null`，则输出将是字符串“`null`”。

TO_TIMESTAMP

将 Unix 时间戳转换为“YYYY-MM-DD HH:MM:SS”格式的 SQL 时间戳。

语法

```
TO_TIMESTAMP(unixEpoch)
```

参数

unixEpoch

采用自“1970-01-01 00:00:00”UTC 以来的毫秒数格式的 Unix 时间戳，以 BIGINT 形式表示。

示例

示例数据集

以下示例基于示例股票数据集，后者是《Amazon Kinesis Analytics 开发人员指南》 中的入门练习的一部分。

Note

已修改示例数据集以包含 Unix 时间戳值 (CHANGE_TIME)。

要运行每个示例，您需要一个 Amazon Kinesis Analytics 应用程序，该应用程序具有示例股票行情的输入流。要了解如何创建 Analytics 应用程序和配置示例股票代码输入流，请参阅《Amazon Kinesis Analytics 开发人员指南》 中的入门练习。

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
change_time     BIGINT,    --The UNIX timestamp value
price           REAL)
```

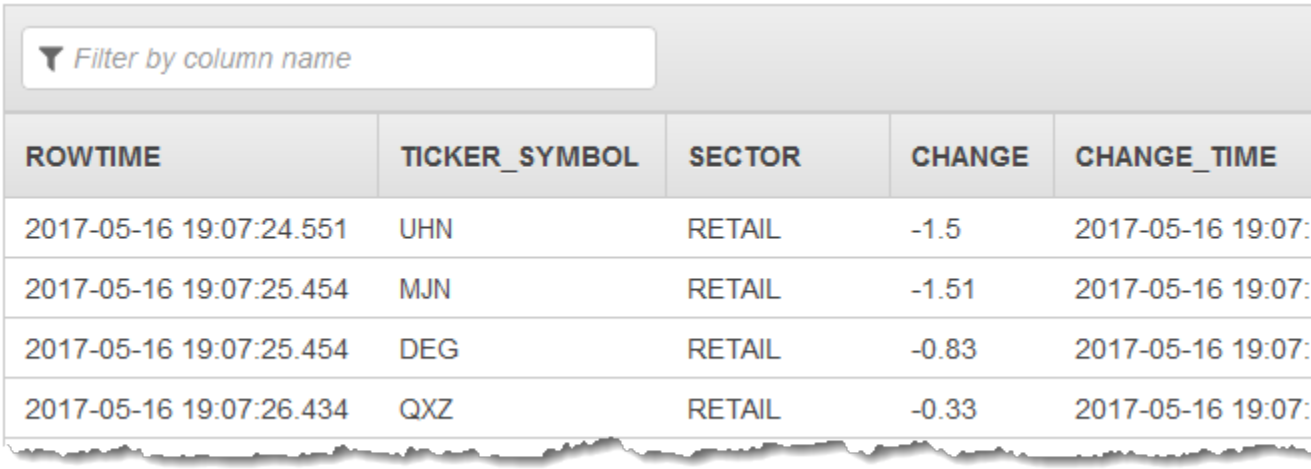
示例 1：将 Unix 时间戳转换为 SQL 时间戳

在此示例中，源流中的 `change_time` 值将转换为应用程序内流中的 SQL TIMESTAMP 值。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    sector VARCHAR(64),
    change REAL,
```

```
    change_time TIMESTAMP,
    price REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM   TICKER_SYMBOL,
                SECTOR,
                CHANGE,
                TO_TIMESTAMP(CHANGE_TIME),
                PRICE

FROM "SOURCE_SQL_STREAM_001"
```

上一示例输出的流与以下内容类似。



注意

TO_TIMESTAMP 不是 SQL: 2008 标准的一部分。它是Amazon Kinesis Data Analytics 直播 SQL 扩展。

UNIX_TIMESTAMP

将 SQL 时间戳转换为 Unix 时间戳，后者使用自“1970-01-01 00:00:00”UTC 以来的毫秒数表示，并采用 BIGINT 格式。

语法

```
UNIX_TIMESTAMP(timeStampExpr)
```

参数

timeStampExpr

一个 SQL TIMESTAMP 值。

示例

示例数据集

以下示例基于示例股票数据集，后者是《Amazon Kinesis Analytics 开发人员指南》 中的入门练习的一部分。

Note

已修改示例数据集以包含时间戳值 (CHANGE_TIME)。

要运行每个示例，您需要一个 Amazon Kinesis Analytics 应用程序，该应用程序具有示例股票行情的输入流。要了解如何创建 Analytics 应用程序和配置示例股票代码输入流，请参阅《Amazon Kinesis Analytics 开发人员指南》 中的入门练习。

具有以下架构的示例股票数据集。

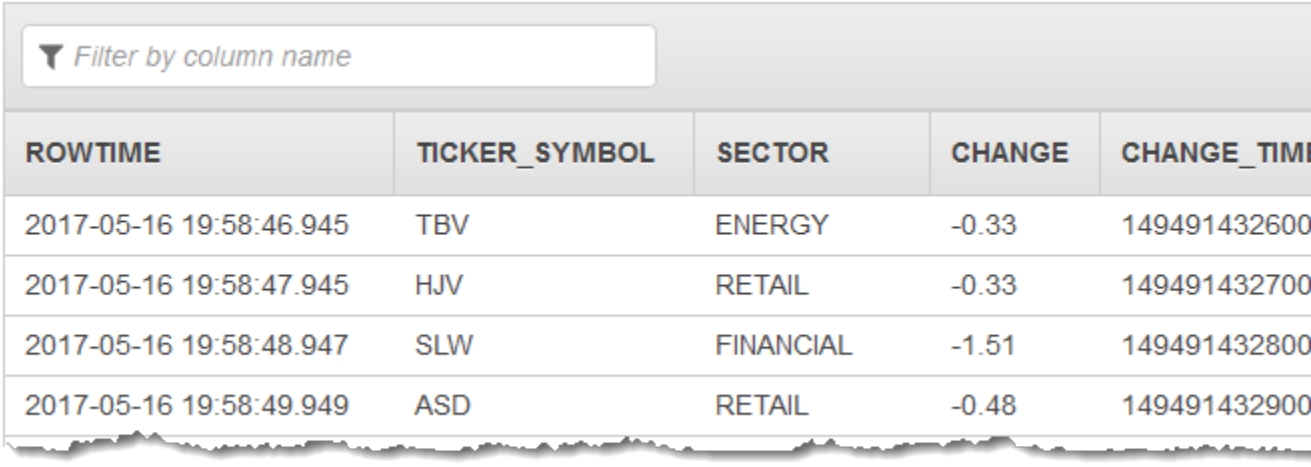
```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
change_time     TIMESTAMP,    --The timestamp value to convert
price           REAL)
```

示例 1：将时间戳转换为 UNIX 时间戳

在此示例中，源流中的 `change_time` 值将转换为应用程序内流中的 TIMESTAMP 值。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
        ticker_symbol VARCHAR(4),
        SECTOR VARCHAR(16),
        CHANGE REAL,
        CHANGE_TIME BIGINT,
        PRICE REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM   TICKER_SYMBOL,
                SECTOR,
                CHANGE,
                UNIX_TIMESTAMP(CHANGE_TIME),
                PRICE
FROM "SOURCE_SQL_STREAM_001"
```

上一示例输出的流与以下内容类似。



注意

UNIX_TIMESTAMP 不是 SQL: 2008 标准的一部分。它是Amazon Kinesis Data Analytics 直播 SQL 扩展。

日期、时间戳和间隔运算符

算术运算符 +、-、+、-、+、-、+、-、+、-、+、-、-、、

操作符	描述	注意
+	加	间隔 + 间隔 = 间隔 间隔 + 日期时间 = 日期时间 日期时间 + 间隔 = 日期时间
-	减	间隔-间隔 = 间隔 datetime-间隔 = 日期时间 (<datetime> - <datetime>) 日期、时间戳和间隔运算符 (p. 124) <interval qualifier> = 时间间隔
*	乘	间隔 * 数字 = 间隔 数字 * 间隔 = 间隔
/	除	间隔/数字 = 间隔

示例

示例	运算	结果
1	间隔 '1' 天 + 间隔 '3' 天	间隔 '4' 天
2	间隔 '1' 天 + 间隔 '3 4' 天到小时	间隔 '+4 04' 日对小时
3	间隔 '1' 天-间隔 '3 4' 天到小时	间隔 '-2 04' 天到小时
4	间隔 '1' 年 + 间隔 '3-4' 年与月	间隔 '+4-04' 年与月
5	2 * 间隔 '3 4' 天到小时	间隔 '6 8' 天到小时
6	间隔 '3 4' 天到小时/2	间隔 '1 14' 天到小时

在示例 3 中，'3 4 DAY 表示 3 天 4 小时，因此该行中的结果表示 24 小时减去 76 小时，结果为负 52 小时，即 2 天零 4 小时的负值。

示例 4 使用 TO MONTH 而不是 TO HOUR，因此指定为 “3-4” 的间隔表示 3 年零 4 个月，即 40 个月。

在示例 6 中，“/2” 适用于间隔 '3 4'，即 76 小时，其中一半为 38，或 1 天零 14 小时。

间隔运算的更多示例

Streaming SQL 还支持减去两个日期时间，给出一个间隔。您可以指定想要的结果间隔类型，如下所示：

```
(<datetime> - <datetime>) <interval qualifier>
```

以下示例显示了在 Amazon Kinesis Data Analytics 应用程序中可能有用的操作。

Example 1 — 时差（以分钟到最接近的秒为单位或以秒为单位）

```
 values cast ((time  '12:03:34' - time '11:57:23') minute to second as varchar(8));
+---------+
 EXPR$0
+---------+
 +6:11
+---------+
1 row selected
.............. 6 minutes, 11 seconds
or
 values cast ((time  '12:03:34' - time '11:57:23') second as varchar(8));
+---------+
 EXPR$0
+---------+
 +371
+---------+
1 row selected
```

Example 2 — 时差（仅限分钟）

```
values cast ((time  '12:03:34' - time '11:57:23') minute as varchar(8));
+---------+
 EXPR$0
+---------+
 +6
+---------+
1 row selected
.............. 6 minutes; seconds ignored.
values cast ((time  '12:03:23' - time '11:57:23') minute as varchar(8));
+---------+
 EXPR$0
+---------+
 +6
+---------+
1 row selected
.............. 6 minutes
```

Example 3 — 时间与时间戳的差异（以天数到最接近的秒数表示）无效

```
values cast ((time '12:03:34'-timestamp '2004-04-29 11:57:23') day to second as
 varchar(8));
Error: From line 1, column 14 to line 1, column 79: Parameters must be of the same type
```

Example 4 — 时间戳差异（以天数到最接近的秒数表示）

```
values cast ((timestamp  '2004-05-01 12:03:34' - timestamp '2004-04-29 11:57:23') day to
                second as varchar(8));
+-----------+
  EXPR$0
+-----------+
 +2 00:06
+-----------+
1 row selected
.............. 2 days, 6 minutes
.............. Although "second" was specified above, the varchar(8) happens to allow
only room enough to show only the minutes, not the seconds.
The example below expands to varchar(11), showing the full result:
values cast ((timestamp  '2004-05-01 12:03:34' - timestamp '2004-04-29 11:57:23') day to
                second as varchar(11));
```

```
+--------------+
    EXPR$0
+--------------+
 +2 00:06:11
+--------------+
1 row selected
.............. 2 days, 6 minutes, 11 seconds
```

Example 5 — 时间戳差异（以天数到最接近的秒数表示）

```
values cast ((timestamp  '2004-05-01 1:03:34' - timestamp '2004-04-29 11:57:23') day to
                second as varchar(11));
+--------------+
    EXPR$0
+--------------+
 +1 13:06:11
+--------------+
1 row selected
.............. 1 day, 13 hours, 6 minutes, 11 seconds
values cast ((timestamp  '2004-05-01 13:03:34' - timestamp '2004-04-29 11:57:23') day to
                second as varchar(11));
+--------------+
    EXPR$0
+--------------+
 +2 01:06:11
+--------------+
1 row selected
.............. 2 days, 1 hour, 6 minutes, 11 seconds
```

Example 6 — 时间戳差异（以天为单位）

```
values cast ((timestamp  '2004-05-01 12:03:34' - timestamp '2004-04-29 11:57:23') day
                as varchar(8));
+---------+
 EXPR$0
+---------+
 +2
+---------+
1 row selected
.............. 2 days
```

Example 7 — 时差（以天为单位）

```
values cast ((date '2004-12-02 ' - date '2003-12-01 ') day  as varchar(8));
Error: Illegal DATE literal '2004-12-02 ': not in format 'yyyy-MM-dd'
.............. Both date literals end with a space;  disallowed.
values cast ((date '2004-12-02' - date '2003-12-01 ') day  as varchar(8));
Error: Illegal DATE literal '2003-12-01 ': not in format 'yyyy-MM-dd'
.............. Second date literal still ends with a space;  disallowed.
values cast ((date '2004-12-02' - date '2003-12-01') day  as varchar(8));
+---------+
 EXPR$0
+---------+
 +367
+---------+
1 row selected
.............. 367 days
```

Example 8 — 不支持（简单的日期差异）

如果您没有指定 “天” 作为预期单位，如下所示，则不支持减法。

```
    values cast ((date '2004-12-02' - date '2003-12-01') as varchar(8));
    Error: From line 1, column 15 to line 1, column 51:
           Cannot apply '-' to arguments of type '<DATE> - <DATE>'.
    Supported form(s): '<NUMERIC> - <NUMERIC>'
                       '<DATETIME_INTERVAL> - <DATETIME_INTERVAL>'
                       '<DATETIME> - <DATETIME_INTERVAL>'
```

为什么在转换示例中使用 “as varchar” ？

<expression>在上面的示例中使用 “values cast (AS varchar (N))” 语法的原因是，尽管上面使用的 SQLLine 客户端（运行 Amazon Kinesis Data Analytics）确实会返回间隔，但 JDBC 不支持返回该结果来显示该结果。因此，使用 “值” 语法来查看/显示它。

如果你关闭Amazon Kinesis Data Analytics（用! kill command）或者如果你在运行 SQLLine 之前没有启动它，那么你可以从 Amazon Kinesis Data Analytics 主目录的 bin 子目录中运行 SQLLineEngine（而不是 SQLLineClient），它可以在没有 Amazon Kinesis Data Analytics 应用程序或 JDBC 的情况下显示你的结果：

指定间隔的规则

日间间隔文字是一个表示单个间隔值的字符串：例如 “10” 秒。请注意，它分为两部分：值（必须始终使用单引号）和限定符（此处为 SECONDS），后者给出值的单位。

该预选赛采用以下形式：

```
DAY  HOUR  MINUTE  SECOND [TO HOUR  MINUTE  SECOND]
```

> **Note**
>
> YEAR TO MONTH 间隔需要用短划线分隔值，而 DAY TO HOUR 间隔使用空格来分隔值，如该主题的第 2、3、5 和 6 个示例所示。

此外，前导术语必须比可选的尾部术语更重要，因此这意味着您只能指定：

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

理解这些最简单的方法可能是将 X 转换为 “X 到最接近的 Y”。因此，DAY TO HOUR 是 “天到最接近的小时”。

当 DAY、HOUR 或 MINUTE 是前导术语时，您可以指定精度，例如 DAY (3) TO HOUR，表示值中关联字段可以包含的位数。最大精度为 10，默认值为 2。您不能在后面的项中为小时或分钟指定精度，它们的精度始终为 2。因此，举例来说，小时 (3) 到分钟是合法的，小时到分钟 (3) 不合法。

SECOND 也可以采用精度，但其指定方式会有所不同，具体取决于它是前导字段还是尾随字段。

- 如果 SECOND 是前导字段，则可以指定小数点前后的数字。例如，SECOND (3,3) 允许您最多指定 999.999 秒。默认为 (2,3)，这实际上与 SQL: 2008 规范有所偏差（应该是 (2,6)，但我们只有毫秒精度）。

- 如果 SECOND 是尾随字段，则只能为小数秒指定精度，即下面显示在秒小数点之后的部分。例如，秒 (3) 表示毫秒。默认值为小数点后的 3 位数，但如上所述，这与 6 的标准有所偏差。

至于值，它采用以下一般形式：

```
  [+-]'[+-]DD HH:MM:SS.SSS'
```

其中 DD 是表示日的数字，HH 小时、MM 分钟，SS.SSS 是秒（如果明确指定了精度，请相应调整位数）。

并非所有值都必须包含所有字段，您可以从正面或背面修剪，但不能从中间修剪。所以你可以把它设为 “DD HH” 或 “MM: SS.SSS”，但不能把它设为 “DD MM”。

但是，无论如何编写，该值都必须与限定符相匹配，如下所示：

```
INTERVAL '25 3' DAY to HOUR ------> legal
INTERVAL '3:45:04.0' DAY TO HOUR --> illegal
```

正如 SQL 规范中所述，如果未明确指定精度，则默示为 2。因此：

- 间隔 '120' 分钟是非法的时间间隔。所需间隔的合法形式为间隔 “120” 分钟 (2)

  和

- 间隔 “120” 秒不合法。所需间隔的合法形式为间隔 '120' 秒 (3)。

```
   values INTERVAL '120' MINUTE(2);
   Error: From line 1, column 8 to line 1, column 31:
                       Interval field value 120 exceeds precision of MINUTE(2) field
   values INTERVAL '120' MINUTE(3);
   Conversion not supported
```

此外，如果 HOUR、MINUTE 或 SECOND 不是前导字段，则它们必须处于以下范围内（摘自 SQL: 2008 基础规范主题 4.6.3 中的表 6），如下所示：

```
  HOUR: 0-23
  MINUTE: 0-59
  SECOND: 0-59.999
```

年月间隔相似，唯一的不同是限定词如下所示：

```
  YEAR
  MONTH
  YEAR TO MONTH
```

可以像指定日和小时一样指定精度，最大值 10 和默认值 2 是相同的。

年月的值格式为：'YY-MM'。如果 MONTH 是尾随字段，则它必须在 0-11 的范围内。

```
<interval qualifier> := <start field> TO <end field>  <single datetime field>

<start field> := <non-second primary datetime field> [ <left paren> <interval leading field
 precision> <right paren> ]

<end field> := <non-second primary datetime field>  SECOND [ <left paren> <interval
 fractional seconds precision> <right paren> ]
```

```
<single datetime field> := <non-second primary datetime field> [ <left paren> <interval
 leading field precision> <right paren> ]
   SECOND [ <left paren> <interval leading field precision>
            [ <comma> <interval fractional seconds precision> ] <right paren> ]
<primary datetime field> := <non-second primary datetime field>       SECOND
<non-second primary datetime field> := YEAR  MONTH  DAY  HOUR      MINUTE
<interval fractional seconds precision> := <unsigned integer>
<interval leading field precision> := <unsigned integer>
```

日期和时间模式

日期和时间格式由日期和时间模式字符串指定。在这些模式字符串中，从 A 到 Z 以及从 a 到 z 的未加引号的字母表示数据或时间值的组成部分。如果将字母或文本字符串括在一对单引号内，则该字母或文本不会被解释，而是按原样使用，模式字符串中的所有其他字符也是如此。在打印期间，该字母或文本将按原样复制到输出字符串；在解析期间，它们将与输入字符串匹配。"''" 表示一个单引号。

为指定的日期或时间组件定义了以下模式字母。从 “A” 到 “Z” 以及从 “a” 到 “z” 的所有其他字符均被保留。有关图案字母的字母顺序信息，请参见按字母顺序排列的日期和时间模式字母 (p. 131).

日期或时间组件	模式字母	以文字或数字形式呈现	示例
时代指示符	G	文本 (p. 130)	广告
年份	y	年份	1996；96
一年中的月	M	月份	七月；七月；07
一年中的月	w	数字	27
星期几	W	数字	2
一年中的日期	D	数字	189
month	d	数字	10
星期几	F	数字	2
星期几	E	文本 (p. 130)	eee=tu；eee=tue；eeee=Tuesday
上午/下午标记	a	文本 (p. 130)	下午
一天中的小时 (0year)	H	数字	0
一天中的小时 (1-24)	k	数字	24
上午/下午的时间 (0-11)	K	数字	0
minute、	h	数字	12
一小时中的分钟	m	数字	30
minute	s	数字	55
毫秒	S	数字	978
时区	z	常规	太平洋标准时间；太平洋标准时间；GMT-08:00

日期或时间组件	模式字母	以文字或数字形式呈现	示例
时区	Z	RFC	-M

图案字母通常是重复的，因为它们的数字决定了确切的显示方式：

文本

格式化时，如果模式字母的数量为 4 或更多，则使用完整表单；否则，使用简短或缩写形式（如果可用）。解析时，两种形式都可接受，与模式字母的数量无关。

数字

格式化时，模式字母的数量是最小位数，较短的数字用零填充到这个数值。在解析时，除非需要分隔两个相邻的字段，否则会忽略模式字母的数量。

年份

如果时区有名称，则将其解释为文本。对于表示 GMT 偏移值的时区，使用以下语法：

```
GMTOffsetTimeZone:
GMT Sign Hours : Minutes
Sign: one of
+ -
Hours:
Digit
Digit Digit
Minutes:
Digit Digit
Digit: one of
0 1 2 3 4 5 6 7 8 9
```

小时数必须介于 0 到 23 之间，分钟必须介于 00 和 59 之间。该格式与区域无关，数字必须取自 Unicode 标准的基本拉丁语块。

为了进行解析，还接受 RFC 822 时区。

RC 822 时区

格式化时使用 RFC 822 4 位数时区格式：

