aws

SQL 参考

Amazon Kinesis Data Analytics SQL 参考

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Kinesis Data Analytics SQL 参考: SQL 参考

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

SQL 参考 1
流式处理 SQL 语言元素 2
标识符 2
数据类型 3
数字类型和精度 7
流式 SQL 运算符 9
IN 运算符 10
EXISTS 运算符 10
标量运算符 10
算术运算符 11
字符串运算符 12
逻辑运算符 20
表达式和文字 26
单调表达式和运算符 29
单调列 30
单调表达式 30
推断单调性的规则 31
条件子句 32
时间谓词 33
语法 35
示例 36
示例使用案例 36
保留字和关键字 37
标准 SQL 运算符 45
CREATE 语句 45
CREATE STREAM 45
CREATE FUNCTION 46
CREATE PUMP 48
INSERT 49
语法 49
数据泵流插入 49
Query 50
语法 50
选择 50

流式集合运算符 51
VALUES 运算符 52
SELECT 语句 53
语法 53
STREAM 关键字和流式 SQL 的原理 53
SELECT ALL 和 SELECT DISTINCT 54
SELECT 子句 55
FROM 子句 58
JOIN 子句 61
HAVING 子句 77
GROUP BY 子句 78
WHERE 子句 80
WINDOW 子句 (滑动窗口) 81
ORDER BY 子句 90
ROWTIME 92
函数 95
聚合函数 95
流式聚合和行时间边界 96
聚合函数列表 97
对流执行的聚合查询的示例（流式聚合） 98
流的窗口式聚合 101
AVG 107
COUNT 111
COUNT_DISTINCT_ITEMS_TUMBLING 函数 115
EXP_AVG 118
FIRST_VALUE 119
LAST_VALUE 119
MAX 120
MIN 123
SUM 127
TOP_K_ITEMS_TUMBLING 函数 130
分析函数 133
相关主题 134
布尔函数 134
ANY 135
EVERY 135

转换函数 135
CAST 136
日期和时间函数 156
时区 157
日期时间转换函数 157
日期、时间戳和间隔运算符 175
日期和时间模式 183
CURRENT_DATE 187
CURRENT_ROW_TIMESTAMP 188
CURRENT_TIME 188
CURRENT_TIMESTAMP 189
EXTRACT 189
LOCALTIME 191
LOCALTIMESTAMP 191
TSDIFF 192
Null 函数 192
COALESCE 192
NULLIF 193
数字函数 194
ABS 194
CEIL/CEILING 195
EXP 196
FLOOR 197
LN 198
LOG10 198
MOD 199
POWER 200
STEP 200
日志解析函数 204
FAST_REGEX_LOG_PARSER 205
FIXED_COLUMN_LOG_PARSE 210
REGEX_LOG_PARSE 211
SYS_LOG_PARSE 214
VARIABLE_COLUMN_LOG_PARSE 214
W3C_LOG_PARSE 215
排序函数 226

Group Rank 226
统计方差和偏差函数 230
HOTSPOTS 232
RANDOM_CUT_FOREST 236
RANDOM_CUT_FOREST_WITH_EXPLANATION 241
STDDEV_POP 249
STDDEV_SAMP 253
VAR_POP 256
VAR_SAMP 260
流式处理 SQL 函数 263
LAG 263
单调函数 266
NTH_VALUE 267
字符串和搜索函数 267
CHAR_LENGTH/CHARACTER_LENGTH 267
INITCAP 268
LOWER 269
OVERLAY 269
POSITION 270
REGEX_REPLACE 271
SUBSTRING 274
TRIM 276
UPPER 277
Kinesis Data Analytics 开发人员指南 279
文档历史记录 280
..... cclxxxii

Amazon Kinesis Data Analytics SQL 参考

> **Note**
>
> 对于新项目，建议您使用 Kinesis Data Analytics Studio，而不是 Kinesis Data Analytics for SQL 应用程序。Kinesis Data Analytics Studio 不仅操作简单，还具有高级分析功能，使您能够在几分钟内构建复杂的流处理应用程序。

有关 Amazon Kinesis Data Analytics 支持的 SQL 语言元素的信息，请参阅 Amazon Kinesis Data Analytics SQL 参考。该语言以 SQL:2008 标准为基础，有一些扩展可以对流数据进行操作。

有关开发 Kinesis Data Analytics 应用程序的信息，请参阅 Kinesis Data Analytics 开发人员指南。

本指南涵盖以下内容：

- 流式处理 SQL 语言元素 – 数据类型, 流式 SQL 运算符, 函数.
- 标准 SQL 运算符 – CREATE 语句, SELECT 语句.
- 用于转换和筛选传入数据的运算符 – WHERE 子句、JOIN 子句、GROUP BY 子句、WINDOW 子句 (滑动窗口)。
- 逻辑运算符 – AS、AND、OR 等。

流式处理 SQL 语言元素

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

主题

- 标识符
- 数据类型
- 流式 SQL 运算符
- 表达式和文字
- 单调表达式和运算符
- 条件子句
- 时间谓词
- 保留字和关键字

标识符

所有标识符最多可以有 128 个字符。标识符可以通过用双引号 (") 括起来进行引用（区分大小写），也可以不加引号（在存储和查找之前都使用隐式大写）。

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

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 字符都接受：代码 0x0000 到 0x001F）。您可以在标识符中加入双引号，方法是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了一个流，该标识符在流定义存储到目录中之前会转换为大写。可以使用其大写名称来引用它，也可以使用隐式转换为大写的未加引号的标识符来引用它。

```
-- 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，其名称会隐式引用，因此可以轻松创建包含小写字符、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，则需要引用其名称。

保留字和关键字

对于某些标识符（称为关键字），如果出现在流式 SQL 语句中的特定位置，则具有特殊含义。这些关键字的子集称为保留字，除非被引用，否则不能用作对象的名称。有关更多信息，请参阅 保留字和关键字。

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	说明	注意
BIGINT	数字	64 位有符号整数	
BINARY	BASE64-encoded 字符串	二进制（非字符）数据	子字符串适用于 BINARY。联接对 BINARY 不起作用。
BOOLEAN	布尔值	TRUE、FALSE 或 NULL	计算结果为 TRUE、FALSE 和 UNKNOWN。
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。

SQL 数据类型	JSON 数据类型	说明	注意
DATE	字符串	日期是日历日（year/month/天）。	精度是日。范围介于最大值 [大约 +229（以年为单位）] 到最小值 -229 之间。
DECIMAL DEC NUMERIC	数字	一个固定点，最多包含 19 位有效数字。	可以用 DECIMAL、DEC 或 NUMERIC 指定。
DOUBLE DOUBLE PRECISION	数字	64 位浮点数	64 位近似值；-1.79E+308 至 1.79E+308。采用 ISO DOUBLE PRECISION 数据类型，53 位用于科学计数法中的数字尾数，表示 15 位数的精度和 8 字节的存储空间。
INTEGER INT	数字		32 位带符号的整数。范围介于 -2147483648 到 2147483647 [2**(31) to 2**(31)- 1] 之间
INTERVAL <timeunit> [TO <timeunit>]	字符串	Day-time 支持间隔，不支持年月间隔	在采用日期算法的表达式中允许，但不可用作表或流中列的数据类型。
<timeUnit>	字符串	INTERVAL 值的单位	支持的单位为 YEAR、MONTH、DAY、HOUR、MINUTE 和 SECOND

SQL 数据类型	JSON 数据类型	说明	注意
SMALLINT	数字	16 位有符号整数	范围介于 -32768 到 32767 之间 [2**(15) to 2**(15)-1]
REAL	数字	32 位浮点数	采用 ISO REAL 数据类型，24 位用于科学计数法中的数字尾数，表示 7 位数的精度和 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	TIME 是一天中的时间 (hour:minute:second)。	其精度是毫秒；其范围是 00:00:00.000 到 23:59:59.999。由于系统时钟采用 UTC，因此不考虑用于 TIME 或 TIMESTAMP 列中存储的值的时区。 用于 TIME 或 TIMESTAMP 列中存储的值。

SQL 数据类型	JSON 数据类型	说明	注意
TIMESTAMP	字符串	TIMESTAMP 是 DATE 和 TIME 的组合。	TIMESTAMP 值的精度始终为 1 毫秒。没有特定的时区。由于系统时钟采用 UTC，因此不考虑用于 TIME 或 TIMESTAMP 列中存储的值的时区。范围介于最大值 [大约 +229（以年为单位）] 到最小值 -229 之间。每个时间戳都存储为带符号的 64 位整数，其中 0 表示 Unix 时代 (1970 年 1 月 1 日零点)。这意味着最大的 TIMESTAMP 值代表 1970 年之后的大约 3 亿年，最小的值代表 1970 年之前的大约 3 亿年。根据 SQL 标准，TIMESTAMP 值的时区未定义。
TINYINT	数字	8 位有符号整数	范围介于 -128 到 127 之间
VARBINARY (n)	BASE64-encoded 字符串	也可指定为 BINARY VARYING	n 必须大于 0 且小于 65535。
VARCHAR (n)	字符串	也可指定为 CHARACTER VARYING	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，当且仅当字符取自相同的字符库并且是数据类型为 CHARACTER 或 CHARACTER VARYING 的值时，字符才可以相互分配。

关于数字：

- 如果数字是数据类型为 NUMERIC、DECIMAL、INTEGER、BIGINT、SMALLINT、TINYINT、REAL 和 DOUBLE PRECISION 的值，则可以相互比较和相互分配。

以下数据类型集是同义词：

- DEC 和 DECIMAL
- DOUBLE PRECISION 和 DOUBLE
- CHARACTER 和 CHAR
- CHAR VARYING 或 CHARACTER VARYING 和 VARCHAR
- BINARY VARYING 和 VARBINARY
- INT 和 INTEGER

- 二进制值（数据类型为 BINARY 和 BINARY VARYING）始终可以相互比较，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，只有当分配的源和目标都是 DATE、TIME 或 TIMESTAMP 类型时，日期时间值才可以相互分配）。

- Amazon Kinesis Data Analytics 时区始终采用 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，我们最多支持 18 位数的精度和小数位数。

精度指定列中可存储的最大小数位数，包括小数点右侧和左侧的小数位数。您可以指定从 1 位到 18 位的精度，也可以使用 18 位的默认精度。

小数位数指定小数点右侧可存储的最大位数。小数位数必须小于或等于精度。您可以指定介于 0 到 18 位之间的小数位数，也可以使用 0 位的默认小数位数。

除法规则

假设 p1、s1 是第一个运算对象的精度和小数位数，例如 DECIMAL (10,1)。

假设 p2、s2 是第二个运算对象的精度和小数位数，例如 DECIMAL (10,3)。

假设 p、s 是结果的精度和小数位数。

假设 d 为结果中的整数位数。然后，结果类型为小数，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) 结果的小数位数 = 14
p = d + s	结果的精度 = 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)，然后再加上足够的整数来表示结果而不会溢出

乘法规则

假设 p1、s1 是第一个运算对象 DECIMAL (10,1) 的精度和小数位数。

假设 p2、s2 是第二个运算对象 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，结果类型为小数，如下所示：

p = p1 + p2	p = 10 + 10

	结果的精度 = 18
s = s1 + s2	s = 1 + 3 结果的小数位数 = 4

求和或减法规则

Type-inference 策略中，调用的结果类型是两个精确数字操作数的十进制和，其中至少一个操作数是十进制。

假设 p1、s1 是第一个运算对象 DECIMAL (10,1) 的精度和小数位数。

假设 p2、s2 是第二个运算对象 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) 结果的小数位数 = 3
p = max(p1 - s1, p2 - s2) + s + 1	p = max(10-1,10-3) + 3 + 1 结果的精度 = 11

s 和 p 的上限为其最大值

流式 SQL 运算符

子查询运算符

在查询和子查询中使用运算符来组合或测试各种属性或关系的数据。

以下主题将介绍可用的运算符，分为以下几类：

- 标量运算符
  - 运算符类型
  - 优先级

- 算术运算符
- 字符串运算符
  - （联接）
  - LIKE 模式
  - SIMILAR TO 模式
- 日期、时间戳和间隔运算符
- 逻辑运算符
  - 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、关系表达式的计算结果或子查询返回的行中找到要测试的值，则返回 TRUE；否则返回 FALSE。

> **Note**
>
> IN 具有不同的含义且用在 CREATE FUNCTION 中。

EXISTS 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：一元运算符只能对一个运算对象进行运算。一元运算符通常以下列格式对其运算对象进行运算：

```
operator operand
```

- 二进制：二进制运算符对两个运算对象进行运算。二进制运算符以下列格式对其运算对象进行运算：

```
operand1 operator operand2
```

下面的运算对象描述中特别注明了一些使用不同格式的运算符。

如果指定给运算符的运算对象是 null，则结果几乎总是 null（有关异常，请参阅关于逻辑运算符的主题）。

优先级

流式 SQL 遵循通常的运算符优先级：

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. 计算加法和减法（+ 和 -）以及逻辑组合（AND 和 OR）。

如果运算对象之一为 NULL，则结果也为 NULL。如果运算对象的类型不同但可比较，则结果的类型将是精度最高的。如果运算对象的类型相同，则结果将与运算对象的类型相同。例如 5/2 = 2，而不是 2.5，因为 5 和 2 都是整数。

算术运算符

运算符	Unary/Binary	说明
+	U	身份
-	U	求反
+	B	加

运算符	Unary/Binary	说明
-	B	减
*	B	乘
/	B	除

这些运算符中的每一个都根据正常的算术行为运作，但需要注意以下几点：

1. 如果运算对象之一为 NULL，则结果也为 NULL。
2. 如果运算对象的类型不同但可比较，则结果的类型将是精度最高的。
3. 如果运算对象的类型相同，则结果将与运算对象的类型相同。例如 5/2 = 2，而不是 2.5，因为 5 和 2 都是整数。

示例

操作	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

您可以使用流式 SQL 的字符串运算符（包括联接和字符串模式比较）来合并和比较字符串。

运算符	Unary/Binary	说明	注意
||	B	联接	也适用于二进制类型
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

此运算符用于联接一个或多个字符串，如下表所示。

操作	结果
'SQL'||'stream'	SQLstream
'SQL'||''||'stream'	SQLstream
'SQL'||'stream'||' Incorporated'	SQLstream Incorporated
<col1>||<col2>||<col3>||<col4>	<col1><col2><col3><col4>

LIKE 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

模式中的字符	效果
<any other character>	只匹配完全相同的字符

如果任一运算对象为 NULL，则 LIKE 运算的结果为 UNKNOWN。

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。

操作	结果
'a' LIKE 'a'	TRUE
'a' LIKE 'A'	FALSE
'a' LIKE 'b'	FALSE
'ab' LIKE 'a_'	TRUE
'ab' LIKE 'a%'	TRUE
'ab' LIKE 'a_' ESCAPE '\'	FALSE
'ab' LIKE 'a\%' ESCAPE '\'	FALSE
'a_' LIKE 'a_' ESCAPE '\'	TRUE
'a%' LIKE 'a\%' ESCAPE '\'	TRUE
'a' LIKE 'a_'	FALSE
'a' LIKE 'a%'	TRUE
'abcd' LIKE 'a_'	FALSE
'abcd' LIKE 'a%'	TRUE
'' LIKE ''	TRUE
'1a' LIKE '_a'	TRUE

操作	结果
'123aXYZ' LIKE '%a%'	TRUE
'123aXYZ' LIKE '_%_a%_'	TRUE

SIMILAR TO 模式

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

在以下 SIMILAR TO 表中，seq 表示任何明确指定的字符序列，例如 “13aq”。 Non-alphanumeric 用于匹配的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，例如在模式中使用短划线时，将使用当前的排序序列。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里面的模式，然后匹配“下一个最里面的模式”，依此类推。

在应用周围运算之前计算括在圆括号内的表达式和匹配运算，同样优先计算最里面的项目。

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级）	1
方括号 [ ]	[seq]	匹配 seq 中的任何单个字符	2
脱字符或音调符号	[^seq]	匹配不在 seq 中的任何单个字符	3
	[seq ^ seq]	匹配 seq 中和不在 seq 中的任何单个字符	4
短划线	<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）	
条形图	[seq seq]	匹配任一 seq	6
星号	seq*	匹配 seq 的零个或多个重复项	7
加号	seq+	匹配 seq 的一个或多个重复项	8
大括号	seq{<number>}	精确匹配 seq 的重复次数	9
	seq{<low number>,<high number>}	匹配 seq 的低重复次数或更多重复次数，最多匹配高重复次数	10
问号	seq?	匹配 seq 的零个或一个实例	11
下划线	_	匹配任何单个字符	12
百分比	%	匹配任何子字符串，包括空字符串	13
字符	<any other character>	只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

分隔符	模式中的字符	效果	规则 ID
Non-alphanumeric	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 在模式末尾指定的 ESCAPE 子句定义的转义字符。	16

下表列出了示例。

操作	结果	规则
'a' SIMILAR TO 'a'	TRUE	14
'a' SIMILAR TO 'A'	FALSE	14
'a' SIMILAR TO 'b'	FALSE	14
'ab' SIMILAR TO 'a_'	TRUE	12
'ab' SIMILAR TO 'a%'	TRUE	13
'a' SIMILAR TO 'a_'	FALSE	12 和 14
'a' SIMILAR TO 'a%'	TRUE	13
'abcd' SIMILAR TO 'a_'	FALSE	12
'abcd' SIMILAR TO 'a%'	TRUE	13
'' SIMILAR TO ''	TRUE	14
'1a' SIMILAR TO '_a'	TRUE	12
'123aXYZ' SIMILAR TO ''	TRUE	14

操作	结果	规则
'123aXYZ' SIMILAR TO '_%_a%_'	TRUE	13 和 12
'xy' SIMILAR TO '(xy)'	TRUE	1
'abd' SIMILAR TO '[ab][bcde]d'	TRUE	2
'bdd' SIMILAR TO '[ab][bcde]d'	TRUE	2
'abd' SIMILAR TO '[ab]d'	FALSE	2
'cd' SIMILAR TO '[a-e]d'	TRUE	2
'cd' SIMILAR TO '[a-e^c]d'	FALSE	4
'cd' SIMILAR TO '[^(a-e)]d'	INVALID	
'yd' SIMILAR TO '[^(a-e)]d'	INVALID	
'amy' SIMILAR TO 'amyfred'	TRUE	6
'fred' SIMILAR TO 'amyfred'	TRUE	6
'mike' SIMILAR TO 'amyfred'	FALSE	6
'acd' SIMILAR TO 'ab*c+d'	TRUE	7 和 8
'accccd' SIMILAR TO 'ab*c+d'	TRUE	7 和 8
'abd' SIMILAR TO 'ab*c+d'	FALSE	7 和 8
'aabc' SIMILAR TO 'ab*c+d'	FALSE	
'abb' SIMILAR TO 'a(b{3})'	FALSE	9
'abbb' SIMILAR TO 'a(b{3})'	TRUE	9
'abbbbb' SIMILAR TO 'a(b{3})'	FALSE	9
'abbbbb' SIMILAR TO 'ab{3,6}'	TRUE	10

操作	结果	规则
'abbbbbbbb' SIMILAR TO 'ab{3,6}'	FALSE	10
'' SIMILAR TO 'ab?'	FALSE	11
'' SIMILAR TO '(ab)?'	TRUE	11
'a' SIMILAR TO 'ab?'	TRUE	11
'a' SIMILAR TO '(ab)?'	FALSE	11
'a' SIMILAR TO 'a(b?)'	TRUE	11
'ab' SIMILAR TO 'ab?'	TRUE	11
'ab' SIMILAR TO 'a(b?)'	TRUE	11
'abb' SIMILAR TO 'ab?'	FALSE	11
'ab' SIMILAR TO 'a_' ESCAPE '\'	FALSE	16
'ab' SIMILAR TO 'a\%' ESCAPE '\'	FALSE	16
'a_' SIMILAR TO 'a_' ESCAPE '\'	TRUE	16
'a%' SIMILAR TO 'a\%' ESCAPE '\'	TRUE	16
'a(b{3})' SIMILAR TO 'a(b{3})'	FALSE	16
'a(b{3})' SIMILAR TO 'a\(b\{3\}\)' ESCAPE '\'	TRUE	16

逻辑运算符

逻辑运算符允许您创建条件并测试其结果。

运算符	Unary/Binary	说明	运算对象
NOT	U	逻辑求反	布尔值
AND	B	连词	布尔值
或	B	析取	布尔值
IS	B	逻辑断言	布尔值
IS NOT UNKNOWN	U	否定未知比较： <expr> IS NOT UNKNOWN	布尔值
IS NULL	U	Null 比较： <expr> IS NULL	Any
IS NOT NULL	U	否定 null 比较： <expr> IS NOT NULL	Any
=	B	等于	Any
!=	B	不等于	Any
<>	B	不等于	Any
>	B	Greater than	有序类型（数字、字符串、日期、时间）
>=	B	大于或等于（不小于）	有序类型
<	B	Less than	有序类型

运算符	Unary/Binary	说明	运算对象
<=	B	小于或等于（不大于）	有序类型
BETWEEN	三元	范围比较： col1 BETWEEN expr1 AND expr2	有序类型
IS DISTINCT FROM	B	区别	Any
IS NOT DISTINCT FROM	B	否定区别	Any

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，其中最后一种等同于布尔值 NULL。TRUE 和 FALSE 运算对象通常根据普通的双态布尔逻辑起作用，但是在将它们与 UNKNOWN 运算对象配对时会适用其他规则，如下表所示。

> **Note**
>
> UNKOWN 表示“可能是 TRUE，也可能是 FALSE”，或者换句话说，“不绝对是 TRUE，也不绝对是 FALSE”。这种理解可以帮助您弄清为什么表中的某些表达式会这样计算。

求反 (NOT)

操作	结果
NOT TRUE	FALSE
NOT FALSE	TRUE
NOT UNKNOWN	UNKNOWN

连词 (AND)

操作	结果
TRUE AND TRUE	TRUE
TRUE AND FALSE	FALSE
TRUE AND UNKNOWN	UNKNOWN
FALSE AND TRUE	FALSE
FALSE AND FALSE	FALSE
FALSE AND UNKNOWN	FALSE
UNKNOWN AND TRUE	UNKNOWN
UNKNOWN AND FALSE	FALSE
UNKNOWN AND UNKNOWN	UNKNOWN

析取 (OR)

操作	结果
TRUE OR TRUE	TRUE
TRUE OR FALSE	TRUE
TRUE OR UNKNOWN	TRUE
FALSE OR TRUE	TRUE
FALSE OR FALSE	FALSE
FALSE OR UNKNOWN	UNKNOWN
UNKNOWN OR TRUE	TRUE
UNKNOWN OR FALSE	UNKNOWN

操作	结果
UNKNOWN OR UNKNOWN	UNKNOWN

断言 (IS)

操作	结果
TRUE IS TRUE	TRUE
TRUE IS FALSE	FALSE
TRUE IS UNKNOWN	FALSE
FALSE IS TRUE	FALSE
FALSE IS FALSE	TRUE
FALSE IS UNKNOWN	FALSE
UNKNOWN IS TRUE	FALSE
UNKNOWN IS FALSE	FALSE
UNKNOWN IS UNKNOWN	TRUE

IS NOT UNKNOWN

操作	结果
TRUE IS NOT UNKNOWN	TRUE
FALSE IS NOT UNKNOWN	TRUE
UNKNOWN IS NOT UNKNOWN	FALSE

IS NOT UNKNOWN 本身就是一个特殊的运算符。表达式“x IS NOT UNKNOWN”等同于“(x IS TRUE) OR (x IS FALSE)”，而不是“x IS (NOT UNKNOWN)”。因此，在上表中替换：

x	操作	结果		在“(x IS TRUE) OR (x IS FALSE)”中替换 x 的结果
TRUE	TRUE IS NOT UNKNOWN	TRUE	变为	“(TRUE IS TRUE) OR (TRUE IS FALSE)”，即 TRUE
FALSE	FALSE IS NOT UNKNOWN	TRUE	变为	“(FALSE IS TRUE) OR (FALSE IS FALSE)”，即 TRUE
UNKNOWN	UNKNOWN IS NOT UNKNOWN	FALSE	变为	“(UNKNOWN IS TRUE) OR (UNKNOWN IS FALSE)”，即 FALSE， 因为 UNKNOWN 既不是 TRUE 也不是 FALSE

由于 IS NOT UNKNOWN 是一个特殊的运算符，因此上述运算在 IS 这个词周围是不可传递的：

操作	结果
NOT UNKNOWN IS TRUE	FALSE
NOT UNKNOWN IS FALSE	FALSE
NOT UNKNOWN IS UNKNOWN	TRUE

IS NULL 和 IS NOT NULL

操作	结果
UNKNOWN IS NULL	TRUE
UNKNOWN IS NOT NULL	FALSE
NULL IS NULL	TRUE
NULL IS NOT NULL	FALSE

IS DISTINCT FROM 和 IS NOT DISTINCT FROM

操作	结果
UNKNOWN IS DISTINCT FROM TRUE	TRUE
UNKNOWN IS DISTINCT FROM FALSE	TRUE
UNKNOWN IS DISTINCT FROM UNKNOWN	FALSE
UNKNOWN IS NOT DISTINCT FROM TRUE	FALSE
UNKNOWN IS NOT DISTINCT FROM FALSE	FALSE
UNKNOWN IS NOT DISTINCT FROM UNKNOWN	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”相反，后者的通常含义是值（true、false 或 unknown）与其自身相同，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，因为它表示“可能是 TRUE，也许是 FALSE”。

其他逻辑运算符

对于所有其他运算符，传递 NULL 或 UNKNOWN 运算对象将导致结果为 UNKNOWN（与 NULL 相同）。

示例

操作	结果
TRUE AND CAST(NULL AS BOOLEAN)	UNKNOWN
FALSE AND CAST(NULL AS BOOLEAN)	FALSE
1 > 2	FALSE
1 < 2	TRUE
'foo' = 'bar'	FALSE
'foo' <> 'bar'	TRUE
'foo' <= 'bar'	FALSE
'foo' <= 'bar'	TRUE
3 BETWEEN 1 AND 5	TRUE
1 BETWEEN 3 AND 5	FALSE
3 BETWEEN 3 AND 5	TRUE
5 BETWEEN 3 AND 5	TRUE
1 IS DISTINCT FROM 1.0	FALSE
CAST(NULL AS INTEGER) IS NOT DISTINCT FROM CAST (NULL AS INTEGER)	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 流式 SQL 支持 Unicode 字符文字，例如 u&'foo'。与使用常规文字一样，您可以对这些文字中的单引号进行转义，例如 u&'can''t'。与常规文字不同，您可以使用 Unicode 转义符：例如，u&'\0009' 是一个仅由制表符组成的字符串。您可以用另一个 \ 来转义 \，例如 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 查询对无限行流进行操作，因此只有在对这些流有所了解的情况下，才能进行某些操作。

例如，对于给定订单流，要求提供按日期和产品汇总订单的流是合理的（因为天数在增加），但要求提供按产品和发货状态汇总订单的流则不合理。我们永远无法完成通过小组件 X 发往俄勒冈州的汇总，因为我们永远看不到通过小组件发往俄勒冈州的“最后一个”订单。

这种按特定列或表达式排序的流的属性称为单调性。

一些与时间相关的定义：

- 单调。如果表达式是升序或降序的，则是单调的。等效的措辞是“不减少或不增加”。
- 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 在流中是升序的。
- 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 在流中是降序的。
- 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 在流中是严格升序的。
- 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 在流中是严格降序的。
- 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 在流中是常量的。

请注意，根据这个定义，常量表达式被认为是单调的。

单调列

ROWTIME 系统列是升序的。ROWTIME 列不是严格升序的：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中的前一行的流中插入一行。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，使得 ROWTIME 列是升序的。

例如，断言 orderId 列是升序的；或者排序顺序中没有 orderId 超过 100 行，显然是非常有用的。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 知道表达式的参数是单调的，就能推断出该表达式是单调的。（另请参阅单调函数。）

另一个定义：

单调的函数或运算符

如果将函数或运算符应用于严格增加的值序列时，生成的结果序列是单调的，则该函数或运算符就是单调的。

例如，将 FLOOR 函数应用于升序的输入 {1.5、3、5、5.8、6.3} 时，会生成 {1、3、5、5、6}。请注意，输入是严格升序的，但输出只是升序的（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才能使流式 GROUP BY 语句有效。在其他情况下，如果 Amazon Kinesis Data Analytics 知道单调性，就能更高效地运行；例如，如果它知道某个特定键再也不会出现在流中，就能从窗口式聚合总数表中删除条目。

为了以这种方式利用单调性，Amazon Kinesis Data Analytics 使用了一套规则来推断表达式的单调性。下面是推断单调性的规则：

Expression	单调性
c	常量
FLOOR(m)	与 m 相同，但不严格
CEIL/CEILING(m)	与 m 相同，但不严格
CEIL/CEILING(m TO timeUnit)	与 m 相同，但不严格
FLOOR(m TO timeUnit)	与 m 相同，但不严格
SUBSTRING(m FROM 0 FOR c)	与 m 相同，但不严格
+ m	与 m 相同
-m	与 m 相反
m + c c + m	与 m 相同
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	与 m 相反
m * c c * m	如果 c 为正数，则与 m 相同； 如果 c 为负数，则与 m 相反；如果 c 为 0，则是常数 (0)

Expression	单调性
c / m	如果 m 始终是正数或始终是负数，并且 c 和 m 的符号相同，则与 m 相同； 如果 m 始终是正数或始终是负数，并且 c 和 m 的符号不同，则与 m 相反； 否则不是单调的
	常量
LOCALTIME LOCALTIMESTAMP CURRENT_ROW_TIMESTAMP CURRENT_DATE	升序

在整个表中，c 是一个常数，m（以及 m1 和 m2）是一个单调表达式。

条件子句

引用者：

- SELECT 子句：HAVING 子句、WHERE 子句 和 JOIN 子句。（另请参阅 SELECT 图表及其 SELECT 子句。）
- DELETE

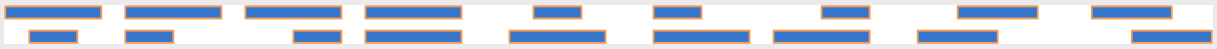
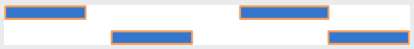
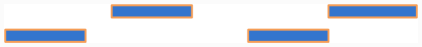
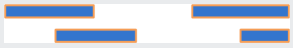
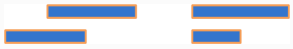
条件是任何类型为 BOOLEAN 的值表达式，如下例所示：

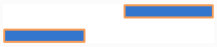
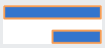
- 2<4
- TRUE
- FALSE
- expr_17 IS NULL
- NOT expr_19 IS NULL AND expr_23 < expr>29

- expr_17 IS NULL OR (NOT expr_19 IS NULL AND expr_23 < expr>29)

时间谓词

下表以图形方式显示了标准 SQL 支持的时间谓词以及对 Amazon Kinesis Data Analytics 支持的 SQL 标准的扩展。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。最后 10 个谓词（以粗体显示）是 Amazon Kinesis Data Analytics 对 SQL 标准的扩展。

谓词	涵盖的关系
CO	
OV	
EQ	
PR	
SU	
IM LY PR	
IM LY SU	
LE	
LA	
ST CO	

谓词	涵盖的关系
STF OV	
STF PR	
STF SU	
STF LE	
STF LA	
IMM LY LE	
IMM LY LA	

为了实现简洁的表达式，Amazon Kinesis Data Analytics 还支持以下扩展：

- 可选 PERIOD 关键字 – 可忽略 PERIOD 关键字。
- 紧凑链接 – 如果这些谓词中有两个背靠背出现并由 AND 分隔，则可以忽略 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 – 此函数将选取两个 TIMESTAMP 参数并返回两者之差（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)
```

