ALIBABA CLOUD

Alibaba Cloud

实时计算（流计算）
Blink独享/共享集群（原产品线）

文档版本：20220125

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.产品公告

本文为您介绍实时计算Flink版产品公告内容，包括版本更新、功能更新、产品活动等。

2021年4月28日-独享模式暂停新购

实时计算Flink版独享模式已于2021年4月28日暂停新购，目前仅支持原有项目的扩缩容和续费操作。如果您有新购需求，推荐使用实时计算Flink全托管。如果您有其他问题，请提交工单。

RocketMQ接入点变更导致实时计算作业适配升级公告

- 升级公告

  因消息队列RocketMQ接入点地域化变更（详情请参见关于TCP内网接入点设置的公告），如果您已使用了Blink 3.7.10以下版本的RocketMQ Connector，则您需要将您的实时计算作业升级至Blink 3.7.10及以上版本，并将作业中EndPoint参数取值更改为新的RocketMQ接入点，EndPoint参数详情请参见：

  - Blink RocketMQ源表文档：创建消息队列MQ源表。
  - Blink RocketMQ结果表文档：创建消息队列MQ结果表。

- 注意事项
  - 旧的RocketMQ接入点在2021年11月后完全不可用，且使用旧的RocketMQ接入点的作业可能存在稳定性风险，因此请您尽快安排作业升级，务必在2021年11月之前完成升级工作。
  - RocketMQ产品承诺：2021年11月前不会下线旧的RocketMQ接入点，但旧的RocketMQ接入点无法保证作业稳定性。
  - 本次升级会导致作业的State无法兼容，请结合业务情况合理安排升级时间，尽量减少对业务的影响。
  - 2021年11月1日以后，实时计算Flink版产品侧不再对使用了旧的RocketMQ接入点的作业进行维护和支持。

2020年8月10日21:00-2020年8月11日02:00-升级公告

2020年8月10日21:00-2020年8月11日02:00，对杭州独享模式管控平台进行升级，升级期间现有运行作业不受影响，但集群创建和扩缩容功能不可用，请知悉。

2020年4月28日21:00-2020年4月29日02:00-升级公告

2020年4月28日21:00-2020年4月29日02:00，对上海独享模式管控平台进行升级，升级期间现有运行作业不受影响，但集群扩缩容功能不可用，请知悉。

2020年4月20日-2020年4月22日-升级公告

2020年4月20日-2020年4月22日，对上海和深圳共享集群的存储服务进行版本升级，旨在为您提供更加稳定的实时计算Flink版服务。正常情况下，此次升级不会对用户的业务造成影响；特殊情况下，Blink3.2和Blink3.3版本的作业会Failover一次后恢复正常。

2019年8月27日-新增资源配置页签

开发页面右侧，新增**资源配置**页签。原控制台**基本属性**页签下的资源配置跳转链接已下线。

2019年5月30日-实时计算Flink版3.0.0以上版本新功能

- 运行信息

  新增Vertex相关信息查询功能，详情请参见运行信息。

- 数据曲线

  新增AutoScaling相关曲线，详情请参见数据曲线。

- Timeline

  新增Timeline功能，详情请参见Timeline。

- 属性参数

  新增AutoScale迭代的历史详情查询功能，详情请参见属性参数。

2019年1月24日-Blink-2.2.7版本发布

`Blink-2.2.7` 是 `Blink-2.x` 系列中最新稳定版本，在Blink-1.x版本（最新稳定版本为 `Blink-1.6.4` ）进行了全面的升级，采用了自主研发的新一代存储Niagara作为Statebackend的底层存储，优化了SQL的性能，增加了一系列新功能：

- 主要特性
  - SQL
    - 新增Window Emit机制，可以控制Window结果的输出策略，例如：1小时窗口，每1分钟输出一次。
    - 双流Join支持miniBatch，针对不同场景优化了Retraction处理和State存储结构，提高了性能。
    - AGG支持Filter语法，可以只聚合满足条件的行。
    - 对Local-global AGG进行优化。
    - 重构了SQL的Optimize阶段，解决了SQL编译时间过长的问题。
    - SortedMapView中KEY支持多种数据类型：BOOLEAN、BYTE、SHORT、INT、LONG、FLOAT、DOUBLE、BIGDECIMAL、BIGINT、BYTE[]和STRING。
    - 优化了MIN、MAX、FIRST和LAST函数存在Retraction场景的性能。
    - 新增多种标量函数，例如时区相关的解析TO_TIMESTAMP_TZ、格式化DATE_FORMAT_TZ和转换函数CONVERT_TZ。
    - 对SQL和Connector模块的错误信息进行了归类，并对每种类型设计了相应的ERROR_CODE。
  - Connector
    - 支持用户自定义的TableFactory注册源表和结果表的Connector。
    - 支持用户通过UDTF方式直接解析数据源类型。
    - 支持读取和写入Kafka。
    - 支持写入到ElasticSearch。
  - Runtime
    - 通过Blink Session机制，统一了用户提交Job、获取执行结果等行为。
    - 开放了调度插件机制，允许计算模型根据需求自定义调度逻辑。
    - 在有限流的情况下，通过避免不必要的全图重启，提高了JobManager和Task FailOver的处理效率。
  - StateBackend
    - 使用NiagaraStateBackend替换RocksDBStateBackend，具备更好的读写性能。
    - `(Experimental) NiagaraStateBackend` 支持计算存储分离，支持Failover过程中State秒级恢复。
- 与 `Blink1.6.4` 不兼容的语法

功能项	影响	解决办法
TableFunction接口修改	所有使用自定义TableFunction的用户。	更新代码，实现新的getResultType接口。
ScalarFunction接口修改	所有使用自定义ScalarFunction的用户。	实现新的getResultType接口。
AggregateFunction接口修改	所有使用自定义AggregateFunction的用户。	实现新的getAccumulatorType和getResultType接口。例如，accumulator类型为 `Row(STRING, LONG)` ，Aggresult的类型为STRING，则需要实现如下代码。 `public DataType getAccumulatorType\(\) { return DataTypes.createRowType\(DataTypes.String, DataTypes.LONG\); } public DataType getResultType\(\) { return DataTypes.String; }`
MapView构造函数修改	MapView构造函数形参类型由之前的TypeInformation变更为DataType。所以在自定义UDAF中声明了MapView的Job都会受影响。	更新代码，按DataType去构造MapView。例如 `MapView map = new MapView<>(Types.STRING, Types.INT);` 需要更新为 `MapView map = new MapView<>(DataTypes.STRING, DataTypes.INT);` 。
当参数是LONG或INT时，除法和AVG返回类型改为DOUBLE	以前的除法和AVG函数返回的是入参的字段类型，现在是DOUBLE，会导致类型不匹配的错误。例如：除法和AVG的结果直接写入结果表，可能会报结果表与Query字段类型不匹配的错误。	在除法和AVG的结果上强制加上CAST。
在比较BigDecimal 和Decimal数据类型的数据时，会考虑精度	用到Decimal的Job，可能会报BigDecimal类型不匹配的错误。	用到Decimal类型的Job，全局替换成带精度的声明方式， `Decimal(38, 18)` 。
NULL与字符串的比较语义	`1.x` 版本NULL与字符串比较返回True，在 `2.x` 版本后遵循了SQL语法语义改为返回False。	所有NULL与字符串比较的地方，例如： `WHERE SPLIT_INDEX(shop_list, ':', 1) <> shop_id` ，如果 `SPLIT_INDEX(shop_list, ':', 1)` 返回了NULL，在 `1.x` 版本上WHERE条件会返回True，在 `2.x` 上会返回False，将数据过滤。

- 如何升级到 `Blink-2.x`

  使用 `Blink-1.x` 版本的Job升级到 `Blink-2.x` ，需要进行数据回溯升级，数据回溯是指用户根据业务需要，在启动Job的时候，指定启动位点，具体操作如下：

  i. 停止待升级Job（清除State）。

ii. 开发界面单击右下角的**Flink版本**下拉箭头，修改Job的Blink版本为 `Blink-2.2.7` ，上线Job。

iii. 启动修改后的Job并指定启动位点。

如果步骤3执行不成功，需要人工介入查明原因后，进行如下操作：

- 快速修复SQL，重复步骤1、2、3。
- 如果无法修复SQL，回退到原有Blink版本。
- 如果无法生成Json Plan，可以尝试设置如下参数：
  - `blink.job.option.jmMemMB=4096`
  - `blink.job.submit.timeoutInSeconds=600`

`Blink-2.0.1` 的UDX第三方插件安装包详情，请参见概述。类似如下的异常，是因为UDX包的版本太低或者包冲突导致的。

```
code:[30016], brief info:[get app plan failed], context info:[detail:[java.lang.NoClassDe
fFoundError: org/apache/flink/table/functions/aggfunctions/DoubleSumWithRetractAggFunctio
n
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:788)
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
at java.net.URLClassLoader.defineClass(URLClassLoader.java:467)
at java.net.URLClassLoader.access$100(URLClassLoader.java:73)
```

2.新功能发布记录

2.1. Blink-3.7.10

本文为您介绍Blink 3.7.10版本的重大功能变更和主要修复缺陷。

版本重大功能变更

消息队列RocketMQ Connector升级底层客户端依赖版本，因此所有的type=mq的DDL需要根据RocketMQ接入点的变更，调整Endpoint参数取值至RocketMQ的TCP内网接入点，详情请参见关于TCP内网接入点设置的公告。

主要缺陷修复

修复AutoScale的缺陷。

2.2. Blink-3.7.9

本文为您介绍Blink 3.7.9版本的重大功能变更和主要修复缺陷。

版本重大功能变更

增加维表Partitioned Join校验。

> **说明**
>
> - 对于普通Patitioned Join，如果上游是更新流，为了保证语义的正确性，上游Unique Key必须包含Shuffle Key。
> - 在维表开启Partitioned Join时，可能发生热点问题。因此，引入了Bucket Partitioned Join解决该问题。但是对于Bucket Partitioned Join，上游不能是更新流。源表计算每条数据应该去的并发，然后打散到下游的某个Bucket里，维表侧会在Bucket并发上都缓存该条维表数据，达到热点数据打散的效果。

主要缺陷修复

修复由于GeminiKeyedStateHandle没有注册FileSegmentStateHandle，导致Checkpoint文件被误删的缺陷。

2.3. Blink-3.7.7

本文为您介绍Blink 3.7.7版本的重大功能变更和主要修复缺陷。

主要缺陷修复

- 修复作业Checkpoint所占存储空间过大的问题。
- 修复Checkpoint文件残留清理机制存在误删文件的问题。
- 修复Blink不能使用带Schema的Hologres物理表。

2.4. Blink-3.6.8

本文为您介绍Blink 3.6.8版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 优化时间序列数据库TSDB写入性能。
- 优化分析型数据库PostgreSQL写入性能。
- 调整表格存储Tablestore连接超时时间为20s。

主要缺陷修复

- 修复AutoScale运行期间出现作业失败无法恢复的问题。
- 修复分析型数据库MySQL版2.0结果表出现NullPointerException报错的问题。
- 修复使用RetracRank算子的作业出现ArrayIndexOutOfBoundsException报错的问题。
- 修复使用RetracRank算子的作业连续收到相同的retract消息时出现数据异常的问题。
- 修复Tablestore源表Tunnel全量转增量时出现数据丢失的问题。

2.5. Blink-3.6.5

本文为您介绍Blink 3.6.5版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 优化DataHub源表CPU使用率过高的问题。
- 优化维表的partitionJOIN功能，支持缓存策略为All的维表加载分区。此外，引入多维表异步并行优化，所有支持异步并行优化的维表可以在WITH参数中，设置partitionedJoin = 'true'，开启partitionJOIN功能。
- 日志服务SLS源表新增startupMode参数。参数取值如下：
  - TIMESTAMP：每个Shard从指定时间开始消费。
  - Earliest：每个Shard从最早位置开始消费。
  - Latest：每个Shard从最新位置开始消费。
  - Group_Offsets：每个Shard优先从服务端保存的Checkpoint开始消费，必须指定ConsumerGroup。

    > **说明** 仅当State中不存在Checkpoint时，Earliest、Latest、Group_Offsets和Timestamp的配置才生效。

- 降低大规模作业拓扑的内存消耗。
- Oracle源表的timeFieldType参数支持多种时间格式：
  - TO_DATE：DATE类型。
  - TIMESTAMP：TIMESTAMP类型。
  - VARCHAR：日期字符串类型。
  - NUMBER：数字类型。
  - 实现OracleSourceQueryCondition接口并配置类名。

主要缺陷修复

- 修复在一个Task Manager存在多个线程的情况下，开启partitionJOIN功能后，MaxCompute维表存在错误数据的缺陷。
- 修复Sink DDL field type与实际插入数据类型不一致时出现报错的缺陷。

- 修复SLS源表不消费数据的缺陷。

2.6. Blink-3.6.2

本文为您介绍Blink 3.6.2版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 优化AutoScale功能：没有流量的作业会触发Scale Down，减少资源浪费。
- 优化资源Rescale。
- 优化SLS Connector解析数据的方法：使用FastLogGroup方法解析数据。
- 降低大规模作业拓扑的内存消耗。

主要缺陷修复

- 修复Blink无法消费表格存储Tablestore源表积压数据的缺陷。
- 修复维表Cache ALL多并发导致NullPointException的缺陷。
- 修复MaxCompute Sink动态分区无法正常提交数据的缺陷。

2.7. Blink-3.6.0

本文为您介绍Blink 3.6.0版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 新增Connector：
  - 新增创建分析型数据库PostgreSQL版结果表。
  - 新增创建Phoenix5维表。
  - 新增创建交互式分析Hologres源表。
  - 新增创建交互式分析Hologres维表。
  - 新增创建交互式分析Hologres结果表。
- 新增功能：
  - MQ结果表CSV格式支持写入KEY，二进制格式不支持写入KEY。
  - 支持Gemini 2.0（Gemini 增强版）。

主要缺陷修复

- 修复Kafka 08版本结果表，语法检查报错：`Kafka dont have sufficient arguments`。
- 修复OTS源表因为某个分区一直没有数据，线上运行源表节点导致的Failover：`java.lang.NullPointerException`。
- 修复OTS维表以注册存储方式引用，Token过期导致的Failover：`OTSNoPermissionAccess, [Message]:You have no permission to access the requested resource, please contact the resource owner`。

2.8. Blink-3.5.0-hotfix

本文为您介绍Blink 3.5.0版本的重大功能变更和主要修复缺陷。

版本重大功能变更

- 新增创建InfluxDB结果表。
- 新增创建Phoenix5结果表。
- 升级DataHub Connector SDK版本。

主要缺陷修复

- 修复使用全局ORDER BY但语法检查没有报错的缺陷。
- 修复底层VARCHAR类型隐式转换为整型，行为错误的缺陷。
- 修复因calcite bug，数据按照时间分组进行聚合，导致计算结果异常的缺陷。
- 修复语法检查DataHub源表产生 `java.lang.NoClassDefFoundError: com/aliyun/datahub/client/auth/Account` 报错的缺陷。

2.9. Blink-3.4.4

本文为您介绍Blink 3.4.4版本的重大功能变更和主要修复缺陷。

版本重大功能变更

Blink-3.4.4版本支持MaxCompute源表监听新分区。在DDL语句中设置subscribeNewPartition参数，参数默认是false。当参数值是true时，不指定partition参数，会动态监听新分区。详情参见创建全量MaxCompute源表。

主要缺陷修复

RDS连接性问题：Blink使用Druid链接池链接RDS，优化Druid链接池链接RDS的方式，避免链接长时间不用导致作业的Failover。

2.10. Blink-3.4.3

本文为您介绍Blink 3.4.3版本的重大功能变更和主要修复的缺陷。

版本重大功能简介

GeminiStateBackend是Blink开发的新一代后台，它使用自研的存储引擎GeminiDB。通过线上部分作业的测试显示，GeminiStateBackend的性能可以达到NiagaraStateBackend性能的1.5倍。GeminiStateBackend的主要特性如下：

- 采用LSM + Hash索引的存储模型。其中LSM提高写性能，Hash索引存放在内存中以弥补LSM读放大的缺点。简单来说，GeminiDB将文件划分为更小粒度的单元（Page），以Page为粒度进行冲洗和压缩处理，Hash索引根据键快速定位其所在的单元，从而减少I/O次数，提高读性能。
- 优化Cache策略。如果新插入的数据或压缩后的数据存在热点，GeminiDB会将其缓存在内存中。而传统的LSM实现会首先将这些数据刷到磁盘上，导致新增数据需要至少经过一次读I/O才能进入缓存，因此缓存命中率下降。
- 优化数据冲洗到磁盘的策略。GeminiDB只有在内存的缓存填满后才会将部分数据冲洗到磁盘，因此，在内存充足并且压缩及时的情况下，不会产生数据文件。而传统LSM的实现中，会将数据冲洗到磁盘以便进行持久化处理。这在Blink的应用场景中是没有必要的，Blink具有原生的Checkpoint机制以保证数据的一致性，因此只需在Checkpoint时进行持久化处理。
- 支持In-memory Compaction。对于内存中数据及时进行压缩，减少写放大以及压缩的读I/O。
- 使用Java消除RocksDB/Niagara中的JNI开销。

- 支持增量Checkpoint。
- 支持Local Recovery，使作业失效后快速恢复。
- 支持存储计算分离，使作业重启或者Rescale后的快速恢复，功能还在完善中。

GeminiStateBackend针对DataStream和SQL的作业配置如下：

- DataStream作业
  - API配置方式

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
GeminiStateBackend stateBackend = new GeminiStateBackend(checkpointDir);
// Configuration for gemini
Configuration config = new Configuration();
config.setString("state.backend.gemini.heap.size", "1024mb");
// set configuration to backend
stateBackend.setConfiguration(conf);
// use GeminiStateBackend as state backend
env.setStateBackend(new GeminiStateBackend(checkpointDir));
```

  - 主要配置项

配置项	类型	单位	默认值	说明
state.backend.gemini.ttl.ms	LONG	ms	-1（未开启）	可选。数据存活时间。
state.backend.gemini.heap.size	STRING	支持以下字节单位： ■ 1024 ■ 1024kb ■ 1024mb ■ 1024gb	无	可选。单个GeminiDB能够使用的内存大小。 **说明** 推荐进行配置，否则backend会根据JVM和TaskManager的配置计算默认值。

- SQL作业

```
# 使用Gemini。
state.backend.type=gemini
# State的TTL的大小。
state.backend.gemini.ttl.ms=129600000
# GeminiDB允许使用的内存大小，单位MB。注意：Operator配置的内存资源要包含这部分，默认为512MB。
state.backend.gemini.heap.size.mb=512
# 推荐的TaskManager的JVM参数。
blink.job.option=-yD env.java.opts.taskmanager='-XX:NewRatio=3 -XX:SurvivorRatio=3 -XX:ParallelGCThreads=8 -XX:+UnlockDiagnosticVMOptions -XX:ParGCCardsPerStrideChunk=4096 -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=75 -Djdk.nio.maxCachedBufferSize=10240'
```

主要缺陷修复

- 修复Calc的codegen代码可能会出现NPE的缺陷。

- 修复状态存储必须存储完整行的缺陷。
- 修复 `code split` 缺陷：JavaCodeSplitter在把 `local variable` 抽取成 `member field` 时，没有替换 `foreach control` 语句中的 `local variable` 。在 `distinct with filter` 场景中可能出现这种缺陷。

2.11. Blink-3.3.0

本文为您介绍实时计算公告内容，包括版本重大功能发布和产品变更介绍。

版本重大功能

- 小CU模式
  - 新增初始plan设置CU预期：生成plan时，作业的初始并发度会根据用户设置的CU预期进行缩放。
  - 新增小CU模式资源配置：当用户设置的CU预期很小时，如果每个Vertex并发都设置为最小值1时，总的CU数还是超过了用户设置的CU，则会触发小CU模式，将多个vertex的subtask放到一个slot（即线程）中运行。同时将所有Vertex调度到一个TaskManager上，减少框架资源开销。
- DataHub
  - 写入DataHub支持通过hashFields的配置，使相同列值的记录写入同一个下游Shard。
  - 修复Blink 3.2版本读取closed shard的DataHub数据Failover，且任务不能恢复的问题。

产品变更

- Auto Scale变更
  - Autoscale最大资源设置语义修改：Max CU限制从作业整体资源的上限调整为Plan的资源上线，以解决Max CU限制，导致作业无法启动的问题。Max CU限制调整后，作业实际使用的资源可能会超过Max CU。
  - Autoscale添加可选作业参数配置，可以手动关闭scale down功能，保证作业稳定性。配置项：`healthmanager.resource.scale.down.enabled`（资源scale down开关）和 `healthmanager.parallelism.scale.down.enabled`（并发度scale down开关）。
  - 新增支持Datastream作业手动设置资源，开启Autoscale。从Blink3.3.0版本开始，Datastream作业支持手动编辑资源Plan，同时允许开启Autoscale。Datastream的Autoscale目前仅处于试用阶段。
- FirstRow on Rowtime

  支持您按照Rowtime字段读取FirstRow，去重后保持time字段的Rowtime属性，即您可以在去重节点后继续使用Window。
- SQL文件大小写变更

  该改动将忽略SQL文件中的大小写（大小写不敏感），导致部分作业在编译时报错。该错误表明您在同一段代码中，使用大小写来区分变量或标识符。
  - 示例

```
    t.et_taobao_bind AS et_taobao_bind
FROM
    view_t_partner_map_taobao_bind t,
    lateral table (STRING_SPLIT(t.utdids, '\004')) AS T(utdid0);

INSERT INTO
```

- 报错信息

```
org.apache.flink.table.api.ValidationException:
************
ERR_ID:
     SQL-00120001
CAUSE:
     SQL validation failed:
     From line 3, column 9 to line 3, column 69: Duplicate relation name 'T' in FROM clause
ACTION:
     Please see descriptions above. If it doesn't help, please contact customer support for this.
DETAIL:
```

2.12. Blink-3.2.3

为了给您带来更好的开发交互体验， 实时计算发布了Blink-3.2.3版本，本文为您介绍版本优化点和修复的问题。

优化

- 优化Vertex Topology页面与作业资源配置界面展示不一致问题。
- 优化3.2.1版本中数据曲线页面不显示Task ID的问题。
- 优化运维界面TaskManager.log日志中乱码问题。
- 优化3.2.1版本输出的GC日志会覆盖正常的调试结果的输出的问题。
- 优化独享集群AutoScale需要增加作业参数才能开启问题。
- 优化作业运行代码为LEFT JOIN，运维界面Vertex拓扑图中显示为INNER JOIN问题。
- 优化代码编辑有误时不定位至具体的一行问题。

问题修复

- REGEXP_EXTRACT函数参数为null或正则表达式不合法时，返回null。
- 开发代码中使用反斜线的正则表达式后，使用用分号编译报错。
- 上线调试运行结果打印至taskmanager.out中，与实际不符。
- LEFT JOIN维表，JOIN时无维表标识不报错。
- Time类型的数据写入ADS结果表出现异常。
- 3.2.1版本MaxCompute维表JOIN时，ON条件的字段数据类型为TIMESTAMP，作业运行报错。
- 3.2.1和3.2.2版本使用 `minibatch` 作业参数，作业运行报错。
- CEP语法检测正常情况下，作业运行时仍然报错。
- 云数据库HBase结果表的 `maxRetryTimes` 参数未生效。
- Kafka源表读取Kafka端的数据时，未显示对应的Group ID。
- 只写入一个VARBINARY类型的数据至MQ，运行报错。

2.13. Blink-3.2.1

2.13.1. Blink-3.2.1新功能发布记录

本文为您介绍Blink-3.2.1版本的核心功能以及新版本对于Datastream和SQL的兼容性。

版本核心功能介绍

Blink-3.2.1是Blink正式开源后第1个基于开源代码的正式版本。Blink-3.2.1版本核心功能介绍如下：

- Job AutoScale

  Blink-3.2.1版本中加入了AutoConfig和AutoScale相结合的自动调优功能。自动调优功能能够在作业运行过程中，根据作业运行状态以及输入流量，自动调整各个算子的并发和资源， 从而保证作业的低延时。在Blink-3.2.1版本中，该功能处于公测阶段。

- 支持Datastream API
  - Blink-3.2.1版本中正式加入了对Datastream API的支持。Blink-3.2.1基于开源Flink 1.5.1分支开发，Blink-3.2.1中Datastream API接口与开源Flink1.5.1版本兼容性对比请参见Blink 3.2与Flink 1.5.1版本API兼容性报告。
  - Datastream Connector新增。

对接系统	Source	Sink
Kafka	兼容	兼容
HBase	不兼容	
JDBC		
RDS<MySQL>		
ES		
MongoDB		

- SQL Connector新增

  Blink-3.2.1在Blink2.0版本基础上新增了如下Connector。

对接系统	Connector类型
ES	DIM
MongoDB	SINK
Redis	DIM
Redis	SINK
消息队列for Apache RocketMQ（4.2.0）	SOURCE
SQL Server	SINK

兼容性

- Datastream兼容性请参见Blink 3.2与Flink 1.5.1版本API兼容性报告。
- SQL兼容性请参见Blink 3.0与Blink 2.0版本SQL不兼容项汇总。

2.13.2. Blink 3.2与Flink 1.5.1版本API兼容性报告

本文为您介绍Blink 3.2与Flink 1.5.1版本API兼容性测试的结果。

校验范围

- flink-clients
- flink-core
- flink-java
- flink-java8
- flink-optimizer
- flink-scala
- flink-scala-shell
- flink-streaming-java
- flink-streaming-scala
- flink-yarn
- flink-connectors
- flink-filesystems
- flink-formats
- flink-metrics
- flink-queryable-state
- flink-state-backends

兼容性详情

- flink-core

  总计方法个数：6126。不兼容数量：1。

序号	严重程度（高、中或低）	Old API	Change	Effect
1	中	GenericCsvInputFormat.supportsMultiPaths()	supportsMultiPaths()方法从该类中删除，返回的默认值发生变化，新的默认值不支持multiPath。	所有GenericCsvInputFormat子类涉及multiPath的功能会受到影响。

- flink-connector-elasticsearch

  总计方法个数：14；不兼容数量：1。

序号	严重程度（高、中或低）	Old API	Change	Effect
1	中	ElasticsearchSink	父类从ElasticsearchSinkBase<T>变为了ElasticsearchSinkBase<T,org.elasticsearch.client.Client>。	Flink 1.5.1不兼容ElasticsearchSinkBase<T>父类的子类。

flink-json

总计方法个数：34；不兼容数量：1。

序号	严重程度（高、中、低）	Old API	Change	Effect
1	中	JsonSchemaConverter	类名重命名成了JsonRowSchemaConverter。	子类不兼容。

flink-streaming-java

总计方法个数：3031。不兼容数量：4。

序号	严重程度（高、中、低）	Old API	Change	Effect
1	中	TwoInputStreamOperator.processElement1 or TwoInputStreamOperator.processElement2	返回类型从void变成TwoInputSelection。TwoInputStreamOperator中新增endInput1和endInput2 abstract方法。	Blink 3.2不兼容所有的TwoInputStreamOperator实现，而Flink 1.5.1兼容所有的TwoInputStreamOperator实现。
2	中	OneInputStreamOperator类	添加endInput() abstract方法。	不兼容所有的OneInputStreamOperator实现。
3	中	StreamOperator类	添加requireState abstract方法。	不兼容所有的StreamOperator实现。
4	中	OneInputStreamOperator类	添加了endInput() abstract方法。	不兼容所有的OneInputStreamOperator实现。

2.13.3. Blink 3.0与Blink 2.0版本SQL不兼容项汇总

SQL语法变更

- 语法变动

  [Over Agg] The window rank function without order by

- 行为变动
  - [Division to double] | 隐式类型转换，从2.2版本开始生效。
  - [Decimal] DDl decimal type default precision changed to (10, 0) | Decimal的默认精度变更，从2.2版本开始生效。
  - [CEP] pattern不可以贪婪匹配结束。例如，不支持pattern (a b+)。可以通过将(a b+)转换为(a b+ c)，并把c定义为not b的方式，解决不能以贪婪匹配结束的问题。
  - [CEP] within语句不支持动态窗口。

开发接口变更

- 重构和语义变更

  [StreamTableSink#emitDataStream returns values changes from void to StreamTableSink]

- Class位置挪动
  - [Parser 继承 com.alibaba.blink.streaming.connectors.common.source.SourceCollector]
  - [Class not found] com/alibaba/blink/exceptions/NotEnoughParamsException
  - [Class not found] com/alibaba/blink/exceptions/UnsupportedTableException
  - [Class not found] org/apache/flink/table/sources/BatchExecTableSource
  - [Class not found] org/apache/flink/table/functions/aggfunctions/DoubleSumWithRetractAggFunction
  - [Class not found] org/apache/flink/table/functions/Monotonicity
  - [Class not found] Lcom/alibaba/blink/cache/Cache
  - [Class not found] org/apache/flink/table/row/GenericRow
- 实现变更
  - [Method not found] com.alibaba.blink.table.api.RichTableSchema.getColumnTypes
  - [Method not found] Lorg/apache/flink/table/types/DataType.of
  - [Verification] java.lang.VerifyError: class com.koubei.blink.connector.sls.CustomTableFactory overrides final method setClassLoader
  - [Class not found] com/aliyun/odps/OdpsException

Connectors

- [ODPS] ODPSTableSink stream mode do not support overwrite
- [ODPS] Only batch mode support overwrite

3.产品概览

3.1. 概述

阿里云实时计算Flink版独享/共享集群（原产品线）支持共享模式和独享模式两种产品模式。独享模式是基于共享模式的补充，具备更加丰富的功能。本文为您介绍实时计算共享模式和独享模式的优势及区别。

共享模式（已停购）

不同用户共享计算集群的物理资源（网络、磁盘、CPU或内存等），通过账号管理、CGroup（Control Groups）等方式进行资源隔离和安全管理。基于账号安全、业务安全和数据安全方面的考虑，共享模式不提供自定义函数功能。

> **说明** 实时计算共享模式已于2019年12月24日正式下线，不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。如果您有新购需求，推荐使用实时计算独享模式或Flink半托管模式。

独享模式

- 独享模式优势

  独享模式是指在阿里云云服务器ECS（Elastic Compute Service）上单独为用户创建的独立计算集群。单个用户独享计算集群的物理资源（网络、磁盘、CPU或内存等），与其它用户的资源完全独立。与共享模式相比，独享模式具有以下优点：

  - 多种硬件均可适配

    实时计算独享集群可以充分复用阿里云在CPU、MEM配比、GPU或FPGA等硬件层面的各类优化，为您解决各类硬件适配问题。

  - 用户间的隔离

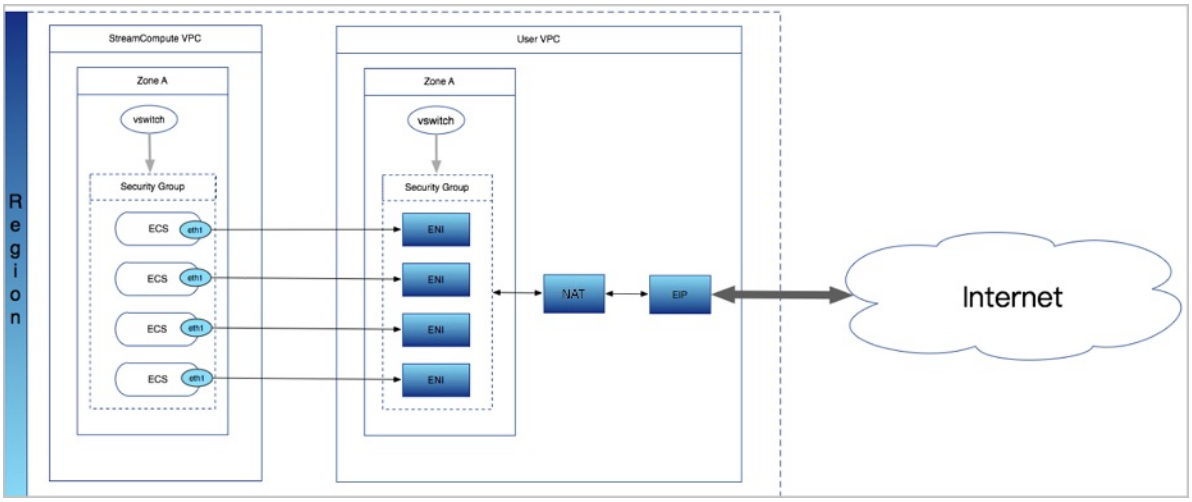
    如果使用ECS独享集群，您能够使用专有网络VPC和独享计算资源。既能满足您对专网专用、资源独享的需求，也能够与您的开发平台连通，满足您的业务需求。

  - 支持自定义函数

    独享模式在网络及物理机层面与其它用户完全的隔离，支持自定义函数和底层API，满足您的业务需求。自定义函数详情，请参见概述。

  - 丰富的功能
    - Data Lake场景下的ETL：通过Flink SQL+UDF的方式，使ETL任务开发更加便利。
    - 异构数据源计算：支持从异构数据源读取数据做分析。例如，从对象存储服务（Object Storage Service，OSS）远程读取归档日志数据，并关联云数据库HBase中的高危IP，完成网络攻击分析。
    - 支持更加丰富的上下游数据存储，例如创建消息队列Kafka源表和创建消息队列Kafka结果表。

- 独享模式系统网络架构



- 实时计算独享模式为全托管模式。您所购买的ECS托管在实时计算VPC下，暂不提供用户登录接口。如果您需要登录ECS，请提交工单。
- 您可以在创建集群时，在您账号下申请弹性网卡，通过弹性网卡，您可以访问其VPC下所有资源。
- 如果需要访问公网，可在弹性网卡上绑定NAT网关及弹性公网IP，具体操作步骤请参见绑定EIP。

  > **说明** 不访问公网时，弹性网卡不会产生任何额外费用。

- 如果您需访问VPC内其它安全组的服务，请配置相应安全组的规则。

独享模式与共享模式区别

类别	独享模式	共享模式
自定义功能开发	支持UDF和API等功能，作业开发更加灵活。	不支持
网络类型	支持阿里云VPC网络。 **说明** 独享模式集群仅能访问同一VPC、同一Region下的上下游存储资源。如果您需要访问其他VPC下的资源，请配置安全组的端口号，通过高速通道等方式打通网络。	不支持
机器类型选择	您可以在购买实时计算独享模式服务时，选择机器类型。	不支持
支持数据源类型	• 数据总线（DataHub） • 日志服务（Log Service） • 消息队列（MQ） • 消息队列（Kafka） **说明** 更多实时计算数据源表介绍，请参见数据源表概述。	• 数据总线（DataHub） • 日志服务（Log Service） • 消息队列（MQ） **说明** 更多实时计算数据源表介绍，请参见数据源表概述。

类别	独享模式	共享模式
支持数据输出类型	• 分析型数据库（AnalyticDB） • 数据总线（DataHub） • 日志服务（Log Service） • 消息队列（MQ） • 表格存储（Table Store） • 云数据库（RDS/DRDS） • 高性能时间序列数据库（TSDB） • 云数据库（HybridDB for MySQL） • Kafka • 云数据库 （HBase） • ElasticSearch（ES） **说明** 更多实时计算数据结果表介绍，请参见数据结果表概述。	• 分析型数据库（AnalyticDB） • 数据总线（DataHub） • 日志服务（Log Service） • 消息队列（MQ） • 表格存储（Table Store） • 云数据库（RDS/DRDS） • 高性能时间序列数据库（TSDB） • 云数据库（HybridDB for MySQL） **说明** 更多实时计算数据结果表介绍，请参见数据结果表概述。
支持地域	• 独享模式（按量付费）：华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）。 • 独享模式（包年包月）：华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）、华北3（张家口）、中国（香港）、新加坡。 如果您有其它地域的开通需求，请提交工单。	华南1（深圳） **说明** 实时计算共享模式已于2019年12月24日正式下线，不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。如果您有新购需求，推荐使用实时计算独享模式或Flink半托管模式。
形态	每个用户在独立的ECS集群上，搭建独立的实时计算引擎。	大集群，共同使用公共资源池。
隔离性	隔离性强。使用ECS隔离网络、安全组等资源，与其他用户的资源相互独立。	隔离性较弱。无法隔离网络等资源。
面向客户群体	具备较强开发技术，对开发的灵活性和可控性具备需求的大数据团队。	仅需实现流式业务，对开发过程无特殊需求的团队或个人。
成本	相对共享模式，独享模式成本较高。高可用模式下每个用户需单独承担额外的3台Master机器的费用。	仅需支付计算服务费用。
集群管理和运维	需要具备集群管理、网络配置等技能。	无需管理集群。

3.2. 发展历程

本文为您介绍阿里云实时计算的发展历程。

阿里云实时计算在原有Flink系统基础上，提供一整套的开发平台和完整的流式数据处理业务流程。受益于阿里巴巴大数据多年的技术和业务沉淀，您在使用实时计算时，可以享受到阿里巴巴集团前沿的计算引擎能力，规避阿里巴巴集团多年在流式大数据业务上的试错结果，更快、更轻松地享受流式计算对大数据业务发展的助力。

- 起源：脱胎于双十一实时大屏业务

  实时计算脱胎于阿里巴巴集团的双十一实时大屏业务。实时计算团队起初仅负责支持双十一大屏展现和部分实时报表业务，在历经多年的摸索和发展后，最终成长为独立稳定的云产品团队。实时计算定位将阿里巴巴集团本身沉淀多年的实时计算产品、架构、业务能够以云产品的方式对外提供服务，助力更多中小企业实时化自身大数据业务。

- 萌芽：以开源Flink作为基础

  最初，阿里巴巴集团支撑双十一大屏等业务同样采用开源的Flink作为基础系统支持，并在上面开发Flink代码。这个时期的实时业务，处于萌芽阶段，规模尚小。数据开发人员使用Flink原生API开发流式作业，开发门槛高，系统调试难，存在大量重复的人力工作。

- 发展：基于Flink的API开发

  阿里巴巴集团的工程师针对这类大量重复的工作，开始考虑进行业务封装和抽象。工程师们基于Flink的API，开发出大量可复用的数据统计组件。例如，实现了简单过滤、聚合、窗口等等作为基础的编程组件，并基于这类组件提供了一套XML语义的业务描述语言。基于这套设计，流式计算用户可以使用XML语言将不同的组件进行拼装描述，最终完成一整套完整的实时计算处理流程。基于XML+Flink组件的编程方式，从底层上避免了用户大量的重复开发工作，同时也降低了使用门槛。但我们的数据分析人员仍然需要熟悉整套编程组件和XML描述语法，这套编程方式离分析人员最熟悉的SQL方式仍然相差甚远。

- 成熟：Flink SQL开发完成

  任何技术的发展一定遵循小众创新到大众普及的成长轨迹。而从小众到大众，从创新到普及的转折点，在于技术的功能成熟和成本降低。阿里巴巴工程师开始思考如何更大程度地降低数据分析产品的门槛，从而普及到更多的用户。得益于用户群对关系型数据库几十年的沉淀，阿里巴巴工程师最终开发了Flink SQL替换了原有的XML+组件的编程方式，使用经典的SQL模式能够完成计算和数据处理。 这套系统成为今天实时计算的核心计算引擎（Flink）。当前这套系统已有单机群数千台机器的规模，日均消息处理数千亿级，流量近PB级，成为阿里巴巴集团核心的流式计算集群。

  Flink SQL的优势：

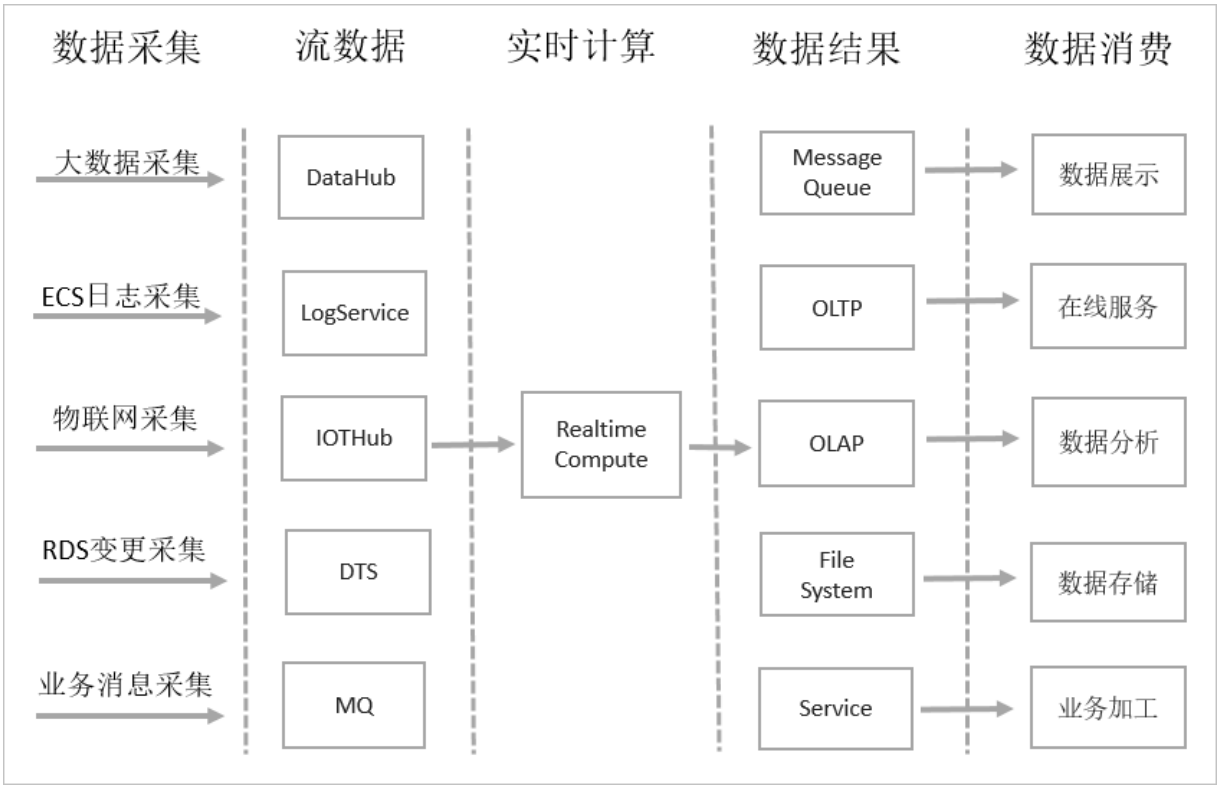
  - Flink SQL对标SQL功能，可以提高您和开发人员的技术成熟度。
  - 您可以在Flink SQL运用您熟悉的SQL模型，降低您使用实时计算的难度。

3.3. 业务流程

本文为您介绍阿里云实时计算业务流程的系统架构和数据链路。

业务流程简介

实时计算业务流程系统架构图如下。



1. 数据采集

   广义的实时数据采集，是指使用流式数据采集工具，将数据实时地采集并传输到大数据Pub/Sub（发布订阅）系统。Pub/Sub系统将为下游实时计算提供源源不断的事件源，触发流式计算作业的运行。阿里云大数据生态提供了针对不同场景领域的流式数据Pub/Sub系统。阿里云实时计算天然集成上图中诸多的Pub/Sub系统，能够集成各类流式数据。

   > **说明** 例如，您可以直接使用实时计算对接日志服务（LogService）的LogHub系统，快速的集成并使用ECS日志。

2. 流式计算

   流数据作为实时计算的触发源，驱动实时计算运行。一个实时计算作业至少使用一个流数据作为数据源。对于复杂的业务场景，实时计算支持和静态数据存储进行关联查询。

   > **说明** 例如，针对DataHub流式数据，实时计算可以根据流式数据的主键，和RDS中数据进行关联查询（即JOIN查询）。

3. 实时集成

   阿里云实时计算可以将计算的结果数据直接写入目的数据存储。阿里云实时计算天然集成了OLTP（例如RDS）、NoSQL（例如OTS）、OLAP（例如ADB）、MessageQueue（例如DataHub、ONS）、MassiveStorage（例如OSS、MaxCompute）等阿里云生态系统，最大程度地降低全链路数据的时延和数据链路的复杂度，保证数据加工的实时性。

4. 数据消费

   流式计算的结果数据进入各类数据存储后，您可以运用个性化的应用，操控结果数据。例如使用数据存储系统访问数据，使用消息投递系统接受信息，或使用告警系统生成异常结果数据警报。

数据链路

部分阿里云生态外部数据存储不能和实时计算系统完全匹配，需要使用其它类型流数据进行转换。

- LogService

  日志服务（LogService）是针对日志类数据的一站式服务。LogService提供了诸多针对日志的采集、消费、投递、查询分析等功能。请参见数据采集概述，了解如何使用日志进行流式数据采集。

- IoTHub

  阿里云物联网平台（IoTHub）是能够帮助开发者搭建安全的数据通道，方便终端（如传感器、执行器、嵌入式设备或智能家电等等）和云端的双向通信。使用IoTHub规则引擎，可以将IoT数据方便投递到DataHub，并利用实时计算和MaxCompute进行数据加工计算。请参见设置数据流转规则，了解如何将IoT数据推送到DataHub。

- MQ

  阿里云MQ服务是一套完整的消息云服务。阿里云MQ服务基于高可用分布式集群技术，搭建了包括发布订阅、消息轨迹、资源统计、定时（延时）、监控报警等功能。

3.4. 支持的上下游存储

实时计算支持丰富的上下游生态。

- 数据源表
  - 创建Oracle数据库源表
  - 创建日志服务SLS源表
  - 创建交互式分析Hologres源表
  - 创建消息队列MQ源表
  - 创建消息队列Kafka源表
  - 创建表格存储Tablestore源表
  - 创建全量MaxCompute源表
  - 创建增量MaxCompute源表
- 数据结果表
  - 创建云原生数据仓库AnalyticDB MySQL版2.0结果表
  - 创建交互式分析Hologres结果表
  - 创建Oracle数据库结果表
  - 创建日志服务SLS结果表
  - 创建消息队列MQ结果表
  - 创建表格存储Tablestore结果表
  - 创建云数据库RDS MySQL版结果表
  - 创建MaxCompute结果表
  - 创建云数据库HBase版结果表
  - 创建Elasticsearch结果表
  - 创建时序数据库结果表
  - 创建消息队列Kafka结果表
  - 创建云数据库HybridDB for MySQL结果表

- 创建云数据库RDS SQL Server版结果表
- 创建云数据库Redis版结果表
- 创建云数据库MongoDB版结果表
- 创建云原生数据仓库AnalyticDB MySQL版3.0结果表
- 创建分析型数据库PostgreSQL版结果表
- 创建自定义结果表
- 创建InfluxDB结果表
- 创建Phoenix5结果表

- 数据维表
  - 创建交互式分析Hologres维表
  - 创建表格存储Tablestore维表
  - 创建云数据库RDS MySQL版维表
  - 创建云数据库HBase版维表
  - 创建MaxCompute维表
  - 创建云数据库Redis维表
  - 创建Phoenix5维表
  - 创建云原生数据仓库AnalyticDB MySQL版3.0维表
  - 创建Elasticsearch维表

> **说明** 如果您还有其它上下游对接需求，请提交工单申请。

3.5. 产品安全

实时计算支持整体全链路实时计算的安全，包括账号安全，业务安全以及数据安全。

账号安全分为实时计算账号安全和数据存储账号安全：

- 实时计算账号安全

  实时计算账号当前仅支持阿里云账号体系（包括登录用户名+密码或签名密钥）。传输链路使用HTTPS协议，保证全链路的用户账户安全。 实时计算账号安全详情，请参见RAM用户授权。

- 数据存储账号安全

  针对将数据存储保存到连接账号的问题，实时计算提供基于RAM或STS的两种连接方式。避免您因为账户信息丢失导致的业务信息泄露。 数据存储账号安全详情，请参见独享模式角色授权。

业务安全

实时计算的业务安全部分主要包含项目隔离安全和业务流程安全：

- 项目隔离安全

  实时计算对不同的项目进行了严格的项目权限区分。不同用户或项目无法访问或操作项目内的所有子产品实体。 项目级别的资源隔离能够保证您与其他用户在使用资源时不会相互干扰。

> **说明** 例如用户A的作业在运行期间，由于数据量的突增，提升了该作业对CPU的使用率。实时计算底层的虚拟化技术对资源进行隔离，保证了该场景下仅用户A的CPU使用率提升，不会影响到其它用户作业的CPU使用状况。

- 业务流程安全

  实时计算对于流式计算开发进行了严格的流程定义，区分了数据开发和数据运维。保证了整体业务流程的完整和安全性。

  - 提供代码版本

    支持代码版本回滚和对比。方便您进行代码追溯、比对、排错。

  - 提供IDE单机调试容器

    避免代码线下运行影响线上真实数据。您可以自行构造线下输入表、维表、输出表相关数据，不影响线上生产作业。

  - 提供发布流程

    发布流程避免了线下代码的改动对生产运行的影响。线下代码调试完成后，通过上线作业将作业提交到数据运维系统。已运行的实时计算作业并不会直接使用新代码，需经过您确认后，停止已运行作业并使用新代码重新运行。从流程上保证发布的严谨性。

数据安全

数据安全分为实时计算系统数据安全和业务数据安全。

- 系统数据安全

  实时计算系统保证自身数据安全：

  - 使用HTTPS访问方式，确保传输链路的安全。
  - 数据存储使用AES高强度加密方式连接信息，避免敏感信息泄露。
  - 实时计算系统通过全面且深入的攻击测试。
  - 阿里云安全团队为实时计算提供安全服务。

- 业务数据安全

  实时计算不负责存储用户的业务数据安全。业务数据安全交由不同的阿里云存储系统管理，具体的业务数据安全机制请通过不同数据存储的安全模型以及最佳安全实践进行了解。

3.6. 使用限制

本文介绍实时计算所支持的服务范围和相应的限制，包括CU处理能力的限制、项目创建的限制。

请您仔细评估下列限制对于您业务的影响情况。如有疑问，请及时提交工单，确认产品是否满足您的业务需求。

- 实时计算共享模式暂不提供自定义函数（UDX）功能。如果需要支持UDX功能（参见概述），请使用实时计算独享模式。
- 实时计算控制台仅支持Chrome浏览器访问。

支持地域

实时计算当前支持地域如下。实际支持地域以实时计算购买页面为准：

- Flink半托管（基于 ACK）：华南1（深圳）、华北2（北京）、华东2（上海）、华北3（张家口）、华东

1（杭州）。

- Blink独享/共享集群（原产品线）：
  - 独享模式（按量付费）：华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）。
  - 独享模式（包年包月）：华东1（杭州）、华北2（北京）、华东2（上海）、华南1（深圳）、华北3（张家口）、中国（香港）、新加坡。
  - 共享模式：华南1（深圳）

> **说明** 实时计算共享模式已于2019年12月24日正式下线，不再支持共享模式新项目的购买，仅支持原有项目的扩缩容、续费操作。如果您有新购需求，推荐使用实时计算独享模式或Flink半托管模式。

CU处理能力

实时计算当前在内部压测场景下，一个CU的处理能力估算如下：

- 简单业务：例如单流过滤、字符串变换等操作，1CU每秒可以处理10000条数据。
- 复杂业务：例如JOIN、窗口、GROUP BY等操作，1CU每秒可以处理1000到5000条数据。

作业、任务数量限制

实时计算对整个项目（Project）下属的作业、Task版本、IDE打开Task页面数量均有不同限制。包括：

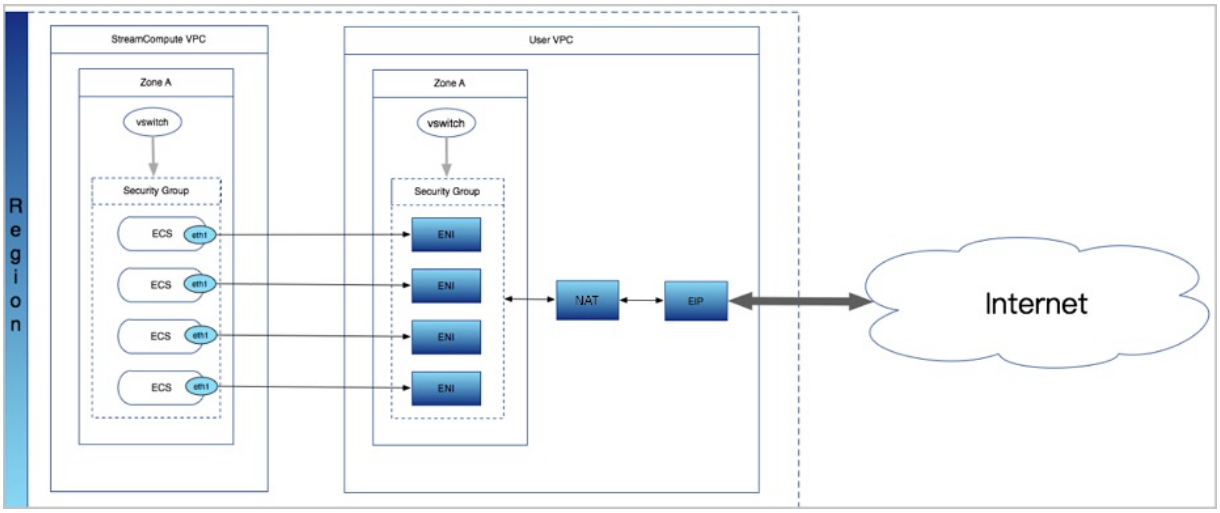
- 单个项目下允许最多创建作业的个数为100。
- 单个项目下允许最多的文件夹的个数为50，层级最大不超过5层。
- 单个项目下允许最多的UDX或JAR个数为50。
- 单个项目下允许最多注册数据存储的个数为50。
- 单个作业允许最多的历史保存版本数为20。

3.7. 独享模式系统架构（已停售）

本文为您介绍实时计算独享模式系统架构。

架构介绍

下图为独享模式系统架构。



- 实时计算独享模式为全托管管理方式。您所购买的ECS都托管在实时计算VPC下，暂不提供登录方式。

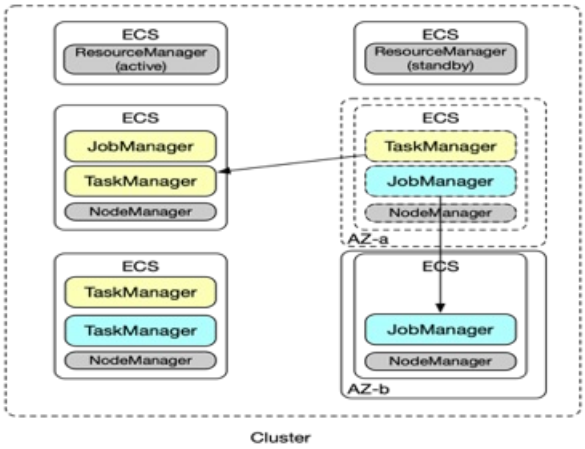
- 为了访问您的VPC内服务，会在创建集群时在您账号下申请弹性网卡。您可以通过弹性网卡访问VPC下所有资源。
- 如果您的实时计算集群需要访问公网，可以在弹性网卡上绑定NAT网关及弹性公网IP，详情请参见绑定ECS实例。
- 实时计算为您申请的弹性网卡存放于您账号下一个单独的安全组内。如果您需要访问VPC内其它安全组的服务，请配置该安全组的规则。

> **说明** 在您不访问公网的情况下，弹性网卡不会产生任何额外费用。

产品特点

- 支持全链路实时计算开发
  - 提供基于Flink SQL的实时数据处理能力，支持故障场景的自动恢复，保证异常场景仍然可以准确处理数据。
  - 支持多种内置函数，例如字符串、日期和聚合函数。
  - 支持多种窗口类型，例如滚动、滑动和会话窗口。
  - 计算资源控制精确，保证作业的隔离性。
  - 以下关键性能指标高于开源Flink:
    - 数据计算的延迟可以达到亚秒级。
    - 单个作业吞吐量可以达到百万条记录/秒，单集群规模达到数千台。
  - 深度整合各类云数据存储。方便您直接读写数据总线DataHub、日志服务SLS、云数据库RDS版、表格存储TableStore和分析型数据库MySQL版等数据存储系统。
- 完全托管的实时计算服务
  - 采用完全托管的流式计算引擎。
  - 无需预置或管理任何基础设施，即可运行和查询流数据。
  - 支持一键启用流式数据服务。
  - 试用和迁移流式计算成本小，集成存储数据、开发数据、运维数据和监控报警等功能。
  - 为不同租户间的托管运行服务提供有效的隔离和全面防护。
- 节省人力和集群成本
  - 大幅优化SQL执行引擎，计算作业比原生Flink作业更经济高效。
  - 开发和运行成本远低于开源流式框架。
- 高可用性

当实时计算产品底层ECS异常或者作业发生Failover时，JobManager及TaskManager可以在同可用区的ECS恢复使用，实现作业高可用性；也可以在不同可用区或地域的ECS恢复使用，实现跨可用区高可用性。



4.Flink SQL参考

4.1. 概述

Flink SQL是阿里云实时计算为了简化计算模型、降低用户使用实时计算门槛而设计的一套符合标准SQL语义的开发语言。

本章节通过以下方面，为您介绍实时计算中Flink SQL的使用方法。

- 基本概念
- 关键字
- 数据类型
- DDL语句
- DML语句
- QUERY语句
- 数据视图
- 窗口函数
- 逻辑函数
- 内置函数
- 自定义函数

4.2. 关键字

本文为您介绍实时计算中已保留的关键字和使用关键字字符的方法。

关键字常用类型

常用类型	关键字
数据类型	• VARCHAR • INT • BIGINT • DOUBLE • DATE • BOOLEAN • TINYINT • SMALLINT • FLOAT • DECIMAL • VARBINARY
DDL	• CREATE TABLE • CREATE FUNCTION • CREATE VIEW

常用类型	关键字
DML	INSERT INTO
SELECT 子句	• SELECT FROM • WHERE • GROUP BY • JOIN

命名规则

源表名称、结果表名称、视图表名称、别名名称遵循标准数据库命名规则定义，必须以字母开头，并且只能包含字母、数字和下划线。

保留关键字

以下字符串组合已经被保留为关键字，以备将来使用。如果您想使用以下字符串作为字段名称，请在关键字两端添加反单引号（`），例如 `` `value` `` 。

```
A,ABS,ABSOLUTE,ACTION,ADA,ADD,ADMIN,AFTER,ALL,ALLOCATE,ALLOW,ALTER,ALWAYS,AND,ANY,ARE,ARRAY
,AS,ASC,ASENSITIVE,ASSERTION,ASSIGNMENT,ASYMMETRIC,AT,ATOMIC,ATTRIBUTE,ATTRIBUTES,AUTHORIZA
TION,AVG,
BEFORE,BEGIN,BERNOULLI,BETWEEN,BIGINT,BINARY,BIT,BLOB,BOOLEAN,BOTH,BREADTH,BY,
C,CALL,CALLED,CARDINALITY,CASCADE,CASCADED,CASE,CAST,CATALOG,CATALOG_NAME,CEIL,CEILING,CENT
URY,CHAIN,CHAR,CHARACTER,CHARACTERISTICTS,CHARACTERS,CHARACTER_LENGTH,CHARACTER_SET_CATALOG
,CHARACTER_SET_NAME,CHARACTER_SET_SCHEMA,CHAR_LENGTH,CHECK,CLASS_ORIGIN,CLOB,CLOSE,COALESCE
,COBOL,COLLATE,COLLATION,COLLATION_CATALOG,COLLATION_NAME,COLLATION_SCHEMA,COLLECT,COLUMN,C
OLUMN_NAME,COMMAND_FUNCTION,COMMAND_FUNCTION_CODE,COMMIT,COMMITTED,CONDITION,CONDITION_NUMB
ER,CONNECT,CONNECTION,CONNECTION_NAME,CONSTRAINT,CONSTRAINTS,CONSTRAINT_CATALOG,CONSTRAINT_
NAME,CONSTRAINT_SCHEMA,CONSTRUCTOR,CONTAINS,CONTINUE,CONVERT,CORR,CORRESPONDING,COUNT,COVAR
_POP,COVAR_SAMP,CREATE,CROSS,CUBE,CUME_DIST,CURRENT,CURRENT_CATALOG,CURRENT_DATE,CURRENT_DE
FAULT_TRANSFORM_GROUP,CURRENT_PATH,CURRENT_ROLE,CURRENT_SCHEMA,CURRENT_TIME,CURRENT_TIMESTA
MP,CURRENT_TRANSFORM_GROUP_FOR_TYPE,CURRENT_USER,CURSOR,CURSOR_NAME,CYCLE,
DATA,DATABASE,DATE,DATETIME_INTERVAL_CODE,DATETIME_INTERVAL_PRECISION,DAY,DEALLOCATE,DEC,DE
CADE,DECIMAL,DECLARE,DEFAULT,DEFAULTS,DEFERRABLE,DEFERRED,DEFINED,DEFINER,DEGREE,DELETE,DEN
SE_RANK,DEPTH,DEREF,DERIVED,DESC,DESCRIBE,DESCRIPTION,DESCRIPTOR,DETERMINISTIC,DIAGNOSTICS,
DISALLOW,DISCONNECT,DISPATCH,DISTINCT,DOMAIN,DOUBLE,DOW,DOY,DROP,DYNAMIC,DYNAMIC_FUNCTION,D
YNAMIC_FUNCTION_CODE,
EACH,ELEMENT,ELSE,END,END-EXEC,EPOCH,EQUALS,ESCAPE,EVERY,EXCEPT,EXCEPTION,EXCLUDE,EXCLUDING
,EXEC,EXECUTE,EXISTS,EXP,EXPLAIN,EXTEND,EXTERNAL,EXTRACT,
FALSE,FETCH,FILTER,FINAL,FIRST,FIRST_VALUE,FLOAT,FLOOR,FOLLOWING,FOR,FOREIGN,FORTRAN,FOUND,
FRAC_SECOND,FREE,FROM,FULL,FUNCTION,FUSION,
G,GENERAL,GENERATED,GET,GLOBAL,GO,GOTO,GRANT,GRANTED,GROUP,GROUPING,
HAVING,HIERARCHY,HOLD,HOUR,IDENTITY,IMMEDIATE,IMPLEMENTATION,IMPORT,
IN,INCLUDING,INCREMENT,INDICATOR,INITIALLY,INNER,INOUT,INPUT,INSENSITIVE,INSERT,INSTANCE,IN
STANTIABLE,INT,INTEGER,INTERSECT,INTERSECTION,INTERVAL,INTO,INVOKER,IS,ISOLATION,
JAVA,JOIN,
K,KEY,KEY_MEMBER,KEY_TYPE,
LABEL,LANGUAGE,LARGE,LAST,LAST_VALUE,LATERAL,LEADING,LEFT,LENGTH,LEVEL,LIBRARY,LIKE,LIMIT,L
N,LOCAL,LOCALTIME,LOCALTIMESTAMP,LOCATOR,LOWER,
M,MAP,MATCH,MATCHED,MAX,MAXVALUE,MEMBER,MERGE,MESSAGE_LENGTH,MESSAGE_OCTET_LENGTH,MESSAGE_T
EXT,METHOD,MICROSECOND,MILLENNIUM,MIN,MINUTE,MINVALUE,MOD,MODIFIES,MODULE,MONTH,MORE,MULTIS
ET,MUMPS,
```