```
RFC822TimeZone:
Sign TwoDigitHours Minutes
TwoDigitHours:
Digit Digit
```

TwoDigitHours 必须介于 00 和 23 之间。其他定义与一般时区相同。

为了进行解析，也接受一般时区。

SimpleDateFormat 还支持 “本地化日期和时间模式” 字符串。在这些字符串中，上述模式字母可以替换为其他依赖于区域设置的模式字母。 SimpleDateFormat 不处理除模式字母以外的文本的本地化；这取决于班级的客户。

示例

以下示例显示了在美国本地环境中如何解释日期和时间模式。给定的日期和时间是美国太平洋时区的当地时间 2001-07-04 12:08:56。

日期和时间模式	结果
“yyyy.mm.dd G 'at 'hh: mm: ss z”	公元 2001.07.04 太平洋夏令时 12:08:56
“哎呀，嗯 d，“yy”	01 年 7 月 4 日星期三
“嗯:mm a”	12:08 PM
“嗯'o'clock'a，zzzz”	太平洋夏令时间下午 12 点
“K: mm a, z”	太平洋夏令时下午 0:08
“yyyy.mmmmmm.dd GGG hh: mm aaa”	02001.July.04 AD 下午 12:08
“哎呀，嗯 yyyy HH: mm: ss Z”	2001 年 7 月 4 日星期三 12:08:56 -0700
“yymmddhhmssz”	010704120856-0700
“yyyyyyyyyyyyyyyyyyyyyyyyyy-M	2001-07-04T 12:08:56 .235-0700

按字母顺序排列的日期和时间模式字母

为了便于参考，下面按字母顺序显示了上面以日期或时间组件顺序显示的相同图案字母。

模式字母	日期或时间组件	以文字或数字形式呈现	示例
a	上午/下午标记	文本	下午
D	一年中的日期	数字	189
d	month	数字	10
E	星期几	文本	eee=tu；eee=tue；eeee=Tuesday
F	星期几	数字	2
G	时代指示符	文本	广告
H	一天中的小时 (0year)	数字	0
h	minute、	数字	12
k	一天中的小时 (1-24)	数字	24
K	上午/下午的时间 (0-11)	数字	0
M	一年中的月	月份	七月；七月；07
m	一小时中的分钟	数字	30
s	minute	数字	55
S	毫秒	数字	978
w	一年中的月	数字	27
W	星期几	数字	2
y	年份	年份	1996；96

模式字母	日期或时间组件	以文字或数字形式呈现	示例
z	时区	常规	太平洋标准时间；太平洋标准时间；GMT-08:00
Z	时区	RFC	-M

CURRENT_DATE

当查询执行时，以 YYY-MM-DD 形式返回当前 Amazon Kinesis Data Analytics 系统日期。

有关更多信息，请参阅 CURRENT_TIME (p. 132)、CURRENT_TIMESTAMP (p. 133)、LOCALTIMESTAMP (p. 135)、LOCALTIME (p. 134) 和当前_ROW_TIMESTAMP (p. 132)。

示例

```
+---------------+
| CURRENT_DATE  |
+---------------+
| 2008-08-27    |
+---------------+
```

当前_ROW_TIMESTAMP

CURRENT_ROW_TIMESTAMP 是 SQL: 2008 规范的Amazon Kinesis Data Analytics 扩展。此函数返回 Amazon Kinesis Data Analytics 应用程序运行环境定义的当前时间戳。CURRENT_ROW_TIMESTAMP 始终以 UTC 形式返回，而不是本地时区。

CURRENT_ROW_TIMESTAMP 类似于LOCALTIMESTAMP (p. 135)，但会为流中的每一行返回一个新的时间戳。

以 LOCALTIMESTAMP（或 CURRENT_TIMESTAMP 或 CURRENT_TIME）作为其中一列运行的查询在首次运行查询时放入所有输出行中。

如果该列改为包含 CURRENT_ROW_TIMESTAMP，则每个输出行都会获得一个新计算的 TIME 值，表示该行的输出时间。

> Note
>
> SQL: 2008 规范中没有定义 CURRENT_ROW_TIMESTAMP；它是Amazon Kinesis Data Analytics 扩展。

有关更多信息，请参阅 CURRENT_TIME (p. 132)、CURRENT_DATE (p. 132)、CURRENT_TIMESTAMP (p. 133)、LOCALTIMESTAMP (p. 135) 和 当前_ROW_TIMESTAMP (p. 132)。

CURRENT_TIME

返回查询执行时的当前 Amazon Kinesis Data Analytics 系统时间。时间采用 UTC，而不是本地时区。

有关更多信息，请参阅 CURRENT_TIMESTAMP (p. 133)、LOCALTIMESTAMP (p. 135)、LOCALTIME (p. 134)、当前_ROW_TIMESTAMP (p. 132) 和CURRENT_DATE (p. 132)。

示例

```
+---------------+
| CURRENT_TIME  |
+---------------+
| 20:52:05      |
```

CURRENT_TIMESTAMP

以日期时间值的形式返回当前数据库系统时间戳（在运行 Amazon Kinesis Data Analytics 的环境中定义）。

有关更多信息，请参阅
CURRENT_TIME (p. 132)、CURRENT_DATE (p. 132)、LOCALTIME (p. 134)、LOCALTIMESTAMP (p. 135)
和当前_ROW_TIMESTAMP (p. 132)。

示例

```
+--------------------+
| CURRENT_TIMESTAMP  |
+--------------------+
| 20:52:05           |
+--------------------+
```

EXTRACT

```
EXTRACT(YEAR|MONTH|DAY|HOUR|MINUTE|SECOND FROM <datetime expression>|<interval expression>)
```

EXTRACT 函数从日期、时间、时间戳或间隔表达式中提取一个字段。为除 “秒” 之外的所有字段返回 BIGINT。对于 SECOND，它返回 DECIMAL (5,3) 并包含毫秒。

语法

示例

函数	结果
`EXTRACT(DAY FROM INTERVAL '2 3:4:5.678' DAY TO SECOND)`	2
`EXTRACT(HOUR FROM INTERVAL '2 3:4:5.678' DAY TO SECOND)`	3
`EXTRACT(MINUTE FROM INTERVAL '2 3:4:5.678' DAY TO SECOND)`	4
`EXTRACT(SECOND FROM INTERVAL '2 3:4:5.678' DAY TO SECOND)`	5.678

函数	结果
`EXTRACT(MINUTE FROM CURRENT_ROW_TIMESTAMP) where CURRENT_ROW_TIMESTAMP is 2016-09-23 04:29:26.234`	29
`EXTRACT (HOUR FROM CURRENT_ROW_TIMESTAMP)` 其中 CURRENT_ROW_TIMESTAMP 是 2016-09-23 04:29:26 .234	4

函数：使用

EXTRACT 可用于调节数据，如以下函数所示，该函数在以下情况下返回 30 分钟下限当前_ROW_TIMESTAMP (p. 132)是 p_time 的输入。

```
CREATE or replace FUNCTION FLOOR30MIN( p_time TIMESTAMP )
RETURNS  TIMESTAMP
CONTAINS SQL
RETURNS NULL ON NULL INPUT
RETURN  floor(p_time to HOUR) + (( EXTRACT (  MINUTE FROM p_time  ) / 30)* INTERVAL '30'
 MINUTE ) ;
```

您可以使用如下代码实现函数：

```
SELECT stream FLOOR30MIN( CURRENT_ROW_TIMESTAMP ) as ROWTIME , * from "MyStream" ) over
 (range current row ) as r
```

Note

上面的代码假设你之前已经创建了一个名为” 的流MyStream。”

LOCALTIME

返回按照运行 Amazon Kinesis Data Analytics 的环境定义执行查询的当前时间。本地时间始终以 UTC (GMT) 返回，而不是本地时区。

有关更多信息，请参阅 CURRENT_TIME (p. 132)、CURRENT_DATE (p. 132)、CURRENT_TIMESTAMP (p. 133)、LOCALTIMESTAMP (p. 135 和当前_ROW_TIMESTAMP (p. 132)。

示例

```
 VALUES localtime;
+-------------+
| LOCALTIME   |
+-------------+
| 01:11:15    |
+-------------+
1 row selected (1.558 seconds)
```

限制

Amazon Kinesis Data Analytics 不支持 <time precision>SQL: 2008 中指定的可选参数。这与 SQL: 2008 标准背道而驰。

LOCALTIMESTAMP

返回正在运行的 Amazon Kinesis Data Analytics 应用程序上的环境定义的当前时间戳。时间始终以 UTC (GMT) 返回，而不是本地时区。

有关更多信息，请参阅 CURRENT_TIME (p. 132)、CURRENT_DATE (p. 132)、CURRENT_TIMESTAMP (p. 133)、LOCALTIME (p. 134) 和当前_ROW_TIMESTAMP (p. 132)。

示例

```
values localtimestamp;
+--------------------------+
|      LOCALTIMESTAMP      |
+--------------------------+
| 2008-08-27 01:13:42.206  |
+--------------------------+
1 row selected (1.133 seconds)
```

限制

Amazon Kinesis Data Analytics 不支持 <timestamp precision>SQL: 2008 中指定的可选参数。这与 SQL: 2008 标准背道而驰。

TSDIFF

如果任何参数为 null，则返回 NULL。

否则，返回两个时间戳之间的差值（以毫秒为单位）。

语法

```
TSDIFF(startTime, endTime)
```

参数

startTime

采用自“1970-01-01 00:00:00”UTC 以来的毫秒数格式的 Unix 时间戳，以 BIGINT 形式表示。

endTime

采用自“1970-01-01 00:00:00”UTC 以来的毫秒数格式的 Unix 时间戳，以 BIGINT 形式表示。

Null 函数

本节中的主题描述了 Amazon Kinesis Data Analytics 直播 SQL 的空函数。

主题

- COALESCE (p. 136)
- NULLIF (p. 136)

COALESCE

```
COALESCE (
      <value-expression>
      {,<value-expression>}... )
```

COALESCE 函数获取一个表达式列表（所有表达式的类型必须相同），并返回列表中的第一个非空参数。如果所有表达式都为空，则 COALESCE 返回空值。

示例

表达式	结果
COALESCE（“椅子”）	椅子
COALESCE（'椅子'、null、'沙发'）	椅子
COALESCE（空、null、'sofa'）	沙发
合并（空值、2、5）	2

NULLIF

```
NULLIF ( <value-expression>, <value-expression> )
```

如果两个输入参数相等，则返回 null，否则返回第一个值。两个参数必须是可比较的类型，否则会引发异常。

示例

函数	结果
NULLIF (4,2)	4
NULLIF (4,4)	<null>
NULLIF（'amy'，'fred'）	AMY
NULLIF（'amy'，cast（null 作为 varchar (3)））	AMY
NULLIF（cast（null 作为 varchar (3)），'fred'）	<null>

数字函数

本节中的主题描述了 Amazon Kinesis Data Analytics 直播 SQL 的数值函数。

主题

ABS

返回输入参数的绝对值。返回值null如果输入参数为空。

```
ABS ( <numeric-expression>  <interval-expression>
      )
```

示例

函数	结果
ABS (2.0)	2.0
ABS (-1.0)	1.0
腹肌 (0)	0
ABS (-3 * 3)	9
ABS (间隔 '-3 4:20 '天到分钟)	间隔 '3 4:20 '天到分钟

如果您使用cast as VARCHAR在 SQLLine 中显示输出，该值返回为+3 04:20.

```
values(cast(ABS(INTERVAL '-3 4:20' DAY TO MINUTE) AS VARCHAR(8)));
 +------------+
   EXPR$0
 +------------+
  +3 04:20
 +------------+
 1 row selected
```

天花板/天花板

```
CEIL | CEILING ( <number-expression> )
CEIL | CEILING ( <datetime-expression> TO <time-unit> )
CEIL | CEILING ( <number-expression> )
CEIL | CEILING ( <datetime-expression> TO <[[time-unit> )
```

当使用数字参数调用时，CICOLING 返回等于或大于输入参数的最小整数。

当与日期、时间或时间戳表达式一起调用时，CEILING 将返回大于或等于输入的最小值，具体取决于 <time unit> 指定的精度。

如果任何输入参数为空，则返回空值。

示例

函数	结果
CEIL (2.0)	2
CEIL (-1.0)	–1
CEIL (5.2)	6
天花板 (-3.3)	-3
上限 (-3 * 3.1)	-9
上限（时间戳 '2004-09-30 13:48:23 '到小时）	时间戳 '2004-09-30 14:00:00 '
上限（时间戳 '2004-09-30 13:48:23 '到分钟）	时间戳 '2004-09-30 13:49:00 '
上限（时间戳 “2004-09-30 13:48:23” 至当天）	时间戳 '2004-10-01 00:00:00 .0'
上限（时间戳 “2004-09-30 13:48:23” 至年）	时间戳 '2005-01-01 00:00:00 .0'

注意

- CEIL 和 CEILING 是 SQL: 2008 标准提供的此函数的同义词。
- CEIL (TO <datetime value expression><time unit>) 是Amazon Kinesis Data Analytics 的扩展程序。
- 有关更多信息，请参阅FLOOR (p. 138)。

EXP

```
EXP ( <number-expression> )
```

返回 e 的值（大约 2.7182828284590455）以输入参数的幂为单位。当输入参数为空时返回空值。

示例

函数	结果
EXP (1)	2.7182818284590455
EXP (0)	1.0
EXP (-1)	0.36787944117144233
EXP (10)	22026.465794806718
EXP (2.5)	12.182493960703473

FLOOR

```
FLOOR ( <time-unit> )
```

当使用数字参数调用时，FLOOR 返回等于或小于输入参数的最大整数。

当与日期、时间或时间戳表达式一起调用时，FLOOR 将返回小于或等于输入的最大值，取决于 <time unit> 指定的精度。

如果任何输入参数为 null，则 FLOOR 返回 null。

示例

函数	结果
地板 (2.0)	2
地板 (-1.0)	–1
地板 (5.2)	5
地板 (-3.3)	-4
楼层 (-3 * 3.1)	-10
楼层（时间戳 '2004-09-30 13:48:23 '到小时）	时间戳 '2004-09-30 13:00:00 '
楼层（时间戳 '2004-09-30 13:48:23 '到分钟）	时间戳 '2004-09-30 13:48:00 '
楼层（时间戳 “2004-09-30 13:48:23” 至当天）	时间戳 '2004-09-30 00:00:00 .0'
楼层（时间戳 '2004-09-30 13:48:23 '到年份）	时间戳 '2004-01-01 00:00:00 .0'

注意

Note

FLOOR (T <datetime expression>O<timeunit>) 是Amazon Kinesis Data Analytics 的扩展程序。STEP 函数与 FLOOR 类似，但可以将值舍入为任意间隔，例如 30 秒。有关更多信息，请参阅STEP (p. 141)。

LN

```
LN ( <number-expression> )
```

返回输入参数的自然对数（即相对于基数 e 的对数）。如果参数为负数或 0，则引发异常。当输入参数为空时返回空值。

有关更多信息，请参阅LOG10 (p. 140)和EXP (p. 138)。

示例

函数	结果
LN (1)	0.0
LN (10)	2.302585092994046

函数	结果
LN (2.5)	0.9162907318741551

LOG10

```
LOG10 ( <number-expression> )
```

返回输入参数的以 10 为底的对数。如果参数为负数或 0，则引发异常。当输入参数为空时返回空值。

示例

函数	结果
LOG10 (1)	0.0
LOG10 (100)	2.0
log10（强制转换（'23' 为十进制））	1.3617278360175928

Note

LOG10 不是 SQL: 2008 标准函数；它是该标准的Amazon Kinesis Data Analytics 扩展。

MOD

```
MOD ( <dividend>, <divisor> )
 <dividend> := <integer-expression>
 <divisor>  := <integer-expression>
```

返回第一个参数（除数除以第二个数字参数（除数）时的余数。如果除数为零，则引发除以零错误。

示例

函数	结果
模组 (4,2)	0
模组 (5,3)	2
模组 (-4,3)	–1
模组 (5,12)	5

限制

Amazon Kinesis Data Analytics 模组函数仅支持小数位为 0（整数）的参数。这与 SQL: 2008 标准背道而驰，后者支持任何数字参数。当然，其他数值参数可以使用 CAST 转换为整数。

POWER

```
POWER ( <base>, <exponent> )
<base> := <number-expression>
<exponent> := <number-expression>
```

返回第一个参数（基数）的值，取第二个参数（指数）的乘方。如果基数或指数为 null，则返回 null；如果基数为零且指数为负，或者如果基数为负且指数不是整数，则引发异常。

示例

函数	结果
力量 (3,2)	9
功率 (-2,3)	-8.
电源 (4, -2)	1/16 ..或.. 0.0625
功率 (10.1,2.5)	324.19285157140644

STEP

```
STEP ( <time-unit> BY INTERVAL '<integer-literal>' <interval-literal> )
STEP ( <integer-expression> BY <integer-literal> )
```

STEP 将输入值（<time-unit> 或 <integer-expression>）向下舍入到 <integer-literal> 的最接近的倍数。

STEP 函数适用于日期时间数据类型或整数类型。STEP 是一个标量函数，用于执行类似于 FLOOR (p. 138) 的操作。但是，通过使用 STEP，您可以指定一个任意时间或整数间隔来向下舍入第一个参数。

如果任何输入参数为 null，则 STEP 返回 null。

具有整数参数的 STEP

当使用整数参数调用时，STEP 返回 <interval-literal> 参数的满足以下条件的最大整数倍数：即等于或小于 <integer-expression> 参数。例如，`STEP(23 BY 5)` 返回 `20`，因为 20 是 5 的满足以下条件的最大倍数：即小于 23。

`STEP ( <integer-expression > BY <integer-literal> )` 等效于以下内容。

```
( <integer-expression> / <integer-literal> ) * <integer-literal>
```

示例

在以下示例中，返回值是 <integer-literal> 的满足以下条件的最大倍数：即等于或小于 <integer-expression>。

函数	结果
STEP(23 BY 5)	20

函数	结果
STEP(30 BY 10)	30

具有日期类型参数的 STEP

当通过日期、时间或时间戳参数调用时，STEP 将返回小于或等于输入的最大值，具体取决于 <time unit> 指定的精度。

`STEP(<datetimeExpression> BY <intervalLiteral>)` 等效于以下内容。

```
(<datetimeExpression> - timestamp '1970-01-01 00:00:00')  /  <intervalLiteral> )  *
 <intervalLiteral> + timestamp '1970-01-01 00:00:00'
```

<intervalLiteral> 可以是以下项之一：

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND

示例

在以下示例中，返回值是以 <integer-literal> 指定的单位表示的 <intervalLiteral> 最新倍数，它等于或早于 <datetime-expression>。

函数	结果
STEP(CAST('2004-09-30 13:48:23' as TIMESTAMP) BY INTERVAL '10' SECOND)	'2004-09-30 13:48:20'
STEP(CAST('2004-09-30 13:48:23' as TIMESTAMP) BY INTERVAL '2' HOUR)	'2004-09-30 12:00:00'
STEP(CAST('2004-09-30 13:48:23' as TIMESTAMP) BY INTERVAL '5' MINUTE)	'2004-09-30 13:45:00'
STEP(CAST('2004-09-27 13:48:23' as TIMESTAMP) BY INTERVAL '5' DAY)	'2004-09-25 00:00:00.0'
STEP(CAST('2004-09-30 13:48:23' as TIMESTAMP) BY INTERVAL '1' YEAR)	'2004-01-01 00:00:00.0'

GROUP BY 子句中的 STEP（滚动窗口）

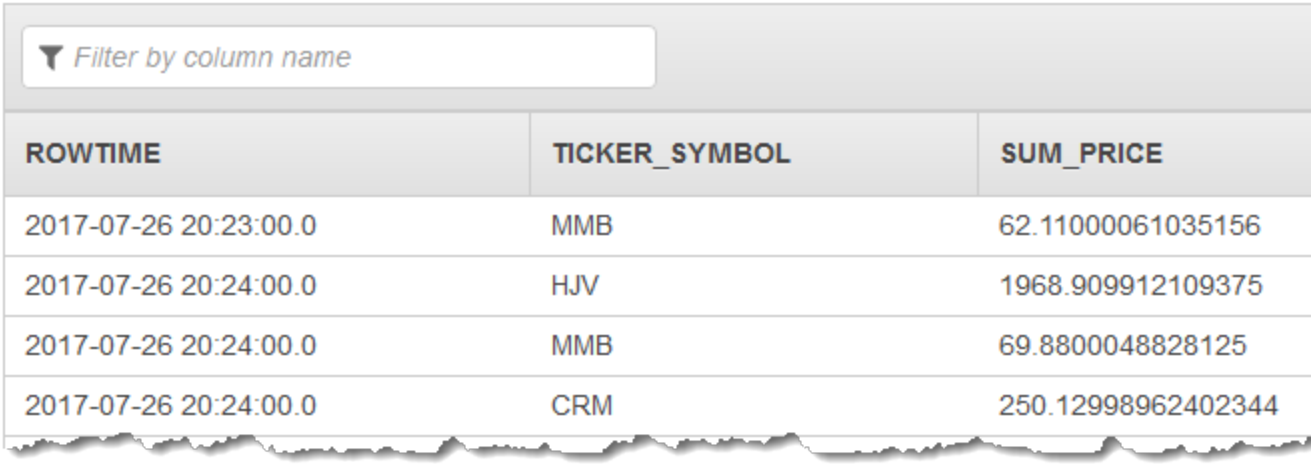
在此示例中，聚合查询具有一个 `GROUP BY` 子句，该子句将 `STEP` 应用于将流分组为有限行的 `ROWTIME`。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
```

```
    ticker_symbol VARCHAR(4),
    sum_price     DOUBLE);
-- CREATE OR REPLACE PUMP to insert into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
  INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM
        ticker_symbol,
        SUM(price) AS sum_price
    FROM "SOURCE_SQL_STREAM_001"
     GROUP BY ticker_symbol, STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '60' SECOND);
```

结果

上一示例输出的流与以下内容类似。

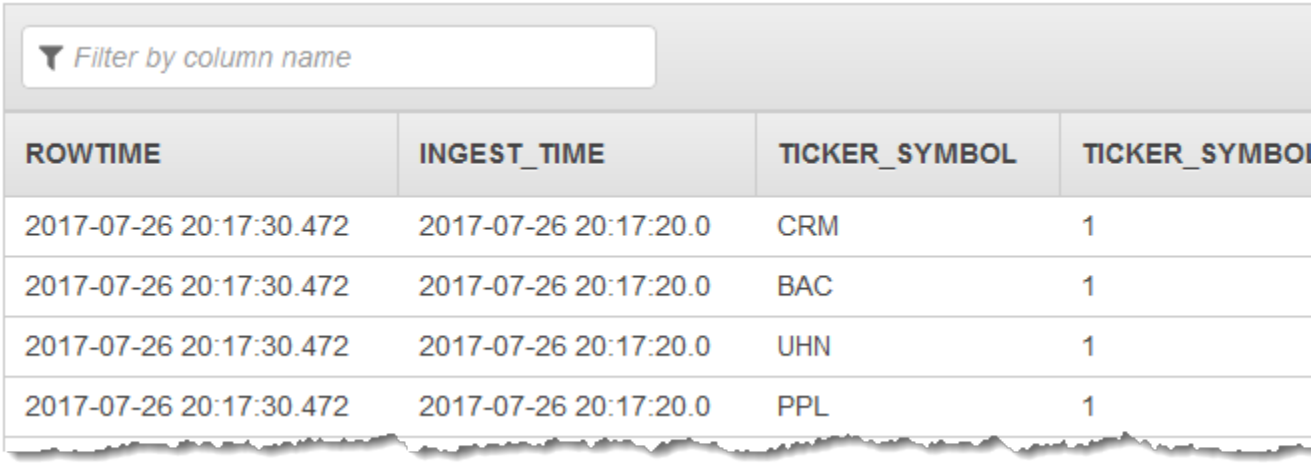


OVER 子句中的 STEP（滑动窗口）

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ingest_time TIMESTAMP,
    ticker_symbol VARCHAR(4),
    ticker_symbol_count integer);