更简洁的版本如下所示：

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

以下简洁的表达方式：

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

即以下表达式：

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

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

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

等效于以下表达式：

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

语法

时间谓词被集成到一个值为 BOOLEAN 的新表达式中：

```
<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 )
```

以毫秒为单位返回 (endTime - startTime) 的值。

示例

以下示例代码会记录因窗户在空调开启时处于打开状态而发出的警报：

```
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);
```

示例使用案例

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，以下查询将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

以下是自版本 5.0.1 起 Amazon Kinesis Data Analytics 应用程序中的保留关键字的列表。

A		
ABS	ALL	ALLOCATE
允许	ALTER	ANALYZE
AND	ANY	APPROXIMATE_ARRIVAL_TIM
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	CREATE	CROSS
CUBE	CUME_DIST	CURRENT
CURRENT_CATALOG	CURRENT_DATE	CURRENT_DEFAULT_TRANSFORM_GROUP
CURRENT_PATH	CURRENT_ROLE	CURRENT_SCHEMA
CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_TRANSFORM_GROUP_FOR_TYPE

CURRENT_USER	CURSOR	CYCLE
	D	
DATE	DAY	DEALLOCATE
DEC	DECIMAL	DECLARE
DEFAULT	DELETE	DENSE_RANK
DEREF	DESCRIBE	DETERMINISTIC
DISALLOW	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	IMPORT
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	新	否
NODE	NONE	NORMALIZE
NOT	NTH_VALUE	NULL
NULLIF	NUMERIC	
	O	
OCTET_LENGTH	OF	OLD
ON	ONLY	OPEN
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
参考	REFERENCING	REGR_AVGX
REGR_AVGY	REGR_COUNT	REGR_INTERCEPT
REGR_R2	REGR_SLOPE	REGR_SXX
REGR_SXY	RELEASE	RESPECT
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
STREAM	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	UPDATE
UPPER	USER	USING
	V	
值	VALUES	VARBINARY
VARCHAR	VARYING	VAR_POP
VAR_SAMP		
	W	
WHEN	WHENEVER	WHERE

WIDTH_BUCKET	WINDOW	WITH
WITHIN	WITHOUT	
	Y	
YEAR		

标准 SQL 运算符

以下主题讨论了标准 SQL 运算符：

主题

- CREATE 语句
- INSERT
- Query
- SELECT 语句

CREATE 语句

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

- CREATE FUNCTION
- CREATE PUMP
- CREATE STREAM

CREATE STREAM

CREATE STREAM 语句创建一个（本地）流。流的名称必须与同一架构中任何其他流的名称不同。最好包括流的描述。

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。它们应映射到您要为其创建流的数据来源。对于 column_name，可以使用任何有效的非保留 SQL 名称。列值不能为 null。

- 指定 OR REPLACE 会重新创建已存在的流，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。在已有数据在运行的流上使用 CREATE OR REPLACE 会结束流并丢失所有历史记录。
- 只有指定了 OR REPLACE，才能指定 RENAME。
- 有关采用 type_specification 的类型和值的完整列表 [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 数据类型”。
- 对于 option_value，可以使用任何字符串。

未解析的日志数据的简单流

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

从智能旅行系统管道捕获传感器数据的流

```
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';
```

从电子商务管道捕获订单数据的流

```
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 提供了很多函数，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。

User-defined 可以使用完全限定名称调用函数，也可以仅使用函数名称来调用函数。

传递给用户定义的函数或转换（或从用户定义的函数或转换返回）的值必须与相应的参数定义具有完全相同的数据类型。换句话说，在将参数传递给用户定义的函数（或从用户定义的函数返回值）时不允许使用隐式转换。

User-Defined 函数 (UDF)

用户定义的函数可以实现复杂的计算，采用零个或多个标量参数并返回标量结果。UDF 的运算方式类似于内置函数，例如 FLOOR() 或 LOWER()。对于 SQL 语句中出现的每个用户定义的函数，每行都会使用标量参数（该行中的常量或列值）调用该 UDF 一次。

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>''
```

SPECIFIC 会分配一个在应用程序中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用同一名称，只要能够通过其参数列表加以区分即可）。

DETERMINISTIC/NOT DETERMINISTIC 表示函数是否会始终针对给定的一组参数值返回相同结果。您的应用程序可能会使用它来进行查询优化。

READS SQL DATA 和 MODIFIES SQL DATA 分别表示函数是否可能读取或修改 SQL 数据。如果函数尝试在未指定 READS SQL DATA 的情况下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。

RETURNS NULL ON NULL INPUT 和 CALLED ON NULL INPUT 表示函数在其任何参数为 null 时是否会被定义为返回 null。如果未指定，则默认为 CALLED ON NULL INPUT。

SQL-defined 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)
;
```

CREATE PUMP

数据泵是 Amazon Kinesis Data Analytics 存储库对象（SQL 标准的扩展），提供连续运行的 INSERT INTO 流 SELECT ... FROM 查询功能，从而使查询的结果能够连续输入到命名流中。

您需要同时为查询和命名流指定列列表（这表示一组源-目标对）。列列表在数据类型方面需要匹配，否则 SQL 验证器将拒绝它们。（它们不必列出目标流中的所有列；您可以为某个列设置数据泵。）

有关更多信息，请参阅 SELECT 语句。

以下代码首先创建并设置架构，然后在此架构中创建两个流：

- “OrderDataWithCreateTime” 它将作为泵的源流。
- “OrderData” 将用作泵的目标流。

```
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数据泵。从 “OrderDataWithCreateTime” 中选择数据并插入到 “OrderData” 中。

```
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";
```

有关更多详细信息，请参阅《适用于 Apache 的亚马逊托管服务 Flink 开发人员指南》中的 “In-Application 流和泵” 主题。

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。也可用在数据泵中，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 > ]
```

有关值的讨论，请参阅 SELECT 语句。

数据泵流插入

也可以将 INSERT 指定为 CREATE PUMP 语句的一部分。

```
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流插入。插入的行的行时间将继承自 select 或 UNION ALL 输出的行的行时间。同样，如果在此插入器之前的其他插入器插入的行的行时间晚于此插入器最初准备的行的行时间，则可能会删除最初的行，因为后者不符合时间顺序。请参阅本指南中的CREATE PUMP主题。

Query

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

上图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。

集合运算符（EXCEPT、INTERSECT、UNION）

集合运算符使用集合运算来组合查询生成的行：

- EXCEPT 返回在第一个集合中但不在第二个集合中的所有行
- INTERSECT 返回同时在第一个和第二个集合中的所有行
- UNION 返回在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，并且列类型必须可以兼容分配。生成的关系的列名称是第一个查询的列名称。

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，如果某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 将生成该行 3 + 2 = 5 次。

当前不支持 ALL 使用 EXCEPT 或 INTERSECT。

所有运算符都是左关联的，INTERSECT 的优先级高于 EXCEPT 或 UNION，后两者的优先级相同。要覆盖默认优先级，您可以使用圆括号。例如：

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

相当于完全带圆括号的查询

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

流式集合运算符

UNION ALL 是唯一可以应用于流的集合运算符。该运算符的两边都必须是流；如果一边是流，另一边是关系，则会出错。

例如，以下查询生成通过电话或网络下达的订单流：

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

```
SELECT STREAM *
  FROM WebOrders
```

行时间生成。流式 UNION ALL 生成的行的行时间与输入行的时间戳相同。

行时间边界：行时间边界是关于流的未来内容的断言。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 通过按时间戳合并传入行，确保 ROWTIME 列为升序。如果第一个集合的行的行时间戳为 10:00 和 10:30，第二个集合只到达 10:15，Kinesis Data Analytics 会暂停第一个集合，并等待第二个集合到达 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。

VALUES 运算符

VALUES 运算符表示查询中的常量关系。（另请参阅本指南 SELECT 主题中对 VALUES 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任意列名称。您可以通过将 VALUES 放入子查询并使用 AS 子句来分配列名称：

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

```
42 Fred
34 Wilma
```

SELECT 语句

SELECT 从流中检索行。您可以将 SELECT 用作顶级语句，也可以用作涉及集合运算的查询的一部分，或其他语句的一部分，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 CREATE PUMP。

本指南中的 SELECT 子句、GROUP BY 子句、流式 GROUP BY、ORDER BY 子句、HAVING 子句、WINDOW 子句 (滑动窗口)和 WHERE 子句主题中介绍了 SELECT 语句的子句。

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 SQL 的原理

SQL 查询语言专为查询存储关系和生成有限关系结果而设计。

流式 SQL 的基础是 STREAM 关键字，以便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，或者根据多种关系计算得出的表达式的变化。

要在 Amazon Kinesis Data Analytics 中查询关系的时间差，我们使用 STREAM 关键字：

```
SELECT STREAM * FROM Orders
```

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

```
ROWTIME  orderId custName   product quantity
======== ======= ========== ======= ========
```

```
10:15:00     102 Ivy Black  Rice           6
10:25:00     103 John Wu    Apples         3
```

在这里，系统表示：“在 10:15:00，我执行了查询 SELECT * FROM Orders，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，因为这是该行出现的时间。这是流的核心思想：一种会随着时间的推移而不断更新的关系。

您可以将此定义应用于更复杂的查询。例如，流

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

在 10:15 有一行但在 10:25 没有行，因为这种关系

```
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 SQL 运算符任意组合的查询。当转换为流时，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）的查询都是明确定义的。结合了流和存储关系的查询也是明确定义的。

SELECT ALL 和 SELECT DISTINCT

如果指定了 ALL 关键字，则查询不会消除重复的行。如果既未指定 ALL 也未指定 DISTINCT，则这是默认行为。

如果指定了 DISTINCT 关键字，则查询会根据 SELECT 子句中的列来消除重复的行。

请注意，出于这些目的，值 NULL 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。

流式 SELECT 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基本原理与流式 GROUP BY 相同。）Amazon Kinesis Data Analytics 在 SELECT DISTINCT 对应的行就绪后将立即生成这些行。

如果 ROWTIME 是 SELECT 子句中的一列，则出于消除重复项的目的，该列将被忽略。在 SELECT 子句中的其他列的基础上消除重复项。

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的集合。

如果您正在进行“GROUP BY floor(ROWTIME TO MINUTE)”，并且在给定的一分钟内有两行，例如 22:49:10 和 22:49:15，那么这些行的摘要就会出现，时间戳为 22:50:00。为什么？ 因为这是该行完成的最早时间。

注意：“GROUP BY ceil(ROWTIME TO MINUTE)”或“GROUP BY floor(ROWTIME TO MINUTE) - INTERVAL '1' DAY”将提供相同的行为。

决定行完成的不是分组表达式的值，而是该表达式的值何时发生变化。

如果您希望输出行的行时间是其生成时间，那么在下面的示例中，您需要从表单 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

<select-clause> 在 STREAM 关键字之后使用以下项目：

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

Expressions

这些表达式中的每一个都可以是：

- 标量表达式
- 对聚合函数的调用，前提是这是聚合查询（请参阅 GROUP BY 子句）
- 对分析函数的调用，前提是这是聚合查询
- 通配符表达式 * 扩展到 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
- 那个 ROWTIME
- 一个 CASE 表达式

可以使用 AS column_name 语法为每个表达式都分配一个别名。这是此查询的结果集中的列名称。如果此查询位于封闭查询的 FROM 子句中，则此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，默认情况下别名 EMPS.DEPTNO 为 DEPTNO。其他表达式则使用像 EXPR$0 这样的别名。您不应假设系统每次都会生成相同的别名。

在流式查询中，将列的别名设置为 ROWTIME 具有特殊含义：有关更多信息，请参阅 ROWTIME。

> **Note**
>
> 所有流都有一个名为 ROWTIME 的隐式列。此列可能会影响您使用现在支持的语法 “AS t (c1, c2,...)”。 SQL:2008以前在 FROM 子句中您只能编写

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

但是 t1 和 t2 的列将与 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 必须恰好有 3 列才能使用此语法）。

如果 r1 是流，则隐式包含 ROWTIME，但它不算作一列。因此，如果一个流有 3 列而不包含 ROWTIME，则不能通过指定 4 列来重命名 ROWTIME。例如，如果 Bids 流有三列，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。

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；每个返回值表达式都在相应的 THEN 子句中指定。可以指定多个这样的 WHEN-THEN 对。

如果您在第一个 WHEN 子句之前指定比较测试表达式，则会将 WHEN 子句中的每个表达式与该比较测试表达式进行比较。第一个匹配比较测试表达式的值会导致返回其相应的 THEN 子句的返回值。如果没有 WHEN 子句表达式与比较测试表达式匹配，则除非指定了 ELSE 子句，否则返回值为 null，在这种情况下，将返回该 ELSE 子句中的返回值。

如果您没有在第一个 WHEN 子句之前指定比较测试表达式，则会计算 WHEN 子句中的每个表达式（从左到右），而第一个为 true 的表达式会导致返回其对应的 THEN 子句的返回值。如果没有 WHEN 子句表达式为 true，则除非指定了 ELSE 子句，否则返回值为 null，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派生自该列中出现的表达式的显式或推断类型。只要允许 SELECT，语法上都允许使用 VALUES。另请参阅本指南“查询”主题中对 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 相同，支持以下类型的操作：

- 您可以编写 VALUES (1)、(2) 来返回两行，每行都有一个匿名列。
- 您可以编写 VALUES (1, 'a')、(2, 'b') 来返回两行，每行都有两列。

- 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，您也可以插入多行：

```
   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，必须将整个 VALUES 子句括在圆括号中，这与其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。

> **Note**
>
> 在流中使用 INSERT 需要考虑行时间、数据泵和 INSERT EXPEDITED 等其他一些注意事项。有关更多信息，请参阅 INSERT。

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 window-specification 和 windowing-clause 图表，请参阅 Window 语句下的 WINDOW 子句 (滑动窗口)。

```
<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 子句中可以出现多种类型的关系：

- 命名关系（表、流）。
- 用圆括号括起来的子查询。
- 合并两种关系的联接（请参阅本指南中的 JOIN 主题）。
- 转换表达式。

本指南中的“查询”主题对子查询进行了更详细的介绍。

下面是子查询的一些示例：

```
// 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 [例如 WHERE 子句 子句 (WHERE 条件子句)] 中的子查询不同，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)
```

包含多种关系的 FROM 子句

如果 FROM 子句包含多种以逗号分隔的关系，则查询将构成这些关系的笛卡尔乘积；也就是说，它会将每种关系中的每一行与所有其他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。

关联名称

可以使用 AS 关联名称为 FROM 子句中的每种关系分配一个关联名称。此名称是备用名称，在整个查询过程中，可以使用此名称在表达式中引用相应关系。（尽管关系可能是子查询或流，但通常将其称为“表别名”，以与 SELECT 子句中定义的列别名区分开来。）

如果没有 AS 子句，则命名关系的名称将成为其默认别名。（在流式查询中，OVER 子句不会阻止这种默认分配的发生。）

如果查询多次使用同一命名关系，或者任何关系是子查询或表表达式，则需要使用别名。

例如，在以下查询中，命名关系 EMPS 被使用了两次；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
```

可以选择在别名后面加上列的列表：

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

OVER 子句

OVER 子句仅适用于流式联接。有关更多详细信息，请参阅本指南中的 JOIN 子句主题。

JOIN 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

- Stream-to-Stream 加入
- Stream-to-Table 加入

Stream-to-Stream 加入

Amazon Kinesis Data Analytics 支持使用 SQL 将应用程序内流与另一个应用程序内流相联接，从而将这一重要的传统数据库功能带入流上下文中。

本节介绍 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，以及有关流式联接的详细信息。

联接类型

联接有五种类型：

INNER JOIN（或者仅 JOIN）	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。
LEFT OUTER JOIN（或者仅 LEFT JOIN）	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。
RIGHT OUTER JOIN（或者仅 RIGHT JOIN）	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。
FULL OUTER JOIN（或者仅 FULL JOIN）	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	返回输入的笛卡尔积：左侧的每个行均与右侧的每个行配对。

Time-Based 窗口联接与 Row-Based 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内要考虑联接的行窗口。

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。

Stream-to-Stream 连接示例

以下示例演示应用程序内流到流联接的工作原理、何时返回联接的结果以及联接结果的行时间。

主题

- 示例数据集
- 示例 1： JOIN (INNER JOIN) 一侧的时间窗口
- 示例 2： JOIN (INNER JOIN) 两侧的时间窗口
- 示例 3：RIGHT JOIN (RIGHT OUTER JOIN) 一侧的时间窗口
- 示例 4：RIGHT JOIN (RIGHT OUTER JOIN) 两侧的时间窗口
- 示例 5：LEFT JOIN (LEFT OUTER JOIN) 一侧的时间窗口
- 示例 6：LEFT JOIN (LEFT OUTER JOIN) 两侧的时间窗口
- Summary

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_ In-Application STREAM 直播

```
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 SHIPMENT In-Application S_STREAM 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： JOIN (INNER JOIN) 一侧的时间窗口

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";
```

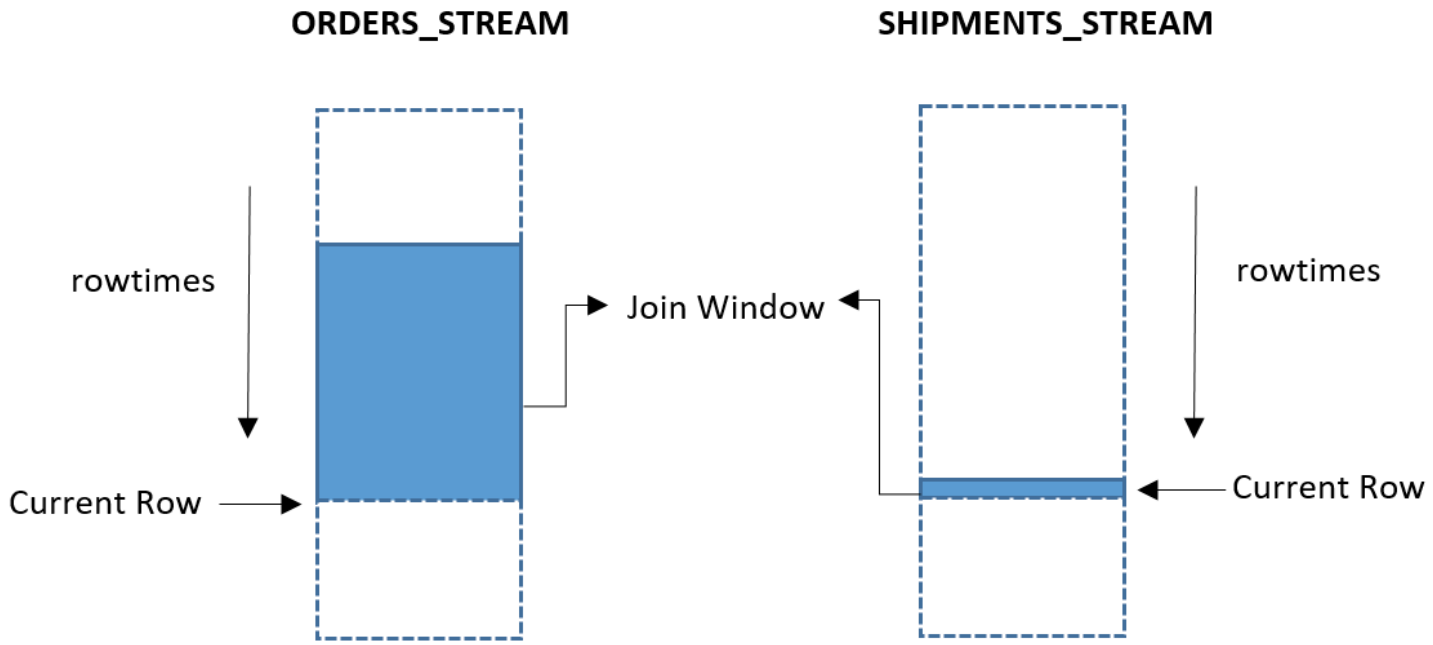
查询结果

ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM			
ROWTIME	orderid	ROWTIME	orderid	resultrowtime	orderid	shipmenttime	OrderTime
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。
- 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： JOIN (INNER JOIN) 两侧的时间窗口

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";
```

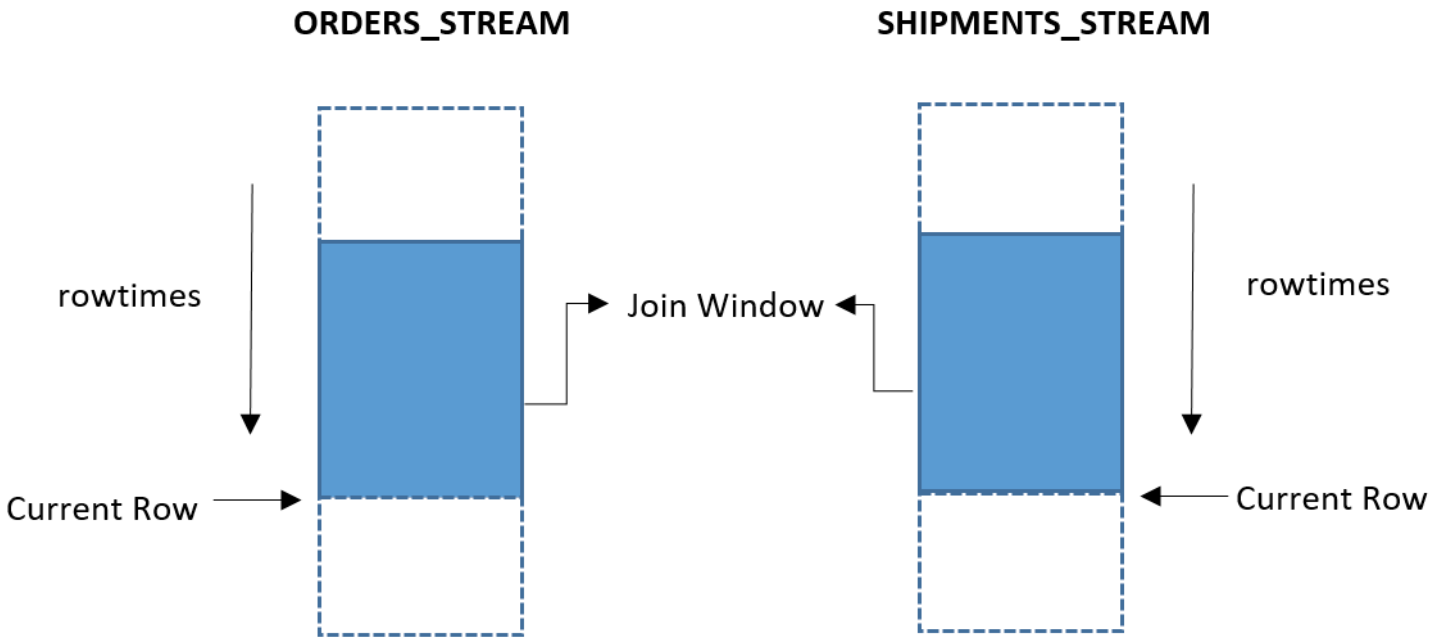
查询结果

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：RIGHT JOIN (RIGHT OUTER JOIN) 一侧的时间窗口

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";
```

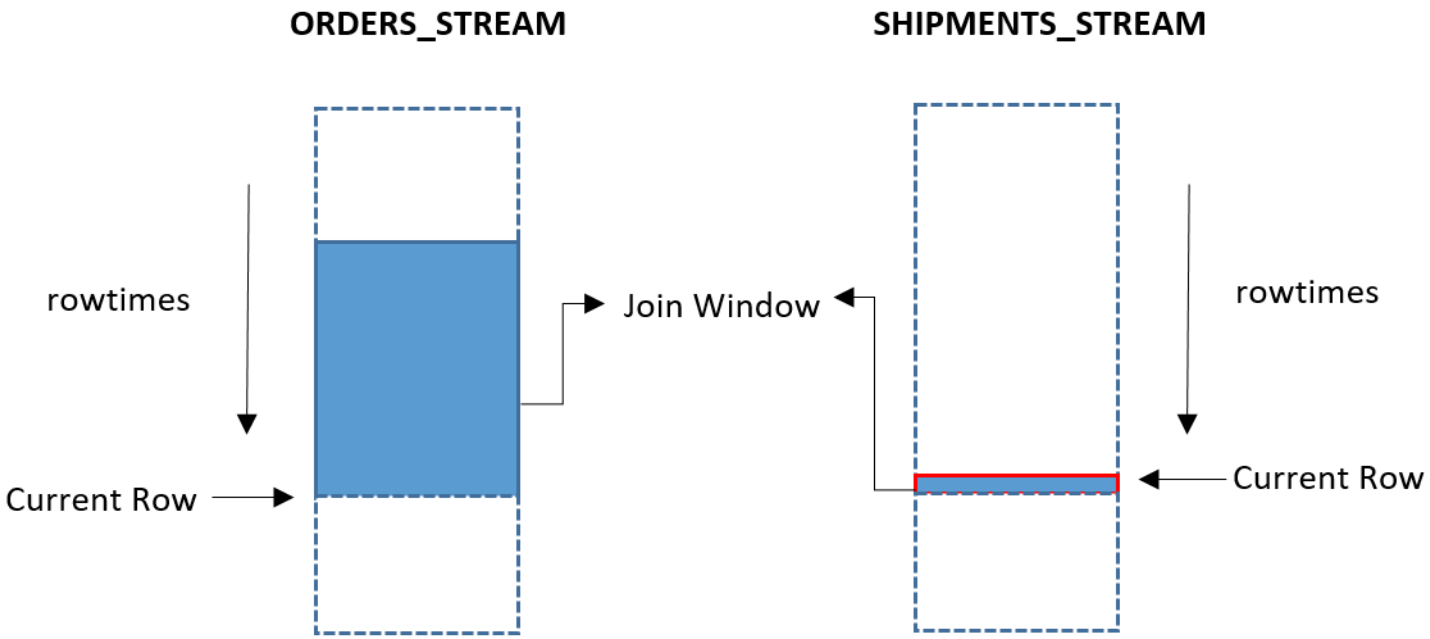
查询结果

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，因此该 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：RIGHT JOIN (RIGHT OUTER JOIN) 两侧的时间窗口