```
NAME,NAMES,NATIONAL,NATURAL,NCHAR,NCLOB,NESTING,NEW,NEXT,NO,NONE,NORMALIZE,NORMALIZED,NOT,NULL,NULLABLE,NULLIF,NULLS,NUMBER,NUMERIC,
OBJECT,OCTETS,OCTET_LENGTH,OF,OFFSET,OLD,ON,ONLY,OPEN,OPTION,OPTIONS,OR,ORDER,ORDERING,ORDINALITY,OTHERS,OUT,OUTER,OUTPUT,OVER,OVERLAPS,OVERLAY,OVERRIDING,
PAD,PARAMETER,PARAMETER_MODE,PARAMETER_NAME,PARAMETER_ORDINAL_POSITION,PARAMETER_SPECIFIC_CATALOG,PARAMETER_SPECIFIC_NAME,PARAMETER_SPECIFIC_SCHEMA,PARTIAL,PARTITION,PASCAL,PASSTHROUGH,PATH,PERCENTILE_CONT,PERCENTILE_DISC,PERCENT_RANK,PLACING,PLAN,PLI,POSITION,POWER,PRECEDING,PRECISION,PREPARE,PRESERVE,PRIMARY,PRIOR,PRIVILEGES,PROCEDURE,PUBLIC,
QUARTER,
RANGE,RANK,READ,READS,REAL,RECURSIVE,REF,REFERENCES,REFERENCING,REGR_AVGX,REGR_AVGY,REGR_COUNT,REGR_INTERCEPT,REGR_R2,REGR_SLOPE,REGR_SXX,REGR_SXY,REGR_SYY,RELATIVE,RELEASE,REPEATABLE,RESET,RESTART,RESTRICT,RESULT,RETURN,RETURNED_CARDINALITY,RETURNED_LENGTH,RETURNED_OCTET_LENGTH,RETURNED_SQLSTATE,RETURNS,REVOKE,RIGHT,ROLE,ROLLBACK,ROLLUP,ROUTINE,ROUTINE_CATALOG,
ROUTINE_NAME,ROUTINE_SCHEMA,ROW,ROWS,ROW_COUNT,ROW_NUMBER,
SAVEPOINT,SCALE,SCHEMA,SCHEMA_NAME,SCOPE,SCOPE_CATALOGS,SCOPE_NAME,SCOPE_SCHEMA,SCROLL,SEARCH,SECOND,SECTION,SECURITY,SELECT,SELF,SENSITIVE,SEQUENCE,SERIALIZABLE,SERVER,SERVER_NAME,SESSION,SESSION_USER,SET,SETS,SIMILAR,SIMPLE,SIZE,SMALLINT,SOME,SOURCE,SPACE,SPECIFIC,SPECIFICTYPE,SPECIFIC_NAME,SQL,SQLEXCEPTION,SQLSTATE,SQLWARNING,SQL_TSI_DAY,SQL_TSI_FRAC_SECOND,SQL_TSI_HOUR,SQL_TSI_MICROSECOND,SQL_TSI_MINUTE,SQL_TSI_MONTH,SQL_TSI_QUARTER,SQL_TSI_SECOND,SQL_TSI_WEEK,SQL_TSI_YEAR,SQRT,START,STATE,STATEMENT,STATIC,STDDEV_POP,STDDEV_SAMP,STREAM,
STRUCTURE,STYLE,SUBCLASS_ORIGIN,SUBMULTISET,SUBSTITUTE,SUBSTRING,SUM,SYMMETRIC,SYSTEM,SYSTEM_USER,
TABLE,TABLESAMPLE,TABLE_NAME,TEMPORARY,THEN,TIES,TIME,TIMESTAMP,TIMESTAMPADD,TIMESTAMPDIFF,
TIMEZONE_HOUR,TIMEZONE_MINUTE,TINYINT,TO,TOP_LEVEL_COUNT,TRAILING,TRANSACTION,TRANSACTIONS_ACTIVE,TRANSACTIONS_COMMITTED,TRANSACTIONS_ROLLED_BACK,TRANSFORM,TRANSFORMS,TRANSLATE,TRANSLATION,TREAT,TRIGGER,TRIGGER_CATALOG,TRIGGER_NAME,TRIGGER_SCHEMA,TRIM,TRUE,TYPE,
UESCAPE,UNBOUNDED,UNCOMMITTED,UNDER,UNION,UNIQUE,UNKNOWN,UNNAMED,UNNEST,UPDATE,UPPER,UPSERT,USAGE,USER,USER_DEFINED_TYPE_CATALOG,USER_DEFINED_TYPE_CODE,USER_DEFINED_TYPE_NAME,USER_DEFINED_TYPE_SCHEMA,USING,
VALUE,VALUES,VARBINARY,VARCHAR,VARYING,VAR_POP,VAR_SAMP,VERSION,VIEW,
WEEK,WHEN,WHENEVER,WHERE,WIDTH_BUCKET,WINDOW,WITH,WITHIN,WITHOUT,WORK,WRAPPER,WRITE,
XML,
YEAR,
ZONE
```

4.3. 基本概念

4.3.1. 时区

实时计算Flink版作业支持配置时区参数调整时间类型数据的输出结果。本文为您介绍配置时区参数的方法。

配置时区参数

- 配置相同时区参数

  实时计算Flink版平台的作业参数中，您可以通过配置作业的时区（例如 `blink.job.timeZone=America/New_York` ）调整时间类型数据的输出结果。默认时区为东八区。

  - 时区列表请参见支持的时区列表。
  - 时区参数配置方法请参见配置作业参数。

- 配置不同时区参数

您可以对不同的源表或结果表配置不同的时区。例如，您需要读写的MySQL数据类型（例如TIME、DATE、TIMESTAMP类型）的时区为 `America/New_York` ，而作业计算需要的时区为 `Asia/Shanghai` ，则可以通过如下方式单独配置源表或结果表的时区。

```
CREATE TABLE mysql_source_my_table (
 -- ...
) WITH (
timeZone='America/New_York'
 -- ...
);
```

时区参数配置示例

实时计算Flink版时区相关函数语义上为自定义的时区。下文以 `Asia/Shanghai` 自定义时区为例进行说明。

- 字符串转时间类型（TO_TIMESTAMP、TIMESTAMP和UNIX_TIMESTAMP）

```
TO_TIMESTAMP('2020-08-03 10:59:45.957')
-- 输出：`2020-08-03 10:59:45.957`。
TIMESTAMP '2020-08-03 10:59:45.957'
-- 输出：`2020-08-03 10:59:45.957`。
UNIX_TIMESTAMP('2020-08-03 10:59:45.957')
-- 输出： `1596423585`。
```

- 时间类型转字符串（DATE_FORMAT和FROM_UNIXTIME）

> **说明** 当参数为TIMESTAMP时，输出结果取决于自定义设定的时区。

```
DATE_FORMAT(TO_TIMESTAMP(1596702949000), 'yyyy-MM-dd HH:mm:ss')
-- 输出：`2020-08-06 16:35:49`。
DATE_FORMAT('2020-08-06 16:35:49', 'yyyy-MM-dd HH:mm:ss', 'yyyy/MM/dd HH:mm:ss')
-- 输出：`2020/08/06 16:35:49`
FROM_UNIXTIME(1596702949000/1000)
-- 输出： `2020-08-06 16:35:49`
```

- 时间相关计算函数

  当参数为TIMESTMP时，EXTRACT、FLOOR、CEIL和DATE_DIFF等函数输出结果取决于自定义的时区。

```
-- 1521503999000 2018-03-19T23:59:59+0000, 2018-03-20T07:59:59+0800
 EXTRACT(DAY FROM TO_TIMESTAMP(1521503999000))
-- 输出：`20`，表示东八区的20日。
```

- 当前时间函数

  当前时间函数包括 `LOCALTIMESTAMP()` 、 `CURRENT_TIMESTAMP()` 、 `NOW()` 和 `UNIX_TIMESTAMP()` 等。

```
-- 当前时间是2020-08-03 10:59:45 (Asia/Shanghai)
LOCALTIMESTAMP
-- 输出：`2020-08-03 10:59:45.957`。
CURRENT_TIMESTAMP
-- 输出：`2020-08-03 10:59:45.957`。
NOW()
-- 输出：`1596423585`。
UNIX_TIMESTAMP()
-- 输出：`1596423585`。
```

- DATE和TIME类型

  对于DATE和TIME类型，SQL内部使用整数进行显示和计算。DATE代表EPOCH DAYS，TIME代表用户时区，从当天的零点开始计算的毫秒数。如果UDF中对DATE和TIME进行计算，从内部类型转换到 `java.sql.Date` 或 `java.sql.Time` 类型时，Java对象中已经完成了时区偏移操作。

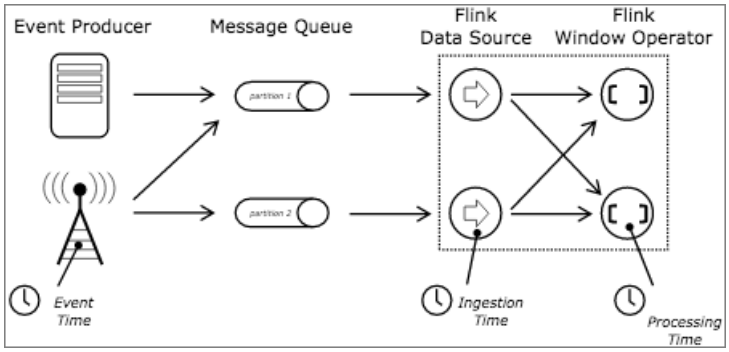
支持的时区列表

实时计算Flink版支持的时区列表请参见TimeZone。

4.3.2. 时间属性

本文为您介绍Flink SQL所支持的Event Time和Processing Time时间类型及属性。

Flink支持三种与流数据处理相关的时间概念：Processing Time、Event Time和Ingestion Time。



Blink SQL仅支持其中的两种时间类型Event Time和Processing Time：

- Event Time：您提供的事件时间（通常是数据的最原始的创建时间）。Event Time必须是您提供在数据储存里的数据。
- Processing Time：系统对事件进行处理的本地系统时间，单位为毫秒。

Event Time

Event Time也称为Row Time。EventTime时间属性必须在源表DDL中声明，可以将源表中的某一字段声明成Event Time。目前只支持将TIMESTAMP类型（将来会支持LONG类型）声明成Row Time字段。如果源表中需要声明为Event Time的列不是TIMESTAMP类型，需要借助计算列，基于现有列构造出一个TIMESTAMP类型的列。

由于数据本身的乱序、网络的抖动（网络堵塞导致的数据传输延迟的变化）或者其它原因，导致了数据到达的顺序和被处理的顺序，可能是不一致的（乱序）。因此需要首先明文定义一个Watermark计算方法，才能定义一个Row Time字段。

窗口函数基于Event Time聚合的示例如下。

```
CREATE TABLE tt_stream (
  a VARCHAR,
  b VARCHAR,
  ts TIMESTAMP,
  WATERMARK wk1 FOR ts as withOffset (ts, 1000) --Watermark计算方法。
) WITH (
  type = 'sls',
  topic = '<yourTopicName>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>'
);
CREATE TABLE rds_output (
  id VARCHAR,
  win_start TIMESTAMP,
  win_end TIMESTAMP,
  cnt BIGINT
) WITH (
  type = 'rds',
  url = 'jdbc:mysql://****3306/test',
  tableName = '<yourTableName>',
  userName = '<yourUserName>',
  password = '<yourPassword>'
);
INSERT
  INTO rds_output
SELECT
  a AS id,
  SESSION_START (ts, INTERVAL '1' SECOND) AS win_start,
  SESSION_END (ts, INTERVAL '1' SECOND) AS win_end,
  COUNT (a) AS cnt
FROM
  tt_stream
GROUP
  BY SESSION (ts, INTERVAL '1' SECOND),
  a
```

Processing Time

Processing Time是系统产生的，不在您的原始数据中，您需要在数据源表的声明中明文定义一个Processing Time列。

```
filedName as PROCTIME()
```

窗口函数基于Processing Time聚合的示例如下。

```
CREATE TABLE mq_stream (
    a VARCHAR,
    b VARCHAR,
    c BIGINT,
    ts AS PROCTIME () --在数据源表的声明中明文定义一个Processing Time列。
  ) WITH (
    type = 'mq',
    topic = '<yourTopic>',
    accessId = '<yourAccessId>',
    accessKey = '<yourAccessSecret>'
  );
CREATE TABLE rds_output (
  id VARCHAR,
  win_start TIMESTAMP,
  win_end TIMESTAMP,
  cnt BIGINT
) with (
  type = 'rds',
  url = '<yourDatebaseURL>',
  tableName = '<yourDatabasTableName>',
  userName = '<yourUserName>',
  password = '<yourPassword>'
);
INSERT
  INTO rds_output
SELECT
  a AS id,
  SESSION_START (ts, INTERVAL '1' SECOND) AS win_start,
  SESSION_END (ts, INTERVAL '1' SECOND) AS win_end,
  COUNT (a) AS cnt
FROM
  mq_stream
GROUP
  BY SESSION (ts, INTERVAL '1' SECOND),
  a
```

时间属性字段传递

时间属性字段经过如下操作后会失去时间属性特性：

- 对时间属性字段以外的字段进行GROUP BY（滚动窗口、滑动窗口或会话窗口中的GROUP BY除外）操作。
- 双流JOIN操作。

> **说明** 双流JOIN详情请参见JOIN语句。

- 复杂事件处理（CEP）语句中的MATCH_RECOGNIZE操作。
- OVER窗口中的PARTITION BY操作。
- UNION操作。UNION = RETRACT + UNION ALL。

如果经过以上操作后，继续使用该时间属性字段进行窗口函数运算，会出现类似 `org.apache.flink.table.api.ValidationException: Window can only be defined over a time attribute column.` 的报错。

4.3.3. Watermark

实时计算可以基于时间属性对数据进行窗口聚合。基于Event Time时间属性的窗口函数作业中，数据源表的声明中需要使用Watermark方法。

定义

Watermark是一种衡量Event Time进展的机制，它是数据本身的一个隐藏属性，Watermark的定义是数据源表DDL定义的一部分。Watermark语法定义如下。

```
WATERMARK [watermarkName] FOR <rowtime_field> AS withOffset(<rowtime_field>, offset)
```

> **说明** 实时计算时间属性详情，请参见时间属性。

参数	是否必填	说明
watermarkName	否	标识Watermark的名字。
<rowtime_field>	是	<rowtime_field>必须是表中已定义的一列（当前仅支持TIMESTAMP类型），基于该列生成Watermark，并且标识该列为Event Time列。您可以使用<rowtime_field>在作业代码中定义窗口。
withOffset	是	Watermark的生成策略，根据<rowtime_field> - offset生成Watermark的值。withOffset的第一个参数必须是<rowtime_field>。
offset	是	Watermark值与Event Time值的偏移量，单位为毫秒。

通常，一条记录中的某个字段就代表了该记录的发生时间。例如，表中rowtime字段的类型为TIMESTAMP，`1501750584000（2017-08-03 08:56:24.000）`。定义一个基于该rowtime列，偏移4秒的Watermark的示例如下。

```
WATERMARK FOR rowtime AS withOffset(rowtime, 4000)
```

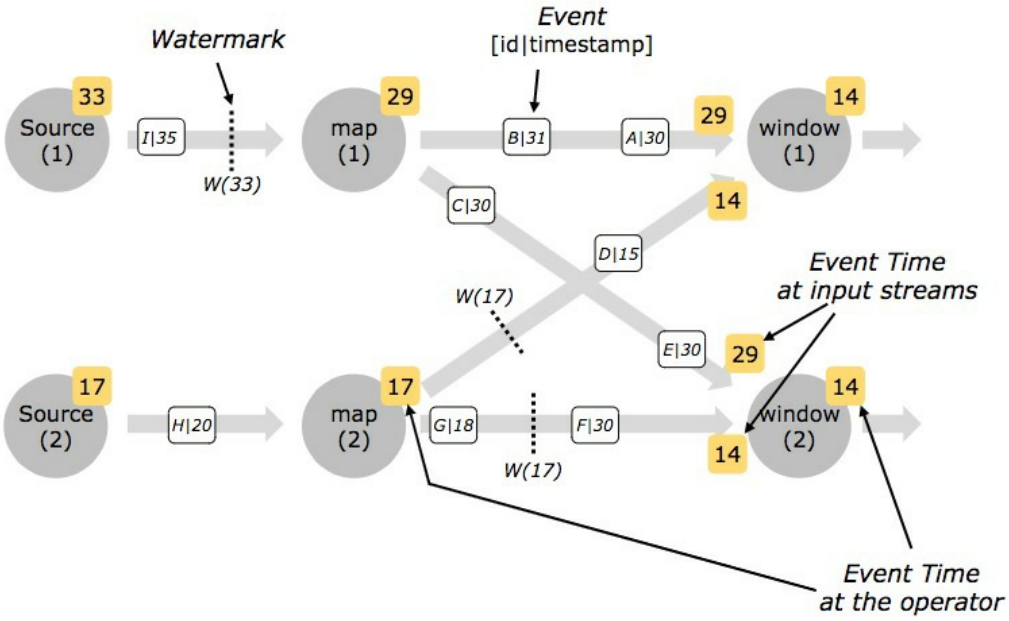
这条数据的Watermark时间为`1501750584000 - 4000 = 1501750580000（2017-08-03 08:56:20.000）`。这条数据的Watermark时间含义：时间戳小于`1501750580000（2017-08-03 08:56:20.000）`的数据已经全部到达。

> **说明**
>
> - 在使用Event Time Watermark时的rowtime必须是TIMESTAMP类型。当前支持毫秒级别的、在Unix时间戳里为13位。如果是其他类型或是在Unix时间戳不是13位，建议使用计算列完成转换。
> - Event Time和Processing Time只能在源表上声明。

使用总结

- Watermark含义是所有时间戳t'< t的事件已经全部发生。如果t（Watermark）已经生效，则后续Event Time小于t的记录将全部丢弃（后续支持用户配置，使Event Time小于t的数据可以更新）。

- 针对乱序的流，Watermark至关重要。即使部分事件延迟到达，也不会影响窗口计算的正确性。
- 并行数据流中，当算子（Operator）有多个输入流时，算子的Event Time以最小流Event Time为准。



4.3.4. 计算列

计算列可以使用其它列的数据，计算出其所属列的数值。如果您的数据源表中没有TIMESTAMP类型的列，可以使用计算列方法从其它类型的字段进行转换。

计算列概念

计算列是虚拟列，并非实际存储在表中。计算列可以通过表达式、内置函数、或是自定义函数等方式，使用其它列的数据，计算出其所属列的数值。计算列在Flink SQL中可以像普通字段一样被使用。

计算列的用途

目前Watermark的Event Time（也称为Rowtime）列只支持TIMESTAMP类型（未来将支持LONG类型）。Watermark只能定义在源表DDL中，如果您的源表中没有TIMESTAMP类型的列，可以使用计算列从其他类型的字段进行转换。

计算列语法

```
column_name AS computed_column_expression
```

计算列示例

Watermark的Rowtime必须是TIMESTAMP数据类型。当前实时计算支持毫秒级别的、在Unix时间戳里是13位TIMESTAMP数据类型。如果DataHub的TIME字段是微秒级别的（16位Unix时间戳），可以用计算列方法转换为13位的时间戳，如下所示。

```
CREATE TABLE test_stream(
  a INT,
  b BIGINT,
  `TIME` BIGINT,
  ts AS TO_TIMESTAMP(`TIME`/1000), --使用计算列方法，将16位时间戳转换为13位时间戳。
  WATERMARK FOR ts AS WITHOFFSET(ts, 1000)
) WITH (
  type = 'datahub',
  ...
);
```

源表数据中的字段 `TIME` 包含时间信息，为BIGINT类型。用计算列的功能将字段 TIME 转换成13位时间戳（TIMESTAMP）类型字段 ts ，并将 ts 字段作为Watermark的Rowtime字段。

4.4. 数据类型

4.4.1. 类型转换

实时计算数据类型转换。

本文为您介绍实时计算支持的数据类型以及数据类型之间的转换方法。

实时计算支持的数据类型

数据类型	说明	值域
VARCHAR	可变长度字符串	VARCHAR最大容量为4MB。
BOOLEAN	逻辑值	取值为TRUE、FALSE或UNKNOWN。
TINYINT	微整型，1字节整数。	-128～127
SMALLINT	短整型，2字节整数。	-32768～32767
INT	整型，4字节整数。	-2147483648～2147483647
BIGINT	长整型，8字节整数。	-9223372036854775808～9223372036854775807
FLOAT	4字节浮点型	6位数字精度
DECIMAL	小数类型	示例： 123.45 是 DECIMAL(5,2) 的值。
DOUBLE	浮点型，8字节浮点型。	15位十进制精度。
DATE	日期类型	示例： DATE'1969-07-20'
TIME	时间类型	示例： TIME '20：17：40'
TIMESTAMP	时间戳，显示日期和时间。	示例： TIMESTAMP '1969-07-20 20:17:40'

数据类型	说明	值域
VARBINARY	二进制数据	`byte[]` 数组。 说明 VARBINARY没有最大容量的限制。

数据类型转换

是否可转	数据类型										
	VARCHAR	BOOLEAN	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	DATE	TIMESTAMP	TIME
VARCHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
BOOLEAN	Y	Y	N	N	N	N	N	N	N	N	N
TINYINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
SMALLINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
INT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
BIGINT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
FLOAT	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
DOUBLE	Y	N	Y	Y	Y	Y	Y	Y	N	N	N
DATE	Y	N	N	N	N	N	N	N	Y	Y	N
TIMESTAMP	Y	N	N	N	N	N	N	N	Y	Y	Y
TIME	Y	N	N	N	N	N	N	N	N	Y	Y

类型转换示例

- 测试数据

var1（VARCHAR）	big1（BIGINT）
1000	323

- 测试语句

```
cast (var1 as bigint) as AA;
cast (big1 as varchar) as BB;
```

- 测试结果

AA（BIGINT）	BB（VARCHAR）
1000	323

4.4.2. 数学和逻辑运算

实时计算数据类型运算

本文为您介绍实时计算数据类型之间的数学运算和逻辑运算关系。

说明 同一运算语句中numeric1和numeric2的数据类型需要保持一致。

运算语句	描述	numeric1和numeric2支持的数据类型	示例
numeric1 + numeric2	返回前后两数字数学运算的和	• INT • DOUBLE • DECIMAL • BIGINT	`2+4.2`
numeric1 - numeric2	返回前后两数字数学运算差值		`3-5.3`
numeric1 * numeric2	返回前后两数字数学运算积值		`2*4`
numeric1 / numeric2	返回前后两数字数学运算商值		`2.4/5`
numeric1 > numeric2	返回前面数字是否大于后面数字的逻辑运算值		`2.4>5`
numeric1 < numeric2	返回前面数字是否小于后面数字的逻辑运算值		`2.4<5`
numeric1 >= numeric2	返回前面数字是否大于等于后面数字的逻辑运算值		`2.4>=5`
numeric1 <= numeric2	返回前面数字是否小于等于后面数字的逻辑运算值		`2.4<=5`
numeric1 = numeric2	返回前后两数字是否相同（等）的逻辑运算值	• INT • DOUBLE • DECIMAL • BIGINT • VARCHAR	`'iphone' = 5`
numeric1 <> numeric2	返回前后两数字是否不相同（等）的逻辑运算值		`'iphone' <> 5`

4.5. 创建数据视图

您可以通过创建实时计算数据视图简化开发过程。

背景信息

通常，业务逻辑比较复杂时，需要将多层嵌套写在DML语句中，但是这种方式定位问题比较困难。此时，您可以通过定义数据视图的方式，将多层嵌套写在数据视图中，简化开发过程。

> **说明** 数据视图仅用于辅助计算逻辑的描述，不会产生数据的物理存储。

语法

```
CREATE VIEW viewName[ (columnName[ , columnName]* ) ] AS queryStatement;
```

- viewName：视图名称。
- columnName：字段名称。
- queryStatement：嵌套语句别名。

示例1

```
CREATE VIEW LargeOrders (r, t, c, u) AS
SELECT
    rowtime,
    productId,
    c,
    units
FROM
    orders;
INSERT INTO
    rds_output
SELECT
    r,
    t,
    c,
    u
FROM
    LargeOrders;
```

示例2

- 测试数据

a（VARCHAR）	b（BIGINT）	c（TIMESTAMP）
test1	1	1506823820000
test2	1	1506823850000
test1	1	1506823810000
test2	1	1506823840000
test2	1	1506823870000
test1	1	1506823830000
test2	1	1506823860000

- 测试语句

```
CREATE TABLE datahub_stream (
   a VARCHAR,
   b BIGINT,
   c TIMESTAMP,
   d AS PROCTIME()
) WITH (
  TYPE='datahub',
  ...
);
CREATE TABLE rds_output (
   a VARCHAR,
   b TIMESTAMP,
   cnt BIGINT,
   PRIMARY KEY(a)
)WITH(
  TYPE = 'rds',
  ...
);
CREATE VIEW rds_view AS
SELECT a,
   CAST(
      HOP_START(d, INTERVAL '5' SECOND, INTERVAL '30' SECOND) AS TIMESTAMP
   ) AS cc,
   SUM(b) AS cnt
FROM
   datahub_stream
GROUP BY
    HOP(d, INTERVAL '5' SECOND, INTERVAL '30' SECOND),a;
INSERT INTO
   rds_output
SELECT
   a,
   cc,
   cnt
FROM
   rds_view
WHERE
   cnt=4;
```

- 测试结果

a(VARCHAR)	b (TIMESTAMP)	cnt (BIGINT)
test2	`2017-11-06 16:54:10`	4

4.6. DDL语句

4.6.1. 概述

本文为您介绍实时计算Flink版DDL语法，以及在DDL使用过程中需要注意的字段映射和大小写敏感问题。

语法

```
CREATE TABLE tableName
      (columnName dataType [, columnName dataType ]*)
      [ WITH (propertyName=propertyValue [, propertyName=propertyValue ]*) ];
```

说明

阿里云实时计算Flink版本身不具有数据存储功能，所有涉及表创建的DDL操作，实际上均是对外部数据表、存储的引用声明。

```
CREATE TABLE mq_stream(
 a VARCHAR,
 b VARCHAR,
 c VARCHAR
) WITH (
 type='mq',
 topic='blink_mq_test',
 accessID='<yourAccessID>',
 accessKey='<yourAccessSecret>'
);
```

以上代码不是在Flink SQL中创建了消息队列（MQ）源表的topic，而是声明了一个名称为 `mq_stream` 的表引用。下游所有对MQ `blink_mq_test` Topic的相关DML操作，均可以使用别名 `mq_stream` 代替。声明外部表时，请注意：

- 实时计算Flink版声明表的作用域是当前作业（1个SQL文件提交后生成1个实时计算作业），即上述有关 `mq_stream` 的声明仅在当前SQL有效。相同实时计算项目下不同SQL文件（作业）中同样可以声明名称为 `mq_stream` 的表。
- 按照SQL标准定义，DDL语法中关键字、表名、列名等不区分大小写。
- 表名、列名必须以字母开头，并且名称中只能包含字母、数字或下划线。
- DDL声明不完全根据名称进行映射（取决于上游插件的性质）。建议您引用声明的字段名称、字段个数和外部表保持一致，避免因定义混乱而导致数据错乱。

  > **说明**
  > - 如果上下游插件支持根据KEY取值，则不要求两者字段数量一致，但字段名称需要一致。
  > - 如果上下游插件不支持根据KEY取值，则需要字段数量和字段顺序一致。

字段映射

声明表的字段映射根据外部数据源是否有Schema，分为两大类：

- 顺序映射

  适用于以MQ为代表的不带有Schema的系统。这类系统通常是非结构化存储系统，不支持根据KEY取值。建议您在DDL SQL声明中对字段名称进行自定义，并且和外部表的字段类型、字段数量保持一致。

  以MQ的1条记录为例。

  ```
  asavfa,sddd32,sdfds
  ```

  按照命名规范设置MQ的字段名。

```
CREATE TABLE mq_stream(
 a VARCHAR,
 b VARCHAR,
 c VARCHAR
) WITH (
 type='mq',
 topic='blink_mq_test',
 accessID='<yourAccessID>',
 accessKey='<yourAccessSecret>'
);
```

- 名称映射

  适用于带有Schema的系统。这类系统在表存储级别定义了字段名称以及字段类型，支持根据KEY取值。建议您在Flink SQL的声明中保持和外部数据存储Schema定义一致，包括字段名称、字段数量以及字段的顺序。

  > **说明** 如果外部数据存储的字段名称是大小写敏感类型，例如，表格存储，则需要在区分大小写的字段名称处，使用反引号（`）进行转换。在DDL语法中，声明表的字段名和外部表的字段名需要一致。

处理大小写敏感

SQL标准定义中，大小写是不敏感的。如下两个示例，语句的含义相同。

```
create table stream_result (
    name varchar,
    value varchar
);
```

```
create table STREAM_RESULT (
    NAME varchar,
    VALUE varchar
);
```

实时计算Flink版引用的大量外部数据源中，有时会存在要求大小写敏感的数据源。例如，表格存储（Table Store）对于大小写是敏感的。如果需要在Table Store定义大写 `NAME` 字段，您应该进行如下定义。

```
create  table STREAM_RESULT (
    `NAME` varchar,
    `VALUE` varchar
);
```

在之后所有的DML操作中，对于这个字段的引用均需要添加反引号（`），如下所示。

```
INSERT INTO tableA
SELECT
  `NAME`,
  `VALUE`
FROM
  tableB;
```

相关章节

请参见以下文档，了解如何创建实时计算Flink版数据源表、数据结果表和数据维表。

- 数据源表概述。
- 数据结果表概述。
- 概述。

4.6.2. 创建数据源表

4.6.2.1. 数据源表概述

实时计算的数据源表是流式数据存储。流式数据存储驱动实时计算的运行，因此每个实时计算作业必须提供至少1个流式数据存储。

语法

```
  CREATE TABLE tableName
      (columnName dataType [, columnName dataType ]*)
      [ WITH (propertyName=propertyValue [, propertyName=propertyValue ]*) ];
```

示例

```
CREATE TABLE metaq_stream(
 x VARCHAR,
 y VARCHAR,
 z VARCHAR
) WITH (
 type='mq',
 topic='<yourTopicName>',
 endpoint='<yourEndpoint>',
 pullIntervalMs='1000',
 accessId='<yourAccessId>',
 accessKey='<yourAccessSecret>',
 startMessageOffset='1000',
 consumerGroup='<yourConsumerGroupName>',
 fieldDelimiter='|'
);
```

获取数据源表属性字段

- 获取数据源表属性字段语法

  实时计算在源表的DDL语句中提供 `HEADER` 关键字，用于获取源表中的属性字段。

```
CREATE TABLE sourcetable
(
 `timestamp`  VARCHAR HEADER,
  name        VARCHAR,
  MsgID       VARCHAR
)WITH(
     type='<yourSourceTableType>'
);
```

上面示例中`timestamp`字段定义为`HEADER`，可从数据的属性字段读取数值，后续当成普通字段使用。

> **说明** 不同的源表（DataHub、Log Service或MQ等）存在不同的默认属性字段，部分源表支持设置自定义的属性字段，具体请参见对应的源表文档。

- 获取源表属性字段示例

  以日志服务（Log service）为例，为您介绍如何获取源表属性字段。目前日志服务默认支持如下3个属性字段。

字段名	说明
`__source__`	消息源
`__topic__`	消息主题
`__timestamp__`	日志时间

> **说明** 获取属性字段，除了按照正常逻辑声明外，还需要在类型声明后面加上`HEADER`。

示例如下：

- 示例数据

```
__topic__:  ens_altar_flow
        result:  {"MsgID":"ems0a","Version":"0.0.1"}
```

- 示例语句

```
CREATE TABLE sls_log (
  __topic__  VARCHAR HEADER,
  result     VARCHAR
)WITH(
  type ='sls'
);
CREATE TABLE sls_out (
  name     varchar,
  MsgID    varchar,
  Version  varchar
)WITH(
  type ='RDS'
);
INSERT INTO sls_out
SELECT
__topic__,
JSON_VALUE(result,'$.MsgID'),
JSON_VALUE(result,'$.Version')
FROM
sls_log
```

- 测试结果

name(VARCHAT)	MsgID(VARCHAT)	Version(VARCHAT)
ens_altar_flow	ems0a	`0.0.1`

包含窗口函数的数据源表

实时计算可以基于Event Time和Processing Time这2种时间属性对数据进行窗口聚合。包含窗口函数的作业中，数据源表的声明中需要使用到Watermark和计算列方法。实时计算基于时间属性的聚合详情，请参见时间属性。

支持创建的数据源表类型

实时计算支持创建多种类型的数据源表：

- 创建日志服务SLS源表
- 创建消息队列MQ源表
- 创建消息队列Kafka源表
- 创建表格存储Tablestore源表
- 创建全量MaxCompute源表

4.6.2.2. 创建Oracle数据库源表

本文为您介绍如何创建Oracle数据库源表，以及创建源表时使用的WITH参数、类型映射、代码示例和常见问题。

> **注意**
>
> - 本文仅适用于Blink 3.4.x及以上版本。
> - 仅支持使用Oracle 11g版本创建Oracle数据库源表。
> - 请不要手动修改Oracle Source节点的并发数量，默认一个Table对应一个并发。

语法示例

实时计算Flink版支持使用Oracle数据库作为源表，代码示例如下。

```
create table oracle_source (
    EMPLOYEE_ID BIGINT,
    START_DATE TIMESTAMP,
    END_DATE TIMESTAMP,
    JOB_ID VARCHAR,
    DEPARTMENT_ID VARCHAR
) with (
    type = 'oracle',
    url = 'jdbc:oracle:thin:@//127.0.0.1:1521/ORACLE',
    userName = 'userName',
    password = 'password',
    dbName = 'hr',
    tableName = 'job_history',
    timeField = 'START_DATE',
    startTime = '2007-1-1 00:00:00'
);
```

WITH 参数

参数	说明	是否必选	备注
type	源表类型	是	固定值为*oracle*。
url	数据库连接串	是	固定句式为 `jdbc:oracle:thin:@//`**数据库IP:端口号/数据库名** 。例如*jdbc:oracle:thin:@//127.0.0.1:1521/XE*。
userName	登录数据库的用户名	是	无
password	登录数据库的密码	是	无

参数	说明	是否必选	备注
tableName	数据库的表名。数据库的表名有以下两种表达方式： • *表名1,表名2* • *数据库名.表名1,表名2* **说明** 多个表名之间用英文逗号（,）隔开。	是	• *table1,table2* • *db1.table1,table2*
timeField	更新数据库的时间	是	无
dbName	数据库名	否	如果tableName参数配置了数据库名，则dbName不需要重复配置。
startTime	读取数据的开始时间	否	*2019-5-15 00:00:00*
timeZone	数据库时区	否	*Asia/Shanghai", "UTC*
queryTimeRangeMs	获取数据的时长，单位为毫秒。 **说明** queryTimeRangeMs参数的取值需要大于queryIntervalMs参数。	否	默认值为*5000*。
queryIntervalMs	查询数据库的时间间隔，单位为毫秒。	否	默认值为*100*。
connectionMaxActive	最大活跃连接数	否	默认值为*10*。
maxRetry	最大连接失败重试次数	否	默认值为*3*。
escapeFields	是否对数据库字段名进行转义。	否	escapeFields参数的取值如下： • *false*（默认值），不区分大小写。 • *true*，区分大小写。

参数	说明	是否必选	备注

lengthCheck	单行字段条数的检查策略	否	lengthCheck参数的取值如下：
			• *NONE*（默认值）：
			  ○ 当解析的字段个数大于定义个数时，按从左到右的顺序，取定义字段的数据使用。
			  ○ 当解析的字段个数小于定义个数时，跳过当前行数据。
			• *SKIP*：当解析的字段个数和定义个数不同时，跳过当前行数据。
			• *EXCEPTION*：当解析的字段个数和定义个数不同时，系统提示异常。
			• *PAD*：
			  ○ 当解析的字段个数大于定义个数时，按从左到右的顺序，取定义字段的数据使用。
			  ○ 当解析的字段个数小于定义个数时，按从左到右顺序，在行尾用*null*填充缺少的字段。

columnErrorDebug	是否打开调试开关。 **说明** 如果打开调试开关，则会将解析异常的日志打印出来。	否	默认值为*false*。

类型映射

Oracle字段类型	实时计算Flink版字段类型
• CHAR • VARCHAR • VARCHAR2	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

代码示例

实时计算Flink版包含Oracle数据库源表的代码示例如下。

```
create table oracle_source (
    EMPLOYEE_ID BIGINT,
    START_DATE TIMESTAMP,
    END_DATE TIMESTAMP,
    JOB_ID VARCHAR,
    DEPARTMENT_ID VARCHAR
) with (
    type = 'oracle',
    url = 'jdbc:oracle:thin:@//127.0.0.1:1521/ORACLE',
    userName = 'userName',
    password = 'password',
    dbName = 'hr',
    tableName = 'job_history',
    timeField = 'START_DATE',
    startTime = '2007-1-1 00:00:00'
);
create table test_out(
    EMPLOYEE_ID BIGINT,
    START_DATE TIMESTAMP,
    END_DATE TIMESTAMP,
    JOB_ID VARCHAR,
    DEPARTMENT_ID VARCHAR
) with (
    type='print'
);
INSERT INTO test_out
SELECT
    EMPLOYEE_ID,
    START_DATE,
    END_DATE,
    JOB_ID,
    DEPARTMENT_ID
from oracle_source;
```

常见问题

Q：查询不到数据该如何处理？

A： 查询不到数据是因为Blink运行故障。您需要查看TaskManager的 `Round start:[{}], end:[{}]` 和 `Round read records` 日志，如果未查询到日志数据，则Blink运行故障。

> **说明**
>
> - `Round start:[{}], end:[{}]` 用来显示查询数据的起始时间戳。
> - `Round read records` 用来显示查询到的数据记录。

4.6.2.3. 创建交互式分析Hologres源表

本文为您介绍创建交互式分析Hologres源表DDL定义，以及创建源表时用到的WITH参数、类型映射和代码示例。

使用限制

本文仅适用于Blink 3.6.0及以上版本，如果您的Blink为3.6.0以下的版本，建议您升级到Blink 3.7.0及以上版本。

注意事项

- 在流数据和批数据处理中都可以使用Hologres源表。
- Hologres源表支持Projection Pushdown操作，您可以只读取需要的列。
- Hologres源表使用快照语句高速读取当前数据，读取完后结束作业。如果出现读取数据失败，将重新执行读取操作。
- 并发的Blink作业都可以读取一个或多个Hologres Shard，建议您配置的Blink并发数小于等于Hologres的Shard数。
- 实时消费Hologres源表的数据需要开启Binlog。开启Binlog的方法，请参见订阅Hologres Binlog。

什么是交互式分析Hologres

交互式分析Hologres兼容PostgreSQL协议，与大数据生态紧密连接，支持高并发、低延时实时分析处理PB级数据，让您轻松使用现有BI（Business Intelligence）工具对数据进行多维分析和业务探索。

DDL定义

```
create table mysource(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='...',
  tablename='...',
  username='...',
  password='...',
  endpoint='...',
  field_delimiter='...' --该参数可选。
);
```

WITH参数

参数	说明	是否必填	备注
type	源表类型	是	固定值为*hologres*。
dbname	数据库名称	是	无
tablename	表名称 **说明** 如果Schema不为Public时，则tableName需要填写为schema.tableName。	是	无

参数	说明	是否必填	备注
username	用户名，请填写阿里云账号的AccessKey ID。	是	无
password	密码，请填写阿里云账号的AccessKey Secret。	是	无
endpoint	Hologres端点	是	详情请参见访问域名。
field_delimiter	导出数据时，不同行之间使用的分隔符。 注意 不能在数据中插入分隔符。	是	默认值为 *"\u0002"*。
bulkread	是否全量读取列存表数据。	否	取值如下： • true：全量读取列存表数据。 • false（默认值）：不全量读取列存表数据。

类型映射

Hologres	BLINK
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL

Hologres	BLINK
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

代码示例

包含Hologres源表的实时计算作业代码示例如下。

```
create table mysource(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='...',
  tablename='...',
  username='...',
  password='...',
  endpoint='...',
  field_delimiter='...' --该参数可选。
);
create table print_output(
 a varchar,
 b BIGINT,
 c BIGINT
) with (
  type='print'
);
INSERT INTO print_output
SELECT
  a, b, c
from mysource;
```

4.6.2.4. 创建日志服务SLS源表

本文为您介绍如何创建日志服务SLS源表，以及创建过程涉及到的属性字段、WITH参数和类型映射。

> **注意** 本文仅适用于Blink 1.4.5及以上版本。

什么是日志服务

日志服务SLS是针对日志类数据的一站式服务，对于日志服务而言，数据格式类似JSON，示例如下。

```
{
    "a": 1000,
    "b": 1234,
    "c": "li"
}
```

日志服务本身是流数据存储，实时计算Flink版能将其作为流式数据的输入。

语法示例

实时计算Flink版日志服务源表DDL示例如下（代码中的 `sls` 代表日志服务）。

```
create table sls_stream(
  a INT,
  b INT,
  c VARCHAR
) with (
  type ='sls',
  endPoint ='http://cn-hangzhou-share.log.aliyuncs.com',
  accessId ='<yourAccessId>',
  accessKey ='<yourAccessKey>',
  startTime = '2017-07-05 00:00:00',
  project ='<yourProjectName>',
  logStore ='<yourLogStoreName>',
  consumerGroup ='<yourConsumerGroupName>'
);
```

WITH参数

参数	说明	是否必选	备注
type	源表类型	是	固定值为sls。
endPoint	消费端点信息	是	服务入口
accessId	AccessKey ID	是	无
accessKey	AccessKey Secret	是	无
project	读取的SLS项目名称	是	无
logStore	Project下的具体的LogStore名称	是	无
startTime	日志消费的开始时间	否	无
consumerGroup	消费组名	否	您可以自定义消费组名（没有固定格式）。
heartBeatIntervalMills	消费客户端心跳间隔时间	否	默认值为10000，单位为毫秒。
maxRetryTimes	读取最大尝试次数	否	默认值为5。
batchGetSize	单次读取logGroup的条数	否	默认值为100。
columnErrorDebug	是否打开调试开关	否	默认值为false，不打开。如果选择打开，则打印解析异常的日志。

参数	说明	是否必选	备注

startupMode	启动消费模式	否	取值如下：
			• TIMESTAMP（默认值）：每个Shard从指定时间开始消费。
			• Earliest：每个Shard从最早位置开始消费。
			• Latest：每个Shard从最新位置开始消费。
			• Group_Offsets：每个Shard优先从服务端保存的Checkpoint开始消费，必须指定consumerGroup。消费模式有以下几种情况：
			  ○ 如果从Flink State中恢复状态成功，则从Flink状态中的Checkpoint开始消费。
			  ○ 如果从Flink State中恢复状态失败：
			  ■ 如果consumerGroup中存在Checkpoint，则尝试从consumerGroup的Checkpoint开始消费。
			  ■ 如果consumerGroup中不存在Checkpoint：
			  ■ 指定了startTime：从startTime开始消费。
			  ■ 未指定startTime：每个shard从最早位置开始消费。
			注意
			• 仅Blink 3.6.5及以上版本支持该参数。
			• 仅当state中不存在Checkpoint时，上述配置才有效。

> 说明
>
> - 实时计算Flink版1.6.0及以下版本，在指定consumerGroup的Shards数目时，可能会影响读取性能，该缺陷正在修复。
> - SLS暂不支持MAP类型的数据。
> - SLS对于不存在字段会置为Null。
> - 字段顺序支持无序（建议字段顺序和表中定义一致）。
> - 输入数据源为JSON形式时，注意定义分隔符，并且需要采用内置函数JSON_VALUE分析，否则就会解析失败，报错如下。
>
> ```
> 2017-12-25 15:24:43,467 WARN [Topology-0 (1/1)] com.alibaba.blink.streaming.connectors.common.source.parse.DefaultSourceCollector - Field missing error, table column number: 3, data column number: 3, data filed number: 1, data: [{"lg_order_code":"LP00000005","activity_code":"TEST_CODE1","occur_time":"2017-12-10 00:00:01"}]
> ```
>
> - batchGetSize设置不能超过1000，否则会报错。
> - batchGetSize指明的是logGroup获取的数量。如果单条logItem的大小和batchGetSize都很大，有可能会导致频繁的垃圾回收（Garbage Collection），这种情况下该参数应调小。

类型映射

日志服务和实时计算Flink版字段类型对应关系如下。建议您使用该对应关系进行DDL声明。

日志服务字段类型	实时计算Flink版字段类型
STRING	VARCHAR

属性字段

目前Flink SQL默认支持3个SLS属性字段的获取，也支持其它自定义字段的写入。属性字段使用方法请参见获取数据源表属性字段。

字段名	注释说明
`__source__`	消息源
`__topic__`	消息主题
`__timestamp__`	日志时间

代码示例

```
create table sls_input(
  a int,
  b int,
  c varchar
) with (
  type ='sls',
  endPoint ='http://cn-hangzhou-share.log.aliyuncs.com',
  accessId ='<yourAccessI>',
  accessKey ='<yourAccessKey>',
  startTime = '2017-07-05 00:00:00',
  project ='ali-cloud-streamtest',
  logStore ='stream-test',
  consumerGroup ='consumerGroupTest1'
);
create table print_output(
 a int,
 b int,
 c varchar
) with (
  type='print'
);
INSERT INTO print_output
SELECT
  a, b, c
from sls_input;
```

常见问题

- Q：为什么Job的整体延迟增加，或有窗口聚合的Job无输出？

  A：如果一个Partition没有新数据写入，会导致上述情况，只需要把并发数调整为读写的Partition数量即可。

- Q：如何设置并发数？

  A：并发数建议等于Partition数量，否则当两个Partition读取速度差异较大时，理论上在追数据场景，存在数据被过滤掉和数据延迟的风险。

- Q：Flink Job延迟增大应该如何排查？

  A：SLS源表可能会发生Shard分裂，分裂后的Shard index会不连续，导致Flink延迟增大。如果发现Flink Job延迟增大，请查看SLS源是否发生分裂。

- Q：如何获取属性字段？

  A：属性字段获取方法，请参见获取数据源表属性字段。

  > **说明** 本地调试时无法抽取到属性字段数据，建议您使用线上调试方法，在日志中进行查看，详情请参见线上调试。

相关文档

- 日志服务帮助文档，请参见什么是日志服务。
- 日志服务中实时计算消费文档，请参见实时计算（Blink）消费。

4.6.2.5. 创建消息队列MQ源表

本文为您介绍实时计算如何创建消息队列MQ源表以及创建过程涉及到的CSV类格式、WITH参数和类型映射。

> **说明** 如果您需要使用带独立命名空间的MQ，请使用Blink 3.x作业版本。

什么是消息队列MQ

消息队列MQ是阿里云专业消息中间件，是企业级互联网架构的核心产品。实时计算可以将消息队列作为流式数据输入。

示例

```
create table mq_stream(
  x varchar,
  y varchar,
  z varchar
) with (
  type='mq',
  topic='<yourTopicName>',
  endpoint='<yourEndpoint>',
  pullIntervalMs='1000',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  startMessageOffset='1000',
  consumerGroup='<yourConsumerGroup>',
  fieldDelimiter='|'
);
```

> **说明** MQ实际上是一个非结构化存储格式，对于数据的Schema不提供强制定义，完全由业务层指定。目前实时计算支持类CSV格式文本和二进制格式。

CSV格式

假设您的1条CSV格式消息记录如下。

```
1,name,male
2,name,female
```

> **说明** 1条MQ消息可以包括0条到多条数据记录，记录之间使用 `\n` 分隔。

在实时计算作业中，声明MQ数据源表的DDL如下。

```
create table mq_stream(
  id varchar,
  name varchar,
  gender varchar
) with (
  type='mq',
  topic='<yourTopicName>',
  endpoint='<ourEndpoint>',
  pullIntervalMs='1000',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  startMessageOffset='1000',
  consumerGroup='<yourConsumerGroup>',
  fieldDelimiter='|'
);
```

二进制格式

二进制格式测试代码如下。

```
create table source_table (
  mess varbinary
) with (
  type = 'mq',
  endpoint = '<yourEndpoint>',
  pullIntervalMs='500',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  topic = '<yourTopicName>',
  consumerGroup='<yourConsumerGroup>'
);
create table out_table (
  commodity varchar
) with (
  type='print'
);
INSERT INTO out_table
SELECT
  cast(mess as varchar)
FROM source_table;
```

说明

- `cast(mess as varchar)` 需在实时计算2.0及以上版本使用，如果版本低于2.0，请先升级。
- VARBINARY只能入参一次。

WITH参数

参数	说明	是否必填	备注
type	源表类型	是	固定值为mq。
topic	topic名称	是	无。

endPoint	endPoint地址	是	阿里云消息队列提供内网服务MQ（非公网region）和公网服务MQ（公网region）两种类型，请务必根据您购买的MQ的类型选择对应正确的接入地址（endPoint）：
			• Blink 3.7.10及以上版本的作业，需要使用TCP协议客户端接入点，详情请参见 关于TCP内网接入点设置的公告。接入点获取方式如下：
			  ◦ 内网服务MQ（阿里云经典网络/VPC）接入地址：在MQ控制台目标实例详情中，选择**接入点 > TCP协议客户端接入点 > 内网访问**，获取对应的endPoint。
			  ◦ 公网服务MQ接入地址：在MQ控制台目标实例详情中，选择**接入点 > TCP协议客户端接入点 > 公网访问**，获取对应的endPoint。
			• Blink 3.7.10（不含）以下版本的作业，使用如下接入点：
			  ◦ 内网服务MQ（阿里云经典网络/VPC）接入地址：
			  ▪ 华东1（杭州）、华东2（上海）、华北1（青岛）、华北2（北京）、华南1（深圳）、中国（香港）：`onsaddr-internal.aliyun.com:8080`
			  ▪ 亚太东南1（新加坡）：`ap-southeastaddr-internal.aliyun.com:8080`。
			  ▪ 中东东部1（迪拜）：`ons-me-east-1-internal.aliyuncs.com:8080`。
			  ▪ 亚太南部1（孟买）：`ons-ap-south-1-internal.aliyuncs.com:8080`。
			  ▪ 亚太东南3（吉隆坡）：`ons-ap-southeast-3-internal.aliyun.com:8080`。
			  ◦ 公网服务MQ接入地址：`onsaddr-internet.aliyun.com:80`。

参数	说明	是否必填	备注
			说明 • 如果您已使用了Blink 3.7.10（不含）以下版本的RocketMQ Connector，则您需要将您的实时计算作业升级至Blink 3.7.10及以上版本，并将作业中EndPoint参数取值更改为新的RocketMQ接入点，旧的RocketMQ接入点存在稳定性风险或不可用的问题，详情请参见RocketMQ接入点变更导致实时计算作业适配升级公告。 • 内网服务无法跨域访问。例如，您所购买的实时计算服务的地域为华东1，但是购买的消息队列MQ服务的地域为华东2，则无法访问。 • 由于阿里云网络安全策略动态变化，实时计算连接公网服务MQ时可能会出现网络连接问题，推荐您使用内网服务MQ，如果在使用公网服务MQ时出现异常，请您提交工单进行咨询。
accessId	AccessKey ID	是	无
accessKey	AccessKey Secret	是	无
consumerGroup	订阅消费group名称	是	无
pullIntervalMs	拉取时间间隔	否	单位为毫秒
startTime	消息消费启动的时间点	否	无
startMessageOffset	消息开始的偏移量	否	如果填写，将优先以startMessageoffset的位点开始加载。
tag	订阅的标签	否	无
lineDelimiter	解析block时行分隔符	否	默认值为 `\n` 。
fieldDelimiter	字段分隔符	否	默认值为 `\u0001` 。表示在只读模式下以 `\u0001` （ `\u0001` 在只读模式不可见）作为分隔符，在编辑模式下以 `^A` 作为分隔符。
encoding	编码格式	否	默认值为 `utf-8` 。

lengthCheck	单行字段条数检查策略	否	默认值为NONE，表示：
			• 解析出的字段数大于定义字段数时，按从左到右的顺序，取定义字段数量的数据。
			• 解析出的字段数小于定义字段数时，跳过这行数据。
			其它可选值为SKIP、EXCEPTION和PAD。
			• SKIP：解析出的字段数和定义字段数不同时跳过这行数据。
			• EXCEPTION：解析出的字段数和定义字段数不同时提示异常。
			• PAD：按从左到右顺序填充。
			  ◦ 解析出的字段数大于定义字段数时，按从左到右的顺序，取定义字段数量的数据。
			  ◦ 解析出的字段数小于定义字段数时，在行尾用null填充缺少的字段。

参数	说明	是否必填	备注
columnErrorDebug	是否打开调试开关	否	默认值为FALSE。如果设置为TRUE，则打印解析异常的Log。
instanceID	MQ实例ID	否	如果MQ实例无独立命名空间，则不可以使用instanceID参数。如果MQ实例有独立命名空间，则instanceID参数必选。

类型映射

MQ字段类型	实时计算字段类型
STRING	VARCHAR

4.6.2.6. 创建消息队列Kafka源表

本文为您介绍如何创建实时计算Flink版消息队列Kafka源表、Kafka版本对应关系及Kafka消息解析示例。

> 注意
> - 本文仅适用于Blink 2.0及以上版本。
> - 本文仅适用于独享模式。
> - Kafka源表支持读取自建Kafka集群，但需要注意版本对应关系，以及自建集群和实时计算Flink版集群的网络环境配置。
> - 二进制数据不支持本地调试，语法检查没有问题请进行线上调试，详情请参见线上调试。

什么是Kafka源表

消息队列Kafka版是阿里云提供的分布式、高吞吐、可扩展的消息队列服务。消息队列Kafka版广泛用于日志收集、监控数据聚合、流式数据处理、在线和离线分析等大数据领域。实时计算Flink版支持将 Kafka作为流式数据的数据源表或结果表。

从Kafka输出的数据为序列化后的VARBINARY（二进制）格式。对于输出的每条数据，需要您编写自定义表值函数（UDTF）将其解析为序列化前的数据结构。Kafka源表数据解析流程通常为：Kafka Source Table -> UDTF -> Realtime Compute for Apache Flink -> Sink。此外，Flink SQL中也支持通过CAST函数将VARBINARY解析为VARCHAR类型。自定义表值函数请参见自定义表值函数（UDTF）。

DDL定义

Kafka源表定义DDL部分必须与以下SQL完全一致，可以更改WITH参数中的设置。

```
create table kafka_stream(   --必须和Kafka源表中的5个字段的顺序和类型保持一致。
  messageKey VARBINARY,
  `message`    VARBINARY,
  topic      VARCHAR,
  `partition`  INT,
  `offset`     BIGINT
) with (
  type ='kafka010',
  topic = '<yourTopicName>',
  `group.id` = '<yourGroupId>',
  ...
);
```

WITH参数

- 通用配置

参数	注释说明	是否必选	备注
type	Kafka对应版本	是	Kafka版本需要是Kafka08、Kafka09、Kafka010或Kafka011。版本对应关系请参见Kafka版本对应关系。
topic	读取的单个topic	是	无
topicPattern	读取一批topic的表达式	否	Topic用竖线（|）分割。例如，`topic1|topic2|topic3`。详情请参见Class Pattern。

参数	注释说明	是否必选	备注
startupMode	启动位点	否 说明 建议手动配置该参数。	启动位点取值如下： ○ GROUP_OFFSETS（默认值）：根据Group读取。 ○ EARLIEST：从Kafka最早分区开始读取。 ○ LATEST：从Kafka最新位点开始读取。 ○ TIMESTAMP：从指定的时间点读取。 您可以在WITH参数中指定startTimeMs（时间戳）或startTime参数（按照 `yyyy-MM-dd HH:mm:ss` 格式）。优先使用startTimeMs，不指定则按实时计算Flink版指定的启动位点时间开始消费。 说明 ○ 如果没有明文指定，则默认为GROUP_OFFSETS模式。 ○ 设置为TIMESTAMP模式时，需要在作业参数中明文指定时区。例如 `blink.job.timeZone=Asia/Shanghai` 。 ○ GROUP_OFFSETS模式下，GroupID的第一次消费，没有设置偏移（Offset）值，默认从Kafka分区末尾开始读取数据。 ○ 仅Kafka010和Kafka011版本支持TIMESTAMP。
partitionDiscoveryIntervalMS	定时检查是否有新分区产生	否	○ Kafka 08版本：系统默认开启该功能。 ○ Kafka 09版本及以上版本：不支持partitionDiscoveryIntervalMS参数。您可以通过设置WITH参数，extraConfig='flink.partition-discovery.interval-millis=60000'达到相同的效果。 单位为毫秒，默认值为60000，即1分钟。
extraConfig	额外的KafkaConsumer配置项目	否	不在可选配置项中，但是期望额外增加的配置。例如：`` `fetch.message.max.bytes=104857600` ``，多个配置使用分号（;）分隔。

- Kafka08配置

- Kafka08必选配置

参数	注释说明	是否必选
group.id	消费组ID	是
zookeeper.connect	zk链接地址	是

- 可选配置Key
  - consumer.id
  - socket.timeout.ms
  - fetch.message.max.bytes
  - num.consumer.fetchers
  - auto.commit.enable
  - auto.commit.interval.ms
  - queued.max.message.chunks
  - rebalance.max.retries
  - fetch.min.bytes
  - fetch.wait.max.ms
  - rebalance.backoff.ms
  - refresh.leader.backoff.ms
  - auto.offset.reset
  - consumer.timeout.ms
  - exclude.internal.topics
  - partition.assignment.strategy
  - client.id
  - zookeeper.session.timeout.ms
  - zookeeper.connection.timeout.ms
  - zookeeper.sync.time.ms
  - offsets.storage
  - offsets.channel.backoff.ms
  - offsets.channel.socket.timeout.ms
  - offsets.commit.max.retries
  - dual.commit.enabled
  - partition.assignment.strategy
  - socket.receive.buffer.bytes
  - fetch.min.bytes

- Kafka09/Kafka010/Kafka011配置

- Kafka09/Kafka010/Kafka011必选配置

参数	注释说明
group.id	消费组ID
bootstrap.servers	Kafka集群地址

- Kafka09/Kafka010/Kafka011可选配置，请参见如下Kafka官方文档进行配置：
  - Kafka09
  - Kafka010
  - Kafka011

  当需要配置某选项时，在DDL中的WITH部分增加对应的参数即可。例如，配置SASL登录，需增加`security.protocol`、`sasl.mechanism`和`sasl.jaas.config`3个参数，示例如下。

```
create table kafka_stream(
  messageKey varbinary,
  `message` varbinary,
  topic varchar,
  `partition` int,
  `offset` bigint
) with (
  type ='kafka010',
  topic = '<yourTopicName>',
  `group.id` = '<yourGroupId>',
  ...,
  `security.protocol`='SASL_PLAINTEXT',
  `sasl.mechanism`='PLAIN',
  `sasl.jaas.config`='org.apache.kafka.common.security.plain.PlainLoginModule required
username="<yourUserName>" password="<yourPassword>";'
);
```

Kafka版本对应关系

type	Kafka版本
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2及以上

Kafka消息解析示例

- 场景1：将Kafka中的数据进行计算，并将计算结果输出到RDS。

  Kafka中保存了JSON格式数据，需要使用实时计算Flink版进行计算，消息格式示例如下。

```
{
  "name":"Alice",
  "age":13,
  "grade":"A"
}
```

- 方法1：Kafka SOURCE->Realtime Compute for Apache Flink->RDS SINK

  Blink 2.2.7及以上版本支持将VARBINARY类型通过CAST函数转换为VARCHAR类型，再通过JSON_VALUE函数对Kafka数据进行解析，示例如下。

```
CREATE TABLE kafka_src (
  messageKey  VARBINARY,
  `message`   VARBINARY,
  topic       VARCHAR,
  `partition` INT,
  `offset`    BIGINT
) WITH (
  type = 'kafka010'   --请参见Kafka版本对应关系。
);
CREATE TABLE rds_sink (
  `name`       VARCHAR,
  age         VARCHAR,
  grade       VARCHAR
) WITH(
  type='rds'
);
CREATE VIEW input_view AS
  SELECT CAST(`message` as VARCHAR ) as `message`
FROM kafka_src;
INSERT INTO rds_sink
SELECT
  JSON_VALUE(`message`,'$.name'),
  JSON_VALUE(`message`,'$.age'),
  JSON_VALUE(`message`,'$.grade')
FROM input_view;
```

- 方法2：Kafka Source->UDTF->Realtime Compute for Apache Flink->RDS Sink

  针对不规则数据、复杂JSON数据，需要您自行编写UDTF代码进行解析，示例如下。

  - SQL

```
-- 定义解析Kafka message的UDTF。
CREATE FUNCTION kafkaparser AS 'com.alibaba.kafkaUDTF';
-- 定义源表。注意：Kafka源表DDL字段必须与以下示例完全一致。WITH中参数可以修改。
CREATE TABLE kafka_src (
  messageKey  VARBINARY,
  `message`   VARBINARY,
  topic       VARCHAR,
  `partition` INT,
  `offset`    BIGINT
) WITH (
  type = 'kafka010',    --请参见Kafka版本对应关系。
  topic = 'test_kafka_topic',
  `group.id` = 'test kafka consumer group',
```

```
  bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'
);
CREATE TABLE rds_sink (
  name       VARCHAR,
  age        INT,
  grade      VARCHAR,
  updateTime TIMESTAMP
) WITH(
  type='rds',
  url='jdbc:mysql://localhost:3306/test',
  tableName='test4',
  userName='test',
  password='<yourDatabasePassword>'
);
-- 使用UDTF，将二进制数据解析成格式化数据。
CREATE VIEW input_view (
  name,
  age,
  grade,
  updateTime
) AS
SELECT
  T.name,
  T.age,
  T.grade,
  T.updateTime
FROM
  kafka_src as S,
  LATERAL TABLE (kafkaparser (`message`)) as T (
  name,
  age,
  grade,
  updateTime
);
-- 使用解析出的格式化数据进行计算，并将结果输出到RDS。
INSERT INTO rds_sink
SELECT
  name,
  age,
  grade,
  updateTime
FROM input_view;
```

- UDTF

  UDTF创建步骤请参见自定义表值函数（UDTF）。Blink 2.2.4版本Maven依赖，示例如下。

```
<dependencies>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-core</artifactId>
        <version>blink-2.2.4-SNAPSHOT</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-streaming-java_2.11</artifactId>
        <version>blink-2.2.4-SNAPSHOT</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-table_2.11</artifactId>
        <version>blink-2.2.4-SNAPSHOT</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
        <version>1.2.9</version>
    </dependency>
</dependencies>
```