--Create pump data into output
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
INSERT INTO "DESTINATION_SQL_STREAM"
-- select the ingest time used in the GROUP BY clause
SELECT STREAM STEP(source_sql_stream_001.approximate_arrival_time BY INTERVAL '10' SECOND)
 as ingest_time,
    ticker_symbol,
   count(*) over w1 as ticker_symbol_count
FROM source_sql_stream_001
WINDOW w1 AS (
    PARTITION BY ticker_symbol,
   -- aggregate records based upon ingest time
       STEP(source_sql_stream_001.approximate_arrival_time BY INTERVAL '10' SECOND)
   -- use process time as a trigger, which can be different time window as the aggregate
   RANGE INTERVAL '10' SECOND PRECEDING);
```

结果

上一示例输出的流与以下内容类似。


Filter by column name

ROWTIME	INGEST_TIME	TICKER_SYMBOL	TICKER_SYMBOL
2017-07-26 20:17:30.472	2017-07-26 20:17:20.0	CRM	1
2017-07-26 20:17:30.472	2017-07-26 20:17:20.0	BAC	1
2017-07-26 20:17:30.472	2017-07-26 20:17:20.0	UHN	1
2017-07-26 20:17:30.472	2017-07-26 20:17:20.0	PPL	1

注意

STEP (<datetime expression>BY<literal expression>) 是Amazon Kinesis Data Analytics 扩展程序。

您可以使用 STEP 通过滚动窗口聚合结果。有关滚动窗口的详细信息，请参阅滚动窗口概念。

日志解析函数

Amazon Kinesis Data Analytics 具有以下日志解析功能：

- FAST_REGEX_LOG_PARSER (p. 144)工作原理与正则表达式解析器类似，但需要多个 “快捷方式” 以确保更快的结果。例如，较快的正则表达式解析程序会在找到第一个匹配项时停止 (称为“延迟”语义。)
- FIXED_COLUMN_LOG_PARSE (p. 148) 解析固定宽度的字段，并自动将其转换为给定的 SQL 类型。
- REGEX_LOG_PARSE (p. 149)使用默认的 Java 正则表达式解析器。有关此解析程序的更多信息，请参阅 Oracle 网站上的 Java 平台文档中的模式。
- SYS_LOG_PARSE (p. 151) 处理常在 UNIX/Linux 系统日志中找到的条目。
- VARIABLE_COLUMN_LOG_PARSE (p. 151) 将一个输入字符串（其第一个参数 <character-expression>）拆分为由分隔符或分隔符字符串分隔的多个字段。
- W3C_LOG_PARSE (p. 152) 处理 W3C 预定义格式日志中的条目。

FAST_REGEX_LOG_PARSER

```
FAST_REGEX_LOG_PARSE('input_string', 'fast_regex_pattern')
```

FAST_REGEX_LOG_PARSE 的工作原理是首先将正则表达式分解为一系列正则表达式，一个用于组内的每个表达式，一个用于组外的每个表达式。任何表达式开头处的任何固定长度部分都将移至前一个表达式的末尾。如果任何表达式的长度完全固定，则将其与前一个表达式合并。然后使用没有回溯的懒惰语义对一系列表达式进行评估。（用正则表达式解析的话来说，“懒惰” 意味着每一步解析的次数都不要超过你需要的范围。 “贪婪” 意味着在每一步都尽可能多地解析。）

返回的列将是 COLUMN1 到 ColumnN，其中 n 是正则表达式中的组数。这些列的类型将为 varchar (1024)。 请参阅下文中“第一个 FRLP 示例”和“更多 FRLP 示例”中的示例用法。

FAST_REGEX_LOG_PARSER (FRLP)

FAST_REGEX_LOG_PARSER 使用延迟搜索——它会在第一场比赛时停止。相比之下，REGEX_LOG_PARSE (p. 149)除非使用所有格量词，否则是贪婪的。

FAST_REGEX_LOG_PARSE 在提供的输入字符串中扫描快速正则表达式模式指定的所有字符。

- 该输入字符串中的所有字符都必须由 Fast Regex 模式中定义的字符和扫描组来解释。扫描组定义 fields-or-columns扫描成功时生成。
- 如果在应用 Fast Regex 模式时考虑了 input_string 中的所有字符，则 FRLP 会根据该快速正则表达式中的每个圆括号表达式创建一个输出字段（列） left-to-right ORDER。第一个（最左边）括号表达式创建第一个输出字段，下一个（第二个）圆括号表达式创建第二个输出字段，直到最后一个括号表达式创建最后一个输出字段。
- 如果 input_string 包含任何未通过应用 Fast Regex 模式解释（匹配）的字符，则 FRLP 根本不返回任何字段。

快速正则表达式的字符类符号

Fast Regex 使用了一组与常规正则表达式解析器不同的字符类符号：

符号或构造	意义
-	字符范围，包括端点
[角色类]	字符类
[^ 字符类]	否定的字符类
|	Union
&	十字路口
?	匹配零或一个匹配项
*	匹配零或多个匹配项
+	一个或多个匹配项
{n}	n 匹配项
{n,}	n 或多个匹配项
{n, m}	n 到 m 次出现次数，包括两者
.	任何单个字符
#	空的语言
@	任何字符串
"<Unicode string without double-quotes>"	一个字符串）
( )	空字符串）
(unionexp)	优先顺序
< <identifier> >	命名模式
<n-m>	数值间隔

符号或构造	意义
charexp:=<Unicode character>	单个非保留字符
\ <Unicode character>	单个字符)

我们支持以下 POSIX 标准标识符作为命名模式：

<Digit> - "[0-9]"

<Upper> - "[A-Z]"

<Lower> - "[a-z]"

<ASCII> - "[\u0000-\u007F]"

<Alpha> - "<Lower>|<Upper>"

<Alnum> - "<Alpha>|<Digit>"

<Punct> - "[!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~]"

<Blank> - "[\t]"

<Space> - "[\t\n\f\r\u000B]"

<Cntrl> - "[\u0000-\u001F\u007F]"

<XDigit> - "0-9a-fA-F"

<Print> - "<Alnum>|<Punct>"

<Graph> - "<Print>"

第一个 FRLP 示例

第一个示例使用 Fast Regex 模式 '(.*)_(._.*)_.*'

```
select t.r."COLUMN1", t.r."COLUMN2" from
. . . . . .> (values (FAST_REGEX_LOG_PARSE('Mary_had_a_little_lamb',
 '(.*)_(._.*)_.*'))) t(r);
+------------------------+-----------------------+
|         COLUMN1        |         COLUMN2       |
+------------------------+-----------------------+
| Mary_had               |     a_little_lamb     |
+------------------------+-----------------------+
1 row selected
```

1. input_string ('Mary_had_a_little_lamb') 的扫描从 Fast Regex 模式中定义的第一个组开始：(.*)，这意味着“查找任何字符 0 次或更多次。”

   '(.*)_(._.*)_.*'
2. 这个组规范定义了要解析的第一列，它要求 Fast Regex Log Parser 接受从输入字符串的第一个字符开始的输入字符串字符，直到它在 Fast Regex 模式中找到下一个组或下一个不在组中的文字字符或字符串（不在括号）。在此示例中，第一个组后面的下一文本字符为下划线：

   '(.*)_(._.*)_.*'
3. 解析器扫描输入字符串中的每个字符，直到找到 Fast Regex 模式中的下一个规范：下划线：

'(.*)_(._.*)_.*'

4. 因此，组 2 以 “a_l” 开头。接下来，解析器需要使用模式中的剩余规范来确定该组的结尾：

   '(.*)_(._.*)_.*'

   Note

   在模式中指定但不在组内指定的字符串或文字必须在输入字符串中找到，但不能包含在任何输出字段中。
   如果 Fast Regex 模式省略了最后的星号，则不会获得任何结果。

更多的 FRLP 示例

下一个示例使用 “+”，这意味着重复最后一个表达式 1 次或多次（“*” 表示 0 次或多次）。

示例 A

在这个例子中，最长的前缀是第一个下划线。第一个字段/列组将在 “Mary” 上匹配，第二个字段/列组不匹配。

```
select t.r."COLUMN1", t.r."COLUMN2" from
      . . . . . .> (values (FAST_REGEX_LOG_PARSE('Mary_had_a_little_lamb',
        '(.*)_+(._.*)'))) t(r);
      +----------+----------+
      | COLUMN1  | COLUMN2  |
      +----------+----------+
      +----------+----------+
      No rows selected
```

前面的示例不返回任何字段，因为必填的 “+” 至少还有一个字段 underscore-in-a-row; 而且 input_string 没有那个。

示例 B

在以下情况下，'+'是多余的，因为语义很懒惰：

```
select t.r."COLUMN1", t.r."COLUMN2" from
      . . . . . .> (values (FAST_REGEX_LOG_PARSE('Mary____had_a_little_lamb',
        '(.*)_+(.*)'))) t(r);
      +-------------------------+-------------------------+
      |         COLUMN1         |         COLUMN2         |
      +-------------------------+-------------------------+
      | Mary                    |    had_a_little_lamb    |
      +-------------------------+-------------------------+
      1 row selected
```

上一个示例成功返回两个字段，因为在找到“_+”规范所需的多条下划线后，组 2 规范 (.*) 接受 .input_string 中的所有剩余字符。下划线不在 “Mary” 后面，也不会在 “had” 开头，因为 “_+” 规范没有用括号括起来。

正如导言中提到的，正则表达式解析术语中的 “懒惰” 意味着每一步解析的次数都不要超过你需要的范围；“贪婪” 意味着在每一步都尽可能多地解析。

本主题中的第一个示例 A 失败，因为当它到达第一条下划线时，正则表达式处理器在没有回溯的情况下无法获知它无法使用下划线来匹配“_+”并且 FRLP 将不回溯，而 REGEX_LOG_PARSE (p. 149) 将回溯。

正上方的搜索 B 变成了三个搜索：

```
(.*)_
_*(._
.*)
```

请注意，第二个字段组在第二次和第三次搜索之间被分开，而且 “_+” 被认为与 “__*” 相同（也就是说，它认为 “下划线 repeat-underscore-1-or-more-times“同样” 下划线下划线 repeat-underscore-0-or-more-times“。 ）

案例 A 演示了 REGEX_LOG_PARSE 和 FAST_REGEX_LOG_PARSE 之间的主要区别，因为 A 中的搜索可以在 REGEX_LOG_PARSE 下运行，因为该函数会使用回溯法。

示例 C

在以下示例中，加号不是多余的，因为“<Alpha>（任何字母字符）”的长度是固定的，因此将用作“+”搜索的分隔符。

```
select t.r."COLUMN1", t.r."COLUMN2" from
. . . . . .> (values (FAST_REGEX_LOG_PARSE('Mary____had_a_little_lamb',
 '(.*)_+(<Alpha>.*)'))) t(r);
+---------------------------+---------------------------+
|          COLUMN1          |          COLUMN2          |
+---------------------------+---------------------------+
| Mary                      | had_a_little_lamb         |
+---------------------------+---------------------------+
1 row selected

'(.*) +(<Alpha>.*)' gets converted into three regular expressions:
'.* '
' *<Alpha>'
'.*$'
```

每个都使用懒惰语义依次匹配。

返回的列将是 COLUMN1 到 ColumnN，其中 n 是正则表达式中的组数。这些列的类型将为 varchar (1024)。

FIXED_COLUMN_LOG_PARSE

解析固定宽度的字段并自动将其转换为给定的 SQL 类型。

```
FIXED_COLUMN_LOG_PARSE ( <string value expression>, <column description string
 expression> )
 <column description string expression> := '<column description> [,...]'
 <column description> :=
   <identifier> TYPE <data type> [ NOT NULL ]
   START <numeric value expression> [FOR <numeric constant expression>]
```

列的起始位置为 0。DATE、TIME 和 TIMESTAMP 类型的列规范支持格式参数，允许用户指定精确的时间分量布局。解析器使用 Java 类java.lang。SimpleDateFormat解析类型 DATE、TIME 和 TIMESTAMP 的字符串。这些区域有：日期和时间模式 (p. 129)主题提供了时间戳格式字符串的完整描述和示例。以下为使用格式字符串定义的列的示例：

```
"name" TYPE TIMESTAMP 'dd/MMM/yyyy:HH:mm:ss'
```

相关主题

REGEX_LOG_PARSE (p. 149)

REGEX_LOG_PARSE

```
REGEX_LOG_PARSE (<character-expression>,<regex-pattern>,<columns>)<regex-pattern> :=
 <character-expression>[OBJECT] <columns> := <columnname> [ <datatype> ] {, <columnname>
 <datatype> }*
```

基于 java.util.regex.pattern 中定义的 Java 正则表达式模式解析字符串。

列基于正则表达式模式中定义的匹配组。每个组定义一列，各组按从左到右的顺序处理。不匹配会生成 NULL 值结果：如果正则表达式与作为第一个参数传递的字符串不匹配，则返回 NULL。

返回的列将是 COLUMN1 到 ColumnN，其中 n 是正则表达式中的组数。这些列的类型将为 varchar (1024)。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是开始使用在里面Amazon Kinesis Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置示例股票代码输入流，请参阅《Amazon Kinesis Analytics 开发人员指南》 中的入门练习。

具有以下架构的示例股票数据集。

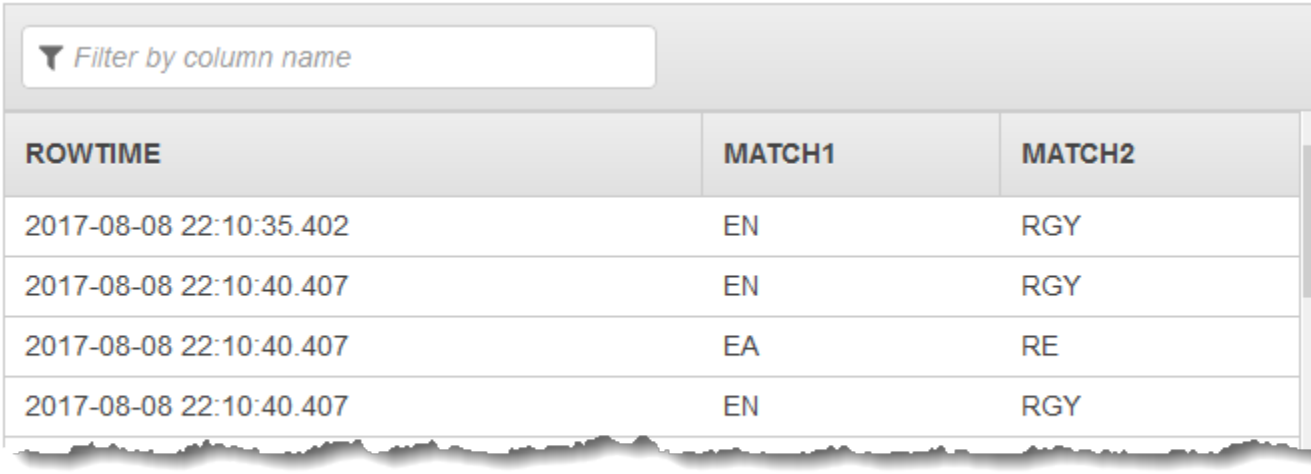
```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：返回来自两个捕获组的结果

以下代码示例搜索 `sector` 字段的内容以查找字母 `E` 及其后面的字符，然后搜索字母 R，并返回该字母及其后面的所有字符：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (match1 VARCHAR(1024), match2
 VARCHAR(1024));

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM T.REC.COLUMN1, T.REC.COLUMN2
    FROM
         (SELECT STREAM SECTOR,
             REGEX_LOG_PARSE(SECTOR, '.*([E].).*([R].*)') AS REC
             FROM SOURCE_SQL_STREAM_001) AS T;
```

前面的代码示例生成类似于以下内容的结果：


Filter by column name

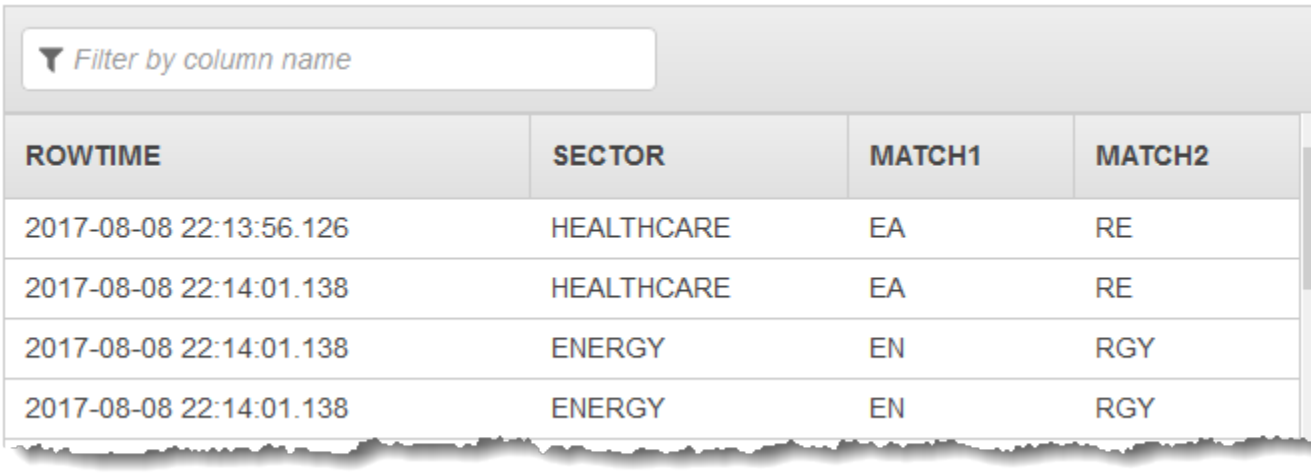
ROWTIME	MATCH1	MATCH2
2017-08-08 22:10:35.402	EN	RGY
2017-08-08 22:10:40.407	EN	RGY
2017-08-08 22:10:40.407	EA	RE
2017-08-08 22:10:40.407	EN	RGY

示例 2：返回来自两个捕获组的流字段和结果

以下代码示例返回 `sector` 字段，搜索 `sector` 字段的内容以查找字母 `E` 并返回此字母及其后面的字符，然后搜索字母 `R`，并返回该字母及其后面的所有字符：

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (sector VARCHAR(24), match1 VARCHAR(24),
 match2 VARCHAR(24));

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM T.SECTOR, T.REC.COLUMN1, T.REC.COLUMN2
    FROM
         (SELECT STREAM SECTOR,
             REGEX_LOG_PARSE(SECTOR, '.*([E].).*([R].*)') AS REC
             FROM SOURCE_SQL_STREAM_001) AS T;
```

前面的代码示例生成类似于以下内容的结果：


Filter by column name

ROWTIME	SECTOR	MATCH1	MATCH2
2017-08-08 22:13:56.126	HEALTHCARE	EA	RE
2017-08-08 22:14:01.138	HEALTHCARE	EA	RE
2017-08-08 22:14:01.138	ENERGY	EN	RGY
2017-08-08 22:14:01.138	ENERGY	EN	RGY

有关更多信息，请参阅 FAST_REGEX_LOG_PARSER (p. 144)。

快速正则表达式参考

有关正则表达式的完整详细信息，请参阅 java.util.regex.pattern

[xyz] 查找单个字符：x、y 或 z	\ w 查找任意单词字符（字母、数字、下划线）
[^abc] 查找除了 x、y 或 z 之外的任意单个字符	\ W 找到任何非单词字符
[r-z] 查找 r-z 之间的任意单个字符	\ b 找到任意单词边界
[r-zr-z] 查找 r-z 或 R-Z 之间的任意单个字符	(...) 捕获随附的所有内容
^ 行开头	(x|y) 查找 x 或 y（也适用于诸如\ d 或\ s 之类的符号）
$ 行尾	x？找到 x 中的零或一个（也适用于诸如\ d 或\ s 之类的符号）
\ A 字符串的开头	x* 查找 x 中的零个或多个（也适用于诸如\ d 或\ s 之类的符号）
\ z 字符串结尾	x+ 找到 x 中的一个或多个（也适用于诸如\ d 或\ s 之类的符号）
. 任何单个字符	x {3} 精确找到 x 中的 3 个（也适用于诸如\ d 或\ s 之类的符号）
\ s 找到任何空格字符	x {3,} 查找 x 中的 3 个或更多（也适用于诸如\ d 或\ s 之类的符号）
\ S 查找任何非空格字符	x {3,6} 在 x 的 3 到 6 之间找到（也适用于诸如\ d 或\ s 之类的符号）
\ d 查找任意数字	
\ D 找到任何非数字	

SYS_LOG_PARSE

解析标准系统日志格式：

```
Mon DD HH:MM:SS server message
```

SYS_LOG_PARSE 处理 UNIX/Linux 系统日志中常见的条目。系统日志条目以时间戳开头，后跟自由格式文本字段。SYS_LOG_PARSE 输出由两列组成。第一列名为 “COLUMN1”，是 SQL 数据类型为 TIMESTAMP。第二列名为 “COLUMN2”，SQL 类型为 VARCHAR ()。

Note

有关 SYSLOG 的更多信息，请参阅 IETF RFC3164。有关日期时间模式和匹配的更多信息，请参见日期和时间模式 (p. 129).