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";
```

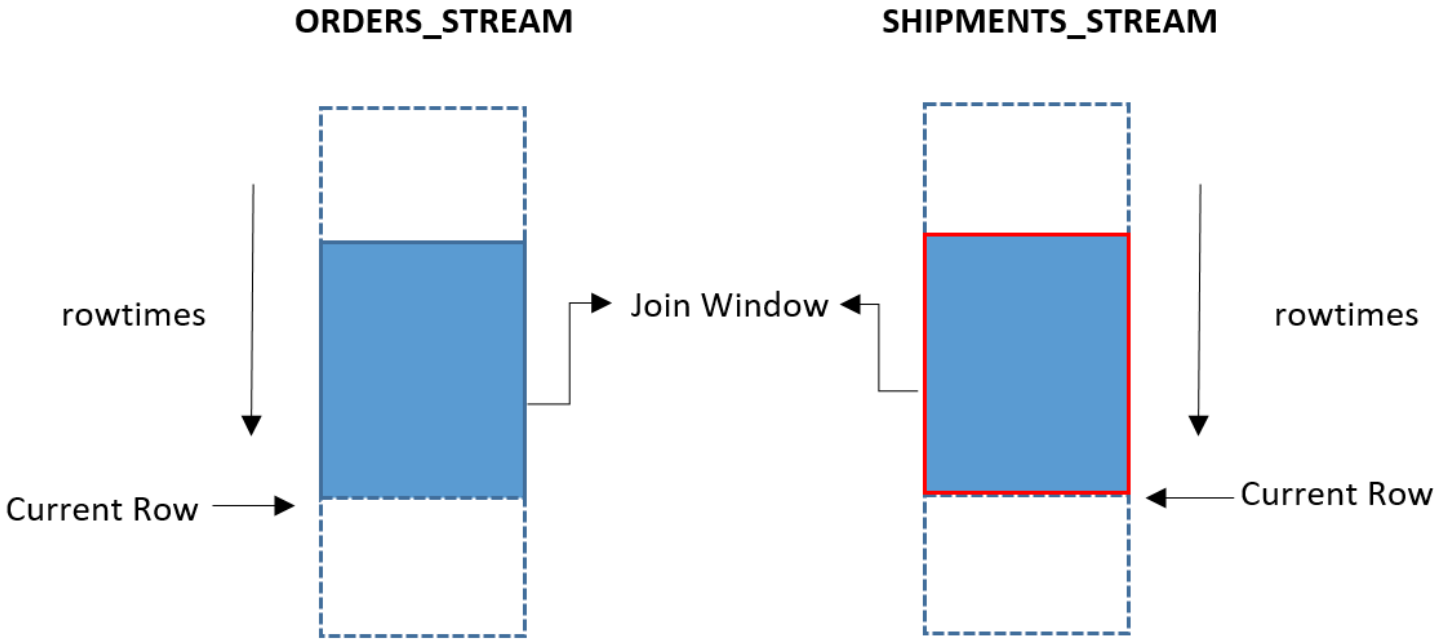
查询结果

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：LEFT JOIN (LEFT OUTER JOIN) 一侧的时间窗口

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

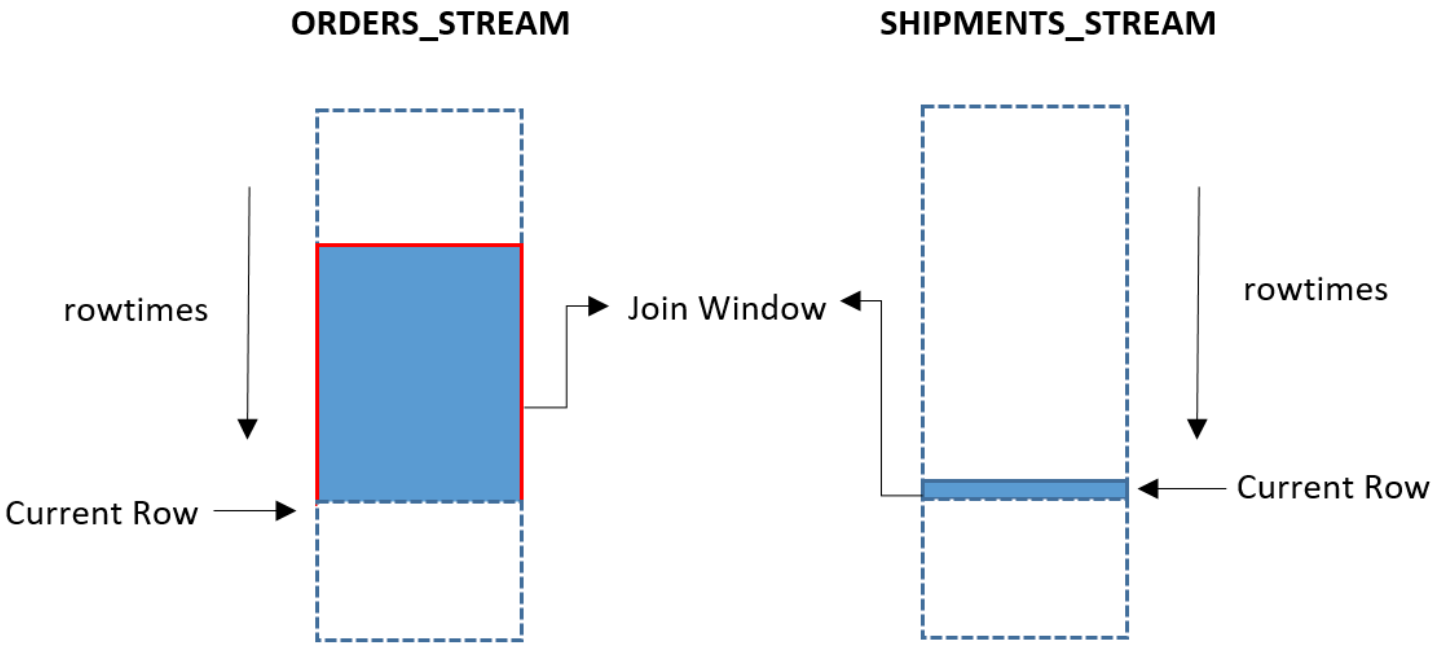
ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM		
ROWTIME	orderid	ROWTIME	orderid	resultrowtime	orderid	OrderTime
10:00:00	101	10:00:00	100			
10:00:20	102					
10:00:30	103					
10:00:40	104					
		10:00:45	104			
10:00:45	100*			10:00:45	104	10:00:40
		10:00:50	105			
				10:01:00	101	10:00:00
				10:01:20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

ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM		
				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：LEFT JOIN (LEFT OUTER JOIN) 两侧的时间窗口

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

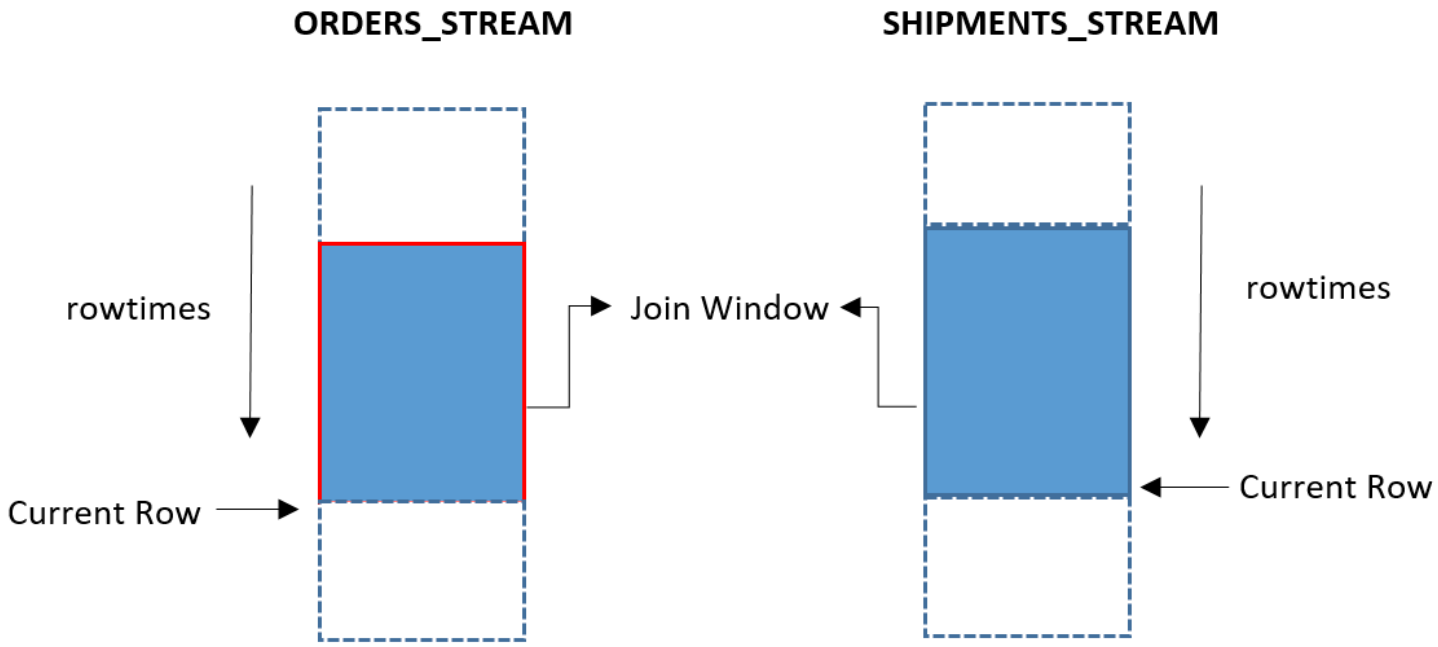
ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM			
ROWTIME	orderid	ROWTIME	orderid	resultrowtime	orderid	shipmenttime	OrderTime
10:00:00	101	10:00:00	100				
10:00:20	102						
10:00:30	103						
10:00:40	104						
		10:00:45	104				
10:00:45	100*			10:00:45	104	10:00:40	10:00:45

ORDERS_STREAM		SHIPMENTS_STREAM		OUTPUT_STREAM			
		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。

Summary

- Kinesis Data Analytics 始终以 ROWTIME 的升序从联接返回行。
- 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。

Stream-to-Table 加入

如果其中一种关系是流，另一种是有限关系，则将其称为流-表联接。对于流中的每一行，查询都会在表中查找与联接条件相匹配的一行或多行。

例如，Orders 是一个直播， PriceList 也是一个表格。联接的效果是将价目表信息添加到订单中。

有关创建参考数据来源并将流联接到参考表的信息，请参阅 Amazon Kinesis Data Analytics 开发人员指南中的示例：添加参考数据来源。

HAVING 子句

SELECT 中的 HAVING 子句用于指定要在组或聚合中应用的条件。换句话说，HAVING 会在对 GROUP BY 子句应用聚合之后筛选行。由于 HAVING 在 GROUP BY 之后计算，因此只能引用由分组

键、聚合表达式和常量构建（或派生）的表达式。（这些规则也适用于 GROUP BY 查询的 SELECT 子句中的表达式。）HAVING 子句必须位于 GROUP BY 子句之后和 ORDER BY 子句之前。HAVING 类似于 WHERE 子句，但适用于组。HAVING 子句生成的结果表示原始行的分组或聚合，而 WHERE 子句生成的结果是各个原始行。

在非流式应用程序中，如果没有 GROUP BY 子句，则假定使用 GROUP BY ()（但由于没有分组表达式，因此表达式只能由常量和聚合表达式组成）。在流式查询中，如果没有 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-condition）的各个行。GROUP BY 条件仅适用于 WHERE 条件选择的行。

这样的分组，例如 “GROUP BY CustomerID”，可以由 a 进一步限定 HAVING-condition，然后在指定分组中选择满足其条件的行的聚合。例如，“按客户身份分组 HAVING SUM (ShipmentValue) > 3600” 将仅选择那些符合 WHERE 标准的各种货件的总价值也超过 3600 的客户。

有关同时适用于 HAVING 和 WHERE 子句的条件，请参阅 WHERE 子句语法图表。

条件必须是布尔谓词表达式。查询仅返回谓词为 TRUE 的行。

以下示例显示的流式查询展示了过去一小时内订单金额超过 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 子句的语法图表

（要查看此子句的适用范围，请参阅 SELECT 语句）

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

- 表达式可以是聚合；
- GROUP BY 子句中使用的所有列名称也必须在 SELECT 语句中。

此外，除非在 SUM (all) 等聚合中，否则不能在 SELECT 语句中命名或无法从 GROUP BY 子句派生的列出现在 SELECT 语句中。OrdersValue

可派生是指，在 GROUP BY 子句中指定的列允许访问要包含在 SELECT 子句中的列。如果列是可派生的，则即使未在 GROUP BY 子句中明确命名该列，SELECT 语句也可以指定该列。

示例：如果表的键在 GROUP BY 子句中，则该表的所有列都可以出现在 select-list 中。由于该键，此类列被视为可访问。

GROUP BY 子句根据分组表达式的值对选定行进行分组，针对每组在所有列中具有相同值的行返回一行摘要信息。

请注意，出于这些目的，值 NULL 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。

流式 GROUP BY

GROUP BY 可以用在流式查询中，但前提是有一个分组表达式是非常量单调表达式或基于时间的表达式。Amazon Kinesis Data Analytics 需要遵循此要求才能继续运行，如下所述。

单调表达式始终朝着同一方向移动：递升或保持不变，或者递降或保持不变；不会逆转方向。不需要严格递升或严格递降，也就是说，每个值始终高于前一个值，或者每个值始终低于前一个值。常量表达式具备单调性，从技术上讲，既可递升又可递降，但显然不适用于这些目的。有关单调性的更多信息，请参阅单调表达式和运算符。

请考虑以下查询：

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

该查询旨在以流的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 永远无法知道已查看给定产品的所有订单，永远无法完成特定行的总数，因此永远无法输出行。Amazon Kinesis Data Analytics 验证器会拒绝而非允许永远无法输出行的查询。

流式 GROUP BY 的语法如下：

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

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

其中 GROUP BY 子句中使用的所有列名称都需要在 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，以下计算每小时产品数量的查询使用单调表达式 FLOOR (Orders.ROWTIME TO HOUR) 因此是有效的：

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

GROUP BY 中必须要有一个表达式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每小时生成一个输出行。GROUP BY 可以指定其他分区项。例如，按楼层分组 (S.ROWTIME) 到小时，USERID 将每个 USERID 值每小时生成一个输出行。如果您确定表达式是单调的，则可以使用单调函数进行声明。如果实际数据不是单调的，则生成的系统行为是不确定的：结果可能不符合预期或期望。

有关更多详细信息，请参阅本指南中的单调函数主题。

流中可能会出现重复的行时间，但只要 ROWTIME 值相同，GROUP BY 操作就会继续累积行。为了输出行，ROWTIME 值必须在某个时刻发生更改。

WHERE 子句

WHERE 子句会提取满足指定条件的记录。条件可以是数字或字符串比较，也可以使用 BETWEEN、LIKE 或 IN 运算符：请参阅流式 SQL 运算符。可以使用 AND、OR 和 NOT 等逻辑运算符组合条件。

WHERE 子句类似于 HAVING 子句。它适用于组，也就是说， WHERE 子句生成的结果是各个原始行，而 HAVING 子句生成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-condition ）的各个行。GROUP BY 条件仅适用于 WHERE 条件选择的行。这样的分组，例如 “GROUP BY CustomerID”，可以由 a 进一步限定 HAVING-condition，然后在指定分组中选择满足其条件的行的聚合。例如，“按客户身份分组 HAVING SUM (ShipmentValue) > 3600” 将仅选择那些符合 WHERE 标准的各种货件的总价值也超过 3600 的客户。

要查看此子句在 SELECT 语句中的位置，请参阅SELECT 语句。

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，则不会生成该行。

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

本指南中的 JOIN 子句和 HAVING 子句主题也对 WHERE 进行了讨论。

WINDOW 子句 (滑动窗口)

滑动窗口式查询的 WINDOW 子句指定一些行，将跨与当前行相关的一组行针对这些行计算分析函数。这些聚合函数生成一个输出行，该行按每个输入行的一个或多个列中的键进行聚合。查询中的 WINDOW 子句指定流中按时间范围间隔或行数分区的记录，以及由 PARTITION BY 子句指定的一组额外的可选列。您可以定义指定的或内联的窗口规范，这些规范可在分析函数和流式 JOIN 子句中使用。有关分析函数的更多信息，请参阅分析函数。

对 OVER 子句中指定的每个列执行滑动窗口查询中的聚合函数。OVER 子句可以引用指定的窗口规范，也可以作为数据泵的 SELECT 语句的一部分内联。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：引用指定的窗口规范的 OVER

以下示例显示的是引用了名为 W1 的窗口规范的聚合函数。在此示例中，基于 W1 窗口规范所指定的记录集来计算平均价格。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。

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

示例 2：引用内联窗口规范的 OVER

以下示例显示的是引用了内联窗口规范的聚合函数。在此示例中，基于每个输入行与内联窗口规范来计算平均价格。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。

```
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。

参数

window-name

指定可从 OVER 子句或后续窗口定义引用的唯一名称。此名称在分析函数和流式 JOIN 子句中使用。有关分析函数的更多信息，请参阅分析函数。

AS

为 WINDOW 子句定义指定的窗口规范。

PARTITION BY partition-name

将行划分到共享相同值的组中。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

根据时间范围间隔指定窗口边界。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，后者是 Amazon Kinesis Analytics 开发人员指南中的入门的一部分。要运行每个示例，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如

何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。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：引用命名窗口规范的 Time-Based 滑动窗口

此示例定义了一个指定的窗口规范，其分区边界为当前行前一分钟。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	MAX_PRICE	AVG_PRICE
2017-01-31 23:30:11.661	QXZ	215.0399932861328	215.0399932861328	215.0399932861328
2017-01-31 23:30:16.673	IOP	118.41999816894531	119.0999984741211	118.75999450683594
2017-01-31 23:30:16.673	AMZN	727.469970703125	727.469970703125	727.469970703125
2017-01-31 23:30:16.673	AMZN	713.0900268554688	727.469970703125	720.280029296875
2017-01-31 23:30:16.673	TBV	164.00999450683594	178.77999877929688	169.89332580566406

示例 2：引用命名窗口规范的 Row-Based 滑动窗口

此示例定义了一个指定的窗口规范，其分区边界为当前行前两到十行。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	AVERAGE_LAST2ROWS	AVERAGE_LAST10ROWS
2017-01-31 23:43:06.414	QXZ	75.33999633789062	114.7199935913086	100.65363311767578
2017-01-31 23:43:06.414	SLW	75.44000244140625	77.4233169555664	85.00454711914062
2017-01-31 23:43:06.414	SAC	41.709999084472656	42.95000076293945	44.18454360961914
2017-01-31 23:43:06.414	QWE	223.1999969482422	221.07000732421875	223.96910095214844
2017-01-31 23:43:06.414	WAS	14.039999961853027	13.993332862854004	13.739998817443848

示例 3：使用行内窗口规范的 Time-Based 滑动窗口

此示例定义了一个内联窗口规范，其分区边界为当前行前一分钟。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	PRICE	AVG_PRICE
2017-02-02 19:41:21.621	TBV	159.39999389648438	172.16751098632812
2017-02-02 19:41:21.621	JKL	15.300000190734863	15.295000076293945
2017-02-02 19:41:21.621	JKL	15.140000343322754	15.243332862854004
2017-02-02 19:41:21.621	QXZ	52.060001373291016	108.17571258544922
2017-02-02 19:41:21.621	WSB	8.430000305175781	8.569999694824219

使用说明

对于窗口子句和终端节点，亚马逊Kinesis Analytics SQL在一定范围内遵循窗口 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 允许和不允许的窗口规范。

相关主题

- 《Kinesis 开发人员指南》中的滑动窗口
- 聚合函数
- SELECT 语句
- CREATE STREAM statement
- CREATE PUMP statement

允许和不允许的窗口规范

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

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

- 无限窗口是指没有边界的窗口。通常指向未来，对于流来说是无限的。例如，不支持“ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING”，因为在流上下文中，此类查询不会生成结果，因为流会随着新数据的到达而不断扩展。不支持 UNBOUNDED FOLLOWING 的所有用法。
- 负数窗口。例如，“ROWS BETWEEN 0 PRECEDING AND 4 PRECEDING”是一个大小为负数的窗口，因此是非法的。在这种情况下，您可以改用“ROWS BETWEEN 4 PRECEDING AND 0 PRECEDING”。
- 偏移窗口是指不以 CURRENT ROW 结尾的窗口。当前版本不支持此类窗口。例如，不支持“ROWS BETWEEN UNBOUNDED PRECEDING AND 4 FOLLOWING”。（窗口跨越 CURRENT ROW，而不是从其开始或结束。）

- 用负整数定义的窗口。例如，“ROWS BETWEEN -4 PRECEDING AND CURRENT ROW”无效，因为不允许使用负整数。

此外，... 0 PRECEDING (and ... 0 FOLLOWING) 不能用于窗口式聚合，但可以使用同义词 CURRENT ROW。

对于窗口式聚合，允许使用分区式窗口，但不得存在 ORDER BY。

对于窗口式联接，不允许使用分区式窗口，但是如果按其中一个输入的 ROWTIME 列排序，则可以存在 ORDER BY。

窗口示例

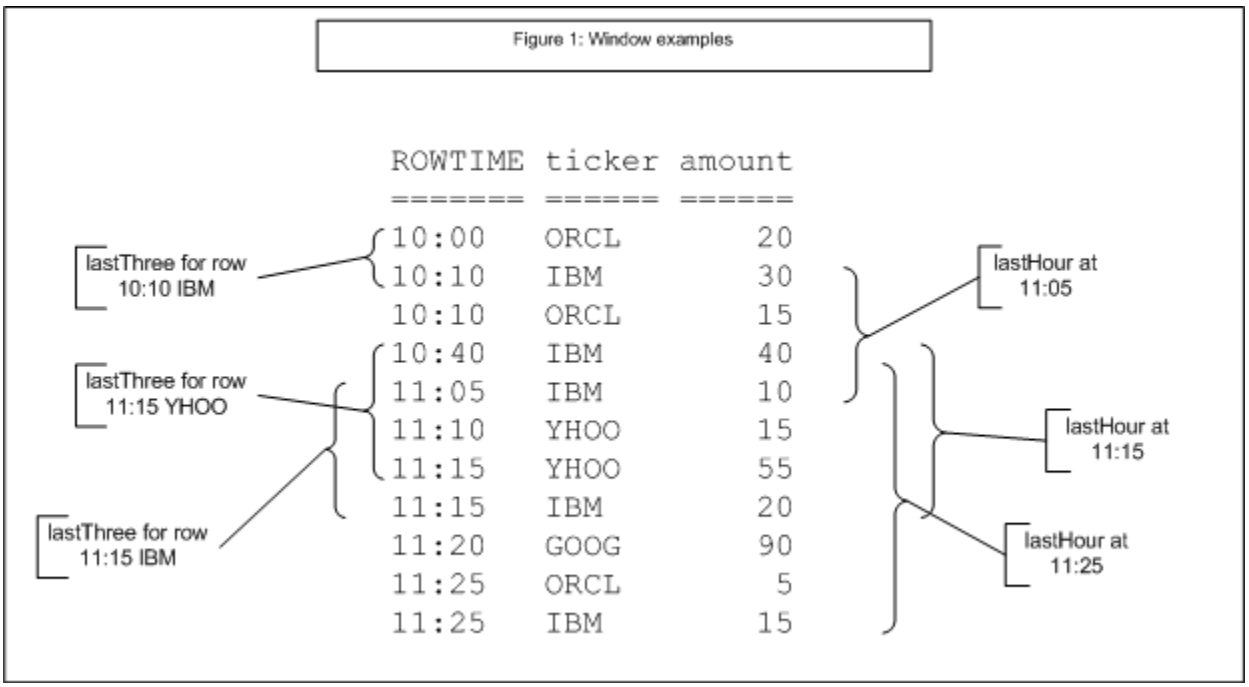
以下示例显示了样本输入数据集、多个窗口的定义以及这些窗口在 10:00 之后的不同时间（在本示例中，是数据开始到达的时间）的内容。

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。



包含行的窗口的示例

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

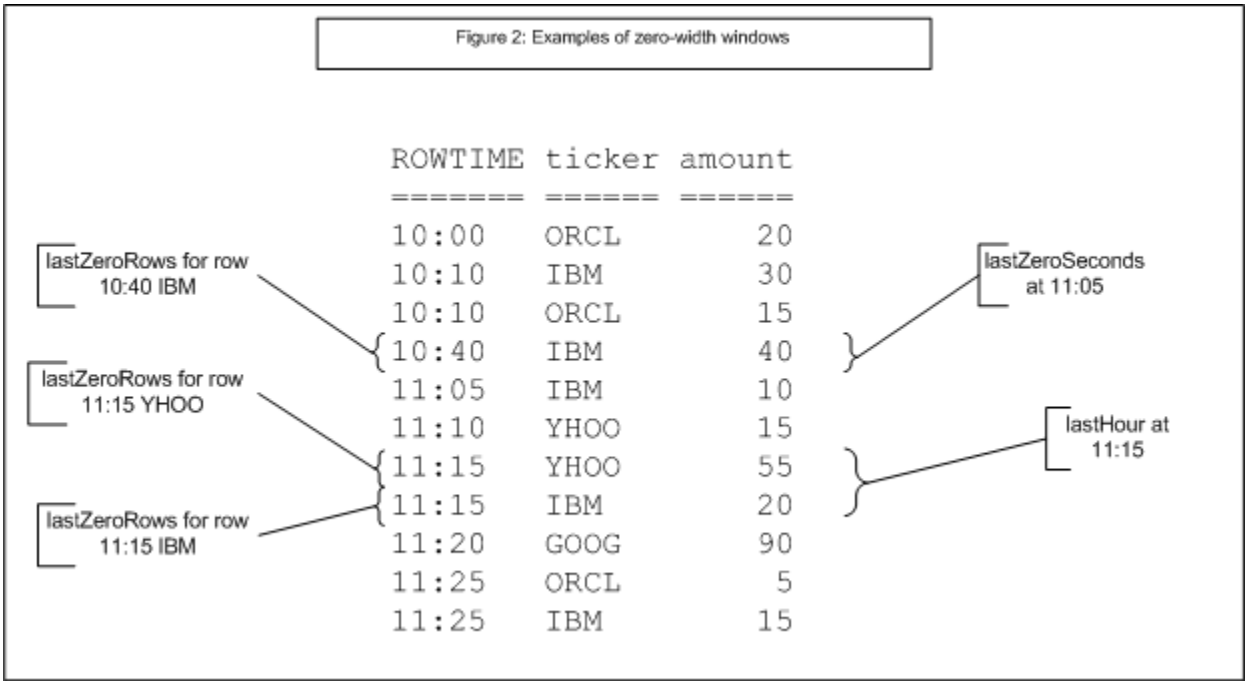
基于行的窗口可以包含多行，这些行的 ROWTIME 值相同，尽管它们到达的时间不同（挂钟时间）。此类行在基于行的窗口中的顺序取决于其到达时间；实际上，行的到达时间可以决定哪个窗口包含该行。

例如，图 1 中间的 lastThree 窗口显示了 YHOO 交易在 ROWTIME 为 11:15 时到达（及其之前的最后三笔交易）。但是，此窗口不包括下一笔交易，即 IBM，其 ROWTIME 也是 11:15，但必须晚于 YHOO 交易到达。此 11:15 IBM 交易包含在“下一个”窗口中，其前身 11:15 YHOO 交易也是如此。

第二个示例：基于行和基于时间的零宽度窗口

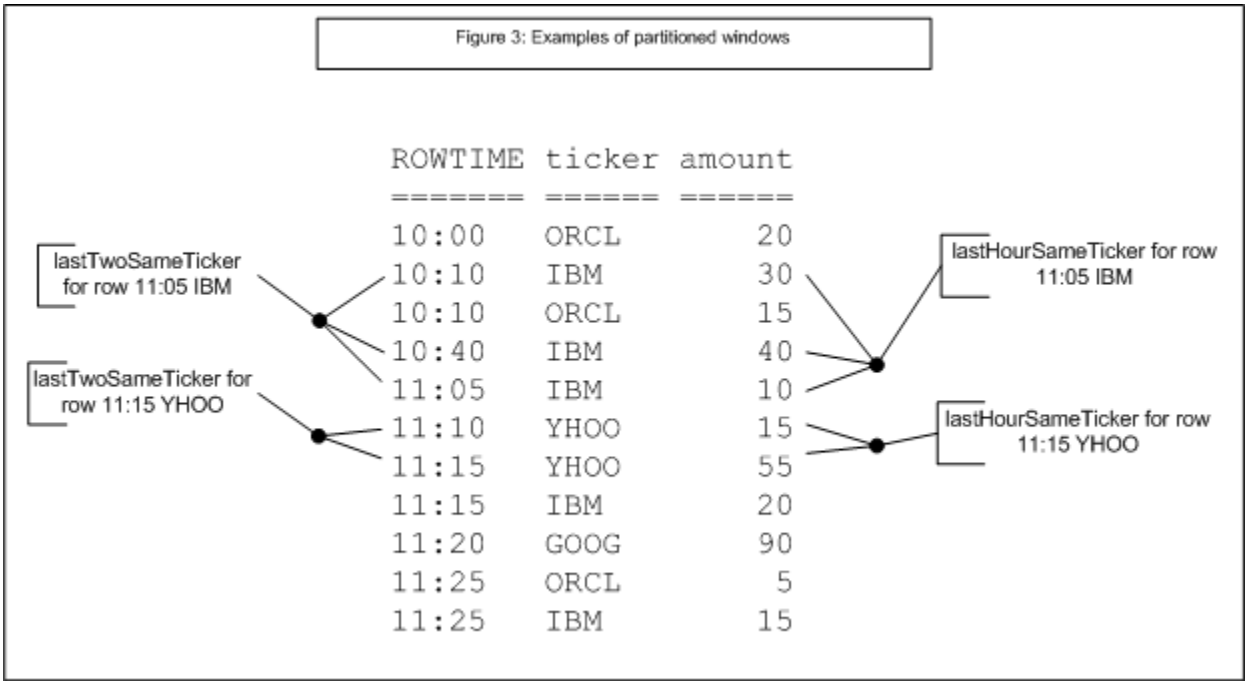
图 2：零宽度窗口的示例显示了宽度为零的基于行和基于时间的窗口。基于行的窗口最后仅 ZeroRows 包含当前行，因此始终只包含一行。请注意，ROWS CURRENT ROW 等同于 ROWS 0 PRECEDING。

基于时间的窗口最后ZeroSeconds 包含具有相同时间戳的所有行，其中可能有几行。请注意，RANGE CURRENT ROW 等同于 RANGE INTERVAL '0' SECOND PRECEDING。



第三个示例：分区适用于基于行和基于时间的窗口

图 3 显示的窗口与图 1 中的窗口类似，但带有 PARTITION BY 子句。对于基于时间的窗口最后一个 TwoSameTicker 和基于行的窗口最后一个HourSameTicker，该窗口包含符合窗口条件且与股票行列值相同的行。注意：分区是在窗口之前计算的。



ORDER BY 子句

如果流式查询的前导表达式基于时间且是单调的，则该查询可以使用 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：

- 对流式 GROUP BY 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。
- 对窗口式联接执行流式 ORDER BY。

针对前导表达式的“基于时间且单调”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。

串流 ORDER BY 使用 ORDER BY 子句的 SQL-2008 兼容语法对行进行排序。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的排序顺序，并且可以使用列序数以及指定（提及）选择列表中项目位置的序数。

> **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。

流式 ORDER BY SQL 声明

流式 ORDER BY 子句包括以下函数属性：

- 收集行，直到流式 ORDER BY 子句中的单调表达式不发生更改。
- 不要求流式 GROUP BY 子句在同一语句中。
- 可以使用任何基本 SQL 数据类型为 TIMESTAMP、DATE、DECIMAL、INTEGER、FLOAT、CHAR 和 VARCHAR 的列。
- 不要求 columns/expressions 在 ORDER BY 子句中存在于语句的 SELECT 列表中。
- 应用 ORDER BY 子句的所有标准 SQL 验证规则。

以下查询是流式 ORDER BY 的一个示例：

```
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-sorting 直播输入

Amazon Kinesis Data Analytics 实时分析依据以下事实：到达的数据按 ROWTIME 排序。但是，有时来自多个来源的数据可能在时间上不同步。

尽管 Amazon Kinesis Data Analytics 可以对来自各个数据来源并且已单独插入到 Amazon Kinesis Data Analytics 应用程序的本机流的数据进行排序，但在某些情况下，此类数据可能已从多个来源合并（例如为了在处理的早期阶段实现高效的使用）。在其他时候，大量数据来源可能会导致无法直接插入。

此外，不可靠的数据来源可能会迫使 Amazon Kinesis Data Analytics 应用程序无限期等待，在所有连接的数据来源交付之前无法继续，从而阻碍进程。在这种情况下，来自该来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 使用基于时间的滑动传入行窗口，按照 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>
```

限制

T-sort 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 列需要完全限定。例如：

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

如果未满足这些要求中的任何一个，该语句将因出错而失败。

附加注释：

- 您不能使用传入的行时间边界。系统会忽略这些内容。
- 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，返回流中特定行的创建时间。