```
package com.alibaba;
import com.alibaba.fastjson.JSONObject;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;
import java.sql.Timestamp;
public class kafkaUDTF extends TableFunction<Row> {
    public void eval(byte[] message) {
        try {
            /* input message :
                {
                  "name":"Alice",
                  "age":13,
                  "grade":"A",
                  "updateTime":1544173862
                }
            */
            String msg = new String(message, "UTF-8");
            try {
                JSONObject data = JSON.parseObject(msg);
                if (data != null) {
                    String name = data.getString("name") == null ? "null" : data.getS
tring("name");
                    Integer age = data.getInteger("age") == null ? 0 : data.getIntege
r("age");
                    String grade = data.getString("grade") == null ? "null" : data.ge
tString("grade");
                    Timestamp updateTime = data.getTimestamp("updateTime");
                    Row row = new Row(4);
                    row.setField(0, name);
                    row.setField(1, age);
                    row.setField(2, grade);
                    row.setField(3,updateTime );
                    System.out.println("Kafka message str ==>" + row.toString());
                    collect(row);
                }
            } catch (ClassCastException e) {
                System.out.println("Input data format error. Input data " + msg + "is
not json string");
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
    @Override
    // 如果返回值是Row，重新加载UDTF这个方法，并指明系统返回的字段类型。
    public DataType getResultType(Object[] arguments, Class[] argTypes) {
        return DataTypes.createRowType(DataTypes.STRING, DataTypes.INT, DataTypes.STR
ING, DataTypes.TIMESTAMP);
    }
}
```

- 场景2：从Kafka读取数据，输入实时计算Flink版进行窗口计算。

  按照实时计算Flink版目前的设计，滚动或滑动等窗口操作，必须在源表DDL上定义Watermark。Kafka源表比较特殊。如果要以Kafka中message字段中的Event Time进行窗口操作，需要先从message字段使用UDX解析出Event Time，才能定义Watermark。在Kafka源表场景中，需要使用计算列。例如Kafka中写入数据：`2018-11-11 00:00:00|1|Anna|female`。计算流程为：Kafka Source->UDTF->Realtime Compute for Apache Flink->RDS Sink。

  - 方法1：Kafka SOURCE->Realtime Compute for Apache Flink->RDS SINK

    Blink 2.2.7及以上版本支持将VARBINARY类型通过CAST函数转换为VARCHAR类型，再通过JSON_VALUE函数对Kafka数据进行解析，示例如下。

    ```
    CREATE TABLE kafka_src (
      messageKey VARBINARY,
      `message` VARBINARY,
      topic VARCHAR,
      `partition` INT,
      `offset` BIGINT,
      ts as to_timestamp(json_value(cast(`message` as VARCHAR ),'$.nodes.time')),
      WATERMARK wk FOR ts as withOffset(ts, 2000)
    ) WITH (type = 'kafka' --请参见Kafka版本对应关系。
    );
    CREATE TABLE rds_sink (
      starttime TIMESTAMP ,
      endtime   TIMESTAMP ,
      `message` BIGINT
    ) WITH (type = 'rds');
    INSERT
      INTO rds_sink
    SELECT
      TUMBLE_START(ts, INTERVAL '1' MINUTE),
      TUMBLE_END(ts, INTERVAL '1' MINUTE),
      count(`message`)
    FROM
      kafka_src
    GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE);
    ```

  - 方法2：Kafka SOURCE->UDTF->Realtime Compute for Apache Flink->RDS SINK

    - SQL

      ```
      -- 定义解析Kafka message的UDTF。
      CREATE FUNCTION kafkapaser AS 'com.alibaba.kafkaUDTF';
      CREATE FUNCTION kafkaUDF AS 'com.alibaba.kafkaUDF';
      -- 定义源表，注意：Kafka源表DDL字段必须与以下示例一模一样。WITH中参数可改。
      create table kafka_src (
        messageKey VARBINARY,
        `message` VARBINARY,
        topic VARCHAR,
        `partition` INT,
        `offset` BIGINT,
        ctime AS TO_TIMESTAMP(kafkaUDF(`message`)), -- 定义计算列，计算列可理解为占位符，源表中并没有这一列，其中的数据可经过下游计算得出。注意:计算列的类型必须为TIMESTAMP才能创建Watermark。
        watermark for `ctime` as withoffset(`ctime`,0) -- 在计算列上定义Watermark。
      ) WITH (
      ```

```
) WITH (
  type = 'kafka010', -- 请参见Kafka版本对应关系。
  topic = 'test_kafka_topic',
  `group.id` = 'test_kafka_consumer_group',
  bootstrap.servers = 'ip1:port1,ip2:port2,ip3:port3'
);
create table rds_sink (
  `name` VARCHAR,
  age INT,
  grade VARCHAR,
  updateTime TIMESTAMP
) WITH(
  type='rds',
  url='jdbc:mysql://localhost:3306/test',
  tableName='test4',
  userName='test',
  password='<yourPassword>'
);
-- 使用UDTF，将二进制数据解析成格式化数据。
CREATE VIEW input_view AS
SELECT
  S.ctime,
  T.`order`,
  T.`name`,
  T.sex
  from
  kafka_src as S,
  LATERAL TABLE (kafkapaser (`message`)) as T (
  ctime,
  `order`,
  `name`,
  sex
);
-- 对input_view中输出的数据做计算。
CREATE VIEW view2 (
  cnt,
  sex
) AS
  SELECT
  COUNT(*) as cnt,
  T.sex
  from
  input_view
Group BY sex, TUMBLE(ctime, INTERVAL '1' MINUTE);
-- 使用解析出的格式化数据进行计算，并将结果输出到RDS。
insert into rds_sink
  SELECT
  cnt,sex
from view2;
```

- UDF&UDTF

  UDF和UDTF创建步骤请参见自定义标量函数（UDF）和自定义表值函数（UDTF）。Blink 2.2.4版本Maven依赖，示例如下。

```
<dependencies>
      <dependency>
          <groupId>org.apache.flink</groupId>
          <artifactId>flink-core</artifactId>
          <version>blink-2.2.4-SNAPSHOT</version>
          <scope>provided</scope>
      </dependency>
      <dependency>
          <groupId>org.apache.flink</groupId>
          <artifactId>flink-streaming-java_2.11</artifactId>
          <version>blink-2.2.4-SNAPSHOT</version>
          <scope>provided</scope>
      </dependency>
      <dependency>
          <groupId>org.apache.flink</groupId>
          <artifactId>flink-table_2.11</artifactId>
          <version>blink-2.2.4-SNAPSHOT</version>
          <scope>provided</scope>
      </dependency>
      <dependency>
          <groupId>com.alibaba</groupId>
          <artifactId>fastjson</artifactId>
          <version>1.2.9</version>
      </dependency>
  </dependencies>
```

- UDTF

```
package com.alibaba;
import com.alibaba.fastjson.JSONObject;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.types.Row;
import java.io.UnsupportedEncodingException;
/**
  以下例子解析输入Kafka中的JSON字符串，并将其格式化输出。
**/
public class kafkaUDTF extends TableFunction<Row> {
    public void eval(byte[] message) {
        try {
          // 读入一个二进制数据，并将其转换为String格式。
            String msg = new String(message, "UTF-8");
                // 提取JSON Object中各字段。
                    String ctime = Timestamp.valueOf(data.split('\\|')[0]);
                    String order = data.split('\\|')[1];
                    String name = data.split('\\|')[2];
                    String sex = data.split('\\|')[3];
                    // 将解析出的字段放到要输出的Row()对象。
                    Row row = new Row(4);
                    row.setField(0, ctime);
                    row.setField(1, age);
                    row.setField(2, grade);
                    row.setField(3, updateTime);
                    System.out.println("Kafka message str ==>" + row.toString());
                    // 输出一行。
                    collect(row);
            } catch (ClassCastException e) {
                System.out.println("Input data format error. Input data " + msg + "
is not json string");
            }
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
    @Override
    // 如果返回值是Row，重新加载UDTF这个方法，并指明系统返回的字段类型。
    // 定义输出Row()对象的字段类型。
    public DataType getResultType(Object[] arguments, Class[] argTypes) {
        return DataTypes.createRowType(DataTypes.TIMESTAMP,DataTypes.STRING, DataTy
pes.Integer, DataTypes.STRING,DataTypes.STRING);
    }
}
```

- UDF

```
package com.alibaba;
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class KafkaUDF extends ScalarFunction {
    // 可选，open方法可以不写。
    // 需要import org.apache.flink.table.functions.FunctionContext;
    public String eval(byte[] message) {
         // 读入一个二进制数据，并将其转换为String格式。
        String msg = new String(message, "UTF-8");
        return msg.split('\\|')[0];
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    //可选，close方法可以不写。
    @Override
    public void close() {
        }
}
```

自建Kafka

- 示例

```
create table kafka_stream(
  messageKey VARBINARY,
  `message` VARBINARY,
  topic varchar,
  `partition` int,
  `offset` bigint
) with (
  type ='kafka011',
  topic = 'kafka_01',
  `group.id` = 'CID_blink',
  bootstrap.servers = '192.168.0.251:****'
);
```

- WITH参数

  详情请参见WITH参数。

  > **说明**
  > - bootstrap.servers参数需要填写自建的地址和端口号。
  > - 仅在Blink 2.2.6及以上版本支持阿里云Kafka或自建Kafka显示TPS和RPS等指标信息。

常见问题

- Q：作业启动时产生如下报错，应该如何处理？

```
ERR_ID:
     SQL-00010007
CAUSE:
     Could not create table 'kafka_source' as source table
ACTION:
     Please refer to details section for hint.
     If it doesn't help, please contact customer support
DETAIL:
     java.lang.IllegalArgumentException: Startup time[1566481803000] must be before curre
nt time[1566453003356].
```

A：时区设置错误导致以上报错。请在作业参数中增加如下参数。

```
blink.job.timeZone=Asia/Shanghai
```

- Q：自建Kafka消费不到数据，应该如何排查？

  A：

  - 问题原因

    Kafka各broker会向zookeeper汇报自身Meta信息，Kafka consumer最终会根据broker Meta信息listener_security_protocol_map中的Endpoint地址（例如IP或本机域名加端口）访问broker拉取数据。如果实时计算Flink版作业所在机器无法访问Endpoint地址，则connector中consumer将无法拉取数据，导致流程的停滞。

  - 排查思路

    a. 查看**zookeeper > broker > listener_security_protocol_map > Endpoint**信息。

    b. 通过**网络探测**功能检测Endpoint的IP或域名是否可访问。

    c. 登录机器再次确认。

  - 解决方案

    - 如果Endpoint采用的是IP形式，则确认Kafka服务端是否配置了数据存储白名单配置。如果未配置，请配置后重试。
    - 如果Endpoint采用的是域名形式，由于实时计算Flink版独享集群内部无法解析域名，所以需要在白名单已配置的情况下，通过以下两种解方案解决：
      - 不能重启Kafka服务

        购买云解析PrivateZone，配置所有Kafka Broker的域名解析，通过域名进行网络探测成功后，重启实时计算Flink版作业。

      - 可以重启Kafka服务

        将Kafka的advertised.listeners属性配置为相应Broker的IP和端口（通常为bootstrap.servers中IP和Port），以保证网络畅通，然后重启实时计算Flink版作业。

- Q：消费Kafka时，为什么作业一启动后马上就结束了？

A：Blink 3.3.0以下版本的Kafka Connector，使用timestamp模式时。如果设置启动位点后，Kafka的所有partition均没数据，则Connector会判定没有partition消费而结束作业。通常对应的 `TaskManager.log` 日志中会存在类似 `Consumer subtask {} initially has no partitions to read from.` 的日志信息。建议升级至Blink 3.3.0及以上版本。

- Q：实时计算Flink版中的COMMIT OFFSET有什么作用？

  A：实时计算Flink版默认采用commitOffsetOnCheckpointing，用户设置的Commit Offset策略不生效。只有开启checkpoint后，在每次checkpoint成功时，才会commit当前分区消费的offset至Kafka，以便在作业失败恢复过程中，从上一次checkpoint的commit位点开始消费，保证计算的Exactly Once。如果将checkpoint间隔设置过大，Kafka端可能会查询不到当前消费offset。

4.6.2.7. 创建表格存储Tablestore源表

本文为您介绍如何创建实时计算Flink版表格存储Tablestore源表。

> **注意** 本文仅适用于实时计算Flink版3.2.2及以上版本。

什么是表格存储Tablestore

表格存储Tablestore是构建在阿里云飞天分布式系统之上的分布式NoSQL数据存储服务。表格存储通过数据分片和负载均衡技术，实现数据规模与访问并发上的无缝扩展，提供海量结构化数据的存储和实时访问服务。

表格存储通道服务

表格存储通道服务是基于表格存储数据接口的全增量一体化服务，通过Tunnel Service API和SDK，为您提供了增量、全量和增量加全量三种类型的分布式数据实时消费通道。通过Tunnel Service数据通道，您可以使用流式计算的方式，消费表中存量或新增数据。实时计算Flink版可以将Tunnel Service数据通道作为流式数据的输入，每条数据类似一个JSON格式。Tunnel Service数据通道的示例如下。

```
{
            "OtsRecordType": "PUT",  // 数据操作类型，包括PUT、UPDATE和DELETE。
            "OtsRecordTimestamp": 1506416585740836, //数据写入时间（单位为微秒），全量数据时为0
。
            "PrimaryKey": [
                {
                    "ColumnName": "pk_1", //第1主键列。
                    "Value": 1506416585881590900
                },
                {
                    "ColumnName": "pk_2", //第2主键列。
                    "Value": "string_pk_value"
                }
            ],
            "Columns": [
                {
                    "OtsColumnType": "PUT", // 列操作类型，包括PUT、DELETE_ONE_VERSION和DELETE
_ALL_VERSION。
                    "ColumnName": "attr_0",
                    "Value": "hello_table_store",
                },
                {
                    "OtsColumnType": "DELETE_ONE_VERSION", // DELETE操作没有Value字段。
                    "ColumnName": "attr_1"
                }
            ]
}
```

DDL定义

实时计算Flink版支持使用Tablestore作为源表，示例代码如下。

```
create table tablestore_stream(
  pk_1 BIGINT,
  pk_2 VARCHAR,
  attr_0 VARCHAR,
  attr_1 DOUBLE,
  OtsRecordType  VARCHAR  HEADER //属性字段后边需要增加HEADER。
) with (
  type ='ots',
  endPoint ='http://blink-demo.cn-hangzhou.vpc.tablestore.aliyuncs.com',
  instanceName = 'blink-demo',
  tableName ='demo_table',
  tunnelName = 'blink-demo-stream',
  accessId ='<yourAccessID>',
  accessKey ='<yourAccessSecret>',
  ignoreDelete = 'false' //是否忽略DELETE操作的数据。
);
```

属性字段

表格存储源表属性字段的获取和使用方法，请参见获取数据源表属性字段。

字段名	说明
`OtsRecordType`	数据操作类型
`OtsRecordTimestamp`	数据操作时间（全量数据时，`OtsRecordTimestamp`为0）
`<列名>_OtsColumnType`	某列的操作类型

WITH参数

参数	说明	备注
type	connector类型	固定值为`ots`。
endPoint	表格存储的实例访问地址	VPC网络环境需要选择实例的VPC Endpoint。
instanceName	表格存储的实例名称	无
tableName	表格存储的数据表名	实时计算读取Tablestore源表数据时，已读取的数据不会再被读取，如果您有再次读取全量数据的需求，则需要重新创建新的数据通道。
tunnelName	表格存储数据表的数据通道名	无
accessId	表格存储读取的AccessKey ID	无
accessKey	表格存储读取的密钥AccessKey Secret	无
ignoreDelete	是否忽略DELETE操作的数据。	可选，默认值为`false`。

4.6.2.8. 创建全量MaxCompute源表

本文为您介绍如何创建全量MaxCompute源表，以及创建源表时使用的WITH参数和类型映射。

> **注意**
> - 本文仅适用于Blink 2.2.7及以上版本。
> - 与数据总线DataHub、Kafka等数据源不同，全量MaxCompute源表通常作为有限流表使用。Blink 3.4.4版本临时支持将全量MaxCompute源表作为无限流表使用，实现不间断监听并读取新增分区的功能。Blink 3.5.0版本废弃该功能，如果您需要使用无限流MaxCompute源表，请参见创建增量MaxCompute源表。

DDL定义

实时计算Flink版支持使用全量MaxCompute作为源表输入，示例代码如下。

```
create table odps_source(
  id INT,
  user_name VARCHAR,
  content VARCHAR
) with (
  type = 'odps',
  endPoint = 'http://service.cn.maxcompute.aliyun-inc.com/api',
  project = '<projectName>',
  tableName = '<tableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****' --如果您的MaxCompute源表为非分区表，不声明该参数即可。
);
```

WITH参数

参数	说明	是否必填	备注
endPoint	MaxCompute服务地址。	是	请参见Endpoint。
tunnelEndpoint	MaxCompute Tunnel服务的连接地址。	否	请参见Endpoint。 **说明** VPC环境下为必填。
project	MaxCompute项目名称。	是	无。
tableName	MaxCompute表名。	是	无。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。
partition	分区名。	否	• 只存在一级分区的全量MaxCompute表 例如，如果只存在1个分区列 `ds` ，则 `` `partition` = 'ds=20180905' `` 表示读 `ds=20180905` 分区的数据。 • 存在多级分区的全量MaxCompute表 例如，如果存在2个分区列 `ds` 和 `hh` ，则 `` `partition`='ds=20180905,hh=*' `` 表示读 `ds=20180905` 分区的数据。 **说明** 分区过滤时需要声明所有分区的值。例如，上述示例中，只声明 `` `partition` = 'ds=20180905' `` ，则不会读取任何分区。

参数	说明	是否必填	备注
subscribeNewPartition	是否监听符合条件的新分区。	否	默认值为false，不监听新产生的分区。 **说明** • subscribeNewPartition参数值为 `true` 时，不可以指定partition的参数值，否则会造成无法读取新产生的分区的状况。 • 该参数只在Blink 3.4.4版本临时存在，Blink 3.5.0版本废弃该参数，请使用创建增量MaxCompute源表。
subscribeIntervalInSec	监听新分区的间隔。	否	默认值为30，单位为秒。 **说明** 监听间隔设置太小，会对全量MaxCompute MetaData服务造成压力，有可能导致监听服务失败。
maxPartitionCount	未设置Partition参数时，读取当前分区表的分区个数。	否	默认值为100。 **说明** 仅Blink 3.0及以上版本支持该参数。

类型映射

全量MaxCompute字段类型	实时计算Flink版字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR

全量MaxCompute字段类型	实时计算Flink版字段类型
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

代码示例

包含全量MaxCompute源表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE odps_source (
  cid varchar,
  rt DOUBLE,
) with (
  type = 'odps',
  endPoint = '<yourEndpointName>',
  project = '<yourProjectName>',
  tableName = '<yourTableName>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessPassword>',
  partition = 'ds=20190712'
);
CREATE TABLE test (
  cid varchar,
  invoke_count BIGINT
) with (
  type='print'
);
INSERT INTO test
SELECT
  cid,
  count(*) as invoke_count
FROM odps_source GROUP BY cid;
```

常见问题

- Q：DDL中endPoint和tunnelEndpoint参数配置错误，有什么影响？

  A：endPoint和tunnelEndpoint参数介绍请参见各地域Endpoint对照表（外网连接方式）。参数配置错误会导致以下问题：

  - endPoint配置错误：任务上线进度停滞在91%处。
  - tunnelEndpoint配置错误：任务运行失败。

- Q：全量MaxCompute数据存储底层是如何读取全量MaxCompute源表的？

  A：全量MaxCompute数据存储通过Tunnel读取全量MaxCompute数据的，读取速度及带宽受全量MaxCompute Tunnel带宽限制。

- Q：实时计算Flink版作业启动后，全量MaxCompute源表是否能读取新写入到全量MaxCompute表或者全量MaxCompute分区的数据？

A：全量MaxCompute数据存储使用Tunnel读取表数据或者分区数据。实时计算Flink版作业启动后，读取完成Reader后，不再读取新写入全量MaxCompute表和分区的数据。

- Q：实时计算Flink版作业启动后，全量MaxCompute源表如何能够读取到全量MaxCompute表或者全量MaxCompute分区的数据？

  A：实时计算Flink版 3.4及以上版本支持subscribeNewPartition参数，控制是否需要监听新产生的分区。新产生的数据可以写入到新的分区，示例代码如下。

```
CREATE TABLE blink_source (
    cid varchar,
    rt DOUBLE,
) with (
    type = 'odps',
    endPoint = '<yourEndpointName>',
    project = '<yourProjectName>',
    tableName = '<yourTableName>',_table_name',
    subscribeNewPartition = 'true'
    --请注意：如果需要监听新的分区，则不能指定partition值。
    accessId = '<yourAccessKeyId>',
    accessKey = '<yourAccessKeySecret>',
);
CREATE TABLE test (
    cid varchar,
    invoke_count BIGINT
) with (
  type='print'
);
INSERT INTO test
SELECT
    cid,
    count(*) as invoke_count
FROM blink_source GROUP BY cid;
```

> **说明** 全量MaxCompute表新分区产生的数据需要写入实时计算Flink版全量MaxCompute源表的新分区。如果写入现有分区，则不生效。

- Q：WITH参数中partition值是否可以使用 `max_pt()` 或者 `max_pt_with_done()` ？

  A：不推荐使用。如果需要使用，请务必了解在没有开启监听新分区和开启监听新分区的场景下 `max_pt()` 在全量MaxCompute源表的作用：

  - 未开启监听新分区

    任务启动后，会通过全量MaxCompute MetaData服务获取当前全量MaxCompute 源表所有分区列表，从中读取 `max_pt()` 。读取结束后，Reader退出。不再读取该分区中的新数据或者监听新分区。

  - 开启监听新分区

    任务启动后，会通过全量MaxCompute MetaData服务获取当前全量MaxCompute源表所有分区列表，从中读取 `max_pt()` 。读取结束后，不再读取该分区中的新数据。间隔一定周期（参见subscribeIntervalInSec参数）监听是否有新分区产生。如果有新分区产生，读取新产生的分区列表，从中获取 `max_pt()` ，并读取。读取完成后还会等待下次监听事件。如果没有新分区产生，等待下次监听事件。

- Q：引用全量MaxCompute作为数据源，在作业启动后，向已有的分区或者表里追加数据，这些新数据是否能被读取？

  A：数据不会被读取，而且可能导致作业失败。全量MaxCompute数据存储使用 `ODPS DOWNLOAD SESSION` 读取表数据或者分区数据。新建 `DOWNLOAD SESSION` ，服务端会创建一个Index文件，相当于创建 `DOWNLOAD SESSION` 那一瞬间数据的映射，后续的数据读取以这个映射为基础。因此在新建 `DOWNLOAD SESSION` 后，追加到全量MaxCompute表或者分区里的数据，正常流程下是不会被读取的。分为两种情况：

  - Tunnel在读取数据的过程中，写入新数据会产生报错 `ErrorCode=TableModified,ErrorMessage=The specified table has been modified since the download initiated.` 。
  - Tunnel已经关闭，写入新数据。这些数据不会被读取的，因为Tunnel已经关闭。而且一旦作业发生Failover或者暂停恢复作业后，不能保证数据正确性，例如，已经读过的数据可能会重读，新追加的数据可能读不全。

- Q：全量MaxCompute源表作业是否支持暂停、恢复和修改源表的并发度？

  A：全量MaxCompute源表作业不支持暂停、恢复和修改源表并发度。系统根据并发度，计算每个并发应该读哪些分区的哪一段内容，并且每个并发会把自己消费的情况记录到State里。以便暂停恢复后或者Failover后，可以从上次读取的位置继续读取。这个逻辑是建立在并发度确定的前提下的。如果修改了全量MaxCompute源表的并发度后进行暂停恢复操作，对作业产生的影响是无法预估的，因为已经读取的数据可能会再次读取，没有读的数据反而被遗漏。

- Q：作业启动位点设置了 `2019-10-11 00:00:00` ，为什么启动位点前的分区也会被读取？

  A：设置启动位点只对消息队列类型的数据源有效（例如DataHub），对全量MaxCompute源表无效。实时计算Flink版作业启动后数据读取的范围如下：

  - 分区表：读取当前所有分区。
  - 非分区表：读取当前存在的数据。

- Q：作业运行过程中报错ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

  A：该报错是因为MaxCompute DDL定义中填写的用户身份信息无法访问MaxCompute。因此，您需要通过阿里云账号、RAM用户账号或RAM角色认证用户身份，详情请参见用户认证。

  如果您有其他疑问，请提交工单咨询，产品名称选择为MaxCompute。

4.6.2.9. 创建增量MaxCompute源表

本文为您介绍如何创建增量MaxCompute源表，以及创建维表时使用的WITH参数、类型映射和常见问题。

> **注意**
> - 本文仅适用于Blink 3.5.0 hotfix及以上版本。
> - 增量MaxCompute源表不支持作为维表使用。
> - 增量MaxCompute源表只支持MaxCompute分区表，不支持非分区表。

语法示例

```
create table odps_source(
    id int,
    user_name VARCHAR,
    content VARCHAR
) with (
    type = 'continuous-odps',
    endPoint = 'your_end_point_name',
    project = 'your_project_name',
    tableName = 'your_table_name',
    accessId = 'your_access_id',
    accessKey = 'your_access_key',
    startPartition = 'ds=20180905'
);
```

WITH参数

参数	说明	是否必填	备注
type	connector类型	是	固定值为 `continuous-odps` 。
endPoint	MaxCompute服务本身的连接地址	是	请参见Endpoint。
tunnelEndpoint	MaxCompute Tunnel服务的连接地址	• 是：VPC环境必填 • 否：其他非VPC环境	请参见Endpoint。
project	表所属的project名称	是	无
tableName	表名	是	无
accessId	AccessKey ID	是	无
accessKey	AccessKey Secret	是	无

参数	说明	是否必填	备注
startPartition	指定读取的起始分区。系统加载分区列表时，会把每个分区列表的所有分区和startPartition按照字母顺序进行比较，加载满足条件的分区的数据。 此外，增量MaxCompute源表可以持续监听增量MaxCompute分区表。读完已有的分区后，任务不会退出，且持续监听并读入新分区。 说明 • 增量MaxCompute源表中必须存在一级分区，二级分区可选。 • 如果指定二级分区，必须写在一级分区的后面。 • 如果startPartition指定的分区不存在，系统会从下一个分区开始读取数据。	是	例如，指定startPartition是ds=20191201，表示加载增量MaxCompute表里所有满足ds >= 20191201的分区数据。 如果一个增量MaxCompute分区表，有一级分区ds和二级分区type两个分区列，假设表里有以下5个分区： • ds=20191201,type=a • ds=20191201,type=b • ds=20191202,type=a • ds=20191202,type=b • ds=20191202,type=c 不同startPartition，满足分区的列表如下： • ds=20191202 ◦ ds=20191202,type=a ◦ ds=20191202,type=b ◦ ds=20191202,type=c • ds=20191201,type=c ◦ ds=20191202,type=a ◦ ds=20191202,type=b ◦ ds=20191202,type=c

类型映射

MaxCompute字段类型	实时计算Flink版字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP

MaxCompute字段类型	实时计算Flink版字段类型
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

> **注意**
> - 增量MaxCompute源表暂不支持CHAR、VARCHAR、ARRAY、MAP和STRUCT数据类型。
> - 您可以临时使用STRING替换VARCHAR。

代码示例

增量MaxCompute源表每天产生一个分区，分区列是ds，从ds=20191201分区开始，加载后续的所有分区，并一直监听新分区的产生。

```
--读增量MaxCompute表，读取的分区范围是[ds=20191201，∞)。
CREATE TABLE odps_source (
    cid VARCHAR,
    rt DOUBLE,
) with (
    type = 'continuous-odps',
    endPoint = 'your_end_point_name',
    project = 'your_project_name',
    tableName = 'your_table_name',
    accessId = 'xxxx',
    accessKey = 'xxxx',
    startPartition = 'ds=20191201'
);
CREATE TABLE test (
    cid VARCHAR,
    rt DOUBLE,
) with (
    type='print'
);
INSERT INTO test
SELECT
    cid, rt FROM odps_source;
```

常见问题

- Q：DDL中endPoint和tunnelEndpoint参数配置错误的影响？

  A：endPoint和tunnelEndpoint参数介绍请参见各地域Endpoint对照表（外网连接方式）。参数配置错误会导致以下问题：

  - endPoint配置错误：任务上线进度停滞在91%处。
  - tunnelEndpoint配置错误：任务运行失败。

- Q：增量MaxCompute数据存储底层是如何读取MaxCompute表的？

A：增量MaxCompute数据存储是通过tunnel读取MaxCompute数据，读取速度及带宽受MaxCompute tunnel带宽限制。

- Q：实时计算Flink版作业启动后，增量MaxCompute源表是否能读取新写入到已经读取过的分区的数据？

  A：增量MaxCompute数据存储使用Tunnel读取MaxCompute源表分区的数据。实时计算Flink版作业启动后，读取完成Reader就退出，不会读取新写入MaxCompute源表已读取分区的数据。

- Q：如何查看MaxCompute分区名？

  A：您可以在**数据表详情**中查看MaxCompute分区名，步骤如下：

  i. 在数据地图，搜索表名称。

  ii. 在**所有表**区域，单击目标表名。

  iii. 在**数据表详情**页面，右侧**明细信息 > 分区信息 > 分区名**查看MaxCompute分区名。

- Q：引用增量MaxCompute作为数据源，在作业启动后，向已有的分区或者表里追加数据，这些新数据是否能被读取？

  A：数据不会被读取，而且可能导致作业失败。MaxCompute数据存储使用 `ODPS DOWNLOAD SESSION` 读取表数据或者分区数据。新建 `DOWNLOAD SESSION` ，服务端会创建一个Index文件，相当于创建 `DOWNLOAD SESSION` 那一瞬间数据的映射，后续的数据读取以这个映射为基础。因此在新建 `DOWNLOAD SESSION` 后，追加到MaxCompute表或者分区里的数据，正常流程下是不会被读取的。分为两种情况：

  - Tunnel在读取数据的过程中，写入新数据会产生报错 `ErrorCode=TableModified,ErrorMessage=The specified table has been modified since the download initiated.` 。
  - Tunnel已经关闭，写入新数据。这些数据不会被读取的，因为Tunnel已经关闭。而且一旦作业发生Failover或者暂停恢复作业后，不能保证数据正确性（例如已经读过的数据可能会重读，新追加的数据可能读不全）。

- Q：增量MaxCompute源表作业是否支持暂停、恢复和修改源表的并发度？

  增量MaxCompute源表作业暂不支持暂停、恢复和修改源表并发度。系统根据并发度，计算每个并发读取指定分区的指定数据。此外，每个并发会把消费情况记录至State，以便暂停恢复后或者Failover后，系统可以从上次读取的位置继续读取数据。

  如果修改了增量MaxCompute源表的并发度后暂停恢复作业，会对作业产生无法预估的影响。因为已经读取的数据可能会被再次读取，没有读的数据可能会被遗漏。

- Q：监听到新分区的时候，如果该分区还有数据没有写完，如何处理？

  A：没有机制可以标志一个分区的数据是否已经完整。目前只要监听到新分区，就会读入。用增量MaxCompute源表读取一个MaxCompute分区表T，分区列是ds，表T的写入过程分为以下两种方式：

  - 不允许的写入方法：先创建好分区，例如ds=20191010，再往分区里写数据。增量MaxCompute源表监听到新分区ds=20191010，立刻读入，如果此时该分区还有数据没有写完，就会漏读数据。
  - 推荐的写入方法：不创建分区，先执行Insert overwrite table T partition (ds='20191010') ...语句，作业结束且成功后，分区和数据一起可见。

- Q：作业运行过程中报错ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

  A：该报错是因为MaxCompute DDL定义中填写的用户身份信息无法访问MaxCompute。因此，您需要通过阿里云账号、RAM用户账号或RAM角色认证用户身份，详情请参见用户认证。

  如果您有其他疑问，请提交工单咨询，产品名称选择为MaxCompute。

4.6.3. 创建数据结果表

4.6.3.1. 数据结果表概述

实时计算使用 `CREATE TABLE` 作为输出结果数据的格式定义，同时定义数据如何写入到目的数据结果表。

实时计算结果表有Append类型和Update类型。

- Append类型：如果输出存储是日志系统、消息系统、或未定义主键的RDS数据库，数据流的输出结果会以追加的方式写入到存储中，不会修改存储中原有的数据。
- Update类型：如果输出存储是声明了主键（primary key）的数据库（例如，RDS和HBase），数据流的输出结果有以下2种情况。
  - 如果根据主键查询的数据在数据库中不存在，则会将该数据插入数据库。
  - 如果根据主键查询的数据在数据库中存在，则会根据主键更新数据。

结果表语法

```
CREATE TABLE tableName
    (columnName dataType [, columnName dataType ]*)
    [ WITH (propertyName=propertyValue [, propertyName=propertyValue ]*) ];
```

结果表示例

```
CREATE TABLE rds_output(
id INT,
len INT,
content VARCHAR,
PRIMARY KEY(id)
) WITH (
type='rds',
url='yourDatabaseURL',
tableName='yourTableName',
userName='yourDatabaseUserName',
password='yourDatabasePassword'
);
```

4.6.3.2. 创建Oracle数据库结果表

本文为您介绍如何创建Oracle结果表，以及创建结果表时使用的WITH参数、类型映射和注意事项。

> **注意**
>
> - 本文仅适用于Blink 3.6.0及以上版本。
> - 仅支持使用Oracle 19c版本创建Oracle数据库结果表。

注意事项

- 实时计算Flink版将每行结果数据拼接成一行SQL语句写入目标数据库。
- 根据数据库中是否存在需要的表，执行以下操作：
  - 当数据库中不存在需要的表时，Oracle结果表会新建一个用于写入结果的表。
  - 当数据库中存在需要的表时，Oracle结果表会向该表中写入或者更新数据。

- 逻辑表和物理表的主键不一定完全相同，要求逻辑表的主键包含物理表的主键。
- 使用以下两种函数处理Oracle数据库结果表的数据：
  - 如果没有定义主键，则执行Append函数向表中插入数据。
  - 如果已经定义主键，则执行Upsert函数向表中插入数据。当主键不存在时，则插入主键；当主键存在时，则更新主键。

语法示例

实时计算Flink版支持使用Oracle数据库作为结果表，代码示例如下。

```
CREATE TABLE oracle_sink(
  employee_id BIGINT,
  employee_name VARCHAR,
  employee_age INT,
  PRIMARY KEY(employee_id)
) WITH (
    type = 'oracle',
    url = '<yourUrl>',
    userName = '<yourUserName>',
    password = '<yourPassword>',
    tableName = '<yourTableName>'
);
```

WITH 参数

参数	描述	是否必选	示例值
type	结果表类型	是	固定值为*oracle*。
url	数据库连接串	是	`jdbc:oracle:thin:@192.168.171.62:1521:sit0`
userName	登录数据库的用户名	是	无
password	登录数据库的密码	是	无
tableName	数据库的表名	是	无
maxRetryTimes	向结果表插入数据的最大尝试次数	否	默认值为10。
batchSize	单次写入数据的批次大小 **说明** - 指定主键后batchSize参数才生效。 - batchSize或bufferSize参数达到阈值时都会触发数据写入。	否	默认值为50。

参数	描述	是否必选	示例值
bufferSize	去重的缓存大小 **说明** • 指定主键后bufferSize参数才生效。 • batchSize或bufferSize参数达到阈值时都会触发数据写入。	否	默认值为500。
flushIntervalMs	写超时时间，单位为毫秒。 **说明** 如果在写超时时间内没有向数据库写入数据，系统会将缓存的数据再写一次。	否	默认值为500。
excludeUpdateColumns	更新表的某行数据时，是否不更新该行指定的列数据。 **说明** 更新表的某行数据时，默认不更新主键数据。	否	默认不填写。
ignoreDelete	是否忽略删除操作。	否	默认值为false。

类型映射

Oracle 字段类型	实时计算Flink版字段类型
• CHAR • VARCHAR • VARCHAR2	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

代码示例

实时计算Flink版包含Oracle数据库结果表的代码示例如下。

```
CREATE TABLE oracle_source (
  employee_id BIGINT,
  employee_name VARCHAR,
  employ_age INT
) WITH (
  type = 'random'
);
CREATE TABLE oracle_sink(
  employee_id BIGINT,
  employee_name VARCHAR,
  employ_age INT,
  primary key(employee_id)
)with(
  type = 'oracle',
  url = 'jdbc:oracle:thin:@192.168.171.62:1521:sit0',
  userName = 'blink_test',
  password = 'blink_test',
  tableName = 'oracle_sink'
);
INSERT INTO oracle_sink
SELECT * FROM oracle_source;
```

4.6.3.3. 创建交互式分析Hologres结果表

本文为您介绍如何创建交互式分析Hologres结果表，以及创建结果表时使用的WITH参数、流式语义和类型映射。

> **注意**
>
> - 本文仅适用于Blink 3.6.0及以上版本，如果您的Blink为3.6.0以下的版本，您可以：
>   - 升级Blink版本至3.6.0及以上版本，详细请参见管理独享集群Blink版本。
>   - 提交工单获取需要的JAR文件，安装使用。
> - 建议您使用Hologres 0.7及以上版本。
> - 由于Hologres是异步写入数据的，因此需要添加blink.checkpoint.fail_on_checkpoint_error=true 作业参数，作业异常时才会触发Failover。

什么是交互式分析Hologres

交互式分析Hologres兼容PostgreSQL协议，与大数据生态紧密连接，支持高并发、低延时实时分析处理PB级数据，让您轻松使用现有BI（Business Intelligence）工具对数据进行多维分析和业务探索。

语法示例

实时计算Flink版支持使用Hologres作为结果表，代码示例如下。

```
create table Hologres_sink(
  name varchar,
  age BIGINT,
  birthday BIGINT
) with (
  type='hologres',
  dbname='<yourDbname>',
  tablename='<yourTablename>',
  username='<yourUsername>',
  password='<yourPassword>',
  endpoint='<yourEndpoint>',
  field_delimiter='|' --该参数可选。
);
```

WITH参数

参数	说明	是否必填	备注
type	结果表类型	是	固定值为*hologres*。
dbname	数据库名称 说明 如果Schema不为Public时，则tableName需要填写为schema.tableName。	是	无
tablename	表名称	是	无
username	用户名	是	无
password	密码	是	无
endpoint	Hologres VPC 端点信息	是	详情请参见访问域名。
field_delimiter	导出数据时，不同行之间使用的分隔符。 注意 不能在数据中插入分隔符，且需要与bulkload语义一同使用。	否	默认值为 *"\u0002"*。
mutateType	流式写入语义，详情请参见流式语义。	否	默认值为*insertorignore*。

参数	说明	是否必填	备注
partitionrouter	分区表写入	否	默认值为*false*。 **说明** 3.6.x版本Hologres Sink暂不支持自动创建分区表。因此，在写入分区表之前，需要您在Hologres中手工创建对应的子表。
ignoredelete	是否忽略撤回消息。	否	默认值为*false*。 **说明** 仅在使用流式语义时生效。
createPartTable	当写入分区表时，是否根据分区值自动创建不存在的分区表。 **说明** 如果分区值中存在短划线（-），暂不支持自动创建分区表。	否	• false（默认值）：不会自动创建。 • true：自动创建。 **说明** 仅Blink 3.7以上版本支持该参数。

流式语义

流处理，也称为流数据或流事件处理，即对一系列无界数据或事件连续处理。执行流数据或流事件处理的系统通常允许您指定一种可靠性模式或处理语义，保证整个系统处理数据的准确性，因为网络或设备故障等可能会导致数据丢失。

根据Hologres Sink的配置和Hologres表的属性，流式语义分为以下两种：

- Exactly-once（仅一次）：即使在发生各种故障的情况下，系统只处理一次数据或事件。
- At-least-once（至少一次）：如果在系统完全处理之前丢失了数据或事件，则从源头重新传输，因此可以多次处理数据或事件。如果第一次重试成功，则不必进行后续重试。

在Hologres结果表中使用流式语义，您需要注意以下几点：

- 如果Hologres物理表未设置主键，则Hologres Sink使用At-least-once语义。
- 如果Hologres物理表已设置主键，则Hologres Sink通过主键确保Exactly-once语义。当同主键数据出现多次时，您需要设置mutateType参数确定更新结果表的方式，mutateType取值如下：
  - insertorignore（默认值）：保留首次出现的数据，忽略后续所有数据。
  - insertorreplace：使用后续出现的数据整行替换已有数据。
  - insertorupdate：使用后续出现的数据选择性替换已有数据。

  > **说明**
  > - 当mutateType设置为insertorupdate或insertorreplace时，系统根据主键更新数据。
  > - Blink定义的结果表中的数据列数不一定要和Hologres物理表的列数一致，您需要保证缺失的列没有非空约束，即列值可以为Null，否则会报错。

- 默认情况下，Hologres Sink只能向一张表导入数据。如果导入数据至分区表的父表，即使导入成功，也会查询数据失败。您可以设置参数partitionRouter为true，开启自动将数据路由到对应分区表的功能。注意事项如下：
  - tablename参数需要填写为父表的表名。
  - Blink Connector不会自动创建分区表，因此，需要您提前手动创建需要导入数据的分区表，否则会导入失败。

宽表Merge和局部更新功能

在把多个流的数据写到一张Hologres宽表的场景中，会涉及到宽表Merge和数据的局部更新。下面通过一个示例来介绍如何设置。

假设有两个Flink数据流，一个数据流中包含A、B和C字段，另一个数据流中包含A、D和E字段，Hologres宽表WIDE_TABLE包含A、B、C、D和E字段，其中A字段为主键。具体操作如下：

1. 使用Flink SQL创建两张Hologres结果表，其中一张表只声明A、B和C字段，另一张表只声明A、D和E字段。这两张表都映射至宽表WIDE_TABLE。
2. 两张结果表的属性设置：
   - mutatetype设置为insertorupdate，可以根据主键更新数据。
   - ignoredelete设置为true，防止回撤消息产生Delete请求。
3. 将两个Flink数据流的数据分别INSERT至对应的结果表中。

> **说明** 在上述场景中，有如下限制：
> - 宽表必须有主键。
> - 每个数据流的数据都必须包含完整的主键字段。
> - 列存模式的宽表Merge场景在高RPS的情况下，CPU使用率会偏高，建议关闭表中字段的Dictionary encoding功能。

类型映射

Hologres	BLINK
INT	INT

Hologres	BLINK
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

4.6.3.4. 创建云原生数据仓库AnalyticDB MySQL版2.0结果表

本文为您介绍如何创建云原生数据仓库AnalyticDB MySQL版2.0结果表，以及云原生数据仓库AnalyticDB MySQL版和实时计算Flink版字段类型之间的映射关系。

> **注意**
> - 本文仅适用于Blink 1.4.5及以上版本。
> - 云原生数据仓库AnalyticDB MySQL版2.0数据库支持自增主键。如果实时计算Flink版写入数据支持自增主键，则在DDL中不声明该自增字段。例如，ID是自增字段，实时计算Flink版DDL不声明该自增字段，则数据库在一行数据写入过程中会自动填补相关自增字段。

什么是云原生数据仓库AnalyticDB MySQL版

云原生数据仓库AnalyticDB MySQL版是阿里巴巴自主研发的海量数据实时高并发在线分析（Realtime OLAP）云计算服务，支持在毫秒级单位时间内，对千亿级数据进行实时地多维分析透视和业务探索。

DDL定义

> **说明** 云原生数据仓库AnalyticDB MySQL版3.0结果表DDL请参见创建云原生数据仓库AnalyticDB MySQL版3.0结果表。

实时计算Flink版支持使用云原生数据仓库AnalyticDB MySQL版2.0作为结果输出。示例代码如下。

```
CREATE TABLE stream_test_hotline_agent (
id INTEGER,
len BIGINT,
content VARCHAR,
PRIMARY KEY(id)
) WITH (
type='ads',
url='yourDatabaseURL',
tableName='<yourDatabaseTableName>',
userName='<yourDatabaseUserName>',
password='<yourDatabasePassword>',
batchSize='20'
);
```

> **说明**
> - 建议使用存储注册功能，详情请参见注册云原生数据仓库AnalyticDB MySQL版。
> - 云原生数据仓库AnalyticDB MySQL版结果表声明中的主键要和云原生数据仓库AnalyticDB MySQL版里的主键保持一致，大小写敏感。如果不一致，则会导致数组索引越界的异常现象。

WITH参数

参数	说明	备注
type	结果表类型	固定值为ads。
url	JDBC连接地址	云原生数据仓库AnalyticDB MySQL版数据库地址。示例：`url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'`。其中参数解释如下： • URI地址查询步骤如下： i. 登录AnalyticDB控制台。 ii. 单击对应的集群名称，进入**基本信息**页面。 iii. 在**连接信息**中查看相应的连接地址。 • 云原生数据仓库AnalyticDB MySQL版数据库名称（databaseName）即云原生数据仓库AnalyticDB MySQL版实例名称。
tableName	表名	无
username	账号	无
password	密码	无
maxRetryTimes	写入重试次数	可选，默认值为10。
bufferSize	流入多少条数据后开始去重	可选，默认值为5000，表示输入的数据达到5000条就开始输出。

参数	说明	备注
batchSize	一次批量写入的条数	可选，默认值为1000。 **说明** 如果错误码是20015，则表示batchSize设置过大。云原生数据仓库AnalyticDB MySQL版batchSize不能超过1 MB。例如，batchSize设置为 `1000` ，则平均每条记录大小不能超过1 KB。
batchWriteTimeoutMs	写入超时时间	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
connectionMaxActive	单连接池最大连接数	可选，默认值为30。
ignoreDelete	是否忽略delete操作	默认值为false。

类型映射

建议使用云原生数据仓库AnalyticDB MySQL版和实时计算Flink版字段类型对应关系进行DDL声明。

云原生数据仓库AnalyticDB MySQL版字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
TINYINT	INT
SMALLINT	
INT	
BIGINT	BIGINT
DOUBEL	DOUBLE
VARCHAR	VARCHAR
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP

4.6.3.5. 创建日志服务SLS结果表

本文为您介绍如何创建实时计算Flink版日志服务SLS结果表。

> **注意**
> - 本文仅适用于Blink 1.4.5及以上版本。
> - 日志服务SLS结果表仅支持VARCHAR类型的字段。

什么是日志服务

日志服务SLS是针对日志类数据的一站式服务。日志服务可以帮助您快捷地完成数据采集、消费、投递以及查询分析，提升运维和运营效率，建立海量日志处理能力。日志服务本身是流数据存储，实时计算Flink版能将其作为流式数据的输入。

DDL定义

实时计算Flink版支持使用日志服务作为结果输出。日志服务结果表声明示例如下。

```
create table sls_stream(
 `name` VARCHAR,
 age BIGINT,
 birthday BIGINT
)with(
 type='sls',
 endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
 project='<yourProjectName>',
 logstore='<yourLogstoreName>'
);
```

> **说明** 建议使用日志服务存储注册功能，详情请参见注册日志服务SLS。

WITH参数

参数	注释说明	是否必填	备注
endPoint	EndPoint地址	是	服务入口
project	项目名	是	无
logstore	表名	是	无
accessId	AccessKey ID	是	无
accessKey	AccessKey Secret	是	无
topic	属性字段	否	默认值为空，可以将选定的字段作为属性字段 `topic` 填充。

参数	注释说明	是否必填	备注
timestampColumn	属性字段	否	默认值为空，可以将选定的字段作为属性字段 `timestamp` 填充（类型必须为INT）。如果未指定，则默认填充当前时间。 **说明** 实时计算Flink版2.2.2及以上版本支持该参数。
source	属性字段。日志的来源地，例如产生该日志机器的IP地址。	否	默认值为空，可以将选定的字段作为属性字段 `source` 填充。
partitionColumn	分区列	否	如果 `mode` 为 `partition` ，则该参数必填。
flushIntervalMs	触发数据写入的周期	否	默认值为2000，单位为毫秒。
reserveMilliSecond	TIMESTAMP类型是否保留毫秒值。	否	默认值为false，不保留。 **说明** 实时计算Flink版2.2.6及以上版本支持该参数。

类型映射

日志服务和实时计算Flink版字段类型对应关系如下。建议您使用该对应关系进行DDL声明。

日志服务字段类型	实时计算Flink版字段类型
STRING	VARCHAR

代码示例

包含日志服务SLS结果表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE random_input (
  a VARCHAR,
  b VARCHAR) with (
    type = 'random'
);
create table sls_output(
 a varchar,
 b varchar
)with(
 type='sls',
 endPoint='http://cn-hangzhou-corp.sls.aliyuncs.com',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>',
 project='ali-cloud-streamtest',
 logStore='stream-test2'
);
INSERT INTO sls_output
SELECT a, b
FROM random_input;
```

常见问题

Q：如何配置输出结果表中的topic?

A：需要配置topic为结果表中的一个字段。例如，本文示例代码中设置 `topic='age'` 。配置完成后，结果值中 `age` 字段的值会写入日志服务，同时日志服务不会把 `age` 字段写入下游。

相关文档

- 日志服务帮助文档，请参见什么是日志服务。
- 日志服务中实时计算消费文档，请参见实时计算（Blink）消费。

4.6.3.6. 创建消息队列MQ结果表

本文为您介绍如何创建实时计算Flink版消息队列MQ结果表，以及创建过程涉及到的WITH参数。

> **注意**
>
> - 本文仅适用于Blink 1.4.5及以上版本。
> - 如果您需要使用带独立命名空间的MQ，请使用Blink 3.x作业版本。

什么是消息列队MQ

消息队列MQ（Message Queue）是阿里云商用的专业消息中间件，是企业级互联网架构的核心产品。消息列队基于高可用分布式集群技术，搭建了包括发布订阅、消息轨迹、资源统计、定时（延时）和监控报警等一套完整的消息云服务。

CSV格式

实时计算Flink版可以将消息队列作为流式数据进行输出，CSV格式输出示例如下。

```
CREATE TABLE stream_test_hotline_agent (
id INTEGER,
len BIGINT,
content VARCHAR
) WITH (
type='mq',
endpoint='<yourEndpoint>',
accessID='<yourAccessId>',
accessKey='<yourAccessSecret>',
topic='<yourTopicName>',
producerGroup='<yourGroupName>',
tag='<yourTagName>',
encoding='utf-8',
fieldDelimiter=',',
retryTimes='5',
sleepTimeMs='500'
);
```

二进制格式

实时计算Flink版可以将消息队列作为流式数据进行输出，二进制格式输出示例如下。

```
CREATE TABLE source_table (
  commodity VARCHAR
)WITH(
  type='random'
);
CREATE TABLE result_table (
  mess VARBINARY
) WITH (
  type = 'mq',
  endpoint='<yourEndpoint>',
  accessID='<yourAccessId>',
  accessKey='<yourAccessSecret>',
  topic='<yourTopicName>',
  producerGroup='<yourGroupName>'
);
INSERT INTO result_table
SELECT
CAST(SUBSTRING(commodity,0,5) AS VARBINARY) AS mess
FROM source_table;
```

> **说明** `cast(varchar as varbinary)` 需要在Blink 2.0及以上版本使用。如果版本低于2.0，请先完成版本升级，详情请参见管理独享集群Blink版本。

WITH参数

参数	说明	备注
type	结果表类型	固定值为mq。

参数	说明	备注
topic	Message Queue队列名称	无

endpoint	地址	阿里云消息队列提供内网服务MQ（非公网region）和公网服务MQ（公网region）两种类型，请务必根据您购买的MQ的类型选择对应正确的接入地址：
		• Blink 3.7.10及以上版本的作业，需要使用TCP协议客户端接入点，详情请参见 关于TCP内网接入点设置的公告。接入点获取方式如下：
		  ◦ 内网服务MQ（阿里云经典网络/VPC）接入地址：在MQ控制台目标实例详情中，选择**接入点 > TCP协议客户端接入点 > 内网访问**，获取对应的endPoint。
		  ◦ 公网服务MQ接入地址：在MQ控制台目标实例详情中，选择**接入点 > TCP协议客户端接入点 > 公网访问**，获取对应的endPoint。
		• Blink 3.7.10（不含）以下版本的作业，使用如下接入点：
		  ◦ 内网服务MQ（阿里云经典网络/VPC）接入地址：
		  ▪ 华东1（杭州）、华东2（上海）、华北1（青岛）、华北2（北京）、华南1（深圳）、中国（香港）：`onsaddr-internal.aliyun.com:8080`
		  ▪ 亚太东南1（新加坡）：`ap-southeastaddr-internal.aliyun.com:8080`。
		  ▪ 中东东部1（迪拜）：`ons-me-east-1-internal.aliyuncs.com:8080`。
		  ▪ 亚太南部1（孟买）：`ons-ap-south-1-internal.aliyuncs.com:8080`。
		  ▪ 亚太东南3（吉隆坡）：`ons-ap-southeast-3-internal.aliyun.com:8080`。
		  ◦ 公网服务MQ接入地址：`onsaddr-internet.aliyun.com:80`。
		注意
		• 如果您已使用了Blink 3.7.10（不含）以下版本的RocketMQ Connector，则您需要将您的实时计算作业升级至Blink 3.7.10及以上版本，并将作业中EndPoint参数取值更改为新的RocketMQ接入点，旧的RocketMQ接入点存在稳定性风险或不可用的问题，详情请参见RocketMQ接入点变更导致实时计算作业适配升级公告。
		• 内网服务无法跨地域访问。例如，您所购买的实时计算Flink版服务的地域为华东1，但是购买的消息队列MQ服务的地域为华东2，则无法访问。
		• 独享集群默认不能访问公网，如果需要请配置NAT网关。
		• 由于阿里云网络安全策略动态变化，实时计算Flink版连接公网服务MQ时可能会出现网络连接问题，推荐您使用内网服务MQ。如果在使用公网服务MQ时出现异常，请您提交工单进行咨询。

参数	说明	备注
accessID	AccessKey ID	无

参数	说明	备注
accessKey	AccessKey Secret	无
producerGroup	写入的群组	无
tag	写入的标签	可选，默认值为空。
fieldDelimiter	字段分割符	可选，默认值为 `\u0001` 。分隔符的使用情况如下所示： • 只读模式：以 `\u0001` 作为分隔符， `\u0001` 在只读模式不可见。 • 编辑模式：以 `^A` 作为分隔符。
encoding	编码类型	可选，默认值为 `utf-8` 。
retryTimes	写入的重试次数	可选，默认值为10。
sleepTimeMs	重试间隔时间	可选，默认值为1000（毫秒）。
instanceID	MQ实例ID	• 如果MQ实例无独立命名空间，则不可以使用instanceID参数。 • 如果MQ实例有独立命名空间，则instanceID参数必选。

4.6.3.7. 创建表格存储Tablestore结果表

本文为您介绍如何创建实时计算Flink版表格存储结果表，以及表格存储和实时计算Flink版字段类型之间的映射关系。

> **注意** 本文仅适用于Blink 1.4.5及以上版本。

什么是表格存储Tablestore

表格存储Tablestore是基于阿里云飞天分布式系统的分布式NoSQL数据存储服务。表格存储通过数据分片和负载均衡技术，实现数据规模与访问并发上的无缝扩展，提供海量结构化数据的存储和实时访问服务。

DDL定义

实时计算Flink版支持使用Tablestore作为结果输出，示例代码如下。

```
CREATE TABLE stream_test_hotline_agent (
 name VARCHAR,
 age BIGINT,
 birthday BIGINT,
 PRIMARY KEY (name,age)
) WITH (
 type='ots',
 instanceName='<yourInstanceName>',
 tableName='<yourTableName>',
 accessId='<yourAccessId>',
 accessKey='<yourAccessSecret>',
 endPoint='<yourEndpoint>',
 valueColumns='birthday'
);
```

> **说明**
> - 推荐使用数据存储注册功能，详情请参见注册表格存储Tablestore。
> - valueColumns值不能是声明的主键，可以是主键之外的任意字段。
> - Tablestore结果表声明中，除主键列外，至少包含一个属性列。

WITH参数

参数	说明	备注
type	结果表类型	固定值为ots。
instanceName	实例名	无
tableName	表名	无
endPoint	实例访问地址	参见服务地址。
accessId	AccessKey ID	无
accessKey	AccessKey Secret	无
valueColumns	指定插入的字段列名	插入多个字段以英文逗号（,）分割。例如 `'ID,NAME'` 。
bufferSize	流入多少条数据后开始输出	可选，默认值为5000，表示输入的数据达到5000条就开始输出。 **说明** 在实时计算Flink版系统，bufferSize根据Tablestore主键对结果数据进行去重后，再在bufferSize的基础上进行batchSize。

参数	说明	备注
batchWriteTimeoutMs	写入超时的时间	可选，单位为毫秒，默认值为5000。表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
batchSize	一次批量写入的条数	可选，默认值为100。
retryIntervalMs	重试间隔时间	可选，单位毫秒，默认值为1000。
maxRetryTimes	最大重试次数	可选，默认值为100。
ignoreDelete	是否忽略DELETE操作	默认值为False。

类型映射

Tablestore字段类型	实时计算Flink版字段类型
INTEGER	BIGINT
STRING	VARCHAR
BOOLEAN	BOOLEAN
DOUBLE	DOUBLE

> **说明** Tablestore结果表必须定义有 `Primary Key` ，以Update方式写入结果数据到Tablestore表。Update方式说明请参见Update类型。

4.6.3.8. 创建云数据库RDS MySQL版结果表

本文为您介绍如何创建实时计算云数据库RDS MySQL版结果表，以及创建结果表时使用的WITH参数和类型映射。

什么是云数据库RDS MySQL版

RDS MySQL基于阿里巴巴的MySQL源码分支，经过双十一高并发、大数据量的考验，拥有优良的性能。RDS MySQL支持实例管理、账号管理、数据库管理、备份恢复、白名单、透明数据加密以及数据迁移等基本功能。除此之外还提供如下高级功能：

- 专属集群MyBase：是由多台主机（底层服务器，如ECS I2服务器、神龙服务器）组成的集群，相对于全托管数据库，可以满足您更多的需求。
- 只读实例：在对数据库有少量写请求，但有大量读请求的应用场景下，单个实例可能无法承受读取压力，甚至对业务产生影响。为了实现读取能力的弹性扩展，分担数据库压力，您可以创建一个或多个只读实例，利用只读实例满足大量的数据库读取需求，增加应用的吞吐量。
- 读写分离：读写分离功能是在只读实例的基础上，额外提供了一个读写分离地址，联动主实例及其所有只读实例，创建自动的读写分离链路。应用程序只需连接读写分离地址进行数据读取及写入操作，读写分离程序会自动将写入请求发往主实例，而将读取请求按照权重发往各个只读实例。用户只需通过添加只读实例的个数，即可不断扩展系统的处理能力，应用程序上无需做任何修改。
- 什么是数据库代理：数据库独享代理服务是使用独立代理计算资源为当前实例提供代理服务，提供更多高

级功能，例如读写分离、短连接优化、事务拆分等。

- 自治服务DAS：针对SQL语句性能、CPU使用率、IOPS使用率、内存使用率、磁盘空间使用率、连接数、锁信息、热点表等，DAS提供了智能的诊断及优化功能，能最大限度发现数据库存在的或潜在的健康问题。DAS的诊断基于单个实例，会提供问题详情及相应的解决方案，为您维护实例带来极大的便利。

RDS MySQL仅支持InnoDB和X-Engine两种存储引擎，具体的功能请参见MySQL功能概览。

使用限制

实时计算Flink版暂不支持通过数据存储功能中存储注册的方式使用RDS MySQL 8.0版本，建议您使用明文方式使用RDS MySQL 8.0版本。数据存储功能详情请参见概述。

语法示例

实时计算Flink版支持使用RDS MySQL版作为结果输出，示例代码如下。

```
CREATE TABLE rds_output(
   id INT,
   len INT,
   content VARCHAR,
   PRIMARY KEY (id,len)
) WITH (
   type='rds',
   url='<yourDatabaseURL>',
   tableName='<yourDatabaseTable>',
   userName='<yourDatabaseUserName>',
   password='<yourDatabasePassword>'
);
```

> **说明**
>
> - 实时计算Flink版写入RDS MySQL数据库结果表原理：针对实时计算Flink版每行结果数据，拼接成一行SQL语句，输入至目标端数据库，然后执行。如果使用批量写，需要在URL后面加上参数 `?rewriteBatchedStatements=true` ，以提高系统性能。
> - RDS MySQL数据库支持自增主键。如果实时计算Flink版写入数据支持自增主键，则在DDL中不声明该自增字段即可。例如，ID是自增字段，实时计算Flink版DDL不声明该自增字段，则数据库在一行数据写入过程中会自动填补相关自增字段。
> - 建议使用数据存储注册方式，请参见注册云数据库RDS版。
> - DDL声明的字段必须至少存在一个非主键的字段，否则产生报错。

WITH参数

参数	说明	是否必填	备注
type	结果表类型	是	固定值为 `rds` 。
url	JDBC（Java DataBase Connectivity）连接地址	是	URL的格式为： `jdbc:mysql://<内网地址>/<databaseName>` ，其中databaseName为对应的数据库名称。内网地址参见如下链接： • Apply for an Internet IP address for RDS

参数	说明	是否必填	备注
tableName	表名	是	无
userName	用户名	是	无
password	密码	是	无
maxRetryTimes	最大重试次数	否	默认值为10。
batchSize	一次批量写入的条数	否	默认值为4096。
bufferSize	流入多少条数据后开始去重	否	默认值为10000。
flushIntervalMs	清空缓存的时间间隔	否	默认值为2000，单位为毫秒。表示如果缓存中的数据在等待2秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
excludeUpdateColumns	忽略指定字段的更新	否	默认值为空（默认忽略PRIMARY KEY字段）。表示更新主键值相同的数据时，忽略指定字段的更新。
ignoreDelete	是否忽略delete操作	否	默认值为false，表示支持delete功能。
partitionBy	分区	否	默认为空。表示写入Sink节点前，会根据该值进行Hash分区，数据会流向相应的Hash节点。

类型映射

RDS字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME

RDS字段类型	实时计算Flink版字段类型
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

JDBC连接参数

参数名称	说明	默认值	最低版本要求
useUnicode	是否使用Unicode字符集，如果参数characterEncoding设置为GB2312或GBK，本参数值必须设置为true。	false	1.1g
characterEncoding	当useUnicode设置为true时，指定字符编码。例如可设置为GB2312或GBK。	false	1.1g
autoReconnect	当数据库连接异常中断时，是否自动重新连接。	false	1.1
autoReconnectForPools	是否使用针对数据库连接池的重连策略。	false	3.1.3
failOverReadOnly	自动重连成功后，连接是否设置为只读。	true	3.0.12
maxReconnects	autoReconnect设置为true时，重试连接的次数。	3	1.1
initialTimeout	autoReconnect设置为true时，两次重连之间的时间间隔，单位为秒。	2	1.1
connectTimeout	和数据库服务器建立socket连接时的连接超时时长，单位为毫秒。0表示永不超时，适用于JDK 1.4及以上版本。	0	3.0.1
socketTimeout	socket操作（读写）超时，单位：毫秒。0表示永不超时。	0	3.0.1

代码示例

包含云数据库RDS版结果表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE source (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'random'
);
CREATE TABLE rds_output(
   id INT,
   len INT,
   content VARCHAR,
   PRIMARY KEY (id,len)
) WITH (
   type='rds',
   url='<yourDatabaseURL>',
   tableName='<yourDatabaseTable>',
   userName='<yourDatabaseUserName>',
   password='<yourDatabasePassword>'
);
INSERT INTO rds_output
SELECT id, len, content FROM source;
```

FAQ

- Q：实时计算Flink版的结果数据写入RDS表，是按主键更新的，还是生成1条新的记录？

  A：如果在DDL中定义了主键，会采用 `INSERT INTO tablename(field1,field2, field3, ...) VALUES(value1, value2, value3, ...) ON DUPLICATE KEY UPDATE field1=value1,field2=value2, field3=value3, ...;` 的方式更新记录，即对于不存在的主键字段会直接插入，存在的主键字段则更新相应的值。如果DDL中没有声明PRIMARY KEY，则会用 `insert into` 方式插入记录，追加数据。

- Q：使用RDS表中的唯一索引进行GROUP BY时需要注意什么？

  A：

  - 需要在作业中的GROUP BY中声明该唯一索引。
  - RDS中只有一个自增主键，实时计算Flink版作业中不能声明为PRIMARY KEY。

4.6.3.9. 创建MaxCompute结果表

本文为您介绍如何创建MaxCompute结果表，以及创建过程中及到的WITH参数、类型映射和常见问题。

> **注意**
>
> - 仅Blink 1.5.1及以上版本支持MaxCompute结果表。
> - MaxCompute中的Clustered Table表不支持作为MaxCompute结果表。

实现原理

MaxCompute Sink可以分为以下两个阶段：

1. 写入数据。调用MaxCompute SDK中的接口将数据写入缓冲区，在缓冲区大小超过64 MB或者每隔指定的时间间隔时，上传数据到MaxCompute的临时文件中。

2. 提交会话。在任务进行Checkpoint时， MaxCompute Sink会调用Tunnel的Commit方法，提交会话，移动临时文件到MaxCompute表的数据目录，并修改元数据。

> **说明** Commit方法不能提供原子性。因此，MaxCompute Sink提供的是At least Once方式，而不是Exactly Once方式。

语法示例

实时计算Flink版支持使用MaxCompute作为结果输出，示例代码如下。

> **说明** DDL中定义的字段需要与MaxCompute物理表中的字段名称、顺序以及类型保持一致，否则可能导致MaxCompute物理表中查询的数据为 `/n` 。

```
create table odps_output(
  id INT,
  user_name VARCHAR,
  content VARCHAR
) with (
  type = 'odps',
  endPoint = '<YourEndPoint>',
  project = '<YourProjectName>',
  tableName = '<YourtableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****'
);
```

WITH参数

参数	说明	是否必填	备注
type	结果表类型。	是	固定值为 `odps` 。
endPoint	MaxCompute服务地址。	是	请参见Endpoint。
tunnelEndpoint	MaxCompute Tunnel服务的连接地址。	是	请参见Endpoint。 **说明** VPC环境下必填。
project	MaxCompute项目名称。	是	无。
tableName	表名。	是	无。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。

参数	说明	是否必填	备注
partition	分区名。	否	如果存在分区表则必填： • 固定分区 例如 `partition` = 'ds=20180905' 表示将数据写入分区 ds= 20180905 。 • 动态分区（Blink 3.2.1及以上版本支持） 如果不明文显示分区的值，则会根据写入数据中的分区列具体的值，写入到不同的分区中。例如 `partition`='ds' 表示根据 ds 字段的值写入分区。 如果要创建多级动态分区，With参数中Partition的字段顺序和结果表的DDL中的分区字段顺序，必须与物理表一致，各个分区字段之间使用逗号（,）分割。 说明 ◦ 动态分区列需要显式写在建表语句中。 ◦ 对于动态分区字段为空的情况，如果数据源中 ds=null 或者 ds='' ，则输出结果为： ▪ 3.2.1及以前的版本会抛出空指针异常（NPE）。 ▪ 3.2.2及以后的版本会创建ds=NULL的分区。
flushIntervalMs	Odps tunnel writer缓冲区Flush间隔，单位毫秒。 MaxCompute Sink写入记录时，先放入数据到MaxCompute的缓冲区中，等缓冲区溢出或者每隔一段时间（flushIntervalMs）时，再把缓冲区里的数据写到目标MaxCompute表。	否	默认值是30000毫秒，即30秒。 说明 Blink 3.6.0及以上版本支持该参数。

参数	说明	是否必填	备注
partitionBy	写入Sink节点前，系统会根据该值做Hash Shuffle，数据就会流向对应的Sink节点。	否	系统按照多个列进行Hash Shuffle，各个列名之间使用逗号（,）分割。默认为空。 **说明** Blink 3.6.0及以上版本支持该参数。
isOverwrite	写入Sink节点前，是否将结果表清空。	否	• Blink 3.2以下版本默认参数值为true。 • Blink 3.2及以上版本默认参数值为false。在声明MaxCompute的流式作业结果表时，isOverwrite参数必须为false，否则在编译时会抛出异常。 **说明** Blink 3.2及以上版本支持isOverwrite参数值变更为true。Blink 3.2以下版本如需变更isOverwrite参数值，请升级版本。
dynamicPartitionLimit	分区数目最大值。	否	默认值是100，内存中会把出现过的分区和Tunnel/writer的映射关系维护到一个Map里，如果这个Map的大小超过了dynamicPartitionLimit设定值，则会出现 `Too many dynamic partitions: 100, which exceeds the size limit: 100` 报错。

类型映射

MaxCompute字段类型	实时计算Flink版字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
STRING	VARCHAR

MaxCompute字段类型	实时计算Flink版字段类型
DECIMAL	DECIMAL

代码示例

包含MaxCompute结果表的实时计算Flink版作业代码示例如下：

- 写入固定分区

```
CREATE TABLE source (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'random'
);
create table odps_sink (
   id INT,
   len INT,
   content VARCHAR
) with (
   type = 'odps',
   endPoint = '<yourEndpoint>',
   project = '<yourProjectName>',
   tableName = '<yourTableName>',
   accessId = '<yourAccessId>',
   accessKey = '<yourAccessPassword>',
   `partition` = 'ds=20180418'
);
INSERT INTO odps_sink
SELECT
   id, len, content
FROM source;
```

- 写入动态分区

```
CREATE TABLE source (
  id INT,
  len INT,
  content VARCHAR,
  c TIMESTAMP
) with (
  type = 'random'
);
create table odps_sink (
  id INT,
  len INT,
  content VARCHAR,
  ds VARCHAR --动态分区列需要显式写在建表语句中。
) with (
  type = 'odps',
  endPoint = '<yourEndpoint>',
  project = '<yourProjectName>',
  tableName = '<yourTableName>',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessPassword>',
  `partition`='ds' --不写分区的值，表示根据ds字段的值写入不同分区。
);
INSERT INTO odps_sink
SELECT
   id,
   len,
   content,
   DATE_FORMAT(c, 'yyMMdd') as ds
FROM source;
```

常见问题

- Q：公共云的endPoint和tunnelEndpoint是什么？配置错误会产生什么结果？

  A：endPoint和tunnelEndpoint参数说明参见Endpoint。VPC环境中这两个参数如果配置错误可能会导致任务异常。

  - endPoint配置错误：任务上线停滞在91%的进度。
  - tunnelEndpoint配置错误：任务运行失败。

- Q: Stream模式的MaxCompute结果表是否支持isOverwrite为true?