VARIABLE_COLUMN_LOG_PARSE

```
VARIABLE_COLUMN_LOG_PARSE(
 <character-expression>, <columns>, <delimiter-string>
 [ , <escape-string>, <quote-string> ] )
 <columns> := <number of columns> | <list of columns>
 <number of columns> := <numeric value expression>
 <list of columns> := '<column description>[, ...]'
```

```
<column description> := <identifier> TYPE <data type> [ NOT NULL ]
<delimiter string> := <character-expression>
<escape-string> := <character-expression>
<quote-string> := '<begin quote character> [ <end quote character> ]'
```

VARIABLE_COLUMN_LOG_PARSE 将一个输入字符串（其第一个参数 <character-expression>）拆分为由分隔符或分隔符字符串分隔的多个字段。因此，它处理以逗号分隔的值或制表符分隔的值。它可以与FIXED_COLUMN_LOG_PARSE (p. 148)处理像 maillog 这样的东西，其中一些字段是固定长度的，而另一些是可变长度的。

Note

不支持解析二进制文件。

参数 <escape-string> 和 <quote-string> 是可选的。指定 <escape-string> 可让字段的值包含嵌入式分隔符。举一个简单的例子，如果 <delimiter-string> 指定了一个逗号，<escape-string> 指定了一个反斜杠，则输入“a,b”将拆分为两个字段“a”和“b”，但输入“a\,b”将生成一个字段“a,b”。

由于Amazon Kinesis Data Analytics表达式和文字 (p. 17)，制表符也可以是分隔符，使用 unicode 转义指定，例如 u&'\ 0009'，它是一个仅由制表符组成的字符串。

指定 <quote-string> 是隐藏嵌入式分隔符的另一种方法。<quote-string> 应为单字符或双字符表达式：第一个字符用作 <begin quote character> 字符；第二个字符（如果有）用作 <end quote character> 字符。如果只提供一个字符，则它既用作带引号的字符串的开头又用作结尾的带引号的字符串。如果输入包含一个带引号的字符串（即，包含在指定为 <quote-string> 的字符中的字符串），该字符串将出现在一个字段中，即使它包含一个分隔符。

请注意，<begin quote character> 和 <end quote character> 是单字符，而且可以不同。<begin quote character> 可用于开始和结束带引号的字符串，或者 <begin quote character> 可以开始带引号的字符串而 <end quote character> 用于结束带引号的字符串。

当列 <list of columns> 的列表作为第二个参数 <columns> 提供时，类型 DATE、TIME 和 TIMESTAMP 的列规范 (<column description>) 支持允许用户指定确切的时间部分布局的格式参数。解析器使用 Java 类java.lang。SimpleDateFormat解析这些类型的字符串。日期和时间模式 (p. 129)提供了时间戳格式字符串的完整描述，并附有示例。以下为使用格式字符串定义的列的示例：

```
    "name" TYPE TIMESTAMP 'dd/MMM/yyyy:HH:mm:ss'
```

默认情况下，输出列被命名为 COLUMN1、COLUMN2、COLUMN3 等，每个列都是 SQL 数据类型 VARCHAR (1024)。

W3C_LOG_PARSE

```
W3C_LOG_PARSE( <character-expression>, <format-string> )
<format-string> := '<predefined-format> | <custom-format>'
<predefined format> :=
        COMMON
      | COMMON WITH VHOST
      | NCSA EXTENDED
      | REFERER
      | AGENT
      | IIS
<custom-format> := [an Apache log format specifier]
```

W3C 预定义格式

使用指示的格式说明符指定以下 W3C 预定义格式名称进行汇总，如以下语句所示：

```
select stream W3C_LOG_PARSE(message, 'COMMON') r  from w3ccommon t;
```

格式名称	W3C 姓名	格式说明符
常见的	常用日志格式 (CLF)	%h %l %u %t “%r” %>s %b
虚拟主机常用	虚拟主机的常用日志格式	%v %h %l %u %t “%r” %>s %b
NCSA 已扩展	NCSA 扩展/组合日志格式	%h %l %u %t “%r” %>s %b “% [Referer] i” “% [User-Agent] i”
裁判员	Referer 日志格式	% [Referer] i —> %U
代理人	代理（浏览器）日志格式	% [用户代理] i

W3C 格式说明符

下面列出了格式说明符。W3C_LOG_PARSE 会自动检测这些说明符和输出记录，每个说明符占一列。列的类型是根据说明符的可能输出自动选择的。例如，%b 表示处理 HTTP 请求时发送的字节数，因此列类型为数字。但是，对于 %B，零字节用破折号表示，强制列类型为文本。注释 A 解释说明符表中显示的“...”和“<”或“>”标记的含义。

下表按命令的字母顺序列出了 W3C 格式说明符。

格式说明符	说明
%	百分号（Apache 2.0.44 及更高版本）
%... a	远程 IP 地址
%... A	本地 IP 地址
%... B	以字节为单位的响应大小，不包括 HTTP 标头。
%... b	以字节为单位的响应大小，不包括 HTTP 标头，采用 CLF 格式，这意味着当没有发送字节时，使用 '-' 而不是 0。
%... [客户数据] C	发送到服务器的请求中 Cookie 客户数据的内容。
%... D	处理请求所花费的时间，以微秒为单位。
%... [客户数据]	环境变量 CUSTOMERDATA 的内容
%... f	文件名
%... h	远程主机
%... H	请求协议
%... [客户数据] i	客户数据的内容：发送到服务器的请求中的标题行。
%... l	远程登录名（如果提供，来自 identd）
%... m	请求方法

格式说明符	说明
%... [客户数据]	备注内容来自另一个模块的客户数据。
%... [客户数据] o	客户数据的内容：回复中的标题行。
%... p	为请求提供服务的服务器的规范端口
%... P	为请求提供服务的孩子的进程 ID。
%... [格式] P	为请求提供服务的子进程的进程 ID 或线程 ID。有效格式为 pid 和 tid。（Apache 2.0.46 及更高版本）
%... q	查询字符串（如果查询字符串存在，则前面有一个“?”，否则为空字符串）
%... r	请求的第一行
%...	状态。对于内部重定向的请求，这是*原始*请求的状态 —%... >s 代表最后一个。
%... t	时间，采用常用日志格式时间格式（标准英文格式）
%... [格式] t	采用指定格式表示的时间，应采用 strimmer(3) 格式。（可能已本地化）
%... T	处理请求所花费的时间，以秒为单位。
%... u	远程用户（来自身份验证；如果返回状态 (%s) 为 401，则可能是假的）
%... U	请求的 URL 路径，不包括任何查询字符串。
%... v	canonical ServerName 服务请求的服务器。
%... V	根据的服务器名称 UseCanonicalName 设置。
%... X	响应完成时的连接状态 X = 在响应完成之前连接中止。 + = 发送响应后，连接可能会保持活动状态。 -= 发送响应后连接将关闭。 （%..X 指令在 Apache 1.3 的后期版本中为 %...c， 但这与历史上的 ssl% 相冲突... [var] c 语法。）
:%... I:	接收的字节（包括请求和标头）不能为零。您需要启用 mod_logio 才能使用此项。
:%... O:	发送的字节（包括标头）不能为零。您需要启用 mod_logio 才能使用此项。

Note

一些 W3C 格式说明符显示为包含“...”指示或“<”或“>”，它们在隐藏或重定向该说明符的输出时为可选控件。“...” 可以是空的（如通用规范 “\ %h %u %r\ %s %b”），也可以表示包含该商品的条件。条件是一个 HTTP 状态码列表，可能以 “!” 开头 ，如果未满足指定条件，则返回的列或字段显示 “-”。

例如，如 Apache 文档中所述，指定“%400,501[User-agent]i”将仅在发生 400 错误和 501 错误（错误请求，未实现）时记录 User-agent。类似地，“%!200,304,302[Referer]i”将在无法返回某种正常状态的所有请求发生时记录 Referer:。
当已在内部重定向请求时，修饰符“<”和“>”可用于选择是应参阅原始请求，还是应参阅最终请求（分别）。默认情况下，% 指令 %s、%U、%T、%D 和 %r 查看原始请求，而所有其他指令查看最终请求。例如，%>s 可用于记录请求的最终状态，而 %<u 可用于针对已内部重定向到未经身份验证的资源的请求来记录原始经身份验证的用户。
出于安全考虑，从 Apache 2.0.46 开始，不可打印字符和其他特殊字符主要通过使用\ xhh 序列进行转义，其中 hh 代表原始字节的十六进制表示形式。此规则的例外是 “” 和\，它们通过在前面加上反斜杠进行转义，以及所有以 C 风格表示法（\ n、\ t 等）书写的空格字符。在 2.0.46 之前的 httpd 2.0 版本中，没有对来自%... r、%... i 和%... o 的字符串进行转义，因此在处理原始日志文件时需要格外小心，因为客户端可能会在日志中插入控制字符。
此外，在 httpd 2.0 中，B 格式字符串仅表示 HTTP 响应的大小（例如，如果连接中断或使用 SSL，则会有所不同）。对于通过网络发送到客户端的实际字节数，请使用 mod_logio 提供的 %O 格式。

按函数或类别划分的 W3C 格式说明符

类别包括发送的字节、连接状态、环境变量的内容、文件名、主机、IP、注释、协议、查询字符串、回复、请求和时间。对于标记“...”或“<”或“>”，请参阅上一个注释。

函数或类别	W3C 格式说明符
发送的字节数，不包括 HTTP 标头	
不发送字节时使用 “0”	%... B
不发送字节时使用 “-”（CLF 格式）	%... b
收到的字节（包括请求和标头）不能为零 必须启用 mod_logio 才能使用此项。	:%... I :
发送的字节（包括标头）不能为零 必须启用 mod_logio 才能使用此项。	:%... O:
响应完成时的连接状态	
在响应完成之前，连接中止	X
发送响应后，连接可能会保持活动状态	+
发送响应后，连接将关闭	-
Note %..X 指令在 Apache 1.3 的后期版本中为 %...c，但这与历史 ssl %...[var]c 语法相冲突。	
环境变量 CUSTOMERDATA	
内容	%... [客户数据]
文件名	%... f
主机（远程）	%... h
协议	%... H
IP 地址	

函数或类别	W3C 格式说明符
远程	%... a
本地	%... A
备注	
备注内容来自另一个模块的客户数据	%... [客户数据]
协议（请求）	%... H
查询字符串 Note 如果存在查询，则在前面加上? 如果不是，则为空字符串。	%... q
回复	
客户数据的内容（回复中的标题行）	%... [客户数据] o

下表列出了响应和时间类别的 W3C 格式说明符。

函数或类别	W3C 格式说明符
请求	
为请求提供服务的服务器的规范端口	%... p
发送到服务器的请求中客户数据的 cookie 内容	%... [客户数据] C
栏目内容:标题行	%... [酒吧]
第一行已发送：	%... r
处理请求所花费的微秒	%... D
协议	%... H
为请求提供服务的孩子的进程 ID	%... P
为请求提供服务的子进程的进程 ID 或线程 ID。 有效格式为 pid 和 tid。（Apache 2.0.46 及更高版本）	%... [格式] P
远程登录名（如果提供，来自 identd）	%... l
远程用户：（来自身份验证；如果返回状态 (%s) 为 401，则可能是虚假的）	%... u
服务器（权威版） ServerName) 处理请求	%... v
服务器名称由 UseCanonicalName 设置	%... V
请求方法	%... m
退货状态	%s

函数或类别	W3C 格式说明符
处理请求所花费的秒数	%... T
内部重定向的 *原始 * 请求的状态	%...
上次请求的状态	%... >
请求的 URL 路径，不包括任何查询字符串	%... U
时间	
常用日志格式时间格式（标准英文格式）	%... t
时间采用 strftime (3) 格式，可能已本地化	%... [格式] t
处理请求所花费的秒数	%... T

W3C 示例

W3C_LOG_PARSE 支持访问由 Apache Web 服务器等兼容 W3C 的应用程序生成的日志，生成输出行，每个说明符占一列。数据类型派生自 Apache mod_log_config 规范中列出的日志条目描述说明符。

示例 1

本示例中的输入取自 Apache 日志文件，代表通用日志格式。

输入

```
(192.168.254.30 - John [24/May/2004:22:01:02 -0700]
                    "GET /icons/apache_pb.gif HTTP/1.1" 304 0),
(192.168.254.30 - Jane [24/May/2004:22:01:02 -0700]
                    "GET /icons/small/dir.gif HTTP/1.1" 304 0);
```

DDL

```
CREATE OR REPLACE PUMP weblog AS
        SELECT STREAM
            l.r.COLUMN1,
            l.r.COLUMN2,
            l.r.COLUMN3,
            l.r.COLUMN4,
            l.r.COLUMN5,
            l.r.COLUMN6,
            l.r.COLUMN7
        FROM (SELECT STREAM W3C_LOG_PARSE(message, 'COMMON')
              FROM "weblog_read) AS l(r);
```

输出

```
192.168.254.30 -  John  [24/May/2004:22:01:02 -0700] GET /icons/apache_pb.gif HTTP/1.1
 304  0
192.168.254.30 -  Jane  [24/May/2004:22:01:02 -0700] GET /icons/small/dir.gif HTTP/1.1
 304  0
```

FROM 子句中 COMMON 的规范是指通用日志格式 (CLF)，它使用说明符 %h %l %u %t “%r” %>s %b。

W3C 预定义的格式显示 COMMON 和其他预定义的说明符集。

FROM 子句中 COMMON 的规范是指通用日志格式 (CLF)，它使用说明符 %h %l %u %t “%r” %>s %b。

下表 “通用日志格式使用的说明符” 描述了 COMMON 在 FROM 子句中使用的说明符。

通用日志格式使用的说明符

输出列	格式说明符	返回值
第 1 栏	%h	远程主机的 IP 地址
第 2 栏	%l	远程登录名称
第 3 栏	%u	远程用户
第 4 栏	%t	time
第 5 栏	“%r”	请求的第一行
第 6 栏	%ess	状态：对于内部重定向的请求， *原始 * 请求的状态 —%... >s 代表最后一个。
第 7 栏	%b	发送的字节数，不包括 HTTP 标头

示例 2

此示例中的 DDL 显示如何重命名输出列和过滤掉不需要的列。

DDL

```
CREATE OR REPLACE VIEW "Schema1".weblogreduced AS
    SELECT STREAM CAST(s.COLUMN3 AS VARCHAR(5)) AS LOG_USER,
    CAST(s.COLUMN1 AS VARCHAR(15)) AS ADDRESS,
    CAST(s.COLUMN4 AS VARCHAR(30)) as TIME_DATES
    FROM "Schema1".weblog s;
```

输出

```
+----------+-----------------+--------------------------------+
| LOG_USER |     ADDRESS     |           TIME_DATES           |
|          |                 |                                |
+----------+-----------------+--------------------------------+
| Jane     | 192.168.254.30  | [24/May/2004:22:01:02 -0700]   |
|          |                 |                                |
| John     | 192.168.254.30  | [24/May/2004:22:01:02 -0700]   |
+----------+-----------------+--------------------------------+
```

W3C 自定义格式

通过直接命名说明符而不是使用 “COMMON” 名称可以得到相同的结果，如下所示：

```
CREATE OR REPLACE FOREIGN STREAM schema1.weblog
    SERVER logfile_server
    OPTIONS (LOG_PATH '/path/to/logfile',
             ENCODING 'UTF-8',
             SLEEP_INTERVAL '10000',
             MAX_UNCHANGED_STATS '10',
             PARSER 'W3C',
             PARSER_FORMAT '%h %l %u %t \"%r\" %>s %b');
or
 CREATE FOREIGN STREAM "Schema1".weblog_read
 SERVER "logfile_server"
 OPTIONS (log_path '/path/to/logfile',
 encoding 'UTF-8',
 sleep_interval '10000',
 max_unchanged_stats '10');
 CREATE OR REPLACE VIEW "Schema1".weblog AS
    SELECT STREAM
        l.r.COLUMN1,
        l.r.COLUMN2,
        l.r.COLUMN3,
        l.r.COLUMN4,
        l.r.COLUMN5,
        l.r.COLUMN6
    FROM (SELECT STREAM W3C_LOG_PARSE(message, '%h %l %u %t \"%r\" %>s %b')
          FROM "Schema1".weblog_read) AS l(r);
```

Note

如果您将 %t 更改为 [%t]，则日期列包含以下内容：

```
        24/May/2004:22:01:02 -0700
```

(而不是[24/May/2004:22:01:02 -0700])

排序函数

本节中的主题描述了 Amazon Kinesis Data Analytics 直播 SQL 的排序函数。

主题

小组Rank

此函数会将 RANK() 函数应用于行的逻辑组，并且（可选）按排序顺序传送该组。

group_rank 的应用包括：

- 对流式处理 GROUP BY 的结果进行排序。
- 确定群组结果中的关系。

Group Rank 可以执行以下操作：

- 将排名应用于指定的输入列。
- 提供已排序或未排序的输出。

- 允许用户为数据刷新指定一段非活动期。

SQL 声明

函数属性和 DDL 将在后面的部分中介绍。

Group_Rank 的函数属性

此函数的作用如下：

- 收集行，直到检测到行时间更改或超过指定的空闲时间限制。
- 接受任何流式处理行集。
- 使用包含基本 SQL 数据类型 INTEGER、CHAR 和 VARCHAR 的任何列作为排名依据。
- 按接收到的顺序或选定列中值的升序或降序对输出行进行排序。

Group_Rank 的 DDL

```
group_rank(c cursor, rankByColumnName VARCHAR(128),
  rankOutColumnName VARCHAR(128), sortOrder VARCHAR(10), outputOrder VARCHAR(10),
  maxIdle INTEGER, outputMax INTEGER)
returns table(c.*, "groupRank" INTEGER)
```

下表中列出了函数。

参数	描述
c	光标转到流式传输结果集
rankByColumnName	一个字符串，指定要用于对组进行排名的列。
rankOutColumnName	字符串命名用于返回排名的列。 此字符串必须与 CREATE FUNCTION 语句的 RETURNS 子句中 groupRank 列的名称匹配。
sortOrder	控制排名分配的行顺序。 有效值如下所示： • 'asc' - 根据排名按升序输出。 • 'desc'-根据排名降序。
outputOrder	控制输出的顺序。有效值如下所示： • 'asc' - 根据排名按升序输出。 • 'desc'-根据排名降序。
maxIdle	持有分组进行排名的时间限制（以毫秒为单位）。 当 maxIdle 过期时，当前组将会释放到流中。值为 0 表示没有空闲超时。
outputMax	该函数将在给定组中输出的最大行数。 值 0 表示没有限制。

示例

示例数据集

以下示例基于样本股票Data，是开始练习在里面Amazon Kinesis Data Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Data Analytics 应用程序。要了解如何创建分析应用程序和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Data Analytics.

示例股票数据集具有以下架构：

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

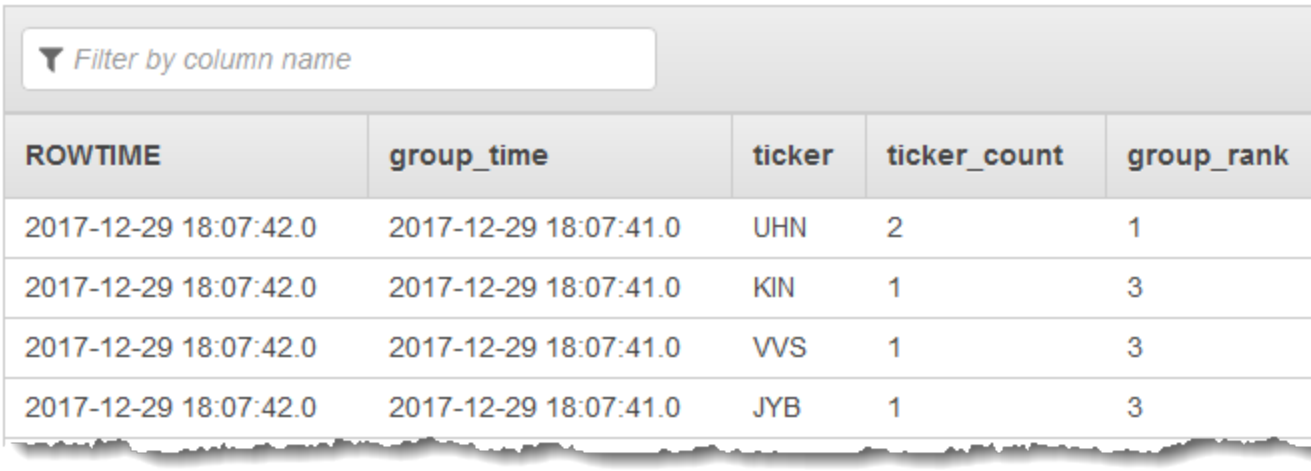
示例 1：对 GROUP BY 子句的结果进行排序

在此示例中，聚合查询有一个GROUP BY子句ROWTIME它将流分组为有限的行。然后，GROUP_RANK 函数对 GROUP BY 子句返回的行进行排序。

```
CREATE OR REPLACE STREAM "ticker_grouped" (
    "group_time" TIMESTAMP,
    "ticker" VARCHAR(65520),
    "ticker_count" INTEGER);

CREATE OR REPLACE STREAM "destination_sql_stream" (
    "group_time" TIMESTAMP,
    "ticker" VARCHAR(65520),
    "ticker_count" INTEGER,
    "group_rank" INTEGER);

CREATE OR REPLACE PUMP "ticker_pump" AS
    INSERT INTO "ticker_grouped"
    SELECT STREAM
        FLOOR(SOURCE_SQL_STREAM_001.ROWTIME TO SECOND),
        "TICKER_SYMBOL",
        COUNT(TICKER_SYMBOL)
    FROM SOURCE_SQL_STREAM_001
    GROUP BY FLOOR(SOURCE_SQL_STREAM_001.ROWTIME TO SECOND), TICKER_SYMBOL;

CREATE OR REPLACE PUMP DESTINATION_SQL_STREAM_PUMP AS
    INSERT INTO "destination_sql_stream"
    SELECT STREAM
        "group_time",
        "ticker",
        "ticker_count",
        "groupRank"
    FROM TABLE(
        GROUP_RANK(
            CURSOR(SELECT STREAM * FROM "ticker_grouped"),
            'ticker_count',
            'groupRank',
            'desc',
            'asc',
            5,
            0));
```

结果

上一示例输出的流与以下内容类似。



Filter by column name

ROWTIME	group_time	ticker	ticker_count	group_rank
2017-12-29 18:07:42.0	2017-12-29 18:07:41.0	UHN	2	1
2017-12-29 18:07:42.0	2017-12-29 18:07:41.0	KIN	1	3
2017-12-29 18:07:42.0	2017-12-29 18:07:41.0	VVS	1	3
2017-12-29 18:07:42.0	2017-12-29 18:07:41.0	JYB	1	3