有四种不同的使用方式：

- 作为运算符
- 作为流的系统列
- 作为列别名，用于覆盖当前行的时间戳
- 作为表中的普通列

有关更多详细信息，请参阅本指南中的“时间戳”、ROWTIME 和 CURRENT_ROW_TIMESTAMP 主题。

ROWTIME 运算符

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

类型始终是 TIMESTAMP NOT NULL。

ROWTIME 系统列

每个流都有一个 ROWTIME 列。要在查询中引用此列，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：其输入流的两个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
```

实际上，leftRowtime 始终等于 joinRowtime，因为指定联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 子句中明确包含了 ROWTIME 列，否则不会从顶级 JDBC 查询中返回该列。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：Orders 流声明了两列，因此“SELECT STREAM *”理应返回两列。

函数

本节中的主题介绍了流式 SQL 支持的函数。

主题

- 聚合函数
- 分析函数
- 布尔函数
- 转换函数
- 日期和时间函数
- Null 函数
- 数字函数
- 日志解析函数
- 排序函数
- 统计方差和偏差函数
- 流式处理 SQL 函数
- 字符串和搜索函数

聚合函数

聚合函数返回的不是根据单个行计算得出的结果，而是根据有限行集中包含的聚合数据或有限行集的相关信息计算得出的结果。聚合函数可能出现在以下任何一项中：

- SELECT 子句的 <selection list> 部分
- ORDER BY 子句
- HAVING 子句

聚合函数不同于分析函数，后者始终相对于必须指定的窗口进行计算，因此不能出现在 HAVING 子句中。本主题后面的表格中列出了其他区别。

使用聚合函数对表执行聚合查询和对流执行聚合查询的操作略有不同，如下所示。如果对表执行的聚合查询包含 GROUP BY 子句，则聚合函数会为输入行集中的每个组返回一个结果。缺少显式 GROUP BY 子句等同于 GROUP BY ()，只会为整个输入行集返回一个结果。

对于流，聚合查询必须在基于行时间的单调表达式上包含一个显式 GROUP BY 子句。否则，唯一的组就是整个流，永远不会结束，因此无法报告任何结果。基于单调表达式添加 GROUP BY 子句会将流分成在时间上连续的有限行集，然后可以聚合每个此类行集并生成相应报告。

每当一个会更改单调分组表达式值的行到达时，就会启动一个新组，而前一组被视为已完成。然后，Amazon Kinesis Data Analytics 应用程序会输出聚合函数的值。请注意，GROUP BY 子句还可能包括其他非单调表达式，在这种情况下可能会为每个行集生成多个结果。

对流执行聚合查询通常称为流式聚合，这与在分析函数和流的窗口式聚合中讨论的窗口式聚合不同。有关流到流联接的更多信息，请参阅 JOIN 子句。

如果输入行在用作数据分析函数输入的列中包含 `null`，则数据分析函数将忽略该行（COUNT 除外）。

聚合函数和分析函数之间的区别

函数类型	输出	使用的行或窗口	注意
聚合函数	每组输入行对应一个输出行。	所有输出列都是根据同一窗口或同一组行计算的。	流式聚合中不允许使用 COUNT DISTINCT。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。	可以使用不同窗口或分区计算每个输出列。	COUNT DISTINCT 不能用作分析函数或用在窗口式聚合中。

流式聚合和行时间边界

通常，当一个会更改 GROUP BY 中单调表达式值的行到达时，聚合查询就会生成一个结果。例如，如果查询按 FLOOR(rowtime TO MINUTE) 进行分组，而且当前行的行时间为 9:59.30，则行时间为 10:00.00 的新行将触发结果。

或者，可以使用行时间边限来推进单调表达式，使查询能够返回结果。例如，如果查询按 FLOOR(rowtime TO MINUTE) 进行分组，而且当前行的行时间为 9:59.30，则在传入的行时间边界为 10:00.00 时，查询会返回结果。

聚合函数列表

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

- AVG
- COUNT
- COUNT_DISTINCT_ITEMS_TUMBLING 函数
- EXP_AVG
- FIRST_VALUE
- LAST_VALUE
- MAX
- MIN
- SUM
- TOP_K_ITEMS_TUMBLING 函数

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

要查找每个部门员工的平均年龄，我们可以在查询中添加一个显式 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

对流执行的聚合查询的示例（流式聚合）

在此示例中，假设下表中的数据流经名为 WEATHERSTREAM 的流。

ROWTIME	CITY	TEMP
2018-11-01 01:00:00.0	丹佛	29
2018-11-01 01:00:00.0	安克雷奇	2
2018-11-01 06:00:00.0	迈阿密	65
2018-11-01 07:00:00.0	丹佛	32
2018-11-01 09:00:00.0	安克雷奇	9
2018-11-01 13:00:00.0	丹佛	50
2018-11-01 17:00:00.0	安克雷奇	10
2018-11-01 18:00:00.0	迈阿密	71
2018-11-01 19:00:00.0	丹佛	43
2018-11-02 01:00:00.0	安克雷奇	4

ROWTIME	CITY	TEMP
2018-11-02 01:00:00.0	丹佛	39
2018-11-02 07:00:00.0	丹佛	46
2018-11-02 09:00:00.0	安克雷奇	3
2018-11-02 13:00:00.0	丹佛	56
2018-11-02 17:00:00.0	安克雷奇	2
2018-11-02 19:00:00.0	丹佛	50
2018-11-03 01:00:00.0	丹佛	36
2018-11-03 01:00:00.0	安克雷奇	1

如果要查找每天任意地方（全球范围内，不分城市）记录的最低和最高温度，则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
2018-11-02 00:00:00.0	2	56

尽管示例数据确实包括 2018-11-03 的温度测量值，但没有这一天对应的行。这是因为在知道 2018-11-03 对应的所有行都到达之前，无法聚合这一天对应的行，而且只有在行时间为 2018-11-04 00:00:00.0（或更晚）或行时间边界为 2018-11-04 00:00:00.0（或更晚）的行到达时，才会进行聚合。如果其中任何一个到达，则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

假设我们不是要查找每天的全球最低和最高气温，而是要查找每天每个城市的最低、最高和平均温度。为此，我们使用 SUM 和 COUNT 聚合函数来计算平均值，然后将 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

FLOOR_DAY	CITY	MIN_TEMP	MAX_TEMP	AVG_TEMP
2018-11-02 00:00:00.0	安克雷奇	2	4	3
2018-11-02 00:00:00.0	丹佛	39	56	47

在本例中，当新一天的温度测量值对应的行到达时，会触发聚合前一天的数据，按 CITY 进行分组，然后为这一天的测量值中包含的每个城市生成一行。

同样，在 2018-11-04 的任何实际测量值出来之前，可以使用行时间边界 2018-11-04 00:00:00.0 来生成 2018-11-03 的结果，如下表所示。

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

流的窗口式聚合

为了说明窗口式聚合在 Amazon Kinesis Data Streams 中的工作原理，假定下表中的数据将流经一个名为 WEATHERSTREAM 的流。

ROWTIME	CITY	TEMP
2018-11-01 01:00:00.0	丹佛	29
2018-11-01 01:00:00.0	安克雷奇	2
2018-11-01 06:00:00.0	迈阿密	65
2018-11-01 07:00:00.0	丹佛	32

ROWTIME	CITY	TEMP
2018-11-01 09:00:00.0	安克雷奇	9
2018-11-01 13:00:00.0	丹佛	50
2018-11-01 17:00:00.0	安克雷奇	10
2018-11-01 18:00:00.0	迈阿密	71
2018-11-01 19:00:00.0	丹佛	43
2018-11-02 01:00:00.0	安克雷奇	4
2018-11-02 01:00:00.0	丹佛	39
2018-11-02 07:00:00.0	丹佛	46
2018-11-02 09:00:00.0	安克雷奇	3
2018-11-02 13:00:00.0	丹佛	56
2018-11-02 17:00:00.0	安克雷奇	2
2018-11-02 19:00:00.0	丹佛	50
2018-11-03 01:00:00.0	丹佛	36
2018-11-03 01:00:00.0	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 01:00:00.0	29	29
2018-11-01 01:00:00.0	2	29
2018-11-01 06:00:00.0	2	65
2018-11-01 07:00:00.0	2	65
2018-11-01 09:00:00.0	2	65
2018-11-01 13:00:00.0	2	65
2018-11-01 17:00:00.0	2	65
2018-11-01 18:00:00.0	2	71
2018-11-01 19:00:00.0	2	71
2018-11-02 01:00:00.0	2	71
2018-11-02 01:00:00.0	2	71
2018-11-02 07:00:00.0	4	71
2018-11-02 09:00:00.0	3	71
2018-11-02 13:00:00.0	3	71
2018-11-02 17:00:00.0	2	71
2018-11-02 19:00:00.0	2	56
2018-11-03 01:00:00.0	2	56
2018-11-03 01:00:00.0	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 01:00:00.0	丹佛	29	29	29
2018-11-01 01:00:00.0	安克雷奇	2	2	2
2018-11-01 06:00:00.0	迈阿密	65	65	65
2018-11-01 07:00:00.0	丹佛	29	32	30
2018-11-01 09:00:00.0	安克雷奇	2	9	5
2018-11-01 13:00:00.0	丹佛	29	50	37
2018-11-01 17:00:00.0	安克雷奇	2	10	7

ROWTIME	CITY	WMIN_TEMP	WMAX_TEMP	WAVG_TEMP
2018-11-01 18:00:00.0	迈阿密	65	71	68
2018-11-01 19:00:00.0	丹佛	29	50	38
2018-11-02 01:00:00.0	安克雷奇	2	10	6
2018-11-02 01:00:00.0	丹佛	29	50	38
2018-11-02 07:00:00.0	丹佛	32	50	42
2018-11-02 09:00:00.0	安克雷奇	3	10	6
2018-11-02 13:00:00.0	丹佛	39	56	46
2018-11-02 17:00:00.0	安克雷奇	2	10	4
2018-11-02 19:00:00.0	丹佛	39	56	46
2018-11-03 01:00:00.0	丹佛	36	56	45
2018-11-03 01:00:00.0	安克雷奇	1	4	2

行时间边界和窗口式聚合示例

以下是窗口式聚合查询的一个示例：

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

因为这是对流的查询，所以行一进入就会弹出此查询。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：

```
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流的整个历史进行总计，但这通常无法计算。但是，此类业务问题通常可以通过查看最后一天、最后一小时或最近 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：
>
> - 包含 ROWS 的整数文字
> - 数字列上的 RANGE 的数字值
> - 在 date/time /timestamp 上的 RANGE 间隔

AVG

从窗口式查询返回一组值的平均值。根据时间或行定义窗口式查询。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。

在使用 AVG 时，请注意以下事项：

- 如果您未使用 OVER 子句，则 AVG 将作为聚合函数进行计算。在这种情况下，聚合查询必须根据将流分组到有限行中的 ROWTIME 在单调表达式中包含 GROUP BY 子句。否则，组将是无限流，并且查询永远无法完成，也不会输出任何行。有关更多信息，请参阅 聚合函数。
- 使用 GROUP BY 子句的窗口式查询在滚动窗口中处理行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您使用 OVER 子句，则 AVG 将作为分析函数进行计算。有关更多信息，请参阅 分析函数。
- 使用 OVER 子句的窗口式查询在滑动窗口中处理行。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

划分流中按时间范围间隔或行数分区的记录。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 BY monotonic-expression | 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，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，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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 子句返回平均值

在此示例中，聚合查询在 ROWTIME 上有一个 GROUP BY 子句，可将流分组到有限行中。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	AVG_PRICE
2017-02-16 21:46:00.0	NFS	85.84222412109375
2017-02-16 21:47:00.0	WAS	16.45800018310547
2017-02-16 21:47:00.0	PPL	23.33571434020996
2017-02-16 21:47:00.0	ALY	76.0999984741211

示例 2：使用 OVER 子句返回平均值

在此示例中，OVER 子句划分流中按之前“1”小时的时间范围间隔分区的记录。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	AVG_PRICE
2017-02-16 21:57:25.509	AAPL	110.0180892944336
2017-02-16 21:57:25.509	TGT	54.96333312988281
2017-02-16 21:57:25.509	RFV	45.46332931518555
2017-02-16 21:57:25.509	SAC	17.972999572753906

使用说明

Amazon Kinesis Analytics 不支持应用于间隔类型的 AVG。此功能与 SQL:2008 标准背道而驰。

当用作分析函数时，如果所计算的窗口不包含任何行，或者所有行均包含 null 值，则 AVG 将返回 null。有关更多信息，请参阅分析函数。AVG 还为 PARTITION BY 子句返回 null，在这种情况下，窗口中匹配输入行的分区不包含任何行或所有行均为 null。有关 PARTITION BY的更多信息，请参阅WINDOW 子句 (滑动窗口)。

AVG 会忽略来自值集或数字表达式的 null 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
- 聚合函数
- GROUP BY 子句
- 分析函数
- 入门练习
- WINDOW 子句 (滑动窗口)

COUNT

从窗口式查询返回一组值的限定行的数目。根据时间或行定义窗口式查询。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：

- 如果您未使用 OVER 子句，则 COUNT 将作为聚合函数进行计算。在这种情况下，聚合查询必须根据将流分组到有限行中的 ROWTIME 在单调表达式中包含 GROUP BY 子句。否则，组将是无限流，并且查询永远无法完成，也不会输出任何行。有关更多信息，请参阅 聚合函数。
- 使用 GROUP BY 子句的窗口式查询在滚动窗口中处理行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您使用 OVER 子句，则 COUNT 将作为分析函数进行计算。有关更多信息，请参阅 分析函数。
- 使用 OVER 子句的窗口式查询在滑动窗口中处理行。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

划分流中按时间范围间隔或行数分区的记录。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 BY monotonic-expression | 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，后者是 Amazon Kinesis Analytics 开发人员指南中的入门的一部分。要运行每个示例，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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 子句返回值的数目

在此示例中，聚合查询在 ROWTIME 上有一个 GROUP BY 子句，可将流分组到有限行中。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	COUNT_PRICE
2017-02-16 23:30:00.0	AAPL	9.0
2017-02-16 23:31:00.0	WSB	8.0
2017-02-16 23:31:00.0	AAPL	6.0
2017-02-16 23:31:00.0	UHN	9.0

示例 2：使用 OVER 子句返回值的数目

在此示例中，OVER 子句划分流中按之前“1”小时的时间范围间隔分区的记录。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	COUNT_PRICE
2017-02-16 23:36:16.729	WMT	3.0
2017-02-16 23:36:16.729	DFG	2.0
2017-02-16 23:36:16.729	RFV	4.0
2017-02-16 23:36:16.729	TGH	3.0

使用说明

Amazon Kinesis Analytics 不支持 COUNT 函数的 FILTER 子句，也不支持在聚合函数或分析函数中使用 COUNT DISTINCT。有关聚合函数和分析函数的更多信息，请参阅聚合函数和分析函数。此功能与 SQL:2008 标准背道而驰。

在用作分析函数时，COUNT 将在所计算的窗口不包含任何行时返回零。有关更多信息，请参阅 分析函数。COUNT 还为 PARTITION BY 子句返回零，在这种情况下，窗口中匹配输入行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)。

COUNT 会忽略来自值集或数字表达式的 null 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
- GROUP BY 子句
- 分析函数
- 入门练习
- WINDOW 子句 (滑动窗口)

COUNT_DISTINCT_ITEMS_TUMBLING 函数

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

有关更多信息，请参阅 HyperLogLog。

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。
- 准确计数不同项目可能效率低下且成本高昂。因此，此函数估算计数。例如，如果有 100000 个不同的项目，则该算法可能会返回 99700 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一个五秒的滚动窗口中获取每个股票代码的不同行的准确计数。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。将滚动窗口的大小指定为参数。

语法

```
COUNT_DISTINCT_ITEMS_TUMBLING (

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

参数

以下各节介绍了这些参数。

in-application-streamPointer

使用此参数，您可以提供指向应用程序内流的指针。您可以使用 CURSOR 函数设置指针。例如，以下语句将设置指向 InputStream 的指针。

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

columnName

应用程序内流中您希望函数用来计数不同值的列名称。请注意有关列名称的以下内容：

- 必须用单引号 (') 括起来。例如 `'column1'`。

windowSize

滚动窗口的大小，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，后者是 Amazon Kinesis Analytics 开发人员指南中的入门的一部分。要运行每个示例，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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：粗略估计列中的唯一值的数量

以下示例演示了如何使用 COUNT_DISTINCT_ITEMS_TUMBLING 函数估算应用程序内流的当前滚动窗口中的不同 TICKER_SYMBOL 值的数目。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（指数移动平均值）。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，指定表达式的值的权重最大，而对于较早的间隔，权重则呈指数级降低。

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，因此最近 10 秒子窗口中（该股票代码对应的）价格的权重是中间 10 秒子窗口中价格权重的两倍，是最早的 10 秒子窗口中价格权重的四倍。

```
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下降两倍的时间段。换句话说，较早的权重是较近的权重的一半。在调用 EXP_AVG 时，time_interval 被指定为 interval '10' second。

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 需要使用 OVER 子句，因此被视为分析函数。FIRST_VALUE 有一个 null 处理选项，定义在下表中。

Null 处理选项	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 处理选项	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

Null 处理选项	效果
LAST_VALUE(x) OVER <window-specification>	返回最后一个值，包括 <window-specification> 中 x 的 null 值

MAX

从窗口式查询返回一组值的最大值。根据时间或行定义窗口式查询。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：

- 如果您未使用 OVER 子句，则 MAX 将作为聚合函数进行计算。在这种情况下，聚合查询必须根据将流分组到有限行中的 ROWTIME 在单调表达式中包含 GROUP BY 子句。否则，组将是无限流，并且查询永远无法完成，也不会输出任何行。有关更多信息，请参阅 聚合函数。
- 使用 GROUP BY 子句的窗口式查询在滚动窗口中处理行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您使用 OVER 子句，则 MAX 将作为分析函数进行计算。有关更多信息，请参阅 分析函数。
- 使用 OVER 子句的窗口式查询在滑动窗口中处理行。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

划分流中按时间范围间隔或行数分区的记录。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 BY monotonic-expression | 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，后者是 Amazon Kinesis Analytics 开发人员指南中的入门的一部分。要运行每个示例，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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，聚合查询在 ROWTIME 上有一个 GROUP BY 子句，可将流分组到有限行中。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	MAX_PRICE
2017-02-17 00:34:00.0	ASD	67.19999694824219
2017-02-17 00:35:00.0	BNM	179.3300018310547
2017-02-17 00:35:00.0	PPL	39.47999954223633
2017-02-17 00:35:00.0	QXZ	222.89999389648438

示例 2：使用 OVER 子句返回最大值

在此示例中，OVER 子句划分流中按之前“1”小时的时间范围间隔分区的记录。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	MAX_PRICE
2017-02-17 00:36:44.854	QAZ	198.2100067138672
2017-02-17 00:36:50.796	QXZ	232.49000549316406
2017-02-17 00:36:50.796	MJN	187.6999969482422
2017-02-17 00:36:50.796	WSB	101.0

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，并且所计算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。

相关主题

- 窗口式查询
- 聚合函数
- GROUP BY 子句
- 分析函数
- 入门练习
- WINDOW 子句 (滑动窗口)

MIN

从窗口式查询返回一组值的最小值。根据时间或行定义窗口式查询。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：

- 如果您未使用 OVER 子句，则 MIN 将作为聚合函数进行计算。在这种情况下，聚合查询必须根据将流分组到有限行中的 ROWTIME 在单调表达式中包含 GROUP BY 子句。否则，组将是无限流，并且查询永远无法完成，也不会输出任何行。有关更多信息，请参阅 聚合函数。
- 使用 GROUP BY 子句的窗口式查询在滚动窗口中处理行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您使用 OVER 子句，则 MIN 将作为分析函数进行计算。有关更多信息，请参阅 分析函数。
- 使用 OVER 子句的窗口式查询在滑动窗口中处理行。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

划分流中按时间范围间隔或行数分区的记录。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 BY monotonic-expression | 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，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，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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，聚合查询在 ROWTIME 上有一个 GROUP BY 子句，可将流分组到有限行中。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	MIN_PRICE
2017-02-17 22:45:00.0	QXZ	48.970001220703125
2017-02-17 22:46:00.0	WMT	62.540000915527344
2017-02-17 22:46:00.0	QWE	214.2100067138672
2017-02-17 22:46:00.0	CRM	28.260000228881836

示例 2：使用 OVER 子句返回最小值

在此示例中，OVER 子句划分流中按之前“1”小时的时间范围间隔分区的记录。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	MIN_PRICE
2017-02-17 22:49:51.147	NFS	84.94000244140625
2017-02-17 22:49:59.058	NFLX	112.37000274658203
2017-02-17 22:49:59.058	ASD	57.790000915527344
2017-02-17 22:49:59.058	DFG	147.3000030517578

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，并且所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。

相关主题

- 窗口式查询
- 聚合函数
- GROUP BY 子句
- 分析函数
- 入门练习
- WINDOW 子句 (滑动窗口)

SUM

从窗口式查询返回一组值的和。根据时间或行定义窗口式查询。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：

- 如果您未使用 OVER 子句，则 SUM 将作为聚合函数进行计算。在这种情况下，聚合查询必须根据将流分组到有限行中的 ROWTIME 在单调表达式中包含 GROUP BY 子句。否则，组将是无限流，并且查询永远无法完成，也不会输出任何行。有关更多信息，请参阅 聚合函数。
- 使用 GROUP BY 子句的窗口式查询在滚动窗口中处理行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您使用 OVER 子句，则 SUM 将作为分析函数进行计算。有关更多信息，请参阅 分析函数。
- 使用 OVER 子句的窗口式查询在滑动窗口中处理行。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

划分流中按时间范围间隔或行数分区的记录。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 BY monotonic-expression | 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，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，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要

了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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 子句返回值的和

在此示例中，聚合查询在 ROWTIME 上有一个 GROUP BY 子句，可将流分组到有限行中。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	SUM_PRICE
2017-02-18 00:28:00.0	KIN	32.630001068115234
2017-02-18 00:29:00.0	VVS	50.409996032714844
2017-02-18 00:29:00.0	HJK	34.81999969482422
2017-02-18 00:29:00.0	KFU	356.6899719238281

使用说明

Amazon Kinesis Analytics 不支持应用于间隔类型的 SUM。此功能与 SQL:2008 标准背道而驰。

SUM 会忽略来自值集或数字表达式的 null 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
- GROUP BY 子句
- 分析函数
- 入门练习
- WINDOW 子句 (滑动窗口)

TOP_K_ITEMS_TUMBLING 函数

在滚动窗口中返回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（最热门）值。

例如，入门练习使用演示流来提供持续的股价更新（股票代码、价格、变动和其他列）。假设您想在每个 1 分钟的滚动窗口中找到交易最频繁的三只股票。您可以使用此函数来查找这些股票代码。

在使用 TOP_K_ITEMS_TUMBLING 时，请注意以下事项：

- 计数流式来源上的每条传入记录效率不高，因此该函数将估算最常出现的值。例如，在寻找交易量最大的三只股票时，该函数可能会返回交易量最大的五只股票中的三只。

此函数在一个滚动窗口内运行。您需要将窗口大小指定为参数。

有关分步指导的示例应用程序，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-streamPointer

指向应用程序内流的指针。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

应用程序内流中您希望用来计算 topK 值的列名称。请注意有关列名称的以下内容：

> **Note**
>
> 列名称必须用单引号 (') 括起来。例如 `'column1'`。

K

使用此参数，您可以指定希望从特定列中返回多少个最常出现的值。值 K 必须大于或等于 1 且不能超过 100000。

windowSize

滚动窗口的大小，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，后者是 Amazon Kinesis Analytics 开发人员指南中的入门的一部分。要运行每个示例，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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_VALUES
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

分析函数会返回根据由 SELECT 子句标识或在 ORDER BY 子句中的有限行集中的数据（或该有限行集的相关数据）计算得出的结果。

SELECT 主题解释了 order-by 子句，显示了 order-by 图表以及窗口化子句（和 window-specification 图表）。要查看 Select 语句中使用 order-by 子句的位置，请参阅本指南 SELECT 主题中的 Select 图表。

1. 分析函数必须指定一个窗口。由于对窗口规范有一些限制，并且指定用于窗口式聚合和窗口式联接的窗口之间存在一些区别，因此请参阅允许和不允许的窗口规范以获取解释。
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列出了其他区别。

使用分析函数执行查询通常被称为窗口式聚合（如下所述），与聚合函数不同。

由于存在窗口规范，因此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，窗口规范都标识了分析函数对其进行操作的不同行集。如果窗口规范还包含 PARTITION BY 子句，则在生成结果时，将只考虑窗口中与输入行共享同一分区的行。

如果输入行在用作分析函数输入的列中包含 null，则分析函数将忽略该行，但 COUNT 除外，它会计数具有 null 值的行。如果窗口（或者窗口中的分区 PARTITION BY）不包含任何行，则分析函数将返回 null。但 COUNT 除外，将返回零。

聚合函数和分析函数之间的区别

函数类型	输出	使用的行或窗口	注意
聚合函数	每组输入行对应一个输出行。	所有输出列都是根据同一窗口或同一组行计算的。	不允许在聚合函数中使用 COUNT DISTINCT。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。	可以使用不同窗口或分区计算每个输出列。	COUNT DISTINCT 不能用作分析函数或用在窗口式聚合中。

相关主题

- 流的窗口式聚合
- SELECT 语句
- SELECT 子句

布尔函数

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

主题

- ANY
- EVERY

ANY

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

如果提供的 boolean_expression 在任一选定行中为 true，则 ANY 将返回 true。如果提供的 boolean_expression 在所有选定行中均不为 true，则将返回 false。

示例

如果交易流中任一代码对应的价格低于 1，则以下 SQL 代码段将返回“true”。如果流中的每个价格都等于或大于 1，则将返回“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 true，则 EVERY 将返回 true。如果提供的 boolean_expression 在任一选定行中为 false，则将返回 false。

示例

如果交易流中所有代码对应的价格均低于 1，则以下 SQL 代码段将返回“true”。如果任何价格等于 1 或更高，则返回“false”。

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

转换函数

本节中的主题介绍了 Amazon Kinesis Data Analytics 流式 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。

主题

- CAST

CAST

CAST 允许您将一种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

对下面第一列中列出的源运算对象类型使用 CAST，可以创建第二列中列出的转换目标类型，没有任何限制。不支持其他目标类型。

源运算对象类型	目标运算对象类型
任何数字类型（NUMERIC、DECIMAL、SMALLINT、INTEGER、BIGINT、REAL、DOUBLE）	VARCHAR、CHAR 或任何数字类型（参阅注释 A。）
VARCHAR、CHAR	以上所有内容，再加上日期、时间、时间戳、DAY-TIME 间隔、布尔值
DATE	DATE、VARCHAR、CHAR、TIMESTAMP
TIME	TIME、VARCHAR、CHAR、TIMESTAMP
TIMESTAMP	TIME、VARCHAR、CHAR、TIMESTAMP、DATE
DAY-TIME 间隔	DAY-TIME 间隔、BIGINT、十进制、字符、VARCHAR
BOOLEAN	VARCHAR、CHAR、BOOLEAN
BINARY、VARBINARY	BINARY、VARBINARY

示例

2.1 日期至 CHAR/VARCHAR

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

（请注意，如果提供的输出规范不充分，则不会选择任何行：

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

（因为日期文字需要包含 10 个字符）

在下一个示例中，日期的右侧填充的是空白（由于 CHAR 数据类型的语义）：

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

将 REAL 转换为 INTEGER

实数（NUMERIC 或 DECIMAL）在转换时会四舍五入：

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

将 STRING 转换为 TIMESTAMP

将字符串转换为时间戳的方式有两种。第一种是使用 CAST，如下一个主题所示。另一种是使用 Char To Timestamp(Sys)。

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、时、分、秒）中的任何一个，或者使用了任何与上述不同的分隔符，则 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，则要将该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。该方法的另一个优点是，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，并创建一个仅包含这些部分的 TIMESTAMP 值。为此，您需要指定一个模板来标识所需部分，例如“yyyy-MM”，以便仅使用年和月这两个部分。

待转换的输入日期时间字符串可以包含完整时间戳的全部或任何部分，即任何或所有标准元素的值（'yyyy-hh: mm: ss'）。MM-dd 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、小时、分钟和秒，例如在 '2009-09-16 03:15:24 '中。yyyy 不能大写；hh 可以大写，表示使用 24 小时制。有关有效说明符的许多示例，请参阅本主题后面的表格和示例。有关所有有效说明符，请参阅 Oracle 网站 SimpleDateFormat上的类别。

CHAR_TO_TIMESTAMP 使用您指定的模板作为函数调用中的参数。该模板使 TIMESTAMP 结果仅使用您在模板中指定的 input-date-time 值的一部分。然后，生成的 TIMESTAMP 中的这些字段将包含取自 input-date-time 字符串的相应数据；模板中未指定的字段将使用默认值（请参阅下文）。CHAR_TO_TIMESTAMP 使用的模板格式由 Class 定义，该链接列出了所有说明符 SimpleDateFormat，其中一些说明符附有示例。有关更多信息，请参阅日期和时间模式。

函数调用语法如下：

```
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

- 位置与 MM 对应的输入字符串元素必须是介于 1 到 12 之间的整数，因为其他任何内容都不能代表有效的月。
- 位置与 dd 对应的输入字符串元素必须是介于 1 到 31 之间的整数，因为其他任何内容都不能代表有效的日。
- 但是，如果 MM 为 2，则 dd 不能是 30 或 31，因为 2 月没有这几日。

但是，对于月或日，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 TIMESTAMP，其他字段（例如时、分或秒）为零，方法是指定以下内容：

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

结果将是 TIMESTAMP 2009-09-16 00:00:00。

示例 2

- 如果调用在模板中保留了时和分，但省略了月、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 TIMESTAMP 将是 2009-01-01 03:15:00。

模板	输入字符串	输出TIMESTAMP	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 'yyy-MM-dd hh: mm: ss'的形式或其子集或重新排序；使用像

模板	输入字符串	输出TIMESTAMP	注意
			“2009 年 9 月 16 日星期三 03:15:24” 这样的输入字符串将不起作用，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个是年和第二个是分，因此输入字符串中的第二个元素（“02”）用作分。 月和日使用默认值（“01”），时和秒使用默认值（“00”）。
'yyyy-ss-mm'	'2012-02-08 07:23:19'	'2012-01-01 00:08:02'	上面的模板仅指定了年、秒和分，按照该顺序，因此输入字符串中的第二个元素（“02”）用作秒，第三个元素（“08”）用作分。月和日使用默认值（“01”），时使用默认值（“00”）。

模板	输入字符串	输出TIMESTAMP	注意
'MMM dd, yyyy'	'March 7, 2010'	'2010-03-07 00:00:00'	上面模板中的 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，输入字符串也必须缺少逗号，否则就没有输出； --- --- 如果输入字符串缺少逗号，模板也必须缺少逗号。
'MMM dd,'	'March 7, 2010'	'1970-03-07 00:00:00'	请注意，上面的模板没有使用年说明符，这会导致输出 TIMESTAMP 使用这个时代最早的一年，即 1970。
'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“March 7, 10”，则输出 TIMESTAMP 将是“0010-03-07 00:00:00”。

模板	输入字符串	输出TIMESTAMP	注意
'M-d'	'2-8'	'1970-02-08 00:00:00'	如上所示，模板中没有 yyyy 说明符，因此使用这个时代最早的一年 (1970)。 输入字符串“2-8 -2012”将生成相同的结果；使用“2012-2-8”不会生成任何结果，因为 2012 不是有效的月。
'MM-dd-yyyy'	'06-23-2012 10:11:12'	'2012-06-23 00:00:00'	如果模板和输入都在相同位置使用短划线作为分隔符（如上所示），则可以使用短划线。由于模板省略了时、分和秒，因此在输出 TIMESTAMP 中都使用零。
'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这些说明符，但前提是该顺序与您提供的输入字符串的含义匹配，如上所示。下面下一个示例的模板和输入字符串与此示例具有相同的含义（和相同的输出 TIMESTAMP），但在日之前指定了月，在时之前指定了秒。

模板	输入字符串	输出TIMESTAMP	注意
'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，月和日说明符的顺序与上面的示例相反，秒的说明符在时之前而不是分之后；但是由于输入字符串还将月放在日之前，将秒放在时之前，因此含义（和输出 TIMESTAMP）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'	上面使用的模板颠倒了（与上面的前一个示例相比）年和月说明符，而输入字符串保持不变。在本例中，输出 TIMESTAMP 使用输入字符串的第一个元素作为年，第二个元素作为日，第三个元素作为月。

模板	输入字符串	输出TIMESTAMP	注意
'dd-MM-yy hh:mm'	'23-06-11 10:11:12'	'2011-06-23 10:11:00'	如上所示，模板中省略了秒，输出 TIMESTAMP 使用 00 秒。任意数量的 y 说明符都会生成相同的结果；但是如果输入字符串无意中对年使用的是 1 而不是 11，例如在“23-06-1 10:11:12”中，则输出 TIMESTAMP 将变为“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'	如果模板和输入都在相同位置使用斜杠作为分隔符（如上所示），则可以使用斜杠；否则，无输出。 使用说明符 hh，输入时间“12:11:10”和“00:11:10”与上午的时间含义相同。

模板	输入字符串	输出TIMESTAMP	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，就像模板字符串 “” 一样。required/expected MM/dd/yy HH:mm:ss
			
			但是，更改模板会生成有用的输出：
			
			`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

模板	输入字符串	输出TIMESTAMP	注意
			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，也就是说，对于下午和晚上的时间，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，时从 00 到 23。 --- --- 使用说明符 HH，输入时间“12:11:10”和“00:11:10”具有不同的含义，第一个是下午的时间，第二个是上午的时间。 --- --- 使用说明符 hh，从 12:00 到 11:59:59 的时间是上午的时间： --- --- 使用说明符 hh:mm:ss，对于输入字符串“12:09:08”和“00:09:08”，输出 TIMESTAMP 都将包括上午的“00:09:08”； --- --- 而

模板	输入字符串	输出TIMESTAMP	注意
			-----给定说明符 HH:mm:ss，早上输入字符串 '00:09:08' 的输出时间戳将包括 '00:09:08' --- --- 而对于下午的输入字符串“12:09:08”，输出 TIMESTAMP 将包括“12:09:08”。

更多示例

以下示例说明了如何将各种模板与 CHAR_TO_TIMESTAMP 搭配使用，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，上述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，由模板和输入字符串中作为分隔符的短划线定义：先是年，后是时，最后是分。由于模板中不存在月和日的说明符，因此其在输入字符串中的值被忽略，输出 TIMESTAMP 中的这两个值都替换为 01。模板指定了时和分作为第二和第三个输入值，因此 09 成为了时，16 成为了分。秒没有说明符，因此使用了 00。

年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说明符结束，其中一个是 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
values (CHAR_TO_TIMESTAMP('yyyy hh','2009-09-16 03:15:24') );
'EXPR$0'
No rows selected
```