  A：Blink 3.2以下版本支持，Blink3.2及以上版本不支持。

  isOverwrite为true，即写入结果表之前会把结果表或者结果数据清空。作业每次启动后和暂停恢复后、写入之前会把原来结果表或者结果分区里的内容删除。

  - Blink 3.2以下版本isOverwrite默认值是true，且不支持修改。流式作业完成暂停或恢复操作后，会造成数据丢失。
  - Blink 3.2及以上版本isOverwrite默认值是false，且在声明MaxCompute流式作业结果表时，isOverwrite参数值需要设置为false，否则在编译时会抛出异常。Stream模式的MaxCompute结果表具备At Least Once数据保障机制，在作业运行失败后，可能会出现数据重复。

- Q：作业运行过程中报错ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

A：该报错是因为MaxCompute DDL定义中填写的用户身份信息无法访问MaxCompute。因此，您需要通过阿里云账号、RAM用户账号或RAM角色认证用户身份，详情请参见用户认证。

如果您有其他疑问，请提交工单咨询，产品名称选择为MaxCompute。

4.6.3.10. 创建云数据库HBase版结果表

本文为您介绍如何创建实时计算云数据库HBase版结果表。

> 注意
> - 本文档仅适用于实时计算独享模式。
> - Blink 3.3.0以下版本仅支持HBase标准版。
> - Blink 3.3.0及以上版本同时支持HBase标准版和HBase增强版。
> - Blink 3.5.0及以上版本支持HBase写入主备切换。
> - 实时计算HBase结果表不支持自建的开源HBase。

DDL定义

实时计算支持使用HBase作为结果输出。

- HBase标准版示例代码如下。

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '2',
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

- HBase增强版示例代码如下。

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    endPoint = '<host:port>', ----HBase增强版的Java API访问地址。
    userName  = 'root', --用户名。
    password = 'root', --密码。
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

- Blink 3.5.0及以上HBase增强版示例代码如下。

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '<host:port>', ----HBase增强版的Java API访问地址。
    userName  = 'root', --用户名。
    password = 'root', --密码。
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

- Blink 3.5.0及以上HBase写入主备切换示例代码如下。

```
create table liuxd_user_behavior_test_front (
    row_key varchar,
    from_topic varchar,
    origin_data varchar,
    record_create_time varchar,
    primary key (row_key)
) with (
    type = 'cloudhbase',
    zkQuorum = '<host:port>', ----HBase高可用访问地址。
    haClusterID = 'ha-xxx', ----HBase高可用实例ID。
    userName  = 'root', --用户名。
    password = 'root', --密码。
    columnFamily = '<yourColumnFamily>',
    tableName = '<yourTableName>',
    batchSize = '500'
);
```

> **说明**
> - PRIMARY KEY支持定义多字段。多字段以 `rowkeyDelimiter`（默认为 `:`）作为分隔符进行连接。
> - HBase执行撤回删除操作时，如果COLUMN定义了多版本，将清空所有版本的COLUMN值。
> - HBase标准版和HBase增强版DDL的区别为连接参数不同：
>   - HBase标准版使用连接参数 `zkQuorum`。
>   - HBase增强版使用连接参数 `endPoint`。

WITH参数

参数	说明	是否必填	备注
type	结果表类型	是	固定值为cloudhbase。
zkQuorum	HBase集群配置的zk地址	是	可以在*hbase-site.xml*文件中查看hbase.zookeeper.quorum相关配置。 **说明** 仅在HBase标准版中生效。
zkNodeParent	集群配置在zk上的路径	否	可以在*hbase-site.xml*文件中查看hbase.zookeeper.quorum相关配置。 **说明** 仅在HBase标准版中生效。
endPoint	HBase地域名称	是	可在购买的HBase实例控制台中获取。 **说明** 仅在HBase增强版中生效。
userName	用户名	否	仅在HBase增强版中生效。
password	密码	否	仅在HBase增强版中生效。
tableName	HBase表名	是	无
columnFamily	列族名	是	仅支持插入同一列族。
maxRetryTimes	最大尝试次数	否	默认值为10。 **说明** 仅实时计算3.2.3及以上版本支持maxRetryTimes参数。

参数	说明	是否必填	备注
bufferSize	流入多少条数据后进行去重	否	默认值为5000。
batchSize	一次批量写入的条数	否	默认值为100。 **说明** 建议batchSize参数值为200~300。过大的batchSize值可能导致任务OOM（内存不足）报错。
flushIntervalMs	周期性清理buffer的间隔，可以减少写入HBase的延迟。	否	默认值为2000，单位为毫秒。
writePkValue	是否写入主键值	否	默认值为false。
stringWriteMod	是否都按照STRING插入	否	默认值为false。
rowkeyDelimiter	rowKey的分隔符	否	默认值为冒号（:）。
isDynamicTable	是否为动态表	否	默认值为false。
haClustserID	HBase高可用实例ID	否	只有访问同城主备实例时才需要配置。

动态表

实时计算部分结果数据需要按某列的值，作为动态列输入HBase。HBase中，以每小时的成交额作为动态列的数据，示例如下。

rowkey	cf:0	cf:1	cf:2
20170707	100	cf:1	300

当isDynamicTable参数值为true时，表明该表为支持动态列的HBase表。

动态表仅支持3列输出，例如，ROW_KEY、COLUMN和VALUE。此时第2列（本示例中的COLUMN）为动态列，动态表中的其它参数与HBase的WITH参数一致。

说明 使用动态表时，所有数据类型需要先转换为STRING类型，再进行输入。

```
CREATE TABLE stream_test_hotline_agent (
  name varchar,
  age varchar,
  birthday varchar,
  primary key (name)
) WITH (
  type = 'cloudhbase',
  ...
  columnFamily = 'cf',
  isDynamicTable ='true'
);
```

说明

- 以上声明将把 `birthday` 插入到以 `name` 为ROW_KEY的 `cf:age` 列中。例如， `(wang,18,2016-12-12)` 会插入ROW_KEY为 `wang` 的行， `cf:18` 列。
- DDL中必选按照从上到下的顺序，声明ROW_KEY（本示例中的 `name` ）、COLUMN（本示例中的 `age` ）和VALUE（本示例中的 `birthday` ），且声明ROW_KEY为PRIMARY KEY。

代码示例

包含HBase结果表的实时计算作业代码示例如下。

```
create table source (
  id   TINYINT,
  name BIGINT
) with (
  type = 'random'
);
create table sink (
  id    TINYINT,
  name  BIGINT,
  primary key (id)
) with (
  type = 'cloudhbase',
  zkQuorum = '<yourZkQuorum>',
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>'
);
INSERT INTO sink
SELECT id, name FROM source;
```

常见问题

Q：为什么HBase结果表作业运行时会报错 `cloudHbase update error, No columns to insert for #10 item` ？

A：HBase结果表要求写入的单条记录的列数据（不包括rowkey）不能全为NULL。请先过滤全为NULL的数据，再输入HBase。

4.6.3.11. 创建Elasticsearch结果表

本文为您介绍如何创建实时计算Flink版Elasticsearch（ES）结果表以及创建结果表时使用的WITH参数。

> **注意** 本文仅适用于Blink 3.2.2及以上版本。

DDL定义

实时计算Flink版支持使用ES作为结果输出，示例代码如下。

```
 CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 VARCHAR,
  PRIMARY KEY(field1)
)WITH(
  type ='elasticsearch',
  endPoint = 'http://es-cn-mp****.public.elasticsearch.aliyuncs.com:****',
  accessId = '<yourUsername>',
  accessKey = '<yourPassword>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>'
);
```

> **说明**
>
> - ES支持根据PRIMARY KEY进行UPDATE，且PRIMARY KEY只能为1个字段。
> - 在指定PRIMARY KEY后，Document的ID为PRIMARY KEY字段的值。
> - 在未指定PRIMARY KEY时，Document的ID为随机，详情请参见Index API。
> - 在full更新模式下，新增的doc会完全覆盖已存在的doc。
> - 在inc更新模式下，系统会依据输入的字段值更新对应的字段。
> - 所有的更新默认为UPSERT语义，即INSERT或UPDATE。

WITH参数（通用配置）

参数	说明	是否必选	默认值
type	connector类型。	是	elasticsearch
endPoint	Server地址，例入：*http://127.0.0.1:9211*。	是	无
accessId	创建ES时的登录名。 **说明** 如果您通过Kibana插件操作ES，请填写Kibana登录ID。	是	无

参数	说明	是否必选	默认值
accessKey	创建ES时的登录密码。 **说明** 如果您通过Kibana插件操作ES，请填写Kibana登录密码。	是	无
index	索引名称。	是	无
typeName	文档类型。	是	_doc
bufferSize	流入多少条数据后开始去重。	否	1000
maxRetryTimes	异常重试次数。	否	30
timeout	读超时，单位为毫秒。	否	600000
discovery	是否开启节点发现。如果开启，客户端每5分钟刷新一次Server List。	否	false
compression	是否使用GZIP压缩Request Bodies。	否	true
multiThread	是否开启JestClient多线程。	否	true
ignoreWriteError	是否忽略写入异常。	否	false
settings	创建Index的Settings配置。	否	无
updateMode	指定主键（PRIMARY KEY）后的更新模式： • full：全量覆盖。 • inc：增量更新。	否	full

WITH参数（动态索引相关）

参数	说明	是否必选	备注
dynamicIndex	是否开启动态索引。	否	参数取值如下： • true：开启。 • false（默认值）：不开启。
indexField	抽取索引的字段名。	dynamicIndex为true时必填。	只支持TIMESTAMP（以秒为单位）、DATE和LONG3种数据类型。
indexInterval	切换索引的周期。	dynamicIndex为true时必填。	参数值如下： • d（默认值）：天 • m：月 • w：周

参数	说明	是否必选	备注
mapping	启用动态索引时，设置文档各字段的类型与格式。例如，设置名为sendTime字段的格式： `{ "properties": { "sendTime": { "type": "date", "format": "yyyy-MM-dd HH:mm:ss" } } }`	否	默认值为空。 **说明** • Blink 3.7.0版本及以上版本支持该参数。 • Elasticsearch 7.0不支持mapping参数。

说明

- 仅Blink 2.2.7及以上版本支持动态索引功能。
- 当开启动态索引后，基本配置中的 `index` 名称会作为后续创建索引的统一Alias，Alias和索引为一对多关系。
- 不同的 `indexInterval` 对应的真实索引名称：
  - d -> Alias "yyyyMMdd"
  - m -> Alias "yyyyMM"
  - w -> Alias "yyyyMMW"
- 对于单个的真实索引可使用Index API进行修改，但对于Alias只支持 `get` 功能。如果需要更新Alias，请参见Index Aliases。

代码示例

创建动态索引的代码示例如下所示。

```
CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 TIMESTAMP,
  PRIMARY KEY(field1)
)WITH(
  type ='elasticsearch',
  endPoint = 'http://es-cn-mp****.public.elasticsearch.aliyuncs.com:****',
  accessId = '<yourAccessId>',
  accessKey = '<yourAccessSecret>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>',
  dynamicIndex = 'true',
  indexField = 'field3',
  indexInterval = 'd'
);
```

4.6.3.12. 创建时序数据库结果表

本文为您介绍如何创建实时计算Flink版时序数据库（TSDB）结果表以及创建结果表时使用的WITH参数。

> **注意**
> - 本文仅适用于Blink 2.0及以上版本。
> - 实时计算Flink版引用时序数据库（TSDB）结果表需要配置数据存储白名单，详情请参见数据存储白名单配置。

什么是时间序列数据库（TSDB）

阿里云时序数据库（Time Series Database，简称TSDB）是一种集时序数据高效读写、压缩存储、实时计算能力为一体的数据库服务，可以广泛应用于物联网和互联网领域，实现对设备及业务服务的实时监控，实时预测告警。

DDL定义

实时计算Flink版支持使用TSDB作为结果输出，示例代码如下。

```
CREATE TABLE stream_test_hitsdb (
    metric      VARCHAR,
    `timestamp` INTEGER,
    `value`     DOUBLE,
    tagk1       VARCHAR,
    tagk2       VARCHAR,
    tagk3       VARCHAR
) WITH (
    type='hitsdb',
    host='<yourHostName>',
    virtualDomainSwitch = 'false',
    httpConnectionPool = '20',
    batchPutSize = '1000'
);
```

建表默认格式：

- 第0列：metric（VARCHAR）。
- 第1列：timestamp（INTEGER），单位为秒。
- 第2列：value（DOUBLE）。
- 第3~N列：TagKey，即时间序列数据库中的fieldName。

> **说明**
> - tag可以为多列。
> - 必须声明metric、timestamp和value，且字段名称、字段顺序和字段数据类型必须和TSDB保持完全一致。
> - 参数设置详情请参见Write data。

WITH参数

参数	说明	备注
type	结果表类型	固定值为 `hitsdb` 。
host	IP或VIP域名	填写注册实例的Host，详情请参见Connect to the instance。
port	端口	默认值为8242。
virtualDomainSwitch	是否使用VIPServer	默认值为false，如果需要使用VIPServer，则virtualDomainSwitch设置为true。
httpConnectionPool	HTTP连接池	默认值为10。
httpCompress	使用GZIP压缩	默认值为false，即不压缩。
httpConnectTimeout	HTTP连接超时时间	默认值为0。
ioThreadCount	IO线程数	默认值为1。
batchPutBufferSize	缓冲区大小	默认值为10000。
batchPutRetryCount	写入重试次数	默认值为3。
batchPutSize	每次提交数据量	默认每次提交500个数据点。
batchPutTimeLimit	缓冲区等待时间	默认值为200，单位为毫秒。
batchPutConsumerThreadCount	序列化线程数	默认值为1。

Blink写入TSDB模型

Blink 3.2 及以上版本支持6种Blink写入TSDB的模型，分别是：

- 支持单值无tag数据点的写入，其schema格式如下所示，包括3个字段，这3个字段名称不可使用其他名称代替。

  ```
  metric,timestamp,value
  ```

- 支持单值带tag数据点的写入，其schema格式如下所示，除metric、timestamp和value关键词名字必须保持一致外，tag的名称可以任意指定。

  ```
  metric,timestamp,value, tagKey1,....,tagKeyN
  ```

- 支持单值带不确定tag个数的数据点的写入，其schema格式如下所示，包括4个字段，这4个字段名称不可使用其他名称代替。

  ```
  metric,timestamp,value,tags
  ```

  其中，tags的内容为形如如下格式的JSON字符串，便于绕过Blink table schema需要固定tag个数的限制。

  ```
  {"tagKey1":"tagValue1","tagKey2":"tagValue2",……,"tagKeyN":"tagValueN"}
  ```

- 支持多值无tag数据点的写入，其schema格式如下所示。

```
metric,timestamp,field_name1,field_name2,……,field_nameN
```

其中metric和timestamp字段名称不可使用其他名称代替。对于多值的fields，由于需要区分tag和field，同时兼容之前的单值写入，这里约定需要给每个field加上固定前缀field，自动识别field字段。例如，对于一个field名称 `field_name1` ，在写入TSDB时，会自动将前缀 `field_` 去掉，只保留name1，即对于上面的schema，实际写入TSDB的格式如下，name1和name2是多值field的名称。

```
metric,timestamp,name1,name2,……,nameN
```

- 支持多值带tag数据点的写入，其schema格式如下所示，除metric和timestamp关键词名字必须保持一致外，tag的名称可以任意指定。

```
metric,timestamp, tagKey1,....,tagKeyN, field_name1,field_name2,……,field_nameN
```

- 支持单值带不确定tag个数的数据点的写入，其schema格式如下所示。

```
metric,timestamp,tags,field_name1,field_name2,……,field_nameN
```

其中，tags的内容类似如下JSON字符串，便于绕过Blink table schema需要固定tag个数的限制。

```
{"tagKey1":"tagValue1","tagKey2":"tagValue2",……,"tagKeyN":"tagValueN"}
```

常见问题

Q：为什么Failover中会报错：LONG类型不能转换成INT类型？

A：Blink 2.2.5以下版本仅支持INT类型。Blink 2.2.5及以上版本支持BIGINT类型。

4.6.3.13. 创建消息队列Kafka结果表

本文档为您介绍如何创建实时计算Flink版消息队列Kafka结果表，以及Kafka版本对应关系。

> 注意
> - 本文仅适用于实时计算2.0及以上版本。
> - 本文仅适用于独享模式。
> - Kafka结果表支持写入自建Kafka集群，但需注意版本对应关系，以及自建集群和实时计算集群的网络环境配置。

什么是Kafka结果表

消息队列Kafka版是阿里云提供的分布式、高吞吐、可扩展的消息队列服务。消息队列Kafka版广泛用于日志收集、监控数据聚合、流式数据处理、在线和离线分析等大数据领域。实时计算采用Kafka作为流式数据的数据源表或结果表。

DDL定义

消息队列Kafka结果表需要定义的DDL如下。

```
create table sink_kafka (
  messageKey VARBINARY,
  `message` VARBINARY,
  PRIMARY KEY (messageKey)
) with (
  type = 'kafka010',
  topic = '<yourTopicName>',
  bootstrap.servers = '<yourServerAddress>'
);
```

> **说明**
> - 创建Kafka结果表时，必须明文指定 `PRIMARY KEY (messageKey)` 。
> - 仅Blink 2.2.6及以上版本，支持阿里云Kafka或自建Kafka的TPS和RPS等指标信息的显示。

WITH参数

- 通用配置

参数	说明	是否必选	备注
type	Kafka对应版本	是	必须是Kafka08、Kafka09、Kafka010、Kafka011中的一种，版本对应关系请参见Kafka版本对应关系。
topic	Topic名称	是	无

- 必选配置
  - Kafka08必选配置

参数	说明
zookeeper.connect	zk连接ID

  - Kafka09/Kafka010/Kafka011必选配置

参数	说明
bootstrap.servers	Kafka集群地址

- 可选配置参数
  - `consumer.id`
  - `socket.timeout.ms`
  - `fetch.message.max.bytes`
  - `num.consumer.fetchers`
  - `auto.commit.enable`
  - `auto.commit.interval.ms`
  - `queued.max.message.chunks`

- `rebalance.max.retries`
- `fetch.min.bytes`
- `fetch.wait.max.ms`
- `rebalance.backoff.ms`
- `refresh.leader.backoff.ms`
- `auto.offset.reset`
- `consumer.timeout.ms`
- `exclude.internal.topics`
- `partition.assignment.strategy`
- `client.id`
- `zookeeper.session.timeout.ms`
- `zookeeper.connection.timeout.ms`
- `zookeeper.sync.time.ms`
- `offsets.storage`
- `offsets.channel.backoff.ms`
- `offsets.channel.socket.timeout.ms`
- `offsets.commit.max.retries`
- `dual.commit.enabled`
- `partition.assignment.strategy`
- `socket.receive.buffer.bytes`
- `fetch.min.bytes`

> **说明** 其它可选配置项请参见以下Kafka官方文档：
>
> - Kafka09
> - Kafka010
> - Kafka011

Kafka版本对应关系

type	Kafka版本
Kafka08	0.8.22
Kafka09	0.9.0.1
Kafka010	0.10.2.1
Kafka011	0.11.0.2及以上

示例

```
create table datahub_input (
  id VARCHAR,
  nm VARCHAR
) with (
  type = 'datahub'
);
create table sink_kafka (
  messageKey VARBINARY,
  `message` VARBINARY,
  PRIMARY KEY (messageKey)
) with (
  type = 'kafka010',
  topic = '<yourTopicName>',
  bootstrap.servers = '<yourServerAddress>'
);
INSERT INTO
  sink_kafka
SELECT
  cast(id as VARBINARY) as messageKey,
  cast(nm as VARBINARY) as `message`
FROM
  datahub_input;
```

4.6.3.14. 创建云数据库HybridDB for MySQL结果表

本文为您介绍如何创建实时计算Flink版云数据库HybridDB for MySQL结果表，以及创建结果表时使用的WITH参数。

> **注意**
>
> - 阿里云数据库HybridDB for MySQL产品已下线。
> - 本文仅适用于Blink 1.4.5及以上版本。

什么是云数据库HybridDB for MySQL

云数据库HybridDB for MySQL（原名PetaData）是同时支持海量数据在线事务（OLTP）和在线分析（OLAP）的HTAP（Hybrid Transaction/Analytical Processing）关系型数据库。HybridDB for MySQL采用一份数据存储来进行OLTP和OLAP处理，避免把一份数据复制多次进行数据分析，极大地降低了数据存储的成本。

DDL定义

实时计算Flink版支持使用HybridDB for MySQL作为结果输出，示例代码如下。

```
create table petadata_output(
 id INT,
 len INT,
 content VARCHAR,
 primary key(id,len)
) with (
 type='petaData',
 url='yourDatabaseURL',
 tableName='yourTableName',
 userName='yourDatabaseUserName',
 password='yourDatabasePassword'
);
```

说明

- 实时计算Flink版写入PetaData数据库结果表原理：针对实时计算Flink版每行结果数据，拼接成一行SQL语句，输入至目标端数据库。
- bufferSize默认值是*1000*，如果到达bufferSize阈值，则会触发写出。因此您配置batchSize的同时还需要配置bufferSize。bufferSize和batchSize大小相同即可。
- batchSize数值不建议设置过大，建议设置*batchSize='4096'*。

WITH参数

参数	说明	是否必填	备注
type	结果表类型	是	固定值为petaData。
url	地址	是	切换网络类型。
tableName	表名	是	无
userName	用户名	是	无
password	密码	是	无
maxRetryTimes	最大重试次数	否	默认值为*3*。
batchSize	一次批量写入的条数	否	默认值为*1000*，表示每次写多少条。
bufferSize	流入多少条数据后开始去重	否	无
flushIntervalMs	清空缓存的时间间隔	否	单位为毫秒。默认值为*3000*，表示如果缓存中的数据在等待5秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
ignoreDelete	是否忽略Delete操作	否	默认值为false。

4.6.3.15. 创建云数据库RDS SQL Server版结果表

本文为您介绍如何创建云数据库RDS SQL Server版结果表，以及创建结果表时使用的WITH参数、类型映射和JDBC连接参数。

> **注意**
> - 本文仅适用于Blink 3.2.0及以上版本。
> - 实时计算Flink版暂不支持引用RDS SQL Server版本作为数据存储。

语法示例

实时计算Flink版支持使用云数据库RDS SQL Server版作为结果表输出，示例代码如下。

```
create table ss_output(
  id INT,
  len INT,
  content VARCHAR,
  primary key(id,len)
) with (
  type='jdbc',
  url='jdbc:sqlserver://ip:port;database=****',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
```

> **说明**
> - 实时计算Flink版写入RDS和DRDS数据库结果表原理：针对实时计算Flink版每行结果数据，拼接成一行SQL语句，输入至目标端数据库。如果使用批量写，需要在URL后面加上参数?rewriteBatchedStatements=true，以提高系统性能。
> - RDS SQL Server数据库支持自增主键。如果需要让实时计算Flink版写入数据支持自增主键，则在DDL中不声明该自增字段。例如，ID是自增字段，实时计算Flink版DDL不写出该自增字段，则数据库在一行数据写入过程中会自动填补相关的自增字段。
> - 如果DRDS有分区表，拆分键必须在实时计算Flink版DDL里primary key（）中声明，否则拆分的表无法写入。
> - DDL声明的字段必选至少存在一个非主键的字段，否则产生报错。

WITH参数

参数	说明	是否必填	备注
type	结果表类型	是	固定值为jdbc。
url	JDBC（Java DataBase Connectivity）连接地址	是	地址请参见： • Apply for an Internet IP address for RDS

参数	说明	是否必填	备注
tableName	表名	是	无
username	账号	是	无
password	密码	是	无
maxRetryTimes	写入重试次数	否	默认值为10。
bufferSize	流入多少条数据后开始去重	否	默认值为10000，表示输入的数据达到10000条开始去重。
flushIntervalMs	清空缓存的时间间隔	否	单位为毫秒，默认值为2000，表示如果缓存中的数据在等待2秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
excludeUpdateColumns	忽略指定字段的更新	否	可选，默认值为空（默认忽略Primary Key字段），表示更新主键值相同的数据时，忽略指定字段的更新。
ignoreDelete	是否忽略Delete操作	否	默认值为False。

类型映射

RDS字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

JDBC连接参数

参数	说明	默认值	最低版本要求
useUnicode	是否使用Unicode字符集。如果参数CharacterEncoding设置为GB2312或GBK，useUnicode值必须设置为True。	False	1.1g
characterEncoding	当useUnicode设置为True时，指定字符编码。例如可设置为GB2312或GBK。	False	1.1g
autoReconnect	当数据库连接异常中断时，是否自动重新连接。	False	1.1
autoReconnectForPools	是否使用针对数据库连接池的重连策略。	False	3.1.3
failOverReadOnly	自动重连成功后，连接是否设置为只读。	True	3.0.12
maxReconnects	autoReconnect设置为True时，重试连接的次数。	3	1.1
initialTimeout	autoReconnect设置为True时，两次重连之间的时间间隔，单位为秒。	2	1.1
connectTimeout	和数据库服务器建立socket连接时的超时，单位为毫秒。	0，表示永不超时，适用于JDK 1.4及以上版本。	3.0.1
socketTimeout	socket操作（读写）超时，单位为毫秒。	0，表示永不超时。	3.0.1

代码示例

包含云数据库RDS SQL Server版结果表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE source (
  id INT,
  len INT,
  content VARCHAR
) with (
  type = 'random'
);
CREATE TABLE rds_output(
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY (id,len)
) WITH (
  type='jdbc',
  url='<yourDatabaseURL>',
  tableName='<yourDatabaseTable>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO rds_output
SELECT id, len, content FROM source;
```

常见问题

- Q：实时计算Flink版的结果数据写入RDS表，是按主键更新，还是生成1条新的记录？

  A：根据DDL中是否定义主键，分为以下两种方式：

  - DDL中定义主键：采用 `insert into on duplicate key update` 的方式更新记录，即对于不存在的主键字段会直接插入，存在的主键字段则更新相应的值。
  - DDL中没有定义主键：采用 `insert into` 的方式插入记录，追加数据。

- Q：使用RDS表中的唯一索引进行GROUP BY操作需要注意什么？

  A：需要注意以下两点：

  - 在作业中的主键（Primary Key）中声明该唯一索引。
  - RDS中只有一个自增主键，实时计算Flink版作业中不能将该自增主键声明为主键。

4.6.3.16. 创建云数据库Redis版结果表

本文为您介绍如何创建实时计算Flink版云数据库Redis版结果表以及创建过程中涉及的WITH参数、类型映射和属性字段。

> 注意
>
> - 本文仅适用于Blink 3.2.0及以上版本。
> - 实时计算Flink版Redis结果表支持自建Redis服务。

什么是云数据库Redis版

阿里云数据库Redis版是兼容开源Redis协议标准、提供内存加硬盘混合存储的数据库服务，基于高可靠双机热备架构及可平滑扩展的集群架构，充分满足高吞吐、低延迟及弹性变配的业务需求。实时计算Flink版支持将云数据库Redis版作为流式数据的输出。

语法示例

云数据库Redis版结果表支持5种Redis数据结构，其DDL定义如下：

- STRING类型

  DDL为两列：第1列为key，第2列为value。Redis插入数据的命令为 `set key value` 。

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'string',
  host = '${redisHost}', -- 例如，'127.0.0.1'。
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认值为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

- LIST类型

  DDL为两列：第1列为key，第2列为value。Redis插入数据的命令为 `lpush key value` 。

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'list',
  host = '${redisHost}', -- 例如，'127.0.0.1'。
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认值为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

- SET类型

  DDL为两列：第1列为key，第2列为value。Redis插入数据的命令为 `sadd key value` 。

```
create table resik_output (
  a varchar,
  b varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'set',
  host = '${redisHost}', -- 例如，'127.0.0.1'。
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认值为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

- HASHMAP类型

  DDL为三列：第1列为key，第2列为hash_key，第3列为hash_key对应的hash_value。Redis插入数据的命令为 `hmset key hash_key hash_value` 。

```
create table resik_output (
  a varchar,
  b varchar,
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'hashmap',
  host = '${redisHost}', -- 例如，'127.0.0.1'。
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认值为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

- SORTEDSET类型

  DDL为三列：第1列为key，第2列为score，第3列为value。Redis插入数据的命令为 `add key score value` 。

```
create table resik_output (
  a varchar,
  b double,  --必须为DOUBLE类型。
  c varchar,
  primary key(a)
) with (
  type = 'redis',
  mode = 'sortedset',
  host = '${redisHost}', -- 例如，'127.0.0.1'。
  port = '${redisPort}', -- 例如，'6379'。
  dbNum = '${dbNum}', -- 默认值为0。
  ignoreDelete = 'true' -- 收到Retraction时，是否删除已插入的数据，默认值为false。
);
```

WITH参数

参数	参数说明	是否必填	取值
type	结果表类型	是	固定值为 `redis` 。
mode	对应Redis的数据结构		取值如下： • string • list • set • hashmap • sortedset
host	Redis Server对应地址		取值示例： `127.0.0.1` 。
port	Redis Server对应端口	否	默认值为6379。
dbNum	Redis Server对应数据库序号		默认值为0。
ignoreDelete	是否忽略Retraction消息		默认值为false，可取值为true或false。如果设置为false，收到Retraction时，同时删除数据对应的key及已插入的数据。
password	Redis Server对应的密码		默认值为空，不进行权限验证。
clusterMode	Redis是否为集群模式		取值如下： • true：Redis为集群模式。 • flalse（默认值）：Redis为单机模式。 **说明** 仅Blink 3.6.x及以上版本支持该参数。

类型映射

Redis和实时计算Flink版字段类型对应关系如下。建议您使用该对应关系进行DDL声明。

Redis字段类型	实时计算Flink版字段类型
STRING	VARCHAR
SCORE	DOUBLE

说明 因为Redis的SCORE类型应用于SORTEDSET（有序集合），所以需要手动为每个Value设置一个DOUBLE类型的SCORE，Value才能按照该SCORE从小到大进行排序。

代码示例

包含Redis结果表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE random_stream (
  v VARCHAR,
  p VARCHAR) with (
    type = 'random'
);
create table resik_output (
  a VARCHAR,
  b VARCHAR,
  primary key(a)
) with (
  type = 'redis',
  mode = 'string',
  host = '<yourRedisHost>',
  password = '<yourRedisPassword>'
);
INSERT INTO resik_output
SELECT v, p
FROM random_stream;
```

4.6.3.17. 创建云数据库MongoDB版结果表

本文为您介绍如何创建实时计算Flink版云数据库MongoDB版结果表，以及创建过程中涉及到的WITH参数。

> **注意**
>
> - 本文仅适用于Blink 3.2.2及以上版本。
> - MongoDB结果表不支持主键更新，数据输入形式为重复插入。

DDL定义

实时计算Flink版支持使用MongoDB作为数据输出的结果表，示例代码如下。

```
CREATE TABLE mongodb_sink (
  `a`  VARCHAR
) WITH (
   type = 'mongodb',
   database = '<yourDatabaseName>',
   collection= '<yourCollectionName>',
   uri='mongodb://{<databaseAccount>}:{<atabasePassword>}@{host}:****?replicaSet=mgset-1224****',
   keepAlive='true',
   maxConnectionIdleTime='20000',
   batchSize='2000'
);
```

WITH参数

参数	说明	是否必填	备注
type	Connector类型	是	固定值为mongodb。
database	数据库名称	是	无
collection	数据集合	是	无
uri	MongoDB连接串	是	无
keepAlive	是否保持长连接	否	默认值为true。
maxConnectionIdleTime	连接超时时长	否	整型值，不能为负数，单位为毫秒。默认值为60000。0表示无连接超时限制。
batchSize	每次批量写入的条数	否	整型值，默认值为1024。系统会设定一个大小为batchSize的缓冲条数，当数据的条数达到batchSize时，触发数据的输出。 **说明** 当Checkpoint时间达到时，即使数据未到达batchSize值，也将触发数据的输出。

4.6.3.18. 创建云原生数据仓库AnalyticDB MySQL版3.0结果表

本文为您介绍如何创建云原生数据仓库AnalyticDB MySQL版3.0结果表，以及创建过程中涉及的WITH参数。

注意

- 云原生数据仓库AnalyticDB MySQL版3.0结果表暂不支持注册存储功能。
- 本文仅适用于 `Blink 3.3.0` 及以上版本。
- 云原生数据仓库AnalyticDB MySQL版2.0结果表创建说明，请参见创建云原生数据仓库AnalyticDB MySQL版2.0结果表。
- 云原生数据仓库AnalyticDB MySQL版3.0数据库支持自增主键。如果实时计算Flink版写入数据支持自增主键，则在DDL中不声明该自增字段。例如，ID是自增字段，实时计算Flink版DDL不声明该自增字段，则数据库在一行数据写入过程中会自动填补相关自增字段。

DDL定义

实时计算Flink版支持使用云原生数据仓库AnalyticDB MySQL版3.0版本作为结果表输出。示例代码如下。

```
CREATE TABLE rds_output (
id INT,
len INT,
content VARCHAR,
PRIMARY KEY(id,len)
) WITH (
type='ADB30',
url='jdbc:mysql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>',
tableName='<yourDatabaseTableName>',
userName='<yourDatabaseUserName>',
password='<yourDatabasePassword>'
);
```

实现原理

实时计算Flink版写入云原生数据仓库AnalyticDB MySQL版3.0结果表可以分为以下两个阶段：

1. 将实时计算Flink版每行的结果数据拼接为一行SQL。
2. 在目标数据库执行拼接后的SQL。

WITH参数

参数	注释说明	是否必选	备注
type	connector类型	是	固定值为 `ADB30` 。
url	jdbc连接地址	是	云原生数据仓库AnalyticDB MySQL版数据库地址。示例： `url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'` 。 **说明** • 云原生数据仓库AnalyticDB MySQL版数据库连接信息，请参见URL地址查询。 • databaseName为云原生数据仓库AnalyticDB MySQL版数据库名称。
tableName	表名	是	无
username	账号	是	无
password	密码	是	无
maxRetryTimes	写入重试次数	否	默认值为3。
bufferSize	流入多少条数据后开始去重	否	默认值为1000，表示输入的数据达到1000条则开始输出。 **说明** 需要指定主键后才能生效。

参数	注释说明	是否必选	备注
batchSize	一次批量写入的条数	否	默认值为1000。 说明 需要指定主键后才能生效。
flushIntervalMs	清空缓存的时间间隔	否	单位为毫秒，默认值为3000。表示如果缓存中的数据在等待3秒后，依然没有达到输出条件，系统会自动输出缓存中的所有数据。
ignoreDelete	是否忽略DELETE操作	否	默认值为false，表示支持DELETE功能。
replaceMode	是否采用replace into语法插入数据	否	取值如下： • true（默认）：采用replace into语法插入数据。 • false：采用insert into on duplicate key语法插入数据。 说明 replaceMode参数生效，需要满足以下两个条件： • Blink为3.x及以上版本。 • ADB为3.1.3.5及以上版本。
excludeUpdateColumns	在更新主键值相同的数据时，忽略指定字段的更新。	否	如果要忽略的字段为多个，则需要使用英文逗号（,）分割。例如 `excludeUpdateColumns=column1,column2`。
reserveMilliSecond	TimeStamp类型是否保留毫秒	否	默认值为false，不保留毫秒数值。

4.6.3.19. 创建自定义结果表

本文为您介绍如何创建实时计算Flink版自定义结果表，自定义结果表可以满足您各种差异化的输出需求。

> 注意
> • 本文仅适用于Blink 1.4.5及以上版本。
> • 本文仅适用于独享模式。

搭建环境

您可以通过以下两种方式搭建自定义结果表的开发环境：

- 直接使用示例中的环境。

  为了便于您快速开发业务，实时计算Flink版提供如下自定义结果表示例：

- 实时计算Flink版3.0版本示例。
- 实时计算Flink版2.0版本示例。
- 实时计算Flink版1.0版本示例。

> **说明** 示例中已为您配置对应版本的开发环境，您无需搭建环境。

- 下载JAR包自行搭建环境

> **说明** Maven工程中引用以下依赖包时，Scope设置为 `<scope>provided</scope>` 。

  - 实时计算Flink版3.0版本
    - JAR包下载
      - blink-connector-custom-blink-3.2.1
      - blink-connector-common-blink-3.2.1

    请在POM文件中添加如下信息，完成 `flink-table_2.11` JAR包的自动下载。

```
<profiles>
  <profile>
     <id>allow-snapshots</id>
        <activation><activeByDefault>true</activeByDefault></activation>
     <repositories>
       <repository>
         <id>snapshots-repo</id>
         <url>https://oss.sonatype.org/content/repositories/snapshots</url>
         <releases><enabled>false</enabled></releases>
         <snapshots><enabled>true</enabled></snapshots>
       </repository>
     </repositories>
   </profile>
</profiles>
```

- 依赖包

```
<dependencies>
 <dependency>
   <groupId>com.alibaba.blink</groupId>
   <artifactId>blink-connector-common</artifactId>
   <version>blink-3.2.1-SNAPSHOT</version>
   <scope>provided</scope>
 </dependency>
 <dependency>
   <groupId>com.alibaba.blink</groupId>
   <artifactId>blink-connector-custom</artifactId>
   <version>blink-3.2.1-SNAPSHOT</version>
   <scope>provided</scope>
 </dependency>
 <dependency>
   <groupId>com.alibaba.blink</groupId>
   <artifactId>flink-table_2.11</artifactId>
   <version>blink-3.2.1-SNAPSHOT</version>
   <scope>provided</scope>
 </dependency>
</dependencies>
```

- 实时计算Flink版2.0版本
  - JAR包下载
    - blink-connector-common-blink-2.2.4
    - blink-connector-custom-blink-2.2.4
    - blink-table-blink-2.2.4
    - flink-table_2.11-blink-2.2.4
    - flink-core-blink-2.2.4
  - 依赖包

```
<dependencies>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-table</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table_2.11</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-core</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-connector-common</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba.blink</groupId>
    <artifactId>blink-connector-custom</artifactId>
    <version>blink-2.2.4-SNAPSHOT</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

- 实时计算Flink版1.0版本
  - JAR包下载
    - blink-connector-common-blink-1.4
    - blink-connector-custom-blink-1.4
    - blink-table-blink-1.4
    - flink-core-blink-1.4
    - flink-streaming-scala_2.11-blink-1.4

- 依赖包

```
<dependencies>
  <dependency>
      <groupId>com.alibaba.blink</groupId>
      <artifactId>blink-connector-common</artifactId>
      <version>blink-1.4-SNAPSHOT</version>
      <scope>provided</scope>
  </dependency>
  <dependency>
      <groupId>com.alibaba.blink</groupId>
      <artifactId>blink-connector-custom</artifactId>
      <version>blink-1.4-SNAPSHOT</version>
      <scope>provided</scope>
  </dependency>
  <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-streaming-scala_${scala.binary.version}</artifactId>
      <version>blink-1.4-SNAPSHOT</version>
      <scope>provided</scope>
  </dependency>
  <dependency>
      <groupId>org.apache.flink</groupId>
      <artifactId>flink-core</artifactId>
      <version>blink-1.4-SNAPSHOT</version>
      <scope>provided</scope>
  </dependency>
  <dependency>
      <groupId>com.alibaba.blink</groupId>
      <artifactId>blink-table</artifactId>
      <version>blink-1.4-SNAPSHOT</version>
      <scope>provided</scope>
  </dependency>
</dependencies>
```

接口说明

自定义结果表Class需要继承自定义Sink插件的基类CustomSinkBase，并使用如下方法实现。

```
protected Map<String,String> userParamsMap;// userParamsMap是自定义SQL的WITH语句中定义的键值对，所有的键均为小写。
protected Set<String> primaryKeys;// primaryKeys是自定义的主键字段名。
protected List<String> headerFields;// headerFields是标记为header的字段列表。
protected RowTypeInfo rowTypeInfo;//字段类型和名称。
/**
 * 初始化方法。每次初始建立和Failover的时候会调用一次。
 *
 * @param taskNumber taskNumber为当前节点的编号。
 * @param numTasks   numTasks为Sink节点的总数。
 * @throws IOException
 */
public abstract void open(int taskNumber,int numTasks) throws IOException;
/**
 * close方法，释放资源。
 *
 * @throws IOException
 */
public abstract void close() throws IOException;
/**
 * 处理插入单行数据。
 *
 * @param row
 * @throws IOException
 */
public abstract void writeAddRecord(Row row) throws IOException;
/**
 * 处理删除单行数据。
 *
 * @param row
 * @throws IOException
 */
public abstract void writeDeleteRecord(Row row) throws IOException;
/**
 * 如果进行批量插入，该方法需要把线程中缓存的数据全部刷入下游存储；如果不进行批量插入，可以不使用该方法。
 *
 * @throws IOException
 */
public abstract void sync() throws IOException;
/**
* 返回类名。
*/
public String getName();
```

自定义Redis结果表示例

下载实时计算Flink版3.0版本示例，进入blink_customersink_3x目录，执行 `mvn clean package` 命令，再在实时计算Flink版开发控制台上传刚编译成功后的JAR包blink_customersink_3x/target/blink-customersink-3.x-1.0-SNAPSHOT-jar-with-dependencies.jar，引用资源之后，对于自定义的Sink插件，需要指明 `type = 'custom'` ，并且指明实现接口的Class。

> **注意** 本示例仅作为自定义结果表开发参考，不适合直接作为生产使用。

```
create table in_table(
    kv varchar
)with(
    type = 'random'
);
create table out_table(
    `key` varchar,
    `value` varchar
)with(
    type = 'custom',
    class = 'com.alibaba.blink.customersink.RedisSink',
    -- 1. 可以定义更多自定义参数，在open函数中通过userParamsMap获取。
    -- 2. with参数里key大小写不敏感。在实时计算Flink版中，参数key的值直接处理为全小写。建议您在引用数据存储的DDL中使用小写声明key。
    host = 'r-uf****.redis.rds.aliyuncs.com',
    port = '6379',
    db = '0',
    batchsize = '10',
    password = '<yourHostPassword>'
);
insert into out_table
select
substring(kv,0,4) as `key`,
substring(kv,0,6) as `value`
from in_table;
```

Redis Sink插件的参数说明如下。

参数	说明	是否必填	备注
host	Redis实例内网连接地址（host）	是	无
port	Redis实例端口号	是	无
password	Redis连接密码	是	无
db	Redis Database编号	否	默认值为0，表示db0。
batchsize	每次批量写入的条数	否	默认值为1，表示不批量写入。

4.6.3.20. 创建InfluxDB结果表

本文为您介绍如何创建实时计算Flink版InfluxDB结果表，以及创建结果表时使用的WITH参数和类型映射。

> **注意**
> - InfluxDB暂不支持注册存储功能。
> - 本文仅适用于Blink 3.5.0-hotfix及以上版本。

DDL定义

实时计算Flink版支持使用InfluxDB作为结果输出，示例代码如下。

```
create table stream_test_influxdb(
    `metric` varchar,
    `timestamp` BIGINT,
    `tag_value1` varchar,
    `field_fieldValue1` Double
)with(
    type = 'influxdb',
    endpoint = 'http://service.cn.influxdb.aliyuncs.com:****',
    database = '<yourDatabaseName>',
    batchPutsize = '1',
    username = '<yourDatabaseUserName>',
    password = '<yourDatabasePassword>'
);
```

建表默认格式：

- 第0列：metric（VARCHAR），必填。
- 第1列：timestamp（BIGINT），必填，单位为毫秒。
- 第2列：tag_value1（VARCHAR），必填，最少填写一个。
- 第3列：field_fieldValue1（DOUBLE），必填，最少填写一个。

  写入多个field_fieldValue时，您需要按照如下格式填写。

  ```
  field_fieldValue1 类型,
  field_fieldValue2 类型,
  ...
  field_fieldValueN 类型
  ```

  示例如下。

  ```
  field_fieldValue1 Double,
  field_fieldValue2 INTEGER,
  ...
  field_fieldValueN INTEGER
  ```

> **说明** 结果表中只支持metric、timestamp、tag_*和field_*，不能出现其他的字段。

WITH参数

参数	说明	是否必填	备注
type	结果表类型	是	固定值为InfluxDB。
endpoint	InfluxDB的Endpoint	是	在InfluxDB中，Endpoint是VPC网络地址，例如：*https://localhost:3242*或*http://localhost:8086*。 Endpoint支持HTTP和HTTPS。

参数	说明	是否必填	备注
database	InfluxDB的数据库名	是	例如：db-blink或者blink。
batchPutSize	批量提交的记录条数	否	默认每次提交500个数据点。
username	InfluxDB的用户名	是	需要对写入的数据库有写权限。
password	InfluxDB的密码	是	默认值为0。
retentionPolicy	保留策略	否	为空时，默认填写为每个database的默认保留策略。

类型映射

InfluxDB字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR

4.6.3.21. 创建Phoenix5结果表

本文为您介绍如何创建实时计算Flink版云数据库Phoenix5结果表。

> 注意
>
> - 本文仅适用于实时计算Flink版独享模式。
> - 本文仅适用于Blink 3.4.0以上版本。
> - Phoenix5仅支持5.x版本。
> - Phoenix5是云数据库HBase实例中的一种HBase SQL服务，须在云数据库HBase实例中开启HBase SQL服务后才可以使用Phoenix5。

DDL定义

实时计算Flink版支持使用Phoenix5作为结果输出，示例代码如下。

```
create table US_POPULATION_SINK (
  `STATE` varchar,
  CITY varchar,
  POPULATION BIGINT,
  PRIMARY KEY (`STATE`, CITY)  --主键必填。
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<yourserverUrl>',
  tableName = '<yourTableName>'
);
```

WITH参数

参数	说明	是否必填	备注
type	结果表类型	是	固定值为 `PHOENIX5` 。
serverUrl	Phoenix5的Query Server地址： • 如果Phoenix5是在集群中创建的，则serverUrl是负载均衡服务的URL地址。 • 如果Phoenix5是在单机中创建的，则serverUrl是单机的URL地址。	是	您需要在云数据库HBase实例中开启Hbase SQL服务。 serverUrl格式为http://host:port，其中： • host：Phoenix5服务的域名。 • port：Phoenix5服务的端口号，固定值为8765。
tableName	读取Phoenix5表名。	是	Phoenix5表名格式为SchemaName.TableName，其中： • SchemaName：模式名，可以为空，即不写模式名，仅写表名，表示使用数据库的默认模式。 • TableName：表名。

代码示例

包含Phoenix5结果表的实时计算Flink版作业代码示例如下。

```
create table `source` (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar
) WITH (
   type = 'random'
);
create table sink (
  `id` varchar,
  `name` varchar,
  `age` varchar,
  `birthday` varchar,
  primary key (id)
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<yourserverUrl>',
  tableName = '<yourTableName>'
);
INSERT INTO sink
  SELECT  `id` ,`name` , `age` ,`birthday`
FROM `source`;
```

4.6.3.22. 创建分析型数据库PostgreSQL版结果表

本文为您介绍如何创建分析型数据库PostgreSQL版结果表，以及创建过程中涉及到的WITH参数和类型映射。

> **注意** 本文仅适用于Blink 3.6.0及以上版本。

实现原理

实时计算Flink版写入分析型数据库PostgreSQL结果表可以分为以下两个阶段：

1. 将实时计算Flink版每行的结果数据拼接为一行SQL。
2. 在目标数据库执行拼接后的SQL。

DDL定义

实时计算Flink版支持使用分析型数据库PostgreSQL版作为结果输出。示例代码如下。

```
create table rds_output(
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY(id)
) with (
  type='adbpg',
  url='jdbc:postgresql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>',
  tableName='<yourDatabaseTableName>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
```

WITH参数

参数	说明	是否必填	备注
type	源表类型。	是	固定值为 `adbpg` 。
url	JDBC连接地址。	是	分析型数据库PostgreSQL版数据库的JDBC连接地址。格式为'jdbc:postgresql://<yourNetworkAddress>:<PortId>/<yourDatabaseName>'，其中： • *yourNetworkAddress*：内网地址。 • *PortId*：连接端口。 • *yourDatabaseName*：连接的数据库名称。 示例：url='jdbc:postgresql://gp-xxxxxx.gpdb.cn-chengdu.rds.aliyuncs.com:3432/postgres'
tableName	表名。	是	无。
username	账号。	是	无。
password	密码。	是	无。
maxRetryTimes	写入重试次数。	否	默认值为3。
useCopy	是否采用Copy API写入数据。	否	参数取值如下： • 0（默认值）：采用INSERT方式写入数据。 • 1：采用copy API方式写入数据。
batchSize	一次批量写入的条数。	否	默认值为5000。
exceptionMode	数据写入过程中出现异常时的处理策略。	否	支持以下两种处理策略： • *ignore*（默认值）：忽略出现异常时写入的数据。 • *strict*：数据写入异常时，Failover报错。

参数	说明	是否必填	备注
conflictMode	当主键冲突或唯一索引出现冲突时的处理策略。	否	支持以下三种处理策略： • *ignore*（默认值）：忽略主键冲突，保留之前的数据。 • *strict*：主键冲突时，Failover报错。 • *update*：主键冲突时，更新新到的数据。
targetSchema	Schema名称。	否	默认值为public。
connectionMaxActive	单个task允许的最大连接数。	否	请根据实际的并发task个数，以及目标端数据库允许的最大连接数进行设置。

类型映射

分析型数据库PostgreSQL和实时计算Flink版字段类型对应关系如下。

分析型数据库PostgreSQL版字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
SMALLINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
DOUBLE PRECISION	DOUBLE
TEXT	VARCHAR
TIMESTAMP	DATETIME
DATE	DATE
REAL	FLOAT
DOUBLE PRECISION	DECIMAL
TIME	TIME
TIMESTAMP	TIMESTAMP

4.6.4. 创建数据维表

4.6.4.1. 概述

在维表DDL语法中增加1行PERIOD FOR SYSTEM_TIME的声明，定义维表的变化周期，即可使用标准的CREATE TABLE语法定义实时计算维表。

示例

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME  --定义维表的变化周期。实时计算3.x及以上版本，维表DDL中可以不声明该句，在维表JOIN时，声明FOR SYSTEM_TIME AS OF PROCTIME () 即可。
) with (
  type = 'RDS',
  ...
);
```

> **说明**
>
> - 维表必须指定主键。维表JOIN时，ON的条件必须包含所有主键的等值条件。
> - 目前仅支持源表 `INNER JOIN` 或 `LEFT JOIN` 维表。
> - 维表的唯一键（UK）必须为数据库表中的唯一键。如果维表声明的唯一键不是数据库表的唯一键会产生以下影响：
>   - 维表的读取速度变慢。
>   - 在维表JOIN时，会从第一条数据进行JOIN，在加入Job的过程中，相同KEY的多条记录在数据库中按顺序发生变化，可能导致JOIN结果错误。

INDEX语法

> **说明** 建议在实时计算2.2.7及以上版本使用 `INDEX` 语法。

实时计算2.2以下版本，维表定义要求声明 `PRIMARY KEY` ，这种情况下只能实现一对一连接。为支持一对多连接的需求，引入了 `INDEX` 语法。非 `Cache All` 的维表JOIN通过 `INDEX LOOKUP` 的方式实现一对多连接的需求。

```
CREATE TABLE Persons (
    ID bigint,
    LastName varchar,
    FirstName varchar,
    Nick varchar,
    Age int,
    [UNIQUE] INDEX(LastName,FirstName,Nick), --定义INDEX，不需要指定具体的类型，例如，fulltext或clustered等。
    PERIOD FOR SYSTEM_TIME
) with (
  type='RDS',
  ...
);
```

`UNIQUE INDEX` 表示一对一连接，而 `INDEX` 表示一对多连接。

> **说明**
> - 实时计算2.2.7及以后版本支持 `UNIQUE CONSTRAINT`（`UNIQUE KEY`），实时计算2.2.7以下版本可以使用 `PRIMARY KEY` 的定义。
> - 在生成执行计划时，引擎优先采用 `UNIQUE INDEX` 。即如果DDL中使用INDEX，但JOIN等值连接条件中同时包含 `UNIQUE` 和 `NON-UNIQUE INDEX` 时，优先使用 `UNIQUE INDEX` 查找右表数据。
> - 支持一对多连接的维表类型，例如RDS和MaxCompute。
> - 您可以增加 `maxJoinRows` 参数，表示在一对多连接时，左表一条记录连接右表的最大记录数（默认值为1024）。在一对多连接的记录数过多时，可能会极大的影响流任务的性能，因此您需要增大Cache的内存（ `cacheSize` 限制的是左表key的个数）。
> - 表格存储Tablestore和Hologres维表不支持使用INDEX进行一对多JOIN。

维表、源表和结果表的区别

类别	源表	结果表	维表
是否能驱动计算	是	否	否
是否能读取数据	是，直接读取。	否	是，仅通过源表和维表JOIN读取。
是否能写入数据	否	是	否
是否支持Cache	否	否	是

4.6.4.2. 创建交互式分析Hologres维表

本文为您介绍如何创建交互式分析Hologres维表，以及创建维表时使用的WITH参数、CACHE参数和类型映射。

> **注意**
> - 本文仅适用于Blink 3.6.0及以上版本。如果您的Blink为3.6.0以下的版本，您可以提交工单获取需要的JAR文件，安装使用。
> - 建议您使用Hologres 0.7及以上版本。

什么是交互式分析Hologres

交互式分析Hologres是实时交互分析产品，兼容PostgreSQL协议，与大数据生态紧密连接，支持高并发、低延时实时分析与处理PB级数据，让您轻松使用现有BI（Business Intelligence）工具对数据进行多维分析和业务探索。

使用限制

- 创建Hologres维表时建议选择行存模式，列存模式对于点查场景性能开销较大。

  选择行存模式创建维表时必须设置主键，并且将主键设置为*clustering key*才可以工作。示例语句如下：

```
begin;
create table test(a int primary key, b text, c text, d float8, e int8);
call set_table_property('test', 'orientation', 'row');
call set_table_property('test', 'clustering_key', 'a');
commit;
```

- Hologres维表的主键必须是Blink Join On的字段，Blink Join On的字段也必须是维表完整的主键字段，两者必须完全匹配。
- Hologres Blink Connector的维表功能不支持一对多的输出。
- 不支持读取Hologres分区表的数据。

语法示例

实时计算Flink版支持使用Hologres作为维表，代码示例如下。

```
CREATE TABLE hologres_dim_table(
  id INT,
  len INT,
  content VARCHAR,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME  --定义维表的变化周期。
) WITH (
  type='hologres',
  endpoint='...',
  dbname='...',
  tablename='...',
  username='...',
  password='...'
);
```

WITH参数

参数	描述	是否必选	备注
type	数据库类型。	是	固定值为*hologres*。
endpoint	Hologres端点。	是	无。
tablename	表名称。 说明 如果Schema不为Public时，则tableName需要填写为schema.tableName。	是	无。
dbname	数据库名称。	是	无。
username	用户名称。	是	无。

参数	描述	是否必选	备注
password	密码。	是	无。

CACHE参数

参数	描述	是否必选	示例值
cache	缓存策略。	否	目前交互式分析Hologres维表支持以下两种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。
cacheSize	缓存大小。	否	当缓存策略选择*LRU*时，可以设置缓存大小，默认为*10000*行。
cacheTTLMs	缓存失效时间，单位为毫秒。	否	当缓存策略选择*LRU*时，可以设置缓存失效时间，默认不过期。
partitionedJoin	是否根据JoinKey进行分区。	否	参数的取值如下： • *false*（默认值）：不根据JoinKey进行分区。 • *true*：：根据JoinKey进行分区，将数据分发到各JOIN节点，提高缓存命中率。
async	是否异步读取数据。	否	async参数的取值如下： • *false*（默认值）：同步读取数据。 • *true*：异步读取数据。

类型映射

Hologres字段类型	实时计算Flink版字段类型
INT	INT
INT[]	ARRAY<INT>
BIGINT	BIGINT
BIGINT[]	ARRAY<BIGINT>
REAL	FLOAT
REAL[]	ARRAY<FLOAT>
DOUBLE PRECISION	DOUBLE
DOUBLE PRECISION[]	ARRAY<DOUBLE>

Hologres字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN
BOOLEAN[]	ARRAY<BOOLEAN>
TEXT	VARCHAR
TEXT[]	ARRAY<VARCHAR>
NUMERIC	DECIMAL
DATE	DATE
TIMESTAMP WITH TIMEZONE	TIMESTAMP

代码示例

实时计算Flink版包含Hologres维表的代码示例如下。

```
create table randomSource (a int, b VARCHAR, c VARCHAR) with (type = 'random');
create table test (
  a int,
  b VARCHAR,
  c VARCHAR,
  PRIMARY KEY (a, b), PERIOD FOR SYSTEM_TIME
) with (
  type = 'hologres',
  ...
);
create table print_sink (
  a int,
  b VARCHAR
) with (
  type = 'print',
  `ignoreWrite` = 'false'
);
insert into print_sink
select randomSource.a, test.b from randomSource
LEFT JOIN test FOR SYSTEM_TIME AS OF PROCTIME()
on randomSource.a = test.a and randomSource.b = test.b;
```

4.6.4.3. 创建表格存储Tablestore维表

本文为您介绍如何创建实时计算Flink版表格存储Tablestore维表。

> **注意** 本文仅适用于Blink 1.4.5及以上版本。

什么是表格存储Tablestore

表格存储Tablestore是构建在阿里云飞天分布式系统之上的分布式NoSQL数据存储服务。表格存储通过数据分片和负载均衡技术，实现数据规模与访问并发上的无缝扩展，提供海量结构化数据的存储和实时访问服务。

示例

实时计算Flink版支持表格存储Tablestore作为维表，示例如下。

```
CREATE TABLE ots_dim_table (
 id int,
 len int,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME--维表标识。
) WITH (
 type='ots',
 endPoint='<yourEndpoint>',
 instanceName='<yourInstanceName>',
 tableName='<yourTableName>',
 accessId='<yourAccessId>',
 accessKey='<yourAccessKey>'
);
```

> **说明**
>
> - 在声明维表时，必须要指名主键。
> - 在维表JOIN时，ON条件必须包含所有主键的等值条件。
> - Tablestore的主键即表的Rowkey。

