flink sql 指南

说明

flink sql 相关知识整理。

表声明语句

flink 可以通过ddl语句声明一个表。表的声明在flink中分为2个部分：connector和format。
connector复杂读写数据（对接外部存储系统），format 负责解析数据。

通过如下方式可以声明一个表：

```
create table tablename(
   field1 field_type
) with (
   'key' = 'value'
)
```

数据类型

```
STRING
BOOLEAN
BYTES   BINARY and VARBINARY are not supported yet.
DECIMAL         Supports fixed precision and scale.
TINYINT
SMALLINT
INTEGER
BIGINT
FLOAT
DOUBLE
DATE
TIME    Supports only a precision of 0.
TIMESTAMP       Supports only a precision of 3.
TIMESTAMP WITH LOCAL TIME ZONE  Supports only a precision of 3.
INTERVAL        Supports only interval of MONTH and SECOND(3).
ARRAY
MULTISET
MAP

```

```
ROW
ANY
```

string 和 varchar 等价。

复制类型：

```
-- 定义数组类型
arr array<int>

-- 定义map类型
`map` map<string, string>

-- 定义嵌套类型
obj row<
  id string,
  name string
  address row<
    city string,
    number int
  >
>
```

连接器

jdbc 连接器

建表sql语句如下：

```
create table dim (
  dim varchar ,
  channel_eight_role_code varchar ,
  channel_source_code varchar,
  CHANNEL_INFO_ID varchar
) with(
  -- 声明连接器类型。flink会通过spi找到连接器，并且进行参数匹配
  'connector.type' = 'jdbc',

  -- jdbc的url
  'connector.url' = 'jdbc:mysql://10.25.76.173:3310/ogg_syncer?useUnicod
e=true&characterEncoding=UTF-8&useSSL=false',

  -- 表名称
  'connector.table' = 'epcis_epcisbase_channel_info',

  -- 驱动类型
```

```
    'connector.driver' = 'com.mysql.jdbc.Driver',

    -- 用过名和密码
    'connector.username' = 'root',
    'connector.password' = 'root',

    -- jdbc作为维表的时候，缓存时间。cache默认未开启。
    'connector.lookup.cache.ttl' = '60s',

    --  jdbc作为维表的时候，缓存的最大行数。cache默认未开启。
    'connector.lookup.cache.max-rows' = '100000',

    -- jdbc作为维表的时候，如果查询失败，最大查询次数
    'connector.lookup.max-retries' = '3',

    -- jdbc写入缓存的最大行数。默认值5000
    'connector.write.flush.max-rows' = '5000',

    -- jdbc 写入缓存flush时间间隔。默认为0，立即写入
    'connector.write.flush.interval' = '2s',

    -- 写入失败，最大重试次数
    'connector.write.max-retries' = '3'
);
```

说明：

1. lookup必须同时设置ttl和max-rows两个参数。
2. jdbc sink 返回的是UpsertStreamSink。

jdbc连接器只支持append/upsert模式，在有些情况下可能无法使用。paic-jdbc基于官方的jdbc开发，
注意添加了如下功能：

1. 添加参数connector.read.sub-query，设置查询jdbc的子查询，如果维度需要自动更新并且需要进行去重
   等操作，去重的sql语句放在connector.read.sub-query中。
2. 支持retract模式，设置update-mode为retract，将返回retract sink。

hbase 连接器

实例：

```
CREATE TABLE MyUserTable (
  hbase_rowkey_name rowkey_type,
```

```
  hbase_column_family_name1 ROW<...>,
  hbase_column_family_name2 ROW<...>
) WITH (
  'connector.type' = 'hbase',
  'connector.version' = '1.4.3',

  -- hbase 表名称
  'connector.table-name' = 'hbase_table_name',  -- required: hbase table
 name

  -- zk地址
  'connector.zookeeper.quorum' = 'localhost:2181',
  -- zk 根节点
  'connector.zookeeper.znode.parent' = '/base',

  -- buffer 缓存大小。默认 2mb。
  'connector.write.buffer-flush.max-size' = '10mb',

  -- 缓冲的最大记录数，无默认值
  'connector.write.buffer-flush.max-rows' = '1000',

  -- flush 时间间隔，默认为0，表示理解刷新到hbase，无缓冲。

  'connector.write.buffer-flush.interval' = '2s'
)
```