运营概述

行由每个组的输入游标缓冲（即行时间相同的行）。行排序要么在行时间不同的行到达之后（要么在空闲超时发生时）完成。继续读取行，同时对行时间相同的行组进行排名。

在分配排名后，`outputMax` 参数将指定要为每个组返回的最大行数。

默认情况下，`group_rank` 支持列传递，如示例中所示：使用 `c.*` 作为标准快捷方式以指示按显示的顺序传递所有输入列。您可以改为使用表示法“`c.columnName`”指定一个子集，以便对列进行重新排序。但是，使用特定的列名称会将 UDX 绑定到一个特定的输入集，而使用 `c.*` 表示法可让 UDX 处理任何输入集。

`rankOutColumnName` 参数将指定要用于返回排名的输出列。此列名称必须与 `CREATE FUNCTION` 语句的 `RETURNS` 子句中指定的列名称匹配。

统计方差和偏差函数

这些函数中的每一个都采用一组数字，忽略空值，可以用作聚合函数或分析函数。有关更多信息，请参阅 聚合函数 (p. 69) 和 分析函数 (p. 97)。

下表中描述了这些函数之间的关系。

函数用途	函数名称	公式	注释
热点	HOTSPOTS (p. 163)(expr)	检测数据流中频繁出现的数据的热点。	
随机森林砍伐	RANDOM_CUT_FOREST (在数据流中检测异常。		
随机砍伐森林以及解释	RANDOM_CUT_FOREST_检测数据流中的异常并根据每列中的数据异常情况返回归因分数。		
总体方差	VAR_POP (p. 184)(expr)	(SUM (expr*expr)-总和 (expr) *SUM (expr) /	应用于空集，它返回 null。

函数用途	函数名称	公式	注释
		COUNT (expr)) /COUNT (expr)	
总体标准差	STDDEV_POP (p. 179)	总体方差 (VAR_POP) 的平方根。	当 VAR_POP 返回空值时，STDDEV_POP 返回空值。
示例方差	VAR_SAMP (p. 186)	(SUM (expr*expr)-总和 (expr) *SUM (expr) / COUNT (expr))/(COUNT (expr) −1)	应用于空集，它返回 null。 应用于包含一个元素的输入集，VAR_SAMP 返回空值。
示例标准差	STDDEV_SAMP (p. 181)(expr)	样本方差的平方根 (VAR_SAMP)。	STDDEV_SAMP 仅应用于 1 行输入数据，返回空值。

HOTSPOTS

检测数据流中的热点 或活动明显高于正常情况的区域。热点定义为数据点相对密集的小空间区域。

使用 HOTSPOTS 函数，您可以使用简单 SQL 函数来识别数据中相对密集的区域，而无需显式构建和训练复杂的机器学习模型。然后，您可以识别需要注意的数据子部分，以便立即采取措施。

例如，数据中的热点可能表示数据中心中有过热的服务器集合、车辆高度集中表明交通瓶颈、特定区域中的乘坐共享行程表明交通繁忙事件或具有类似功能的类别中的产品销量增加。

> Note
>
> 的能力HOTSPOTS检测频繁数据点的功能取决于应用程序。 要投射您的业务问题以便通过此功能解决问题，需要领域的专业知识。例如，您可能需要确定输入流中哪些列组合传递给函数，以及如何在必要时对数据进行规范化。

该算法接受DOUBLE,INTEGER,FLOAT,TINYINT,SMALLINT,REAL，以及BIGINT数据类型。DECIMAL 不是支持的类型。请改用 DOUBLE。

> Note
>
> HOTSPOT 函数不返回构成热点的记录。您可以使用 ROWTIME 列以确定哪些记录属于给定的热点。

语法

```
HOTSPOTS (inputStream,
  windowSize,
  scanRadius,
  minimumNumberOfPointsInAHotspot)
```

参数

以下部分介绍 HOTSPOT 函数参数。

输入流

指向您的输入流的指针。您可以使用以下命令来设置指针CURSOR函数。例如，以下语句将设置指向 InputStream 的指针：

```
--Select all columns from input stream
CURSOR(SELECT STREAM * FROM InputStream)
--Select specific columns from input stream
CURSOR(SELECT STREAM PRICE, CHANGE FROM InputStream)
-- Normalize the column X value.
CURSOR(SELECT STREAM IntegerColumnX / 100, IntegerColumnY FROM InputStream)
-- Combine columns before passing to the function.
CURSOR(SELECT STREAM IntegerColumnX - IntegerColumnY FROM InputStream)
```

> **Note**
>
> 仅对输入流中的数值列进行热点分析。HOTSPOTS 函数忽略游标中包含的其他列。

窗口大小

指定滑动窗口在流上为每个时间段考虑的记录数。

您可以将此值设置为介于 100 和 1000 之间（含 100 和 1,000）。

通过增加窗口大小，您可以更好地估计热点位置和密度（相关性），但这也会增加运行时间。

scanRadius

指定热点与其最近相邻点之间的典型距离。

此参数类似于ε中的值 DBSCAN算法。

将此参数设置为一个值，该值小于不在热点中的点之间的典型距离，但足够大，以便热点中的点在此距离内具有相邻点。

您可以将此值设置为任何大于零的双精度值。scanRadius 的值越小，属于同一热点的任何两个记录越相似。但是，较低的 scanRadius 值也会增加运行时间。scanRadius 的值越低，会导致热点越小，但数量越多。

minimumNumberOfPointsIn热点

指定形成热点的记录所需的记录数。

> **Note**
>
> 设置此参数时应考虑窗口大小 (p. 164)。最好将 minimumNumberOfPointsInAHotspot 视为 windowSize 的某个部分。具体是哪个部分可以通过实验发现。

您可以将此值设置为 2 与您为窗口大小配置的值（含）之间。根据您选择的窗口大小值，选择一个最能模拟您要解决的问题的值。

输出

HOTSPOTS 函数的输出是一个表对象，该表对象具有与输入相同的架构，并带有以下附加列：

HOTSPOT_RESULTS

描述在记录周围找到的所有热点的 JSON 字符串。该函数返回所有潜在的热点；您可以在应用程序中筛选出低于特定 `density` 阈值的热点。该字段具有以下节点，每个输入列都有相应的值：

- `density`：热点中的记录数除以热点大小。您可以使用此值来确定热点的相对相关性。
- `maxValues`：每个数据列的热点中记录的最大值。
- `minValues`：每个数据列的热点中记录的最小值。

数据类型：VARCHAR.

> **Note**
>
> 当 Kinesis Data Analytics 服务执行服务维护时，机器学习函数用来确定分析分数的趋势很少会被重置。发生服务维护后，您可能意外地看到分析分数为 0。我们建议您设置筛选条件或其他机制，以便在这些值出现时适当地处理它们。

示例

以下示例在演示流上执行 `HOTSPOTS` 函数，演示流中包含的随机数据不含有意义的热点。举个例子，它执行了`HOTSPOTS`在带有有意义的数据热点的自定义数据流上执行函数，请参阅示例：检测 Hottics.

示例数据集

以下示例基于样本股票数据集，该数据集是开始使用在里面Amazon Kinesis Data Analytics. 要运行该示例，您需要一个具有示例股票行情输入流的 Kinesis Data Analytics 应用程序。要了解如何创建 Kinesis Data Analytics 应用程序和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Data Analytics.

示例股票数据集具有以下架构：

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：返回样本数据流上的热点

在此示例中，将为 HOTSPOTS 函数的输出创建目标流。然后创建一个数据泵，此数据泵对于示例数据流中的指定值运行 HOTSPOTS 函数。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"(
    CHANGE REAL,
    PRICE REAL,
    HOTSPOTS_RESULT VARCHAR(10000));

CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
        SELECT
            "CHANGE",
            "PRICE",
            "HOTSPOTS_RESULT"
        FROM TABLE (
            HOTSPOTS(
```

```
            CURSOR(SELECT STREAM "CHANGE", "PRICE" FROM "SOURCE_SQL_STREAM_001"),
            100,
            0.013,
            20)
    );
```

结果

本示例输出类似于以下内容的流。

Filter by column name

ROWTIME	CHANGE	PRICE	HOTSPOTS_RESULT
2018-03-16 22:43:45.681	-2.14	531.16	{"hotspots":[{"density":1.2859434343255243,"minValues":[-0.9300000071525565,9.659999847412106],"maxValues":[0.6000000238418584,
2018-03-16 22:43:45.681	-0.3	62.19	{"hotspots":[{"density":1.2859434343255243,"minValues":[-0.9300000071525565,9.659999847412106],"maxValues":[0.6000000238418584,
2018-03-16 22:43:45.681	0.83	38.64	{"hotspots":[{"density":1.2859434343255243,"minValues":[-0.9300000071525565,9.659999847412106],"maxValues":[0.6000000238418584,
2018-03-16 22:43:45.681	0.43	57.2	{"hotspots":[{"density":1.2859434343255243,"minValues":[-0.9300000071525565,9.659999847412106],"maxValues":[0.6000000238418584,

RANDOM_CUT_FOREST

检测数据流中的异常。 如果某个记录与其他记录相距较远，则表明该记录是异常的。 要检测各个记录列中的异常情况，请参阅RANDOM_CUT_FOREST_WITH_EXPLANATION (p. 171)。

> **Note**
>
> 这些区域有：RANDOM_CUT_FOREST函数检测异常的能力取决于应用程序。 要投射您的业务问题以便通过此功能解决问题，需要领域的专业知识。例如，确定要将输入流中的哪个列组合传递给函数，并可能对数据进行标准化。有关更多信息，请参阅 输入流 (p. 167)。

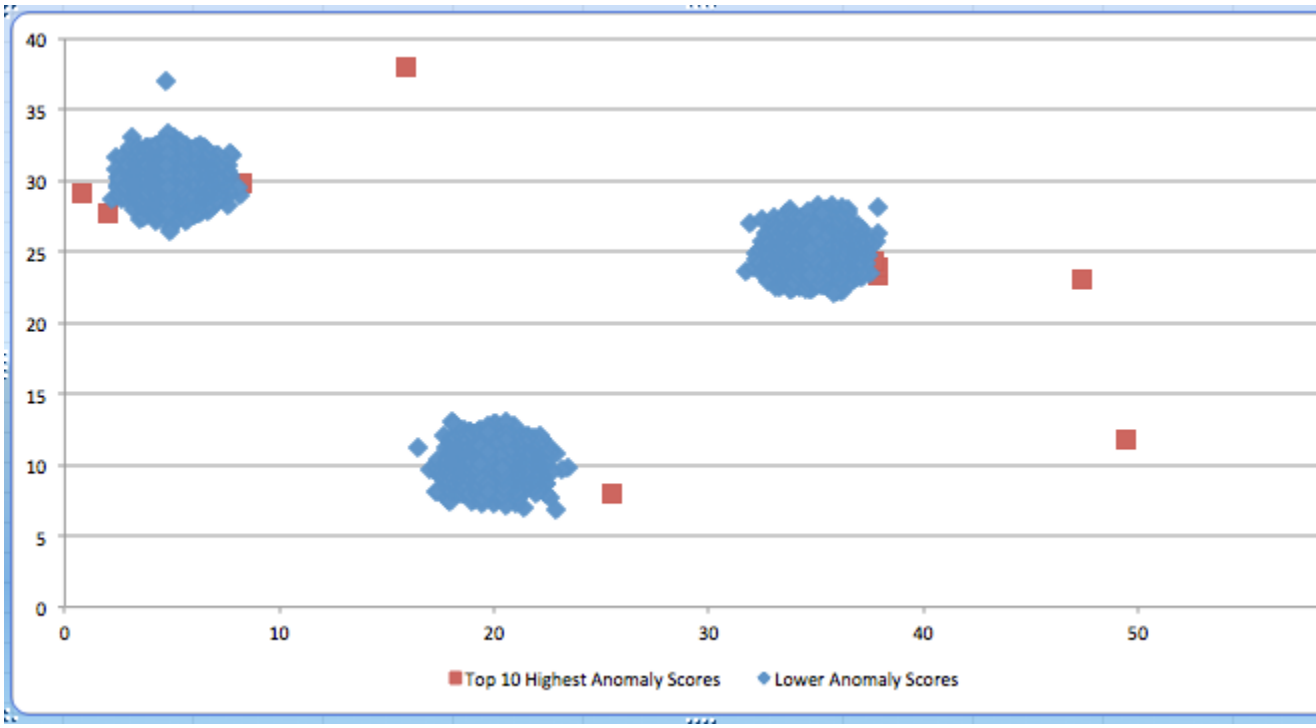
流记录可以有非数字列，但该函数仅使用数字列来分配异常分数。记录可以有一个或多个数字列。该算法使用所有数值数据来计算异常分数。 如果一个记录有 n 个数字列，底层算法将假定每个记录都是 n 维空间中的一个点。n 维空间中与其他点相距较远的点将获得较高的异常分数。

当您启动应用程序时，该算法开始使用流中的当前记录开发机器学习模型。该算法不使用流中较旧的记录进行机器学习，也不使用来自应用程序的之前执行的统计数据。

该算法接受DOUBLE,INTEGER,FLOAT,TINYINT,SMALLINT,REAL，以及BIGINT数据类型。

> **Note**
>
> DECIMAL 不是支持的类型。请改用 “双精度”。

以下是异常检测的示例。 该图显示了随机插入的三个聚类和几个异常。 红色方块显示根据异常分数最高的记录RANDOM_CUT_FOREST函数。 蓝钻代表剩余的记录。 请注意分数最高的记录如何倾向于位于集群外面。



对于示例应用程序与函数 step-by-step 说明，请参阅检测异常.

语法

```
RANDOM_CUT_FOREST (inputStream,
                   numberOfTrees,
                   subSampleSize,
                   timeDecay,
                   shingleSize)
```

参数

以下各节介绍参数。

输入流

指向您的输入流的指针。您可以使用以下命令来设置指针CURSOR函数。例如，以下语句将指针设置为InputStream.