上面的模板之所以失败，是因为在“hh”之前有一个空格作为分隔符，而不是输入字符串的日期规范中使用的短划线分隔符；

而下面的四个模板之所以起作用，是因为使用相同的分隔符将年说明符与下一个说明符分隔，就像在输入字符串的日期规范中使用的那样（第一个示例中是短划线，第二个示例中是空格，第三个示例中是斜杠，第四个示例中是短划线）。

```
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)，则除非还指定了日，否则将无法指定时、分或秒：

模板仅指定年份和月份，因此在生成的 TIMESTAM days/hours P minutes/seconds 中省略了/：

```
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了，缺少“日”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，使用了“日”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” 仅指定小时 (hh)，没有指定分钟或秒。由于 hh 是模板 token/element 的第 4 个，因此其值将取自输入字符串 “2009-09-16 03:15:24” 的第 token/element 4 个；第 4 个元素是 03，然后用作小时的输出值。由于未指定 mm 或 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 'yyy-MM-dd ss' 指定将输入字符串 token/element 的第 4 个用作秒 (ss)。输入字符串“2009-09-16 03:15:24”的第 4 个元素是 03，将成为模板中指定的秒的值输出；而且由于模板中未指定 hh 和 mm，因此使用默认值或初始值，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 token/element 的第 4 个用作分钟 (mm)。输入字符串“2009-09-16 03:15:24”的第 4 个元素是 03，将成为模板中指定的分的值输出；而且由于模板中未指定 hh 和 ss，因此使用默认值或初始值，即零。

更多失败，缺少“日”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匹配；输入字符串中的值必须是其对应的模板说明符可以接受的。

按照一般惯例，冒号用于分隔时和分以及分和秒。同样，一般惯例是使用短划线或斜杠分隔年和月以及月和日。任何并行用法似乎都行，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
```

上面的示例之所以失败，是因为 2009 不是可接受的月值，月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，是因为分隔符是并行的（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以接受的。

```
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，是因为分隔符是并行的（所有斜杠），并且每个值对于相应的说明符都是可以接受的。

```
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，是因为分隔符是并行的，并且值对于每个说明符都是可以接受的。

在以下示例中，请注意，提供的字符串中的省略可能会导致模板值“yyyy”生成合乎逻辑但意想不到或出乎意料的结果。在生成的 TIMESTAMP 值中作为年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
```

将 TIMESTAMP 转换为 STRING

```
 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 要求的完整格式必须 TIMESTAMP-literal 为 'yyyy-mm-dd hh: mm: ss '。如果缺少该完整格式的任何部分，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，如果提供的输出规范不充分，则不会选择任何行：

```
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 CAST 为 TIME 或 DATE 类型。

将 STRING 转换为 TIME

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

有关更多信息，请参阅注释 A。

将 STRING 转换为 DATE

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

> **注释 A**
>
> 请注意，对字符串进行 CAST 要求用于转换为 TIME 或 DATE 的字符串运算对象采用分别表示 TIME 或 DATE 所需的确切格式。

如下所示，在以下情况下，转换将失败：

- 字符串运算对象包括与目标类型无关的数据，或者
- INTERVAL 运算对象 ('day hours:minutes:seconds.milliseconds') 未包含必要的数据，或者
- 指定的输出字段太小，无法保留转换结果。

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

失败，因为包含不允许作为 TIME 的日期信息。

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

失败，因为包含不允许作为 DATE 的时间信息。

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

失败，因为包含不允许作为 DATE 的时间信息。

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

成功，因为包含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，因为在转换为 DATE 之前正确地将字符串转换为了 TIMESTAMP。

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

失败，因为缺少 TIME 所需的时间信息（秒）。

（允许指定小数秒，但并非必需。）

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

失败，因为包含不正确的小数秒表示形式。

```
 values(cast('21:23:34.11' AS TIME));
'EXPR$0'
'21:23:34'
1 row selected
```

成功，因为包含正确的小数秒表示形式。

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

此示例之所以成功，是因为包含正确的秒表示形式，而没有小数秒。

将 INTERVAL 转换为确切数字

对间隔进行 CAST 要求 INTERVAL 运算对象中只有一个字段，例如 MINUTE、HOUR、SECOND。

如果 INTERVAL 运算对象有多个字段，例如 MINUTE TO SECOND，则转换将失败，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，不允许将小数作为输入和输出。

```
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 不支持直接将数字值转换为间隔值。这与 SQL:2008 标准背道而驰。将数字转换为间隔的推荐方法是将数字值乘以特定的间隔值。例如，将整数 time_in_millis 转换为 day-time 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相关。

主题

- 时区
- 日期时间转换函数
- 日期、时间戳和间隔运算符
- 日期和时间模式
- CURRENT_DATE
- CURRENT_ROW_TIMESTAMP
- CURRENT_TIME
- CURRENT_TIMESTAMP
- EXTRACT
- LOCALTIME
- LOCALTIMESTAMP
- TSDIFF

其中，SQL 扩展 CURRENT_ROW_TIMESTAMP 对流上下文最有用，因为向您提供有关流数据出现时间的相关信息，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“打开”状态，生成更多数据，因此查询运行时间的时间戳无法提供有效信息。

LOCALTIMESTAMP、LOCALTIME、CURRENT_DATE 和 CURRENT_TIMESTAMP 生成的结果都会设置为查询首次执行时的值。只有 CURRENT_ROW_TIMESTAMP 生成一行，每行都有唯一的时间戳（日期和时间）。

以 LOCALTIMESTAMP（或 CURRENT_TIMESTAMP 或 CURRENT_TIME）作为其中一列运行的查询在查询首次运行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

时区

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

日期时间转换函数

您可以使用模式化字母来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”和从“a”到“z”的未加引号的字母，每个字母都代表一个格式元素。

有关更多信息，请参阅 Oracle 网站 SimpleDateFormat上的课程。

> **Note**
>
> 如果包含其他字符，则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。

下表中的模式字母已定义（从“A”到“Z”和从“a”到“z”的所有其他字符均已保留）。

字母	日期或时间组件	呈现方式	示例
y	Year	Year	yyyy; yy 2018;18
Y	周 年	Year	YYYY; YY 2009; 09
M	年中某月	Month	MMM;MM;MM July; Jul; 07

字母	日期或时间组件	呈现方式	示例
w	年中某周	数字	ww; 27
W	月中某周	数字	W 2
D	年中某日	数字	DDD 321
d	月中某日	数字	dd 10
F	月中某周某日	数字	F 2
E	周中某日名称	文本	Tuesday; Tue
u	周中某日编号（1 = 星期一……7 = 星期日）	数字	1
a	Am/pm 标记	文本	PM
H	一天中的时 (0-23)	数字	0
k	一天中的时 (1-24)	数字	24
K	一小时后 am/pm (0-11)	数字	0
h	一小时后 am/pm (1-12)	数字	12
m	时中的分	数字	30
s	分中的秒	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

字母	日期或时间组件	呈现方式	示例
X	时区	ISO 8601 时区	-08; -0800; -08:00

您可以按照 YYYY 的思路通过重复模式字母来确定确切的呈现方式。

文本

如果重复的模式字母数为 4 或更多，则使用完整形式；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，与模式字母数无关。

数字

格式化时，模式字母数是最小位数，较短的数字将用零填补到此数量。解析时，除非需要分隔两个相邻字段，否则模式字母数将被忽略。

Year

如果格式化程序的日历是公历，则适用以下规则。

- 格式化时，如果模式字母数为 2，则年将截断为 2 位数；否则将被解释为数字。
- 解析时，如果模式字母数大于 2，则年按字面解释，与位数无关。因此，使用模式 MM/dd/yyyy “”，“01/11/12” 可以解析到 12 年 1 月 11 日。 A.D

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创建 SimpleDateFormat 实例前 80 年和创建后 20 年之内来实现此目的。例如，使用 “MM/dd/yy” 模式和在 2018 年 1 月 1 日创建的 SimpleDateFormat 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) 定义的恰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全是数字的两位数字符串（例如“-1”），均按字面解释。因此，使用与01/02公元 3 年 1 月 2 日相同的模式解析 “01/02/3” 或 “/003”。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格式。在格式化和解析时，如果模式字母数为 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格式。

Char To Timestamp(Sys)

Char to Timestamp 函数是最常用的系统函数之一，因为允许您使用任何格式正确的输入字符串创建时间戳。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，并创建一个仅包含这些部分的 TIMESTAMP 值。为此，您需要指定一个模板来标识所需时间戳的各个部分。例如，要仅使用年和月，则指定“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-hh: mm: ss'）。MM-dd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、小时、分钟和秒，例如在 '2009-09-16 03:15:24 '中。yyyy 不能大写；hh 可以大写，表示使用 24 小时制。

有关所有有效说明符，请参阅 Oracle 网站 SimpleDateFormat上的类别。

CHAR_TO_TIMESTAMP 使用您指定的模板作为函数调用中的参数。该模板使 TIMESTAMP 结果仅使用您在模板中指定的 input-date-time 值的一部分。生成的 TIMESTAMP 中的这些字段包含取自 input-date-time 字符串的相应数据。未在模板中指定的字段将使用默认值（请参阅下文）。CHAR_TO_TIMESTAMP 使用的模板格式由甲骨文网站上的学 SimpleDateFormat员定义。有关更多信息，请参阅日期和时间模式。

函数调用语法如下：

```
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，位置与 MM 对应的输入字符串元素必须是介于 1 到 12 之间的整数，因为其他任何内容都不能代表有效的月。同样，位置与 dd 对应的输入字符串元素必须是介于 1 到 31 之间的整数，因为其他任何内容都不能代表有效的日。（但是，如果 MM 为 2，则 dd 不能是 30 或 31，因为 2 月没有这几日。）

对于时、分或秒，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。对于月或日，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 TIMESTAMP，其他字段（例如时、分或秒）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。

如果调用在模板中保留了时和分，但省略了月、日和秒，如以下调用所示。

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

然后，生成的 TIMESTAMP 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显示了用于创建指定输出 TIMESTAMP 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'yyy-MM-dd hh: mm: ss'的形式或其子集或重新排序。因此，使用“Wednesday, 16 September 2009 03:15:24”这样的输入字符串将不起作用，这意味着不会生成任何输出。

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匹配，并且输入字符串中的值必须是其对应的模板说明符可以接受的。

按照一般惯例，冒号用于分隔时和分以及分和秒。同样，一般惯例是使用短划线或斜杠分隔年和月以及月和日。

例如，以下模板的值与输入字符串正确一致。

```
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是其对应的模板说明符不可接受的，则结果将失败，如下例所示。

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

此示例未返回任何行，因为 2009 不是可接受的月值，月是模板中的第一个说明符 (MM)。

提供的字符串中的省略可能会导致模板值“yyyy”生成合乎逻辑但意想不到或出乎意料的结果。以下示例返回的均是错误的年，但其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 TIMESTAMPS 的示例

模板的顺序必须与输入字符串匹配。这意味着您不能在“yyyy”之后指定“hh”，也不能期望该方法自动找到时。例如，以下模板先指定年，后指定时，最后指定分，但返回错误的结果。

```
 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，因此其在输入字符串中的值被忽略，输出 TIMESTAMP 中的这两个值都替换为 01。模板指定了时和分作为第二和第三个输入值，因此 09 成为了时，16 成为了分。秒没有说明符，因此使用了 00。

年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说明符结束，其中一个是 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，下面的模板之所以失败，是因为在“hh”之前有一个空格作为分隔符，而不是输入字符串的日期规范中使用的短划线分隔符。

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

```
  'EXPR$0'
  No rows selected
```

下面的四个模板之所以起作用，是因为使用相同的分隔符将年说明符与下一个说明符分隔，就像在输入字符串的日期规范中使用的那样（第一个示例中是短划线，第二个示例中是空格，第三个示例中是斜杠，第四个示例中是短划线）。

```
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)，则除非还指定了日，否则将无法指定时、分或秒。

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TIMESTAMP	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个是年和第二个是分，因此输入字符串中的第二个元素（“02”）用作分。月和日使用默认值（“01”），

模板	输入字符串	输出TIMESTAMP	注意
			时和秒使用默认值（“00”）。
'MMM dd, yyyy'	'March 7, 2010'	'2010-03-07 00:00:00'	上面模板中的 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，输入字符串也必须缺少逗号，否则就没有输出； 如果输入字符串缺少逗号，模板也必须缺少逗号。
'MMM dd,'	'March 7, 2010'	'1970-03-07 00:00:00'	请注意，上面的模板没有使用年说明符，这会导致输出 TIMESTAMP 使用这个时代最早的一年，即 1970。
'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“March 7, 10”，则输出 TIMESTAMP 将是“0010-03-07 00:00:00”。

模板	输入字符串	输出TIMESTAMP	注意
'M-d'	'2-8'	'1970-02-08 00:00:00'	如上所示，模板中没有 yyyy 说明符，因此使用这个时代最早的一年 (1970)。 输入字符串“2–8–2011”将生成相同的结果；使用“2011–2–8”不会生成任何结果，因为 2011 不是有效的月。
'MM-dd-yyyy'	'06-23-2011 10:11:12'	'2011-06-23 00:00:00'	如果模板和输入都在相同位置使用短划线作为分隔符（如上所示），则可以使用短划线。由于模板省略了时、分和秒，因此在输出 TIMESTAMP 中都使用零。
'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这些说明符，但前提是该顺序与您提供的输入字符串的含义匹配。下面下一个示例的模板和输入字符串与此示例具有相同的含义（和相同的输出 TIMESTAMP），但在日之前指定了月，在时之前指定了秒。

模板	输入字符串	输出TIMESTAMP	注意
'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，月和日说明符的顺序与上面的示例相反，秒的说明符在时之前而不是分之后；但是由于输入字符串还将月放在日之前，将秒放在时之前，因此含义（和输出 TIMESTAMP）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'	上面使用的模板颠倒了（与上面的前一个示例相比）年和月说明符，而输入字符串保持不变。在本例中，输出 TIMESTAMP 使用输入字符串的第一个元素作为年，第二个元素作为日，第三个元素作为月。

模板	输入字符串	输出TIMESTAMP	注意
'dd-MM-yy hh:mm'	'23-06-11 10:11:12'	'2011-06-23 10:11:00'	如上所示，模板中省略了秒，输出 TIMESTAMP 使用 00 秒。任意数量的 y 说明符都会生成相同的结果；但是如果输入字符串无意中对年使用的是 1 而不是 11，例如在“23-06-1 10:11:12”中，则输出 TIMESTAMP 将变为“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'	如果模板和输入都在相同位置使用斜杠作为分隔符（如上所示），则可以使用斜杠。使用说明符 hh，输入时间“12:11:10”和“00:11:10”与上午的时间含义相同。

模板	输入字符串	输出TIMESTAMP	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'
			使用此模板时，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'不是模板字符串那样 required/expected 是一个月。'MM/dd/yy HH:mm:ss'
			
			但是，更改模板会生成有用的输出：
			
			`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'`
			
			已选定 1 行
			
			对于上述模板 ('y/MM/dd')，'12/19/11 00:11:12' 会失败，因为 19 不是有效的月；提供“12/11/19 00:11:12' 有效。

模板	输入字符串	输出TIMESTAMP	注意
			'2011-12-19 12:11:12' 作为输入会失败，因为短划线与模板中的斜杠不匹配，'2011/12/19 12:11:12' 有效。 请注意，对于中午 12 点之后的时间，也就是说，对于下午和晚上的时间，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，时从 00 到 23。 使用说明符 HH，输入时间“12:11:10”和“00:11:10”具有不同的含义，第一个是下午的时间，第二个是上午的时间。 使用说明符 hh，从 12:00 到 11:59:59 的时间是上午的时间： • 使用说明符 hh:mm:ss，对于输入字符串“12:09:08”和 '00:09:08' ，输出 TIMESTAMP

模板	输入字符串	输出TIMESTAMP	注意
			都将包括上午的'00:09:08' ； • 给定说明符 HH:mm:ss，早上输入字符串'00:09:08' 的输出时间戳将包括'00:09:08' 而对于下午的输入字符串'12:09:08' ，输出 TIMESTAMP 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

使用格式字符串对时间进行格式化。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，您需要一个具有样本股票代码输入流的 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。

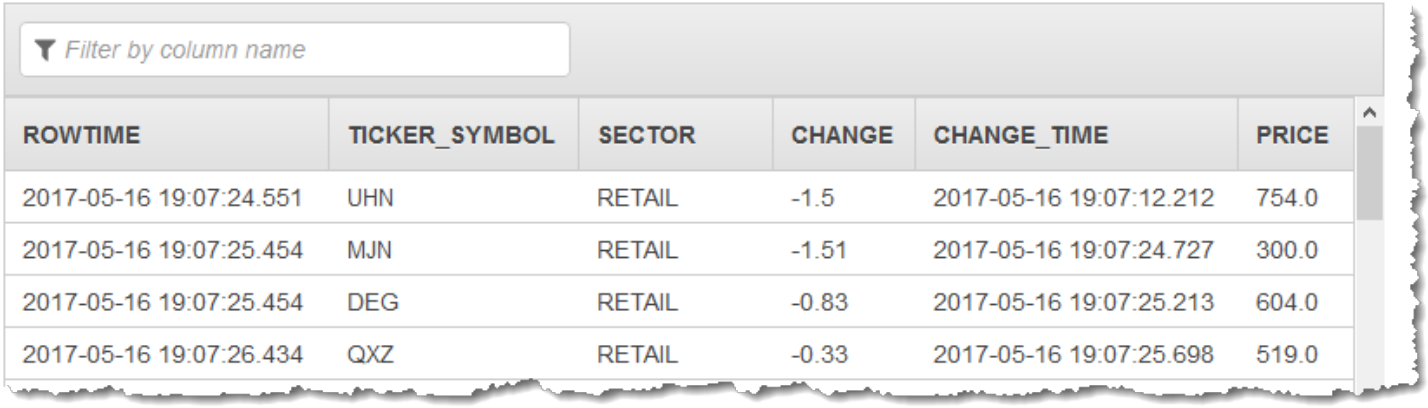
```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
change_time     BIGINT,    --The UNIX timestamp value
price           REAL)
```

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。



Filter by column name

ROWTIME	TICKER_SYMBOL	SECTOR	CHANGE	CHANGE_TIME	PRICE
2017-05-16 19:07:24.551	UHN	RETAIL	-1.5	2017-05-16 19:07:12.212	754.0
2017-05-16 19:07:25.454	MJN	RETAIL	-1.51	2017-05-16 19:07:24.727	300.0
2017-05-16 19:07:25.454	DEG	RETAIL	-0.83	2017-05-16 19:07:25.213	604.0
2017-05-16 19:07:26.434	QXZ	RETAIL	-0.33	2017-05-16 19:07:25.698	519.0

注意

TO_TIMESTAMP 不是标准的一部分。 SQL:2008 它是 Amazon Kinesis Data Analytics 流式 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

时间StampExpr

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，您需要一个具有样本股票代码输入流的 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。

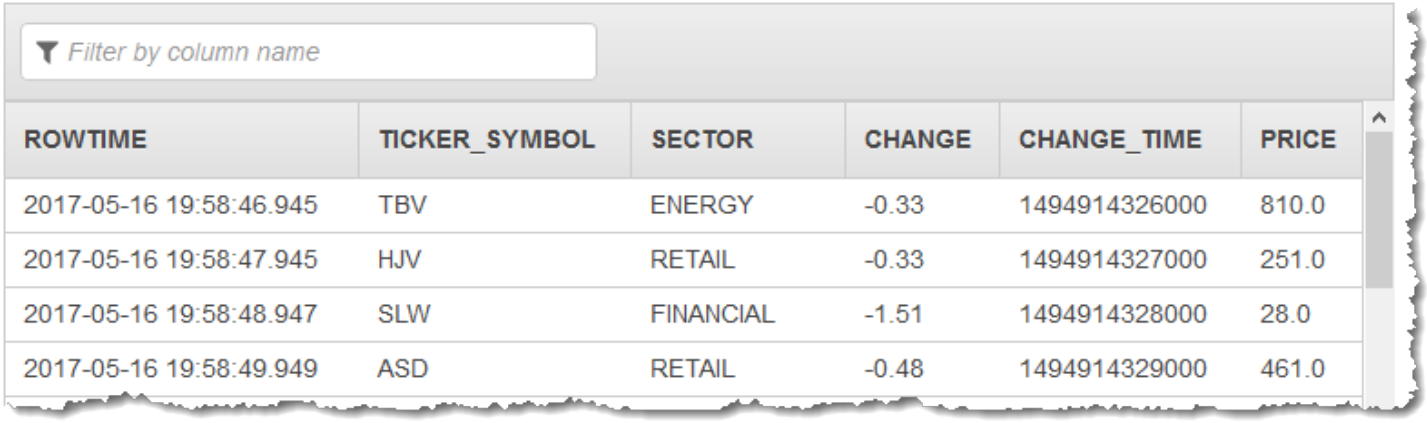
```
(ticker_symbol  VARCHAR(4),
sector          VARCHAR(16),
change          REAL,
change_time     TIMESTAMP,    --The timestamp value to convert
price           REAL)
```

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。


Filter by column name

ROWTIME	TICKER_SYMBOL	SECTOR	CHANGE	CHANGE_TIME	PRICE
2017-05-16 19:58:46.945	TBV	ENERGY	-0.33	1494914326000	810.0
2017-05-16 19:58:47.945	HJV	RETAIL	-0.33	1494914327000	251.0
2017-05-16 19:58:48.947	SLW	FINANCIAL	-1.51	1494914328000	28.0
2017-05-16 19:58:49.949	ASD	RETAIL	-0.48	1494914329000	461.0

注意

UNIX_TIMESTAMP 不是标准的一部分。 SQL:2008 它是 Amazon Kinesis Data Analytics 流式 SQL 扩展。

日期、时间戳和间隔运算符

算术运算符“+”、“-”、“*”和“/”都是二进制运算符。

运算符	说明	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 日期时间 - 间隔 = 日期时间 (<datetime> - <datetime>) 日期、时间戳和间隔运算符 <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	操作	结果
1	INTERVAL '1' DAY + INTERVAL '3' DAY	INTERVAL '4' DAY
2	INTERVAL '1' DAY + INTERVAL '3 4' DAY TO HOUR	INTERVAL '+4 04' DAY TO HOUR
3	INTERVAL '1' DAY - INTERVAL '3 4' DAY TO HOUR	INTERVAL '-2 04' DAY TO HOUR
4	INTERVAL '1' YEAR + INTERVAL '3-4' YEAR TO MONTH	INTERVAL '+4-04' YEAR TO MONTH
5	2 * INTERVAL '3 4' DAY TO HOUR	INTERVAL '6 8' DAY TO HOUR
6	INTERVAL '3 4' DAY TO HOUR / 2	INTERVAL ' 1 14' DAY TO HOUR

在示例 3 中，'3 4 DAY 表示 3 天零 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，即负 2 天零 4 小时。

示例 4 使用的是 TO MONTH 而不是 TO HOUR，因此指定为“3-4”的 INTERVAL 表示 3 年零 4 个月或 40 个月。

在示例 6 中，“/2”适用于 INTERVAL“3 4”，即 76 小时，其中一半为 38 小时，或 1 天零 14 小时。

间隔运算的更多示例

流式 SQL 还支持减去两个日期时间，从而生成一个间隔。您可以为结果指定想要的间隔类型，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运算。

Example 1 – 时间差异（以精确到秒的分钟或秒表示）

```
 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 – 时间差异（仅以分钟表示）

```
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 — Time-to-Timestamp 差异（以天数到最接近的秒数计）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（以精确到秒的天表示）

```
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（以精确到秒的天表示）

```
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（以天表示）

```
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 – 时间差异（以天表示）

```
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（简单日期差异）

如果您未将“天”指定为预期单位，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”？

在上述示例中使用“values cast (<expression> AS varchar(N))”语法的原因是，虽然上面使用的 SQLline 客户端（运行了 Amazon Kinesis Data Analytics）返回了间隔，但 JDBC 不支持返回该结果，因此是为了显示该结果。因此，使用了 “值” 语法。 see/show

如果您（通过运行 !kill 命令）关闭 Amazon Kinesis Data Analytics 或在运行 SQLline 之前未启动该应用程序，则可从 Amazon Kinesis Data Analytics 主目录的 bin 子目录运行 sqllineEngine（而不是 sqllineClient），这可在没有 Amazon Kinesis Data Analytics 应用程序或 JDBC 的情况下显示结果：

指定间隔的规则

间 Day-Time 隔文字是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），后者表示值的单位。

限定符采用以下格式：

```
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，前导项必须比可选的尾随项更重要，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
```

理解这些项最简单的方法可能是将 X TO Y 转换为“Xs to the nearest Y”。因此，DAY TO HOUR 是“days to the nearest hour”。

当 DAY、HOUR 或 MINUTE 是前导项时，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。最大精度是 10，默认值是 2。您不能在尾随项中为 HOUR 或 MINUTE 指定精度，其精度始终为 2。例如，HOUR(3) TO MINUTE 是合法的，而 HOUR TO MINUTE(3) 则不合法。

SECOND 也可以有精度，但其指定方式有所不同，具体取决于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位数。例如，SECOND(3,3) 允许您指定最多 999.999 秒。默认值是 (2,3)，这实际上是与 SQL:2008 规范的偏差（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，即下面显示的秒小数点之后的部分。例如，SECOND(3) 表示毫秒。默认值为小数点后 3 位数，但如上所示，这与标准的 6 位数有偏差。

至于值，采用以下一般格式：

```
[+-]'[+-]DD HH:MM:SS.SSS'
```

其中 DD 是表示天、HH 小时、MM 分钟的数字， SS.SSS 是秒（如果明确指定了精度，则相应地调整位数）。

并非所有值都必须包含所有字段，您可以去除前后部分，但不能去除中间部分。所以你可以把它设为“DD HH” 或 “MM:SS.SSS”，但不能改成 “DD MM”。

但是，无论如何编写，该值都必须与限定符匹配，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，则暗示精度为 2。因此：

- INTERVAL '120' MINUTE 是非法间隔。所需间隔的合法格式为 INTERVAL '120' MINUTE(2)

and

- INTERVAL '120' SECOND 不合法。所需间隔的合法格式为 INTERVAL '120' SECOND(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 “小时”、“分钟” 或 “秒” 不是主要字段，则它们必须处于以下范围内（取自 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
```

Year-month 间隔类似，唯一的不同是限定符如下所示：

```
YEAR
MONTH
YEAR TO MONTH
```

与 DAY 和 HOUR 一样，可以指定精度，同样，最大值为 10，默认值为 2。

年月的值格式为：'YY-MM'。如果 MONTH 是尾随字段，则必须位于 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 和从 a 到 z 的未加引号的字母表示数据或时间值的组件。如果字母或文本字符串用一对单引号括起来，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。

以下模式字母是为指定的日期或时间组件定义的。从“A”到“Z”和从“a”到“z”的所有其他字符均已保留。有关模式字母的字母顺序，请参阅按字母顺序排列的日期和时间模式字母。

日期或时间组件	模式字母	以文本或数字形式呈现	示例
时代标志	G	文本	AD
Year	y	Year	1996；96
年中某月	M	Month	七月；7 月；07
年中某周	w	数字	27
月中某周	W	数字	2
年中某日	D	数字	189
月中某日	d	数字	10
月中某周某日	F	数字	2
周中某日	E	文本	EE=Tu；EEE=Tue；EEEE=Tuesday
Am/pm 标记	a	文本	PM
一天中的时 (0-23)	H	数字	0
一天中的时 (1-24)	k	数字	24

日期或时间组件	模式字母	以文本或数字形式呈现	示例
一小时后 am/pm (0-11)	K	数字	0
一小时后 am/pm (1-12)	h	数字	12
时中的分	m	数字	30
分中的秒	s	数字	55
毫秒	S	数字	978
时区	z	General	太平洋标准时间；太平洋标准时间；GMT-08:00
时区	Z	RFC	-0800

模式字母通常是重复的，因为它们的数字决定了确切的呈现方式：

文本

格式化时，如果模式字母数为 4 或更多，则使用完整形式；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，与模式字母数无关。

数字

格式化时，模式字母数是最小位数，较短的数字将用零填补到此数量。解析时，除非需要分隔两个相邻字段，否则模式字母数将被忽略。

Year

如果时区有名称，则将被解释为文本。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
```

时必须介于 0 和 23 之间，分必须介于 00 和 59 之间。格式与语言环境无关，数字必须取自 Unicode 标准的基本拉丁语块。

解析时，还接受 RFC 822 时区。

RFC 822 时区

格式化时，使用 RFC 822 4 位数的时区格式：

```
RFC822TimeZone:
Sign TwoDigitHours Minutes
TwoDigitHours:
Digit Digit
```

TwoDigitHours 必须介于 00 到 23 之间。其他定义与一般时区相同。

解析时，还接受一般时区。

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取决于区域设置的模式字母。 SimpleDateFormat 除了图案字母外，不处理文本的本地化问题；这取决于全班的客户。

示例

以下示例显示了如何在 U.S. locale 中解释日期和时间模式。给定的日期和时间是当地时间 2001-07-04 12:08:56。 U.S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”	2001.07.04 AD at 12:08:56 PDT
"EEE, MMM d, "yy"	Wed, Jul 4, '01

日期和时间模式	结果
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
“K:mm a、z”	0:08 PM, PDT
“yyyyy。MMMMM.dd GGG 哈哈:嗯 aaa”	02001。July.04 广告下午 12:08
“哎哟，嗯嗯 YYYY HH:mm:ss Z”	Wed, 4 Jul 2001 12:08:56 -0700
“是的MMddHHmmssZ”	010704120856-0700
“yyyy-MM-dd 'T'” HH:mm:ss.SSSZ	2001-07-04 T12:08:56.235-0700

按字母顺序排列的日期和时间模式字母

上面按日期或时间组件顺序显示的相同模式字母按字母顺序显示在下面，便于参考。

模式字母	日期或时间组件	以文本或数字形式呈现	示例
a	Am/pm 标记	文本	PM
D	年中某日	数字	189
d	月中某日	数字	10
E	周中某日	文本	EE=Tu；EEE=Tue；EEEE=Tuesday
F	月中某周某日	数字	2
G	时代标志	文本	AD
H	一天中的时 (0-23)	数字	0
h	一小时后 am/pm (1-12)	数字	12

模式字母	日期或时间组件	以文本或数字形式呈现	示例
k	一天中的时 (1-24)	数字	24
K	一小时后 am/pm (0-11)	数字	0
M	年中某月	Month	七月；7 月；07
m	时中的分	数字	30
s	分中的秒	数字	55
S	毫秒	数字	978
w	年中某周	数字	27
W	月中某周	数字	2
y	Year	Year	1996；96
z	时区	General	太平洋标准时间；太平洋标准时间；GMT-08:00
Z	时区	RFC	-0800

CURRENT_DATE

返回当前 Amazon Kinesis 数据分析系统执行日期和执行查询 YYYY-MM-DD 时的日期。

有关更多信息，请参阅
CURRENT_TIME、CURRENT_TIMESTAMP、LOCALTIMESTAMP、LOCALTIME和CURRENT_ROW_TIME

示例