说明在定义hbase的schema中，唯一的非row类型的字段，会被当做rowkey处理。
一个完整的实例如下：

```
-- cf 为列族名称， row里面的是列名称
create table hbase_sink(
  rowkey varchar,
  cf row<a1 string, a2 string>
) with(
   'connector.type' = 'hbase',
   'connector.version' = '1.4.3',
   'connector.zookeeper.quorum' = '10.25.76.175:2181,10.25.76.173:2181',
   'connector.zookeeper.znode.parent' = '/hbase',
   'connector.table-name' = 'xuen',
   'connector.write.buffer-flush.interval' = '1s'
);

-- row() 用于创建一个row，按位置对应。
insert into hbase_sink select rowkey, row(a1, a2) from kfk_source;
```

paic-hbase 基于官方的hbase开发，提供如下功能：

1. 支持retract模式，设置update-mode为retract，将返回retract sink。
2. 添加参数connector.write.null，表示是否写入null值。如果为false，值为null的列将不写入hbase，默认为true

kafka连接器

```
CREATE TABLE MyUserTable (
  ...
) WITH (
  'connector.type' = 'kafka',
  'connector.version' = 'universal',

  -- topic名称
  'connector.topic' = 'topic_name',

  -- 固定值。必须有
  'update-mode' = 'append',

  -- 设置kafka集群地址
  'connector.properties.0.key' = 'bootstrap.servers',
  'connector.properties.0.value' = 'localhost:9092',

  -- 设置group id
  'connector.properties.1.key' = 'group.id',
  'connector.properties.1.value' = 'testGroup',

  -- 设置启动模式。如果指定了checkpoint，将从checkpoint读取offset
  -- earliest-offset 最早的offset
  -- latest-offset 最近的offset
  -- group-offsets group 的offset
  -- specific-offsets 指定的offset
  'connector.startup-mode' = 'earliest-offset',

  -- 指定的offset。
  'connector.specific-offsets.0.partition' = '0',
  'connector.specific-offsets.0.offset' = '42',
  'connector.specific-offsets.1.partition' = '1',
  'connector.specific-offsets.1.offset' = '300',

  -- sink分区器。默认是
  'connector.sink-partitioner' = '...',

  -- 指定分区器的类。
```

```
  'connector.sink-partitioner-class' = 'org.mycompany.MyPartitioner'
)
```

paic-kafka基于官方的kafka开发，添加的功能如下：

1. 返回的sink为retract sink。

es 连接器

```


CREATE TABLE MyUserTable (
  ...
) WITH (
  'connector.type' = 'elasticsearch',
  'connector.version' = '6',

  -- 定义host/端口/协议类型
  'connector.hosts.0.hostname' = 'host_name',
  'connector.hosts.0.port' = '9092',
  'connector.hosts.0.protocol' = 'http',

  -- 索引名称
  'connector.index' = 'MyUsers',

  -- es doc-type
  'connector.document-type' = 'user',

  -- update mode。append 将只有insert操作。
  'update-mode' = 'append',

  -- 生成文档id的连接符
  'connector.key-delimiter' = '$',

  -- key null值占位符，默认null
  'connector.key-null-literal' = 'n/a',

  -- 错误处理handler
  'connector.failure-handler' = '...',

  -- optional: configure how to buffer elements before sending them in bu
lk to the cluster for efficiency
  'connector.flush-on-checkpoint' = 'true',   -- optional: disables flushi
ng on checkpoint (see notes below!)
                                              -- ("true" by default)
  -- 每个each bulk request的最大操作数量