```
CURSOR(SELECT STREAM * FROM InputStream)
CURSOR(SELECT STREAM IntegerColumnX, IntegerColumnY FROM InputStream)
-- Perhaps normalize the column X value.
CURSOR(SELECT STREAM IntegerColumnX / 100, IntegerColumnY FROM InputStream)
-- Combine columns before passing to the function.
CURSOR(SELECT STREAM IntegerColumnX - IntegerColumnY FROM InputStream)
```

这些区域有：CURSOR函数，函数RANDOM_CUT_FOREST函数。函数，假设其他参数的默认值为：

numberOfTrees = 100

subSampleSize = 256

getacCetCetCap

sageCapgeCapgeCap

使用此函数时，您的输入流最多可以有 30 个数字列。

numberOfTrees

使用此参数，您可以指定森林中随机砍伐的树木的数量。

> Note
>
> 默认情况下，该算法会构造许多树，每棵树都是使用给定数量的样本记录构造的（参见`subSampleSize`稍后在此列表中）来自输入流。该算法使用每棵树来分配异常分数。所有这些分数的平均值是最终的异常分数。

的默认值`numberOfTrees`为 100。您可以在 1 和 1,000（含）之间设置此值。通过增加森林中的树木数量，你可以更好地估计异常分数，但这也会增加运行时间。

subSampleSize

使用此参数，您可以指定在构造每棵树时希望算法使用的随机样本的大小。 森林中每棵树都是使用记录的一个（不同的）随机样本构建的。 该算法使用每棵树来分配异常分数。当样本达到 `subSampleSize` 条记录时，会随机删除记录，较旧记录的删除概率高于较新记录。

的默认值`subSampleSize`为 256。 您可以将此值设置为介于 10 和 1,000 之间（含 10 和 1,000）。

请注意，`subSampleSize`必须小于`timeDecay`参数（默认设置为 100,000）。 增大样本大小将为每棵树提供更大的数据视图，但也会延长运行时间。

> Note
>
> 在训练机器学习模型时，算法为首批 `subSampleSize` 个记录返回零。

时间衰减

这些区域有：`timeDecay`参数允许您在计算异常分数时指定要考虑最近的过去。 这是因为数据流会随着时间的推移而自然演变。例如，电子eCommerce 网站的收入可能会持续增加，或者全球气温可能会随着时间的推移而升高。 在这些情况下，我们希望对照较早的数据对最近的数据中的异常进行标记。

默认值为 100,000 条记录（或者，如果使用 singling，则为 100,000 个带状疱疹，如下一节所述）。 您可以在 1 和最大整数（即 2147483647）之间设置此值。该算法以指数方式降低了旧数据的重要性。

如果您选择`timeDecay`默认值为 100,000，异常检测算法执行以下操作：

- 在计算中仅使用最近的 100,000 条记录（并忽略较早的记录）。
- 在最近的 100,000 条记录中，在异常检测计算中，为最近的记录分配的权重呈指数级增加，而为较旧记录分配的权重较小。

如果不想使用默认值，则可以计算要在算法中使用的记录数。为此，请将每天的预期记录数乘以您希望算法考虑的天数。例如，如果您预计每天有 1,000 条记录，并且您希望分析 7 天的记录，请将此参数设置为 7,000 (1,000 * 7)。

这些区域有：timeDecay参数确定在异常检测算法的工作集中保存的最近记录的最大数量。 如果数据改变得很快，则需要较小的 timeDecay 值。 怎样的 timeDecay 值最合适取决于应用程序。

singleSize

此处给出的说明针对的是一维流（即，只有一个数值列的流），但也可用于多维流。

带状疱疹是最近记录的连续序列。 例如，函数shingleSize当时有 10 个t对应于截至并包括时间在内的最近 10 条记录的矢量t. 算法将此序列视为跨最后 shingleSize 个记录的向量。

如果数据以统一的时间到达，则时间 t 处的大小为 10 的瓦形对应于在时间 t-9、t-8、…、t 处收到的数据。在时间 t+1 处，瓦形跨一个单位滑动，且包含来自时间 t-8、t-7、…、t、t+1 的数据。 随着时间的推移收集的这些瓦形记录对应于一个 10 维向量集合，异常检测算法将对该集合运行。

直觉是，瓦片捕捉了最近的形状。 您的数据可能有一个典型的形状。 例如，如果您的数据是每小时收集一次，大小为 24 的瓦形可以捕获您的数据的每日节奏。

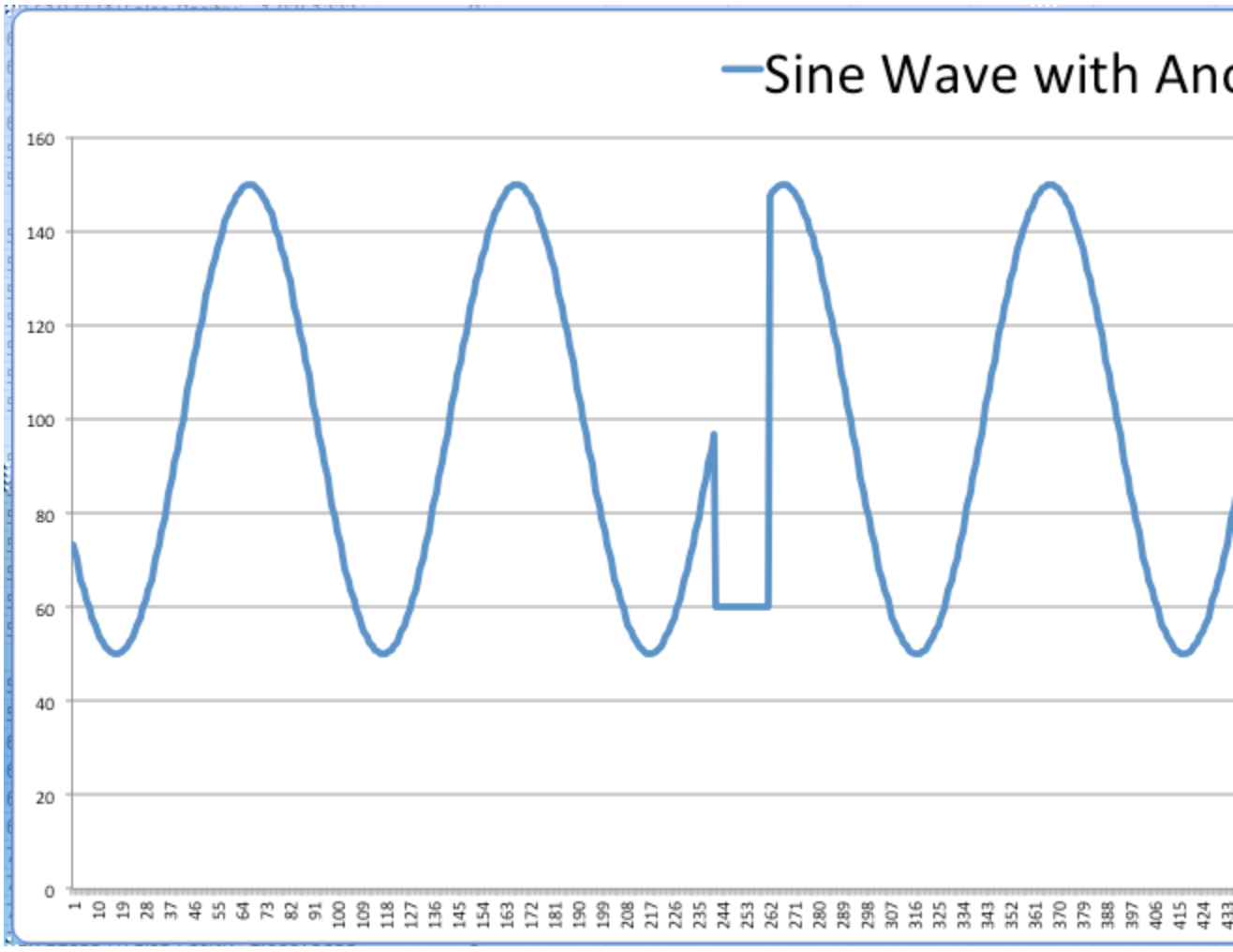
默认值shingleSize是一条记录（因为瓦片大小取决于数据）。您可以在 1 到 30（含）之间设置此值。

请记下有关设置的以下内容shingleSize：

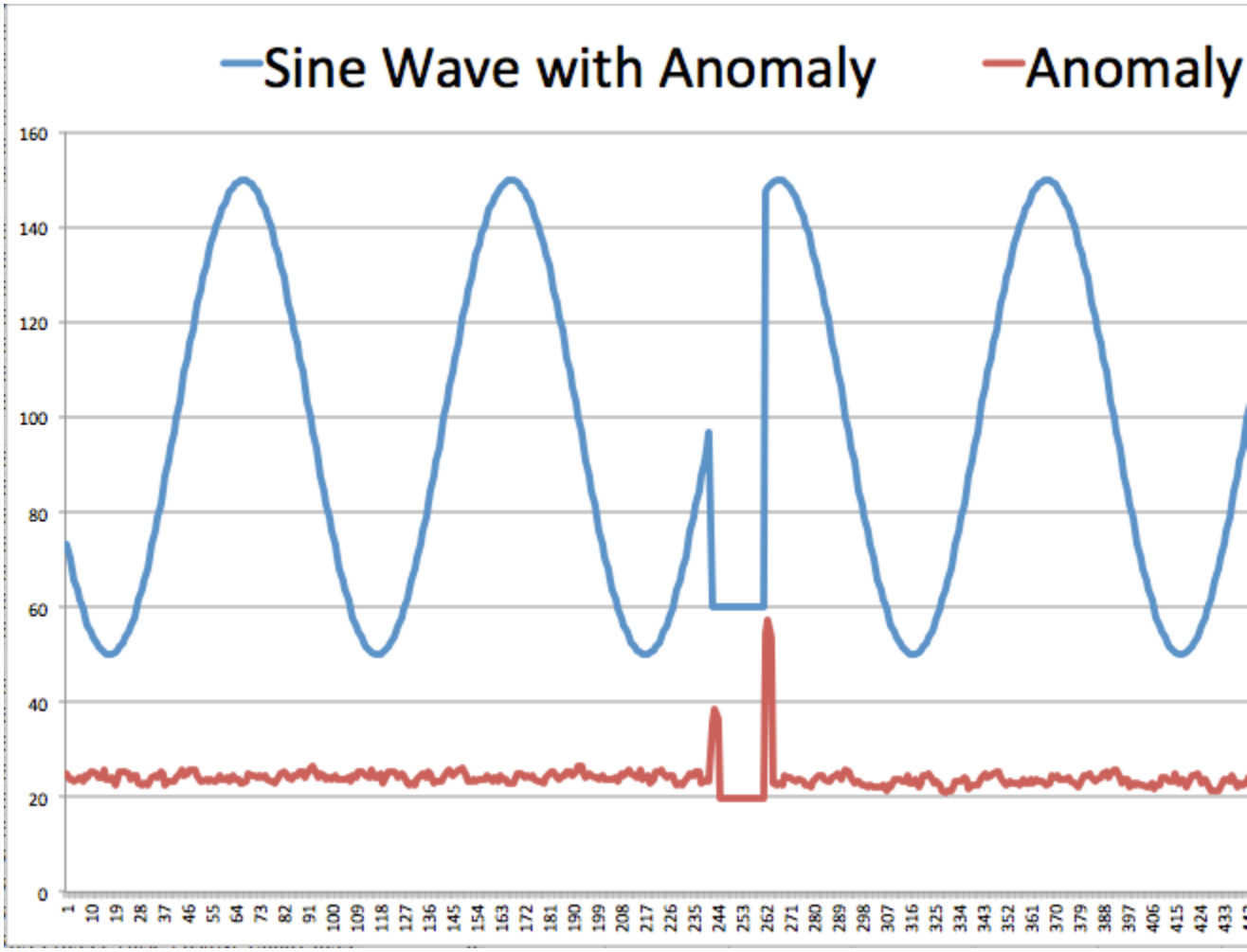
- 如果你设置了shingleSize太小，算法将更容易受到数据微小波动的影响，从而导致非异常记录的异常分数很高。
- 如果你设置了shingleSize太大，检测异常记录可能需要更长时间，因为瓦片中有更多的非异常记录。确定异常情况已结束也可能需要更长时间。
- 确定正确的瓦片大小取决于应用。 请试验不同的瓦形大小以确定影响。

以下示例说明了在监控异常分数最高的记录时如何catch 异常。在这个特殊的例子中，两个最高的异常分数也预示着人工注入异常的开始和结束。

以这个以正弦波表示的程式化一维流为例，旨在捕捉昼夜节律。 此曲线显示了某个电子商务网站每小时收到的订单的典型数量、已登录服务器的用户的数量、每小时收到的广告点击量等。 图的中部人为插入了 20 个连续记录的急剧下降。



我们跑了RANDOM_CUT_FOREST函数，带状大小为四条记录。结果如下所示。 红线显示异常分数。 请注意，异常的开头和结尾获得高分数。



当您使用此函数时，我们建议您将最高分数作为潜在异常情况进行调查。

Note

当 Kinesis Data Analytics 服务执行服务维护时，机器学习函数用来确定分析分数的趋势很少会被重置。发生服务维护后，您可能意外地看到分析分数为 0。我们建议您设置筛选条件或其他机制，以便在这些值出现时适当地处理它们。

有关更多信息，请参阅 Journal of Machine Learning Research 网站上的针对随机砍伐的森林在流上进行可靠的异常情况检测白皮书。

RANDOM_CUT_FOREST_WITH_EXPLANATION

计算异常分数并针对数据流中的每条记录解释此分数。记录的异常分数指示它与最近针对流观察到的趋势有多大的不同。该函数还根据记录中每一列的数据的异常程度，返回对应列的归因分数。对于每条记录，所有列的归因分数总和等于异常分数。

您还可以选择获取有关给定列为异常的方向的信息（相对于对流中给定列最近观察到的数据趋势，该分数是高还是低）。

例如，在电子商务应用中，您可能想知道最近观察到的交易模式何时发生变化。您可能还想知道有多少变化是由于每小时购买次数的变化造成的，有多少是由于每小时放弃的购物车数量的变化造成的，这些信息由

归因分数表示。您可能还需要查看方向性，以了解是否会由于上述每个值的增加或减少而收到有关更改的通知。

> **Note**
>
> 这些区域有：`RANDOM_CUT_FOREST_WITH_EXPLANATION`函数检测异常的能力取决于应用程序。 投射您的业务问题以便通过此功能解决需要领域的专业知识。例如，您可能需要确定输入流中的哪些列组合传递给此函数，并且您可能因对数据规范化而受益。有关更多信息，请参阅 输入流 (p. 173)。

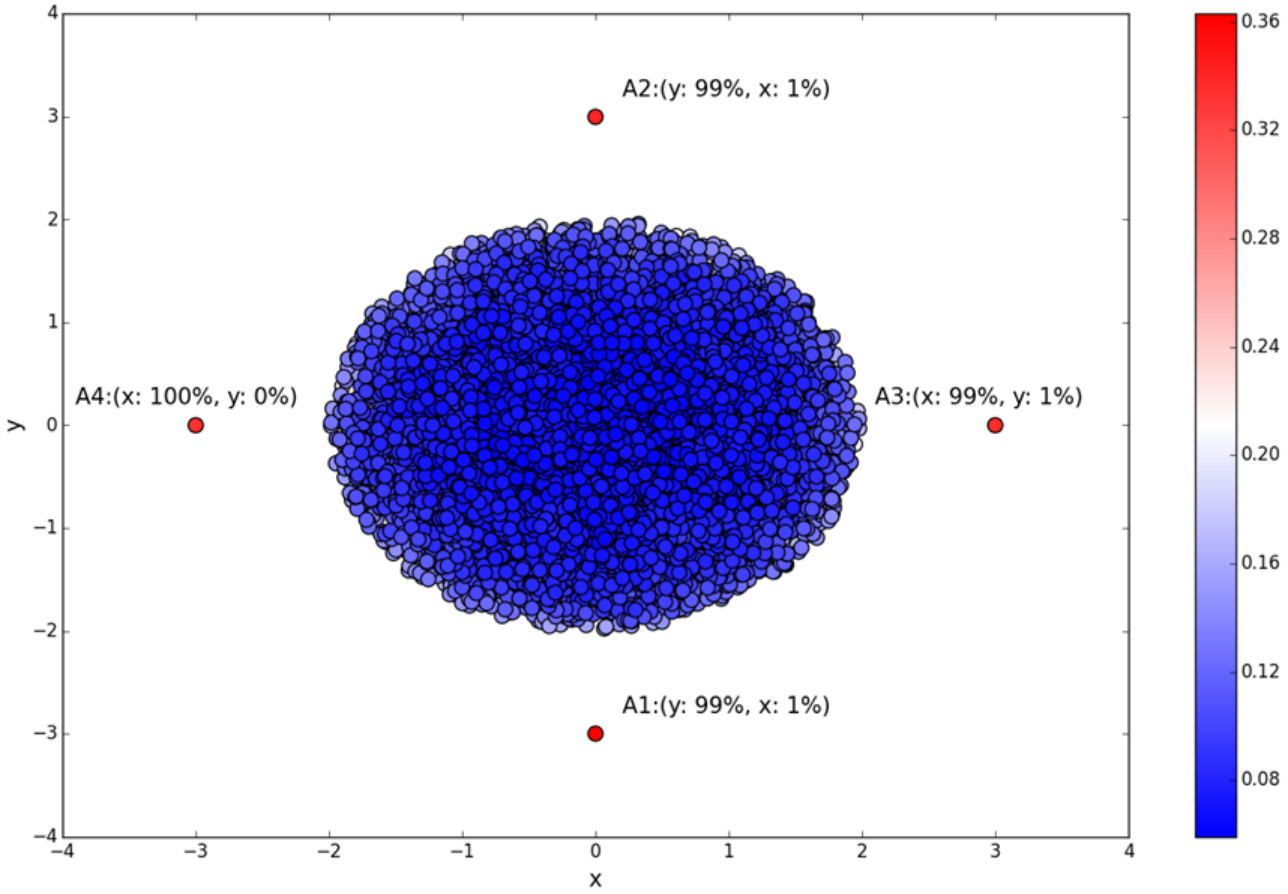
流记录可以有非数字列，但该函数仅使用数字列来分配异常分数。记录可以有一个或多个数字列。算法使用所有数值数据来计算异常分数。

当您启动应用程序时，该算法开始使用流中的当前记录开发机器学习模型。该算法不使用流中较旧的记录进行机器学习，也不使用来自应用程序的之前执行的统计数据。

该算法接受`DOUBLE`,`INTEGER`,`FLOAT`,`TINYINT`,`SMALLINT`,`REAL`，以及`BIGINT`数据类型。

> **Note**
>
> `DECIMAL` 不是受支持的类型。请改用 `DOUBLE`。

以下是在二维空间中使用不同归因分数进行异常检测的简单视觉示例。 该图显示了一组蓝色数据点和四个显示为红点的异常值。 红点有相似的异常分数，但由于不同的原因，这四个分是异常的。对于点 A1 和 A2，异常在极大程度上归因于它们的离心 y 值。对于 A3 和 A4，您可以在极大程度上将异常归因于它们的离心 x 值。方向性为：A1 的 y 值为 LOW（低），A2 的 y 值为 HIGH（高），A3 的 x 值为 HIGH（高），A4 的 x 值为 LOW（低）。



语法

```
RANDOM_CUT_FOREST_WITH_EXPLANATION (inputStream,
                    numberOfTrees,
                    subSampleSize,
                    timeDecay,
                    shingleSize,
                    withDirectionality
)
```

参数

以下部分介绍 RANDOM_CUT_FOREST_WITH_EXPLANATION 函数的参数。

输入流

指向您的输入流的指针。您可以使用以下命令来设置指针CURSOR函数。例如，以下语句将设置指向 InputStream 的指针。

```
CURSOR(SELECT STREAM * FROM InputStream)
CURSOR(SELECT STREAM IntegerColumnX, IntegerColumnY FROM InputStream)
-- Perhaps normalize the column X value.
CURSOR(SELECT STREAM IntegerColumnX / 100, IntegerColumnY FROM InputStream)
-- Combine columns before passing to the function.
CURSOR(SELECT STREAM IntegerColumnX - IntegerColumnY FROM InputStream)
```

这些区域有：CURSOR函数，是唯一需要的参数RANDOM_CUT_FOREST_WITH_EXPLANATION函数。函数假定其他参数的默认值如下：

numberOfTrees = 100

subSampleSize = 256

timeDecay = 100,000

shingleSize = 1

withDirectionality = FALSE

使用此函数时，输入流最多可以有 30 个数值列。

numberOfTrees

使用此参数，您可以指定森林中随机砍伐的树木的数量。

> Note
>
> 默认情况下，该算法会构造许多树，每棵树都是使用给定数量的样本记录构造的（参见subSampleSize稍后在此列表中）来自输入流。该算法使用每棵树来分配异常分数。所有这些分数的平均值是最终的异常分数。

的默认值numberOfTrees为 100。您可以在 1 和 1,000（含）之间设置此值。通过增加森林中树的数量，您可以获得异常分数和归因分数的更准确估算，但这也会延长运行时间。

subSampleSize

使用此参数，您可以指定在构造每棵树时希望算法使用的随机样本的大小。森林中每棵树都是使用记录的一个（不同的）随机样本构建的。该算法使用每棵树来分配异常分数。当样本达到 subSampleSize 条记录时，会随机删除记录，较旧记录的删除概率高于较新记录。

的默认值subSampleSize为 256。您可以将此值设置为介于 10 和 1,000 之间（含 10 和 1,000）。

subSampleSize 必须小于 timeDecay 参数（默认情况下设置为 100,000）。增大样本大小将为每棵树提供更大的数据视图，但它也会延长运行时间。

> Note
>
> 在训练机器学习模型时，算法为首批 subSampleSize 个记录返回零。

时间衰减

您可以使用timeDecay参数用于指定在计算异常分数时要考虑最近的过去。 数据流会随着时间的推移而自然演变。例如，电子商务网站的收入可能会持续增加，或者全球气温可能会随着时间的推移而升高。 在这种情况下，你希望标记相对于最近数据的异常，而不是来自遥远过去的数据。

默认值为 100,000 条记录（或者，如果使用 singling，则为 100,000 个带状疱疹，如下一节所述）。 您可以在 1 和最大整数（即 2147483647）之间设置此值。该算法以指数方式降低了旧数据的重要性。

如果您选择选择timeDecay默认值为 100,000，异常检测算法执行以下操作：

- 在计算中仅使用最近的 100,000 条记录（并忽略较早的记录）。
- 在最近的 100,000 条记录中，在异常检测计算中，为最近的记录分配的权重呈指数级增加，而为较旧记录分配的权重较小。

这些区域有：timeDecay参数确定在异常检测算法的工作集中保存的最近记录的最大数量。 更小timeDecay如果数据变化迅速，则值是理想的。 最好的timeDecay价值取决于应用程序。

singleSize

此处给出的说明适用于一维流（即，只有一个数值列的流），但瓦形也可用于多维流。

带状疱疹是最近记录的连续序列。例如，shingleSize当时是 10 个t对应于截至并包括时间在内的最近 10 条记录的矢量t。 该算法将该序列视为最后一个序列的向量shingleSize记录数。

如果数据按时均匀到达，则一次大小为 10 的瓦片t对应于当时接收到的数据t-9、t-8、...、t。 Timet+1，瓦片滑过一个单元并由来自时的数据组成t-8、t-7、...、t、t+1。 随着时间的推移收集的这些带状记录对应于异常检测算法在其上运行的 10 维向量的集合。

直觉是，瓦片捕捉了最近的形状。 您的数据可能具有典型的形状。 例如，如果您的数据每小时收集一次，则大小为 24 的瓦片可能会捕捉数据的每日节奏。

默认 shingleSize 是一个记录（因为瓦形大小取决于数据）。您可以在 1 到 30（含）之间设置此值。

请注意有关设置的以下内容shingleSize：

- 如果将 shingleSize 设置得过小，算法更容易受数据的细微波动的影响，从而导致并非异常的记录获得高异常分数。
- 如果你设置了shingleSize太大，检测异常记录可能需要更长时间，因为瓦片中有更多的非异常记录。还可能需要更多时间才能确定异常现象是否已结束。

- 确定正确的瓦片大小取决于应用。 尝试不同的瓦片大小以确定效果。

withDirectionality

默认为 `false` 的布尔参数。设置为 `true` 时，它会告诉您每个维度对异常分数的贡献方向。它还提供了针对该方向性的推荐强度。

结果

该函数返回 0 或更高的异常分数，并返回 JSON 格式的解释。

当算法进入学习阶段时，流中所有记录的异常分数从 0 开始。然后，您开始看到异常分数的正值。并非所有正的异常分数都是重要的；只有最高的分数才是重要的。为了更好地理解结果，请查看解释。

该解释为记录中的每个列提供了以下值：

- 归因分数：一个非负数，表示此列对记录的异常分数的贡献程度。换句话说，它表示此列的值与基于最近观察到的趋势的预期值有多大的不同。记录的所有列的归因分数总和等于异常分数。
- Strengt 一个非负数，表示方向推荐的强度。强度值高，表示对函数返回的方向性具有高的置信度。在学习阶段，强度为 0。
- 方向性：如果该列的值高于最近观察到的趋势，则为 HIGH；如果该列的值低于趋势，则为 LOW。在学习阶段，此值默认为 LOW（低）。

> **Note**
>
> 当 Kinesis Data Analytics 服务执行服务维护时，机器学习函数用来确定分析分数的趋势很少会被重置。发生服务维护后，您可能意外地看到分析分数为 0。我们建议您设置筛选条件或其他机制，以便在这些值出现时适当地处理它们。

示例

股票代码数据示例

此示例基于样本股票数据集，该数据集是开始练习在里面Amazon Kinesis Analytics. 要运行该示例，您需要一个具有示例股票行情输入流的 Kinesis Data Analytics 应用程序。要了解如何创建 Kinesis Data Analytics 应用程序和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Analytics.

示例股票数据集具有以下架构：

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

在此示例中，应用程序为记录计算异常分数，并为 `PRICE` 和 `CHANGE` 列计算归因分数，这些是输入流中仅有的数值列。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (anomaly REAL, ANOMALY_EXPLANATION
 VARCHAR(20480));
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
```

```
SELECT "ANOMALY_SCORE", "ANOMALY_EXPLANATION" FROM TABLE
 (RANDOM_CUT_FOREST_WITH_EXPLANATION(CURSOR(SELECT STREAM * FROM "SOURCE_SQL_STREAM_001"),
 100, 256, 100000, 1, true)) WHERE ANOMALY_SCORE > 0
```

上一示例输出的流与以下内容类似。

Filter by column name

ROWTIME	ANOMALY	ANOMALY_EXPLANATION
2017-10-23 20:09:21.466	0.57327205	{"CHANGE":{"DIRECTION":"LOW","STRENGTH":"0.4824","ATTRIBUTION_SCORE":"0.3
2017-10-23 20:09:21.466	0.73723656	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.3178","ATTRIBUTION_SCORE":"0.3
2017-10-23 20:09:21.466	0.6443901	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.1731","ATTRIBUTION_SCORE":"0.3
2017-10-23 20:09:21.466	0.55428815	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.0888","ATTRIBUTION_SCORE":"0.2
2017-10-23 20:09:21.466	0.5416738	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.0316","ATTRIBUTION_SCORE":"0.2
2017-10-23 20:09:21.466	0.67421293	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.3364","ATTRIBUTION_SCORE":"0.3
2017-10-23 20:09:21.466	0.6528528	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.2020","ATTRIBUTION_SCORE":"0.3
2017-10-23 20:09:21.466	0.83734107	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.0177","ATTRIBUTION_SCORE":"0.2
2017-10-23 20:09:21.466	0.9346224	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.2364","ATTRIBUTION_SCORE":"0.2
2017-10-23 20:09:21.466	0.6726167	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.0397","ATTRIBUTION_SCORE":"0.3
2017-10-23 20:09:21.466	0.6636287	{"CHANGE":{"DIRECTION":"LOW","STRENGTH":"0.0571","ATTRIBUTION_SCORE":"0.3
2017-10-23 20:09:21.466	0.7353514	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.1851","ATTRIBUTION_SCORE":"0.1

网络和 CPU 利用率示例

本理论示例显示了两组遵循振荡模式的数据。在下图中，它们由顶部的红色曲线和蓝色曲线表示。红色曲线显示随时间推移的网络利用率，蓝色曲线显示同一计算机系统随时间推移的闲置 CPU。这两个彼此有相位差的信号在大多数时间是有规律的。但它们都显示偶尔的异常，这些异常在图表中显示为不规则。下面解释图形中的曲线所表示的内容（按从顶部曲线到底部曲线的顺序）。

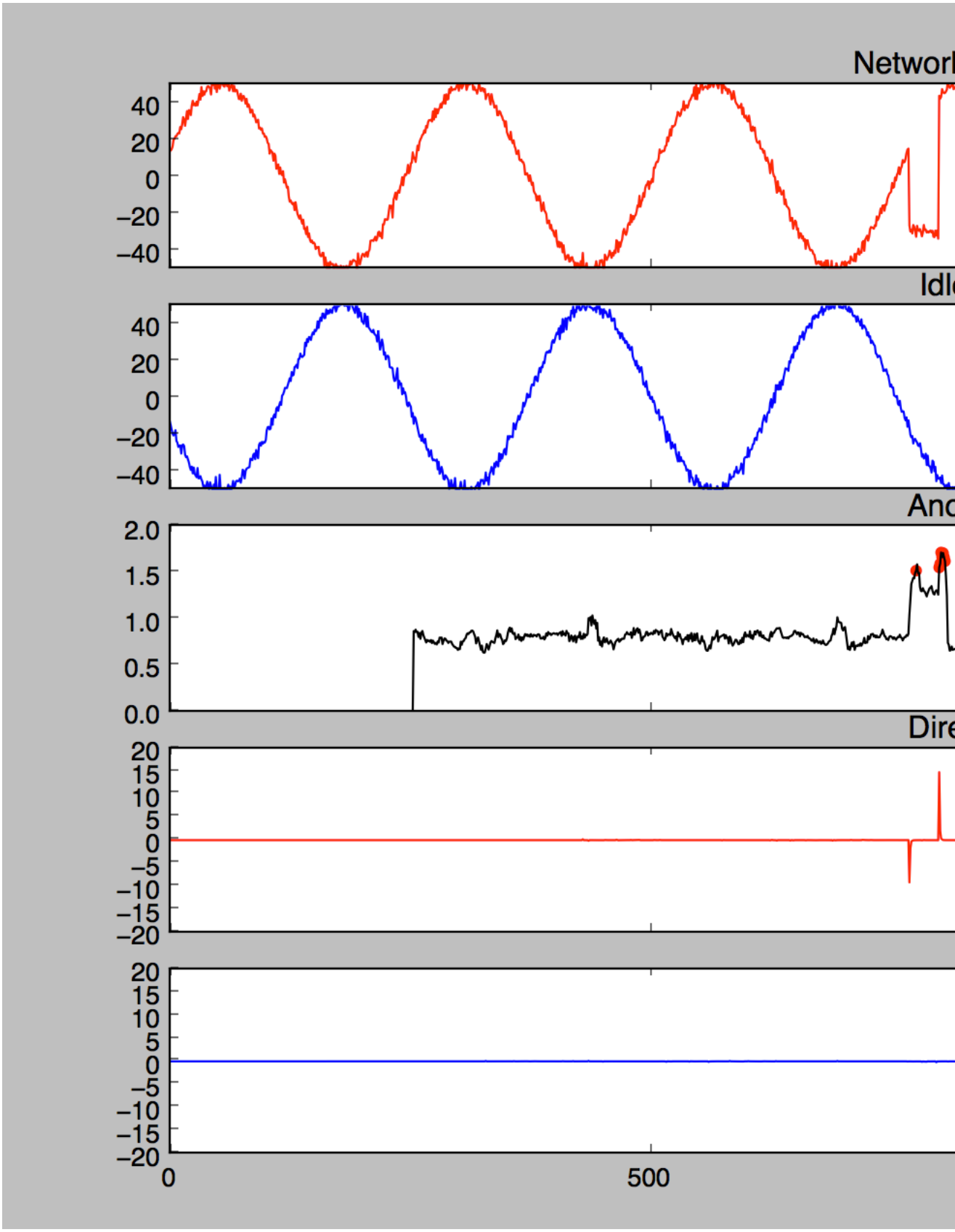
- 顶部曲线为红色，表示随时间推移的网络利用率。它遵循循环模式，大部分时间是有规律的，除了两个异常期间（每个期间都表示利用率下降）之外。第一个异常期间发生在时间值 500 到 1,000 之间。第二个异常期间发生在时间值 1,500 到 2,000 之间。
- 顶部的第二条曲线（蓝色）是随着时间推移的空闲 CPU。它遵循循环模式，大部分时间是有规律的，除了两个异常期间之外。第一个异常期间发生在时间值 1,000 到 1,500 之间，并显示空闲 CPU 时间下降。第二个异常期间发生在时间值 1,500到 2,000 之间，并显示空闲 CPU 时间增加。
- 顶部的第三条曲线显示异常分数。在开始时，有一个学习阶段，此时异常分数为 0。在学习阶段之后，曲线上有稳定的噪声，但异常很突出。