```
+---------------+
| CURRENT_DATE  |
+---------------+
```

```
| 2008-08-27    |
+---------------+
```

CURRENT_ROW_TIMESTAMP

CURRENT_ROW_TIMESTAMP 是该规范的亚马逊 Kinesis 数据分析扩展。 SQL:2008 此函数会返回 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 而不是本地时区返回。

CURRENT_ROW_TIMESTAMP 类似于 LOCALTIMESTAMP，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查询首次运行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 规范中未定义 CURRENT_ROW_TIMESTAMP；它是亚马逊 Kinesis 数据分析扩展。

有关更多信息，请参阅CURRENT_TIME、CURRENT_DATE、CURRENT_TIMESTAMP、LOCALTIMESTAMP、LOCALTIME和

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、LOCALTIMESTAMP、LOCALTIME、CURRENT_ROW_TIMESTAMP和CURREN

示例

```
+---------------+
| CURRENT_TIME  |
+---------------+
| 20:52:05      |
```

CURRENT_TIMESTAMP

将（Amazon Kinesis Data Analytics 运行环境定义的）当前数据库系统时间戳返回为日期时间值。

有关更多信息，请参阅
CURRENT_TIME、CURRENT_DATE、LOCALTIME、LOCALTIMESTAMP和CURRENT_ROW_TIMESTAM

示例

```
+--------------------+
| CURRENT_TIMESTAMP  |
+--------------------+
| 20:52:05           |
+--------------------+
```

EXTRACT

```
EXTRACT(YEAR|MONTH|DAY|HOUR|MINUTE|SECOND FROM <datetime expression>|<interval
 expression>)
```

EXTRACT 函数从 DATE、TIME、TIMESTAMP 或 INTERVAL 表达式中提取一个字段。对于 SECOND 以外的所有字段，都返回 BIGINT。对于 SECOND，返回 DECIMAL(5,3)，包括毫秒。

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

函数	结果
`EXTRACT(SECOND FROM INTERVAL '2 3:4:5.678' DAY TO SECOND)`	5.678
`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

用在函数中

EXTRACT 可用于条件数据，例如在下面的函数中，当为 CURRENT_ROW_TIMESTAMP 输入 p_time 时，返回 30 分钟的下限。

```
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 ) ;
```

您可以使用下面的代码来实现此函数：

```
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创建了一个名为 “” 的流MyStream。

LOCALTIME

返回 Amazon Kinesis Data Analytics 运行环境所定义执行查询的当前时间。LOCALTIME 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、CURRENT_DATE、CURRENT_TIMESTAMP、LOCALTIMESTAMP和CURRENT_ROW

示例

```
 VALUES localtime;
+------------+
| LOCALTIME  |
+------------+
| 01:11:15   |
+------------+
1 row selected (1.558 seconds)
```

限制

亚马逊 Kinesis 数据分析不支持<time precision>中 SQL:2008指定的可选参数。这与 SQL:2008 标准背道而驰。

LOCALTIMESTAMP

返回 Amazon Kinesis Data Analytics 应用程序运行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、CURRENT_DATE、CURRENT_TIMESTAMP、LOCALTIME和CURRENT_ROW_TIMES

示例

```
values localtimestamp;
+-------------------------+
|      LOCALTIMESTAMP     |
+-------------------------+
| 2008-08-27 01:13:42.206  |
+-------------------------+
1 row selected (1.133 seconds)
```

限制

亚马逊 Kinesis 数据分析不支持<timestamp precision>中 SQL:2008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

本节中的主题介绍了 Amazon Kinesis Data Analytics 流式 SQL 的 null 函数。

主题

- COALESCE
- NULLIF

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 null 参数。如果所有表达式均为 null，则 COALESCE 将返回 null。

示例

Expression	结果
COALESCE('chair')	chair
COALESCE('chair', null, 'sofa')	chair
COALESCE(null, null, 'sofa')	sofa
COALESCE(null, 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 as varchar(3)))	amy
NULLIF(cast(null as varchar(3)),'fred')	<null>

数字函数

本节中的主题介绍了 Amazon Kinesis Data Analytics 流式 SQL 的数字函数。

主题

- ABS
- CEIL/CEILING
- EXP
- FLOOR
- LN
- LOG10
- MOD
- POWER
- STEP

ABS

返回输入参数的绝对值。如果输入参数为 null，则返回 null。

```
ABS ( <numeric-expression>  <interval-expression>
      )
```

示例

函数	结果
ABS(2.0)	2.0
ABS(-1.0)	1.0
ABS(0)	0
ABS(-3 * 3)	9
ABS(INTERVAL '-3 4:20' DAY TO MINUTE)	INTERVAL '3 4:20' DAY TO MINUTE

如果您在 SQLline 中使用 `cast as VARCHAR` 来显示输出，则返回的值将为 `+3 04:20`。

```
values(cast(ABS(INTERVAL '-3 4:20' DAY TO MINUTE) AS VARCHAR(8)));
 +-----------+
   EXPR$0
 +-----------+
  +3 04:20
 +-----------+
 1 row selected
```

CEIL/CEILING

```
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，CEILING 将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 null，则返回 null。

示例

函数	结果
CEIL(2.0)	2
CEIL(-1.0)	–1
CEIL(5.2)	6
CEILING(-3.3)	-3
CEILING(-3 * 3.1)	-9
CEILING(TIMESTAMP '2004-09-30 13:48:23' TO HOUR)	TIMESTAMP '2004-09-30 14:00:00'

函数	结果
CEILING(TIMESTAMP '2004-09-30 13:48:23' TO MINUTE)	TIMESTAMP '2004-09-30 13:49:00'
CEILING(TIMESTAMP '2004-09-30 13:48:23' TO DAY)	TIMESTAMP '2004-10-01 00:00:00.0'
CEILING(TIMESTAMP '2004-09-30 13:48:23' TO YEAR)	TIMESTAMP '2005-01-01 00:00:00.0'

注意

- CEIL 和 CEILING 是该 SQL:2008 标准提供的此函数的同义词。
- CEIL(<datetime value expression> TO <time unit>) 是 Amazon Kinesis Data Analytics 扩展。
- 有关更多信息，请参阅 FLOOR。

EXP

```
EXP ( <number-expression> )
```

返回 e 的值（约为 2.7182818284590455），乘以输入参数的幂。如果输入参数为 null，则返回 null。

示例

函数	结果
EXP(1)	2.718281828284590455
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 将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
FLOOR(2.0)	2
FLOOR(-1.0)	–1
FLOOR(5.2)	5
FLOOR(-3.3)	-4
FLOOR(-3 * 3.1)	-10
FLOOR(TIMESTAMP '2004-09-30 13:48:23' TO HOUR)	TIMESTAMP '2004-09-30 13:00:00'
FLOOR(TIMESTAMP '2004-09-30 13:48:23' TO MINUTE)	TIMESTAMP '2004-09-30 13:48:00'
FLOOR(TIMESTAMP '2004-09-30 13:48:23' TO DAY)	TIMESTAMP '2004-09-30 00:00:00.0'
FLOOR(TIMESTAMP '2004-09-30 13:48:23' TO YEAR)	TIMESTAMP '2004-01-01 00:00:00.0'

注意

> **Note**
>
> FLOOR (<datetime expression> TO <timeunit>) 是 Amazon Kinesis Data Analytics 扩展。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。

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，则会引发异常。如果输入参数为 null，则返回 null。

有关更多信息，请参阅LOG10和EXP。

示例

函数	结果
LN(1)	0.0
LN(10)	2.302585092994046
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，则会引发异常。如果输入参数为 null，则返回 null。

示例

函数	结果
LOG10(1)	0.0
LOG10(100)	2.0
log10(cast('23' as decimal))	1.3617278360175928

> **Note**
>
> LOG10 不是 SQL:2008 标准函数；它是该标准的亚马逊 Kinesis 数据分析扩展。

MOD

```
MOD ( <dividend>, <divisor> )
 <dividend> := <integer-expression>
 <divisor>  := <integer-expression>
```

返回第一个参数（被除数）除以第二个数字参数（除数）的余数。如果除数为零，则会引发除以零的错误。

示例

函数	结果
MOD(4,2)	0
MOD(5,3)	2
MOD(-4,3)	–1
MOD(5,12)	5

限制

Amazon Kinesis Data Analytics MOD 函数仅支持小数位数为 0 的参数（整数）。这与支持任何数字参数的 SQL:2008 标准背道而驰。当然，其他数字参数可以转换为整数。

POWER

```
POWER ( <base>, <exponent> )
<base> := <number-expression>
<exponent> := <number-expression>
```

返回第一个参数（基数）的值乘以第二个参数（指数）的幂。如果基数或指数为 null，则返回 null；如果基数为零且指数为负，或者基数为负且指数不是整数，则会引发异常。

示例

函数	结果
POWER(3,2)	9
POWER(-2,3)	-8
POWER(4,-2)	1/16 ... 或者。。0.0625
POWER(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 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
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。

Filter by column name

ROWTIME	TICKER_SYMBOL	SUM_PRICE
2017-07-26 20:23:00.0	MMB	62.11000061035156
2017-07-26 20:24:00.0	HJV	1968.909912109375
2017-07-26 20:24:00.0	MMB	69.8800048828125
2017-07-26 20:24:00.0	CRM	250.12998962402344

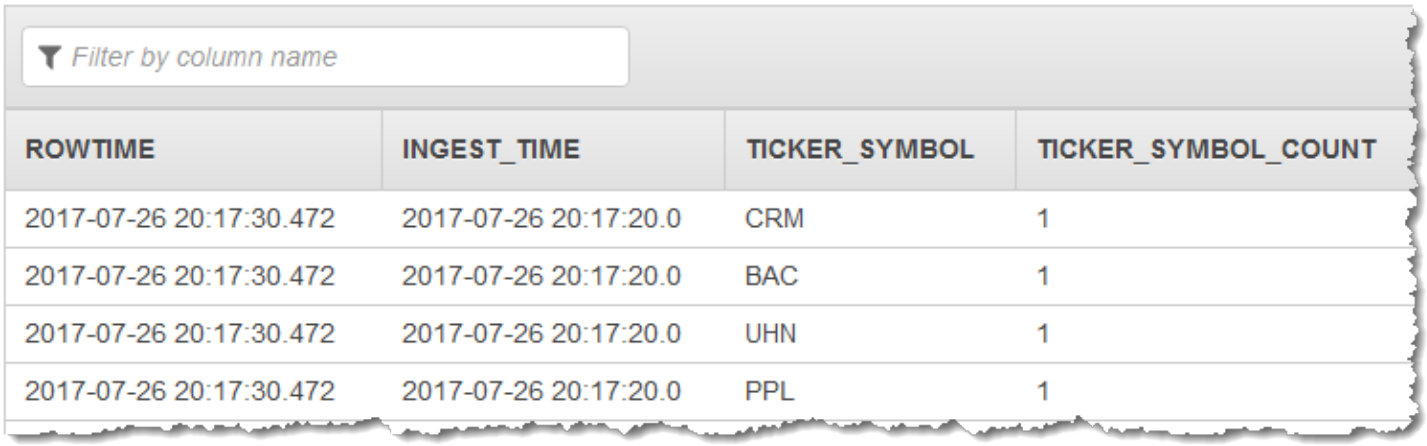
OVER 子句中的 STEP（滑动窗口）

```
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_COUNT
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。

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 具有以下日志解析函数：

- FAST_REGEX_LOG_PARSER 的工作原理类似于正则表达式解析器，但采取了一些“捷径”以确保更快生成结果。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 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 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处理 UNIX/Linux 系统日志中常见的条目。
- 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。
- W3C_LOG_PARSE处理 W3C-predefined-format 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。任何表达式开头的任何固定长度部分都移动到前一个表达式的结尾。如果任何表达式的长度完全固定，则会与前一个表达式合并。然后，使用惰性语义计算一系列表达式，不回溯。（在正则表达式解析术语中，“惰性”意味着在每个步骤中解析的内容都不超过所需。“贪婪”意味着在每个步骤中尽可能多地解析。）

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 使用惰性搜索，即在第一次匹配时停止搜索。相比之下，REGEX_LOG_PARSE 是贪婪的，除非使用了占有限定符。

FAST_REGEX_LOG_PARSE 会扫描所提供的输入字符串来查找 Fast Regex 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。扫描组定义扫描成功时生成的字段或列。
- 如果在应用 Fast Regex 模式时解释了输入字符串中的所有字符，则 FRLP 会按从左到右的顺序从该 Fast Regex 模式中每个带圆括号的表达式创建一个输出字段（列）。第一个（最左边）带圆括号的表达式创建第一个输出字段，下一个（第二个）带圆括号的表达式创建第二个输出字段，直到最后一个带圆括号的表达式创建最后一个输出字段。
- 如果输入字符串包含任何未通过应用 Fast Regex 模式解释（匹配）的字符，则 FRLP 根本不会返回任何字段。

Fast Regex 的字符类符号

Fast Regex 使用与常规正则表达式解析器不同的字符类符号集：

符号或构造	含义
-	字符范围，包括端点
[charclasses]	字符类

符号或构造	含义
[^ charclasses]	否定字符类
|	Union
&	交集
?	出现零次或一次
*	出现零次或多次
+	出现一次或多次
{n}	出现 n 次
{n,}	出现 n 次或更多次
{n,m}	出现 n 到 m 次，包括这两者
.	任何单个字符
#	空白语言
@	任何字符串
"<Unicode string without double-quotes>"	字符串)
( )	空字符串)
(unionexp)	优先级覆盖
< <identifier> >	命名模式
<n-m>	数字间隔
charexp:=<Unicode character>	单个非保留字符
\ <Unicode character>	单个字符)

我们支持使用以下 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') 的扫描从快速正则表达式模式中定义的第一组开始：(.*)，这意味着 “查找任意字符 0 次或更多次”。

'(.*)_(._.*)_.*'

2. 此组规范定义了要解析的第一列，要求 Fast Regex 日志解析器接受从输入字符串的第一个字符开始的输入字符串字符，直到在 Fast Regex 模式中找到下一个组或下一个不在组内（不在圆括号中）的文字字符或字符串。在此示例中，第一个组后面的下一文本字符为下划线：

   '(.*)_(._.*)_.*'

3. 解析器会扫描输入字符串中的每个字符，直到在 Fast Regex 模式中找到下一个规范：下划线：

   '(.*)_(._.*)_.*'

4. Group-2 因此以 “a_l” 开头。接下来，解析器需要使用模式中的其余规范来确定此组的结尾：

   '(.*)_(._.*)_.*'

> **Note**
>
> Character-strings 或在模式中指定但不在组内的文字必须在输入字符串中找到，但不会包含在任何输出字段中。
> 如果 Fast Regex 模式省略了最后一个星号，则不会得到任何结果。

更多 FRLP 示例

下一个示例使用的是“+”，表示重复最后一个表达式 1 次或多次（“*”表示 0 次或多次）。

示例 A

在本例中，最长的前缀是第一个下划线。第一 field/column 组将在 “玛丽” 上匹配，第二组不匹配。

```
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
```

前一个示例不会返回任何字段，因为“+”要求一行中至少有一个下划线，而输入字符串没有。

示例 B

在以下示例中，由于采用了惰性语义，“+”是多余的：

```
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。下划线不会出现在“Mary”后面，也不会出现在“had”前面，因为“_+”规范没有用圆括号括起来。

（正如简介中提到的，在正则表达式解析术语中，“惰性”意味着在每个步骤中解析的内容都不超过所需。“贪婪”意味着在每个步骤中尽可能多地解析。）

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 将回溯。

正上方的搜索 B 变成了三次搜索：

```
(.*)_
_*(._
.*)
```

请注意，第二个字段组在第二次和第三次搜索之间被拆分，“_+”也被视为与“__*”相同（也就是说，认为“underscore repeat-underscore-1-or-more-times”与“underscore underscore repeat-underscore-0-or-more-times”相同。）

示例 A 演示了 REGEX_LOG_PARSE 和 FAST_REGEX_LOG_PARSE 之间的主要区别，因为 A 中的搜索将使用 REGEX_LOG_PARSE，因为该函数将使用回溯。

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>'
'.*$'
```

使用惰性语义时，将依次与每个匹配。

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，允许用户指定确切的时间组件布局。解析器使用 Java 类 java.lang。SimpleDateFormat解析日期、时间和时间戳类型的字符串。日期和时间模式主题提供了时间戳格式字符串的完整描述和示例。下面是带有格式字符串的列定义的一个示例：

```
"name" TYPE TIMESTAMP 'dd/MMM/yyyy:HH:mm:ss'
```

相关主题

REGEX_LOG_PARSE

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，并按从左到右的顺序处理这些组。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，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，您需要一个具有样本股票代码输入流的 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)
```

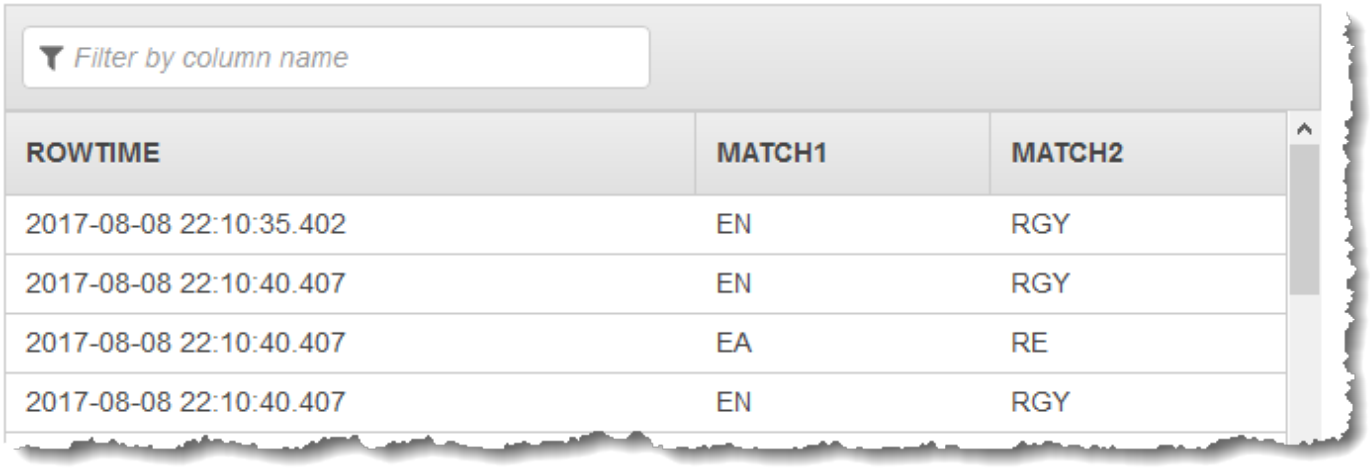
示例 1：从两个捕获组返回结果

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：

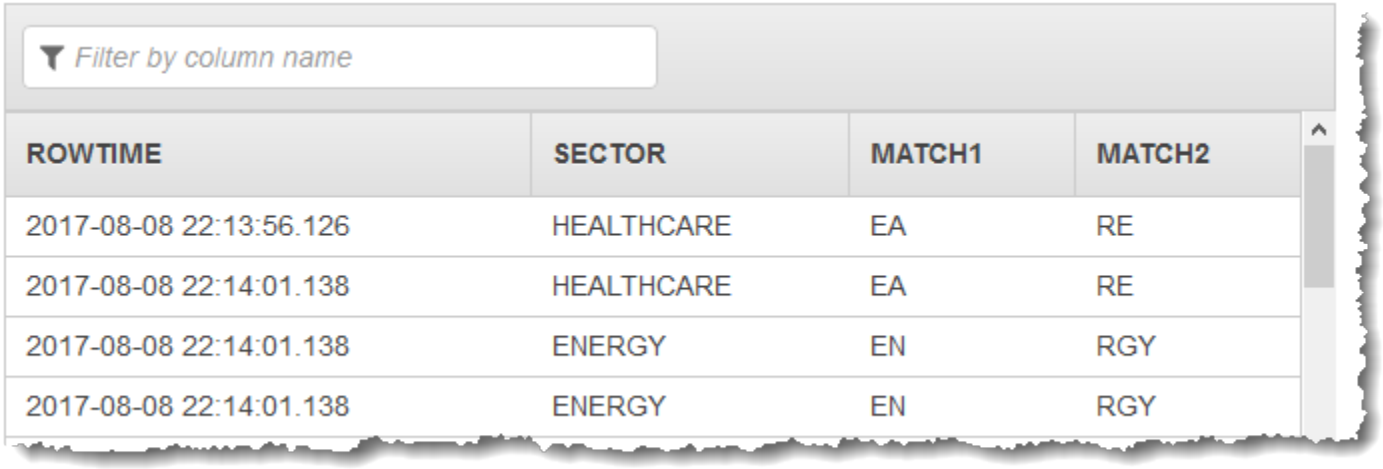


示例 2：从两个捕获组返回一个流字段和结果

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。

正则表达式快速参考

有关正则表达式的完整详细信息，请参阅 java.util.regex.pattern

[xyz] 查找 x、y 或 z 的单个字符

[^abc] 查找除了 x、y 或 z 之外的任何单个字符

[r-z] 查找 r-z 之间的任何单个字符

[r-zR-Z] 查找 r-z 之间的任意单个字符或 R-Z

^ 行首

$ 行尾

\A 字符串开始

\z 字符串结束

. 任何单个字符

\s 查找任何空格字符

\S 查找任何非空格字符

\d 查找任何数字

\w 查找任何单词字符（字母、数字、下划线）

\W 查找任何非单词字符

\b 查找任何单词边界

(...) 捕获括起来的所有内容

(x|y) 查找 x 或 y（也适用于诸如 \d 或 \s 之类的符号）

x? 查找零个或一个 x（也适用于诸如 \d 或 \s 之类的符号）

x* 查找零个或多个 x（也适用于诸如 \d 或 \s 之类的符号）

x+ 查找一个或多个 x（也适用于诸如 \d 或 \s 之类的符号）

x{3} 精确查找 3 个 x（也适用于诸如 \d 或 \s 之类的符号）

\D 查找任何非数字	x{3,} 查找 3 个或更多 x（也适用于诸如 \d 或 \s 之类的符号）
	x{3,6} 查找 3 到 6 个 x（也适用于诸如 \d 或 \s 之类的符号）

SYS_LOG_PARSE

解析标准系统日志格式：

```
Mon DD HH:MM:SS server message
```

SYS_LOG_PARSE 处理系统日志中常见的条目。 UNIX/Linux 系统日志条目以时间戳开头，后面是自由格式的文本字段。SYS_LOG_PARSE 输出由两列组成。第一列名为“COLUMN1”，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，请参阅日期和时间模式。

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，它可以处理逗号分隔的值或制表符分隔的值。它

可以与 FIXED_COLUMN_LOG_PARSE 结合使用来处理诸如 maillog 之类的内容，其中一些字段是固定长度的，而另一些则是可变长度的。

> **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 支持 表达式和文字，制表符也可以是使用 unicode 转义符指定的分隔符，例如 u&'\0009'，它是仅包含一个制表符的字符串。

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，则将其用作带引号的字符串的开头和结尾。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。日期和时间模式提供了时间戳格式字符串的完整描述，并附有示例。下面是带有格式字符串的列定义的一个示例：

```
    "name" TYPE TIMESTAMP 'dd/MMM/yyyy:HH:mm:ss'
```

默认情况下，输出列名为 COLUMN1、COLUMN2、COLUMN3 等，每个的 SQL 数据类型都是 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

指定以下 W3C-predefined-format 名称将使用所示的格式说明符进行汇总，如以下语句所示：

```
select stream W3C_LOG_PARSE(message, 'COMMON') r  from w3ccommon t;
```

格式名称	W3C 名称	格式说明符
COMMON	通用日志格式 (CLF)	%h %l %u %t "%r" %>s %b
COMMON WITH VHOST	虚拟主机的通用日志格式	%v %h %l %u %t "%r" %>s %b
NCSA EXTENDED	NCSA extended/combined 日志格式	%h %l %u %t “%r” %>s %b “% [Referer] i” “% [] i” User-agent
REFERER	Referer 日志格式	%[Referer]i ---> %U
AGENT	代理（浏览器）日志格式	% [User-agent] 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，并输出每个说明符对应一列的记录。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，零字节用短划线表示，强制将列类型设置为文本。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-address
%...A	本地 IP-address
%...B	响应大小（以字节为单位），不包括 HTTP 标头。
%...b	CLF 格式的响应大小（以字节为单位），不包括 HTTP 标头，这意味着当未发送字节时，使用“-”而不是 0。
%...[Customerdata]C	发送到服务器的请求中 cookie Customerdata 的内容。
%...D	处理请求所花费的时间，以微秒为单位。
%...[CUSTOMERDATA]e	环境变量 CUSTOMERDATA 的内容
%...f	文件名
%...h	远程主机
%...H	请求协议
%...[Customerdata]i	发送到服务器的请求中 Customerdata：标头行的内容。
%...l	远程登录名（如果提供，则来自 identd）
%...m	请求方法
%...[Customerdata]n	来自其他模块的备注 Customerdata 的内容。
%...[Customerdata]o	回复中 Customerdata：标头行的内容。
%...p	处理请求的服务器的规范端口

格式说明符	说明
%...P	处理请求的子项的进程 ID。
%...[format]P	处理请求的子项的进程 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
%...s	状态。对于内部重定向的请求，这是*原始*请求的状态，而对于最后一个，是 %...>s。
%...t	时间，采用通用日志格式的时间格式（标准英文格式）
%...[format]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，则可能是虚假的）
%...U	请求的 URL 路径，不包括任何查询字符串。
%...v	服务请求的服务器 ServerName 的权威规范。
%...V	根据 UseCanonicalName 设置的服务器名称。

格式说明符	说明
%...X	响应完成时的连接状态 X = 连接在响应完成前已中止。 + = 连接在响应发送后可能会保持活动状态。 - = 连接在响应发送后将关闭。 （%..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。）
:%...I:	收到的字节数（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:	发送的字节数（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。“...”可以为空（如通用规范“\%h %u %r \%s %b”中所述），也可以表示包含该项目的条件。条件是 HTTP 状态代码列表，前面可能带有“!”，如果不满足指定的条件，则返回的列或字段将显示为“-”。

例如，如 Apache 文档中所述，指定 “%400,501 [User-agent] i” 将 User-agent 仅记录 400 个错误和 501 个错误（错误请求，未实现）。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，而所有其他指令则查看最终请求。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，以及所有以其 C-style表示法（\n、\ t 等）书写的空白字符。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，因为客户端本可以在日志中插入控制字符。
此外，在 httpd 2.0 中，B 格式的字符串仅表示 HTTP 响应的大小（以字节为单位）（例如，如果连接中止或使用 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数、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	
未发送字节时使用“0”	%...B
未发送字节时使用“-”（CLF 格式）	%...b
收到的字节数（包括请求和标头）不能为零 必须启用 mod_logio 才能使用此项。	:% ... I:
发送的字节数（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	
连接在响应完成前已中止	X
连接在响应发送后可能会保持活动状态	+
连接在响应发送后将关闭	-

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。

函数或类别	W3C 格式说明符
环境变量 CUSTOMERDATA	
内容	%...[CUSTOMERDATA]e
文件名	%...f
主机（远程）	%...h
协议	%...H
IP 地址	
远程	%...a
本地	%...A
备注	
来自其他模块的备注 Customerdata 的内容	%...[Customerdata]n
协议（请求）	%...H
查询字符串 Note 如果存在查询，则在前面加上？ 如果不存在，则为空字符串。	%...q
回复	
Customerdata（回复中的标头行）的内容	%...[Customerdata]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	
处理请求的服务器的规范端口	%...p
发送到服务器的请求中 cookie Customerdata 的内容	%... [Customerdata]C
各 BAR:header 行的内容	%... [BAR]i
第一行已发送：	%...r
处理请求所花费的时间（以微秒为单位）	%...D
协议	%...H
处理请求的子项的进程 ID	%...P
处理请求的子项的进程 ID 或线程 ID。 有效格式为 pid 和 tid。（Apache 2.0.46 及更高版本）	%...[format]P
远程登录名（如果提供，则来自 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
为请求提供服务的服务器（规范 ServerName）	%...v
按 UseCanonicalName 设置排列的服务器名称	%...V
请求方法	%...m
返回状态	%s
处理请求所花费的时间（以秒为单位）	%...T
内部重定向的*原始*请求的状态	%...s

函数或类别	W3C 格式说明符
上次请求的状态	%...>s
请求的 URL 路径，不包括任何查询字符串	%...U
时间	
通用日志格式的时间格式（标准英文格式）	%...t
时间采用 strftime(3) 格式，可能已本地化	%...[format]t
处理请求所花费的时间（以秒为单位）	%...T

W3C 示例

W3C_LOG_PARSE 支持访问由 Apache Web 服务器等 W3C-compliant 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

此示例中的输入取自 Apache 日志文件，代表通用日志格式。

Input

```
(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);
```

Output

```
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 子句中的 COMMON 规范是指通用日志格式 (CLF)，使用说明符 %h %l %u %t "%r" %>s %b。

这些W3C-predefined 格式显示了 COMMON 和其他预定义的说明符集。

FROM 子句中的 COMMON 规范是指通用日志格式 (CLF)，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”列出了 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
COLUMN1	%h	远程主机的 IP 地址
COLUMN2	%l	远程登录名
COLUMN3	%u	远程用户
COLUMN4	%t	时间
COLUMN5	"%r"	请求的第一行
COLUMN6	%>s	状态：对于内部重定向的请求， 是*原始*请求的状态 而对于最后一个，是 %...>s。

输出列	格式说明符	返回值
COLUMN7	%b	发送的字节数，不包括 HTTP 标头

示例 2

此示例中的 DDL 显示了如何重命名输出列和筛选出不需要的列。

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;
```

Output

```
+----------+-----------------+---------------------------------+
| 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”名称可以生成相同的结果，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

本节中的主题介绍了 Amazon Kinesis Data Analytics 流式 SQL 的排序函数。

主题

- Group Rank

Group 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。
- 允许用户指定数据刷新的非活动时间段。

SQL 声明

函数属性和 DDL 将在后面的部分中介绍。

的功能属性 Group_Rank

此函数的作用如下：

- 收集行，直至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。

DDL 用于 Group_Rank

```
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)
```

下表中列出了该函数的参数。

参数	说明
c	指向流式结果集的光标
rankByColumnName	命名要用于对组进行排名的列的字符串。
rankOutColumnName	命名要用于返回排名的列的字符串。

参数	说明
	此字符串必须与 CREATE FUNCTION 语句的 RETURNS 子句中 groupRank 列的名称匹配。
sortOrder	控制行的排序以进行排名分配。 有效值如下所示： • 'asc' - 根据排名，按升序。 • 'desc' - 根据排名，按降序。
outputOrder	控制输出的排序。有效值如下所示： • 'asc' - 根据排名，按升序。 • 'desc' - 根据排名，按降序。
maxIdle	保留组以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。 值为 0 表示没有限制。