WITH参数

参数	说明	备注
type	维表类型	固定值为 `ots` 。
endPoint	表格存储的实例访问地址	VPC网络环境需要选择实例的VPC Endpoint。
instanceName	表格存储的实例名称	无
tableName	表格存储的数据表名	无
accessId	表格存储读取的AccessKey ID	无
accessKey	表格存储读取的密钥AccessKey Secret	无

CACHE参数

参数	说明	备注
cache	缓存策略	表格存储维表支持以下两种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表来一条数据，系统会先查找Cache，如果没有找到，则去物理维表中查询。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。
cacheSize	缓存大小	当选择LRU缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLMs	缓存超时时间，单位为毫秒。	当选择LRU缓存策略后，可以设置缓存失效的超时时间。

代码示例

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name        VARCHAR,
age           BIGINT
) WITH (
type='datahub'
);
CREATE TABLE phoneNumber(
name VARCHAR,
phoneNumber bigint,
primary key(name),
PERIOD FOR SYSTEM_TIME--维表标识。
)with(
type='ots'
);
CREATE TABLE result_infor(
id bigint,
phoneNumber bigint,
name VARCHAR
)with(
type='rds'
);
INSERT INTO result_infor
SELECT
t.id,
w.phoneNumber,
t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定此声明。
ON t.name = w.name;
```

维表的详细语法请参见维表JOIN语句。

4.6.4.4. 创建云数据库RDS MySQL版维表

本文为您介绍如何创建实时计算云数据库RDS MySQL版维表，以及创建维表时使用的WITH参数、CACHE参数和类型映射。

云数据库RDS MySQL版

RDS MySQL基于阿里巴巴的MySQL源码分支，经过双十一高并发、大数据量的考验，拥有优良的性能。RDS MySQL支持实例管理、账号管理、数据库管理、备份恢复、白名单、透明数据加密以及数据迁移等基本功能。除此之外还提供如下高级功能：

- 专属集群MyBase：是由多台主机（底层服务器，如ECS I2服务器、神龙服务器）组成的集群，相对于全托管数据库，可以满足您更多的需求。
- 只读实例：在对数据库有少量写请求，但有大量读请求的应用场景下，单个实例可能无法承受读取压力，甚至对业务产生影响。为了实现读取能力的弹性扩展，分担数据库压力，您可以创建一个或多个只读实例，利用只读实例满足大量的数据库读取需求，增加应用的吞吐量。
- 读写分离：读写分离功能是在只读实例的基础上，额外提供了一个读写分离地址，联动主实例及其所有只读实例，创建自动的读写分离链路。应用程序只需连接读写分离地址进行数据读取及写入操作，读写分离程序会自动将写入请求发往主实例，而将读取请求按照权重发往各个只读实例。用户只需通过添加只读实例的个数，即可不断扩展系统的处理能力，应用程序上无需做任何修改。
- 什么是数据库代理：数据库独享代理服务是使用独立代理计算资源为当前实例提供代理服务，提供更多高级功能，例如读写分离、短连接优化、事务拆分等。
- 自治服务DAS：针对SQL语句性能、CPU使用率、IOPS使用率、内存使用率、磁盘空间使用率、连接数、锁信息、热点表等，DAS提供了智能的诊断及优化功能，能最大限度发现数据库存在的或潜在的健康问题。DAS的诊断基于单个实例，会提供问题详情及相应的解决方案，为您维护实例带来极大的便利。

RDS MySQL仅支持InnoDB和X-Engine两种存储引擎，具体的功能请参见MySQL功能概览。

使用限制

实时计算Flink版暂不支持通过数据存储功能中存储注册的方式使用RDS MySQL 8.0版本，建议您使用明文方式使用RDS MySQL 8.0版本。数据存储功能详情请参见概述。

语法示例

实时计算Flink版支持使用RDS MySQL版作为维表，示例代码如下。

```
CREATE TABLE rds_dim_table(
 id INT,
 len INT,
 content VARCHAR,
 PRIMARY KEY (id),
 PERIOD FOR SYSTEM_TIME --定义维表的变化周期。
) with (
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
```

说明 声明维表时，必须要指名主键。维表JOIN时，ON条件必须包含所有主键的等值条件。RDS或DRDS的主键可以定义为表的主键或唯一索引列。

WITH参数

参数	说明	是否必填	备注
type	维表类型	是	固定值为 `rds` 。
url	JDBC（Java DataBase Connectivity）连接地址	是	URL的格式为： `jdbc:mysql://<内网地址>/<databaseName>` ，其中databaseName为对应的数据库名称。 内网地址参见如下链接： • Apply for an Internet IP address for RDS
tableName	表名	是	无
userName	用户名	是	无
password	密码	是	无
maxRetryTimes	最大尝试连接次数	否	默认值为10。

CACHE参数

参数	说明	是否必填	备注

参数	说明	是否必填	备注
cache	缓存策略	否	云数据库RDS（DRDS）版维表支持以下三种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查找数据都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。 需要配置相关参数：缓存更新时间间隔（cacheTTLMs），更新时间黑名单（cacheReloadTimeBlackList）。 **说明** 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。
cacheSize	缓存大小	否	当选择 `LRU` 缓存策略后，可以设置缓存大小，默认10000行。
cacheTTLMs	缓存超时时间，单位为毫秒	否	当选择 `LRU` 缓存策略后，可以设置缓存失效的超时时间，默认不过期。当选择 `ALL` 策略，则为缓存加载的间隔时间，默认不重新加载。
cacheReloadTimeBlackList	缓存策略选择 `ALL` 时启用。更新时间黑名单，防止在此时间内做cache更新（例如双11场景）。	否	默认为空。自定义输入格式为 `2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00` 。用逗号（,）来分隔多个黑名单，用箭头（->）来分割黑名单的起始结束时间。
maxJoinRows	主表中每一条数据查询维表时，匹配后最多返回的结果数。	否	默认值为1024。如果您可以预估一条数据对应的维表数据最多为n条，则可以设置maxJoinRows='n'，以确保实时计算匹配处理效率。 **说明** 进行Join时，主表输入一条数据，对应维表匹配后返回的结果总数受该参数限制。

代码示例

包含云数据库RDS版维表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name        VARCHAR,
age           BIGINT
) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='<yourProjectName>',
  topic='<yourTopic>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  startTime='2017-07-21 00:00:00'
);
create table phoneNumber(
name VARCHAR,
phoneNumber BIGINT,
primary key(name),
PERIOD FOR SYSTEM_TIME--定义维表的变化周期。
)WITH(
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
CREATE table result_infor(
id BIGINT,
phoneNumber BIGINT,
name VARCHAR
)WITH(
 type='rds',
 url='<yourDatabaseURL>',
 tableName='<yourDatabaseTableName>',
 userName='<yourDatabaseUserName>',
 password='<yourDatabasePassword>'
);
INSERT INTO result_infor
SELECT
t.id,
w.phoneNumber,
t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定此声明。
ON t.name = w.name;
```

维表详细语法请参见维表JOIN语句。

类型映射

RDS字段类型	实时计算Flink版字段类型
BOOLEAN	BOOLEAN

RDS字段类型	实时计算Flink版字段类型
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DECIMAL	DECIMAL
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
VARBINARY	VARBINARY

4.6.4.5. 创建云数据库HBase版维表

本文为您介绍如何创建实时计算云数据库HBase版维表，以及创建维表时使用的WITH参数和CACHE参数。

> 注意
> - Blink 3.3.0以下版本仅支持HBase企业标准版。
> - Blink 3.3.0及以上版本同时支持HBase企业标准版和HBase性能增强版。
> - Blink 3.5.0及以上版本支持HBase写入主备切换。
> - 云数据库HBase版维表的JOIN语法详情请参见维表JOIN语句。
> - 实时计算HBase维表不支持自建的开源HBase。
> - HBase维表仅支持一个PK（Primary Key）。

DDL定义

- HBase企业标准版

```
CREATE TABLE hbase (
  `key` varchar,
  `name` varchar,
  PRIMARY KEY (`key`), --HBase中的rowkey字段。
  PERIOD FOR SYSTEM_TIME --维表标识。
) with (
  TYPE = 'cloudhbase',
  zkQuorum = '<yourzkQuorum>',
  columnFamily = '<yourColumnFamilyName>',
  tableName = '<yourTableName>'
);
```

- HBase性能增强版

```
CREATE TABLE hbase (
  `key` varchar,
  `name` varchar,
  PRIMARY KEY (`key`), --HBase中的rowkey字段。
  PERIOD FOR SYSTEM_TIME --维表标识。
) with (
  TYPE = 'cloudhbase',
  endPoint = '<host:port>',--HBase增强版的Java API访问地址。
  userName  = 'root', --HBase用户名。
  password = 'root', --HBase密码。
  columnFamily = '<yourColumnFamilyName>',
  tableName = '<yourTableName>'
);
```

- Blink-3.5.0以上HBase性能增强版

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  zkQuorum = '<host:port>', --HBase增强版的Java API访问地址。
  userName  = 'root', --HBase用户名。
  password = 'root', --HBase密码。
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```

- Blink-3.5.0以上支持HBase写入主备切换

```
create table liuxd_user_behavior_test_front (
  row_key varchar,
  from_topic varchar,
  origin_data varchar,
  record_create_time varchar,
  primary key (row_key)
) with (
  type = 'cloudhbase',
  zkQuorum = '<host:port>', --HBase高可用访问地址。
  haClusterID = 'ha-xxx', --HBase高可用实例ID。
  userName  = 'root', --HBase用户名。
  password = 'root', --HBase密码。
  columnFamily = '<yourColumnFamily>',
  tableName = '<yourTableName>',
  batchSize = '500'
);
```

说明

- 在声明维表时，必须要指名主键。
- 在维表进行JOIN时，ON的条件必须包含所有主键的等值条件。示例中HBase中的主键是row_key。
- HBase企业标准版和HBase性能增强版DDL的区别为连接参数不同：
  - HBase企业标准版：`zkQuorum`。
  - HBase性能增强版：`endPoint`。
  - Blink 3.5.0以上标准版和增强版：`zkQuorum`。

WITH参数

参数	说明	是否必填	备注
type	维表类型	是	固定值为`cloudhbase`。
zkQuorum	HBase集群配置的zk地址，是以逗号（,）分隔的主机列表。	是	可以在hbase-site.xml文件中查看hbase.zookeeper.quorum相关配置。 **说明** 仅在HBase企业标准版中生效。
zkNodeParent	集群配置在zk上的路径	否	可以在hbase-site.xml文件中查看hbase.zookeeper.quorum相关配置。 **说明** 仅在HBase企业标准版中生效。

参数	说明	是否必填	备注
endPoint	HBase地域名称	是	可在购买的HBase实例控制台中获取。 **说明** 仅在HBase性能增强版中生效。
userName	用户名	否	**说明** 仅在HBase性能增强版中生效。
password	密码	否	**说明** 仅在HBase性能增强版中生效。
tableName	HBase表名	是	无
columnFamily	列族名	是	仅支持插入同一列族。
maxRetryTimes	最大尝试次数	否	默认值为10次。
partitionedJoin	是否使用joinKey进行分区。	否	默认值为False。在设置partitionedJoin为True时，使用joinKey进行分区，将数据分发到各JOIN节点，提高缓存命中率。
shuffleEmptyKey	是否将上游EMPTY KEY随机发送到下游节点。	否	默认值为True。参数取值如下： • True：如果上游有多个EMPTY KEY，会将所有EMPTY KEY随机发送到各个JOIN节点。 • False：如果上游有多个EMPTY KEY，会将所有EMPTY KEY发送至一个JOIN节点。 **说明** shuffleEmptyKey在partitionedJoin生效后才能使用。

CACHE参数

参数	说明	是否必填	备注

参数	说明	是否必填	备注
cache	缓存策略	否	HBase维表支持以下三种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查找数据都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。 需要配置相关参数：缓存更新时间间隔（cacheTTLMs），更新时间黑名单（cacheReloadTimeBlackList）。 **说明** 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。
cacheSize	缓存大小	否	当选择 `LRU` 缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLMs	缓存超时时间，单位为毫秒。	否	当选择 `LRU` 缓存策略后，可以设置缓存失效的超时时间，默认不过期。当选择 `ALL` 策略，则为缓存加载的间隔时间，默认不重新加载。
cacheReloadTimeBlackList	缓存策略选择 `ALL` 时启用。更新时间黑名单，防止在此时间内做Cache更新（例如双11场景）。	否	默认为空。自定义输入格式如下。 `2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00` 使用逗号（,）分隔多个黑名单，使用箭头（->）分割黑名单的起始和结束时间。
cacheScanLimit	缓存策略选择 `ALL` 时启用。读取全量HBase数据，服务端一次RPC返回给客户端的行数。	否	默认值为100条。

代码示例

包含HBase维表的实时计算作业代码示例如下。

```
create table source (
  id   TINYINT,
  name BIGINT
) with (
  type = 'random'
);
create table dim (
  id   TINYINT,
  score BIGINT
  primary key(id),
  PERIOD FOR SYSTEM_TIME
)with(
  type = 'cloudhbase',
  zkQuorum = '<yourzkQuorum>',
  columnFamily = '<yourColumnFamilyName>',
  tableName = '<yourTableName>'
);
CREATE table result_infor(
  id BIGINT,
  score BIGINT
)with(
  type='rds'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.score
FROM source as t
JOIN dim FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.id = w.id;
```

4.6.4.6. 创建MaxCompute维表

本文为您介绍如何创建实时计算Flink版MaxCompute维表，以及创建维表时使用的WITH参数、CACHE参数和类型映射。

> **注意**
>
> - Blink 2.1.1及以上版本支持MaxCompute维表。
> - 维表的Query语法请参见维表JOIN语句。
> - 使用MaxCompute表作为维表，需要先赋予MaxCompute账号读权限。

DDL定义

```
CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME --维表的标识。
) WITH (
  type = 'odps',
  endPoint = '<YourEndPoint>',
  project = '<YourProjectName>',
  tableName = '<YourtableName>',
  accessId = '<yourAccessKeyId>',
  accessKey = '<yourAccessKeySecret>',
  `partition` = 'ds=2018****',
  cache = 'ALL'
);
```

说明

- 声明维表时，必须要指名主键，MaxCompute维表主键必须具有唯一性，否则会被去重。
- 在维表进行JOIN时，ON条件必须包含所有主键的等值条件。
- partition是关键字，需要使用反引号（`）注释，例如 `` `partition` `` 。
- 如果是分区表，目前不支持将分区列写入到DDL定义中。

WITH参数

参数	说明	是否必填	备注
type	维表类型。	是	固定值为 `odps` 。
endPoint	MaxCompute服务地址。	是	请参见Endpoint。
tunnelEndpoint	MaxCompute Tunnel服务的连接地址。	是	请参见Endpoint。 **说明** VPC环境下为必填。
project	MaxCompute项目名称。	是	无。
tableName	表名。	是	无。
accessId	AccessKey ID。	是	无。
accessKey	AccessKey Secret。	是	无。

参数	说明	是否必填	备注

partition	分区名。	否	• 固定分区
			  ○ 只存在一个分区MaxCompute表
			  例如，如果只存在1个分区列 `ds` ，则 `` `partition` = 'ds=20180905' `` 表示读 `ds=20180905` 分区的数据。
			  ○ 存在多个分区的MaxCompute表
			  例如，如果存在2个分区列 `ds` 和 `hh` ，则 `` `partition`='ds=20180905,hh=*' `` 表示读 `ds=20180905` 分区的数据。
			  说明 分区过滤时需要声明所有分区的值。例如，上述示例中，只声明 `` `partition` = 'ds=20180905' `` ，则不会读取任何分区。
			• 非固定分区
			  ○ Blink 2.2.0及以上版本支持 `` `partition` = 'max_pt()' `` 功能， 即每次加载所有分区列表中字典序最大的分区。
			  ○ Blink 3.2.2及以上版本支持 `` `partition` = 'max_pt_with_done()' `` 功能，即每次加载所有分区列表中字典序最大且伴随有 `.done` 的分区。

maxRowCount	可加载的最大表格数量。	否	默认值为100000。 **说明** 如果您的数据超过100000，需要设置maxRowCount参数。建议设定值比实际值大。

CACHE参数

参数	说明	备注

参数	说明	备注
cache	缓存策略	目前MaxCompute维表仅支持 `ALL` 策略，必须显式声明。 *ALL*策略：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查询都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。需要配置缓存更新时间间隔（cacheTTLMs）和更新时间黑名单（cacheReloadTimeBlackList）参数。 **说明** • 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的至少4倍，具体值与MaxCompute存储压缩算法有关。 • 在使用超大MaxCompute维表时，如果频繁GC（Allocation Failure）导致作业异常，且在增加维表JOIN节点的内存仍无改善的情况下，建议： ◦ Blink 3.6.0及以后版本，设置参数partitionedJoin = 'true'，即打开PartitionedJoin优化。 ◦ 改为支持LRU 缓存策略的KV型维表，例如云数据库Hbase版维表。
cacheSize	缓存大小	可以设置缓存大小，MaxCompute默认缓存值为100000行。
cacheTTLMs	缓存超时时间	单位为毫秒，当cache选择为 `ALL` 策略，则为缓存加载的间隔时间，默认为不重新加载。
cacheReloadTimeBlackList	更新时间黑名单。在缓存策略选择为ALL时，启用更新时间黑名单，防止在此时间内做Cache更新（例如双11场景）。	默认为空，格式为 `2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00` 。分隔符的使用情况如下所示： • 用逗号 `,` 来分隔多个黑名单。 • 用箭头 `->` 来分割黑名单的起始结束时间。
partitionedJoin	当开启PartitionedJoin优化时，每个并发内存里只缓存维表的部分数据，即该并发上需要的缓存数据。	可选，默认值为false，表示每个并发内存里缓存全量维表数据。

代码示例

包含MaxCompute维表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE datahub_input1 (
  id    BIGINT,
  name  VARCHAR,
  age   BIGINT
) with (
  type='datahub'
);
CREATE TABLE odps_dim (
   name VARCHAR,
   phoneNumber BIGINT,
   PRIMARY KEY (name),
   PERIOD FOR SYSTEM_TIME --维表的标识。
) with (
   type = 'odps',
   endPoint = '<yourEndpointName>',
   project = '<yourProjectName>',
   tableName = '<yourTableName>',
   accessId = '<yourAccessId>',
   accessKey = '<yourAccessPassword>',
   `partition` = 'ds=20180905', --动态分区、固定分区请参见WITH参数说明。
   cache = 'ALL'
);
CREATE table result_infor(
  id BIGINT,
  phoneNumber BIGINT,
  name VARCHAR
)with(
  type='print'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN odps_dim FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定此声明。
ON t.name = w.name;
```

类型映射

MaxCompute和Flink全托管字段类型对应关系如下，建议使用该对应关系时进行DDL声明。

MaxCompute	BLINK
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT

MaxCompute	BLINK
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
DATETIME	TIMESTAMP
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
DECIMAL	DECIMAL
BINARY	VARBINARY
STRING	VARCHAR

> **说明** 实时计算Flink版MaxCompute维表仅支持上述MaxCompute字段类型。

常见问题

- `max_pt()` 和 `max_pt_with_done()` 的区别是什么？

  `max_pt()` 仅仅选取所有分区中字典序最大的分区。`max_pt_with_done()` 选取的是所有分区中字典序最大，且伴随有 `.done` 分区的分区。

分区列表
ds=20190101
ds=20190101.done
ds=20190102
ds=20190102.done
ds=20190103

  示例中 `max_pt()` 和 `max_pt_with_done()` 的区别如下：

  - `` `partition`='max_pt_with_done()' `` 匹配的分区是 `ds=20190102` 。
  - `` `partition`='max_pt()' `` ，匹配的分区是 `ds=20190103` 。

  > **说明** 仅Blink 3.3.2及以上版本支持 `` `partition`='max_pt_with_done()' `` 功能。

- Q：任务出现了 `RejectedExecutionException: Task java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTas` 的Failover该怎么处理？

  A：Blink 1.0版本中维表JOIN存在一定的问题，推荐升级到Blink 2.1.1及以上版本。如果需要继续使用原有版本，需要对作业进行暂停恢复操作，根据Failover History中第一条报错信息进行排查。

- Q：公共云的endPoint和tunnelEndpoint是指什么？如果配置错误会产生什么结果？

  A：endPoint和tunnelEndpoint参数说明参见Endpoint。VPC环境中，如果这两个参数配置错误可能会导致任务异常：

  - endPoint配置错误：任务上线停滞在91%的进度。
  - tunnelEndpoint配置错误：任务运行失败。

- Q：作业运行过程中报错ErrorMessage=Authorization Failed [4019], You have NO privilege'ODPS:***'

  A：该报错是因为MaxCompute DDL定义中填写的用户身份信息无法访问MaxCompute。因此，您需要通过阿里云账号、RAM用户账号或RAM角色认证用户身份，详情请参见用户认证。

  如果您有其他疑问，请提交工单咨询，产品名称选择为MaxCompute。

4.6.4.7. 创建云数据库Redis维表

本文为您介绍如何创建云数据库Redis维表，以及创建过程中涉及的WITH参数、CACHE参数、类型映射和代码示例。

> **注意**
> - 本文仅适用于Blink 3.2.2及以上版本。
> - 实时计算Flink版Redis维表仅支持引用Redis数据存储中STRING类型的数据。
> - 实时计算Flink版Redis维表支持自建Redis服务。

语法示例

实时计算Flink版支持使用Redis作为数据存储维表。Redis维表语法示例如下。

```
CREATE TABLE white_list (
  id VARCHAR,
  name VARCHAR,
  PRIMARY KEY (id),  --Redis中的Row Key字段。
  PERIOD FOR SYSTEM_TIME --维表标识。
) WITH (
  type = 'redis',
  host = '<yourHostName>',
  port = '<yourPort>',
  password = '<yourPassword>',
  dbNum = '<yourDatabaseNumber>'
);
```

> **说明**
> - Redis维表必须声明且只能声明一个主键。
> - 维表JOIN时，ON条件必须包含所有主键的等值条件。
> - Redis维表仅支持声明两个字段，且字段类型必须为VARCHAR。

WITH参数

参数	说明	是否必填	备注
type	维表类型	是	固定值为 `redis` 。
host	Redis连接地址	是	无
port	Redis连接端口	否	默认值为6379。
dbNum	选择操作的数据库	否	默认值为0。
password	Redis密码	否	默认值为空，不进行权限验证。
hashName	Hash模式下的Hash Key名称	否	默认值为空，实时计算Flink版从Redis中读取STRING类型的数据。 通常，Redis维表中的数据类型为STRING类型，即 `key-value` 对。如果设置hashName参数，则Redis维表中的数据类型为HASHMAP类型，即 `key-{field-value}` 对，其中： • key为hashName参数值。 • field为您在CREATE TABLE中指明的key参数值。 • value为key对应的赋值，和STRING类型 `key-value` 中value语义相同。

CACHE参数

参数	说明	是否必填	备注
cache	缓存策略	否	云数据库Redis维表支持以下两种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表来一条数据，系统会先查找Cache，如果没有找到，则去物理维表中查询。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。
cacheSize	缓存大小	否	选择 `LRU` 缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLMs	缓存超时时长	否	默认缓存不超时，单位为毫秒。可选LRU缓存策略，即设置缓存失效的超时时长。
cacheEmpty	是否缓存空结果	否	默认值为true。

类型映射

Redis和实时计算Flink版字段类型对应关系如下。建议您使用该对应关系进行DDL声明。

Redis字段类型	实时计算Flink版字段类型
STRING	VARCHAR

代码示例

包含Redis维表的实时计算Flink版作业代码示例如下。

```
CREATE TABLE event (
  id VARCHAR,
  data VARCHAR) with (
  type = 'random'
);
CREATE TABLE white_list (
  id VARCHAR,
  name VARCHAR,
  PRIMARY KEY (id),  --Redis中的Row Key字段。
  PERIOD FOR SYSTEM_TIME --维表的标识。
) WITH (
  type = 'redis',
  host = '<yourRedisHost>',
  password = '<yourRedisPassword>'
);
SELECT e.*, w.*
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id;
```

4.6.4.8. 创建Elasticsearch维表

本文为您介绍如何创建实时计算Flink版Elasticsearch（ES）维表，以及创建维表时使用的WITH参数和CACHE参数。

> **注意** 本文仅适用于Blink 3.2.2及以上版本。

DDL定义

实时计算Flink版支持使用ES作为维表，示例代码如下。

```
 CREATE TABLE es_stream_sink(
  field1 LONG,
  field2 VARBINARY,
  field3 VARCHAR,
  PRIMARY KEY(field1),
  PERIOD FOR SYSTEM_TIME
) WITH (
  type ='elasticsearch',
  endPoint = '<yourEndPoint>',
  accessId = '<yourUsername>',
  accessKey = '<yourPassword>',
  index = '<yourIndex>',
  typeName = '<yourTypeName>'
);
```

> **说明** ES维表支持根据ES的PRIMARY KEY进行PRIMARY KEYUPDATE，且PRIMARY KEY只能为1个字段。

WITH参数

参数	说明	默认值	是否必选
type	维表类型	elasticsearch	是
endPoint	Server地址，例如：*http://127.0.0.1:9211*。	无	是
accessId	创建ES时的登录名。 > **说明** 如果您通过Kibana插件操作ES，请填写Kibana登录ID。	无	是
accessKey	创建ES时的登录密码 。 > **说明** 如果您通过Kibana插件操作ES，请填写Kibana登录密码。	无	是
index	索引名称，类似于数据库Database的名称。	无	是
typeName	Type名称，类似于数据库的Table名称。	无	是
maxRetryTimes	异常重试次数	30	否
timeout	读取超时时长，单位为毫秒。	600000	否
discovery	是否开启节点发现。如果开启，客户端每5分钟刷新一次Server List。	false	否

参数	说明	默认值	是否必选
compression	是否使用GZIP压缩Request Bodies。	true	否
multiThread	是否开启JestClient多线程。	true	否

CACHE参数

参数	说明	备注
cache	缓存策略	• *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表来一条数据，系统会先查找Cache，如果没有找到，则去物理维表中查询。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查询都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。
cacheSize	缓存大小	选择`LRU`缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLMs	缓存更新时间间隔	默认缓存不超时，单位为毫秒。不同缓存策略下的功能如下： • LRU：设置缓存失效的超时时长。 • ALL：设置缓存加载的间隔时长，默认不重新加载。

4.6.4.9. 创建Phoenix5维表

本文为您介绍如何创建实时计算Flink版Phoenix5维表，以及创建维表时使用的WITH参数和CACHE参数。

> **注意** 仅Blink 3.4.0以上版本支持Phoenix5维表。

语法示例

```
create table US_POPULATION_DIM (
  `STATE` varchar,
  CITY varchar,
  POPULATION BIGINT,
  PRIMARY KEY (`STATE`, CITY),
  PERIOD FOR SYSTEM_TIME
) WITH (
  type = 'PHOENIX5',
  serverUrl = '<YourServerUrl>',
  tableName = '<YourTableName>'
);
```

WITH参数

参数	说明	是否必填	备注
type	维表类型	是	固定值为 `PHOENIX5` 。
serverUrl	Phoenix5的Query Server地址： • 如果Phoenix5是在集群中创建的，则serverUrl是负载均衡服务的URL地址。 • 如果Phoenix5是在单机中创建的，则serverUrl是单机的URL地址。	是	您需要在云数据库HBase实例中开启Hbase SQL服务。 serverUrl格式为http://host:port，其中： • host：Phoenix5服务的域名。 • port：Phoenix5服务的端口号，固定值为8765。
tableName	Phoenix5表名	是	Phoenix5表名格式为SchemaName.TableName，其中： • SchemaName：模式名，可以为空，即不写模式名，仅写表名，表示使用数据库的默认模式。 • TableName：表名。

CACHE参数

参数	说明	是否必填	备注

参数	说明	是否必填	备注
cache	缓存策略	否	目前Phoenix5维表支持以下三种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查询都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。 需要配置相关参数：缓存更新时间间隔（cacheTTLMs），更新时间黑名单（cacheReloadTimeBlackList）。 **说明** 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。
cacheSize	缓存大小	否	选择*LRU*缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLMs	缓存超时时间，单位为毫秒。	否	当选择*LRU*缓存策略后，可以设置缓存失效的超时时间，默认不过期。当选择*ALL*策略，则为缓存加载的间隔时间，默认不重新加载。
cacheReloadTimeBlackList	更新时间黑名单。在缓存策略选择为ALL时，启用更新时间黑名单，防止在此时间内做Cache更新（例如双11场景）。	否	可选，默认空，格式为 '2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00'。用逗号（,）来分隔多个黑名单，用箭头（->）来分割黑名单的起始或结束时间。

代码示例

```
CREATE TABLE datahub_input1 (
  id  BIGINT,
  name  VARCHAR,
  age   BIGINT
) WITH (
  type='datahub'
);
create table phoneNumber(
  name VARCHAR,
  phoneNumber BIGINT,
  primary key(name),
  PERIOD FOR SYSTEM_TIME--定义维表的变化周期。
)with(
  type='PHOENIX5'
);
CREATE table result_infor(
  id BIGINT,
  phoneNumber BIGINT,
  name VARCHAR
)with(
  type='rds'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定该声明。
ON t.name = w.name;
```

4.6.4.10. 创建云原生数据仓库AnalyticDB MySQL版3.0维表

本文为您介绍如何创建云原生数据仓库AnalyticDB MySQL版3.0维表、以及创建维表时使用的WITH参数和CACHE参数。

> **注意** 本文仅适用于Blink-3.5.0-hotfix及以上版本。

语法示例

```
CREATE TABLE dim_ads(
    `name` VARCHAR,
     id VARCHAR,
     PRIMARY KEY (`name`),
     PERIOD FOR SYSTEM_TIME
)with(
     type='ADB30',
     url='jdbc:mysql://<内网地址>/<databaseName>',
     tableName='xxx',
     userName='xxx',
     password='xxx'
);
```

说明

- 在声明一个维表时，必须指明主键。
- 在维表进行JOIN时，ON条件必须包含所有主键的等值条件。
- 云原生数据仓库AnalyticDB MySQL版的主键可以定义为表的主键或唯一索引列。

WITH参数

参数	说明	是否必选	备注
type	维表类型。	是	固定值为ADB30。
url	云原生数据仓库AnalyticDB MySQL版数据库地址。	是	云原生数据仓库AnalyticDB MySQL版数据库地址。示例：`url='jdbc:mysql://databaseName****-cn-shenzhen-a.ads.aliyuncs.com:10014/databaseName'`。 **说明** • 云原生数据仓库AnalyticDB MySQL版数据库连接信息，请参见注册云原生数据仓库AnalyticDB MySQL版。 • databaseName：云原生数据仓库AnalyticDB MySQL版数据库名称。
tableName	表名。	是	无。
userName	用户名。	是	无。
password	密码。	是	无。
maxRetryTimes	写入重试次数。	否	默认值为3。

CACHE参数

参数	说明	是否必填	备注
cache	缓存策略	否	目前云原生数据仓库AnalyticDB MySQL版3.0支持以下三种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表来一条数据，系统会先查找Cache，如果没有找到，则去物理维表中查询。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查询都会通过Cache进行。如果在Cache中无法找到数据，则KEY不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。 需要配置相关参数：缓存更新时间间隔（cacheTTLMs），更新时间黑名单（cacheReloadTimeBlackList）。 **说明** • 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。 • 对于数据量比较大的维表，选择CACHE ALL时，可能会出现OOM或者Full GC耗时很久的情况，针对这个问题，可以选择以下两种解决方式： ◦ 对于支持Cache All策略的维表，开启PartitionedJoin优化。3.6.0版本之前，每个并发默认加载维表全量数据。3.6.0版本之后，CACHE ALL策略支持PartitionedJoin优化。开启PartitionJoin优化后，每个并发只缓存自己并发所需要的数据。 ◦ 使用HBase或者RDS等Key-Value类型的维表。
cacheSize	缓存大小	否	当选择*LRU*缓存策略后，可以设置缓存大小，默认为10000行。
cacheTTLMs	缓存更新时间间隔。系统会根据您设置的缓存更新时间间隔，重新加载一次维表中的最新数据，保证源表能JOIN到维表的最新数据。	否	单位为毫秒。默认不设置此参数，表示不重新加载维表中的新数据。

参数	说明	是否必填	备注
cacheReloadTimeBlackList	更新时间黑名单。在缓存策略选择为ALL时，启用更新时间黑名单，防止在此时间内做Cache更新（例如双11场景）。	否	可选，默认空，格式为 '2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00' 。其中分割符使用情况如下： • 用逗号（,）来分隔多个黑名单。 • 用箭头（->）来分割黑名单的起始结束时间。
partitionedJoin	是否开启partitionedJoin。在开启partitionedJoin优化时，主表会在关联维表前，先按照Join KEY进行Shuffle，这样做有以下优点： • 在缓存策略为*LRU*时，可以提高缓存命中率。 • 在缓存策略为*ALL*时，节省内存资源，因为每个并发只缓存自己并发所需要的数据。	否	默认情况下为false，表示不开启partitionedJoin。 **说明** 使用partitionedJoin优化前，需要您手动设置partitionedJoin = 'true'。
maxJoinRows	主表中每一条数据查询维表时，匹配后最多返回的结果数。	否	默认值为1024。如果您可以预估一条数据对应的维表数据最多为n条，则可以设置maxJoinRows='n'，以确保实时计算匹配处理效率。 **说明** 进行Join时，主表输入一条数据，对应维表匹配后返回的结果总数受该参数限制。

代码示例

```
CREATE TABLE datahub_input1 (
  id      BIGINT,
  name    VARCHAR,
  age     BIGINT
) WITH (
  type='datahub'
);
create table phoneNumber (
  name VARCHAR,
  phoneNumber BIGINT,
  primary key(name),
  PERIOD FOR SYSTEM_TIME--维表标识。
) with (
  type='ADB30'
);
CREATE table result_infor (
  id BIGINT,
  phoneNumber BIGINT,
  name VARCHAR
) with (
  type='rds'
);
INSERT INTO result_infor
SELECT
  t.id,
  w.phoneNumber,
  t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w --维表JOIN时必须指定该声明。
ON t.name = w.name;
```

4.6.4.11. 创建Oracle维表

本文为您介绍如何创建Oracle维表以及创建过程中涉及到的WITH参数、类型映射和属性字段等。

DDL定义

```
CREATE TABLE oracle_dim(
  employee_id BIGINT,
  phone_number BIGINT,
  dollar DOUBLE,
  PRIMARY KEY (employee_id)
) WITH (
  type = 'oracle_dim',
  url = '<yourUrl>',
  userName = '<yourUserName>',
  password = '<yourPassword>',
  tableName = '<yourTableName>',
  cache = 'ALL'
);
```

WITH 参数

参数	描述	是否必选	示例值
type	维表类型	是	固定值为oracle_dim。
url	JDBC的Oracle地址	是	`jdbc:oracle:thin:@ip:port:sid`
userName	数据库的用户名	是	无
password	数据库的密码	是	无
tableName	表名称	是	无
maxRetryTimes	读取维表异常重试的最大次数	否	默认值为10。

CACHE参数

参数	描述	是否必选	示例值
cache	缓存策略	否	目前Oracle维表支持以下三种缓存策略： • *None（默认值）*：无缓存。 • *LRU*：缓存维表里的部分数据。源表的每条数据都会触发系统先在Cache中查找数据，如果没有找到，则去物理维表中查找。 需要配置相关参数：缓存大小（cacheSize）和缓存更新时间间隔（cacheTTLMs）。 • *ALL*：缓存维表里的所有数据。在Job运行前，系统会将维表中所有数据加载到Cache中，之后所有的维表查找数据都会通过Cache进行。如果在Cache中无法找到数据，则关键健不存在，并在Cache过期后重新加载一遍全量Cache。 适用于远程表数据量小且MISS KEY（源表数据和维表JOIN时，ON条件无法关联）特别多的场景。 需要配置相关参数：缓存更新时间间隔（cacheTTLMs），更新时间黑名单（cacheReloadTimeBlackList）。 **说明** 因为系统会异步加载维表数据，所以在使用CACHE ALL时，需要增加维表JOIN节点的内存，增加的内存大小为远程表数据量的两倍。
cacheSize	缓存大小，即缓存多少行数据。	否	当缓存策略选择*LRU*时，可以设置缓存大小，默认值为*10000*行。

参数	描述	是否必选	示例值
cacheTTLMs	缓存超时时间，单位为毫秒。	否	• 当缓存策略选择*LRU*时，可以设置缓存失效时间，默认不过期。 • 当缓存策略选择*ALL*时，缓存失效时间为缓存重新加载的间隔时间，默认不重新加载。
cacheReloadTimeBlackList	更新时间黑名单。在缓存策略选择为ALL时，启用更新时间黑名单，防止在此时间内做Cache更新（例如双11场景）。	否	默认空，格式为 '2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00' 。分割符如下： • 用逗号（,）来分隔多个黑名单。 • 用箭头（->）来分割黑名单的起始结束时间。
maxJoinRows	一对多连接时，左表一条记录连接右表的最大记录数。	否	默认值为1024。 **说明** 一对多连接的记录数过多时，需要调cache的内存。因为cacheSize限制的是左表key个数，单条左表记录对应的右表记录较多时，可能会极大地影响流任务的性能。

类型映射

Oracle字段类型	实时计算Flink字段类型
• CHAR • VARCHAR • VARCHAR2	VARCHAR
FLOAT	DOUBLE
NUMBER	BIGINT
DECIMAL	DECIMAL

代码示例

```
CREATE TABLE oracle_source (
  employee_id BIGINT,
  employee_name VARCHAR,
  employee_age INT
) WITH (
  type ='random'
);
CREATE TABLE oracle_dim (
  employee_id BIGINT,
  phone_number BIGINT,
  dollar DOUBLE,
  PRIMARY KEY (employee_id)
) WITH (
  type = 'oracle_dim',
  url = '<yourUrl>',
  userName = '<yourUserName>',
  password = '<yourPassword>',
  tableName = '<yourTableName>',
  cache = 'ALL'
);
CREATE TEMPORARY TABLE oracle_sink (
  employee_id BIGINT,
  phone_number BIGINT,
  employee_name VARCHAR
) WITH (
  type = 'oracle',
  url = '<yourUrl>',
  userName = '<yourUserName>',
  password = '<yourPassword>',
  tableName = '<yourTableName>'
);
INSERT INTO oracle_sink
SELECT t.employee_id, w.phone_number, t.employee_name
FROM oracle_source as t JOIN oracle_dim FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.employee_id = w.employee_id;
```

4.7. DML语句

4.7.1. EMIT语句

EMIT语句可以使QUERY根据不同场景，定义不同的输出策略，从而达到控制延迟或提高数据准确性的效果。

> 注意　EMIT语句仅支持实时计算2.0以上版本。

使用限制

- EMIT策略只支持TUMBLE和HOP窗口，暂不支持SESSION窗口。
- 如果一个Job有多个输出，则多个输出的EMIT需要定义成相同策略，后续会支持不同策略。
- EMIT语法还不能用来配置minibatch的allowLateness，后续计划使用EMIT策略来声明allowLateness。

什么是EMIT策略

EMIT策略是指在Flink SQL中，QUERY根据不同场景选择不同的输出策略（例如最大延迟时长）。传统的ANSI SQL语法不支持该类输出策略。例如，1小时的时间窗口，窗口触发之前希望每分钟都能看到最新的结果，窗口触发之后希望不丢失迟到一天内的数据。如果1小时窗口内的统计结果无变化，则不更新输出结果；如果1小时窗口内的统计结果有变化，则更新输出结果。

针对这类场景，实时计算抽象出EMIT语法，并扩展到SQL语法。以下为不同场景下EMIT策略的示例：

- 窗口结束之前，按1分钟延迟输出，窗口结束之后无延迟输出。

```
EMIT
  WITH DELAY '1'MINUTE BEFORE WATERMARK,
  WITHOUT DELAY AFTER WATERMARK
```

- 窗口结束之前不输出，窗口结束之后无延迟输出。

```
EMIT WITHOUT DELAY AFTER WATERMARK
```

- 全局都按1分钟的延迟输出（您可以启用minibatch参数来增加延迟）。

```
EMIT WITH DELAY '1'MINUTE
```

- 窗口结束之前按1分钟延迟输出。

```
EMIT WITH DELAY '1'MINUTE BEFORE WATERMARK
```

EMIT语法的用途

EMIT语法能够实现以下两种功能：

- 控制延迟：针对窗口，设置窗口触发之前的EMIT输出频率，减少结果输出延迟。
- 提高数据精确性：不丢弃窗口触发之后的迟到的数据，修正输出结果。

> **说明** 选择EMIT策略时，请权衡业务复杂程度和资源消耗情况。越低的输出延迟、越高的数据精确性，需要提供越高的计算开销。

EMIT语法

EMIT语法定义在INSERT INTO输出语句中，定义输出结果的策略。当INSERT INTO中未配置EMIT语法时，保持原有默认行为，即只在WATERMARK触发时，输出一个窗口结果。

> **说明** EMIT语句只能置于INSERT INTO语句中的所有QUERY语句之后，不能置于VIEW语句中。

```
INSERT INTO tableName
<Query>
EMIT strategy [, strategy]*
strategy ::= {WITH DELAY timeInterval | WITHOUT DELAY}
                [BEFORE WATERMARK |AFTER WATERMARK]
timeInterval ::='string' timeUnit
```

参数	说明
`WITH DELAY`	可延迟输出，即按指定时长间隔输出。
`WITHOUT DELAY`	不可以延迟输出，即每来一条数据就输出。
`BEFORE WATERMARK`	窗口结束之前的策略配置，即WATERMARK触发之前。
`AFTER WATERMARK`	窗口结束之后的策略配置，即WATERMARK触发之后。 **说明** 如果配置了AFTER WATERMARK策略，需要使用明文方式声明 `blink.state.ttl.ms` ，标识最大延迟时长。

其中 `strategy` 的配置方式包括以下几种：

- 配置为一个BEFORE。
- 配置为一个AFTER。
- 配置为一个BEFORE和一个AFTER。

说明 `strategy` 不支持同时配置为多个BEFORE或者多个AFTER。

生命周期

AFTER策略允许接收迟到的数据，窗口的状态（State）允许保留一定时长，等待迟到的数据。这段保留的时长称为生命周期TTL。运用AFTER策略后，通过明文声明 `blink.state.ttl.ms` 参数，您可以设置状态允许的生命周期。例如， `blink.state.ttl.ms=3600000` 表示状态允许保留超时时长为1小时内的数据，超时时长大于1小时的数据不被录入状态。

EMIT语法示例

窗口区间为1小时的滚动窗口 `tumble_window` 的语法示例如下。

```
CREATE VIEW tumble_window AS
SELECT
  `id`,
  TUMBLE_START(rowtime, INTERVAL '1' HOUR) as start_time,
  COUNT(*) as cnt
FROM source
GROUP BY `id`, TUMBLE(rowtime, INTERVAL '1' HOUR);
```

默认 `tumble_window` 的输出需要等到1小时结束才能显示。如果您需要尽早看到窗口的结果（即使是不完整的结果），例如每分钟看到最新的窗口结果，可以添加如下语句。

```
INSERT INTO result
SELECT * FROM tumble_window
EMIT WITH DELAY '1' MINUTE BEFORE WATERMARK; --窗口结束之前，每隔1分钟输出一次更新结果。
```

默认 `tumble_window` 会忽略并丢弃窗口结束后到达的数据，如果您需要将窗口结束后1天到达的数据统计进入结果，并且需要每接收1条数据后立刻更新结果，可以添加如下语句。

```
INSERT INTO result
SELECT * FROM tumble_window
EMIT WITH DELAY '1' MINUTE BEFORE WATERMARK,
     WITHOUT DELAY AFTER WATERMARK;  --窗口结束后，每收到一条数据输出一次更新结果。
```

此外，您还需要在**作业参数**中配置blink.state.ttl.ms = 86400000（增加1天状态生命周期）。

DELAY概念

EMIT策略中的 `DELAY` 指的是用户可接受的数据延迟时长，该延迟是指从用户的数据进入实时计算，到看到结果数据（从实时计算系统输出）的时间（Event Time或Processing Time）。延迟的计算基于系统时间。动态表（流式数据在实时计算内部的存储）中的数据发生变化的时间和结果表（实时计算外部的存储）中显示新记录的时间的间隔，称为延迟。

假设，实时计算系统的处理耗时是0，则在流式数据积攒和Window等待窗口数据的过程可能会导致延迟。如果您指定了最多延迟30秒，则30秒可用于流式数据的积攒。如果Query是1小时的窗口，则最多延迟30秒的含义是每隔30秒更新结果数据。

- 配置 `EMIT WITH DELAY '1' MINUTE`

  对于Group By聚合，系统会在1分钟内积攒流式数据。如果有Window且Window的Size大于1分钟，Window就每隔1分钟更新一次结果数据。如果Window的Size小于1分钟，因为窗口依靠Watermark的输出就能保证Latency SLA，所以系统就会忽略这个配置。

- 配置 `EMIT WITHOUT DELAY`

  对于Group By聚合，不会启用minibatch参数来增加延迟，每来一条数据都会触发计算和输出。对于Window函数，也是每来一条数据都触发计算和输出。

4.7.2. INSERT INTO语句

本文为您介绍在实时计算Flink版中如何使用INSERT INTO语句及使用限制。

操作约束

表类型	使用限制
源表	只能引用（FROM），不可执行INSERT。
维表	只能引用（JOIN），不可执行INSERT。
结果表	仅支持INSERT操作。
视图	只能引用（FROM）。

语法

```
INSERT INTO tableName
[ (columnName[ , columnName]*) ]
queryStatement;
```

示例

```
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;
INSERT INTO Orders(z,v)
SELECT c,d FROM OO;
```

说明

- 单个实时计算Flink版作业支持在一个SQL作业中包含多个DML操作，同样也允许包含多个数据源、数据目标端和维表。例如，一个作业文件中包含两段业务上完全独立的SQL，分别写到不同的数据目标端。
- 实时计算Flink版不支持单独的SELECT查询，必须在CREATE VIEW或INSERT INTO内才能操作。
- INSERT INTO支持UPDATE更新。例如，向设置了主键字段的RDS结果表中插入一个KEY值。如果这个KEY值存在就更新，如果不存在就插入一条新的KEY值。

4.8. QUERY语句

4.8.1. SELECT语句

SELECT语句用于从表中选取数据。

语法

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression;
```

测试数据

t

a（VARCHAR）	b（INT）	c（DATE）
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

简单查询

- 测试语句

  ```
  SELECT * FROM t;
  ```

- 测试结果

a（VARCHAR）	b（INT）	c（DATE）
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

重命名查询

- 测试语句

```
SELECT a, c AS d FROM t;
```

- 测试结果

a（VARCHAR）	d（DATE）
a1	1990-02-20
b1	2018-05-12
c1	2010-06-14
a1	2016-04-05

去重查询

- 测试语句

```
SELECT DISTINCT a FROM t;
```

- 测试结果

a（VARCHAR）
a1
b1
c1

子查询

普通的SELECT是从几张表中读数据，例如 `SELECT column_1, column_2 … FROM table_name` ，但查询的对象也可以是另外一个SELECT操作。

> **说明** 当查询的对象是另一个SELECT操作时，必须为子查询加别名。

- 测试语句

```
INSERT INTO result_table
SELECT * FROM
               (SELECT   t.a,
                         sum(t.b) AS sum_b
                FROM     t1 t
                GROUP BY t.a
               ) t1
WHERE  t1.sum_b > 100;
```

- 测试结果

a（VARCHAR）	b（INT）
a1	211
b1	120
a1	257

> **说明** 此结果为调试结果，会显示出计算过程。如果您的结果表是DataHub、消息队列Kafka或消息队列MQ等，正式上线也会显示过程数据。但如果您的结果表是云数据RDS等关系型数据库，正式上线，主键相同的记录显示为一条数据。

4.8.2. WHERE语句

WHERE语句可用于对SELECT语句中的数据进行筛选。

语法

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ];
```

下面的运算符可在WHERE语句中使用。

操作符	描述
=	等于
<>	不等于
>	大于
>=	大于等于
<	小于
<=	小于等于

示例

- 测试数据

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing
Changan Street	shanghai

- 测试语句

```
SELECT * FROM XXXX WHERE City='Beijing';
```

- 测试结果

Address	City
Oxford Street	Beijing
Fifth Avenue	Beijing

4.8.3. HAVING语句

在使用聚合函数时，您需要添加HAVING语句，以实现与WHERE语句一样的过滤效果。

语法

```
SELECT [ ALL | DISTINCT ]{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ]
[ GROUP BY { groupItem [, groupItem ]* } ]
[ HAVING booleanExpression ];
```

示例

- 测试数据

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- 测试语句

```
SELECT Customer,SUM(OrderPrice) FROM XXX
GROUP BY Customer
HAVING SUM(OrderPrice)<2000;
```

- 测试结果

Customer	SUM（OrderPrice）
Carter	1700

4.8.4. GROUP BY语句

GROUP BY语句用于根据一个或多个列对结果集进行分组。

语法

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ GROUP BY { groupItem [, groupItem ]* } ];
```

示例

- 测试数据（T1）

Customer	OrderPrice
Bush	1000
Carter	1600
Bush	700
Bush	300
Adams	2000
Carter	100

- 测试语句

```
SELECT Customer,SUM(OrderPrice) FROM T1
GROUP BY Customer;
```

- 测试结果

Customer	SUM(OrderPrice)
Bush	2000
Carter	1700

Customer	SUM(OrderPrice)
Adams	2000

4.8.5. JOIN语句

实时计算的JOIN和传统批处理JOIN的语义一致，都用于将两张表关联起来。区别为实时计算关联的是两张动态表，关联的结果也会动态更新，以保证最终结果和批处理结果一致。

语法

```
tableReference [, tableReference ]* | tableexpression
[ LEFT ] JOIN tableexpression [ joinCondition ];
```

- tableReference：表名称。
- tableexpression：表达式。
- joinCondition：JOIN条件。

> 注意
> - 只支持等值连接，不支持非等值连接。
> - 只支持INNER JOIN和LEFT OUTER JOIN两种JOIN方式。

Orders JOIN Products表的数据示例

- 测试数据 Orders

rowtime	productId	orderId	units
10:17:00	30	5	4
10:17:05	10	6	1
10:18:05	20	7	2
10:18:07	30	8	20
11:02:00	10	9	6
11:04:00	10	10	1
11:09:30	40	11	12
11:24:11	10	12	4

Products

productId	name	unitPrice
30	Cheese	17
10	Beer	0.25
20	Wine	6
30	Cheese	17
10	Beer	0.25
10	Beer	0.25
40	Bread	100
10	Beer	0.25

- 测试语句

```
SELECT o.rowtime, o.productId, o.orderId, o.units,p.name, p.unitPrice
FROM Orders AS o
JOIN Products AS p
ON o.productId = p.productId;
```

- 测试结果

o.rowtime	o.productId	o.orderId	o.units	p.name	p.unitPrice
10:17:00	30	5	4	Cheese	17
10:17:05	10	6	1	Beer	0.25
10:18:05	20	7	2	Wine	6
10:18:07	30	8	20	Cheese	17
11:02:00	10	9	6	Beer	0.25
11:04:00	10	10	1	Beer	0.25
11:09:30	40	11	12	Bread	100
11:24:11	10	12	4	Beer	0.25

datahub_stream1 JOIN datahub_stream2表的数据示例

- 测试数据 datahub_stream1

a（BIGINT）	b（BIGINT）	c（VARCHAR）
0	10	test11

a（BIGINT）	b（BIGINT）	c（VARCHAR）
1	10	test21

datahub_stream2

a（BIGINT）	b（BIGINT）	c（VARCHAR）
0	10	test11
1	10	test21
0	10	test31
1	10	test41

- 测试语句

```
SELECT s1.c,s2.c
FROM datahub_stream1 AS s1
JOIN datahub_stream2 AS s2
ON s1.a =s2.a
WHERE s1.a = 0;
```

- 测试结果

s1.c（VARCHAR）	s2.c（VARCHAR）
test11	test11
test11	test31

4.8.6. 维表JOIN语句

对于每条流式数据，可以关联一个外部维表数据源，为实时计算Flink版提供数据关联查询。

> **说明** 维表是一张不断变化的表，在维表JOIN时，需指明该条记录关联维表快照的时刻。维表JOIN仅支持对当前时刻维表快照的关联，未来会支持关联左表rowtime所对应的维表快照。关于维表的详细介绍，请参见概述。

维表JOIN语法

```
SELECT column-names
FROM table1  [AS <alias1>]
[LEFT] JOIN table2 FOR SYSTEM_TIME AS OF PROCTIME() [AS <alias2>]
ON table1.column-name1 = table2.key-name1;
```

事件流JOIN白名单维表，示例如下。

```
SELECT e.*, w.*
FROM event AS e
JOIN white_list FOR SYSTEM_TIME AS OF PROCTIME() AS w
ON e.id = w.id;
```

说明

- 维表支持 `INNER JOIN` 和 `LEFT JOIN`，不支持 `RIGHT JOIN` 或 `FULL JOIN`。
- 必须加上 `FOR SYSTEM_TIME AS OF PROCTIME()`，表示JOIN维表当前时刻所看到的每条数据。
- 源表后面进来的数据只会关联当时维表的最新信息，即JOIN行为只发生在处理时间（Processing Time）。如果JOIN行为发生后，维表中的数据发生了变化（新增、更新或删除），则已关联的维表数据不会被同步变化。
- ON条件中必须包含维表所有的PRIMARY KEY的等值条件（且要求与真实表定义一致）。此外，ON条件中也可以有其他等值条件。
- 如果您有一对多JOIN需求，请在维表DDL INDEX中指定关联的KEY，详情请参见INDEX语法。
- 维表和维表不能进行JOIN。
- ON条件中维表字段不能使用CAST等类型转换函数。如果您有类型转换需求，请在源表字段进行操作。

示例

- 测试数据 datahub_input1

id（bigint）	name（varchar）	age（bigint）
1	lilei	22
2	hanmeimei	20
3	libai	28

phoneNumber

name（varchar）	phoneNumber（bigint）
dufu	13900001111
baijuyi	13900002222
libai	13900003333
lilei	13900004444

- 测试语句

```
CREATE TABLE datahub_input1 (
id            BIGINT,
name        VARCHAR,
age           BIGINT
) WITH (
type='datahub'
);
create table phoneNumber(
name VARCHAR,
phoneNumber bigint,
primary key(name),
PERIOD FOR SYSTEM_TIME
)with(
type='rds'
);
CREATE table result_infor(
id bigint,
phoneNumber bigint,
name VARCHAR
)with(
type='rds'
);
INSERT INTO result_infor
SELECT
t.id,
w.phoneNumber,
t.name
FROM datahub_input1 as t
JOIN phoneNumber FOR SYSTEM_TIME AS OF PROCTIME() as w
ON t.name = w.name;
```

- 测试结果

id（bigint）	phoneNumber（bigint）	name（varchar）
1	13900004444	lilei
3	13900003333	libai

4.8.7. IntervalJoin语句

IntervalJoin语句可以让两个流进行JOIN时，左流和右流中每条记录只关联另外一条流上同一时间段内的数据，且进行完JOIN后，仍然保留输入流上的时间列，让您继续进行基于Event Time的操作。

语法格式

```
SELECT column-names
FROM table1  [AS <alias1>]
[INNER | LEFT | RIGHT |FULL ] JOIN table2
ON table1.column-name1 = table2.key-name1 AND TIMEBOUND_EXPRESSION
```

> **说明**
>
> - 支持INNER JOIN、LEFT JOIN、RIGHT JOIN和FULL JOIN，如果直接使用JOIN，默认为INNER JOIN。
> - 暂不支持SEMI JOIN和ANTI JOIN。
> - TIMEBOUND_EXPRESSION为左右两个流时间属性列上的区间条件表达式，支持以下三种条件表达式：
>   - ltime = rtime
>   - ltime >= rtime AND ltime < rtime + INTERVAL '10' MINUTE
>   - ltime BETWEEN rtime - INTERVAL '10' SECOND AND rtime + INTERVAL '5' SECOND

示例1（基于Event Time）

统计下单后4个小时内的物流信息。

- 测试数据
  - 订单表（orders）

id	productName	orderTime
1	iphone	2020-04-01 10:00:00.0
2	mac	2020-04-01 10:02:00.0
3	huawei	2020-04-01 10:03:00.0
4	pad	2020-04-01 10:05:00.0

  - 物流表（shipments）

shipId	orderId	status	shiptime
0	1	shipped	2020-04-01 11:00:00.0
1	2	delivered	2020-04-01 17:00:00.0
2	3	shipped	2020-04-01 12:00:00.0
3	4	shipped	2020-04-01 11:30:00.0

- 测试语句

```
CREATE TABLE Orders(
  id BIGINT,
  productName VARCHAR,
  orderTime TIMESTAMP,
  WATERMARK wk FOR orderTime as withOffset(orderTime, 2000)  --为rowtime定义Watermark。
) WITH (
  type='datahub',
  endpoint='<yourEndpoint>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  projectName='<yourProjectName>',
  topic='<yourTopic>',
  project='<yourProjectName>'
);
CREATE TABLE Shipments(
  shipId BIGINT,
  orderId BIGINT,
  status VARCHAR,
  shiptime TIMESTAMP,
  WATERMARK wk FOR shiptime as withOffset(shiptime, 2000)  --为rowtime定义Watermark。
) WITH (
  type='datahub',
  endpoint='<yourEndpoint>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  projectName='<yourProjectName>',
  topic='<yourTopic>',
  project='<yourProjectName>'
);
--使用RDS作为结果表
CREATE TABLE rds_output(
  id BIGINT,
  productName VARCHAR,
  status VARCHAR
) WITH (
  type='rds',
  url='<yourDatabaseURL>',
  tableName='<yourDatabaseTablename>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO rds_output
SELECT id, productName, status
FROM Orders AS o
JOIN Shipments AS s on o.id = s.orderId AND
     o.ordertime BETWEEN s.shiptime - INTERVAL '4' HOUR AND s.shiptime;
```

- 测试结果

id(bigint)	productName(varchar)	status(varchar)
1	iphone	shipped

id(bigint)	productName(varchar)	status(varchar)
3	huawei	shipped
4	pad	shipped

示例2（基于Processing Time）

- 测试数据
  - datahub_stream1

k1	v1
1	val1
2	val2
3	val3

  - datahub_stream2

k1	v1
1	val1
2	val2
3	val3

- 测试语句

```
CREATE TABLE datahub_stream1 (
  k1 BIGINT,
  v1 VARCHAR,
  d AS PROCTIME()
) WITH (
  type='datahub',
  endpoint='<yourEndpoint>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  projectName='<yourProjectName>',
  topic='<yourTopic>',
  project='<yourProjectName>'
);
CREATE TABLE datahub_stream2 (
  k2 BIGINT,
  v2 VARCHAR,
  e AS PROCTIME()
) WITH (
  type='datahub',
  endpoint='<yourEndpoint>',
  accessId='<yourAccessID>',
  accessKey='<yourAccessSecret>',
  projectName='<yourProjectName>',
  topic='<yourTopic>',
  project='<yourProjectName>'
);
--使用RDS作为结果表
CREATE TABLE rds_output(
  k1 BIGINT,
  v1 VARCHAR,
  v2 VARCHAR
) WITH (
  type='rds',
  url='<yourDatabaseURL>',
  tableName='<yourDatabaseTablename>',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO rds_output
SELECT k1, v1, v2
FROM datahub_stream1 AS o
JOIN datahub_stream2 AS s on o.k1 = s.k2 AND
     o.d BETWEEN s.e - INTERVAL '4' MINUTE AND s.e;
```

说明 由于结果取决于两个流里每条数据进入系统的时间，具有不确定性，因此该示例暂不提供预期结果。

4.8.8. UNION ALL语句

UNION ALL语句将两个流式数据合并。两个流式数据的字段必须完全一致，包括字段类型和字段顺序。

语法

```
select_statement
UNION ALL
select_statement;
```

说明 实时计算Flink版同样支持 `UNION` 函数。 `UNION ALL` 允许重复值， `UNION` 不允许重复值。在实时计算Flink版系统中， `UNION` 相当于 `UNION ALL+Distinct` ，运行效率低，通常不推荐使用 `UNION` 。

示例

- 测试数据 test_source_union1

a（varchar）	b（bigint）	c（bigint）
test1	1	10

test_source_union2

a（varchar）	b（bigint）	c（bigint）
test1	1	10
test2	2	20

test_source_union3

a（varchar）	b（bigint）	c（bigint）
test1	1	10
test2	2	20
test1	1	10

- 测试语句

```
SELECT
    a,
    sum(b) as d,
    sum(c) as e
FROM
    (SELECT * from test_source_union1
    UNION ALL
    SELECT * from test_source_union2
    UNION ALL
    SELECT * from test_source_union3
    )t
 GROUP BY a;
```

- 测试结果

a（varchar）	d（bigint）	e（bigint）
test1	1	10
test2	2	20
test1	2	20
test1	3	30
test2	4	40
test1	4	40

说明 此结果为调试结果，会显示出计算过程。如果您的结果表是DataHub、消息队列Kafka或消息队列MQ等，正式上线也会显示过程数据。但如果您的结果表是云数据RDS等关系型数据库，正式上线，主键相同的记录显示为一条数据。

4.8.9. TopN语句

TopN语句用于对实时数据中某个指标的前N个最值的筛选。Flink SQL可以基于OVER窗口操作灵活地完成TopN的功能。

语法

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
    ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
  FROM table_name)
WHERE rownum <= N [AND conditions]
```

说明

- `ROW_NUMBER()`：行号计算函数`OVER`的窗口，行号计算从1开始。
- `PARTITION BY col1[, col2..]`：指定分区的列，可以不指定。
- `ORDER BY col1 [asc|desc][, col2 [asc|desc]...]`：指定排序的列和每列的排序方向。

如上语法所示，TopN需要两层Query：

- 查询中使用`ROW_NUMBER()`窗口函数来对数据根据排序列进行排序并标上排名。
- 外层查询中对排名进行过滤，只取前N条。例如N=10即为取前10条的数据。

在执行过程中，Flink SQL会对输入的数据流根据排序键进行排序。如果某个分区的前N条记录发生了改变，则会将改变的那几条数据以更新流的形式发给下游。

说明 如果需要将TopN的数据输出到外部存储，后接的结果表必须是一个带主键的表。

WHERE条件的限制

为了使Flink SQL能识别出这是一个TopN的query，外层循环中必须要指定 `rownum <= N` 的格式来指定前N条记录。此时，您不可以将 `rownum` 置于某个表达式中，例如 `rownum - 5 <= N` 。WHERE条件中，可以额外带上其他条件，但是必须使用 `AND` 连接条件。

示例1

本例中，先统计查询流中每小时、每个城市和关键字被查询的次数，然后输出每小时、每个城市被查询最多的前100个关键字。在输出表中，小时、城市、排名三者可以唯一确定一条记录，所以需要将这三列声明成联合主键（在外部存储中也需要有同样的主键设置）。

```
CREATE TABLE rds_output (
  rownum BIGINT,
  start_time BIGINT,
  city VARCHAR,
  keyword VARCHAR,
  pv BIGINT,
  PRIMARY KEY (rownum, start_time, city)
) WITH (
  type = 'rds',
  ...
)
INSERT INTO rds_output
SELECT rownum, start_time, city, keyword, pv
FROM (
  SELECT *,
     ROW_NUMBER() OVER (PARTITION BY start_time, city ORDER BY pv desc) AS rownum
  FROM (
        SELECT SUBSTRING(time_str,1,12) AS start_time,
            keyword,
            count(1) AS pv,
            city
        FROM tmp_search
        GROUP BY SUBSTRING(time_str,1,12), keyword, city
    ) a
) t
WHERE rownum <= 100
```

示例2

- 测试数据

ip（VARCHAR）	time（VARCHAR）
`192.168.1.1`	100000000
`192.168.1.2`	100000000
`192.168.1.2`	100000000
`192.168.1.3`	100030000

ip（VARCHAR）	time（VARCHAR）
192.168.1.3	100000000
192.168.1.3	100000000

- 测试语句