```

```
    'connector.bulk-flush.max-actions' = '42',

    -- 缓冲区大小。only MB granularity is supported
    'connector.bulk-flush.max-size' = '42 mb',

    -- flush 频率
    'connector.bulk-flush.interval' = '60000',

    -- bulk 重试方式
    -- optional: backoff strategy ("disabled" by default)
    -- valid strategies are "disabled", "constant",
    -- or "exponential"
    'connector.bulk-flush.back-off.type' = '...',

    -- 最大重试次数
    'connector.bulk-flush.back-off.max-retries' = '3',

    -- 重试间隔时间
    'connector.bulk-flush.back-off.delay' = '30000',

    -- optional: connection properties to be used during REST communicatio
n to Elasticsearch
    -- optional: maximum timeout (in milliseconds)
    -- between retries
    'connector.connection-max-retry-timeout' = '3',

     -- optional: prefix string to be added to every
     -- REST communication
    'connector.connection-path-prefix' = '/v1'
)
```

update-mode=append， es将使用es自动生成的文档ID，也就是只有insert操作。
update-mode=upsert，将使用group-by的字段值作为文档ID进行put操作。

format

目前官方提供的连接器中，只有kafka是需要format的。
这里介绍json format。

```


CREATE TABLE MyUserTable (
  ...
) WITH (
  'format.type' = 'json',
```

```
7
8     -- optional: flag whether to fail if a field is missing or not, false by
9    default
10    'format.fail-on-missing-field' = 'true',
11
12    -- required: define the schema either by using a type string which pars
13   es numbers to corresponding types
14    'format.fields.0.name' = 'lon',
15    'format.fields.0.type' = 'FLOAT',
16    'format.fields.1.name' = 'rideTime',
17    'format.fields.1.type' = 'TIMESTAMP',
18
19    -- or by using a JSON schema which parses to DECIMAL and TIMESTAMP
20    'format.json-schema' =
21      '{
22        "type": "object",
23        "properties": {
24          "lon": {
25            "type": "number"
26          },
27          "rideTime": {
28            "type": "string",
29            "format": "date-time"
30          }
31        }
32      }',
33
34    --  use the table's schema
      'format.derive-schema' = 'true'
    )
```

通常使用’format.derive-schema’ = ‘true’, 不在with中单独定义schema。

flink-json的time attr只支持utc类型的timestamp，这可不太好用。 平台开发了新的json格式，使用如下：

```
1     'format.type' = 'text',
2     'format.udf' = 'com.paic.bentley.flink.sql.format.udf.JSONForNestedUdf'
3   ,
      'format.derive-schema' = 'true',
```

增加功能如下：

1. json解析失败不会导致任务结束，会返回一个null的row
2. 嵌套的json 定义为string，会将这个嵌套对象进行序列化，返回string。方便使用paic_explode_map进行展开，以规避udf

无法处理嵌套类型的问题。

3. 支持多种类型的time attr
   数字类型的毫秒时间戳
   utc时间
   cst时间
   yyyy-[m]m-[d]d hh:mm:ss[.f…] 格式的时间
4. flink 无法定义array类型，请定义为map<int,string>来规避此问题。

flink sink 更新模式

append 模式

没有聚合操作或者有状态的操作，可以使用append模式。历史消息不会更新，只有追加的操作

upsert 模式

需要有group by操作或者append only 为false才可使用。group by 的字段值就是flink更新状态的unique key。
upsert 模式的消息是一个truple: (Boolean, Row)。

append 消息： (true, Row)
delete 消息：(false, Row)，表示删除消息
upsert 消息：(true, Row)，表现已经存在的的唯一key的状态发生了变更。

retract

retract 是通用的类型，任务情况下都可以使用。retract也会group by 的字段值就是flink更新状态的unique key。
retract 模式的消息是一个truple: (Boolean, Row)

append 消息： (true, Row)
delete 消息：(false, Row)，表示删除消息
update 消息：(false, Row)：表示删除这个key，row的值是之前的状态；(false, Row)表示插入这个key，row的值是现在状态