示例

示例数据集

以下示例基于样本股票数据集，后者是 Amazon Kinesis Data Analytics 开发人员指南中的入门练习的一部分。要运行每个示例，您需要一个具有样本股票代码输入流的 Amazon Kinesis Data Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Data Analytics 开发人员指南中的入门。

样本股票数据集的架构如下：

```
(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

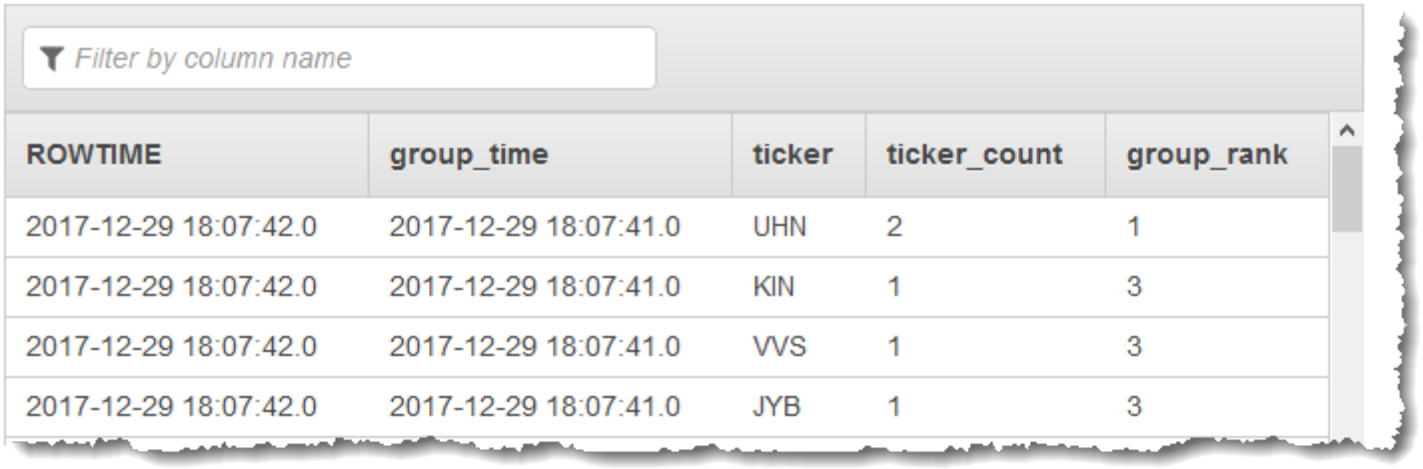
在此示例中，聚合查询在 ROWTIME 上有一个 GROUP BY 子句，可将流分组到有限行中。然后，GROUP_RANK 函数对 GROUP BY 子句返回的行进行排序。

```
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

操作概述

通过输入光标为每个组（即行时间相同的行）缓冲行。在行时间不同的行到达之后（或出现空闲超时之时）对行进行排名。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.columName”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

这些函数都接受一组数字，忽略 null，并且既可用作聚合函数，也可用作分析函数。有关更多信息，请参阅聚合函数和分析函数。

下表列出了这些函数之间的关系。

函数目的	函数名称	Formula	评论
热点	HOTSPOTS (expr)	检测数据流中频繁出现的数据的热点。	
Random Cut Forest	RANDOM_CUT_FOREST (expr)	在数据流中检测异常。	
随机砍伐森林以及解释	RANDOM_CUT_FOREST_WITH_EXPLANATION (expr)	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(expr)	(SUM(expr*expr) - SUM(expr)*SUM(expr) / COUNT(expr)) / COUNT(expr)	如果应用于空集，则返回 null。
总体标准差	STDDEV_POP	总体方差的平方根 (VAR_POP)。	当 VAR_POP 返回 null 时，STDDEV_POP 返回 null。
样本方差	VAR_SAMP	(SUM(expr*expr) - SUM(expr)*SUM(expr) / COUNT(expr)) / (COUNT(expr)−1)	如果应用于空集，则返回 null。 如果应用于由一个元素组成的输入集，则 VAR_SAMP 返回 null。
样本标准差	STDDEV_SAMP (expr)	样本方差的平方根 (VAR_SAMP)。	如果仅应用于 1 行输入数据，则 STDDEV_SAMP 返回 null。

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**
>
> HOTSPOTS 函数检测频繁数据点的能力取决于应用程序。要确定业务问题以便通过此函数解决，需要具备领域专业知识。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 和 BIGINT 数据类型。DECIMAL 不是受支持的类型。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。

inputStream

指向输入流的指针。您可以使用 CURSOR 函数设置指针。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**
>
> 仅对输入流中的数字列进行热点分析。HOTSPOTS 函数忽略游标中包含的其他列。

windowSize

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 DBSCAN 算法中的 ε 值。

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。

最低限度 NumberOfPointsInAHotspot

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windowSize。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。

Output

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 服务进行服务维护时，机器学习功能用于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。有关在包含有意义的数据热点的自定义数据流上执行 HOTSPOTS 函数的示例，请参阅示例：检测热点。

示例数据集

以下示例基于样本股票数据集，后者是 Amazon Kinesis Data Analytics 开发人员指南中的入门练习的一部分。要运行此示例，您需要一个具有样本股票代码输入流的 Kinesis Data Analytics 应用程序。要了解如何创建 Kinesis Data Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Data Analytics 开发人员指南中的入门。

样本股票数据集的架构如下：

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

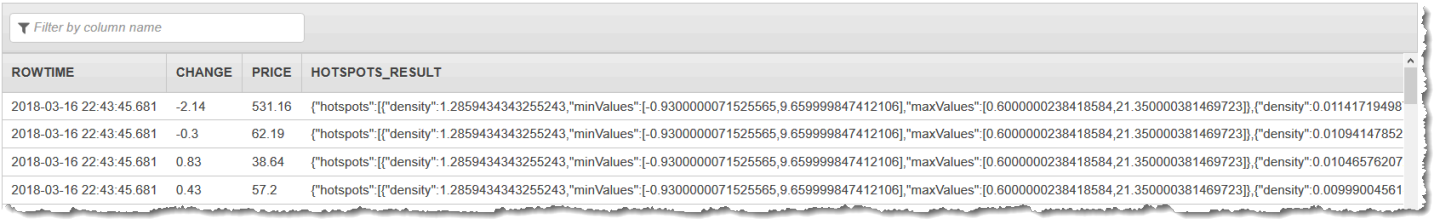
示例 1：返回示例数据流上的热点

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

此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":[0.6000000238418584,21.350000381469723]},{"density":0.01141719498
2018-03-16 22:43:45.681	-0.3	62.19	{"hotspots":[{"density":1.2859434343255243,"minValues":[-0.9300000071525565,9.659999847412106],"maxValues":[0.6000000238418584,21.350000381469723]},{"density":0.01094147852
2018-03-16 22:43:45.681	0.83	38.64	{"hotspots":[{"density":1.2859434343255243,"minValues":[-0.9300000071525565,9.659999847412106],"maxValues":[0.6000000238418584,21.350000381469723]},{"density":0.01046576207
2018-03-16 22:43:45.681	0.43	57.2	{"hotspots":[{"density":1.2859434343255243,"minValues":[-0.9300000071525565,9.659999847412106],"maxValues":[0.6000000238418584,21.350000381469723]},{"density":0.00999004561

RANDOM_CUT_FOREST

在数据流中检测异常。 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。

> **Note**
>
> RANDOM_CUT_FOREST 函数检测异常的能力取决于应用程序。要确定业务问题以便通过此函数解决，需要具备领域专业知识。例如，确定要将输入流中的哪些列组合传递给此函数，并可能对数据进行规范化。有关更多信息，请参阅 inputStream。

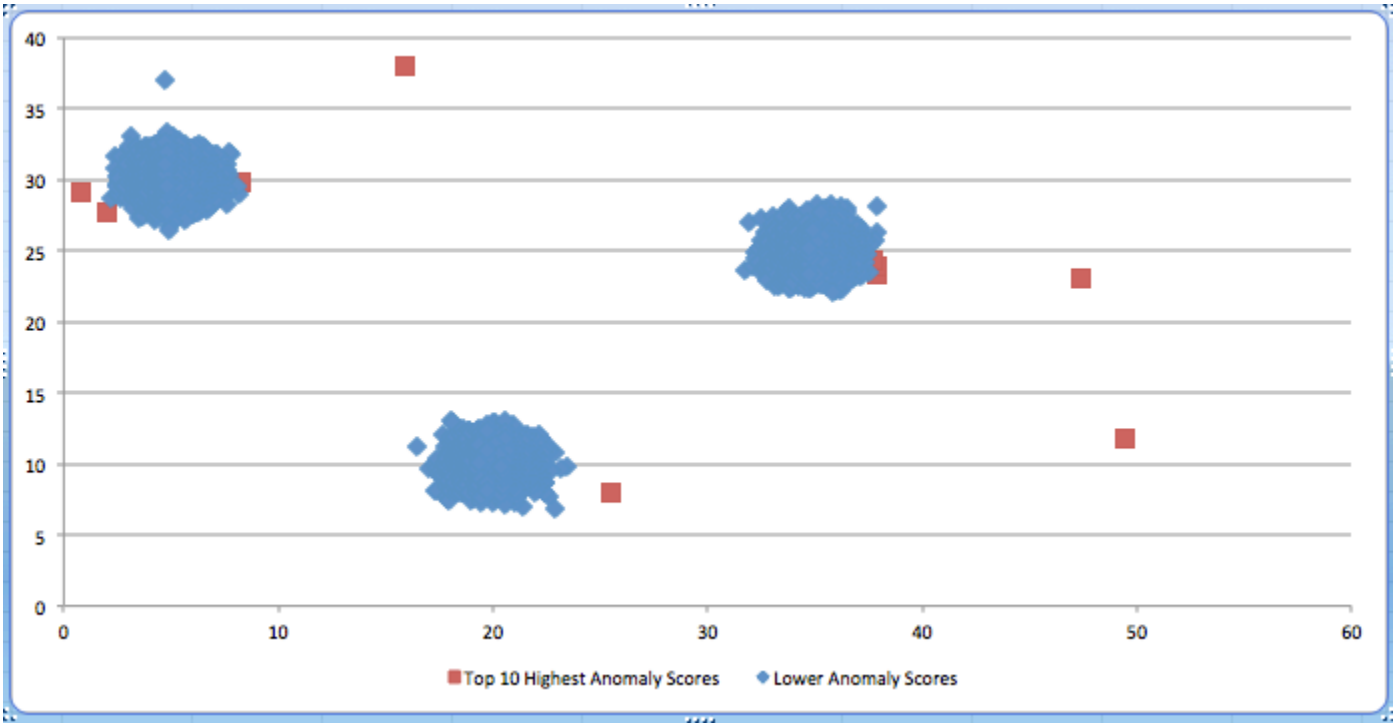
流记录可以有非数字列，但该函数仅使用数字列来分配异常分数。一条记录可以有一个或多个数字列。该算法使用所有数字数据来计算异常分数。 如果一个记录有 n 个数字列，底层算法将假定每个记录都是 n 维空间中的一个点。n 维空间中与其他点相距较远的点将获得较高的异常分数。

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 和 BIGINT 数据类型。

> **Note**
>
> DECIMAL 不是受支持的类型。请改用 DOUBLE。

以下是异常检测的示例。该图显示了三个集群和几个随机插入的异常。根据 RANDOM_CUT_FOREST 函数，红色方块显示获得最高异常分数的记录。蓝色菱形代表剩余的记录。 请注意分数最高的记录如何倾向于位于集群外面。



有关带分步指导的示例应用程序，请参阅检测异常情况。

语法

```
RANDOM_CUT_FOREST (inputStream,
                   numberOfTrees,
                   subSampleSize,
                   timeDecay,
                   shingleSize)
```

参数

以下各节介绍了这些参数。

inputStream

指向输入流的指针。您可以使用 CURSOR 函数设置指针。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)
```

CURSOR 函数是 RANDOM_CUT_FOREST 函数的唯一必需参数。该函数假设其他参数的默认值如下：

数字 OfTrees = 100

sub SampleSize = 256

timeDecay = 100,000

shingleSize = 1

使用此函数时，输入流最多可以有 30 个数字列。

数字 OfTrees

使用此参数，您可以指定森林中随机砍伐的树木数量。

> **Note**
>
> 默认情况下，该算法会构造许多树，每棵树都是使用输入流中给定数量的样本记录（请参阅本列表后面的 subSampleSize）构造的。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。

numberOfTrees 的默认值为 100。您可以将此值设置为介于 1 和 1000 之间（含 1 和 1000）。通过增加森林中树的数量，您可以获得异常分数的更准确估算，但这也会延长运行时间。

潜艇 SampleSize

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。

subSampleSize 的默认值为 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（默认情况下设置为 100000）。 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。

timeDecay

timeDecay 参数允许您指定计算异常分数时要考虑过去的多长时间。这是因为数据流会随着时间的推移而自然演变。例如，随着时间的推移，电子商务网站的收入可能会不断增加，或者全球温度可能会逐渐升高。 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 100000 条记录（如果使用瓦形，则为 100000 个瓦形，如下一节所述）。您可以将此值设置为介于 1 和最大整数（即 2147483647）之间。该算法以指数方式降低了旧数据的重要性。

如果您选择 timeDecay 的默认值 100000，则异常检测算法将执行以下操作：

- 在计算中仅使用最近的 100000 条记录（并忽略较早的记录）。
- 在最近的 100000 条记录中，进行异常检测计算时，近期记录的权重呈指数级增长，而较早记录的权重则呈指数级下降。

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)。

timeDecay 参数决定了在异常检测算法的工作集中保留的最大最近记录数量。 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。

shingleSize

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。

瓦形是最近记录的连续序列。 例如，时间 t 处大小为 10 的 shingleSize 对应于截至时间 t（含该时间）收到的最后 10 条记录的向量。 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。

直觉告诉我们，瓦形可以捕获近期的形状。 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。

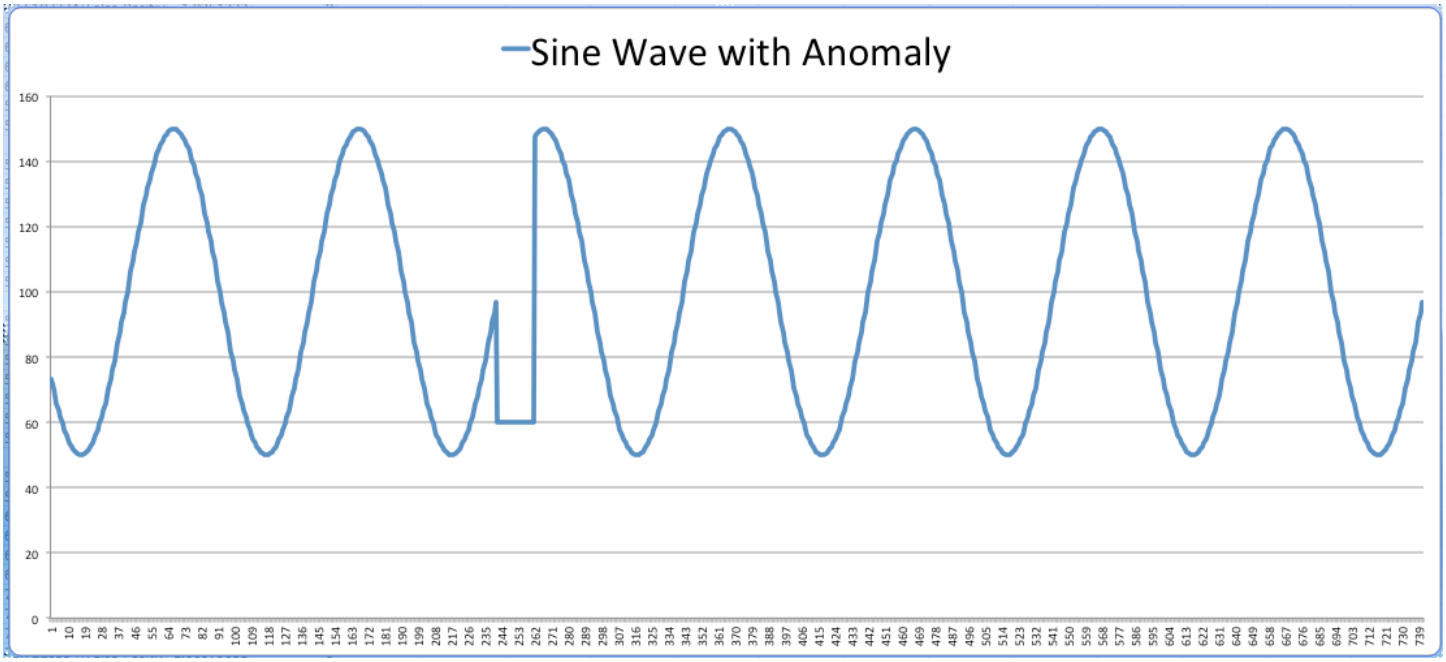
默认 shingleSize 是一条记录（因为瓦形大小取决于数据）。您可以将此值设置为介于 1 和 30 之间（含 1 和 30）。

请注意有关设置 shingleSize 的以下内容：

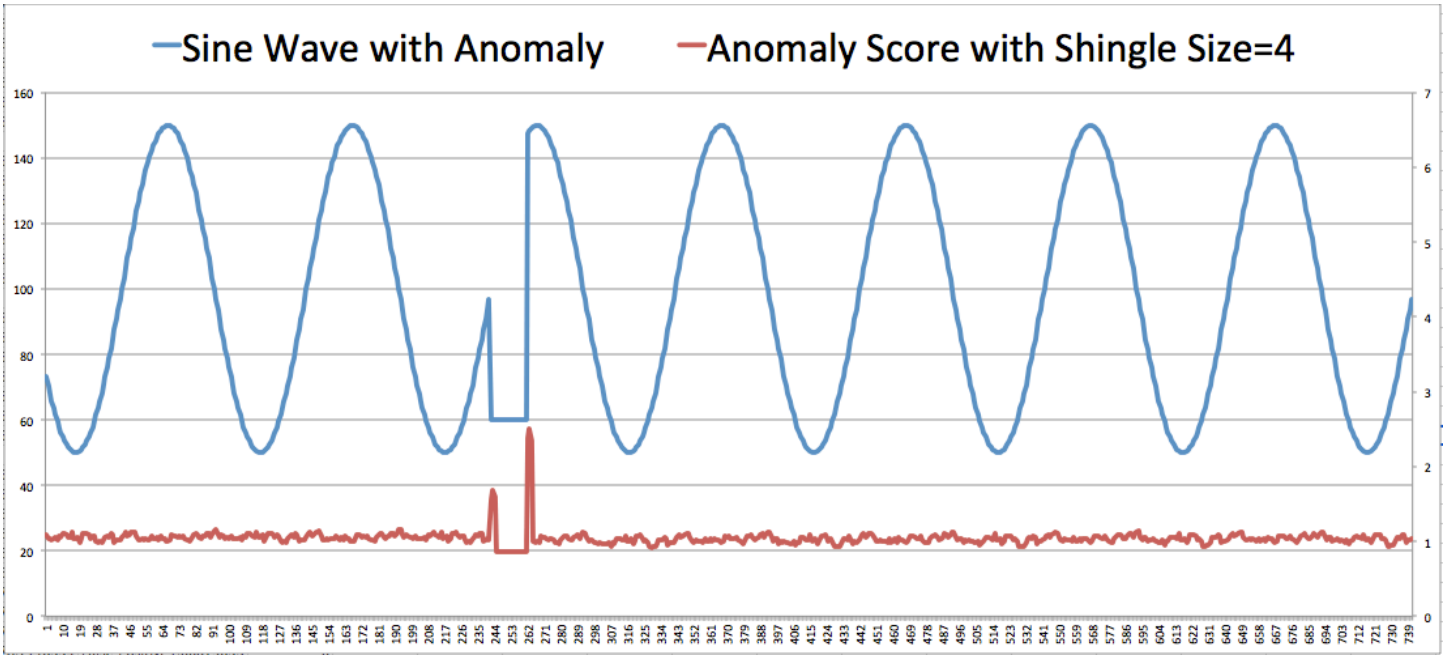
- 如果将 shingleSize 设置得过小，算法更容易受数据的细微波动的影响，从而导致并非异常的记录获得高异常分数。
- 如果将 shingleSize 设置得过大，则可能需要更多时间来检测异常记录，因为非异常的瓦形中有更多记录。 确定异常情况已结束也可能需要更长时间。
- 确定正确的瓦形大小取决于应用程序。 请试验不同的瓦形大小以确定影响。

以下示例说明了在监控异常分数最高的记录时如何捕获异常。在这个特殊的示例中，两个最高的异常分数也表示人为注入的异常的开始和结束。

考虑这种以正弦波表示的程式化一维流，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。



我们使用四条记录的瓦形大小运行 RANDOM_CUT_FOREST 函数。结果如下所示。红线表示异常分数。 请注意，异常的开头和结尾获得高分数。



使用此函数时，建议您将最高分数作为潜在异常进行调查。

> **Note**
>
> 当 Kinesis Data Analytics 服务进行服务维护时，机器学习功能用于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。您可能还想知道变化在多大程度上是由于每小时购买数量的变化而引起的，在多大程度上是由于每小时放弃的购物车数量的变

化而引起的，这些是归因分数表示的信息。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**
>
> RANDOM_CUT_FOREST_WITH_EXPLANATION 函数检测异常的能力取决于应用程序。要确定业务问题以便通过此函数解决，需要具备领域专业知识。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 inputStream。

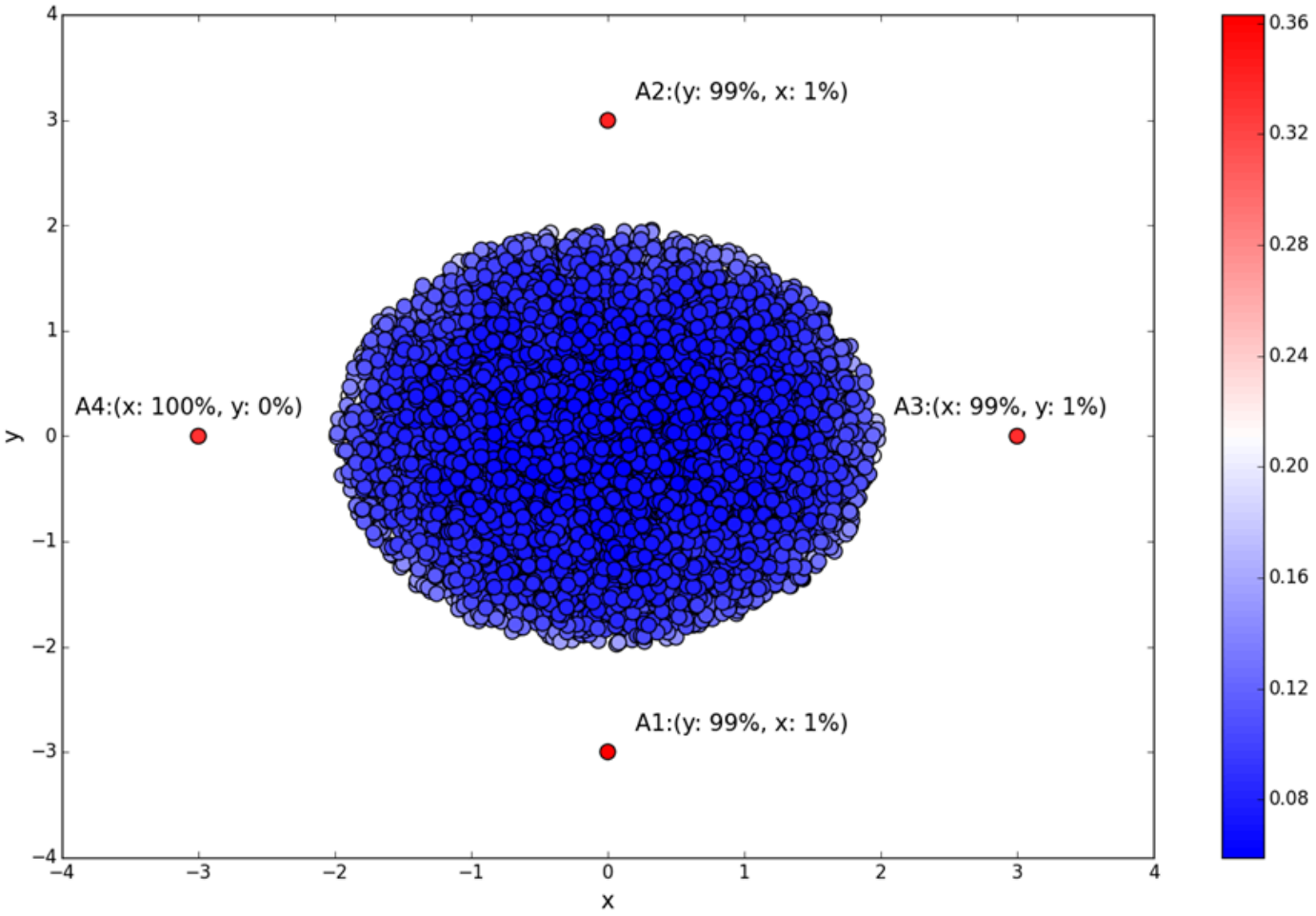
流记录可以有非数字列，但该函数仅使用数字列来分配异常分数。一条记录可以有一个或多个数字列。算法使用所有数值数据来计算异常分数。

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 和 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直观示例。该图显示了一个由蓝色数据点组成的集群和四个显示为红点的异常值。红点具有相似的异常分数，但这四个点异常的原因不同。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。

inputStream

指向输入流的指针。您可以使用 CURSOR 函数设置指针。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)
```

CURSOR 函数是 RANDOM_CUT_FOREST_WITH_EXPLANATION 函数的唯一必需参数。该函数假设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。

数字 OfTrees

使用此参数，您可以指定森林中随机砍伐的树木数量。

> **Note**
>
> 默认情况下，该算法会构造许多树，每棵树都是使用输入流中给定数量的样本记录（请参阅本列表后面的 subSampleSize）构造的。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。

numberOfTrees 的默认值为 100。您可以将此值设置为介于 1 和 1000 之间（含 1 和 1000）。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。

潜艇 SampleSize

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。

subSampleSize 的默认值为 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。

timeDecay

您可以使用 timeDecay 参数指定计算异常分数时要考虑过去的多长时间。数据流会随着时间的推移而自然演变。例如，随着时间的推移，电子商务网站的收入可能会不断增加，或者全球温度可能会逐渐升高。在这种情况下，您希望根据最近的数据而不是遥远过去的数据来标记异常。

默认值为 100000 条记录（如果使用瓦形，则为 100000 个瓦形，如下一节所述）。您可以将此值设置为介于 1 和最大整数（即 2147483647）之间。该算法以指数方式降低了旧数据的重要性。

如果您选择 timeDecay 的默认值 100000，则异常检测算法将执行以下操作：

- 在计算中仅使用最近的 100000 条记录（并忽略较早的记录）。
- 在最近的 100000 条记录中，进行异常检测计算时，近期记录的权重呈指数级增长，而较早记录的权重则呈指数级下降。

timeDecay 参数决定了在异常检测算法的工作集中保留的最大最近记录数量。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。

shingleSize

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。

瓦形是最近记录的连续序列。例如，时间 t 处大小为 10 的 shingleSize 对应于截至时间 t（含该时间）收到的最后 10 条记录的向量。该算法将此序列视为跨最后 shingleSize 条记录的向量。

如果数据以统一的时间到达，则时间 t 处大小为 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。

直觉告诉我们，瓦形可以捕获近期的形状。您的数据可能有典型的形状。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 `shingleSize` 是一个记录（因为瓦形大小取决于数据）。您可以将此值设置为介于 1 和 30 之间（含 1 和 30）。

请注意有关设置 `shingleSize` 的以下内容：

- 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。
- 如果将 `shingleSize` 设置得过大，则可能需要更多时间来检测异常记录，因为非异常的瓦形中有更多记录。此外，还可能需要更多时间才能确定异常已经结束。
- 确定正确的瓦形大小取决于应用程序。使用不同的瓦形大小进行实验来确定效果。

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：指示此列对记录的异常分数的贡献程度的非负数。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。
- 强度：表示方向推荐强度的非负数。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：如果列的值高于最近观察到的趋势，则为 HIGH（高）；如果低于趋势，则为 LOW（低）。在学习阶段，此值默认为 LOW（低）。

> **Note**
>
> 当 Kinesis Data Analytics 服务进行服务维护时，机器学习功能用于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，后者是 Amazon Kinesis Analytics 开发人员指南中的入门练习的一部分。要运行此示例，您需要一个具有样本股票代码输入流的 Kinesis Data Analytics 应用程序。要了解如何创建 Kinesis Data Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Data Analytics 开发人员指南中的入门。