```
CREATE TABLE source_table (
  IP VARCHAR,
  `TIME` VARCHAR
)WITH(
  type='datahub',
  endPoint='<yourEndpoint>',
  project='<yourProjectName>',
  topic='<yourTopicName>',
  accessId='<yourAccessId>',
  accessKey='<yourAccessSecret>'
);
CREATE TABLE result_table (
  rownum BIGINT,
  start_time VARCHAR,
  IP VARCHAR,
  cc BIGINT,
  PRIMARY KEY (start_time, IP)
) WITH (
  type = 'rds',
  url='<yourDatabaseAddress>',
  tableName='blink_rds_test',
  userName='<yourDatabaseUserName>',
  password='<yourDatabasePassword>'
);
INSERT INTO result_table
SELECT rownum,start_time,IP,cc
FROM (
  SELECT *,
     ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY cc DESC) AS rownum
  FROM (
        SELECT SUBSTRING(`TIME`,1,2) AS start_time,--可以根据真实时间取相应的数值，这里取得是测试数据。
        COUNT(IP) AS cc,
        IP
        FROM  source_table
        GROUP BY SUBSTRING(`TIME`,1,2), IP
    )a
) t
WHERE rownum <= 3 --可以根据真实top值取相应的数值，这里取得是测试数据。
```

- 测试结果

rownum（BIGINT）	start_time（VARCHAR）	ip（VARCHAR）	cc（BIGINT）
1	10	192.168.1.3	3

rownum（BIGINT）	start_time（VARCHAR）	ip（VARCHAR）	cc（BIGINT）
2	10	192.168.1.2	2
3	10	192.168.1.1	1

无排名优化

- 使用无排名优化，您可以解决数据膨胀问题。
  - 数据膨胀问题

    根据TopN的语法，`rownum`字段会作为结果表的主键字段之一写入结果表。但是这可能导致数据膨胀的问题。例如，收到一条原排名9的更新数据，更新后排名上升到1，则从1到9的数据排名都发生变化了，需要将这些数据作为更新都写入结果表。这样就产生了数据膨胀，导致结果表因为收到了太多的数据而降低更新速度。
  - 无排名优化方法

    结果表中不保存`rownum`，最终的`rownum`由前端计算。因为TopN的数据量通常不会很大，前端排序100个数据很快。当收到一条原排名9、更新后排名上升到1的数据，也只需要发送这一条数据，而不用把排名1到9的数据全发送下去。这种优化能够提升结果表的更新速度。
- 无排名优化语法

```
SELECT col1, col2, col3
FROM (
 SELECT col1, col2, col3
   ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]]
   ORDER BY col1 [asc|desc][, col2 [asc|desc]...]) AS rownum
 FROM table_name)
WHERE rownum <= N [AND conditions]
```

语法与上文类似，只是在外层查询中将rownum字段裁剪掉即可。

> **说明** 在无rownum的场景中，对于结果表主键的定义需要特别小心。如果定义有误，会直接导致TopN结果的不正确。 无rownum场景中，主键应为TopN上游GROUP BY节点的KEY列表。

- 无排名优化示例

  本示例来源于自视频行业的案例。用户每个视频在分发时会产生大量流量，依据视频产生的流量可以分析出最热门的视频。本例用于统计出每分钟流量最大的Top5的视频。
  - 测试语句

```
--从SLS读取数据原始存储表。
CREATE TABLE sls_cdnlog_stream (
vid VARCHAR, -- video id
rowtime TIMESTAMP, -- 观看视频的时间。
response_size BIGINT, -- 观看产生的流量。
WATERMARK FOR rowtime as withOffset(rowtime, 0)
) WITH (
type='sls',
...
);
--1分钟窗口统计vid带宽数。
CREATE VIEW cdnvid_group_view AS
SELECT vid,
TUMBLE_START(rowtime, INTERVAL '1' MINUTE) AS start_time,
SUM(response_size) AS rss
FROM sls_cdnlog_stream
GROUP BY vid, TUMBLE(rowtime, INTERVAL '1' MINUTE);
--存储表。
CREATE TABLE hbase_out_cdnvidtoplog (
vid VARCHAR,
rss BIGINT,
start_time VARCHAR,
   -- 注意结果表中不存储rownum字段。
   -- 特别注意该主键的定义，为TopN上游GROUP BY的KEY。
PRIMARY KEY(start_time, vid)
) WITH (
type='RDS',
...
);
-- 统计每分钟Top5消耗流量的vid，并输出。
INSERT INTO hbase_out_cdnvidtoplog
-- 注意次外层查询，不选出rownum字段。
SELECT vid, rss, start_time FROM
(
SELECT
vid, start_time, rss,
ROW_NUMBER() OVER (PARTITION BY start_time ORDER BY rss DESC) as rownum
FROM
cdnvid_group_view
)
WHERE rownum <= 5;
```

- 测试数据

vid（VARCHAR）	rowtime（Timestamp）	response_size（BIGINT）
10000	2017-12-18 15:00:10	2000
10000	2017-12-18 15:00:15	4000
10000	2017-12-18 15:00:20	3000
10001	2017-12-18 15:00:20	3000
10002	2017-12-18 15:00:20	4000
10003	2017-12-18 15:00:20	1000
10004	2017-12-18 15:00:30	1000
10005	2017-12-18 15:00:30	5000
10006	2017-12-18 15:00:40	6000
10007	2017-12-18 15:00:50	8000

- 测试结果

start_time（VARCHAR）	vid（VARCHAR）	rss（BIGINT）
2017-12-18 15:00:00	10000	9000
2017-12-18 15:00:00	10007	8000
2017-12-18 15:00:00	10006	6000
2017-12-18 15:00:00	10005	5000
2017-12-18 15:00:00	10002	4000

4.8.10. GROUPING SETS语句

如果您经常需要对数据进行多维度聚合分析（例如既需要按照a列聚合，也要按照b列聚合，同时要按照a和b两列聚合），您可以使用GROUPING SETS语句进行多维度聚合分析，避免多次使用UNION ALL影响性能。

语法格式

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
GROUP BY
[GROUPING SETS { groupItem [, groupItem ]* } ];
```

示例

- 测试数据

username	month	day
Lily	10	1
Lucy	11	21
Lily	11	21

- 测试案例

```
SELECT
    `month`,
    `day`,
    count(distinct `username`) as uv
FROM tmall_item
group by
grouping sets((`month`),(`month`,`day`));
```

- 测试结果

month	day	uv
10	1	1
10	null	1
11	21	1
11	null	1
11	21	2
11	null	2

> **说明** 此结果为调试结果，会显示出计算过程。如果您的结果表是DataHub、消息队列Kafka或消息队列MQ等，正式上线也会显示过程数据。但如果您的结果表是云数据RDS等关系型数据库，正式上线，主键相同的记录显示为一条数据。

4.8.11. 复杂事件处理（CEP）语句

复杂事件处理（CEP）语句MATCH_RECOGNIZE，用于识别输入流中符合指定规则的事件，并按照指定方式输出。

语法

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[MATCH_RECOGNIZE (
[PARTITION BY {partitionItem [, partitionItem]*}]
[ORDER BY {orderItem [, orderItem]*}]
[MEASURES {measureItem AS col [, measureItem AS col]*}]
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MATCH WITH TIMEOUT ROWS]
[AFTER MATCH SKIP]
PATTERN (patternVariable[quantifier] [ patternVariable[quantifier]]*) WITHIN intervalExpression
DEFINE {patternVariable AS patternDefinationExpression [, patternVariable AS patternDefinationExpression]*}
)];
```

参数	说明
PARTITION BY	分区的列，可选项。
ORDER BY	可以指定多列，但是必须以 `EVENT TIME` 或 `PROCESS TIME` 列作为排序的首列，可选项。
MEASURES	定义如何根据匹配成功的输入事件构造输出事件。
ONE ROW PER MATCH	对于每一次成功的匹配，只产生一个输出事件。
ONE ROW PER MATCH WITH TIMEOUT ROWS	除了匹配成功时产生的输出外，超时也会产生输出。超时时间由PATTERN语句中的WITHIN语句定义。 **说明** Blink 3.6.0及以后版本，由于Calcite升级，不再支持ONE ROW PER MATCH WITH TIMEOUT ROWS参数。
ALL ROWS PER MATCH	对于每一次成功的匹配，对应于每一个输入事件，都会产生一个输出事件。
ALL ROWS PER MATCH WITH TIMEOUT ROWS	除了匹配成功时产生输出外，超时也会产生输出。超时时间由PATTERN语句中的WITHIN语句定义。 **说明** Blink 3.6.0及以后版本，由于Calcite升级，不再支持ALL ROWS PER MATCH WITH TIMEOUT ROWS参数。
[ONE ROW PER MATCH|ALL ROWS PER MATCH|ONE ROW PER MATCH WITH TIMEOUT ROWS|ALL ROWS PER MATCH WITH TIMEOUT ROWS]	可选项，默认为 `ONE ROW PER MATCH` 。
AFTER MATCH SKIP TO NEXT ROW	匹配成功后，从匹配成功的事件序列中的第一个事件的下一个事件开始进行下一次匹配。

参数	说明
AFTER MATCH SKIP PAST LAST ROW	匹配成功后，从匹配成功的事件序列中的最后一个事件的下一个事件开始进行下一次匹配。
AFTER MATCH SKIP TO FIRST patternItem	匹配成功后，从匹配成功的事件序列中第一个对应于patternItem的事件开始进行下一次匹配。
AFTER MATCH SKIP TO LAST patternItem	匹配成功后，从匹配成功的事件序列中最后一个对应于patternItem的事件开始进行下一次匹配。
PATTERN	定义待识别的事件序列需要满足的规则，需要定义在 `()` 中，由一系列自定义的patternVariable构成。 **说明** • patternVariable之间如果以空格间隔，表示符合这两种patternVariable的事件中间不存在其他事件。 • patternVariable之间如果以->间隔，表示符合这两种patternVariable的事件之间可以存在其它事件。 • Blink 3.6.0及以后版本，由于Calcite升级，不再支持PATTERN参数。

参数解释

- quantifier

  quantifier用于指定符合patternVariable定义事件的出现次数。

参数	说明
*	0次或多次
+	1次或多次
?	0次或1次
{n}	n次
{n,}	大于等于n次
{n, m}	大于等于n次，小于等于m次
{,m}	小于等于m次

  默认为贪婪匹配。例如，对于 `pattern: A -> B+ -> C` ，输入为 `a bc1 bc2 c` （其中bc1和bc2表示既匹配B也匹配C），则输出为 `a bc1 bc2 c` 。可以在quantifier符号后面加？来表示非贪婪匹配。例如：

  - `*?`

- `+?`
- `{n}?`
- `{n,}?`
- `{n, m}?`
- `{,m}?`

> **说明** Blink 3.x以上版本不支持(e1 e2+)贪婪匹配，您可以使用e1 e2+ e3 e3 as not e2绕行方案，但请务必要使用一个e3，才能有数据输出。

此时，对于上面例子中的PATTERN及输入，产生的输出为a bc1 bc2,a bc1 bc2 c。

- WITHIN定义符合规则的事件序列的最大时间跨度。
- 静态窗口格式：`INTERVAL 'string' timeUnit [ TO timeUnit ]` 。例如：`INTERVAL '10' SECOND, INTERVAL '45' DAY, INTERVAL '10:20' MINUTE TO SECOND, INTERVAL '10:20.10' MINUTE TO SECOND, INTERVAL '10:20' HOUR TO MINUTE, INTERVAL '1-5' YEAR TO MONTH` 。
- 动态窗口格式：`INTERVAL intervalExpression` 。例如：`INTERVAL A.windowTime + 10` ，其中A为PATTERN定义中第一个patternVariable。在intervalExpression的定义中，可以使用PATTERN定义中出现过的patternVariable。当前只能使用第一个patternVariable。intervalExpression中可以使用UDF，intervalExpression的结果必须为LONG，单位为millisecond，表示窗口的大小。
- DEFINE定义在PATTERN中出现的patternVariable的具体含义，如果某个patternVariable在DEFINE中没有定义，则认为对于每一个事件，该patternVariable都成立。

- MEASURES和DEFINE语句函数

函数	说明
Row Pattern Column References	形式为 `patternVariable.col` 。表示访问patternVariable所对应的事件的指定的列。
PREV	只能用在DEFINE语句中，通常与 `Row Pattern Column References` 合用。用于访问指定的PATTERN所对应的事件之前，偏移指定的offset所对应的事件的指定的列。 例如，`DOWN AS DOWN.price < PREV(DOWN.price)` ，`PREV(A.price)` 表示当前事件的前一个事件的 `price` 列的值。 **说明** • `DOWN.price` 等价于 `PREV(DOWN.price, 0)` 。 • `PREV(DOWN.price)` 等价于 `PREV(DOWN.price, 1)` 。
FIRST、LAST	一般与 `Row Pattern Column References` 合用，用于访问指定的PATTERN所对应的事件序列中的指定偏移位置的事件。例如： • `FIRST(A.price, 3)` 表示 `PATTERN A` 所对应的事件序列中的第4个事件。 • `LAST(A.price, 3)` 表示 `PATTERN A` 所对应的事件序列中的倒数第4个事件。

函数	说明

- 输出列

函数	输出列
ONE ROW PER MATCH	包括 `PARTITION BY` 中指定的列及 `MEASURES` 中定义的列。对于 `PARTITION BY` 中已经指定的列，在 `MEASURES` 中无需重复定义。
ONE ROW PER MATCH WITH TIMEOUT ROWS	除匹配成功的时候产生输出外，超时的时候也会产生输出，超时时间由PATTERN语句中的WITHIN语句定义。

> **说明**
> - 当定义PATTERN时，最好也定义WITHIN，否则可能会造成STATE越来越大。
> - ORDER BY中定义的首列必须为EVENT TIME列或者PROCESS TIME列。

示例

- 示例语法

```
SELECT *
FROM Ticker MATCH_RECOGNIZE (
PARTITION BY symbol
ORDER BY tstamp
MEASURES STRT.tstamp AS start_tstamp,
LAST(DOWN.tstamp) AS bottom_tstamp,
LAST(UP.tstamp) AS end_tstamp
ONE ROW PER MATCH
AFTER MATCH SKIP TO NEXT ROW
PATTERN (STRT DOWN+ UP+) WITHIN INTERVAL '10' SECOND
DEFINE
DOWN AS DOWN.price < PREV(DOWN.price),
UP AS UP.price > PREV(UP.price)
);
```

- 测试数据

timestamp（TIMESTAMP）	card_id(VARCHAR)	location（VARCHAR）	action（VARCHAR）
`2018-04-13 12:00:00`	1	Beijing	Consumption

timestamp（TIMESTAMP）	card_id(VARCHAR)	location（VARCHAR）	action（VARCHAR）
2018-04-13 12:05:00	1	Shanghai	Consumption
2018-04-13 12:10:00	1	Shenzhen	Consumption
2018-04-13 12:20:00	1	Beijing	Consumption

- 测试语法

  当相同的card_id在十分钟内，在两个不同的location发生刷卡现象，就会触发报警机制，以便监测信用卡盗刷等现象。

```
CREATE TABLE datahub_stream (
    `timestamp`               TIMESTAMP,
    card_id                   VARCHAR,
    location                  VARCHAR,
    `action`                  VARCHAR,
    WATERMARK wf FOR `timestamp` AS withOffset(`timestamp`, 1000)
) WITH (
    type = 'datahub'
    ...
);
CREATE TABLE rds_out (
    start_timestamp               TIMESTAMP,
    end_timestamp                 TIMESTAMP,
    card_id                       VARCHAR,
    event                         VARCHAR
) WITH (
    type= 'rds'
    ...
);
--定义计算逻辑。
insert into rds_out
select
`start_timestamp`,
`end_timestamp`,
card_id, `event`
from datahub_stream
MATCH_RECOGNIZE (
    PARTITION BY card_id   --按card_id分区，将相同卡号的数据分发到同一个计算节点。
    ORDER BY `timestamp`   --在窗口内，对事件时间进行排序。
    MEASURES               --定义如何根据匹配成功的输入事件构造输出事件。
        e2.`action` as `event`,
        e1.`timestamp` as `start_timestamp`,   --第一次的事件时间为start_timestamp。
        LAST(e2.`timestamp`) as `end_timestamp` --最新的事件时间为end_timestamp。
    ONE ROW PER MATCH           --匹配成功输出一条。
    AFTER MATCH SKIP TO NEXT ROW --匹配后跳转到下一行。
    PATTERN (e1 e2+) WITHIN INTERVAL '10' MINUTE  --定义两个事件，e1和e2。
    DEFINE                      --定义在PATTERN中出现的patternVariable的具体含义。
        e1 as e1.action = 'Consumption',    --事件一的action标记为Consumption。
        e2 as e2.action = 'Consumption' and e2.location <> e1.location --事件二的action标记为Consumption，且事件一和事件二的location不一致。
);
```

> **说明** 如果存在满足CEP条件的数据，但没有结果输出是因为只有Watermark > e2.ts的数据才会被处理，但由于e2后面已经没有数据了，导致Watermark一直是e2.ts -1000，所以e2的数据一直没有被处理，从而导致没有结果输出。

- 测试结果

start_timestamp（TIMESTAMP）	end_timestamp（TIMESTAMP）	card_id（VARCHAR）	event（VARCHAR）
2018-04-13 12:00:00.0	2018-04-13 12:05:00.0	1	Consumption
2018-04-13 12:05:00.0	2018-04-13 12:10:00.0	1	Consumption

4.8.12. 去重语句

您可以通过多种方式实现去重需求，例如FIRST_VALUE、LAST_VALUE和DISTINCT等。本文为您介绍如何使用TopN方法实现去重，以及使用过程中的注意事项。

去重的方案通常有两种：

- 保留第一条。
- 保留最后一条。

> **说明** ORDER BY后的时间属性字段必须在源表中定义。

语法

由于SQL没有直接去重的语法，因此我们使用SQL的 `ROW_NUMBER OVER WINDOW` 功能实现去重。 `ROW_NUMBER OVER WINDOW` 与TopN语句方法类似，可以理解为一种特殊的TopN。

```
SELECT *
FROM (
   SELECT *,
     ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]
     ORDER BY timeAttributeCol [asc|desc]) AS rownum
   FROM table_name)
WHERE rownum = 1;
```

参数说明：

- `ROW_NUMBER()` ：计算行号，行号计算从1开始。
- `PARTITION BY col1[, col2..]` ：指定分区的列，即去重的Key，也可以不指定分区的列。
- `ORDER BY timeAttributeCol [asc|desc])` ：指定排序的列，必须是时间属性字段（Processing Time或Event Time）。可以指定为顺序（Keep First Row）或倒序（Keep Last Row）。
- 外层查询 `rownum` 必须为 `= 1` 或者 `<= 1` 。条件必须是 `AND` ，且不能存在Undeterministic的UDF的条件。

如上语法所示，去重需要两层Query：

- 子查询中：使用 `ROW_NUMBER()` ，按照时间属性列对数据进行排序编号。
- 外层查询中：对排名进行过滤，只取第一条，达到去重的目的。时间列排序方向可以为：
  - 顺序： `deduplicate keep first row` 。
  - 倒序： `deduplicate keep last row` 。

当排序字段是Processing Time列时，Flink会按系统时间去重，其每次运行结果不确定。当排序字段是Event Time列时，Flink会按业务时间去重，其每次运行结果是确定的。

Deduplicate Keep First Row

保留首行的去重策略，即保留指定Key下第一条出现的数据，之后出现在该Key下的数据会被丢弃掉。因为其State中只存储了Key数据，因此性能较优。示例如下。

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b ORDER BY proctime) as rowNum
  FROM T
)
WHERE rowNum = 1;
```

本例中，将T表按照b字段进行去重，并按照系统时间保留第一条数据。proctime在以上示例中是源表T中的一个具有Processing Time属性的字段。如果您按照系统时间去重，也可以将proctime字段简化成 `PROCTIME()` 函数进行调用，可以省略proctime字段的声明。

> **说明** Blink-3.3.1版本后，FirstRow支持使用Event Time进行开窗，并且不会产生Retraction。

Deduplicate Keep Last Row

> **注意** LastRow不支持使用Event Time进行开窗。

LastRow的作用也是去重，且只保留该主键下最后一条出现的数据。其性能略胜于LAST_VALUE函数，示例如下。

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY proctime DESC) as rowNum
  FROM T
)
WHERE rowNum = 1;
```

FAQ

Q：执行 `ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY now() as time DESC)` 语句时，产生如下报错，应该如何处理？

```
java.lang.RuntimeException: Can not retract a non-existent record:
    38c30001,1b800000008,1c000000013,85000035343a3731,5d304013.
    This should never happen.
```

A：通常有以下两种原因，您可以按照以下方式进行处理：

- 问题原因：由代码中 `now()` 导致。因为TopN不支持非确定性的排序字段， `now()` 每次输出的值不同，所以导致Retraction会找不到之前的值。

  解决方法：Event Time或源表中一个具有Processing Time属性的字段。

- 问题原因：`blink.state.ttl.ms` 或 `state.backend.niagara.ttl.ms` 参数值设置过小。

  解决方法：设置过小的TTL参数使用默认配置，或调大参数值。

4.9. 窗口函数

4.9.1. 概述

本文为您介绍Flink SQL支持的窗口函数以及窗口函数支持的时间属性和窗口类型。

窗口函数

窗口函数Flink SQL支持基于无限大窗口的聚合（无需在SQL Query中，显式定义任何窗口）以及对一个特定的窗口的聚合。例如，需要统计在过去的1分钟内有多少用户点击了某个的网页，可以通过定义一个窗口来收集最近1分钟内的数据，并对这个窗口内的数据进行计算。

Flink SQL支持的窗口聚合主要是两种：Window聚合和Over聚合。本文档主要为您介绍Window聚合。Window聚合支持Event Time和Processing Time两种时间属性定义窗口。每种时间属性类型支持三种窗口类型：滚动窗口（TUMBLE）、滑动窗口（HOP）和会话窗口（SESSION）。

时间属性

Flink SQL支持以下两种时间属性。实时计算可以基于这两种时间属性对数据进行窗口聚合。

- Event Time：您提供的事件时间（通常是数据的最原始的创建时间），Event Time一定是您提供在Schema里的数据。
- Processing Time：对事件进行处理的本地系统时间。

> **说明** 实时计算时间属性详情，请参见时间属性。

级联窗口

Rowtime列在经过窗口操作后，其Event Time属性将丢失。您可以使用辅助函数 `TUMBLE_ROWTIME` 、 `HOP_ROWTIME` 或 `SESSION_ROWTIME` ，获取窗口中的Rowtime列的最大值 `max(rowtime)` 作为时间窗口的Rowtime，其类型是具有Rowtime属性的TIMESTAMP，取值为 `window_end - 1` 。例如 `[00:00, 00:15)` 的窗口，返回值为 `00:14:59.999` 。

示例逻辑为：基于1分钟的滚动窗口聚合结果，进行1小时的滚动窗口聚合，可以满足您的多维度开窗需求。

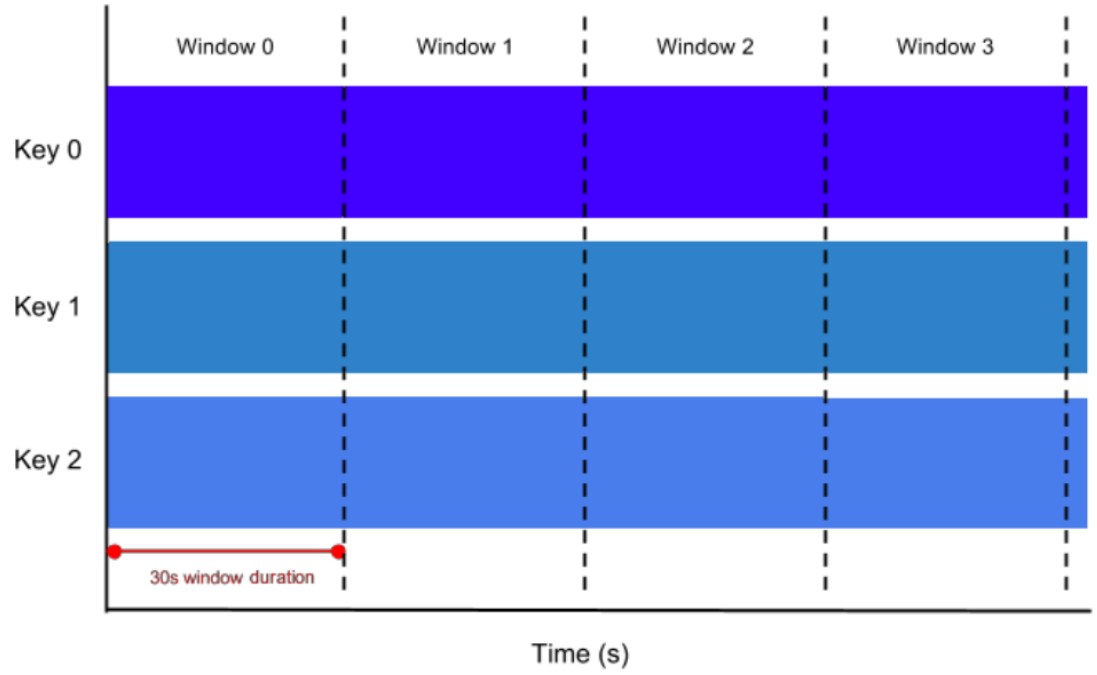
```
CREATE TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000)  --为Rowtime定义Watermark。
) with (
  type='datahub',
  ...
);
CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  type='print'
);
CREATE VIEW one_minute_window_output as
SELECT
  // 使用TUMBLE_ROWTIME作为二级Window的聚合时间。
  TUMBLE_ROWTIME(ts, INTERVAL '1' MINUTE) as rowtime,
  username,
  COUNT(click_url) as cnt
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
INSERT INTO tumble_output
SELECT
  TUMBLE_START(rowtime, INTERVAL '1' HOUR),
  TUMBLE_END(rowtime, INTERVAL '1' HOUR),
  username,
  SUM(cnt)
FROM one_minute_window_output
GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR), username;
```

4.9.2. 滚动窗口

本文为您介绍如何使用实时计算Flink版滚动窗口函数。

定义

滚动窗口（TUMBLE）将每个元素分配到一个指定大小的窗口中。通常，滚动窗口有一个固定的大小，并且不会出现重叠。例如，如果指定了一个5分钟大小的滚动窗口，无限流的数据会根据时间划分为 `[0:00, 0:05)` 、 `[0:05, 0:10)` 、 `[0:10, 0:15)` 等窗口。下图展示了一个30秒的滚动窗口。



语法

TUMBLE函数用在GROUP BY子句中，用来定义滚动窗口。

```
TUMBLE(<time-attr>, <size-interval>)
<size-interval>: INTERVAL 'string' timeUnit
```

> **说明** `<time-attr>` 参数必须是时间流中的一个合法的时间属性字段，指定为Processing Time或Event Time，请参见概述，了解如何定义时间属性和Watermark。

标识函数

使用标识函数选出窗口的起始时间或者结束时间，窗口的时间属性用于下级Window的聚合。

窗口标识函数	返回类型	描述
`TUMBLE_START(time-attr, size-interval)`	TIMESTAMP	返回窗口的起始时间（包含边界）。例如 `[00:10,00:15]` 窗口，返回 `00:10` 。
`TUMBLE_END(time-attr, size-interval)`	TIMESTAMP	返回窗口的结束时间（包含边界）。例如 `[00:00, 00:15]` 窗口，返回 `00:15` 。

窗口标识函数	返回类型	描述
`TUMBLE_ROWTIME(time-attr, size-interval)`	TIMESTAMP(rowtime-attr)	返回窗口的结束时间（不包含边界）。例如 `(00:00, 00:15)` 窗口，返回 `00:14:59.999` 。返回值是一个rowtime attribute，即可以基于该字段做时间属性的操作，例如，级联窗口只能用在基于Event Time的Window上，详情请参见级联窗口。
`TUMBLE_PROCTIME(time-attr, size-interval)`	TIMESTAMP(rowtime-attr)	返回窗口的结束时间（不包含边界）。例如 `(00:00, 00:15)` 窗口，返回 `00:14:59.999` 。返回值是一个Proctime Attribute，即可以基于该字段做时间属性的操作。例如，级联窗口只能用在基于Processing Time的Window上，详情请参见级联窗口。

使用Event Time统计每个用户每分钟在指定网站的单击数示例

- 测试数据

username（VARCHAR）	click_url（VARCHAR）	ts（TIMESTAMP）
Jark	*http://taobao.com/xxx*	`2017-10-10 10:00:00.0`
Jark	*http://taobao.com/xxx*	`2017-10-10 10:00:10.0`
Jark	*http://taobao.com/xxx*	`2017-10-10 10:00:49.0`
Jark	*http://taobao.com/xxx*	`2017-10-10 10:01:05.0`
Jark	*http://taobao.com/xxx*	`2017-10-10 10:01:58.0`
Timo	*http://taobao.com/xxx*	`2017-10-10 10:02:10.0`

- 测试语句

```
CREATE TABLE user_clicks(
  username varchar,
  click_url varchar,
  ts timeStamp,
  WATERMARK wk FOR ts as withOffset(ts, 2000) --为rowtime定义Watermark。
) with (
  type='datahub',
  ...
);
CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  type='RDS'
);
INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE) as window_start,
TUMBLE_END(ts, INTERVAL '1' MINUTE) as window_end,
username,
COUNT(click_url)
FROM user_clicks
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
```

- 测试结果

window_start（TIMESTAMP）	window_end（TIMESTAMP）	username（VARCHAR）	clicks（BIGINT）
2017-10-10 10:00:00.0	2017-10-10 10:01:00.0	Jark	3
2017-10-10 10:01:00.0	2017-10-10 10:02:00.0	Jark	2
2017-10-10 10:02:00.0	2017-10-10 10:03:00.0	Timo	1

使用Processing Time统计每个用户每分钟在指定网站的单击数示例

- 测试数据

username（VARCHAR）	click_url（VARCHAR）
Jark	*http://taobao.com/xxx*
Jark	*http://taobao.com/xxx*
Jark	*http://taobao.com/xxx*
Jark	*http://taobao.com/xxx*

username （VARCHAR）	click_url（VARCHAR）
Jark	*http://taobao.com/xxx*
Timo	*http://taobao.com/xxx*

- 测试语句

```
CREATE TABLE window_test (
  username   VARCHAR,
  click_url  VARCHAR,
  ts as PROCTIME()
) WITH (
  type='datahub',
  ...
);
CREATE TABLE tumble_output(
  window_start TIMESTAMP,
  window_end TIMESTAMP,
  username VARCHAR,
  clicks BIGINT
) with (
  type='print'
);
INSERT INTO tumble_output
SELECT
TUMBLE_START(ts, INTERVAL '1' MINUTE),
TUMBLE_END(ts, INTERVAL '1' MINUTE),
username,
COUNT(click_url)
FROM window_test
GROUP BY TUMBLE(ts, INTERVAL '1' MINUTE), username;
```

- 测试结果

window_start （TIMESTAMP）	window_end （TIMESTAMP）	username （VARCHAR）	clicks（BIGINT）
2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Jark	5
2019-04-11 14:43:00.000	2019-04-11 14:44:00.000	Timo	1

> **说明** 因为本地调试是瞬时的，处理时间可能小于1秒，所以使用Processing Time时间属性对数据进行窗口聚合，可能会出现本地调试没有结果的情况。

4.9.3. 滑动窗口

本文为您介绍如何使用实时计算滑动窗口函数。

> **说明** 实时计算滑动窗口（HOP）暂不支持与LAST_VALUE、FIRST_VALUE或TopN函数共同使用。

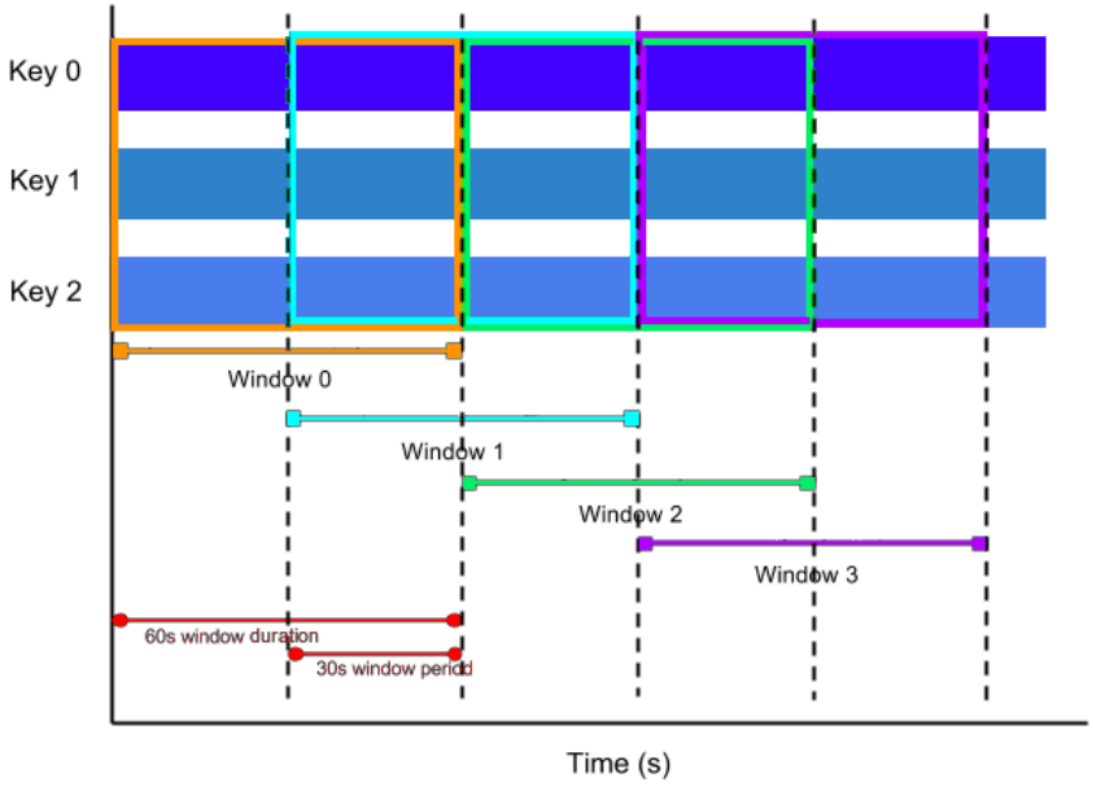
什么是滑动窗口

滑动窗口（HOP），也被称作Sliding Window。不同于滚动窗口，滑动窗口的窗口可以重叠。

滑动窗口有两个参数：slide和size。slide为每次滑动的步长，size为窗口的大小。

- slide < size，则窗口会重叠，每个元素会被分配到多个窗口。
- slide = size，则等同于滚动窗口（TUMBLE）。
- slide > size，则为跳跃窗口，窗口之间不重叠且有间隙。

通常，大部分元素符合多个窗口情景，窗口是重叠的。因此，滑动窗口在计算移动平均数（moving averages）时很实用。例如，计算过去5分钟数据的平均值，每10秒钟更新一次，可以设置slide为*10*秒，size为*5*分钟。下图为您展示间隔为30秒，窗口大小为1分钟的滑动窗口。



滑动窗口函数语法

HOP函数用在group by子句中，用来定义滑动窗口。

```
HOP(<time-attr>, <slide-interval>,<size-interval>)
<slide-interval>: INTERVAL 'string' timeUnit
<size-interval>: INTERVAL 'string' timeUnit
```

> 说明
>
> `<time-attr>` 参数必须是流中的一个合法的时间属性字段，指定为Processing Time或Event Time。请参见概述，了解如何定义时间属性和Watermark。

滑动窗口标识函数

使用滑动窗口标识函数选出窗口的起始时间或者结束时间，窗口的时间属性用于下级Window的聚合。

窗口标识函数	返回类型	描述
`HOP_START (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP	返回窗口的起始时间（包含边界）。例如 `[00:10, 00:15)` 窗口，返回 `00:10` 。
`HOP_END (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP	返回窗口的结束时间（包含边界）。例如 `[00:00, 00:15)` 窗口，返回 `00:15` 。
`HOP_ROWTIME (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP（rowtime-attr）	返回窗口的结束时间（不包含边界）。例如 `(00:00, 00:15)` 窗口，返回 `00:14:59.999` 。返回值是一个rowtime attribute，即可以基于该字段做时间类型的操作，例如级联窗口，只能用在基于event time的window上。
`HOP_PROCTIME (<time-attr>, <slide-interval>, <size-interval>)`	TIMESTAMP（rowtime-attr）	返回窗口的结束时间（不包含边界）。例如 `(00:00, 00:15)` 窗口，返回 `00:14:59.999` 。返回值是一个proctime attribute，即可以基于该字段做时间类型的操作，例如级联窗口，只能用在基于processing time的window上。

示例

统计每个用户过去1分钟的单击次数，每30秒更新1次，即1分钟的窗口，30秒滑动1次。

- 测试数据

username（VARCHAR）	click_url（VARCHAR）	ts（TIMESTAMP）
Jark	`http://taobao.com/xxx`	`2017-10-10 10:00:00.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:00:10.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:00:49.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:01:05.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:01:58.0`
Timo	`http://taobao.com/xxx`	`2017-10-10 10:02:10.0`

- 测试语句

```
CREATE TABLE user_clicks (
    username VARCHAR,
    click_url VARCHAR,
    ts TIMESTAMP,
    WATERMARK wk FOR ts AS WITHOFFSET (ts, 2000)--为rowtime定义Watermark。
) WITH ( TYPE = 'datahub',
        ...);
CREATE TABLE hop_output (
    window_start TIMESTAMP,
    window_end TIMESTAMP,
    username VARCHAR,
    clicks BIGINT
) WITH (TYPE = 'rds',
        ...);
INSERT INTO
    hop_output
SELECT
    HOP_START (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    HOP_END (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    username,
    COUNT (click_url)
FROM
    user_clicks
GROUP BY
    HOP (ts, INTERVAL '30' SECOND, INTERVAL '1' MINUTE),
    username
```

- 测试结果

window_start（TIMESTAMP）	window_end（TIMESTAMP）	username （VARCHAR）	clicks （BIGINT）
2017-10-10 09:59:30.0	2017-10-10 10:00:30.0	Jark	2
2017-10-10 10:00:00.0	2017-10-10 10:01:00.0	Jark	3
2017-10-10 10:00:30.0	2017-10-10 10:01:30.0	Jark	2
2017-10-10 10:01:00.0	2017-10-10 10:02:00.0	Jark	2
2017-10-10 10:01:30.0	2017-10-10 10:02:30.0	Jark	1
2017-10-10 10:02:00.0	2017-10-10 10:03:00.0	Timo	1
2017-10-10 10:02:30.0	2017-10-10 10:03:30.0	Timo	1

HOP窗口无法读取数据进入的时间，第一个窗口的开启时间会前移。前移时长=窗口时长-滑动步长，示例如下表。

窗口时长（秒）	滑动步长（秒）	Event Time	第一个窗口StartTime	第一个窗口EndTime
120	30	`2019-07-31 10:00:00.0`	`2019-07-31 09:58:30.0`	`2019-07-31 10:00:30.0`
60	10	`2019-07-31 10:00:00.0`	`2019-07-31 09:59:10.0`	`2019-07-31 10:00:10.0`

4.9.4. 会话窗口

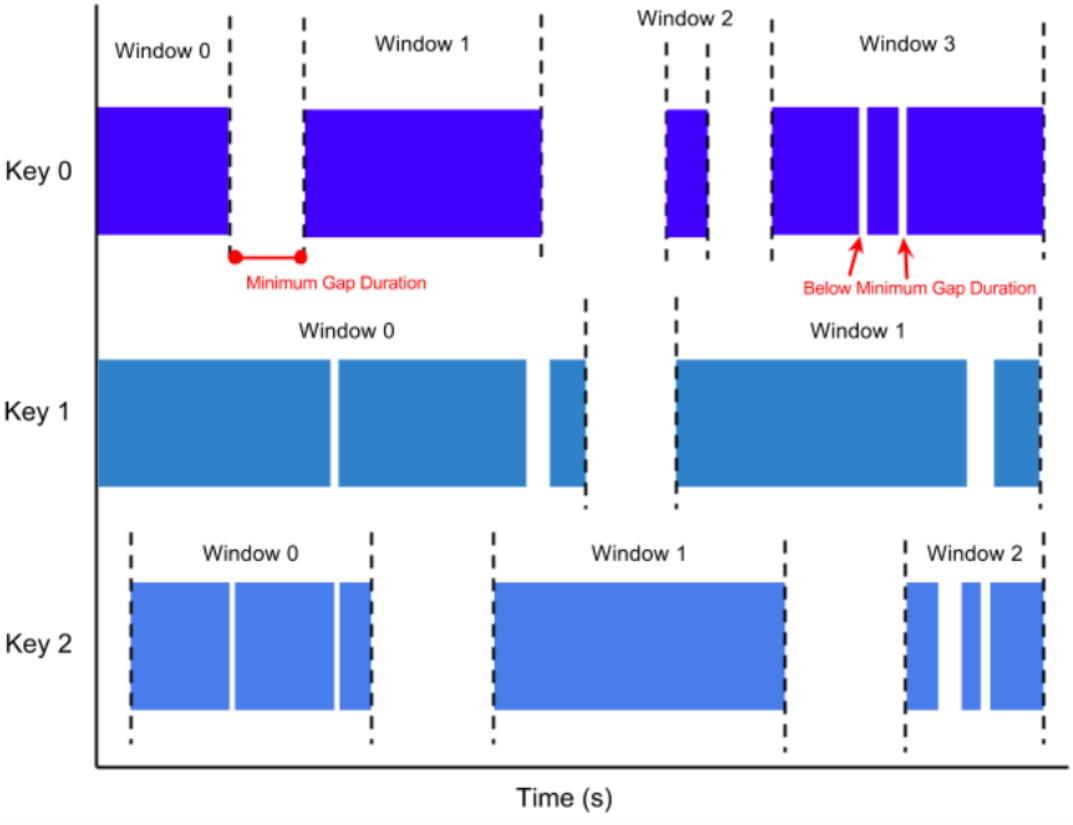
本文为您介绍如何使用实时计算Flink版会话窗口函数。

什么是会话窗口

会话窗口（SESSION）通过SESSION活动来对元素进行分组。会话窗口与滚动窗口和滑动窗口相比，没有窗口重叠，没有固定窗口大小。相反，当它在一个固定的时间周期内不再收到元素，即会话断开时，该窗口就会关闭。

会话窗口通过一个间隔时间（Gap）来配置，这个间隔定义了非活跃周期的长度。例如，一个表示鼠标单击活动的数据流可能具有长时间的空闲时间，并在两段空闲之间散布着高浓度的单击。如果数据在指定的间隔（Gap）之后到达，则会开始一个新的窗口。

会话窗口示例如下图，每个Key由于不同的数据分布，形成了不同的Window。



会话窗口函数语法

SESSION函数用于在GROUP BY子句中定义会话窗口。

```
SESSION(<time-attr>, <gap-interval>)
<gap-interval>: INTERVAL 'string' timeUnit
```

> **说明** `<time-attr>` 参数必须是数据流中的一个合法的时间属性字段，指定为Processing Time或Event Time，详情请参见概述，了解如何定义时间属性和Watermark。

会话窗口标识函数

使用标识函数选出窗口的起始时间或者结束时间，窗口的时间属性用于下级Window的聚合。

窗口标识函数	返回类型	描述
`SESSION_START(<time-attr>, <gap-interval>)`	Timestamp	返回窗口的起始时间（包含边界）。例如 `[00:10,00:15]` 的窗口，返回 `00:10` ，即为此会话窗口内第一条记录的时间。
`SESSION_END(<time-attr>, <gap-interval>)`	Timestamp	返回窗口的结束时间（包含边界）。例如 `[00:00,00:15]` 的窗口，返回 `00:15` ，即为此会话窗口内最后一条记录的时间+ `<gap-interval>` 。
`SESSION_ROWTIME(<time-attr>, <gap-interval>)`	Timestamp（rowtime-attr）	返回窗口的结束时间（不包含边界）。例如 `(00:00,00:15)` 的窗口，返回 `00:14:59.999` 。返回值是一个rowtime attribute，也就是可以基于该字段进行时间类型的操作，如级联窗口。该参数只能用于基于Event Time的Window。
`SESSION_PROCTIME(<time-attr>, <gap-interval>)`	Timestamp（rowtime-attr）	返回窗口的结束时间（不包含边界）。例如 `(00:00,00:15)` 的窗口，返回 `00:14:59.999` 。返回值是一个Proctime Attribute，也就是可以基于该字段进行时间类型的操作，如级联窗口。该参数只能用于基于Processing Time的Window。

示例

统计每个用户在每个活跃会话期间的单击次数，会话超时时长为30秒。

- 测试数据

username （VARCHAR）	click_url （VARCHAR）	ts （TIMESTAMP）
Jark	`http://taobao.com/xxx`	`2017-10-10 10:00:00.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:00:10.0`
Jark	`http://taobao.com/xxx`	`2017-10-10 10:00:49.0`

username （VARCHAR）	click_url （VARCHAR）	ts （TIMESTAMP）
Jark	http://taobao.com/xxx	2017-10-10 10:01:05.0
Jark	http://taobao.com/xxx	2017-10-10 10:01:58.0
Timo	http://taobao.com/xxx	2017-10-10 10:02:10.0

- 测试语句

```
CREATE TABLE user_clicks(
username varchar,
click_url varchar,
ts timeStamp,
WATERMARK wk FOR ts as withOffset(ts, 2000) -- 为rowtime定义watermark
) with (
type='datahub',
...
);
CREATE TABLE session_output(
window_start TIMESTAMP,
window_end TIMESTAMP,
username VARCHAR,
clicks BIGINT
) with (
type='rds',
...
);
INSERT INTO session_output
SELECT
SESSION_START(ts, INTERVAL '30' SECOND),
SESSION_END(ts, INTERVAL '30' SECOND),
username,
COUNT(click_url)
FROM user_clicks
GROUP BY SESSION(ts, INTERVAL '30' SECOND), username;
```

- 测试结果

window_start （TIMESTAMP）	window_end （TIMESTAMP）	username （VARCHAR）	clicks （BIGINT）
2017-10-10 10:00:00.0	2017-10-10 10:00:40.0	Jark	2
2017-10-10 10:00:49.0	2017-10-10 10:01:35.0	Jark	2
2017-10-10 10:01:58.0	2017-10-10 10:02:28.0	Jark	1
2017-10-10 10:02:10.0	2017-10-10 10:02:40.0	Timo	1

4.9.5. OVER窗口

OVER窗口（OVER Window）是传统数据库的标准开窗，不同于Group By Window，OVER窗口中每1个元素都对应1个窗口。OVER窗口可以按照实际元素的行或实际的元素值（时间戳值）确定窗口，因此流数据元素可能分布在多个窗口中。

在应用OVER窗口的流式数据中，每1个元素都对应1个OVER窗口。每1个元素都触发1次数据计算，每个触发计算的元素所确定的行，都是该元素所在窗口的最后1行。在实时计算的底层实现中，OVER窗口的数据进行全局统一管理（数据只存储1份），逻辑上为每1个元素维护1个OVER窗口，为每1个元素进行窗口计算，完成计算后会清除过期的数据。

语法

```
SELECT
    agg1(col1) OVER (definition1) AS colName,
    ...
    aggN(colN) OVER (definition1) AS colNameN
FROM Tab1;
```

- agg1(col1)：按照GROUP BY指定col1列对输入数据进行聚合计算。
- OVER (definition1)：OVER窗口定义。
- AS colName：别名。

> **说明**
> - agg1到aggN所对应的OVER definition1必须相同。
> - 外层SQL可以通过AS的别名查询数据。

类型

Flink SQL中对OVER窗口的定义遵循标准SQL的定义语法，传统OVER窗口没有对其进行更细粒度的窗口类型命名划分。按照计算行的定义方式，OVER Window可以分为以下两类：

- ROWS OVER Window：每1行元素都被视为新的计算行，即每1行都是一个新的窗口。
- RANGE OVER Window：具有相同时间值的所有元素行视为同一计算行，即具有相同时间值的所有行都是同一个窗口。

属性

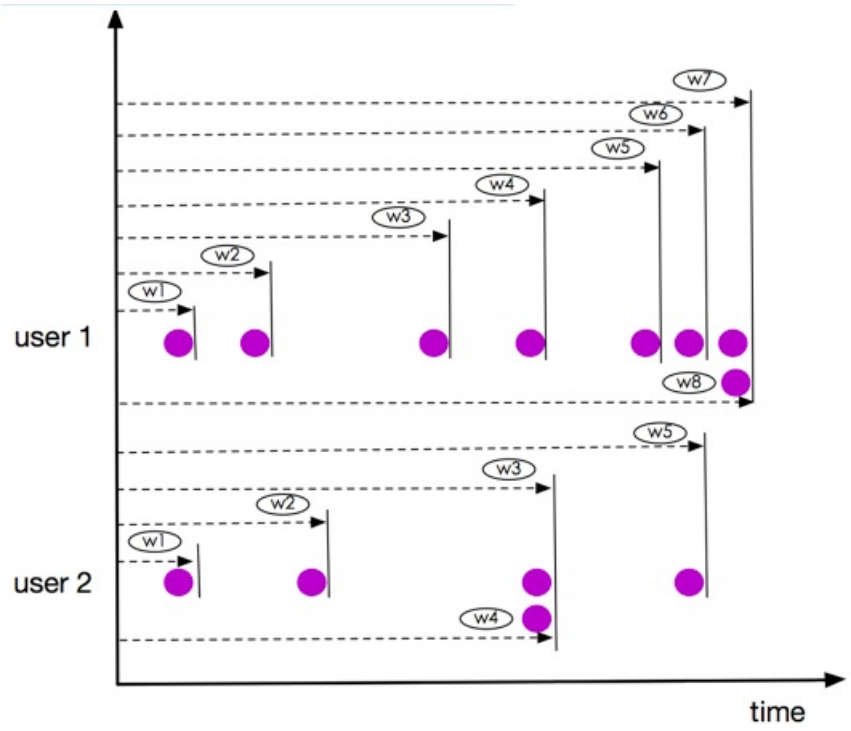
正交属性	说明	proctime	eventtime
ROWS OVER Window	按照实际元素的行确定窗口。	支持	支持
RANGE OVER Window	按照实际的元素值（时间戳值）确定窗口。	支持	支持

Rows OVER Window语义

- 窗口数据

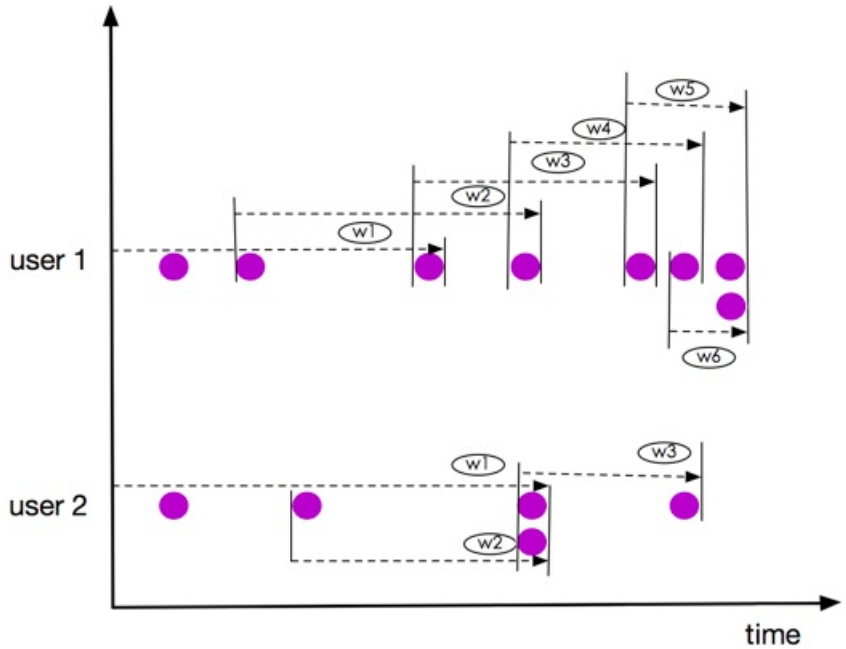
ROWS OVER Window的每个元素都确定一个窗口。ROWS OVER Window分为Unbounded（无界流）和Bounded（有界流）两种情况。

Unbounded ROWS OVER Window数据示例如下图所示。



> **说明** 虽然上图所示窗口user1的w7、w8及user2的窗口w3、w4都是同一时刻到达，但它们仍然在不同的窗口，这一点与RANGE OVER Window不同。

Bounded ROWS OVER Window数据以3个元素（往前2个元素）的窗口为例，如下图所示。



> **说明** 虽然上图所示窗口user1的w5、w6及user2的窗口w1、w2都是同一时刻到达，但它们仍然在不同的窗口，这一点与RANGE OVER Window不同。

- 窗口语法

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     ROWS
     BETWEEN (UNBOUNDED | rowCount) PRECEDING AND CURRENT ROW) AS colName, ...
FROM Tab1;
```

  - value_expression：分区值表达式。
  - timeCol：元素排序的时间字段。
  - rowCount：定义根据当前行开始向前追溯几行元素。

- 案例

  以Bounded ROWS OVER Window场景为例。假设，一张商品上架表，包含有商品ID、商品类型、商品上架时间、商品价格数据。要求输出在当前商品上架之前同类的3个商品中的最高价格。

  - 测试数据

商品ID	商品类型	上架时间	销售价格
ITEM001	Electronic	2017-11-11 10:01:00	20
ITEM002	Electronic	2017-11-11 10:02:00	50
ITEM003	Electronic	2017-11-11 10:03:00	30
ITEM004	Electronic	2017-11-11 10:03:00	60
ITEM005	Electronic	2017-11-11 10:05:00	40
ITEM006	Electronic	2017-11-11 10:06:00	20
ITEM007	Electronic	2017-11-11 10:07:00	70
ITEM008	Clothes	2017-11-11 10:08:00	20

- 测试代码

```
CREATE TABLE tmall_item(
   itemID VARCHAR,
   itemType VARCHAR,
   onSellTime TIMESTAMP,
   price DOUBLE,
   WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)
)
WITH (
  type = 'sls',
   ...
);
SELECT
    itemID,
    itemType,
    onSellTime,
    price,
    MAX(price) OVER (
        PARTITION BY itemType
        ORDER BY onSellTime
        ROWS BETWEEN 2 preceding AND CURRENT ROW) AS maxPrice
  FROM tmall_item;
```

- 测试结果

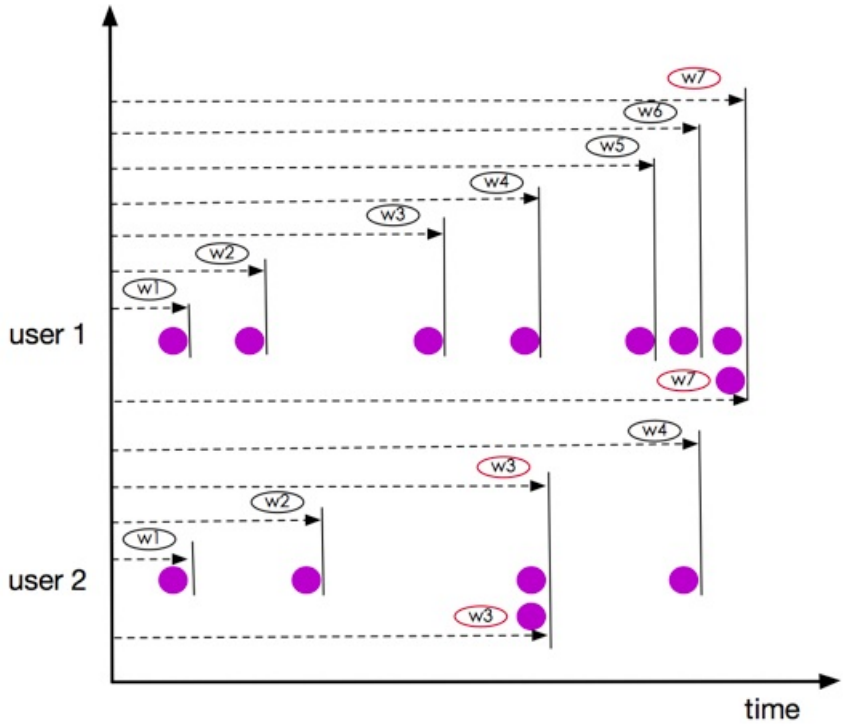
itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	20	60
ITEM007	Electronic	2017-11-11 10:07:00	70	70
ITEM008	Clothes	2017-11-11 10:08:00	20	20

RANGE OVER Window语义

- 窗口数据

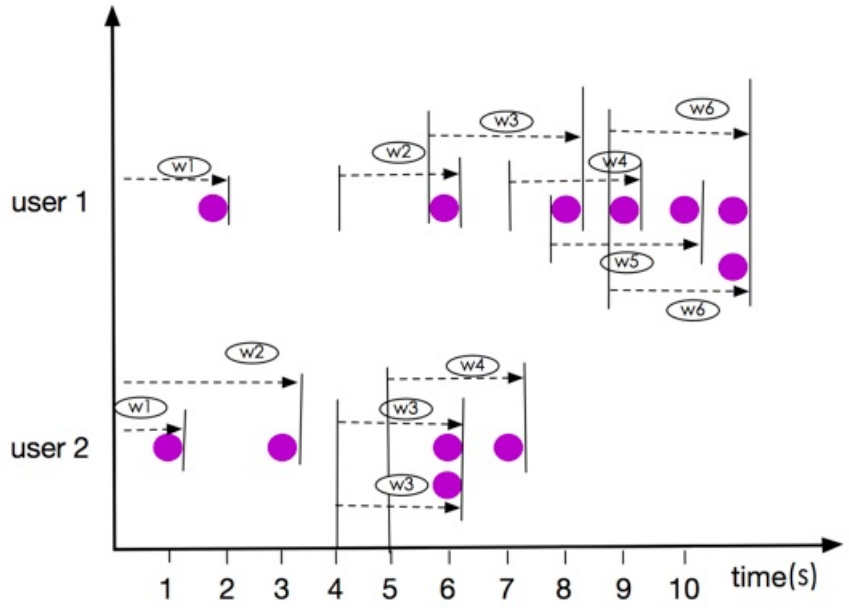
  RANGE OVER Window所有具有共同元素值（元素时间戳）的元素行确定一个窗口，RANGE OVER Window分为Unbounded和Bounded的两种情况。

  Unbounded RANGE OVER Window数据示例如下图所示。



> **说明** 上图所示窗口user1的w7、user2的窗口w3，两个元素同一时刻到达，属于相同的window，这一点与ROWS OVER Window不同。

Bounded RANGE OVER Window数据，以3秒中数据 `(INTERVAL '2' SECOND)` 的窗口为例，如下图所示。



> **说明** 上图所示窗口user1的w6、user2的窗口w3，元素都是同一时刻到达，属于相同的window，这一点与ROWS OVER Window不同。

- 窗口语法

```
SELECT
    agg1(col1) OVER(
     [PARTITION BY (value_expression1,..., value_expressionN)]
     ORDER BY timeCol
     RANGE
     BETWEEN (UNBOUNDED | timeInterval) PRECEDING AND CURRENT ROW) AS colName,
...
FROM Tab1;
```

  - value_expression：进行分区的字表达式。
  - timeCol：元素排序的时间字段。
  - timeInterval：定义根据当前行开始向前追溯指定时间的元素行。

- 案例

  Bounded RANGE OVER Window场景示例：假设一张商品上架表，包含有商品ID、商品类型、商品上架时间、商品价格数据。需要求比当前商品上架时间早2分钟的同类商品中的最高价格。

- 测试数据

商品ID	商品类型	上架时间	销售价格
ITEM001	Electronic	2017-11-11 10:01:00	20
ITEM002	Electronic	2017-11-11 10:02:00	50
ITEM003	Electronic	2017-11-11 10:03:00	30
ITEM004	Electronic	2017-11-11 10:03:00	60
ITEM005	Electronic	2017-11-11 10:05:00	40
ITEM006	Electronic	2017-11-11 10:06:00	20
ITEM007	Electronic	2017-11-11 10:07:00	70
ITEM008	Clothes	2017-11-11 10:08:00	20

- 测试代码

```
CREATE TABLE tmall_item(
   itemID VARCHAR,
   itemType VARCHAR,
   onSellTime TIMESTAMP,
   price DOUBLE,
   WATERMARK onSellTime FOR onSellTime as withOffset(onSellTime, 0)
)
WITH (
  type = 'sls',
   ...
);
SELECT
    itemID,
    itemType,
    onSellTime,
    price,
    MAX(price) OVER (
        PARTITION BY itemType
        ORDER BY onSellTime
        RANGE BETWEEN INTERVAL '2' MINUTE preceding AND CURRENT ROW) AS maxPrice
  FROM tmall_item;
```

- 测试结果

itemID	itemType	onSellTime	price	maxPrice
ITEM001	Electronic	2017-11-11 10:01:00	20	20
ITEM002	Electronic	2017-11-11 10:02:00	50	50
ITEM003	Electronic	2017-11-11 10:03:00	30	50
ITEM004	Electronic	2017-11-11 10:03:00	60	60
ITEM005	Electronic	2017-11-11 10:05:00	40	60
ITEM006	Electronic	2017-11-11 10:06:00	20	40
ITEM007	Electronic	2017-11-11 10:07:00	70	70
ITEM008	Clothes	2017-11-11 10:08:00	20	20

4.10. 内置函数

4.10.1. 字符串函数

4.10.1.1. CHAR_LENGTH

本文为您介绍如何使用实时计算字符串函数CHAR_LENGTH。

语法

```
CHAR_LENGTH(A)
```

入参

参数	数据类型
A	VARCHAR

功能描述

返回字符串中的字符的数量。

示例

- 测试数据

var1(INT)
ss
231ee

- 测试语句

```
SELECT  CHAR_LENGTH(var1) as aa
FROM T1;
```

- 测试结果

aa(INT)
2
5

4.10.1.2. CHR

本文为您介绍如何使用实时计算字符串函数CHR。

语法

```
VARCHAR CHR(INT ascii)
```

入参

参数	数据类型	说明
ascii	INT	是0到255之间的整数。如果不在此范围内，则返回NULL。

功能描述

将ASCII码转换为字符。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
255	97	65

- 测试语句

```
SELECT CHR(int1) as var1, CHR(int2) as var2, CHR(int3) as var3
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
ÿ	a	A

4.10.1.3. CONCAT

本文为您介绍如何使用实时计算字符串函数CONCAT。

语法

```
VARCHAR CONCAT(VARCHAR var1, VARCHAR var2, ...)
```

入参

参数	数据类型	说明
var1	VARCHAR	普通字符串值
var2	VARCHAR	普通字符串值

功能描述

连接两个或多个字符串值从而组成一个新的字符串。如果任一参数为NULL时，则跳过该参数。

示例

- 测试数据

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
Hello	My	World
Hello	null	World
null	null	World
null	null	null

- 测试语句

```
SELECT CONCAT(var1, var2, var3) as var
FROM T1;
```

- 测试结果

var(VARCHAR)
HelloMyWorld
HelloWorld

var(VARCHAR)
World
N/A

4.10.1.4. CONCAT_WS

本文为您介绍如何使用实时计算字符串函数CONCAT_WS。

语法