  第一个异常在黑色异常分数曲线上以红色标记，它更多地归因于网络利用率数据。第二个异常标记为蓝色，它更多地归因于 CPU 数据。此图中提供红色和蓝色标记以更好地显示效果。它们不是由 `RANDOM_CUT_FOREST_WITH_EXPLANATION` 函数生成的。以下是获得这些红色和蓝色标记的方法：
  - 运行函数后，我们选择了前 20 个异常分数值。
  - 从这个前 20 个异常分数值的集合中，我们选择其网络利用率归因大于或等于 CPU 归因的 1.5 倍的那些值。我们在图中使用红色标记来给此新值集中的点着色。
  - 我们使用蓝色标记对其 CPU 归因分数大于或等于网络利用率归因分数的 1.5 倍的点进行着色。
- 底部的第二条曲线是网络利用率信号方向性的图形表示。我们通过以下方法获得此曲线：运行此函数，将强度乘以 -1 以表示 LOW（低）方向性，乘以 +1 以表示 HIGH（高）方向性，并根据时间绘制结果。

当网络利用率的周期性模式下降时，方向性会出现相应的负峰值。当网络利用率显示回升到常规模式时，方向性显示与该升高幅度相对应的正峰值。后来，出现了另一个负峰值，紧接着是另一个正峰值。它们共同表示网络利用率曲线中出现的第二个异常。

- 底部曲线是 CPU 信号方向性的图形表示。我们通过以下方法获得此曲线：将强度乘以 -1 以表示 LOW（低）方向性，乘以 +1 以表示 HIGH（高）方向性，并根据时间绘制结果。

  对于空闲 CPU 曲线中的第一个异常，此方向性曲线会显示一个负峰值，随后立即出现一个较小的正峰值。空闲 CPU 曲线中的第二个异常产生正峰值，然后产生一个方向性负峰值。


Networl
40
20
0
−20
−40
Idl
40
20
0
−20
−40
And
2.0
1.5
1.0
0.5
0.0
Dire
20
15
10
5
0
−5
−10
−15
−20
20
15
10
5
0
−5
−10
−15
−20
0
500

血压示例

有关更详细的示例，其中包含检测和解释血压读数异常的代码，请参阅示例：检测数据异常并获得解释.

STDDEV_POP

返回为组中其余每行计算的 <number expression> 的 VAR_POP (p. 184) 总体方差的平方根。

在使用 STDDEV_POP 时，请注意以下事项：

- 当输入集没有非 null 数据时，STDDEV_POP 将返回 NULL。
- 如果你不使用OVER子句，STDDEV_POP以聚合函数的形式计算。在这种情况下，聚合查询必须包含GROUP BY 子句 (p. 55)在基于以下条件的单调表达式上ROWTIME它将流分组为有限的行。否则，该组是无限流，查询将永远不会完成，也不会发出任何行。有关更多信息，请参阅 聚合函数 (p. 69)。
- 一个窗口式查询，它使用GROUP BY子句处理滚动窗口中的行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您将OVER子句，STDDEV_POP以分析函数的形式计算。有关更多信息，请参阅 分析函数 (p. 97)。
- 一个窗口式查询，它使用OVER子句处理滑动窗口中的行。有关更多信息，请参阅滑动窗口

语法

```
STDDEV_POP ( [DISTINCT | ALL] number-expression )
```

参数

ALL

在输入集中包含重复值。ALL 是默认值。

DISTINCT

在输入集中排除重复值。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是开始使用在里面Amazon Kinesis Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics App 和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Analytics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：在翻滚窗口查询中确定列中总体的标准差

以下示例演示如何使用 STDDEV_POP 函数来确定示例数据集的 PRICE 列的滚动窗口中的值的标准差。未指定 DISTINCT，因此计算中将包含重复值。

使用 STEP（推荐）

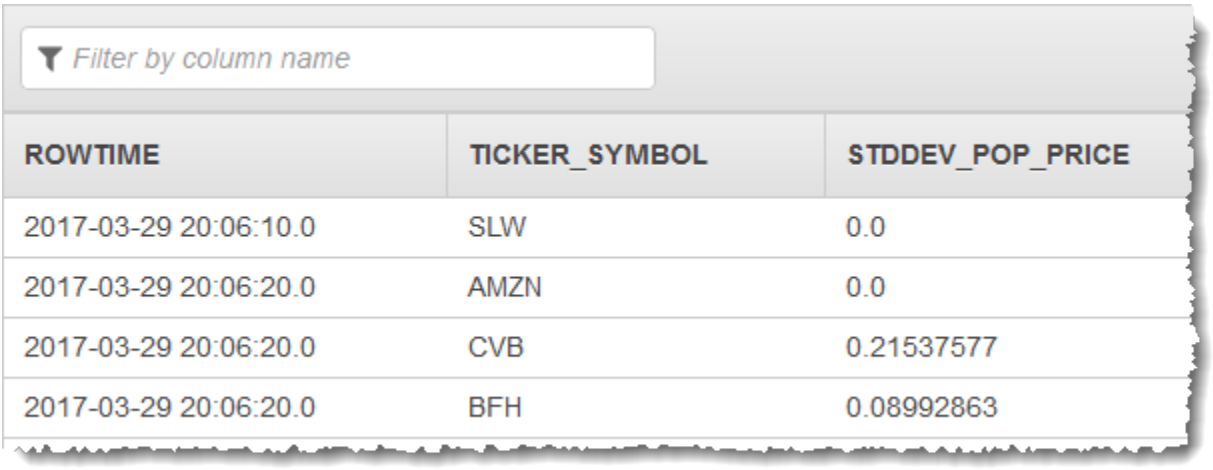
```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 stddev_pop_price REAL);

CREATE OR REPLACE  PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, STDDEV_POP(price) AS stddev_pop_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, STEP(("SOURCE_SQL_STREAM_001".ROWTIME) BY INTERVAL '60'
 SECOND);
```

使用 FLOOR

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 stddev_pop_price REAL);

CREATE OR REPLACE  PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, STDDEV_POP(price) AS stddev_pop_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR(("SOURCE_SQL_STREAM_001".ROWTIME - TIMESTAMP '1970-01-01
 00:00:00') SECOND / 10 TO SECOND);
```

结果

上一示例输出的流与以下内容类似：



示例 2：确定滑动窗口查询中列中值的总体标准差

以下示例演示如何使用 STDDEV_POP 函数来确定示例数据集的 PRICE 列的滑动窗口中的值的标准差。未指定 DISTINCT，因此计算中将包含重复值。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 stddev_pop_price REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, STDDEV_POP(price) OVER TEN_SECOND_SLIDING_WINDOW AS
 stddev_pop_price
FROM "SOURCE_SQL_STREAM_001"
```

```
WINDOW TEN_SECOND_SLIDING_WINDOW AS (
  PARTITION BY ticker_symbol
  RANGE INTERVAL '10' SECOND PRECEDING);
```

上一示例输出的流与以下内容类似：

Filter by column name

ROWTIME	TICKER_SYMBOL	STDDEV_POP_PRICE
2017-03-29 20:09:11.957	UHN	11.155467
2017-03-29 20:09:11.957	RFV	0.8669914
2017-03-29 20:09:11.957	PJN	0.6344981
2017-03-29 20:09:11.957	BNM	2.418462

另请参阅

- 样本标准差: STDDEV_SAMP (p. 181)
- 样本方差: VAR_SAMP (p. 186)
- 总体方差: VAR_POP (p. 184)

STDDEV_SAMP

以 <number-expression> 形式返回所有值的统计标准差，统计标准差针对组中余下的每一行进行计算并定义为 VAR_SAMP (p. 186) 的平方根。

在使用 `STDDEV_SAMP` 时，请注意以下事项：

- 当输入集没有非 null 数据时，`STDDEV_SAMP` 将返回 `NULL`。
- 如果你不使用`OVER`子句，`STDDEV_SAMP`以聚合函数的形式计算。在这种情况下，聚合查询必须包含GROUP BY 子句 (p. 55)在基于以下条件的单调表达式上`ROWTIME`它将流分组为有限的行。否则，该组是无限流，查询将永远不会完成，也不会发出任何行。有关更多信息，请参阅 聚合函数 (p. 69)。
- 使用 GROUP BY 子句的窗口式查询处理滚动窗口中的行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您将`OVER`子句，`STDDEV_SAMP`以分析函数的形式计算。有关更多信息，请参阅 分析函数 (p. 97)。
- 使用 OVER 子句的窗口式查询处理滑动窗口中的行。有关更多信息，请参阅滑动窗口
- `STD_DEV` 是 `STDDEV_SAMP` 的别名。

语法

```
STDDEV_SAMP ( [DISTINCT | ALL] number-expression )
```

参数

ALL

在输入集中包含重复值。ALL 是默认值。

DISTINCT

在输入集中排除重复值。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是开始使用在里面Amazon Kinesis Aletics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建Analytics应用程序和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Aletics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：确定滚动窗口查询中列中值的统计标准差

以下示例演示如何使用 STDDEV_SAMP 函数来确定示例数据集的 PRICE 列的滚动窗口中的值的标准差。未指定 DISTINCT，因此计算中将包含重复值。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 stddev_samp_price REAL);

CREATE OR REPLACE  PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, STDDEV_SAMP(price) AS stddev_samp_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR(("SOURCE_SQL_STREAM_001".ROWTIME - TIMESTAMP '1970-01-01
 00:00:00') SECOND / 10 TO SECOND);
```

结果

上一示例输出的流与以下内容类似：

Filter by column name

ROWTIME	TICKER_SYMBOL	STDDEV_SAMP_PRICE
2017-03-29 22:23:30.0	AMZN	7.6294384
2017-03-29 22:23:30.0	WSB	54.68945
2017-03-29 22:23:30.0	JKL	0.08468548
2017-03-29 22:23:30.0	QXZ	68.81256

示例 2：确定滑动窗口查询中列中值的统计标准差

以下示例演示如何使用 STDDEV_SAMP 函数来确定示例数据集的 PRICE 列的滑动窗口中的值的标准差。未指定 DISTINCT，因此计算中将包含重复值。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 stddev_samp_price REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, STDDEV_SAMP(price) OVER TEN_SECOND_SLIDING_WINDOW AS
 stddev_samp_price
FROM "SOURCE_SQL_STREAM_001"

WINDOW TEN_SECOND_SLIDING_WINDOW AS (
  PARTITION BY ticker_symbol
  RANGE INTERVAL '10' SECOND PRECEDING);
```

上一示例输出的流与以下内容类似：

Filter by column name

ROWTIME	TICKER_SYMBOL	STDDEV_SAMP_PRICE
2017-03-29 20:02:58.683	SAC	
2017-03-29 20:03:06.692	TGT	1.4704113
2017-03-29 20:03:06.692	QAZ	1.8484228
2017-03-29 20:03:06.692	KIN	0.41638768

另请参阅

- 总体标准差: STDDEV_POP (p. 179)
- 样本方差: VAR_SAMP (p. 186)
- 总体方差: VAR_POP (p. 184)

VAR_POP

返回一组非空数字的总体方差（忽略空值）

VAR_POP 使用以下计算方法：

- (SUM (expr*expr)-总和 (expr) *SUM (expr) /COUNT (expr)) /COUNT (expr)

换句话说，对于一组给定的非空值，使用 S1 作为值的总和，使用 S2 作为值的平方和，VAR_POP 返回结果 (S2-S1*S1/N) /N。

在使用 VAR_POP 时，请注意以下事项：

- 当输入集没有非 null 数据或应用于空集时，VAR_POP 将返回 NULL。
- 如果你不使用OVER子句，VAR_POP以聚合函数的形式计算。在这种情况下，聚合查询必须包含GROUP BY 子句 (p. 55)在基于以下条件的单调表达式上ROWTIME它将流分组为有限的行。否则，该组是无限流，查询将永远不会完成，也不会发出任何行。有关更多信息，请参阅 聚合函数 (p. 69)。
- 一个窗口式查询，它使用GROUP BY子句处理滚动窗口中的行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您将OVER子句，VAR_POP以分析函数的形式计算。有关更多信息，请参阅 分析函数 (p. 97)。
- 一个窗口式查询，它使用OVER子句处理滑动窗口中的行。有关更多信息，请参阅滑动窗口

语法

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

参数

ALL

在输入集中包含重复值。ALL 是默认值。

DISTINCT

在输入集中排除重复值。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是开始使用在里面Amazon Kinesis Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics App 和配置示例股票行情输入流，请参阅开始使用在里面Amazon Kinesis Analytics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：在翻滚窗口查询中确定列中的总体方差

以下示例演示如何使用 VARPOP 函数来确定示例数据集的 PRICE 列的滚动窗口中的值的总体方差。未指定 DISTINCT，因此计算中将包含重复值。

使用 STEP（推荐）

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4), var_pop_price
 REAL);

CREATE OR REPLACE  PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, VAR_POP(price) AS var_pop_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, STEP(("SOURCE_SQL_STREAM_001".ROWTIME) BY INTERVAL '60'
 SECOND);
```

使用 FLOOR

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4), var_pop_price
 REAL);

CREATE OR REPLACE  PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, VAR_POP(price) AS var_pop_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR(("SOURCE_SQL_STREAM_001".ROWTIME - TIMESTAMP '1970-01-01
 00:00:00') SECOND / 10 TO SECOND);
```

结果

上一示例输出的流与以下内容类似：

Filter by column name

ROWTIME	TICKER_SYMBOL	VAR_POP_PRICE
2017-03-29 22:21:40.275	BNM	0.0
2017-03-29 22:21:45.29	PJN	0.0
2017-03-29 22:21:45.29	MJN	0.0
2017-03-29 22:21:45.29	PPL	0.0

示例 2：确定滑动窗口查询中列中值的总体方差

以下示例演示如何使用 VARPOP 函数来确定示例数据集的 PRICE 列的滑动窗口中的值的总体方差。未指定 DISTINCT，因此计算中将包含重复值。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4), var_pop_price
 REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, VAR_POP(price) OVER TEN_SECOND_SLIDING_WINDOW AS var_pop_price
FROM "SOURCE_SQL_STREAM_001"

WINDOW TEN_SECOND_SLIDING_WINDOW AS (
  PARTITION BY ticker_symbol
  RANGE INTERVAL '10' SECOND PRECEDING);
```

上一示例输出的流与以下内容类似：

Filter by column name

ROWTIME	TICKER_SYMBOL	VAR_POP_PRICE
2017-03-29 20:26:00.258	QXZ	0.0
2017-03-29 20:26:00.258	ALY	0.0
2017-03-29 20:26:00.258	AZL	0.0
2017-03-29 20:26:00.258	BNM	0.0

另请参阅

- 总体标准差: STDDEV_POP (p. 179)
- 样本标准差: STDDEV_SAMP (p. 181)
- 样本方差: VAR_SAMP (p. 186)

VAR_SAMP

返回数字的非 null 集的样本方差 (null 值被忽略)。

VAR_SAMP 使用以下计算方法：

- (SUM (expr*expr)-总和 (expr) *SUM (expr) /COUNT (expr))/(COUNT (expr) -1)

换言之，对于一组给定的非 null 值，使用 S1 作为值的和，使用 S2 作为值的平方和，VAR_SAMP 会返回结果 (S2-S1*S1/N)/(N-1)。

在使用 VAR_SAMP 时，请注意以下事项：

- 当输入集没有非 null 数据时，VAR_SAMP 将返回 NULL。如果提供为 null 或包含一个元素的输入集，则 VAR_SAMP 将返回 null。
- 如果你不使用OVER子句，VAR_SAMP以聚合函数的形式计算。在这种情况下，聚合查询必须包含GROUP BY 子句 (p. 55)在基于以下条件的单调表达式上ROWTIME它将流分组为有限的行。否则，该组是无限流，查询将永远不会完成，也不会发出任何行。有关更多信息，请参阅 聚合函数 (p. 69)。
- 一个窗口式查询，它使用GROUP BY子句处理滚动窗口中的行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您将OVER子句，VAR_SAMP以分析函数的形式计算。有关更多信息，请参阅 分析函数 (p. 97)。
- 一个窗口式查询，它使用OVER子句处理滑动窗口中的行。有关更多信息，请参阅滑动窗口

语法

```
VAR_SAMP ( [DISTINCT | ALL] number-expression )
```

参数

ALL

在输入集中包含重复值。ALL 是默认值。

DISTINCT

在输入集中排除重复值。

示例

示例数据集

以下示例基于样本股票数据集，该数据集是开始使用在Amazon Kinesis Analytics. 要运行每个示例，您需要一个具有示例股票行情输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics App 和配置示例股票行情输入流，请参阅开始使用在Amazon Kinesis Analytics.

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：在滚动窗口查询中确定列中的样本方差

以下示例演示如何使用 VAR_SAMP 函数来确定示例数据集的 PRICE 列的滚动窗口中的值的样本方差。未指定 DISTINCT，因此计算中将包含重复值。

使用 STEP（推荐）

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4), var_samp_price
 REAL);

CREATE OR REPLACE  PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
```

```
SELECT STREAM ticker_symbol, VAR_SAMP(price) AS var_samp_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, STEP(("SOURCE_SQL_STREAM_001".ROWTIME) BY INTERVAL '60'
 SECOND);
```

使用 FLOOR

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4), var_samp_price
 REAL);

CREATE OR REPLACE  PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, VAR_SAMP(price) AS var_samp_price
    FROM "SOURCE_SQL_STREAM_001"
    GROUP BY ticker_symbol, FLOOR(("SOURCE_SQL_STREAM_001".ROWTIME - TIMESTAMP '1970-01-01
 00:00:00') SECOND / 10 TO SECOND);
```

结果

上一示例输出的流与以下内容类似：

Filter by column name

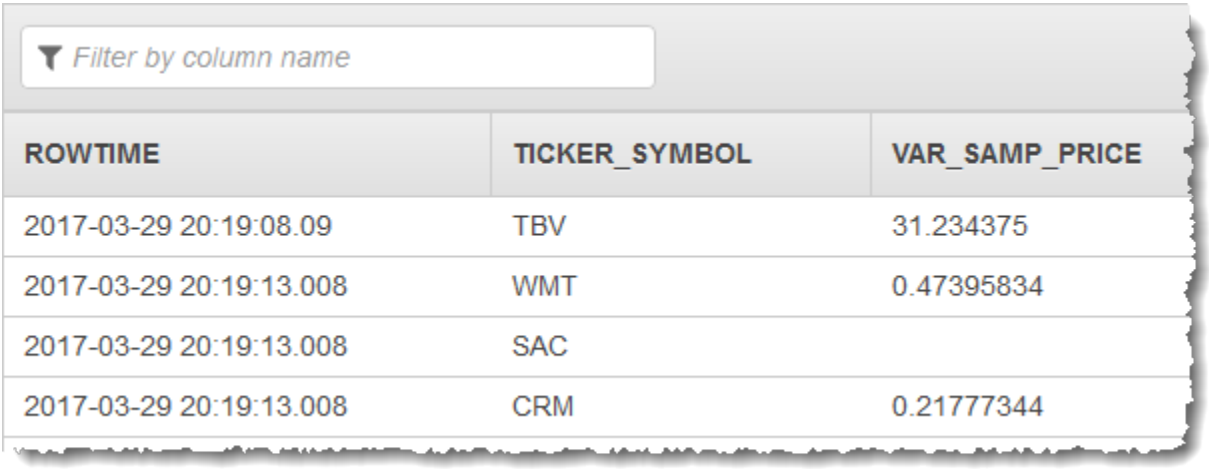
ROWTIME	TICKER_SYMBOL	VAR_SAMP_PRICE
2017-03-29 20:16:30.0	DEG	0.3784485
2017-03-29 20:16:40.0	WMT	
2017-03-29 20:16:40.0	QXZ	12260.502
2017-03-29 20:16:40.0	NFLX	

示例 2：确定滑动窗口查询中列中值的样本方差

以下示例演示如何使用 VAR_SAMP 函数来确定示例数据集的 PRICE 列的滑动窗口中的值的样本方差。未指定 DISTINCT，因此计算中将包含重复值。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4), var_samp_price
 REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM ticker_symbol, VAR_SAMP(price) OVER TEN_SECOND_SLIDING_WINDOW AS
 var_samp_price
FROM "SOURCE_SQL_STREAM_001"

WINDOW TEN_SECOND_SLIDING_WINDOW AS (
  PARTITION BY ticker_symbol
  RANGE INTERVAL '10' SECOND PRECEDING);
```