样本股票数据集的架构如下：

```
(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":"0.3780"},"PRICE":{"DIRECTION":"HIGH","STRENGTH":"0.0675","ATTRIBUTION_SCORE":"0.1953"}}
2017-10-23 20:09:21.466	0.73723656	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.3178","ATTRIBUTION_SCORE":"0.3757"},"PRICE":{"DIRECTION":"HIGH","STRENGTH":"0.1695","ATTRIBUTION_SCORE":"0.3616"}}
2017-10-23 20:09:21.466	0.6443901	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.1731","ATTRIBUTION_SCORE":"0.3396"},"PRICE":{"DIRECTION":"LOW","STRENGTH":"0.2367","ATTRIBUTION_SCORE":"0.3048"}}
2017-10-23 20:09:21.466	0.55428815	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.0888","ATTRIBUTION_SCORE":"0.2327"},"PRICE":{"DIRECTION":"LOW","STRENGTH":"0.1295","ATTRIBUTION_SCORE":"0.3216"}}
2017-10-23 20:09:21.466	0.5416738	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.0316","ATTRIBUTION_SCORE":"0.2805"},"PRICE":{"DIRECTION":"HIGH","STRENGTH":"0.0208","ATTRIBUTION_SCORE":"0.2612"}}
2017-10-23 20:09:21.466	0.67421293	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.3364","ATTRIBUTION_SCORE":"0.3058"},"PRICE":{"DIRECTION":"LOW","STRENGTH":"0.2090","ATTRIBUTION_SCORE":"0.3684"}}
2017-10-23 20:09:21.466	0.6528528	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.2020","ATTRIBUTION_SCORE":"0.3746"},"PRICE":{"DIRECTION":"HIGH","STRENGTH":"0.0107","ATTRIBUTION_SCORE":"0.2783"}}
2017-10-23 20:09:21.466	0.83734107	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.0177","ATTRIBUTION_SCORE":"0.2954"},"PRICE":{"DIRECTION":"LOW","STRENGTH":"0.0204","ATTRIBUTION_SCORE":"0.5419"}}
2017-10-23 20:09:21.466	0.9346224	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.2364","ATTRIBUTION_SCORE":"0.2299"},"PRICE":{"DIRECTION":"LOW","STRENGTH":"0.3877","ATTRIBUTION_SCORE":"0.7047"}}
2017-10-23 20:09:21.466	0.6726167	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.0397","ATTRIBUTION_SCORE":"0.3335"},"PRICE":{"DIRECTION":"HIGH","STRENGTH":"0.0631","ATTRIBUTION_SCORE":"0.3392"}}
2017-10-23 20:09:21.466	0.6636287	{"CHANGE":{"DIRECTION":"LOW","STRENGTH":"0.0571","ATTRIBUTION_SCORE":"0.3147"},"PRICE":{"DIRECTION":"HIGH","STRENGTH":"0.1943","ATTRIBUTION_SCORE":"0.3489"}}
2017-10-23 20:09:21.466	0.7353514	{"CHANGE":{"DIRECTION":"HIGH","STRENGTH":"0.1851","ATTRIBUTION_SCORE":"0.1992"},"PRICE":{"DIRECTION":"HIGH","STRENGTH":"0.0564","ATTRIBUTION_SCORE":"0.5361"}}

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。

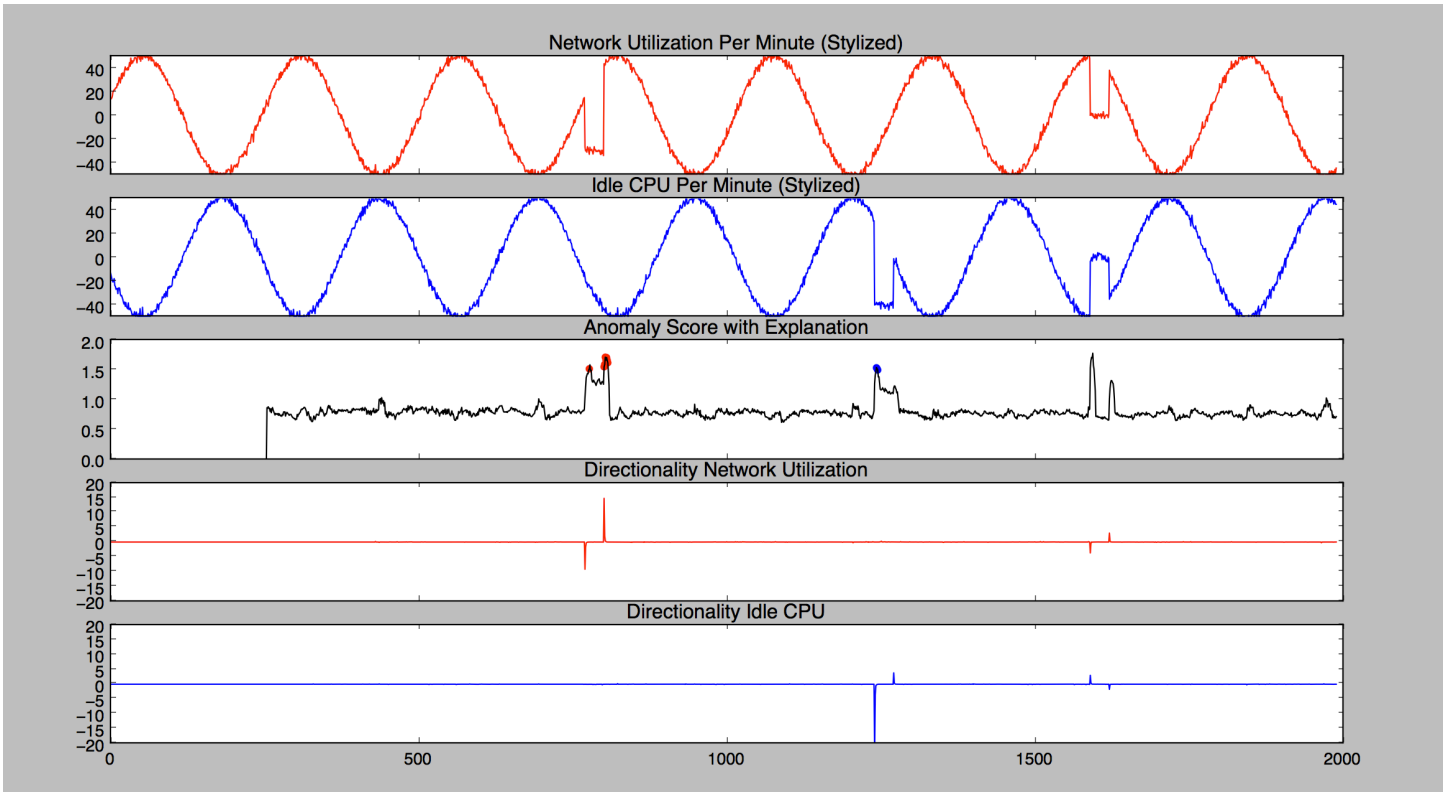
  第一个异常在黑色异常分数曲线上以红色标记，它更多地归因于网络利用率数据。第二个异常标记为蓝色，它更多地归因于 CPU 数据。此图中提供红色和蓝色标记以更好地显示效果。它们不是由 RANDOM_CUT_FOREST_WITH_EXPLANATION 函数生成的。以下是获得这些红色和蓝色标记的方法：

  - 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。



血压示例

有关更详细的示例以及检测和解释血压读数异常的代码，请参阅示例：检测数据异常并获取解释。

STDDEV_POP

返回为组中其余每行计算的 <number expression> 的 VAR_POP 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。
- 如果您未使用 OVER 子句，则 STDDEV_POP 将作为聚合函数进行计算。在这种情况下，聚合查询必须根据将流分组到有限行中的 ROWTIME 在单调表达式中包含 GROUP BY 子句。否则，组将是无限流，并且查询永远无法完成，也不会输出任何行。有关更多信息，请参阅 聚合函数。
- 使用 GROUP BY 子句的窗口式查询在滚动窗口中处理行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您使用 OVER 子句，则 STDDEV_POP 将作为分析函数进行计算。有关更多信息，请参阅 分析函数。
- 使用 OVER 子句的窗口式查询在滑动窗口中处理行。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，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，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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：确定滚动窗口查询内的列中的总体标准差

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）

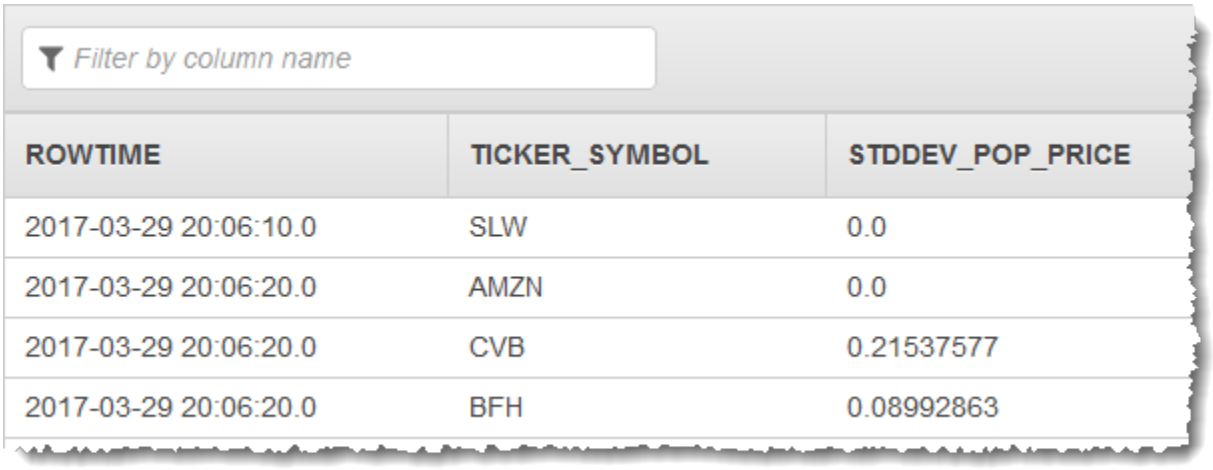
```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 stddev_pop_price REAL);

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：


Filter by column name

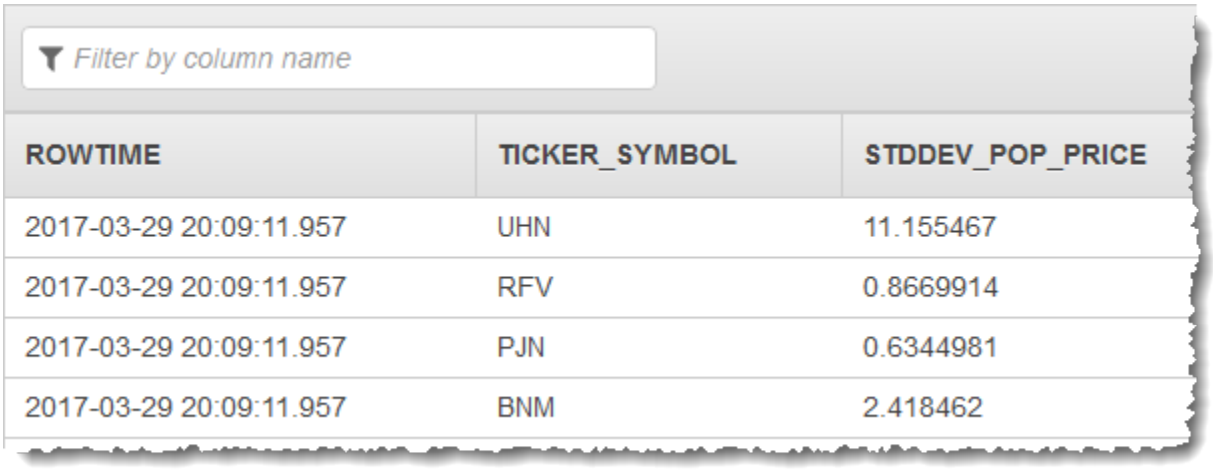
ROWTIME	TICKER_SYMBOL	STDDEV_POP_PRICE
2017-03-29 20:06:10.0	SLW	0.0
2017-03-29 20:06:20.0	AMZN	0.0
2017-03-29 20:06:20.0	CVB	0.21537577
2017-03-29 20:06:20.0	BFH	0.08992863

示例 2：确定滑动窗口查询内的列中的值的总体标准差

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
- 样本方差: VAR_SAMP
- 总体方差: VAR_POP

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 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。
- 如果您未使用 OVER 子句，则 STDDEV_SAMP 将作为聚合函数进行计算。在这种情况下，聚合查询必须根据将流分组到有限行中的 ROWTIME 在单调表达式中包含 GROUP BY 子句。否则，组将是无限流，并且查询永远无法完成，也不会输出任何行。有关更多信息，请参阅 聚合函数。
- 使用 GROUP BY 子句的窗口式查询在滚动窗口中处理行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您使用 OVER 子句，则 STDDEV_SAMP 将作为分析函数进行计算。有关更多信息，请参阅 分析函数。
- 使用 OVER 子句的窗口式查询在滑动窗口中处理行。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，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，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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：确定滚动窗口查询内的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

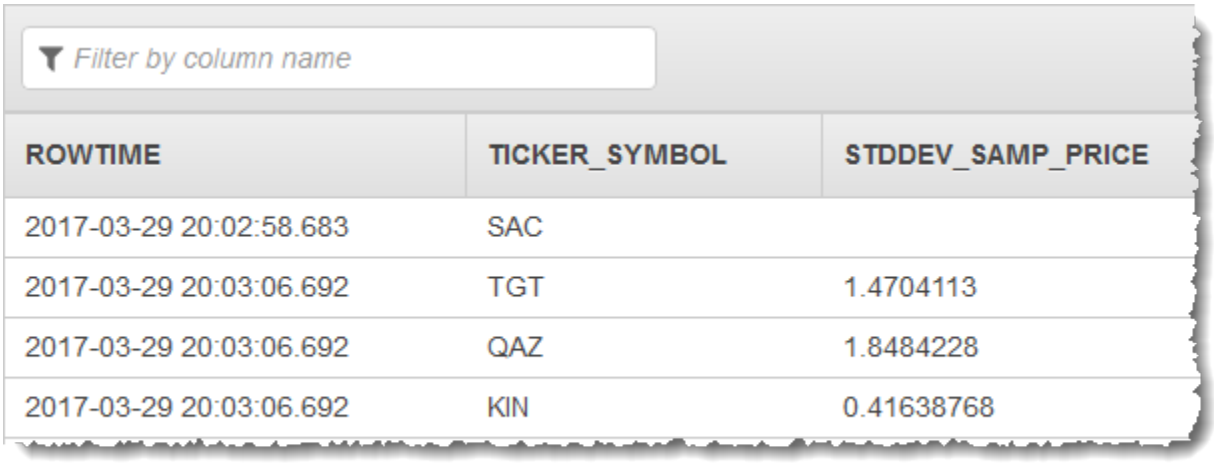
ROWTIME	TICKER_SYMBOL	STDDEV_SAMP_PRICE
2017-03-29 22:23:30.0	AMZN	7.6294384
2017-03-29 22:23:30.0	WSB	54.68945
2017-03-29 22:23:30.0	JKL	0.08468548
2017-03-29 22:23:30.0	QXZ	68.81256

示例 2：确定滑动窗口查询内的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
- 样本方差: VAR_SAMP
- 总体方差: VAR_POP

VAR_POP

返回数字的非 null 集的总体方差（null 值被忽略）

VAR_POP 使用以下计算方式：

- (SUM(expr*expr) - SUM(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*) /N。S1/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。
- 如果您未使用 OVER 子句，则 VAR_POP 将作为聚合函数进行计算。在这种情况下，聚合查询必须根据将流分组到有限行中的 ROWTIME 在单调表达式中包含 GROUP BY 子句。否则，组将是无限流，并且查询永远无法完成，也不会输出任何行。有关更多信息，请参阅 聚合函数。
- 使用 GROUP BY 子句的窗口式查询在滚动窗口中处理行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。

- 如果您使用 OVER 子句，则 VAR_POP 将作为分析函数进行计算。有关更多信息，请参阅 分析函数。
- 使用 OVER 子句的窗口式查询在滑动窗口中处理行。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，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，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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：确定滚动窗口查询内的列中的总体方差

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）

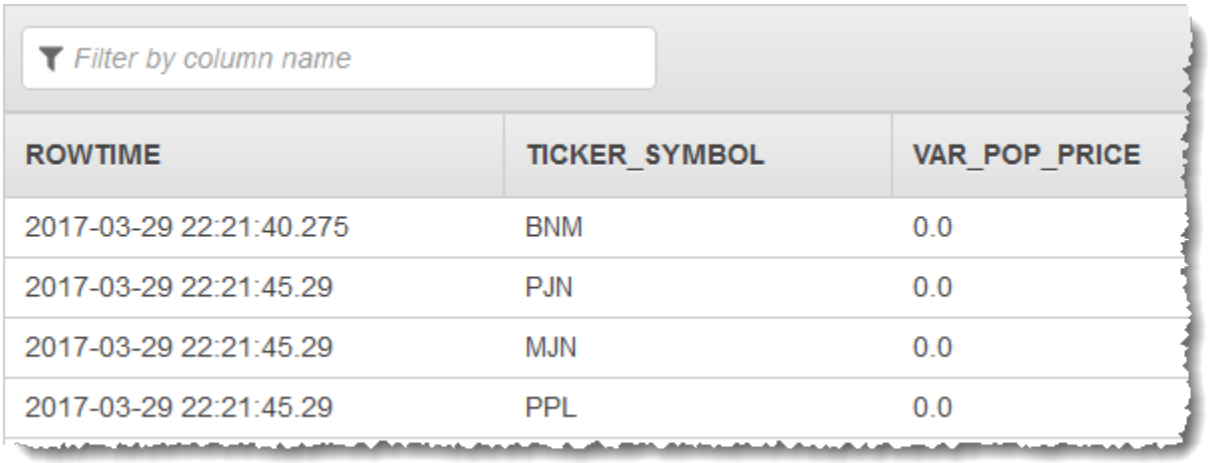
```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (ticker_symbol VARCHAR(4),
 var_pop_price REAL);

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：



示例 2：确定滑动窗口查询内的列中的值的总体方差

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
- 样本标准差: STDDEV_SAMP
- 样本方差: VAR_SAMP

VAR_SAMP

返回数字的非 null 集的样本方差 (null 值被忽略)。

VAR_SAMP 使用以下计算方式：

- (SUM(expr*expr) - SUM(expr)*SUM(expr) / COUNT(expr)) / (COUNT(expr)-1)

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_SAMP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。
- 如果您未使用 OVER 子句，则 VAR_SAMP 将作为聚合函数进行计算。在这种情况下，聚合查询必须根据将流分组到有限行中的 ROWTIME 在单调表达式中包含 GROUP BY 子句。否则，组将是无限流，并且查询永远无法完成，也不会输出任何行。有关更多信息，请参阅 聚合函数。
- 使用 GROUP BY 子句的窗口式查询在滚动窗口中处理行。有关更多信息，请参阅滚动窗口（使用 GROUP BY 的聚合）。
- 如果您使用 OVER 子句，则 VAR_SAMP 将作为分析函数进行计算。有关更多信息，请参阅 分析函数。
- 使用 OVER 子句的窗口式查询在滑动窗口中处理行。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，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，您需要一个具有样本股票代码输入流的 Amazon Kinesis Analytics 应用程序。要了解如何创建 Analytics 应用程序和配置样本股票代码输入流，请参阅 Amazon Kinesis Analytics 开发人员指南中的入门。

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：确定滚动窗口查询内的列中的样本方差

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);
```

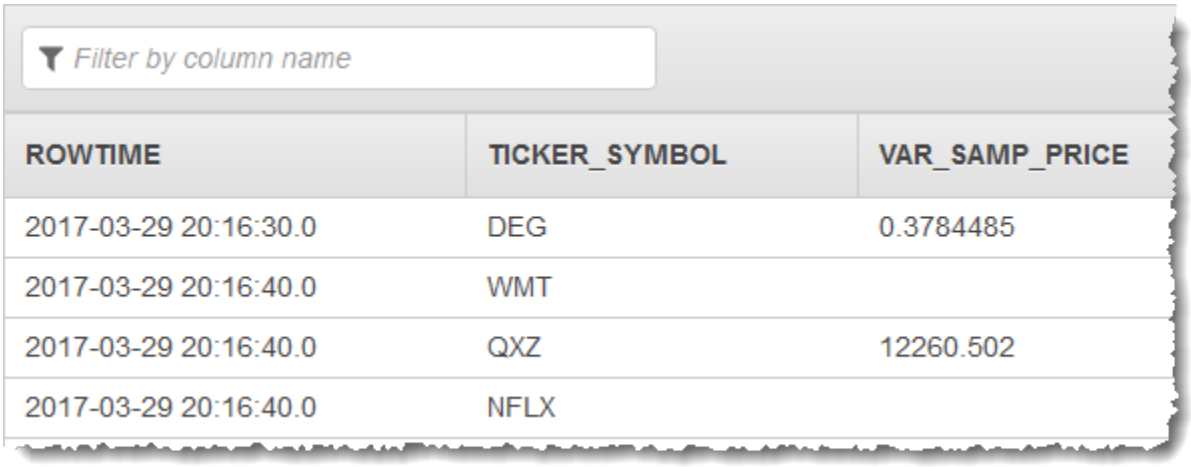
使用 FLOOR

```
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：

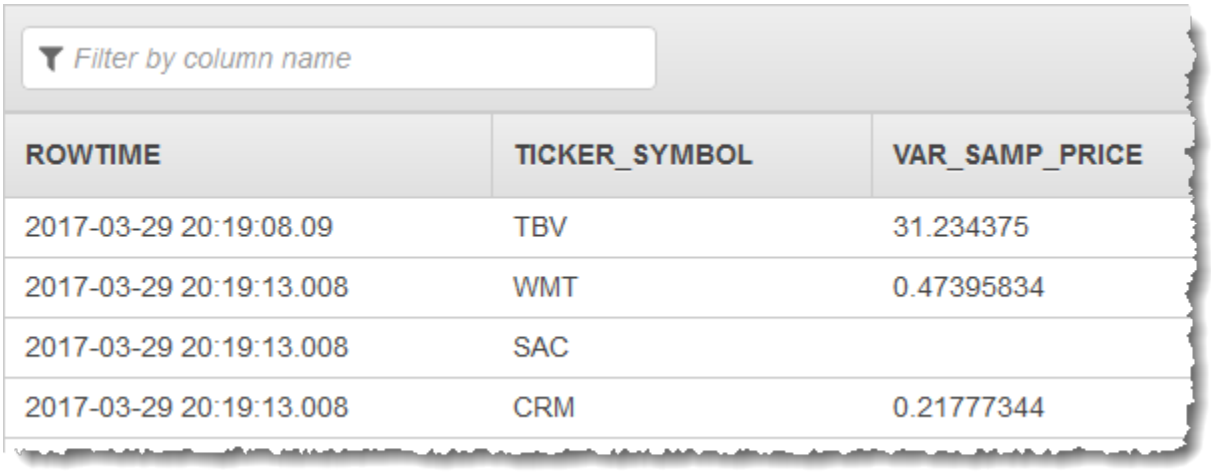


示例 2：确定滑动窗口查询内的列中的值的样本方差

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
- 样本标准差: STDDEV_SAMP
- 总体方差:VAR_POP

流式处理 SQL 函数

本节中的主题介绍了 Amazon Kinesis Data Analytics 流式 SQL 的流函数。

主题

- LAG
- 单调函数
- NTH_VALUE

LAG

LAG 为记录返回表达式的计算结果（例如列名称），它是给定窗口中的当前记录之前的 N 条记录。根据当前记录计算偏移量和默认值。如果没有此类记录，LAG 将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的数目。默认值为 1。

defaultExpr

在查询的记录（当前记录之前的 n 条）位于窗口外部的情况下返回的与 expr 类型相同的表达式。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

一个指定在确定偏移量时不计入 null 值的子句。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

一个指定在确定偏移量时计入 null 值的子句。此行为是默认行为。

OVER window-specification

一个划分流中按时间范围间隔或记录数分区的记录的子句。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，您需要一个具有样本股票代码输入流的 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：在 OVER 子句中返回前面的记录中的值

在此示例中，OVER 子句划分流中按之前“1”分钟的时间范围间隔分区的记录。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	PREVIOUS_PRICE	PREVIOUS_PRICE_2
2017-05-15 22:48:45.929	DFT	74.61000061035156	72.94999694824219	72.5
2017-05-15 22:48:50.901	IOP	107.80999755859375	108.73999786376953	110.80000305175781
2017-05-15 22:48:50.901	NGC	4.619999885559082	4.550000190734863	4.510000228881836
2017-05-15 22:48:50.901	NFLX	101.54000091552734	101.76000213623047	102.19999694824219

注意

LAG 不是 SQL:2008 标准的一部分。它是 Amazon Kinesis Data Analytics 流式 SQL 扩展。

单调函数

```
MONOTONIC(<expression>)
```

流式 GROUP BY 要求至少有一个分组表达式是单调且非常量的。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。

MONOTONIC 函数允许您声明给定表达式是单调的，从而使流式 GROUP BY 能够使用该表达式作为键。

MONOTONIC 函数会计算其参数并返回结果（与其参数的类型相同）。

通过将表达式包含在 MONOTONIC 中，即表示您断言该表达式的值要么不增加，要么不减少，而且永远不会改变方向。例如，如果您有一个由订单行项目组成的流 LINEITEMS，并且您编写了 MONOTONIC(orderId)，那么您就是在断言行项目在该流中是连续的。如果有订单 1000 的行项目，然后是订单 1001 的行项目，再然后是订单 1005 的行项目，那就没问题。如果接着有订单 1001 的行项目（也就是说，行项目序列变为 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，但以下行项目序列是非法的：

- 987、974、823、973
- 987、974、823、1056

声明为单调的表达式可以减少，甚至具有任意顺序。

请注意，MONOTONIC 的定义正是 GROUP BY 要继续运作所需的。

如果声明为单调的表达式不是单调的（也就是说，如果断言对实际数据无效），则表示未指定 Amazon Kinesis Data Analytics 行为。

换句话说，如果您确定表达式是单调的，则可以使用此 MONOTONIC 函数让 Amazon Kinesis Data Analytics 能够将该表达式视为单调的。

但是，如果您是错的，并且通过计算此表达式得出的值从升序更改为降序或从降序更改为升序，则可能会引发意外结果。Amazon Kinesis Data Analytics 流式 SQL 将相信您的话，并根据您确定表达式是单

调表达式来运行。但是，如果实际上不是单调的，那么生成的 Amazon Kinesis Data Analytics 行为就无法事先确定，因此结果可能不符合预期或期望。

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 从窗口中的第一个或最后一个值中返回 x 的第 n 个值。默认为第一个。如果 <null treatment> 设置为 IGNORE NULLS，则函数将在计数时跳过 null。

如果窗口中没有足够的行来达到第 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 流式 SQL 的字符串和搜索函数。

主题

- CHAR_LENGTH/CHARACTER_LENGTH
- INITCAP
- LOWER
- OVERLAY
- POSITION
- REGEX_REPLACE
- SUBSTRING
- TRIM
- UPPER

CHAR_LENGTH/CHARACTER_LENGTH

```
CHAR_LENGTH | CHARACTER_LENGTH ( <character-expression> )
```

返回作为输入参数传递的字符串的字符长度。如果输入参数为 null，则返回 null。

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 流式 SQL 不支持可选的 USING CHARACTERS | OCTETS 子句。这与 SQL:2008 标准背道而驰。

INITCAP

```
INITCAP ( <character-expression> )
```

返回转换后的输入字符串版本，以便每个以空格分隔的单词的第一个字符都是大写的，所有其他字符都是小写的。

示例

函数	结果
INITCAP ('每个 first l EtTe R 均为大写字母')	每个首字母均为大写

> **Note**
>
> INITCAP 功能不是标准的一部分。 SQL:2008 它是 Amazon Kinesis Data Analytics 扩展。

LOWER

```
LOWER ( <character-expression> )
```

将字符串转换为全小写字符。如果输入参数为 null，则返回 null；如果输入参数为空字符串，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的一部分（原始字符串）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应覆盖替换字符串的字符位置。可选的字符串长度参数确定要替换原始字符串的字符数（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，则只需附加剩余的字符即可。

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）字符将添加到结果之前，其余字符叠加在原始字符串上（请参阅下面的示例）。

如果字符串长度小于零，则会引发异常。

如果任何输入参数为 null，则结果为 null。

示例

函数	结果
OVERLAY ('12345' PLACING 'foo' FROM 1)	foo45
OVERLAY ('12345' PLACING 'foo' FROM 0)	foo345
OVERLAY ('12345' PLACING 'foo' FROM -2)	foo12345
OVERLAY ('12345' PLACING 'foo' FROM 4)	123foo
OVERLAY ('12345' PLACING 'foo' FROM 17)	12345foo
OVERLAY ('12345' PLACING 'foo' FROM 2 FOR 0)	1foo2345
OVERLAY ('12345' PLACING 'foo' FROM 2 FOR 2)	1foo45
OVERLAY ('12345' PLACING 'foo' FROM 2 FOR 9)	1foo

限制

亚马逊 Kinesis 数据分析不支持中定义的可选 USING CHARACTERS | OCTETS 子句 SQL:2008；仅假定使用字符。Strict 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。这些都偏离了标准。

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）。如果搜索字符串为空字符串，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 null，则 POSITION 返回 null。

示例

函数	结果
POSITION ('findme' IN '1234findmeXXX')	5
POSITION ('findme' IN '1234not-hereXXX')	0
POSITION ('1' IN '1234567')	1
POSITION ('7' IN '1234567')	7
POSITION ('' IN '1234567')	1

限制

亚马逊 Kinesis Data Analytics 串流 SQL 不支持中定义的可选 USING CHARACTERS | OCTETS 子句 SQL:2008；仅假设使用字符。这偏离了该标准。

REGEX_REPLACE

REGEX_REPLACE 将子字符串替换为备用子字符串。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

用于替换 original 字符串中的 regex 匹配项的字符串。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，您需要一个具有样本股票代码输入流的 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：将源字符串中的所有字符串值替换为新值

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	CHANGE	PRICE
2017-05-16 22:30:39.464	MMB	ENERGY	0.72	32.16
2017-05-16 22:30:39.464	TGT	RETAIL	2.26	268.77
2017-05-16 22:30:39.464	CVB	INFORMATION TECHNOLOGY	0.06	44.73
2017-05-16 22:30:39.464	PJN	RETAIL	-0.09	7.94

注意

REGEX_REPLACE 不是标准的一部分。 SQL:2008 它是 Amazon Kinesis Data Analytics 流式 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 将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。如果字符串中的剩余字符数少于该数量，则仅返回剩余的字符。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，则会按照起始位置为 1 来解释它，并且字符串长度将减去 (1 - 起始位置)。例如，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

这些部分由一个双引号 (") 和一个指定的转义字符分隔。有关更多信息，请参阅以下Similar...Escape示例。

示例

FROM/ FOR

函数	结果
SUBSTRING('123456789' FROM 3 FOR 4)	3456
SUBSTRING('123456789' FROM 17 FOR 4)	<empty string>
SUBSTRING('123456789' FROM -1 FOR 4)	12
SUBSTRING('123456789' FROM 6 FOR 0)	<empty string>
SUBSTRING('123456789' FROM 8 FOR 4)	89

FROM Regex

函数	结果
SUBSTRING（'TECHNOLOGY '来自 'L [A-Z]*' ）	LOGY
SUBSTRING('TECHNOLOGY' FROM 'FOO')	null
子字符串（'O [A-Z] '中的'技术' ）	OL

数值

函数	结果
SUBSTRING('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

- 亚马逊 Kinesis Data Analytics 串流 SQL 不支持中定义的可选 “使用字符 | 八位字节” 子句。SQL:2008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）不是标准的一部分。 SQL:2008它们是针对 Amazon Kinesis Data Analytics 的流式 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>
```

TRIM 根据修剪规范（即前导、 and/or 尾随或两者）的规定，从修剪源字符串的开头结尾删除指定修剪字符的实例。如果指定了 LEADING，则仅删除源字符串开头处的重复剪裁字符。如果指定了 TRAILING，则仅删除源字符串结尾处的重复剪裁字符。如果指定了 BOTH，或者完全省略了剪裁说明符，则会从源字符串开头和结尾删除重复项。

如果未明确指定剪裁字符，则默认为空格字符 (' ')。只允许使用一个剪裁字符；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 'xxxTrimx Backxxx')`	`'xxxTrimxBack'`
`TRIM (BOTH 'y' FROM 'xxxNo y to trimxxx')`	`'xxxNo y to trimxxx'`

UPPER

```
< UPPER ( <character-expression> )
```

将字符串转换为全大写字符。如果输入参数为 null，则返回 null；如果输入参数为空字符串，则返回空字符串。

示例

函数	结果
UPPER('abcDEFghi123')	ABCDEFGHI123

Kinesis Data Analytics 开发人员指南

有关开发 Kinesis Data Analytics 应用程序的信息，请参阅 Kinesis Data Analytics 开发人员指南。

文档历史记录

下表列出了自《Amazon Kinesis Data Analytics SQL 参考》上一次发布以来对文档所做的重要更改。

- API 版本：2015-08-14
- 上次文档更新日期：2018 年 3 月 19 日

更改	描述	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。	2018 年 3 月 19 日
Stream-to-stream 加入示例	JOIN 子句查询的示例。有关更多信息，请参阅 JOIN 子句。	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。	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。	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。	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、LAG、TO_TIMESTAMP、UNIX_TIMESTAMP、REGEX_REP	2017 年 8 月 3 日

更改	描述	日期
	LACE，以及添加对 SUBSTRING 的正则表达式支持	
新指南	这是 Amazon Kinesis Data Analytics SQL 参考指南的第一个版本。	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。