```
VARCHAR CONCAT_WS(VARCHAR separator, VARCHAR var1, VARCHAR var2, ...)
```

入参

参数	数据类型	说明
separator	VARCHAR	分隔符
var1	VARCHAR	-
var2	VARCHAR	-

功能描述

将每个参数值和第一个参数separator指定的分隔符依次连接到一起组成新的字符串，长度和类型取决于输入值。

> **说明** 如果separator取值为null，则将separator视作与空串进行拼接。如果其它参数为NULL，在执行拼接过程中跳过取值为NULL的参数。

示例

- 测试数据

sep(VARCHAR)	str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)
|	Jack	Harry	John
null	Jack	Harry	John
|	null	Harry	John
|	Jack	null	null

- 测试语句

```
SELECT CONCAT_WS(sep, str1, str2, str3) as var FROM T1;
```

- 测试结果

var(VARCHAR)
Jack|Harry|John
JackHarryJohn
Harry|John
Jack

4.10.1.5. FROM_BASE64

本文为您介绍如何使用实时计算字符串函数FROM_BASE64。

语法

```
BINARY FROM_BASE64(str)
```

入参

参数	数据类型	说明
str	VARCHAR	base64编码的字符串。

功能描述

将base64编码的字符串str解析成对应的binary类型数据输出。

示例

- 测试数据

a(INT)	b(BIGINT)	c(VARCHAR)
1	1L	null

- 测试语句

```
SELECT
from_base64(c) as var1,from_base64('SGVsbG8gd29ybGQ=') as var2
FROM T1;
```

- 测试结果

var1(BINARY)	var2(BINARY)
null	Byte Array: [72('H'), 101('e'), 108('l'), 108('l'), 111('o'), 32(' '), 119('w'), 111('o'), 114('r'), 108('l'), 100('d')]

4.10.1.6. HASH_CODE

本文为您介绍如何使用实时计算字符串函数HASH_CODE。

语法

```
INT HASH_CODE(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

返回字符串的HASH_CODE()的绝对值。

示例

- 测试数据

str1(VARCHAR)	str2(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	k1:v1,k2:v2	null

- 测试语句

```
SELECT  HASH_CODE(str1) as var1, HASH_CODE(str2) as var2, HASH_CODE(nullstr) as var3
FROM T1;
```

- 测试结果

var1(INT)	var2(INT)	var3(INT)
1099348823	401392878	null

4.10.1.7. INITCAP

本文为您介绍如何使用实时计算字符串函数INITCAP。

语法

```
VARCHAR INITCAP(A)
```

入参

参数	数据类型
A	VARCHAR

功能描述

返回字符串，每个字转换器的第一个字母大写，其余为小写。

示例

- 测试数据

var1(VARCHAR)
aADvbn

- 测试语句

```
SELECT INITCAP(var1)as aa
FROM T1;
```

- 测试结果

aa(VARCHAR)
Aadvbn

4.10.1.8. INSTR

本文为您介绍如何使用实时计算Flink版字符串函数INSTR。

> **注意** 仅Blink 2.2.0及以上版本支持INSTR函数。

语法

```
INT instr( string1, string2 )
INT instr( string1, string2 [, start_position [, nth_appearance ] ] )
```

入参

参数	数据类型	说明
string1	VARCHAR	源字符串，要在该字符串中查找string2。
string2	VARCHAR	目标字符串，string1中查找的字符串。
start_position	INT	起始位置，表示在string1中开始查找的其实位置： • 该参数省略（默认）： 字符串索引从1开始。 • 该参数为正：从左到右开始检索。 • 该参数为负：从右到左开始检索。

参数	数据类型	说明
nth_appearance	INT	匹配序号代表要查找第几次出现的string2： • 该参数省略（默认）：第1次出现。 • 该参数为负：系统报错。

功能描述

返回目标字符串在源字符串中的位置，如果在源字符串中未找到目标字符串，则返回0。

示例

- 测试数据

string1(VARCHAR)
helloworld

- 测试语句

```
SELECT
instr('helloworld','lo') as res1,
instr('helloworld','l',-1,1) as res2,
instr('helloworld','l',3,2) as res3
FROM T1;
```

- 测试结果

res1(INT)	res2(INT)	res3(INT)
4	9	4

4.10.1.9. JSON_VALUE

本文为您介绍如何使用实时计算字符串函数JSON_VALUE。

语法

```
VARCHAR JSON_VALUE(VARCHAR content, VARCHAR path)
```

入参

- content

  VARCHAR类型，需要解析的JSON对象，使用字符串表示。

- path

  VARCHAR类型，解析JSON的路径表达式。目前path支持如下表达式。

符号	功能
$	根对象
[]	数组下标
*	数组通配符
.	取子元素

功能描述

从JSON字符串中提取指定path的值，不合法的JSON和null都统一返回null。

示例

- 测试数据

id(INT)	json(VARCHAR)	path1(VARCHAR)
1	[10, 20, [30, 40]]	$[2][*]
2	{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}	$.ccc.hhh[*]
3	{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}	$.ccc.hhh[1]
4	[10, 20, [30, 40]]	NULL
5	NULL	$[2][*]
6	"{xx]"	"$[2][*]"

- 测试语句

```
SELECT
    id,
    JSON_VALUE(json, path1) AS `value`
FROM
    T1;
```

- 测试结果

id (INT)	value (VARCHAR)
1	[30,40]
2	["h0","h1","h2"]
3	h1

id (INT)	value (VARCHAR)
4	NULL
5	NULL
6	NULL

4.10.1.10. KEYVALUE

本文为您介绍如何使用实时计算字符串函数KEYVALUE。

语法

```
VARCHAR KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR key_name)
```

入参

参数	数据类型	说明
str	VARCHAR	字符串中的key-value（kv）对。
split1	VARCHAR	kv对的分隔符。
split2	VARCHAR	kv的分隔符。
key_name	VARCHAR	键的名称

功能描述

解析str字符串中，匹配有split1（kv对的分隔符）和split2（kv的分隔符）的key-value对，根据key_name返回对应的数值。如果key_name值不存在或异常时，返回NULL。

示例

- 测试数据

str(VARCHAR)	split1(VARCHAR)	split2(VARCHAR)	key1(VARCHAR)
k1=v1;k2=v2	;	=	k2
null	;	|	:
k1:v1|k2:v2	null	=	:
k1:v1|k2:v2	|	=	null
k1:v1|k2:v2	|	=	:

- 测试语句

```
SELECT  KEYVALUE(str, split1, split2, key1) as `result`
FROM T1;
```

- 测试结果

result(VARCHAR)
v2
null
null
null
null

4.10.1.11. LOWER

本文为您介绍如何使用实时计算字符串函数LOWER。

语法

```
VARCHAR LOWER(A)
```

入参

- A
- VARCHAR类型。

功能描述

返回转换为小写字符的字符串。

示例

- 测试数据

var1(VARCHAR)
Ss
yyT

- 测试语句

```
SELECT LOWER(var1) as aa
FROM T1;
```

- 测试结果

aa(VARCHAR)
ss
yyt

4.10.1.12. LPAD

本文为您介绍如何使用实时计算字符串函数LPAD。

语法

```
VARCHAR LPAD(VARCHAR str, INT len, VARCHAR pad)
```

入参

参数	数据类型	说明
str	VARCHAR	启始的字符串。
len	INT	新的字符串的长度。
pad	VARCHAR	需要重复补充的字符串。

功能描述

字符串str左端填充若干个字符串pad，直到新的字符串达到指定长度len为止。

任意参数为null时返回null。

`len` 为负数时返回为null。

`pad` 为空串时，如果 `len` 不大于 `str` 长度，返回 `str` 裁剪后的结果。如果 `len` 大于 `str` 长度时，则返回null。

示例

- 测试数据

str(VARCHAR)	len(INT)	pad(VARCHAR)
空	-2	空
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null

str(VARCHAR)	len(INT)	pad(VARCHAR)
asd	2	空
空	2	s
asd	4	空
空	0	空

- 测试语句

```
SELECT LPAD(str, len, pad) AS result
FROM T1;
```

- 测试结果

result(VARCHAR)
null
JohnJHelloWorld
Jo
HelC
null
null
as
ss
null
空

4.10.1.13. MD5

本文为您介绍如何使用实时计算字符串函数MD5。

语法

```
VARCHAR MD5(VARCHAR str)
```

入参

- str
- VARCHAR类型

功能描述

返回字符串的MD5值。如果参数为空串（即参数为''）时，则返回空串。

示例

- 测试数据

str1(VARCHAR)	str2(VARCHAR)
k1=v1;k2=v2	空

- 测试语句

```
SELECT
   MD5(str1) as var1,
   MD5(str2) as var2
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)
19c17f42b4d6a90f7f9ffc2ea9bdd775	空

4.10.1.14. OVERLAY

本文为您介绍如何使用实时计算字符串函数OVERLAY。

语法

```
VARCHAR OVERLAY ( (VARCHAR x PLACING VARCHAR y FROM INT start_position [ FOR INT length ])
)
```

入参

参数	数据类型
x	VARCHAR
y	VARCHAR
start_position	INT
length（可选）	INT

功能描述

用y替换x的子串。从start_position开始，替换length+1个字符。

示例

- 测试语句

```
OVERLAY('abcdefg' PLACING 'hij' FROM 2 FOR 2) as result
FROM  T1;
```

- 测试结果

result(VARCHAR)
ahijdefg

4.10.1.15. PARSE_URL

本文为您介绍如何使用实时计算字符串函数PARSE_URL。

语法

```
VARCHAR PARSE_URL(VARCHAR urlStr, VARCHAR partToExtract [, VARCHAR key])
```

入参

参数	数据类型	说明
urlStr	VARCHAR	链接
partToExtract	VARCHAR	指定链接中解析的部分。取值如下： • HOST • PATH • QUERY • REF • PROTOCOL • FILE • AUTHORITY • USERINFO
key	VARCHAR	可选，指定截取部分中，具体的键。

功能描述

返回urlStr中指定的部分解析后的值。如果urlStr参数值为null时，则返回值为null。

示例

- 测试数据

url1(VARCHAR)	nullstr(VARCHAR)
http://facebook.com/path/p1.php?query=1	null

- 测试语句

```
SELECT PARSE_URL(url1, 'QUERY', 'query') as var1,
       PARSE_URL(url1, 'QUERY') as var2,
       PARSE_URL(url1, 'HOST') as var3,
       PARSE_URL(url1, 'PATH') as var4,
       PARSE_URL(url1, 'REF') as var5,
       PARSE_URL(url1, 'PROTOCOL') as var6,
       PARSE_URL(url1, 'FILE') as var7,
       PARSE_URL(url1, 'AUTHORITY') as var8,
       PARSE_URL(nullstr, 'QUERY') as var9,
       PARSE_URL(url1, 'USERINFO') as var10,
       PARSE_URL(nullstr, 'QUERY', 'query') as var11
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)	var11(VARCHAR)
1	query=1	facebook.com	/path/p1.php	null	http	/path/p1.php?query=1	facebook.com	null	null	null

4.10.1.16. POSITION

本文为您介绍如何使用实时计算字符串函数POSITION。

语法

```
INTEGER POSITION( x IN  y)
```

入参

参数	数据类型
x	VARCHAR
y	VARCHAR

功能描述

返回目标字符串x在被查询字符串y里第一次出现的位置。如果目标字符串x在被查询字符串y中不存在，返回值为0。

示例

- 测试语句

```
POSITION('in' IN 'china') as result
FROM T1;
```

- 测试结果

result(INT)
3

4.10.1.17. REGEXP

本文为您介绍如何使用实时计算字符串函数REGEXP。

语法

```
BOOLEAN REGEXP(VARCHAR str, VARCHAR pattern)
```

入参

参数	数据类型	说明
str	VARCHAR	指定的字符串。
pattern	VARCHAR	指定的匹配模式。

功能描述

对指定的字符串执行一个正则表达式搜索，并返回一个Boolean值表示是否找到指定的匹配模式。如果str或者pattern为空或为NULL时，则返回false。

示例

- 测试数据

str1(VARCHAR)	pattern1(VARCHAR)
k1=v1;k2=v2	k2*
k1:v1|k2:v2	k3
null	k3
k1:v1|k2:v2	null
k1:v1|k2:v2	(

- 测试语句

```
SELECT  REGEXP(str1, pattern1) AS result
FROM T1;
```

- 测试结果

result(BOOLEAN)
true
false
false
false
false

4.10.1.18. REGEXP_EXTRACT

本文为您介绍如何使用实时计算字符串函数REGEXP_EXTRACT。

语法

```
VARCHAR REGEXP_EXTRACT(VARCHAR str, VARCHAR pattern, INT index)
```

入参

参数	数据类型	说明
str	VARCHAR	指定的字符串。
pattern	VARCHAR	匹配的字符串。
index	INT	第几个被匹配的字符串。

> **注意** 正则常量请按照Java代码来写。CodeGen会将SQL常量字符串自动转化为Java代码。如果要描述一个数字\d，需要写成'\d'，即和Java中正则相同。

功能描述

使用正则模式Pattern匹配抽取字符串Str中的第Index个子串，Index从1开始，正则匹配提取。当参数为NULL或者正则不合法时，则返回NULL。

示例

- 测试数据

str1 (VARCHAR)	pattern1(VARCHAR)	index1 (INT)
foothebar	foo(.*?)(bar)	2
100-200	(\\d+)-(\\d+)	1
null	foo(.*?)(bar)	2

str1 (VARCHAR)	pattern1(VARCHAR)	index1 (INT)
foothebar	null	2
foothebar	空	2
foothebar	(	2

- 测试语句

```
SELECT  REGEXP_EXTRACT(str1, pattern1, index1) as result
FROM T1;
```

- 测试结果

result(VARCHAR)
bar
100
null
null
null
null

4.10.1.19. REGEXP_REPLACE

本文为您介绍如何使用实时计算字符串函数REGEXP_REPLACE。

语法

```
VARCHAR REGEXP_REPLACE(VARCHAR str, VARCHAR pattern, VARCHAR replacement)
```

入参

参数	数据类型	说明
str	VARCHAR	指定的字符串。
pattern	VARCHAR	被替换的字符串。
replacement	VARCHAR	用于替换的字符串。

注意 请您按照Java代码编写正则常量。Codegen会自动将SQL常量字符串转化为Java代码。描述一个数值（`\d`）的正则表达式和Java中一样，为 `'\d'`。

功能描述

用字符串 `replacement` 替换字符串 `str` 中正则模式为 `pattern` 的部分，并返回新的字符串。如果参数为NULL或者正则不合法时，则返回NULL。

如果您的业务数据中存在特殊字符（即系统使用的字符或者SQL关键字），则需要对特殊字符进行转义处理。常见的特殊字符和转义方式如下表所示。

特殊字符	字符的转义方式
'	\'
\	\$
.	\\.
$	\\$

示例

- 测试数据

str1(VARCHAR)	pattern1(VARCHAR)	replace1(VARCHAR)
2014-03-13	-	空
NULL	-	空
2014-03-13	-	NULL
2014-03-13	空	s
2014-03-13	(	s
100-200	(\d+)	num

- 测试语句

```
SELECT  REGEXP_REPLACE(str1, pattern1, replace1) as result
FROM T1;
```

- 测试结果

result(VARCHAR)
20140313
null
null
2014-03-13
null

result(VARCHAR)
num-num

4.10.1.20. REPEAT

本文为您介绍如何使用实时计算字符串函数REPEAT。

语法

```
VARCHAR REPEAT(VARCHAR str, INT n)
```

入参

参数	数据类型	说明
str	VARCHAR	重复字符串值。
n	INT	重复次数。

功能描述

返回以字符串值为str，重复次数为N的新的字符串。如果参数为null时，则返回null。如果重复次数为0或负数，则返回空串。

示例

- 测试数据

str(VARCHAR)	n(INT)
J	9
Hello	2
Hello	-9
null	9

- 测试语句

```
SELECT  REPEAT(str,n) as var1
FROM T1;
```

- 测试结果

var1(VARCHAR)
JJJJJJJJ
HelloHello

var1(VARCHAR)
空
null

4.10.1.21. REPLACE

本文为您介绍如何使用实时计算字符串函数REPLACE。

语法

```
VARCHAR REPLACE(str1, str2, str3)
```

入参

参数	数据类型	说明
str1	VARCHAR	原字符
str2	VARCHAR	目标字符
str3	VARCHAR	替换字符

功能描述

字符串替换函数。

示例

- 测试数据

str1(INT)	str2(INT)	str3(INT)
alibaba blink	blink	flink

- 测试语句

```
SELECT REPLACE(str1, str2, str3) as `result`
FROM T1;
```

- 测试结果

result(VARCHAR)
alibaba flink

4.10.1.22. REVERSE

本文为您介绍如何使用实时计算字符串函数REVERSE。

语法

```
VARCHAR REVERSE(VARCHAR str)
```

入参

参数	数据类型	说明
str	VARCHAR	普通字符串值。

功能描述

反转字符串，返回字符串值的相反顺序。如果任一参数为null时，则返回null。

示例

- 测试数据

str1(VARCHAR)	str2(VARCHAR)	str3(VARCHAR)	str4(VARCHAR)
iPhoneX	Alibaba	World	null

- 测试语句

```
SELECT  REVERSE(str1) as var1,REVERSE(str2) as var2,
        REVERSE(str3) as var3,REVERSE(str4) as var4
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)
XenohPi	ababilA	dlroW	null

4.10.1.23. RPAD

本文为您介绍如何使用实时计算字符串函数RPAD。

语法

```
VARCHAR RPAD(VARCHAR str, INT len, VARCHAR pad)
```

入参

参数	数据类型	说明
str	VARCHAR	启始的字符串。
len	INT	新的字符串的长度。

参数	数据类型	说明
pad	VARCHAR	需要重复补充的字符串。

功能描述

字符串str右端填充若干个字符串pad，直到新的字符串达到指定长度 `len` 为止。

- 如果任意参数为null时，则返回null。
- 如果 `len` 长度为负数时，则返回null。
- 当 `pad` 为空串，如果 `len` 不大于 `str` 长度时，则返回 `str` 裁剪后的结果。
- 如果 `len` 大于 `str` 长度，则返回null。

示例

- 测试数据

str(VARCHAR)	len(INT)	pad(VARCHAR)
空	-2	空
HelloWorld	15	John
John	2	C
C	4	HelloWorld
null	2	C
c	2	null
asd	2	空
空	2	s
asd	4	空
空	0	空

- 测试语句

```
SELECT  RPAD(str, len, pad) as result
FROM T1;
```

- 测试结果

result(VARCHAR)
null
HelloWorldJohnJ
Jo

result(VARCHAR)
CHel
null
null
as
ss
null
空

4.10.1.24. SPLIT_INDEX

本文为您介绍如何使用实时计算字符串函数SPLIT_INDEX。

语法

```
VARCHAR SPLIT_INDEX(VARCHAR str, VARCHAR sep, INT index)
```

入参

参数	数据类型	说明
str	VARCHAR	被分隔的字符串。
sep	VARCHAR	分隔符的字符串。
index	INT	截取的字段位置。

功能描述

以 `sep` 作为分隔符，将字符串 `str` 分隔成若干段，取其中的第 `index` 段。 `index` 从0开始，如果取不到字段，则返回null。如果任一参数为NULL，则返回null。

示例

- 测试数据

str(VARCHAR)	sep(VARCHAR)	index(INT)
Jack,John,Mary	,	2
Jack,John,Mary	,	3
Jack,John,Mary	null	0

str(VARCHAR)	sep(VARCHAR)	index(INT)
null	,	0

- 测试语句

```
SELECT  SPLIT_INDEX(str, sep, index) as var1
FROM T1;
```

- 测试结果

var1(VARCHAR)
Mary
null
null
null

4.10.1.25. STR_TO_MAP

本文为您介绍如何使用实时计算字符串函数STR_TO_MAP。

语法

```
MAP STR_TO_MAP(VARCHAR text)
MAP STR_TO_MAP(VARCHAR text, VARCHAR listDelimiter, VARCHAR keyValueDelimiter)
```

功能描述

使用listDelimiter将text分隔成K-V对，然后使用keyValueDelimiter分隔每个K-V对，组装成MAP返回。默认listDelimiter为（,），keyValueDelimiter为（=）。

入参

参数	数据类型	说明
text	VARCHAR	输入文本。
listDelimiter	VARCHAR	用来将text分隔成K-V对。默认为（,）。
keyValueDelimiter	VARCHAR	用来分隔每个key和value。默认为（=）。

> **注意** 这里的Delimiter使用的是Java的正则表达式，遇到特殊字符需要转义。

测试语句

```
SELECT
  STR_TO_MAP('k1=v1,k2=v2')['k1'] as a
FROM T1;
```

测试结果

a(VARCHAR)
v1

4.10.1.26. SUBSTRING

本文为您介绍如何使用实时计算字符串函数SUBSTRING。

语法

```
VARCHAR SUBSTRING(VARCHAR a, INT start)
VARCHAR SUBSTRING(VARCHAR a, INT start, INT len)
```

入参

参数	数据类型	说明
a	VARCHAR	指定的字符串。
start	INT	在字符串a中开始截取的位置。
len	INT	类截取的长度。

功能描述

获取字符串子串。截取从位置start开始，长度为len的子串。如果未指定len，则截取到字符串结尾。start从1开始，start为0当1看待，为负数时表示从字符串末尾倒序计算位置。

示例

- 测试数据

str(VARCHAR)	nullstr(VARCHAR)
k1=v1;k2=v2	null

- 测试语句

```
SELECT SUBSTRING('', 222222222) as var1,
       SUBSTRING(str, 2) as var2,
       SUBSTRING(str, -2) as var3,
       SUBSTRING(str, -2, 1) as var4,
       SUBSTRING(str, 2, 1) as var5,
       SUBSTRING(str, 22) as var6,
       SUBSTRING(str, -22) as var7,
       SUBSTRING(str, 1) as var8,
       SUBSTRING(str, 0) as var9,
       SUBSTRING(nullstr, 0) as var10
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)	var7(VARCHAR)	var8(VARCHAR)	var9(VARCHAR)	var10(VARCHAR)
空	1=v1;k2=v2	v2	v	1	空	空	k1=v1;k2=v2	k1=v1;k2=v2	null

4.10.1.27. TO_BASE64

本文为您介绍如何使用实时计算字符串函数TO_BASE64。

语法

```
VARCHAR TO_BASE64(bin)
```

入参

参数	数据类型
bin	BINARY

功能描述

将BINARY类型数据转换成对应base64编码的字符串输出。

示例

- 测试数据

c(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

- 测试语句

```
SELECT  TO_BASE64(FROM_BASE64(c)) as var1
FROM T1;
```

- 测试结果

var1(VARCHAR)
SGVsbG8gd29ybGQ=
SGk=
SGVsbG8=

4.10.1.28. TRIM

本文为您介绍如何使用实时计算字符串函数TRIM。

语法

```
VARCHAR TRIM( VARCHAR x )
```

入参

参数	数据类型
x	VARCHAR

功能描述

除掉一个字串中的字头或字尾。最常见的用途是移除字首或字尾的空格。

示例

- 测试语句

```
SELECT TRIM('   Sample   ') as result
FROM   T1;
```

- 测试结果

result(VARCHAR)
Sample

4.10.1.29. UPPER

本文为您介绍如何使用实时计算字符串函数UPPER。

语法

```
VARCHAR UPPER(A)
```

入参

参数	数据类型
A	VARCHAR

功能描述

返回转换为大写字符的字符串。

示例

- 测试数据

var1(VARCHAR)
ss
ttee

- 测试语句

```
SELECT UPPER(var1) as aa
FROM T1;
```

- 测试结果

aa(VARCHAR)
SS
TTEE

4.10.2. 数学函数

4.10.2.1. 加

本文为您介绍如何使用实时计算数学函数加法。

语法

```
A + B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A加B的结果。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
10	20	30

- 测试语句

```
SELECT int1+int2+int3 as aa
FROM T1;
```

- 测试结果

aa(int)
60

4.10.2.2. 减

本文为您介绍如何使用实时计算数学函数减法。

语法

```
A - B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A减B的结果。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
10	10	30

- 测试语句

```
SELECT int3 - int2 - int1 as aa
FROM T1;
```

- 测试结果

aa(int)
10

4.10.2.3. 乘

本文为您介绍如何使用实时计算数学函数乘法。

语法

```
A * B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A乘以B的结果。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
10	20	3

- 测试语句

```
SELECT int1*int2*int3 as aa
FROM T1;
```

- 测试结果

aa(int)
600

4.10.2.4. 除

本文为您介绍如何使用实时计算数学函数除法。

语法

```
A / B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

返回A除以B的结果。

示例

- 测试数据

int1(INT)	int2(INT)
8	4

- 测试语句

```
SELECT int1 / int2 as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
2.0

4.10.2.5. ABS

本文为您介绍如何使用实时计算数学函数ABS。

语法

```
DOUBLE ABS(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的绝对值。

示例

- 测试数据

in1(DOUBLE)
4.3

- 测试语句

```
SELECT ABS(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
4.3

4.10.2.6. ACOS

本文为您介绍如何使用实时计算数学函数ACOS。

语法

```
ACOS (A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的反余弦值。

示例

- 测试数据

in1(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ACOS(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.7707963267948966
1.1592794807274085

4.10.2.7. BIN

本文为您介绍如何使用实时计算数学函数BIN。

语法

```
VARCHAR BIN(BIGINT number)
```

入参

参数	数据类型
number	BIGINT 说明 参数的隐式转换支持INT，不支持其他类型。

功能描述

将长整型参数转换为二进制形式，返回类型为字符串。

示例

- 测试数据

id(INT)	x(BIGINT)
1	12L
2	10L
3	0L
4	10000000000L

说明 测试数据x列中的字母 `L` 代表的是数据类型Long，并不是做二进制转换的数值的一部分。

- 测试语句

```
SELECT id, bin(x) as var1
FROM T1;
```

- 测试结果

id(INT)	var1(VARCHAR)
1	1100
2	1010
3	0
4	100101010000001011111001000000000

4.10.2.8. ASIN

本文为您介绍如何使用实时计算数学函数ASIN。

语法

```
DOUBLE ASIN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的反正弦值。

示例

- 测试数据

in1(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ASIN(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.8
0.41151684606748806

4.10.2.9. ATAN

本文为您介绍如何使用实时计算数学函数ATAN。

语法

```
DOUBLE ATAN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的反正切值。

示例

- 测试数据

in1(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ATAN(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.6222796222326533
0.3805063771123649

4.10.2.10. BITAND

本文为您介绍如何使用实时计算数学函数BITAND。

语法

```
INT BITAND(INT number1,INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

功能描述

运算符按位“与”操作。输入和输出类型均为INT整型，且类型一致。

示例

- 测试数据

a(INT)	b(INT)
2	3

- 测试语句

```
SELECT BITAND(a, b) as intt
FROM T1;
```

- 测试结果

intt(INT)
2

4.10.2.11. BITNOT

本文为您介绍如何使用实时计算数学函数BIT NOT。

语法

```
INT BITNOT(INT number)
```

入参

参数	数据类型
number	INT

功能描述

按位取反。输入和输出类型均为整型，且类型一致。

示例

- 测试数据

a(INT)
7

- 测试语句

```
SELECT BITNOT(a) as var1
FROM T1;
```

- 测试结果

var1(INT)
0xfff8

4.10.2.12. BITOR

本文为您介绍如何使用实时计算数学函数BITOR。

语法

```
INT BITOR(INT number1, INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

功能描述

按位取“或”。输入和输出类型均为整型，且类型一致。

示例

- 测试数据

a(INT)	b(INT)
2	3

- 测试语句

```
SELECT BITOR(a, b) as var1
FROM T1;
```

- 测试结果

var1(INT)
3

4.10.2.13. BITXOR

本文为您介绍如何使用实时计算数学函数BITXOR。

语法

```
INT BITXOR(INT number1, INT number2)
```

入参

参数	数据类型
number1	INT
number2	INT

功能描述

按位取“异或”。输入和输出类型均为整型，且类型一致。

示例

- 测试数据

a(INT)	b(INT)
2	3

- 测试语句

```
SELECT BITXOR(a, b) as var1
FROM T1;
```

- 测试结果

var1(INT)
1

4.10.2.14. CARDINALITY

本文为您介绍如何使用实时计算数学函数CARDINALITY。

语法

```
CARDINALITY(str)
```

入参

参数	数据类型
number1	数组

功能描述

返回一个集合中的元素数量。

示例

- 测试语句

```
SELECT cardinality(array[1,2,3]) AS `result`
FROM T1;
```

- 测试结果

result(INT)
3

4.10.2.15. CONV

本文为您介绍如何使用实时计算进制转换内置函数CONV。

> **说明** blink-3.2.2及以上版本支持该函数。

语法

```
VARCHAR CONV(BIGINT number, INT FROM_BASE, INT TO_BASE)
or
VARCHAR CONV(VARCHAR number, INT FROM_BASE, INT TO_BASE)
```

入参

参数	数据类型
number	BIGINT、VARCHAR。
FROM_BASE	INT，非负数， 取值范围[2, 36]。
TO_BASE	INT，可以为正数（无符号整数）、负数（有符号整数）、ABS(TO_BASE)，取值范围[2, 36]。

功能描述

将长整型数字或字符串形式数字从一种进制转换为另一种进制，返回类型为字符串，CONV()精度为64位。

> **说明** 当number是null或非法字符时，结果返回为NULL。

样例

- 测试数据

id(INT)	x(BIGINT)	y (VARCHAR)
1	12L	'12'
2	10L	'10'
3	0L	'test'
4	NULL	NULL

- 测试案例

```
SELECT  id, conv(x, 10, 16) as var1, conv(y, 10, 2) as var2
FROM T1;
```

- 测试结果

id(INT)	var1(VARCHAR)	var2(VARCHAR)
1	C	1100
2	A	1010
3	0	NULL
4	NULL	NULL

4.10.2.16. COS

本文为您介绍如何使用实时计算数学函数COS。

语法

```
DOUBLE COS(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的余弦值。

示例

- 测试数据

in1(DOUBLE)
0.8
0.4

- 测试语句

```
SELECT COS(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.6967067093471654
0.9210609940028851

4.10.2.17. COT

本文为您介绍如何使用实时计算数学函数COT。

语法

```
DOUBLE COT(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的余切值。

示例

- 测试数据

in1(DOUBLE)
0.8

in1(DOUBLE)
0.4

- 测试语句

```
SELECT COT(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

4.10.2.18. EXP

本文为您介绍如何使用实时计算数学函数EXP。

语法

```
DOUBLE EXP(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回自然常数e的A次幂的DOUBLE类型数值。

示例

- 测试数据

in1(DOUBLE)
8.0
10.0

- 测试语句

```
SELECT EXP(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
2980.9579870417283
22026.465794806718

4.10.2.19. E

本文为您介绍如何使用实时计算数学函数E。

语法

```
DOUBLE E()
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回自然常数e的DOUBLE类型值。

示例

- 测试数据

in1(DOUBLE)
8.0
10.0

- 测试语句

```
SELECT  e() as dou1, E() as dou2
FROM T1;
```

- 测试结果

dou1(DOUBLE)	dou2(DOUBLE)
2.718281828459045	2.718281828459045

4.10.2.20. FLOOR

本文为您介绍如何使用实时计算数学函数FLOOR。

语法

```
B FLOOR(A)
```

入参

参数	数据类型
A	INT、BIGINT、FLOAT、DOUBLE

功能描述

返回小于或等于A的最大整数数值，即舍去小数位的数值。B的数据类型与A的数据类型一致。

示例

- 测试数据

in1(DOUBLE)	in2(BIGINT)
8.123	3

- 测试语句

```
SELECT
FLOOR(in1) as out1,
FLOOR(in2) as out2
FROM T1;
```

- 测试结果

out1(DOUBLE)	out2(BIGINT)
8.0	3

4.10.2.21. LN

本文为您介绍如何使用实时计算数学函数LN。

语法

```
DOUBLE ln(DOUBLE number)
```

入参

参数	数据类型

参数	数据类型
number	DOUBLE 说明 若输入为VARCHAR类型或BIGINT类型，会隐式转换到DOUBLE类型后参与运算。

功能描述

返回number的自然对数。返回值是DOUBLE类型的对数值。

示例

- 测试数据

ID(INT)	X(DOUBLE)
1	100.0
2	8.0

- 测试语句

```
SELECT id, ln(x) as dou1, ln(e()) as dou2
FROM T1;
```

- 测试结果

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	4.605170185988092	1.0
2	2.0794415416798357	1.0

4.10.2.22. LOG

本文为您介绍如何使用实时计算数学函数LOG。

语法

```
DOUBLE LOG(DOUBLE base, DOUBLE x)
DOUBLE LOG(DOUBLE x)
```

入参

参数	数据类型
base	DOUBLE

参数	数据类型
x	DOUBLE

功能描述

返回以base为底的x的自然对数。返回值是DOUBLE类型的对数值。如果是只有一个参数的，返回以x为底的自然对数。

示例

- 测试数据

ID(INT)	BASE(DOUBLE)	X(DOUBLE)
1	10.0	100.0
2	2.0	8.0

- 测试语句

```
SELECT id, LOG(base, x) as dou1, LOG(2) as dou2
FROM T1;
```

- 测试结果

ID(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	2.0	0.6931471805599453
2	3.0	0.6931471805599453

4.10.2.23. LOG10

本文为您介绍如何使用实时计算数学函数LOG10。

语法

```
DOUBLE LOG10(DOUBLE x)
```

入参

参数	数据类型
x	DOUBLE

功能描述

返回以10为底的自然对数。若x为NULL，则返回NULL。若x为负数会引发异常。

示例

- 测试数据

id(INT)	X(INT)
1	100
2	10

- 测试语句

```
SELECT id, log10(x) as dou1
FROM T1;
```

- 测试结果

id(INT)	dou1(DOUBLE)
1	2.0
2	1.0

4.10.2.24. LOG2

本文为您介绍如何使用实时计算数学函数LOG2。

语法

```
DOUBLE LOG2(DOUBLE x)
```

入参

参数	数据类型
x	DOUBLE

功能描述

返回以2为底的自然对数。若x为NULL，则返回NULL。若x为负数，会引发异常。

示例

- 测试数据

id(INT)	X(INT)
1	8
2	2

- 测试语句

```
SELECT id, log2(x) as dou1
FROM T1;
```

- 测试结果

id(INT)	dou1(DOUBLE)
1	3.0
2	1.0

4.10.2.25. PI

本文为您介绍如何使用实时计算数学函数PI。

语法

```
DOUBLE PI()
```

功能描述

获取圆周率常量PI值。

示例

- 测试数据

ID(INT)	X(INT)
1	8

- 测试语句

```
SELECT id, PI() as dou1
FROM T1;
```

- 测试结果

ID(INT)	dou1(DOUBLE)
1	3.141592653589793

4.10.2.26. POWER

本文为您介绍如何使用实时计算数学函数POWER。

语法

```
DOUBLE POWER(A, B)
```

入参

参数	数据类型
A	DOUBLE
B	DOUBLE

功能描述

返回A的B次幂。

示例

- 测试数据

in1(DOUBLE)	in2(DOUBLE)
2.0	4.0

- 测试语句

```
SELECT POWER(in1, in2) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
16.0

4.10.2.27. RAND

本文为您介绍如何使用实时计算数学函数RAND。

语法

```
DOUBLE RAND([BIGINT seed])
```

入参

参数	数据类型	说明
seed	BIGINT	Seed取值为随机数，决定随机数序列的起始值。

功能描述

返回大于等于0小于1的DOUBLE类型随机数。

示例

- 测试数据

id(INT)	X(INT)
1	8

- 测试语句

```
SELECT id, rand(1) as dou1, rand(3) as dou2
FROM T1;
```

- 测试结果

id(INT)	dou1(DOUBLE)	dou2(DOUBLE)
1	0.7308781907032909	0.731057369148862

4.10.2.28. SIN

本文为您介绍如何使用实时计算数学函数SIN。

语法

```
DOUBLE SIN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的正弦值。

示例

- 测试数据

in1(DOUBLE)
8.0
0.4

- 测试语句

```
SELECT SIN(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
0.9893582466233818
0.3894183423086505

4.10.2.29. SQRT

本文为您介绍如何使用实时计算数学函数SQRT。

语法

```
DOUBLE SQRT(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

返回A的平方根。

示例

- 测试数据

in1(DOUBLE)
8.0

- 测试语句

```
SELECT SQRT(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
2.8284271247461903

4.10.2.30. TAN

本文为您介绍如何使用实时计算数学函数TAN。

语法

```
DOUBLE TAN(A)
```

入参

参数	数据类型
A	DOUBLE

功能描述

计算A的正切值。

示例

- 测试数据

in1(DOUBLE)
0.8
0.4

- 测试语句

```
SELECT TAN(in1) as aa
FROM T1;
```

- 测试结果

aa(DOUBLE)
1.0296385570503641
0.4227932187381618

4.10.2.31. CEIL

本文为您介绍如何使用实时计算数学函数CEIL。

语法

```
B CEIL(A)
```

入参

参数	数据类型
A	INT、BIGINT、FLOAT或DOUBLE
B	INT、BIGINT、FLOAT或DOUBLE

功能描述

输出B为大于或等于输入值A的最小整数数值。输出B的数据类型与输入参数A的数据类型一致。

示例

- 测试数据

in1(INT)	in2（DOUBLE）
1	2.3

- 测试语句

```
SELECT
CEIL （in1） as out1
CEIL （in2） as out2
FROM T1;
```

- 测试结果

out1(INT)	out2(DOUBLE)
1	3.0

4.10.2.32. CHARACTER_LENGTH

本文为您介绍如何使用实时计算数学函数CHARACTER_LENGTH。

语法

```
INTEGER CHARACTER_LENGTH( VARCHAR x )
```

入参

参数	数据类型
x	VARCHAR

功能描述

返回字符串x中的字符数。

示例

- 测试数据

ID(INT)	X(VARCHAR)
1	StreamCompute

- 测试语句

```
SELECT CHARACTER_LENGTH( x ) as result
FROM T1;
```

- 测试结果

ID(INT)	result(INT)
1	13

4.10.2.33. DEGREES

本文为您介绍如何使用实时计算数学函数DEGREES。

语法

```
DOUBLE DEGREES( double x )
```

入参

参数	数据类型
x	DOUBLE

功能描述

将弧度x转换为数值。

示例

- 测试语句

```
SELECT DEGREES (PI()) as result
FROM T1;
```

- 测试结果

result(DOUBLE)
180.0

4.10.2.34. MOD

本文为您介绍如何使用实时计算数学函数MOD。

语法

```
INTEGER MOD(INTEGER x,INTEGER y)
```

入参

参数	数据类型
x	INTEGER

参数	数据类型
y	INTEGER

功能描述

整数运算中，求整数x除以整数y的余数。当x为负值时，或者x、y均为负值时，结果为负值。

示例

- 测试数据

X（INT）	Y（INT）
29	3
-29	3
-29	-3

- 测试语句

```
SELECT MOD(x,y) as result
FROM T1;
```

- 测试结果

ID(INT)	result(INT)
1	2
2	-2
3	-2

4.10.2.35. ROUND

本文为您介绍如何使用实时计算数学函数ROUND。

语法

```
T ROUND( T x, INT n)
```

入参

参数	数据类型
x	T参数类型，支持DECIMAL、TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE
n	INT

功能描述

把数值x字段舍入为指定的小数n位数。

示例1

- 测试数据 T1

in1(DECIMAL)
0.7173560908995228
0.4

- 测试语句

```
SELECT ROUND(in1,2) as `result`
FROM T1;
```

- 测试结果

result(DECIMAL)
0.72
0.40

示例2

- 测试数据 T2

in2(DOUBLE)
0.7173560908995228
0.4

- 测试语句

```
SELECT ROUND(in2,2) as `result`
FROM T2;
```

- 测试结果

result(DOUBLE)
0.72
0.4

4.10.3. 日期函数

4.10.3.1. LOCALTIMESTAMP

本文为您介绍如何使用实时计算字符串函数LOCALTIMESTAMP。

语法

```
timestamp LOCALTIMESTAMP
```

入参

无

功能描述

返回当前系统的时间戳。

示例

- 测试语句

```
SELECT
LOCALTIMESTAMP as `result`
FROM T1;
```

- 测试结果

result (TIMESTAMP)
2018-07-27 14:04:38.998

4.10.3.2. CURRENT_DATE

本文为您介绍如何使用实时计算日期函数CURRENT_DATE。

语法

```
CURRENT_DATE
```

功能描述

返回当前系统日期。

示例

- 测试语句

```
SELECT CURRENT_DATE as res
FROM T1;
```

- 测试结果

res(DATE)
2018-09-20

4.10.3.3. CURRENT_TIMESTAMP

本文为您介绍如何使用实时计算日期函数CURRENT_TIMESTAMP。

语法

```
TIMESTAMP CURRENT_TIMESTAMP
```

> **说明** Blink 3.6.0以下版本，语法格式为TIMESTAMP CURRENT_TIMESTAMP（）。

功能描述

返回当前UTC（GMT+0）时间戳，时间戳单位为毫秒。

示例

- 测试语句

```
SELECT CURRENT_TIMESTAMP as var1
FROM T1;
```

- 测试结果

var1(TIMESTAMP)
2007-04-30 13:10:02.047

4.10.3.4. DATEDIFF

本文为您介绍如何使用实时计算日期函数DATEDIFF。

> **说明** 建议在实时计算3.3.0及以上版本使用该函数，在3.3.0以下版本使用该函数时，数据结果可能不符合您的预期。

语法

```
INT DATEDIFF(VARCHAR enddate, VARCHAR startdate)
INT DATEDIFF(TIMESTAMP enddate, VARCHAR startdate)
INT DATEDIFF(VARCHAR enddate, TIMESTAMP startdate)
INT DATEDIFF(TIMESTAMP enddate, TIMESTAMP startdate)
```

入参

参数	数据类型
startdate	TIMESTAMP或VARCHAR
enddate	TIMESTAMP或VARCHAR

> **说明** VARCHAR日期格式：yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。

功能描述

计算从enddate到startdate两个时间的天数差值，返回整数。若有参数为NULL或解析错误，返回NULL。

示例

- 测试数据

datetime1(VARCHAR)	datetime2(VARCHAR)	nullstr(VARCHAR)
2017-10-15 00:00:00	2017-09-15 00:00:00	null

- 测试语句

```
SELECT  DATEDIFF(datetime1, datetime2) as int1,
        DATEDIFF(TIMESTAMP '2017-10-15 23:00:00',datetime2) as int2,
        DATEDIFF(datetime2,TIMESTAMP '2017-10-15 23:00:00') as int3,
        DATEDIFF(datetime2,nullstr) as int4,
        DATEDIFF(nullstr,TIMESTAMP '2017-10-15 23:00:00') as int5,
        DATEDIFF(nullstr,datetime2) as int6,
        DATEDIFF(TIMESTAMP '2017-10-15 23:00:00',TIMESTAMP '2017-9-15 00:00:00')as int7
FROM T1;
```

- 测试结果

int1(INT)	int2(INT)	int3(INT)	int4(INT)	int5(INT)	int6(INT)	int7(INT)
30	31	-31	null	null	null	31

4.10.3.5. DATE_ADD

本文为您介绍如何使用实时计算日期函数DATE_ADD。

语法

```
VARCHAR DATE_ADD(VARCHAR startdate, INT days)
VARCHAR DATE_ADD(TIMESTAMP time, INT days)
```

入参

参数	数据类型
startdate	TIMESTAMP或VARCHAR 说明 VARCHAR类型日期格式：yyyy-MM-dd或 yyyy-MM-dd HH:mm:ss。
enddate	TIMESTAMP
days	INT

功能描述

返回指定startdate日期days天数后的VARCHAR类型日期，返回string格式的日期为 `yyyy-MM-dd` 。如果有参数为null或解析错误，返回null。

示例

- 测试数据

datetime1(VATCHAR)	nullstr(VATCHAR)
2017-09-15 00:00:00	null

- 测试语句

```
SELECT DATE_ADD(datetime1, 30) as var1,
 DATE_ADD(TIMESTAMP '2017-09-15 23:00:00',30) as var2,
 DATE_ADD(nullstr,30) as var3
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-10-15	2017-10-15	null

4.10.3.6. DATE_FORMAT

本文为您介绍如何使用实时计算日期函数DATE_FORMAT。

语法

```
VARCHAR DATE_FORMAT(TIMESTAMP time, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR to_format)
VARCHAR DATE_FORMAT(VARCHAR date, VARCHAR from_format, VARCHAR to_format)
```

入参

参数	数据类型
date	VARCHAR 说明 默认日期格式：yyyy-MM-dd HH:mm:ss。
time	TIMESTAMP
from_format	VARCHAR
to_format	VARCHAR

功能描述

将字符串类型的日期从源格式转换至目标格式。第一个参数（time 或 date）为源字符串。第二个参数from_format可选，为源字符串的格式，默认为yyyy-MM-dd hh:mm:ss。第三个参数为返回日期的格式，返回值为转换格式后的字符串类型日期。如果有参数为NULL或解析错误，则返回NULL。

示例

- 测试数据

date1(VARCHAR)	datetime1(VARCHAR)	nullstr(VARCHAR)
0915-2017	2017-09-15 00:00:00	NULL

- 测试语句

```
SELECT DATE_FORMAT(datetime1, 'yyMMdd') as var1,
 DATE_FORMAT(nullstr, 'yyMMdd') as var2,
 DATE_FORMAT(datetime1, nullstr) as var3,
 DATE_FORMAT(date1, 'MMdd-yyyy', nullstr) as var4,
 DATE_FORMAT(date1, 'MMdd-yyyy', 'yyyyMMdd') as var5,
 DATE_FORMAT(TIMESTAMP '2017-09-15 23:00:00', 'yyMMdd') as var6
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)	var4(VARCHAR)	var5(VARCHAR)	var6(VARCHAR)
170915	null	null	null	20170915	170915

4.10.3.7. DATE_SUB

本文为您介绍如何使用实时计算日期函数DATE_SUB。

语法

```
VARCHAR DATE_SUB(VARCHAR startdate, INT days)
VARCHAR DATE_SUB(TIMESTAMP time, INT days)
```

入参

参数	数据类型
startdate	VARCHAR 说明 VARCHAR类型日期格式：yyyy-MM-dd或yyyy-MM-dd HH:mm:ss。
time	TIMESTAMP
days	INT

功能描述

返回startdate减去days天数的日期。返回VARCHAR类型的yyyy-MM-dd日期格式。如果有参数为null或解析错误，返回null。

示例

- 测试数据

date1(VARCHAR)	nullstr(VARCHAR)
2017-10-15	null

- 测试语句

```
SELECT DATE_SUB(date1, 30) as var1,
 DATE_SUB(TIMESTAMP '2017-10-15 23:00:00',30) as var2,
 DATE_SUB(nullstr,30) as var3
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-09-15	2017-09-15	null

4.10.3.8. DAYOFMONTH

本文为您介绍如何使用实时计算日期函数DAYOFMONTH。

语法

```
BIGINT DAYOFMONTH(TIMESTAMP time)
BIGINT DAYOFMONTH(DATE date)
```

入参

参数	数据类型
date	DATE
time	TIMESTAMP

功能描述

返回输入时间参数date或time中所指代的“日”。返回值范围为1～31。

示例

- 测试数据

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate(DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

- 测试语句

```
SELECT DAYOFMONTH(TIMESTAMP '2016-09-15 00:00:00') as int1,
 DAYOFMONTH(DATE '2017-09-22') as int2,
 DAYOFMONTH(tdate) as int3,
 DAYOFMONTH(ts) as int4,
 DAYOFMONTH(CAST(dateStr AS DATE)) as int5,
 DAYOFMONTH(CAST(tsStr AS TIMESTAMP)) as int6
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
15	22	10	15	15	15

4.10.3.9. EXTRACT

本文为您介绍如何使用实时计算日期函数EXTRACT。

语法

```
BIGINT EXTRACT(unit FROM time)
```

入参

参数	数据类型
time	任意日期表达式。

参数	数据类型
unit	可取值如下： • MICROSECOND • SECOND • MINUTE • HOUR • DAY • WEEK • MONTH • QUARTER • YEAR • SECOND_MICROSECOND • MINUTE_MICROSECOND • MINUTE_SECOND • HOUR_MICROSECOND • HOUR_SECOND • HOUR_MINUTE • DAY_MICROSECOND • DAY_SECOND • DAY_MINUTE • DAY_HOUR • YEAR_MONTH

功能描述

返回日期/时间的单独部分，例如年、月、日、小时、分钟、周数等。

示例

- 测试语句

```
EXTRACT(YEAR FROM CURRENT_TIMESTAMP) AS OrderYear,
EXTRACT(MONTH FROM CURRENT_TIMESTAMP) AS OrderMonth,
EXTRACT(DAY FROM CURRENT_TIMESTAMP) AS OrderDay,
EXTRACT(WEEK FROM CURRENT_TIMESTAMP) AS OrderWeek
```

- 测试结果

OrderYear(BIGINT)	OrderMonth(BIGINT)	OrderDay(BIGINT)	OrderWeek(BIGINT)
2018	10	11	41

4.10.3.10. FROM_UNIXTIME

本文为您介绍如何使用实时计算日期函数FROM_UNIXTIME。

语法

```
VARCHAR FROM_UNIXTIME(BIGINT unixtime[, VARCHAR format])
```

入参

参数	数据类型
unixtime	BIGINT
format	VARCHAR

> **说明**
> - 参数unixtime为长整型，是以秒为单位的时间戳。
> - 参数format可选，为日期格式，默认格式为yyyy-MM-dd HH:mm:ss，表示返回VARCHAR类型的符合指定格式的日期，如果有参数为null或解析错误，则返回null。

功能描述

返回值为VARCHAR类型的日期值，默认日期格式：yyyy-MM-dd HH:mm:ss，若指定日期格式按指定格式输出任一输入参数是NULL，返回NULL。

示例

- 测试数据

unixtime1(BIGINT)	nullstr(VARCHAR)
1505404800	null

- 测试语句

```
SELECT FROM_UNIXTIME(unixtime1) as var1,
 FROM_UNIXTIME(unixtime1,'MMdd-yyyy') as var2,
 FROM_UNIXTIME(unixtime1,nullstr) as var3
FROM T1;
```

- 测试结果

var1(VARCHAR)	var2(VARCHAR)	var3(VARCHAR)
2017-09-15 00:00:00	0915-2017	null

4.10.3.11. HOUR

本文为您介绍如何使用实时计算日期函数HOUR。

语法

```
BIGINT HOUR(TIME time)
BIGINT HOUR(TIMESTAMP timestamp)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数time或timestamp中的24小时制的小时数，范围0～23。

示例

- 测试数据

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- 测试语句

```
SELECT HOUR(TIMESTAMP '2016-09-20 23:33:33') AS int1,
 HOUR(TIME '23:30:33') AS int2,
 HOUR(time2) AS int3,
 HOUR(timestamp1) AS int4,
 HOUR(CAST(time1 AS TIME)) AS int5,
 HOUR(TO_TIMESTAMP(datetime1)) AS int6
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
23	23	22	11	22	11

4.10.3.12. LOCALTIME

本文为您介绍如何使用实时计算日期函数LOCALTIME。

语法

```
TIME LOCALTIME
```

功能描述

以数据类型TIME的值返回会话时区中的当前时间。LOCALTIME可当变量直接使用。

示例

- 测试语句

```
SELECT LOCALTIME as `result`
FROM T1;
```

- 测试结果

result(TIME)
19:00:47

4.10.3.13. MINUTE

本文为您介绍如何使用实时计算日期函数MINUTE。

语法

```
BIGINT MINUTE(TIME time)
BIGINT MINUTE(TIMESTAMP timestamp)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数中time或timestamp中的“分钟”部分。取值范围0～59。

示例

- 测试数据

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- 测试语句

```
SELECT MINUTE(TIMESTAMP '2016-09-20 23:33:33') as int1,
 MINUTE(TIME '23:30:33') as int2,
 MINUTE(time2) as int3,
 MINUTE(timestamp1) as int4,
 MINUTE(CAST(time1 AS TIME)) as int5,
 MINUTE(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	30	23	12	23	12

4.10.3.14. MONTH

本文为您介绍如何使用实时计算日期函数MONTH。

语法

```
BIGINT MONTH(TIMESTAMP timestamp)
BIGINT MONTH(DATE date)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数中的月，范围1～12。

示例

- 测试数据

a(TIMESTAMP)	b(DATE)
2016-09-15 00:00:00	2017-10-15

- 测试语句

```
SELECT
 MONTH(cast( a as TIMESTAMP)) as int1,
 MONTH(cast( b as DATE)) as int2
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)
9	10

4.10.3.15. NOW

本文为您介绍如何使用实时计算日期函数NOW。

语法

```
BIGINT NOW()
BIGINT NOW(a)
```

入参

参数	数据类型
a	INT

功能描述

- 未指定参数时返回当前时区时间的时间戳，单位为秒。
- 可以在括号内输入INT类型参数作为偏移值（单位：秒），返回偏移后的时间戳。例如，`now(100)`返回当前时间戳加100秒的时间戳。

> **说明** 偏移值a为NULL时，NOW(a)返回值为NULL。

示例

- 测试数据 T1

a(INT)
null

- 测试语句

```
SELECT
 NOW() as now,
 NOW(100) as now_100,
 NOW(a) as now_null
FROM T1;
```

- 测试结果

now(BIGINT)	now_100(BIGINT)	now_null(BIGINT)
1403006911	1403007011	null

4.10.3.16. SECOND

本文为您介绍如何使用实时计算日期函数SECOND。

语法

```
BIGINT SECOND(TIMESTAMP timestamp)
 BIGINT SECOND(TIME time)
```

入参

参数	数据类型
time	TIME
timestamp	TIMESTAMP

功能描述

返回输入时间参数中的“秒”部分，范围0～59。

示例

- 测试数据

datetime1(VARCHAR)	time1(VARCHAR)	time2(TIME)	timestamp1(TIMESTAMP)
2017-10-15 11:12:13	22:23:24	22:23:24	2017-10-15 11:12:13

- 测试语句

```
SELECT SECOND(TIMESTAMP '2016-09-20 23:33:33') as int1,
 SECOND(TIME '23:30:33') as int2,
 SECOND(time2) as int3,
 SECOND(timestamp1) as int4,
 SECOND(CAST(time1 AS TIME)) as int5,
 SECOND(CAST(datetime1 AS TIMESTAMP)) as int6
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
33	33	24	13	24	13

4.10.3.17. TIMESTAMPADD

本文为您介绍如何使用实时计算日期函数TIMESTAMPADD。

语法

```
TIMESTAMP TIMESTAMPADD(interval,INT int_expr,TIMESTAMP datetime_expr)
DATE TIMESTAMPADD(interval,INT int_expr,DATE datetime_expr)
```

入参

参数	数据类型
interval	VARCHAR
int_expr	INT

参数	数据类型
datetime_expr	TIMESTAMP或DATE

interval可取值如下。

interval参数	时间间隔单位
FRAC_SECOND	毫秒
SECOND	秒
MINUTE	分钟
HOUR	小时
DAY	天
WEEK	星期
MONTH	月
QUARTER	季度
YEAR	年

功能描述

返回类型与datetime_expr类型相同。

将整型表达式int_expr添加日期或日期时间到表达式datetime_expr中，返回会话时区中的当前时间（数据类型TIME的值）。

示例

- 测试数据

a(TIMESTAMP)	b(DATE)
2018-07-09 10:23:56	1990-02-20

- 测试语句

```
SELECT
TIMESTAMPADD(HOUR,3,a) AS `result1`
TIMESTAMPADD(DAY,3,b) AS `result2`
FROM T1;
```

- 测试结果

result1(TIMESTAMP)	result2(DATE)
2018-07-09 13:23:56.0	1990-02-23

4.10.3.18. TO_DATE

本文为您介绍如何使用实时计算日期函数TO_DATE。

语法

```
Date TO_DATE(INT time)
Date TO_DATE(VARCHAR date)
Date TO_DATE(VARCHAR date,VARCHAR format)
```

入参

参数	数据类型
time	INT 说明 表示从1970-1-1到所表示时间之间天数。
date	VARCHAR 说明 默认格式为yyyy-MM-dd。
format	VARCHAR

功能描述

将INT类型的日期或者VARCHAR类型的日期转换成DATE类型。

示例

- 测试数据

date1(INT)	date2(VARCHAR)	date3(VARCHAR)
100	2017-09-15	20170915

- 测试语句

```
SELECT TO_DATE(date1) as var1,
 TO_DATE(date2) as var2,
 TO_DATE(date3,'yyyyMMdd') as var3
FROM T1;
```

- 测试结果

var1(DATE)	var2(DATE)	var3(DATE)
1970-04-11	2017-09-15	2017-09-15

4.10.3.19. TO_TIMESTAMP

本文为您介绍如何使用实时计算Flink版日期函数TO_TIMESTAMP。

语法

```
TIMESTAMP TO_TIMESTAMP(BIGINT time)
TIMESTAMP TO_TIMESTAMP(VARCHAR date)
TIMESTAMP TO_TIMESTAMP(VARCHAR date, VARCHAR format)
```

入参

参数	数据类型
time	BIGINT **说明** 单位为毫秒。
date	VARCHAR **说明** 默认格式为 `yyyy-MM-dd HH:mm:ss` 。如果您的date为非默认格式，请使用自定义函数编写Java代码进行转化，详情请参见自定义标量函数（UDF）。
format	VARCHAR

功能描述

将BIGINT类型的日期或者VARCHAR类型的日期转换成TIMESTAMP类型。

示例

- 测试数据

timestamp1(BIGINT)	timestamp2(VARCHAR)	timestamp3(VARCHAR)
1513135677000	2017-09-15 00:00:00	20170915000000

- 测试语句

```
SELECT TO_TIMESTAMP(timestamp1) as var1,
 TO_TIMESTAMP(timestamp2) as var2,
 TO_TIMESTAMP(timestamp3, 'yyyyMMddHHmmss') as var3
FROM T1;
```

- 测试结果

var1(TIMESTAMP)	var2(TIMESTAMP)	var3(TIMESTAMP)
2017-12-13 03:27:57.0	2017-09-15 00:00:00.0	2017-09-15 00:00:00.0

4.10.3.20. UNIX_TIMESTAMP

本文为您介绍如何使用实时计算日期函数UNIX_TIMESTAMP。

语法

```
BIGINT UNIX_TIMESTAMP()
BIGINT UNIX_TIMESTAMP(VARCHAR date)
BIGINT UNIX_TIMESTAMP(TIMESTAMP timestamp)
BIGINT UNIX_TIMESTAMP(VARCHAR date, VARCHAR format)
```

入参

参数	数据类型
timestamp	TIMESTAMP
date	VARCHAR **说明** 默认日期格式为 `yyyy-MM-dd HH:mm:ss` 。
format	VARCHAR **说明** 默认格式为 `yyyy-MM-dd hh:mm:ss` 。

功能描述

返回date转换成的长整型的时间戳，单位为秒。无参数时返回当前时间的时间戳，单位为秒，与now语义相同。如果有参数为null或解析错误，返回null。

示例

- 测试数据

nullstr（VARCHAR）
null

- 测试语句

```
SELECT UNIX_TIMESTAMP() as big1,
       UNIX_TIMESTAMP(nullstr) as big2
FROM T1;
```

- 测试结果

big1（BIGINT）	big2（BIGINT）
1403006911	null

4.10.3.21. WEEK

本文为您介绍如何使用实时计算日期函数WEEK。

语法

```
BIGINT WEEK(DATE date)
BIGINT WEEK(TIMESTAMP timestamp)
```

入参

参数	数据类型
date	DATE
timestamp	TIMESTAMP

功能描述

计算指定日期在一年中的第几周，周数取值区间1~53。

示例

- 测试数据

dateStr(VARCHAR)	date1(DATE)	ts1(TIMESTAMP)
2017-09-15	2017-11-10	2017-10-15 00:00:00

- 测试语句

```
SELECT WEEK(TIMESTAMP '2017-09-15 00:00:00') as int1,
 WEEK(date1) as int2,
 WEEK(ts1) as int3,
 WEEK(CAST(dateStr AS DATE)) as int4
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)
37	45	41	37

4.10.3.22. YEAR

本文为您介绍如何使用实时计算日期函数YEAR。

语法

```
BIGINT YEAR(TIMESTAMP timestamp)
BIGINT YEAR(DATE date)
```

入参

参数	数据类型
date	DATE
timestamp	TIMESTAMP

功能描述

返回输入时间的年份。

示例

- 测试数据

tsStr(VARCHAR)	dateStr(VARCHAR)	tdate(DATE)	ts(TIMESTAMP)
2017-10-15 00:00:00	2017-09-15	2017-11-10	2017-10-15 00:00:00

- 测试语句

```
SELECT YEAR(TIMESTAMP '2016-09-15 00:00:00') as int1,
 YEAR(DATE '2017-09-22') as int2,
 YEAR(tdate) as int3,
 YEAR(ts) as int4,
 YEAR(CAST(dateStr AS DATE)) as int5,
 YEAR(CAST(tsStr AS TIMESTAMP)) as int6
FROM T1;
```

- 测试结果

int1(BIGINT)	int2(BIGINT)	int3(BIGINT)	int4(BIGINT)	int5(BIGINT)	int6(BIGINT)
2016	2017	2017	2017	2015	2017

4.10.4. 逻辑函数

4.10.4.1. =

本文为您介绍如何使用实时计算逻辑运算函数=。

语法

```
A = B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A等于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	65	65

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 = int2;
```

- 测试结果

aa(int)
97

4.10.4.2. >

本文为您介绍如何使用实时计算逻辑运算函数>。

语法

```
A > B
```

入参

参数	数据类型
A	INT

参数	数据类型
B	INT

功能描述

如果A大于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	65	100

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 > int2;
```

- 测试结果

aa(int)
97

4.10.4.3. >=

本文为您介绍如何使用实时计算逻辑运算函数>=。

语法

```
A >= B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A大于等于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	65	65
9	6	61

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 >= int2;
```

- 测试结果

aa(int)
97
9

4.10.4.4. <=

本文为您介绍如何使用实时计算逻辑运算函数<=。

语法

```
A <= B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A小于等于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	66	65
9	6	5

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 <= int2;
```

- 测试结果

aa(int)
97
9

4.10.4.5. <

本文为您介绍如何使用实时计算逻辑运算函数<。

语法

```
A < B
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果A小于B，返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	66	65
9	6	5

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 < int2;
```

- 测试结果

aa(int)
97
9

4.10.4.6. <>

本文为您介绍如何使用实时计算逻辑运算函数<>。

语法

```
A <> B
```

入参

参数	数据类型
A	TIMESTAMP、BIGINT、INT、VARCHAR、DECIMAL
B	TIMESTAMP、BIGINT、INT、VARCHAR、DECIMAL

功能描述

如果A不等于B，则返回TRUE，否则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
97	66	6

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int3 <> int2;
```

- 测试结果

aa(int)
97

4.10.4.7. AND

本文为您介绍如何使用实时计算逻辑运算函数AND。

语法

```
A AND B
```

入参

参数	数据类型
A	BOOLEAN
B	BOOLEAN

功能描述

如果A和B均为TRUE，则为TRUE，否则为FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
255	97	65

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=255 AND int3=65;
```

- 测试结果

aa(int)
97

4.10.4.8. BETWEEN AND

本文为您介绍如何使用实时计算Flink版逻辑运算函数BETWEEN AND。

语法