上一示例输出的流与以下内容类似：


Filter by column name

ROWTIME	TICKER_SYMBOL	VAR_SAMP_PRICE
2017-03-29 20:19:08.09	TBV	31.234375
2017-03-29 20:19:13.008	WMT	0.47395834
2017-03-29 20:19:13.008	SAC	
2017-03-29 20:19:13.008	CRM	0.21777344

另请参阅

- 总体标准差: STDDEV_POP (p. 179)
- 样本标准差: STDDEV_SAMP (p. 181)
- 总体方差:VAR_POP (p. 184)

流式处理 SQL 函数

本节中的主题描述了 Amazon Kinesis Data Analytics 流式处理 SQL 的流式处理函数。

主题

- LAG (p. 189)
- 单调函数 (p. 191)
- NTH_VALUE (p. 192)

LAG

LAG 返回对给定窗口中当前记录之前 N 条记录的表达式（例如列名）的求值。偏移量和默认值都是根据当前记录计算的。如果没有这样的记录，LAG 会改为返回指定的默认表达式。LAG 返回与表达式的类型相同的值。

语法

```
LAG(expr [ , N [ , defaultExpr]]) [ IGNORE NULLS | RESPECT NULLS ] OVER [ window-
definition ]
```

参数

expr

根据记录计算的表达式。

否

要查询的当前记录之前的记录数。默认为 1。

defaultExpr

与相同类型的表达式expr如果查询记录，则返回该值 (否在当前记录之前) 落在窗口之外。如果未指定，则为位于窗口外部的值返回 null。

Note

defaultExpr 表达式不会替换从源流返回的实际 null 值。

IGNORE NULLS

一个子句，它指定在确定偏移量时不计入空值。例如，假设查询 `LAG(expr, 1)`，并且上一条记录具有为 null 的 expr 值。随后，将查询之前的第二条记录，依此类推。

RESPECT NULLS

一个子句，它指定在确定偏移量时计入空值。此行为是默认行为。

OVER window-specification

一种子句，用于将流中的记录除以时间范围间隔或记录数分区。窗口规范定义流中记录的划分方式 (按时间范围间隔或记录数)。

示例

示例数据集

以下示例基于示例股票数据集，后者是《Amazon Kinesis Analytics 开发人员指南》 中的入门练习的一部分。要运行每个示例，您需要一个 Amazon Kinesis Analytics 应用程序，该应用程序具有示例股票行情的输入流。要了解如何创建 Analytics 应用程序和配置示例股票代码输入流，请参阅《Amazon Kinesis Analytics 开发人员指南》 中的入门练习。

具有以下架构的示例股票数据集。

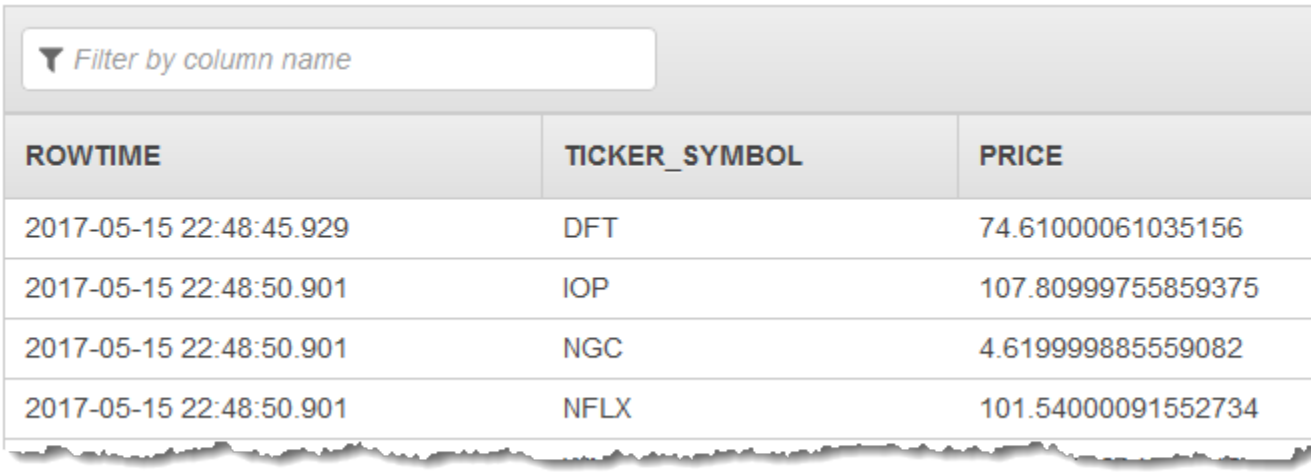
```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：返回 OVER 子句中先前记录的值

在此示例中，OVER 子句将流中的记录除以前 '1' 分钟的时间范围间隔。随后，LAG 函数从包含给定股票代码的前 2 条记录中检索价格值，如果 `price` 为 null，则跳过记录。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
    ticker_symbol VARCHAR(4),
    price     DOUBLE,
    previous_price DOUBLE,
    previous_price_2 DOUBLE);
CREATE OR REPLACE PUMP "STREAM_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
    SELECT STREAM ticker_symbol,
         price,
         LAG(price, 1, 0) IGNORE NULLS OVER (
            PARTITION BY ticker_symbol
            RANGE INTERVAL '1' MINUTE PRECEDING),
         LAG(price, 2, 0) IGNORE NULLS OVER (
            PARTITION BY ticker_symbol
            RANGE INTERVAL '1' MINUTE PRECEDING)
    FROM "SOURCE_SQL_STREAM_001"
```

上一示例输出的流与以下内容类似。


Filter by column name

ROWTIME	TICKER_SYMBOL	PRICE
2017-05-15 22:48:45.929	DFT	74.61000061035156
2017-05-15 22:48:50.901	IOP	107.80999755859375
2017-05-15 22:48:50.901	NGC	4.619999885559082
2017-05-15 22:48:50.901	NFLX	101.54000091552734

注意

LAG 不是 SQL: 2008 标准的一部分。它是 Amazon Kinesis Data Analytics Streamics

单调函数

```
MONOTONIC(<expression>)
```

Streaming GROUP BY 要求分组表达式中至少有一个是单调且不变的。事先已知的唯一单调列是 ROWTIME。有关更多信息，请参阅 单调表达式和运算符 (p. 19)。

MONOTONIC 函数允许您声明给定表达式为单调表达式，从而允许流式传输 GROUP BY 使用该表达式作为键。

MONOTONIC 函数计算其参数并返回结果（与其参数的类型相同）。

通过在 MONOTONIC 中包含一个表达式，就是在断言该表达式的值要么是非递增的，要么是非递减的，并且从不改变方向。例如，如果你有一个由订单行项目组成的 LINEITEMS 流，并且你写了 MONOTONIC (orderID)，那么你就是在断言订单流中单列项目是连续的。如果订单 1000 有订单项目，其次是订单 1001 的行项目，然后是订单 1005 的行项目，那就没问题了。如果当时订单 1001 有单项商品（也就是说，行项目序列变成 1000、1001、1005、1001），那将是非法的。同样，987、974、823 的行项目序列是合法的，但以下行项目序列将是非法的：

- 987、974、823、973
- 987、974、823、1056

声明为单调的表达式可以减小，甚至具有任意顺序。

请注意，MONOTONIC 的定义正是 GROUP BY 取得进展所需要的。

如果声明为单调的表达式不是单调表达式（即，如果断言对实际数据无效），则 Amazon Kinesis Data Analytics 行为是未指定的。

换句话说，如果您确定某个表达式是单调的，则可以使用此 MONOTONIC 函数让 Amazon Kinesis Data Analytics 将该表达式视为单调表达式。

但是，如果您错了，并且计算表达式所得的值从升序变为降序或从降序变为升序，则可能会出现意想不到的结果。Amazon Kinesis Data Analytics Streaming SQL 将让你信守诺言，并保证表达式是单调的。但是，如

果事实上它不是单调的，则无法事先确定由此产生的 Amazon Kinesis Data Analytics 行为，因此结果可能不符合预期或预期。

NTH_VALUE

```
NTH_VALUE(x, n) [ <from first or last> ] [ <null treatment> ] over w
```

其中：

<null treatment> := RESPECT NULLS | IGNORE NULL

<from first or last> := FROM FIRST | FROM LAST

NTH_VALUE 返回 x 从窗口中第一个或最后一个值开始的第 n 个值。默认为第一。如果 <null treatment> 设置为 IGNORE NULLS，则函数将在计数时跳过 null。

如果窗口中的行数不足以达到第 n 个值，则该函数返回 NULL。

字符串和搜索函数

本节中的主题介绍了 Amazon Kinesis Data Analytics 直播 SQL 的字符串和搜索函数。

主题

CHAR_LENGTH/字符长度

```
CHAR_LENGTH | CHARACTER_LENGTH ( <character-expression> )
```

返回作为输入参数传递的字符串的长度（以字符为单位）。如果输入参数为空，则返回空值。

示例

`CHAR_LENGTH('one')`	3
`CHAR_LENGTH('')`	0
`CHARACTER_LENGTH('fred')`	4

`CHARACTER_LENGTH( cast (null as varchar(16) )`	`null`
`CHARACTER_LENGTH( cast ('fred' as char(16) )`	`16`

限制

Amazon Kinesis Data Analytics 直播 SQL 不支持可选的 “使用字符 | OCTETS” 子句。这与 SQL: 2008 标准背道而驰。

INITCAP

```
INITCAP ( <character-expression> )
```

返回输入字符串的转换版本，即每个以空格分隔的单词的第一个字符为大写，而所有其他字符均为小写。

示例

函数	结果
INITCAP ('每一个都是第一个 IEtTeR 是大写的 ')	每个首字母均为大写

> Note
>
> INITCAP 函数不是 SQL: 2008 标准的一部分。它是Amazon Kinesis Data Analytics 的扩展程序。

LOWER

```
LOWER ( <character-expression> )
```

将字符串转换为全部小写字符。如果输入参数为 null，则返回 null；如果输入参数是空字符串，则返回空字符串。

示例

函数	结果
LOWER ('abcdefghi123')	abcdefghi123

OVERLAY

```
OVERLAY ( <original-string>
          PLACING <replacement-string>
          FROM <start-position>
          [ FOR <string-length> ]
```

```
        )
<original-string> := <character-expression>
<replacement-string> := <character-expression>
<start-position> := <integer-expression>
<string-length> := <integer-expression>
```

OVERLAY 函数用于将第一个字符串参数（原始字符串）的一部分替换为第二个字符串参数（替换字符串）。

起始位置表示原始字符串中的字符位置，替换字符串应在此重叠的位置。可选的字符串长度参数确定要替换的原始字符串的多少个字符（如果未指定，则默认为替换字符串的长度）。如果替换字符串中的字符多于原始字符串中剩余的字符，则只需附加剩余字符即可。

如果起始位置大于原始字符串的长度，则只需附加替换字符串即可。如果起始位置小于 1，则替换字符串的（1-起始位置）字符优先于结果，其余字符叠加在原始字符串上（参见以下示例）。

如果字符串长度小于零，则引发异常。

如果任何输入参数为 null，则结果为 null。

示例

函数	结果
叠加（'12345' 放置 1 中的 'foo'）	foo45
叠加（'12345' 从 0 放置 'foo'）	foo345
叠加（'12345' 从 -2 放置 'foo'）	foo12345
叠加（'12345' 放置 4 中的 'foo'）	123foo
叠加（'12345' 放置 17 中的 'foo'）	12345foo
叠加（'12345' 将 2 中的 'foo' 置入 0）	1foo2345
叠加（'12345' 将来自 2 的 'foo' 置入 2）	1foo45
叠加（'12345' 将 2 中的 'foo' 置入 9）	1foo

限制

Amazon Kinesis Data Analytics 不支持 SQL: 2008 中定义的可选 “使用字符 | OCTETS” 子句；只是假设使用字符。严格的 SQL: 2008 还要求小于 1 的起始位置返回空结果，而不是上述行为。这些偏离了标准。

POSITION

```
POSITION ( <search-string> IN <source-string> )
search-string := <character-expression>
source-string := <character-expression>
```

POSITION 函数在第二个输入参数（源字符串）中搜索第一个输入参数（搜索字符串）。

如果在源字符串中找到搜索字符串，POSITION 将返回搜索字符串的第一个实例的字符位置（忽略后续实例）。如果搜索字符串是空字符串，则 POSITION 返回 1。

如果未找到搜索字符串，则 POSITION 返回 0。

如果搜索字符串或源字符串为空，则 POSITION 返回空值。

示例

函数	结果
位置 ('1234FindmexXX'中的 “找我”)	5
位置 (在 “1234not-Herexxx” 中找我)	0
位置 ('1234567' 中的 '1')	1
位置 ('1234567' 中的 '7')	7
位置 ('1234567'中的 “”)	1

限制

Amazon Kinesis Data Analytics 直播 SQL 不支持 SQL: 2008 中定义的可选 USING CHARACTERS | OCTETS 子句；只是假设使用字符。这与标准背道而驰。

REGEX_REPLACE

REGEX_REPLACE 用备用子字符串替换子字符串。它返回以下 Java 表达式的值。

```
java.lang.String.replaceAll(regex, replacement)
```

语法

```
REGEX_REPLACE(original VARCHAR(65535), regex VARCHAR(65535), replacement VARCHAR(65535),
 startPosition int, occurence int)

RETURNS VARCHAR(65535)
```

参数

original

要对其执行正则表达式操作的字符串。

regex

要匹配的正则表达式。如果 regex 的编码与 original 的编码不匹配，则将向错误流写入错误。

replacement

要替换的字符串正则表达式匹配项原版的字符串。如果 replacement 的编码与 original 或 regex 的编码不匹配，则将向错误流写入错误。

startPosition

要搜索的 original 字符串中的第一个字符。如果 startPosition 小于 1，则将向错误流写入错误。如果 startPosition 大于 original 的长度，则将返回 original。

occurence

要替换的与 regex 表达式匹配的字符串数量。如果 occurence 为 0，则将替换所有匹配 regex 的子字符串。如果 occurence 小于 0，则将向错误流写入错误。

示例

示例数据集

以下示例基于示例股票数据集，后者是《Amazon Kinesis Analytics 开发人员指南》 中的入门练习的一部分。

要运行每个示例，您需要一个 Amazon Kinesis Analytics 应用程序，该应用程序具有示例股票行情的输入流。要了解如何创建 Analytics 应用程序和配置示例股票代码输入流，请参阅《Amazon Kinesis Analytics 开发人员指南》 中的入门练习。

具有以下架构的示例股票数据集。

```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
price           REAL)
```

示例 1：用新值替换源字符串中的所有字符串值

在此示例中，如果 `sector` 字段中的所有字符串都与正则表达式匹配，则它们都将被替换。

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
        ticker_symbol VARCHAR(4),
        SECTOR VARCHAR(24),
        CHANGE REAL,
        PRICE REAL);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"

SELECT STREAM   TICKER_SYMBOL,
                REGEX_REPLACE(SECTOR, 'TECHNOLOGY', 'INFORMATION TECHNOLOGY', 1, 0);
                CHANGE,
                PRICE
FROM "SOURCE_SQL_STREAM_001"
```

上一示例输出的流与以下内容类似。

Filter by column name

ROWTIME	TICKER_SYMBOL	SECTOR	CHANG
2017-05-16 22:30:39.464	MMB	ENERGY	0.72
2017-05-16 22:30:39.464	TGT	RETAIL	2.26
2017-05-16 22:30:39.464	CVB	INFORMATION TECHNOLOGY	0.06
2017-05-16 22:30:39.464	PJN	RETAIL	-0.09

注意

REGEX_REPLACE 不是 SQL: 2008 标准的一部分。它是Amazon Kinesis Data Analytics SQL 扩展。

如果所有参数都为 `null`，则 REGEX_REPLACE 返回 `null`。

SUBSTRING

```
SUBSTRING ( <source-string> FROM <start-position> [ FOR <string-length> ] )
SUBSTRING ( <source-string>, <start-position> [ , <string-length> ] )
SUBSTRING ( <source-string> SIMILAR <pattern> ESCAPE <escape-char> )
<source-string> := <character-expression>
<start-position> := <integer-expression>
<string-length> := <integer-expression>
<regex-expression> := <character-expression>
<pattern> := <character-expression>
<escape-char> := <character-expression>
```

SUBSTRING 提取第一个参数中指定的源字符串的一部分。提取从 start-position 的值或第一个与 regex-expression 的值匹配的表达式开始。

如果为 string-length 指定一个值，则仅返回该数量的字符。如果字符串中剩余的字符不多，则只返回剩下的字符。如果未指定 string-length，该字符串长度默认为输入字符串的剩余长度。

如果起始位置小于 1，则将起始位置解释为 1，字符串长度将减去 (1-起始位置)。有关示例，请参阅以下内容。如果起始位置大于字符串中的字符数，或长度参数为 0，则结果为空字符串。

参数

source-string

用于搜索位置或正则表达式匹配项的字符串。

start-position

要返回的 source-string 的第一个字符。如果 start-position 大于 source-string 的长度，则 SUBSTRING 返回 null。

string-length

要返回的 source-string 的字符数。

regex-expression

要匹配并从 source-string 返回的字符模式。仅返回第一个匹配项。

pattern

包含以下内容的三部分字符模式：

- 要在返回的子字符串之前查找的字符串
- 返回的子字符串
- 要在返回的子字符串之后查找的字符串

各部分由双引号 (“) 和指定的转义字符分隔。有关更多信息，请参阅以下Similar...Escape (p. 198)示例。

示例

FROM/ FOR

函数	结果
子字符串（从 3 到 4 的 '123456789'）	3456
子字符串（来自 17 的 '123456789' 对 4）	<empty string>
子字符串（来自 -1 的 '123456789' 代表 4）	12
子字符串（'123456789' 来自 6 代表 0）	<empty string>
子字符串（'123456789' 来自 8 对 4）	89

FROM Regex

函数	结果
SUBSTRING('TECHNOLOGY' FROM 'L[A-Z]*')	LOGY
SUBSTRING('TECHNOLOGY' FROM 'FOO')	null
SUBSTRING('TECHNOLOGY' FROM 'O[A-Z]')	OL

数值

函数	结果
子字符串 ('123456789'、3、4)	3456
SUBSTRING('123456789', 7, 4)	789
SUBSTRING('123456789', 10, 4)	null

Similar...Escape

函数	结果
SUBSTRING('123456789' SIMILAR '23#"456#"78' ESCAPE '#')	456
SUBSTRING('TECHNOLOGY' SIMILAR 'TECH%"NOLO%"GY' ESCAPE '%')	NOLO

注意

- Amazon Kinesis Data Analytics 直播 SQL 不支持 SQL: 2008 中定义的可选 “使用字符 | 八位字节” 子句。只采用 USING CHARACTERS。
- 前面列出的 SUBSTRING 函数的第二和第三种形式（使用正则表达式，使用逗号而不是 FROM... FOR）不是 SQL: 2008 标准的一部分。它们是Amazon Kinesis Data Analytics 直播 SQL 扩展的一部分。

TRIM

```
TRIM ( [ [ <trim-specification> ] [ <trim-character> ] FROM ] <trim-source> )
 <trim-specification> := LEADING | TRAILING | BOTH
 <trim-character> := <character-expression>
 <trim-source> := <character-expression>
```

根据修剪规范（即，前导、尾部或两者），TRIM 从修剪源字符串的开头和/或结尾移除指定修剪字符的实例。如果指定了 LEADING，则只删除源字符串开头处重复的修剪字符。如果指定了 Trailing，则只删除源字符串末尾重复的修剪字符。如果指定了 BOTH，或者完全省略了 trim 说明符，则会删除源字符串开头和结尾的重复内容。

如果未明确指定修剪字符，则默认为空格字符 (')。只允许使用一个修剪字符；指定空字符串或长度超过一个字符的字符串会导致异常。

如果任一输入为 null，则返回 null。

示例

函数	结果
`TRIM(' Trim front and back ')`	`'Trim front and back'`
`TRIM (BOTH FROM ' Trim front and back ')`	`'Trim front and back'`
`TRIM (BOTH ' ' FROM ' Trim front and back ')`	`'Trim front and back'`
`TRIM (LEADING 'x' FROM 'xxxTrim frontxxx')`	`'Trim frontxxx'`
`TRIM (TRAILING 'x' FROM 'xxxTrimxBackxxx')`	`'xxxTrimxBack'`
`TRIM (BOTH 'y' FROM 'xxxNo y to trimxxx')`	`'xxxNo y to trimxxx'`

UPPER

```
< UPPER ( <character-expression> )
```

将字符串转换为全部大写字符。如果输入参数为 null，则返回 null；如果输入参数是空字符串，则返回空字符串。

示例

函数	结果
UPPER ('abcdefghi123')	ABCDEFGHI123

Kinesis Data Analytics

有关开发 Kinesis Data Analytics 应用程序的信息，请参阅Kinesis Data Analytics.

文档历史记录

下表描述了自 Amazon Kinesis Data Analytics SQL 参考资料以来对文档所做的重要更改。

- API 版本：2015 年 8 月 14 日
- 上次文档更新日期：2018 年 3 月 19 日

更改	说明	日期
新的 HOTSPOTS 函数	查找和返回有关数据中相对密集的区域的信息。有关更多信息，请参阅 HOTSPOTS (p. 163)。	2018 年 3 月 19 日
Stream-to-stream JOIN 示例	JOIN 子句查询的示例。有关更多信息，请参阅 加入子句 (p. 41)。	2018 年 2 月 28 日
新的 TSDIFF 函数	获取两个时间戳之间的差异。有关更多信息，请参阅 TSDIFF (p. 135)。	2017 年 12 月 11 日
新的 RANDOM_CUT_FOREST_WITH_EXPLANATION 函数	了解在数据流中哪些字段会产生异常评分。有关更多信息，请参阅 RANDOM_CUT_FOREST_WITH_EXPLANATION (p. 171)。	2017 年 11 月 2 日
新的 REGEX_LOG_PARSE 函数	从列式源字符串中获取正则表达式匹配项。有关更多信息，请参阅 REGEX_LOG_PARSE (p. 149)。	2017 年 8 月 21 日
目录重组	主题类别现在更加直观。	2017 年 8 月 18 日
新的 SQL 函数	添加 STEP (p. 141)、LAG (p. 189)、TO_TIMESTAMP (p. 121)、UNIX_TIMESTAMP (p. 1 以及添加对 SUBSTRING (p. 197) 的正则表达式支持	2017 年 8 月 3 日
新指南	这是的第一个版本Amazon Kinesis Data Aletic指南。	2016 年 8 月 11 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。