```
A BETWEEN B AND C
```

入参

参数	数据类型
A	DOUBLE，BIGINT，INT，VARCHAR，DATE，TIMESTAMP，TIME
B	DOUBLE，BIGINT，INT，VARCHAR，DATE，TIMESTAMP，TIME

参数	数据类型
C	DOUBLE，BIGINT，INT，VARCHAR，DATE，TIMESTAMP，TIME

功能描述

BETWEEN操作符用于选取介于两个值之间的数据范围内的值。

示例1

- 测试数据

int1(INT)	int2(INT)	int3(INT)
90	80	100
11	10	7

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int1 BETWEEN int2 AND int3;
```

- 测试结果

aa(int)
90

示例2

- 测试数据

var1(varchar)	var2(varchar)	var3(varchar)
b	a	c

- 测试语句

```
SELECT var1 as aa
FROM T1
WHERE var1 BETWEEN var2 AND var3;
```

- 测试结果

aa(varchar)
b

示例3

- 测试数据

TIMESTAMP1(TIMESTAMP)	TIMESTAMP2(TIMESTAMP)	TIMESTAMP3(TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:20	1969-07-20 20:17:45

- 测试语句

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

- 测试结果

aa(TIMESTAMP)
1969-07-20 20:17:30

4.10.4.9. IS NOT FALSE

本文为您介绍如何使用实时计算逻辑运算函数IS NOT FALSE。

语法

```
A IS NOT FALSE
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE时，则返回TRUE。如果A是FALSE时，则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS NOT FALSE;
```

- 测试结果

aa(int)
97

4.10.4.10. IS NOT NULL

本文为您介绍如何使用实时计算逻辑运算函数IS NOT NULL。

语法

```
value IS NOT NULL
```

入参

参数	数据类型
value	任意数据类型

功能描述

如果value为 `NULL` 时，则返回 `FALSE` ，否则返回 `TRUE` 。

示例

- 测试数据

int1(INT)	int2(VARCHAR)
97	NULL
9	ww123

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NOT NULL;
```

- 测试结果

aa(int)
9

4.10.4.11. IS NOT TRUE

本文为您介绍如何使用实时计算逻辑运算函数IS NOT TRUE。

语法

```
A IS NOT TRUE
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE时，则返回FALSE。如果A是FALSE时，则返回TRUE。

示例

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int1=25 IS NOT TRUE;
```

- 测试结果

aa(int)
97

4.10.4.12. IS NOT UNKNOWN

本文为您介绍如何使用实时计算逻辑运算函数IS NOT UNKNOWN。

语法

```
A IS NOT UNKNOWN
```

入参

参数	数据类型
A	BOOLEAN

功能描述

A为逻辑比较表达式，例如：6<8。

正常情况下数值型与数值型作逻辑比较时，A值为TRUE或者FALSE。当其中一个不为数值型数据类型时，就会出现无法比较的情况。`IS NOT UNKNOWN`就是判断这种情况是否存在。 当两边无法进行正常的逻辑判断时，即A值既不是`TRUE`也不是`FALSE`，返回`FALSE`。可正常逻辑判断时，即A值为`TRUE`或者`FALSE`，返回`TRUE`。

示例一

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=25 IS NOT UNKNOWN;
```

- 测试结果

aa(int)
97

示例二

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1 < null IS NOT UNKNOWN;
```

- 测试结果

aa(int)
空

4.10.4.13. IS NULL

本文为您介绍如何使用实时计算逻辑运算函数IS NULL。

语法

```
value IS NULL
```

入参

参数	数据类型
value	任意数据类型

功能描述

如果value为 `NULL` 时，则返回 `TRUE` ，否则返回 `FALSE` 。

示例

- 测试数据

int1(INT)	int2(VARCHAR)
97	无
9	www

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int2 IS NULL;
```

- 测试结果

aa(int)
97

4.10.4.14. IS TRUE

本文为您介绍如何使用实时计算逻辑运算函数IS TRUE。

语法

```
A IS TRUE
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是TRUE时，则返回TRUE。如果A是FALSE时，则返回FALSE。

示例

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=255 IS TRUE;
```

- 测试结果

aa(int)
97

4.10.4.15. IS UNKNOWN

本文为您介绍如何使用实时计算逻辑运算函数IS UNKNOWN。

语法

```
A IS UNKNOWN
```

入参

参数	数据类型
A	BOOLEAN

功能描述

IS UNKNOWN通过逻辑判断关系，返回结果：

- A（逻辑比较表达式）值既不是 `TRUE` 也不是 `FALSE` ，即无法进行正常的逻辑判断，返回 `TRUE` 。
- A（逻辑比较表达式）值为 `TRUE` 或者 `FALSE` ，即可以进行正常逻辑判断，返回 `FALSE` 。

正常情况下数值型与数值型进行逻辑比较时（例如 `6<>8` ），A值为TRUE或者FALSE。但是，当其中一个不为数值型数据类型时，就会出现无法比较的情况。 `IS UNKNOWN` 用于判断这种情况是否存在。

示例一

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=25 IS UNKNOWN;
```

- 测试结果

aa(int)
空

示例二

- 测试数据

int1(INT)	int2(INT)
255	97

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1 > null IS UNKNOWN;
```

- 测试结果

aa(int)
97

4.10.4.16. LIKE

本文为您介绍如何使用实时计算逻辑运算函数LIKE。

语法

```
A LIKE B
```

入参

参数	数据类型
A	VARCHAR
B	VARCHAR

功能描述

LIKE函数用于模糊查询。 `%` 用于定义通配符。

示例1

- 测试数据

int1(INT)	VARCHAR2(VARCHAR)	VARCHAR3(VARCHAR)
90	ss97	97ss
99	ss10	7ho7

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR2 LIKE 'ss%';
```

- 测试结果

aa(int)
90
99

示例2

- 测试数据

int1(INT)	VARCHAR2(VARCHAR)	VARCHAR3(VARCHAR)
90	ss97	97ss
99	ss10	7ho7

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE VARCHAR3 LIKE '%ho%';
```

- 测试结果

aa(int)
99

4.10.4.17. NOT

本文为您介绍如何使用实时计算逻辑运算函数NOT。

语法

```
NOT A
```

入参

参数	数据类型
A	BOOLEAN

功能描述

如果A是 `TRUE` 时，则返回 `FALSE` 。如果A是 `FALSE` 时，则返回 `TRUE` 。

示例

- 测试数据

int2(INT)	int3(INT)
97	65

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE NOT int3=62;
```

- 测试结果

aa(int)
97

4.10.4.18. NOT BETWEEN AND

本文为您介绍如何使用实时计算逻辑运算函数NOT BETWEEN AND。

语法

```
A NOT BETWEEN B AND C
```

入参

参数	数据类型
A	DOUBLE、BIGINT、INT、VARCHAR、DATE、TIMESTAMP、TIME
B	DOUBLE、BIGINT、INT、VARCHAR、DATE、TIMESTAMP、TIME
C	DOUBLE、BIGINT、INT、VARCHAR、DATE、TIMESTAMP、TIME

功能描述

`NOT BETWEEN AND` 操作符用于选取不存在与两个值之间的数据范围内的值。

示例1

- 测试数据

int1(INT)	int2(INT)	int3(INT)
90	97	80
11	10	7

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE int1 NOT BETWEEN int2 AND int3;
```

- 测试结果

aa(int)
11

示例2

- 测试数据

var1(varchar)	var2(varchar)	var3(varchar)
d	a	c

- 测试语句

```
SELECT int1 as aa
FROM T1
WHERE var1 NOT BETWEEN var2 AND var3;
```

- 测试结果

aa(varchar)
d

示例3

- 测试数据

TIMESTAMP1(TIMESTAMP)	TIMESTAMP2(TIMESTAMP)	TIMESTAMP3(TIMESTAMP)
1969-07-20 20:17:30	1969-07-20 20:17:40	1969-07-20 20:17:45

- 测试语句

```
SELECT TIMESTAMP1 as aa
FROM T1
WHERE TIMESTAMP1 NOT BETWEEN TIMESTAMP2 AND TIMESTAMP3;
```

- 测试结果

aa(TIMESTAMP)
1969-07-20 20:17:30

4.10.4.19. IN

本文为您介绍如何使用实时计算逻辑运算函数IN。

语法

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

入参

参数	数据类型
value1	常数
value2	常数

功能描述

用来查找在参数中的记录。

示例

- 测试数据

id(Int)	LastName(Varchar)
1	Adams
2	Bush
3	Carter

- 测试语句

```
SELECT *
FROM T1
WHERE LastName IN ('Adams','Carter');
```

- 测试结果

id(Int)	LastName(Varchar)
1	Adams
3	Carter

4.10.4.20. OR

本文为您介绍如何使用实时计算逻辑运算函数OR。

语法

```
A OR B
```

入参

参数	数据类型
A	BOOLEAN
B	BOOLEAN

功能描述

如果A和B中至少有一个为TRUE，则为TRUE，否则为FALSE。

示例

- 测试数据

int1(INT)	int2(INT)	int3(INT)
255	97	65

- 测试语句

```
SELECT int2 as aa
FROM T1
WHERE int1=255 OR int3=65;
```

- 测试结果

aa(int)
97

4.10.4.21. IS DISTINCT FROM

本文为您介绍如何使用实时计算逻辑运算函数IS DISTINCT FROM。

语法

```
A IS DISTINCT FROM B
```

入参

参数	数据类型
A	任意数据类型
B	任意数据类型

功能描述

- A和B的数据类型、值不完全相同则返回 `TRUE` 。
- A和B的数据类型、值都相同返回 `FALSE` 。
- 将空值视为相同。

示例

- 测试数据

A(int)	B(varchar)
97	97
null	sss
null	null

- 测试语句

```
SELECT
A IS DISTINCT FROM B as 'result'
FROM T1;
```

- 测试结果

result(BOOLEAN)
true
true
false

4.10.4.22. IS NOT DISTINCT FROM

本文为您介绍如何使用实时计算逻辑运算函数IS NOT DISTINCT FROM。

语法

```
A IS NOT DISTINCT FROM B
```

入参

参数	数据类型
A	任意数据类型
B	任意数据类型

功能描述

- A和B的数据类型、值不完全相同则返回 `FALSE` 。
- A和B的数据类型、值都相同返回 `TRUE` 。
- 将空值视为相同。

示例

- 测试数据

A(int)	B(varchar)
97	97
null	sss
null	null

- 测试语句

```
SELECT
A IS NOT DISTINCT FROM B as `result`
FROM T1;
```

- 测试结果

result(BOOLEAN)
false
false
true

4.10.4.23. NOT IN

本文为您介绍如何使用实时计算逻辑运算函数NOT IN。

语法

```
SELECT column_name(s)
FROM table_name
WHERE column_name NOT IN (value1,value2,...)
```

入参

参数	数据类型
value1	常数
value2	常数

功能描述

用来查找在不在参数中的记录。

示例

- 测试数据

id(Int)	LastName(Varchar)
1	Adams
2	Bush
3	Carter

- 测试语句

```
SELECT *
FROM T1
WHERE LastName NOT IN ('Adams','Carter');
```

- 测试结果

id(Int)	LastName(Varchar)
2	Bush

4.10.5. 条件函数

4.10.5.1. CASE WHEN

本文为您介绍如何使用实时计算条件函数CASE WHEN。

语法

```
CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END
```

功能描述

如果表达式a为TRUE，则返回b；如果表达式a为FALSE、c为TRUE，则返回d；如果表达式a和c都为FALSE，则返回e。

注意事项

CASE WHEN返回常量字符串时，会在字符串后面补全空格。例如，当满足else条件时，返回值`ios`后面会多几个空格。

```
case when device_type = 'android'
then 'android'
else 'ios'
end as os
```

解决方法：

- 利用TRIM函数去除空格，该示例中，涉及`os`的字段都改为`TRIM(os)`。
- 利用CAST函数去除空格，将常量字符串转为VARCHAR类型。

示例

- 测试数据

device_type(VARCHAR)
android
ios
win

- 测试语句
  - 利用TRIM函数

```
SELECT
   trim(os), --添加trim。
   CHAR_LENGTH(trim(os)) --添加trim。
from(
   SELECT
     case when device_type = 'android'
     then 'android'
     else 'ios'
end as os
FROM T1
);
```

  - 利用CAST函数

```
SELECT
 os,
CHAR_LENGTH(os)
from
(SELECT
 case when device_type = 'android'
 then cast('android' as varchar) --添加cast。
 else cast('ios' as varchar) --添加cast。
end as os
FROM T1
);
```

- 测试结果

os(VARCHAR)	length(INT)
android	7
ios	3
ios	3

4.10.5.2. COALESCE

本文为您介绍如何使用实时计算条件函数COALESCE。

语法

```
COALESCE(A,B,...)
```

入参

参数	数据类型
A	任意数据类型
B	任意数据类型

> **说明** 所有这些值类型必须相同或为null，否则会引发异常。

功能描述

返回列表中第一个非null的值，返回值类型和参数类型相同。如果列表中所有的值都是null，则返回null。

> **说明** 至少要有一个参数，否则引发异常。

示例

- 测试数据

var1(VARCHAR)	var2(VARCHAR)
null	30

- 测试语句

```
SELECT COALESCE(var1,var2) as aa
FROM T1;
```

- 测试结果

aa(VARCHAR)
30

4.10.5.3. IF

本文为您介绍如何使用实时计算条件函数IF。

语法

```
T if (BOOLEAN testCondition, T valueTrue, T valueFalseOrNull)
```

> **说明** T代表任意类型的返回值。

入参

参数	数据类型
testCondition	BOOLEAN
valueTrue	可以为任意类型，但valueTrue和valueFalseOrNull类型需保持一致。
valueFalseOrNull	可以为任意类型，但valueTrue和valueFalseOrNull类型需保持一致。

功能描述

以testCondition的布尔值为判断标准，返回对应的参数值：

- true：返回第2个参数valueTrue。
- false：返回第3个参数valueFalseOrNull。

> **说明**
> - 返回值类型为第2和第3参数的数据类型。
> - 如果testCondition为null时，则判定结果为false。
> - 如果其它参数为null时，则按照正常语义运行运算。

示例

- 测试数据

int1(INT)	int2(INT)	str1(VARCHAR)	str2(VARCHAR)
1	2	Jack	Harry
1	2	Jack	null

int1(INT)	int2(INT)	str1(VARCHAR)	str2(VARCHAR)
1	2	null	Harry

- 测试语句

```
SELECT IF(int1 < int2,str1, str2) as int1
FROM T1;
```

- 测试结果

int1(VARCHAR)
Jack
Jack
null

4.10.5.4. IS_ALPHA

本文为您介绍如何使用实时计算条件函数IS_ALPHA。

语法

```
BOOLEAN IS_ALPHA(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

如果str中只包含字母，则返回true，否则返回false。

示例

- 测试数据

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- 测试语句

```
SELECT IS_ALPHA(e) as boo1,IS_ALPHA(f) as boo2,IS_ALPHA(g) as boo3
FROM T1;
```

- 测试结果

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
false	true	false

4.10.5.5. IS_DECIMAL

本文为您介绍如何使用实时计算条件函数IS_DECIMAL。

语法

```
BOOLEAN IS_DECIMAL(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

str字符串如果可以转换为十进制数值，则返回TRUE，否则返回FALSE。

示例

- 测试数据

a(VARCHAR)	b(VARCHAR)	c(VARCHAR)	d(VARCHAR)	e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
1	123	2	11.4445	3	asd	null

- 测试语句

```
SELECT
IS_DECIMAL(a) as boo1,
IS_DECIMAL(b) as boo2,
IS_DECIMAL(c) as boo3,
IS_DECIMAL(d) as boo4,
IS_DECIMAL(e) as boo5,
IS_DECIMAL(f) as boo6,
IS_DECIMAL(g) as boo7
FROM T1;
```

- 测试结果

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)	boo4(BOOLEAN)	boo5(BOOLEAN)	boo6(BOOLEAN)	boo7(BOOLEAN)
true	true	true	true	true	false	false

4.10.5.6. IS_DIGIT

本文为您介绍如何使用实时计算条件函数IS_DIGIT。

语法

```
BOOLEAN IS_DIGIT(VARCHAR str)
```

入参

参数	数据类型
str	VARCHAR

功能描述

如果str中只包含数字，则返回true，否则返回false。返回值为BOOLEAN类型。

示例

- 测试数据

e(VARCHAR)	f(VARCHAR)	g(VARCHAR)
3	asd	null

- 测试语句

```
SELECT
IS_DIGIT(e) as boo1,
IS_DIGIT(f) as boo2,
IS_DIGIT(g) as boo3
FROM T1;
```

- 测试结果

boo1(BOOLEAN)	boo2(BOOLEAN)	boo3(BOOLEAN)
true	false	false

4.10.5.7. NULLIF

本文为您介绍如何使用实时计算条件函数NULLIF。

语法

```
NULLIF(A,B)
```

入参

参数	数据类型
A	INT
B	INT

功能描述

如果两个参数的值相同，则返回null，如果值不同则返回第一个参数的值。

示例

- 测试数据

var1(INT)	var2(INT)
30	30

- 测试语句

```
SELECT NULLIF(var1,var2) as aa
FROM T1;
```

- 测试结果

aa(INT)
null

4.10.6. 表值函数

4.10.6.1. GENERATE_SERIES

本文为您介绍如何使用实时计算表值函数GENERATE_SERIES。

语法

```
GENERATE_SERIES(INT from, INT to)
```

入参

参数	数据类型
from	INT类型，指定下界，包含下界值。
to	INT类型，指定上界，不包含上界值。

功能描述

生成一串连续的数值，分别为from、from+1、 from+2 ... to-1。

示例

- 测试数据

s(INT)	e(INT)
1	3
-2	1

- 测试语句

```
SELECT s, e, v
FROM T1, lateral table(GENERATE_SERIES(s, e))
as T(v);
```

- 测试结果

s(INT)	e(INT)	v(INT)
1	3	1
1	3	2
-2	1	-2
-2	1	-1
-2	1	-0

4.10.6.2. JSON_TUPLE

本文为您介绍如何使用实时计算Flink版表值函数JSON_TUPLE。

> 注意　仅Blink支持JSON_TUPLE，Flink不支持JSON_TUPLE。

语法

```
JSON_TUPLE(str, path1, path2 ..., pathN)
```

入参

参数	数据类型	说明
str	VARCHAR	JSON字符串
path1 ~ pathN	VARCHAR	表示路径的字符串，前面不需要`$`。

功能描述

从JSON字符串中取出各路径字符串所表示的值。

示例

- 测试数据

d(VARCHAR)	s(VARCHAR)
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	qwe3
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	qwe2

- 测试语句

```
SELECT d, v
FROM T1, lateral table(JSON_TUPLE(d, 'qwe', s))
AS T(v);
```

- 测试结果

d(VARCHAR)	v(VARCHAR)
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	asd
{"qwe":"asd","qwe2":"asd2","qwe3":"asd3"}	asd3
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	asd4
{"qwe":"asd4","qwe2":"asd5","qwe3":"asd3"}	asd5

4.10.6.3. STRING_SPLIT

本文为您介绍如何使用实时计算表值函数STRING_SPLIT。

语法

```
string_split(string, separator)
```

入参

参数	数据类型	说明
string	VARCHAR	目标字符串
separator	VARCHAR	指定的分隔符 **说明** 分隔符separator暂不支持多字符串形式，只支持单个字符串形式。

功能描述

根据指定的分隔符将目标字符串拆分为子字符串行，返回子字符串的单列的表。需要注意以下几点：

- 如果目标字符串为NULL，则STRING_SPLIT表值函数返回一个空行。
- 如果目标字符串包含两个或多个连续出现的分隔符时，则返回长度为零的空子字符串。
- 如果目标字符串未包含指定分隔符，则只返回目标字符串。

示例

- 测试数据 T1

d(varchar)	s(varchar)
abc-bcd	-
hhh	-

- 测试语句

```
select d,v
from T1,
lateral table(string_split(d, s)) as T(v);
```

- 测试结果

d(varchar)	v(varchar)
abc-bcd	abc
abc-bcd	bcd
hhh	hhh

4.10.6.4. MULTI_KEYVALUE

本文为您介绍如何使用实时计算表值函数MULTI_KEYVALUE。

> **说明** 仅支持实时计算2.2.2及以上版本。

语法

```
MULTI_KEYVALUE(VARCHAR str, VARCHAR split1, VARCHAR split2, VARCHAR key_name1, VARCHAR key_name2, ...)
```

入参

参数	数据类型	说明
str	VARCHAR	字符串中的key-value（kv）对。

参数	数据类型	说明
split1	VARCHAR	kv对的分隔符。当split1为null时，表示按照whitespace作为kv对的分割符。当split1的长度>1时，split1仅表示分隔符的集合，每个字符都表示一个有效的分隔符。
split2	VARCHAR	kv的分隔符。当split2为null时，表示按照whitespace作为kv的分割符。当split2的长度>1时，split2仅表示分隔符的集合，每个字符都表示一个有效的分隔符。
key_name1, key_name2, ...	VARCHAR	需要获取value的key值列表。

功能描述

解析str字符串中的key-value对，匹配有split1和split2的key-value对，并返回参数列表里key_name1，key_name2等对应的value值列表。key_name值不存在时，对应的value值是null。

示例

- 测试数据

str(VARCHAR)	split1(VARCHAR)	split2(VARCHAR)	key1(VARCHAR)	key2(VARCHAR)
k1=v1;k2=v2	;	=	k1	k2
null	;	=	k1	k2
k1:v1;k2:v2	;	:	k1	k3
k1:v1;k2:v2	;	=	k1	k2
k1:v1;k2:v2	,	:	k1	k2
k1:v1;k2=v2	;	:	k1	k2
k1:v1abck2:v2	cab	:	k1	k2
k1:v1;k2=v2	;	:=	k1	k2
k1:v1 k2:v2	null	:	k1	k2
k1 v1;k2 v2	;	null	k1	k2

- 测试语句

```
SELECT c1, c2
FROM T1, lateral table(MULTI_KEYVALUE(str, split1, split2, key1, key2))
as T(c1, c2);
```

- 测试结果

c1(VARCHAR)	c2(VARCHAR)
v1	v2
null	null
v1	null
null	null
null	null
v1	null
v1	v2
v1	v2
v1	v2
v1	v2

4.10.7. 类型转换函数

4.10.7.1. CAST

本文为您介绍如何使用实时计算类型转换函数CAST。

语法

```
CAST(A AS type)
```

入参

参数	数据类型
A	请参见类型转换。

功能描述

将A值转换为给定类型。如果转换后的类型和目标表字段类型不匹配时，会出现类似`Insert into: Query result and target table 'test_result' field type(s) not match.`的报错。

示例

- 测试数据

var1(VARCHAR)	var2(INT)
1000	30

- 测试语句

```
SELECT CAST(var1 AS INT) as aa
FROM T1;
```

- 测试结果

aa(INT)
1000

4.10.8. 聚合函数

4.10.8.1. AVG

本文为您介绍如何使用实时计算聚合函数AVG。Flink SQL中使用AVG函数返回指定表达式中所有值的平均值。

语法

```
AVG(A)
```

入参

参数	数据类型
A	TINYINT、SMALLINT、INT、BIGINT、FLOAT、DECIMAL和DOUBLE。

功能描述

返回指定表达式中所有值的平均值。

> **说明** 返回值默认为DOUBLE类型，如果您的结果表字段为非DOUBLE类型，您需要使用CAST进行转化。

示例

- 测试数据

var1(INT)	var2(INT)
4	30
6	30

- 测试语句

```
SELECT AVG(var1) as aa
FROM T1;
```

- 测试结果

aa(INT)
5

4.10.8.2. CONCAT_AGG

本文为您介绍如何使用实时计算聚合函数CONCAT_AGG。Flink SQL中使用CONCAT_AGG函数将对应字段的所有字符串连接成新的字符串。

语法

```
CONCAT_AGG([linedelimiter,] value )
```

入参

参数	数据类型
linedelimiter	目前只支持字符串常量。可选。

功能描述

连接对应字段的字符串，默认连接符 `\n` ，连接完成后新生成的字符串。返回值VARCHAR类型。

示例

- 测试数据

b(VARCHAR)	c(VARCHAR)
Hi	milk
Hi	milk
Hi	milk
Hi	milk
Hi	milk
Hi	milk
Hello	cola
Hello	cola
Happy	suda
Happy	suda

- 测试语句

```
SELECT
      b,
      concat_agg(c) as var1,
      concat_agg('-', c) as var2
FROM MyTable
GROUP BY b;
```

- 测试结果

b (VARCHAR)	var1(VARCHAR)	var2(VARCHAR)
Hi	milk milk milk milk milk milk	milk-milk-milk-milk-milk-milk
Hello	cola cola	cola-cola
Happy	suda suda	suda-suda

4.10.8.3. COUNT

本文为您介绍如何使用实时计算聚合函数COUNT。Flink SQL中使用COUNT函数返回输入列的数量。

语法

```
COUNT(A)
```

入参

参数	数据类型
A	• 支持TINYINT、SMALLINT、INT、BIGINT、FLOAT、DECIMAL、DOUBLE、BOOLEAN和VARCHAR类型。 • 不支持DATE、TIME、TIMESTAMP和VARBINARY类型。

示例

- 测试数据

var1(VARCHAR)
1000
100
10
1

- 测试语句

```
SELECT COUNT(var1) as aa
FROM T1;
```

- 测试结果

aa(BIGINT)
4

4.10.8.4. FIRST_VALUE

本文为您介绍如何使用实时计算聚合函数FIRST_VALUE。Flink SQL中使用FIRST_VALUE函数返回数据流的第1条非null数据。

语法

```
T FIRST_VALUE( T value )
T FIRST_VALUE( T value, BIGINT order )
```

入参

参数	数据类型
value	任意参数类型，但输入参数只能为同一种类型。
order	BIGINT

功能描述

获取数据流的第1条非null数据。根据order判定FIRST_VALUE所在的行，取order值最小的记录作为FIRST_VALUE。

示例1

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	“Hello”

a(BIGINT)	b(INT)	c(VARCHAR)
2L	2	“Hello”
3L	3	“Hello”
4L	4	“Hello”
5L	5	“Hello”
6L	6	“Hello”
7L	7	“Hello World”
8L	8	“Hello World”
20L	20	“Hello World”

- 测试语句

```
SELECT c,
 FIRST_VALUE(b)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```

- 测试结果

c(VARCHAR)	var1(BIGINT)
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	1
“Hello World”	7
“Hello World”	7
“Hello World”	7

示例2

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)	order (BIGINT)
1L	1	“Hello”	7
2L	2	“Hello”	7
3L	3	“Hello”	7
4L	4	“Hello”	7
5L	5	“Hello”	6
6L	6	“Hello”	4
7L	7	“Hello World”	3
8L	8	“Hello World”	2
20L	20	“Hello World”	1

- 测试语句

```
SELECT c,
 FIRST_VALUE(b,order)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```

- 测试结果

c(VARCHAR)	var1(INT)
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	1
“Hello”	5
“Hello”	6
“Hello World”	7
“Hello World”	8
“Hello World”	20

4.10.8.5. LAST_VALUE

本文为您介绍如何使用实时计算聚合函数LAST_VALUE。Flink SQL中使用LAST_VALUE函数返回指定数据流的最后1条非NULL数据。

语法

```
T LAST_VALUE(T value)
T LAST_VALUE(T value, BIGINT order)
```

入参

参数	数据类型
value	任意参数类型 说明 所有输入参数需要为相同的数据类型。
order	BIGINT

功能描述

获取数据流的最后1条非NULL数据。根据ORDER判定LAST_VALUE所在的行，取ORDER值最大的记录作为LAST_VALUE。

示例1

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)
1L	1	“Hello”
2L	2	“Hello”
3L	3	“Hello”
4L	4	“Hello”
5L	5	“Hello”
6L	6	“Hello”
7L	7	“Hello World”
8L	8	“Hello World”
20L	20	“Hello World”

- 测试语句

```
SELECT c,
 LAST_VALUE(b)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```

- 测试结果

c(VARCHAR)	var1(INT)
“Hello”	1
“Hello”	2
“Hello”	3
“Hello”	4
“Hello”	5
“Hello”	6
“Hello World”	7
“Hello World”	8
“Hello World”	20

示例2

- 测试数据

a(BIGINT)	b(INT)	c(VARCHAR)	order (BIGINT)
1L	1	“Hello”	5
2L	2	“Hello”	5
3L	3	“Hello”	6
4L	4	“Hello”	5
5L	5	“Hello”	6
6L	6	“Hello”	5
7L	7	“Hello World”	4
8L	8	“Hello World”	8
20L	20	“Hello World”	9

- 测试语句

```
SELECT c,
 LAST_VALUE(b,order)
OVER (
PARTITION BY c
ORDER BY PROCTIME() RANGE UNBOUNDED PRECEDING
) AS var1
FROM T1;
```

- 测试结果

c(VARCHAR)	var1(INT)
“Hello”	1
“Hello”	1
“Hello”	3
“Hello”	3
“Hello”	3
“Hello”	3
“Hello World”	7
“Hello World”	8
“Hello World”	20

4.10.8.6. MAX

本文为您介绍如何使用实时计算聚合函数MAX。Flink SQL中使用MAX函数返回所有输入值的最大值。

语法

```
MAX(A)
```

入参

参数	数据类型
A	• 支持TINYINT，SMALLINT、INT、BIGINT、FLOAT、DECIMAL、DOUBLE、BOOLEAN和VARCHAR类型。 • 不支持DATE、TIME、TIMESTAMP和VARBINARY类型。

功能描述

返回所有输入值的最大值。

示例

- 测试数据

var1(INT)
4
8

- 测试语句

```
SELECT MAX(var1) as aa
FROM T1;
```

- 测试结果

aa(INT)
8

4.10.8.7. MIN

本文为您介绍如何使用实时计算聚合函数MIN。Flink SQL中使用MIN函数返回所有输入值的最小值。

语法

```
MIN(A)
```

入参

参数	数据类型
A	• 支持TINYINT、SMALLINT、INT、BIGINT、FLOAT、DECIMAL、DOUBLE、BOOLEAN和VARCHAR类型。 • 不支持DATE、TIME、TIMESTAMP和VARBINARY类型。

功能描述

返回所有输入值的最小值。

示例

- 测试数据

var1(INT)
4
8

- 测试语句

```
SELECT MIN(var1) as aa
FROM T1;
```

- 测试结果

aa(INT)
4

4.10.8.8. SUM

本文为您介绍如何使用实时计算聚合函数SUM。Flink SQL中使用SUM函数来返回所有输入值的数值之和。

语法

```
SUM(A)
```

入参

参数	数据类型
A	TINYINT、SMALLINT、INT、BIGINT、FLOAT、DECIMAL和DOUBLE。

功能描述

返回所有输入值的数值之和。

示例

- 测试数据

var1(INT)
4
4

- 测试语句

```
SELECT sum(var1) as aa
FROM T1;
```

- 测试结果

aa(INT)
8

4.10.8.9. VAR_POP

Flink SQL中使用VAR_POP函数返回指定表达式中所有值的总体统计方差。

语法

```
T VAR_POP(T value)
```

入参

参数	数据类型
value	数值型，可以为BIGINT和DOUBLE等类型。

功能描述

返回所有输入值的方差。

示例

- 测试数据

a(BIGINT)	c(VARCHAR)
2900	Hi
2500	Hi
2600	Hi
3100	Hello
11000	Hello

- 测试语句

```
SELECT
VAR_POP(a) as `result`,
c
FROM MyTable
GROUP BY c;
```

- 测试结果

result(BIGINT)	c
28889	Hi
15602500	Hello

4.10.8.10. STDDEV_POP

本文为您介绍如何使用实时计算聚合函数STDDEV_POP。Flink SQL中使用STDDEV_POP函数来返回数值的总体标准差。

语法

```
T STDDEV_POP(T value)
```

入参

参数	数据类型
value	BIGINT或DOUBLE

功能描述

返回数值的总体标准差。

示例

- 测试数据

a(DOUBLE)	c(VARCHAR)
0	Hi
1	Hi
2	Hi
3	Hi
4	Hi
5	Hi
6	Hi
7	Hi
8	Hi
9	Hi

- 测试语句

```
SELECT c, STDDEV_POP(a) as dou1
FROM MyTable
GROUP BY c;
```

- 测试结果

c(VARCHAR)	dou1(DOUBLE)
Hi	2.8722813232690143

4.10.9. 其他函数

4.10.9.1. UUID

本文为您介绍如何使用实时计算UUID。Flink SQL中使用UUID函数返回通用唯一标识字符。

语法

```
VARCHAR UUID()
```

功能描述

返回通用唯一标识字符。

示例

- 测试语句

```
SELECT uuid() as `result`
FROM T1;
```

- 测试结果

result(VARCHAR)
a364e414-e68b-4e5c-9166-65b3a153e257

4.10.9.2. DISTINCT

DISTINCT用于SELECT语句中，可以对查询结果进行去重。

DISTINCT语法

```
SELECT DISTINCT expressions
FROM tables;
```

- `DISTINCT` 必须放到开始位置。和其他函数一起使用时，`DISTINCT` 也必须放到开始位置，例如，`concat_agg(DISTINCT ',' ,device_id)` 。
- `expressions` 是一个或多个expression，可以是具体的column，也可以是function等任何合法表达式。

DISTINCT示例

- SQL语句

  Flink SQL中DISTINCT的示例如下所示。

```
CREATE TABLE distinct_tab_source(
 FirstName VARCHAR,
 LastName VARCHAR
)WITH(
 type='random'
) ;
CREATE TABLE distinct_tab_sink(
 FirstName  VARCHAR,
 LastName VARCHAR
)WITH(
 type = 'print'
) ;
INSERT INTO distinct_tab_sink
SELECT DISTINCT FirstName, LastName --按照FirstName和LastName两个列进行去重。
FROM distinct_tab_source;
```

- 测试数据

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN
SUN	SHENGRAN

- 查询结果

> **说明**
>
> - 测试数据有4条记录。`DISTINCT FirstName, LastName` 将SUN和SHENGRAN去重后，输出3条结果记录。
> - `SUNS,HENGRAN` 和 `SUN,SHENGRAN` 两条记录并没有被去重，说明 `DISTINCT FirstName, LastName` 是对两个字段分别处理的，而不是Concat到一起再进行去重。

DISTINCT的等效写法

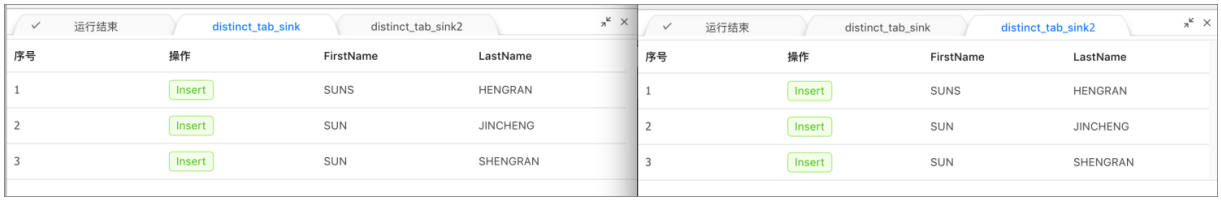
在SQL中利用 `GROUP BY` 语句也可以到达和 `DISTINCT` 类似的去重效果。 `GROUP BY` 语法如下。

```
SELECT expressions
FROM tables
GROUP BY expressions
;
```

通过多路输出的方式编写的和DISTINCT示例等效的SQL，示例如下。

```
CREATE TABLE distinct_tab_source(
 FirstName VARCHAR,
 LastName VARCHAR
)WITH(
 type='random'
);
CREATE TABLE distinct_tab_sink(
 FirstName VARCHAR,
 LastName VARCHAR
)WITH(
 type = 'print'
);
CREATE TABLE distinct_tab_sink2(
 FirstName VARCHAR,
 LastName VARCHAR
)WITH(
 type = 'print'
);
INSERT INTO distinct_tab_sink
    SELECT DISTINCT FirstName, LastName --按照FirstName和LastName两个列进行去重。
        FROM distinct_tab_source;
INSERT INTO distinct_tab_sink2
    SELECT FirstName, LastName
        FROM distinct_tab_source
         GROUP BY FirstName, LastName; --按照FirstName和LastName两个列进行去重。
```

和DISTINCT示例使用相同的测试数据，最终的结果数据相同，说明两种方式的语义一致。



DISTINCT在聚合函数COUNT中的作用

`DISTINCT` 能使聚合函数 `COUNT` 统计去重后的计数。

```
COUNT(DISTINCT expression)
```

> **注意** `expression` 目前只支持一个表达式。

COUNT DISTINCT语法示例

- SQL语法

```
CREATE TABLE distinct_tab_source(
 FirstName VARCHAR,
 LastName VARCHAR
)WITH(
 type='random'
);
CREATE TABLE distinct_tab_sink(
 cnt BIGINT,
 distinct_cnt BIGINT
)WITH(
 type = 'print'
) ;
INSERT INTO distinct_tab_sink
    SELECT
      COUNT(FirstName),  --不去重。
      COUNT(DISTINCT FirstName)  --按照FirstName去重。
    FROM distinct_tab_source;
```

- 测试数据

FirstName	LastName
SUNS	HENGRAN
SUN	JINCHENG
SUN	SHENGRAN
SUN	SHENGRAN

- 测试结果

运行结束 | distinct_tab_sink

序号	操作	cnt	distinct_cnt
1	Insert	1	1
2	Insert	2	2
3	Insert	3	2
4	Insert	4	2

4.11. 自定义函数（UDX）

4.11.1. 概述

本文为您介绍如何搭建实时计算Flink版自定义函数的环境并使用自定义函数。

> **注意**
>
> - 仅独享模式支持自定义函数。
> - Blink在开源Flink SQL的基础上对性能进行了增强，Blink是阿里云实时计算版本的Flink。UDX函数仅适用于Blink，对开源Flink暂不适用。
> - 为了避免JAR依赖冲突，您需要注意以下几点：
>   - 开发页面选择的Blink版本，请和Pom依赖Blink版本保持一致。
>   - Blink相关依赖，scope请使用provided，即 `<scope>provided</scope>`
>   - 其他第三方依赖请采用Shade方式打包，Shade打包详情参见Apache Maven Shade Plugin。

UDX分类

实时计算Flink版支持以下3类自定义函数。

UDX分类	描述
UDF（User Defined Scalar Function）	用户自定义标量值函数，其输入与输出是一对一的关系，即读入一行数据，写出一条输出值。
UDAF（User Defined Aggregation Function）	自定义聚合函数，其输入与输出是多对一的关系，即将多条输入记录聚合成一条输出值。可以与SQL中的GROUP BY语句一起使用。具体语法请参见聚合函数。
UDTF（User Defined Table-valued Function）	自定义表值函数，调用一次函数输出多行或多列数据。

UDX示例

实时计算Flink版为您提供UDX示例，便于您快速开发业务。实时计算Flink版UDX示例中包含UDF、UDAF和UDTF的实现，示例如下。

> **说明**
>
> - 示例中已为您配置对应版本的开发环境，您无需进行环境搭建。
> - 示例为Maven项目，您可以使用IntelliJ IDEA进行开发。开发方法请参见开发。

- 实时计算Flink版3.0版本

  Blink_UDX_3x
- 实时计算Flink版2.0版本

  Blink_UDX_2x
- 实时计算Flink版1.0版本

  Blink_UDX_1x

环境搭建

以上示例主要使用的依赖JAR包如下，如果您需要单独使用，可以自行下载。

- 基于实时计算Flink版3.2.1以下版本
  - flink-streaming-java_2.11
  - flink-table_2.11
  - flink-core-blink-2.2.4
- 基于实时计算Flink版3.2.1及以上版本

  请根据需求自行添加开源版本所支持的POM依赖包。下载查看完整依赖包示例。

  > **说明** 如果您需要依赖Snapshot版本，可以自行添加Snapshot版本所支持的POM依赖包。

注册使用

1. 登录实时计算控制台。
2. 在顶部菜单中，单击**开发**。
3. 在左侧的导航栏中，单击**资源引用**。
4. 在**资源引用**页签的右上角，单击**新建资源**。
5. 在**上传资源**页面，输入资源配置信息。

参数名称	说明
上传方式	实时计算控制台上仅支持本地上传。 **说明** 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。
资源选择	单击**选择资源**，选择需要引用的资源。
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

6. 在**资源引用**页签中，将鼠标悬停在对应作业的右侧的**更多**上。
7. 在下拉列表中，选择**引用**。
8. 在作业的编辑窗口的顶部，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

参数与返回值类型

实时计算Flink版支持定义Java UDX时，使用Java类型作为参数和返回值。下面为实时计算Flink版类型和Java类型的映射关系。

实时计算Flink版数据类型	Java类型
TINYINT	java.lang.Byte

实时计算Flink版数据类型	Java类型
SMALLINT	java.lang.Short
INT	java.lang.Integer
BIGINT	java.lang.Long
FLOAT	java.lang.Float
DOUBLE	java.lang.Double
DECIMAL	java.math.BigDecimal
BOOLEAN	java.lang.Boolean
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
CHAR	java.lang.Character
STRING	java.lang.String
VARBINARY	java.lang.byte[]
ARRAY	暂不支持
MAP	暂不支持

获取自定义函数参数

自定义函数中提供了可选的 `open(FunctionContext context)` 方法， `FunctionContext` 具备参数传递功能，自定义配置项可以通过此对象来传递。

假设，您需要在作业中添加如下两个参数。

```
testKey1=lincoln
test.key2=todd
```

以UDTF为例，在Open方法中通过 `context.getJobParameter` 即可获取，示例如下。

```
public void open(FunctionContext context) throws Exception {
      String key1 = context.getJobParameter("testKey1", "empty");
      String key2 = context.getJobParameter("test.key2", "empty");
      System.err.println(String.format("end open: key1:%s, key2:%s", key1, key2));
}
```

> **说明** 具体的作业参数请参见作业参数。

4.11.2. 自定义标量函数（UDF）

本文为您介绍如何为实时计算Flink版自定义标量函数（UDF）搭建开发环境、编写业务代码及上线。

> **注意** 阿里云实时计算Flink版共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

定义

自定义标量函数（UDF）将0个、1个或多个标量值映射到一个新的标量值。

搭建环境

搭建开发环境，请参见环境搭建。

业务代码

UDF需要在ScalarFunction类中实现 `eval` 方法。 `open` 方法和 `close` 方法可选。

> **注意** UDF默认对于相同的输入会有相同的输出。如果UDF不能保证相同的输出，例如，在UDF中调用外部服务，相同的输入值可能返回不同的结果，建议您使用 `override isDeterministic()` 方法，返回 `False` 。否则在某些条件下，输出结果不符合预期。例如，UDF算子前移。

以Java为例，示例代码如下。

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class StringLengthUdf extends ScalarFunction {
    // 可选，open方法可以不写。
    // 如果编写open方法需要声明'import org.apache.flink.table.functions.FunctionContext;'。
    @Override
    public void open(FunctionContext context) {
        }
    public long eval(String a) {
        return a == null ? 0 : a.length();
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    //可选，close方法可以不写。
    @Override
    public void close() {
        }
}
```

编写SQL语句

在指定的Class中编写SQL语句，自定义函数中SQL语句示例如下。

```
-- udf str.length()
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
create table sls_stream(
    a int,
    b int,
    c varchar
) with (
    type='sls',
    endPoint='<yourEndpoint>',
    accessKeyId='<yourAccessId>',
    accessKeySecret='<yourAccessSecret>',
    startTime = '2017-07-04 00:00:00',
    project='<yourProjectName>',
    logStore='<yourLogStoreName>',
    consumerGroup='consumerGroupTest1'
);
create table rds_output(
    id int,
    len bigint,
    content VARCHAR
) with (
    type='rds',
    url='yourDatabaseURL',
    tableName='<yourDatabaseTableName>',
    userName='<yourDatabaseUserName>',
    password='<yourDatabasePassword>'
);
insert into rds_output
select
    a,
    stringLengthUdf(c),
    c as content
from sls_stream;
```

注册使用

1. 登录实时计算控制台。
2. 在顶部菜单中，单击**开发**。
3. 在左侧的导航栏中，单击**资源引用**。
4. 在**资源引用**页签的右上角，单击**新建资源**。
5. 在**上传资源**页面，输入资源配置信息。

参数名称	说明

参数名称	说明
上传方式	实时计算控制台上仅支持本地上传。 **说明** 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。
资源选择	单击**选择资源**，选择需要引用的资源。
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

6. 在**资源引用**页签中，将鼠标悬停在对应作业的右侧的**更多**上。
7. 在下拉列表中，选择**引用**。
8. 在作业的编辑窗口的顶部，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

上线和启动

在作业**开发**页面单击**上线**后，在**运维**页面单击**启动**，即可完成自定义函数的上线。

常见问题

Q：为什么实现了一个随机数生成函数，在运行时产生的值却一样？

A：如果自定义函数是无参的，并且没有声明是非确定性函数，编译期间可能会被优化成一个常量值。如果需要让函数变成非确定性函数，不被优化为常量，建议您使用 `override isDeterministic()` 方法，返回 `False` 。

4.11.3. 自定义聚合函数（UDAF）

本文为您介绍如何为实时计算Flink版自定义聚合函数（UDAF）搭建开发环境、编写业务代码及上线。

注意 阿里云实时计算Flink版共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

定义

自定义聚合函数（UDAF）可以将多条记录聚合成1条记录。

UDAF抽象类内部方法

> **说明** 虽然UDAF可以使用Java或Scala实现，但是建议您使用Java，因为Scala的数据类型有时会造成不必要的性能损失。

AggregateFunction的核心接口方法，如下所示：

- createAccumulator和getValue方法

```
/*
* @param <T> UDAF的输出结果的类型。
* @param <ACC> UDAF的accumulator的类型。accumulator是UDAF计算中用来存放计算中间结果的数据类型。您可以需要根据需要自行设计每个UDAF的accumulator。
*/
public abstract class AggregateFunction<T, ACC> extends UserDefinedFunction {
/*
* 初始化AggregateFunction的accumulator。
* 系统在进行第一个aggregate计算之前，调用一次此方法。
*/
public ACC createAccumulator();
/*
* 系统在每次aggregate计算完成后，调用此方法。
*/
public T getValue(ACC accumulator);
}
```

> **说明**
> - createAccumulator和getValue可以定义在AggregateFunction抽象类内。
> - UDAF必须包含1个accumulate方法。

- accumulate方法

```
public void accumulate(ACC accumulator, ...[用户指定的输入参数]...);
```

> **说明**
> - 您需要实现一个accumulate方法，来描述如何计算输入的数据，并更新数据到accumulator中。
> - accumulate方法的第一个参数必须是使用AggregateFunction的ACC类型的accumulator。在系统运行过程中，runtime代码会把accumulator的历史状态和您指定的上游数据（支持任意数量，任意类型的数据）作为参数，一起传递给accumulate方法。

- retract和merge方法

  createAccumulator、getValue和accumulate 3个方法一起使用，可以设计出一个最基本的UDAF。但是实时计算Flink版一些特殊的场景需要您提供retract和merge两个方法才能完成。

  通常，计算都是对无限流的一个提前的观测值（early firing）。既然有early firing，就会有对发出的结果的修改，这个操作叫作撤回（retract）。SQL翻译优化器会帮助您自动判断哪些情况下会产生撤回的数据，哪些操作需要处理带有撤回标记的数据。但是您需要实现一个retract方法来处理撤回的数据。

```
public void retract(ACC accumulator, ...[您指定的输入参数]...);
```

> **说明**
> - retract方法是accumulate方法的逆操作。例如，实现Count功能的UDAF，在使用accumulate方法时，每来一条数据要加1；在使用retract方法时，就要减1。
> - 类似于accumulate方法，retract方法的第1个参数必须使用AggregateFunction的ACC类型的accumulator。在系统运行过程中，runtime代码会把accumulator的历史状态，和您指定的上游数据（任意数量，任意类型的数据）一起发送给retract计算。

在实时计算Flink版中一些场景需要使用merge方法，例如session window。由于实时计算Flink版具有out of order的特性，后输入的数据有可能位于2个原本分开的session中间，这样就把2个session合为1个session。此时，需要使用merge方法把多个accumulator合为1个accumulator。

```
public void merge(ACC accumulator, Iterable<ACC> its);
```

> **说明**
> - merge方法的第1个参数，必须是使用AggregateFunction的ACC类型的accumulator，而且第1个accumulator是merge方法完成之后，状态所存放的地方。
> - merge方法的第2个参数是1个ACC类型的accumulator遍历迭代器，里面有可能存在1个或多个accumulator。

搭建开发环境

搭建开发环境请参见环境搭建。

编写业务逻辑代码

以Java为例，举例代码如下。

```
import org.apache.flink.table.functions.AggregateFunction;
public class CountUdaf extends AggregateFunction<Long, CountUdaf.CountAccum> {
    //定义存放count UDAF状态的accumulator的数据的结构。
    public static class CountAccum {
        public long total;
    }
    //初始化count UDAF的accumulator。
    public CountAccum createAccumulator() {
        CountAccum acc = new CountAccum();
        acc.total = 0;
        return acc;
    }
    //getValue提供了如何通过存放状态的accumulator计算count UDAF的结果的方法。
    public Long getValue(CountAccum accumulator) {
        return accumulator.total;
    }
    //accumulate提供了如何根据输入的数据更新count UDAF存放状态的accumulator。
    public void accumulate(CountAccum accumulator, Object iValue) {
        accumulator.total++;
    }
    public void merge(CountAccum accumulator, Iterable<CountAccum> its) {
         for (CountAccum other : its) {
            accumulator.total += other.total;
         }
    }
}
```

> **说明** AggregateFunction的子类支持open和close方法作为可选方法，请参见自定义标量函数（UDF）或自定义表值函数（UDTF）的写法。

注册使用

1. 登录实时计算控制台。
2. 在顶部菜单中，单击**开发**。
3. 在左侧的导航栏中，单击**资源引用**。
4. 在**资源引用**页签的左上角，单击**新建资源**。
5. 在**上传资源**页面，输入资源配置信息。

参数名称	说明
上传方式	实时计算控制台上仅支持本地上传。 **说明** 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。
资源选择	单击**选择资源**，选择需要引用的资源。

参数名称	说明
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

6. 在**资源列表**中，将鼠标悬停在目标资源右侧的**更多**上。
7. 在下拉列表中，单击**引用**。
8. 在作业编辑窗口，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

上线和启动

自定义聚合函数（UDAF）的上线和启动步骤，请参见上线和启动。

示例

```
-- UDAF计算count
CREATE FUNCTION countUdaf AS 'com.hjc.test.blink.sql.udx.CountUdaf';
create table sls_stream(
   a int,
   b bigint,
   c varchar
) with (
   type='sls',
   endPoint='yourEndpoint',
   accessKeyId='yourAccessId',
   accessKeySecret='yourAccessSecret',
   startTime='2017-07-04 00:00:00',
   project='<yourPorjectName>',
   logStore='stream-test2',
   consumerGroup='consumerGroupTest3'
);
create table rds_output(
   len1 bigint,
   len2 bigint
) with (
   type='rds',
   url='yourDatabaseURL',
   tableName='<yourDatabaseTableName>',
   userName='<yourDatabaseUserName>',
   password='<yourDatabasePassword>'
);
insert into rds_output
select
   count(a),
   countUdaf(a)
from sls_stream;
```

4.11.4. 自定义表值函数（UDTF）

本文为您介绍如何为实时计算Flink版自定义表值函数（UDTF）搭建开发环境、编写业务代码以及上线。

> **说明** 阿里云实时计算Flink版共享模式暂不支持自定义函数，仅独享模式支持自定义函数。

定义

与自定义的标量函数类似，自定义的表值函数（UDTF）将0个、1个或多个标量值作为输入参数（可以是变长参数）。与标量函数不同，表值函数可以返回任意数量的行作为输出，而不仅是1个值。返回的行可以由1个或多个列组成。

搭建开发环境

参见环境搭建。

编写业务逻辑代码

UDTF需要在TableFunction类中实现eval方法。open方法和close方法可选。以Java为例，示例代码如下。

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.TableFunction;
public class SplitUdtf extends TableFunction<String> {
    // 可选，open方法可不编写。如果编写，则需要添加声明'import org.apache.flink.table.functions.F
unctionContext;'。
    @Override
    public void open(FunctionContext context) {
        // ... ...
        }
    public void eval(String str) {
        String[] split = str.split("\\|");
        for (String s : split) {
            collect(s);
        }
    }
    // 可选，close方法可不编写。
    @Override
    public void close() {
        // ... ...
        }
}
```

多行返回

UDTF可以通过多次调用 `collect()` 实现将1行的数据转为多行返回。

多列返回

UDTF不仅可以进行1行转多行，还可以1列转多列。如果您需要UDTF返回多列，只需要将返回值声明成Tuple或Row。Tuple或Row解释如下：

- 返回值为Tuple

实时计算Flink版支持使用Tuple1到Tuple25，定义1个字段到25个字段。用Tuple3来返回3个字段的UDTF示例如下。

```
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.table.functions.TableFunction;
// 使用Tuple作为返回值，一定要显式声明Tuple的泛型类型，例如，String、Long和Integer。
public class ParseUdtf extends TableFunction<Tuple3<String, Long, Integer>> {
public void eval(String str) {
String[] split = str.split(",");
// 以下代码仅作示例，实际业务需要添加更多的校验逻辑。
String first = split[0];
long second = Long.parseLong(split[1]);
int third = Integer.parseInt(split[2]);
Tuple3<String, Long, Integer> tuple3 = Tuple3.of(first, second, third);
collect(tuple3);
}
}
```

> **说明** 使用Tuple时，字段值不能为null，且最多只能存在25个字段。

- 返回值为Row

  使用Row来实现返回3个字段的UDTF示例如下。

```
import org.apache.flink.table.types.DataType;
import org.apache.flink.table.types.DataTypes;
import org.apache.flink.table.functions.TableFunction;
import org.apache.flink.types.Row;
public class ParseUdtf extends TableFunction<Row> {
public void eval(String str) {
String[] split = str.split(",");
String first = split[0];
long second = Long.parseLong(split[1]);
int third = Integer.parseInt(split[2]);
Row row = new Row(3);
row.setField(0, first);
row.setField(1, second);
row.setField(2, third);
collect(row);
}
@Override
// 如果返回值是Row，则必须重载实现getResultType方法，显式地声明返回的字段类型。
public DataType getResultType(Object[] arguments, Class[] argTypes) {
return DataTypes.createRowType(DataTypes.STRING, DataTypes.LONG, DataTypes.INT);
}
}
```

> **说明** Row的字段值可以是null，但如果需要使用Row，必须重载实现 `getResultType` 方法。

SQL语法

UDTF支持cross join和left join，在使用UDTF时需要添加 `lateral` 和 `table` 关键字。以 `ParseUdtf` 为例，需要先注册一个Function名字。

```
CREATE FUNCTION parseUdtf AS 'com.alibaba.blink.sql.udtf.ParseUdtf';
```

- cross join

  左表的每一行数据都会关联上UDTF产出的每一行数据，如果UDTF不产出任何数据，则这1行不会输出。

  ```
  select S.id, S.content, T.a, T.b, T.c
  from input_stream as S,
  lateral table(parseUdtf(content)) as T(a, b, c);
  ```

- left join

  左表的每一行数据都会关联上UDTF产出的每一行数据，如果UDTF不产出任何数据，则这1行的UDTF的字段会用null值填充。

  > **说明** left join UDTF语句后面必须添加 `on true` 参数。

  ```
  select S.id, S.content, T.a, T.b, T.c
  from input_stream as S
  left join lateral table(parseUdtf(content)) as T(a, b, c) on true;
  ```

注册使用

1. 登录实时计算控制台。
2. 在顶部菜单中，单击**开发**。
3. 在左侧的导航栏中，单击**资源引用**。
4. 在**资源引用**页签的右上角，单击**新建资源**。
5. 在**上传资源**页面，输入资源配置信息。

参数名称	说明
上传方式	实时计算控制台上仅支持本地上传。 **说明** 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。
资源选择	单击**选择资源**，选择需要引用的资源。
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

6. 在**资源引用**页签中，将鼠标悬停在对应作业的右侧的**更多**上。
7. 在下拉列表中，选择**引用**。

8. 在作业的编辑窗口的顶部，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```

上线和启动

自定义聚合函数（UDTF）的上线和启动步骤，请参见上线和启动。

UDTF示例

```
-- UDTF str.split("\\|");
create function splitUdtf as 'com.hjc.test.blink.sql.udx.SplitUdtf';
create table sls_stream(
a INT,
b BIGINT,
c VARCHAR
) with (
type='sls',
endPoint='yourEndpoint',
accessKeyId='yourAccessKeyId',
accessKeySecret='yourAccessSecret',
startTime = '2017-07-04 00:00:00',
project='yourProjectName',
logStore='yourLogStoreName',
consumerGroup='consumerGroupTest2'
);
-- 将c字段传入splitUdtf，切分后得到多行1列的表T(s)。s表示字段名字。
create view v1 as
select a,b,c,s
from sls_stream,
lateral table(splitUdtf(c)) as T(s);
create table rds_output(
id INT,
len BIGINT,
content VARCHAR
) with (
type='rds',
url='yourDatabaseURL',
tableName='yourDatabaseTableName',
userName='yourDatabaseUserName',
password='yourDatabasePassword'
);
insert into rds_output
select
a,b,s
from v1;
```

4.11.5. 使用IntelliJ IDEA开发自定义函数

本文为您介绍如何使用IntelliJ IDEA开发实时计算Flink版自定义函数，包括搭建开发环境和实时计算Flink版作业中引用自定义函数。

背景信息

> **注意**
> - 仅独享模式支持自定义函数功能。
> - 请使用IntelliJ IDEA工具开发自定义函数。

配置Maven

1. 下载Maven。
   i. 登录Maven官网下载页面。
   ii. 下载apache-maven-3.5.3-bin.tar.gz。
   iii. 解压下载的安装包到指定目录，例如：*/Users/<userName>/Documents/maven*。
2. 配置环境变量。
   i. 在Terminal中，执行 `vim ~/.bash_profile` 命令。
   ii. 在.bash_profile文件中添加如下命令。

   ```
   export M2_HOME=/Users/<userName>/Documents/maven/apache-maven-3.5.3
   export PATH=$PATH:$M2_HOME/bin
   ```

   iii. 保存并关闭.bash_profile文件。
   iv. 执行 `source ~/.bash_profile` 命令使配置生效。
3. 执行 `mvn -v` 命令查看配置是否生效。

   如果打印如下信息，则配置生效。

   ```
   Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-04T03:39:06+08:00
   )
   Maven home: /Users/<userName>/Documents/maven/apache-maven-3.5.0
   Java version: 1.8.0_121, vendor: Oracle Corporation
   Java home: /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/jre
   Default locale: zh_CN, platform encoding: UTF-8
   OS name: "mac os x", version: "10.12.6", arch: "x86_64", family: "mac"
   ```

搭建开发环境

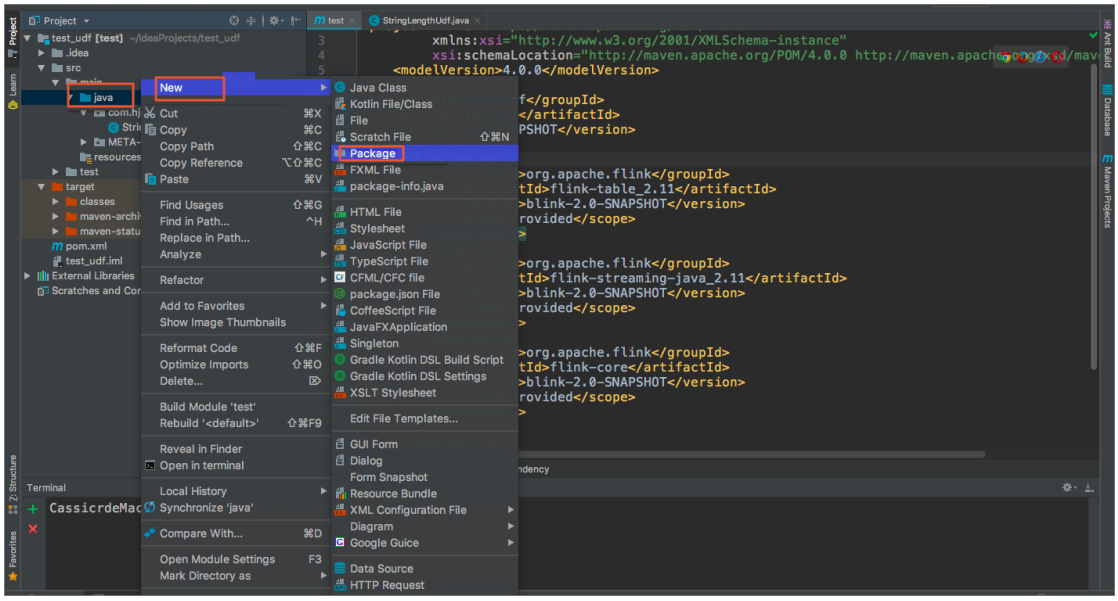
1. 下载UDX示例。
2. 在Linux环境下解压下载文件。

   ```
   tar xzvf RealtimeCompute-udxDemo.gz
   ```

3. 打开**IntelliJ IDEA**，单击**Open**打开下载的UDX示例。

实时计算Flink版作业引用JAR包

1. 创建Package。
   i. 右键单击java > New > Package。



   ii. 在New Package中输入Package名称。本文以 `com.hjc.test.blink.sql.udx` 为例。
   iii. 单击OK。
2. 创建Class。
   i. 右键单击com.hjc.test.blink.sql.udx > New > Java Class。
   ii. 在Create New Class中，输入Class名称。

> 说明　Kind保持默认选择Class。

iii. 单击OK。

3. 在Class中输入以下代码。

```
package com.hjc.test.blink.sql.udx;
import org.apache.flink.table.functions.FunctionContext;
import org.apache.flink.table.functions.ScalarFunction;
public class StringLengthUdf extends ScalarFunction {
    // 可选，open方法可以不编写。
    // 如果编写open方法，需要声明'import org.apache.flink.table.functions.FunctionContext;
'。
    @Override
    public void open(FunctionContext context) {
        }
    public long eval(String a) {
        return a == null ? 0 : a.length();
    }
    public long eval(String b, String c) {
        return eval(b) + eval(c);
    }
    //可选，close方法可以不编写。
    @Override
    public void close() {
        }
}
```

4. 在Terminal中，执行 `mvn package` 或 `mvn assembly:assembly` ，将项目写入JAR包。

> **说明**
> - 如果需要将第三方依赖写入JAR包，请使用 `mvn assembly:assembly` 。
> - 编译后的JAR包为RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT.jar或RealtimeCompute-udxDemo/target/RTCompute-udx-1.0-SNAPSHOT-jar-with-dependencies.jar（将第三方依赖写入JAR包）。

5. 实时计算Flink版作业引用JAR包。
   i. 登录实时计算控制台。
   ii. 在顶部菜单中，单击**开发**。
   iii. 在左侧的导航栏中，单击**资源引用**。

iv. 在**资源引用**页签的右上角，单击**新建资源**。

参数名称	说明
上传方式	实时计算控制台上仅支持本地上传。 **说明** 本地上传JAR包的大小上限为300 MB。如果JAR包大小超过300 MB，请在集群绑定的OSS控制台上，或通过OpenAPI的方式上传JAR包。
资源选择	单击**选择资源**，选择需要引用的资源。
资源名称	输入资源名称。
资源备注	输入资源备注信息。
资源类型	选择引用资源类型，JAR、DICTIONARY或PYTHON。

v. 在**资源引用**页签中，鼠标悬停在对应作业的右侧的**更多**上。

vi. 在下拉列表中，选择**引用**。

vii. 在作业的编辑窗口顶部，输入自定义函数声明，示例如下。

```
CREATE FUNCTION stringLengthUdf AS 'com.hjc.test.blink.sql.udx.StringLengthUdf';
```