3中模式的使用

一个查询语句需要insert到sink中的时候，flink 会进行更新模式的教程。主要是判断查询 sql 是否有如下状态：

1. appendOnly的是。如果sql查询不包含有状态操作，没有group by，
   appendOnly=true。

2. 是否有unique key，通常就是group by的字段。

如果appendOnly为true: 可以使用append，upsert模式
如果有unique key, 可以使用upsert模式。
retract模式，无显著条件。

需要注意的使用 upsert模式中，group 的字段必须出现在select中，否则会报错。如例子：

```
-- word 必须出现在select中。
select word, count(*) from t group by word
```

flink sql 时间属性（time attr）

在flink 中使用group window 必须定义时间字段。
目前时间字段值能定义在table定义中，查询语句是无法定义时间属性的

定义process time

schema.位置，表示引用字段定义的schema。位置从上到下，从0开始。
schema.0，表示第一个字段。

```
create table t(
    ts timestamp
) with(
  -- 表示此字段为进程处理时间。flink会自动填充值。
  'schema.0.proctime' = 'true'
)
```

定义 event time

```
create table t1(
  ts timestamp
) with (

  -- 声明字段来源
  'schema.2.from' = 'ts',

  -- 声明时间来源于字段
  'schema.2.rowtime.timestamps.type' = 'from-field',

  -- 字段名称
  'schema.2.rowtime.timestamps.from' = 'ts',

```

```
  -- 定义watermark 类型，periodic-bounded表示周期行生成边界
  'schema.2.rowtime.watermarks.type' = 'periodic-bounded',

  -- watermark 最大延迟时间。
  'schema.2.rowtime.watermarks.delay' = '60000'
);
```

group window 函数

group window操作和返回的字段类型，都必须是timestamp类型。

窗口函数，下面这些函数必须出现在group by中，表示按窗口聚合：

函数	
TUMBLE(time_attr, interval)	
HOP(time_attr, interval, interval)	
SESSION(time_attr, interval)	

获取窗口的开始时间（包含）：

```
TUMBLE_START(time_attr, interval)
HOP_START(time_attr, interval, interval)
SESSION_START(time_attr, interval)
```

获取窗口结束时间（不包含）：

```
TUMBLE_END(time_attr, interval)
HOP_END(time_attr, interval, interval)
SESSION_END(time_attr, interval)
```

在group by 中使用了窗口函数，select 必须出现一个START/END函数， 这个group by的字段
必须出现在select中是同一个道理，如下实例：

```
insert into yp_audit_stats_um
  select
    TUMBLE_START(execTime, INTERVAL '1' day) as dt,
    count(distinct userUM) as um_num,
    'day' as dim
  from um_log
    group by TUMBLE(execTime, INTERVAL '1' day);
```

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

```
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;

-- 时间窗口二次聚合。
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
```

TUMBLE_PROCTIME(time_attr, interval)
HOP_PROCTIME(time_attr, interval, interval)
SESSION_PROCTIME(time_attr, interval) 这3个函数是针对的proctime的。和上面的功能一样。

watermark

schema.#.rowtime.watermarks.type 定义水印类型，有如下3中：

1. periodic-ascending：事件的最大时间戳 -1 ,基本相当于无延迟
2. periodic-bounded ， 需要设置最大延迟时间delay，水印的大小为最大时间戳 - delay
3. from-source 保留源中的水印。

默认情况下，watermark到达窗口结束后，完成聚合操作，只会执行1次，相关于如果定义了1天的时间窗口，1天之后才
能看下结果。这个时候如果需要实时看到结果，需要定义触发器

watermark到达窗口结束前的发射策略是否开启：table.exec.emit.early-fire.enabled，默认false
table.exec.emit.early-fire.delay，窗口结束前的发射间隔，单位毫秒。=0，无间隔，>0 间隔时间，<0 非法值。无默认值

watermark到达窗口结束后的发射策略是否开启 table.exec.emit.late-fire.enabled，默认fasle
table.exec.emit.late-fire.delay，设置间隔时间

设置实例：

```
-- set 是平台的功能，非flnik本身的。
set table.exec.emit.early-fire.enabled = true;
set table.exec.emit.early-fire.delay = 1.s;
```

时间单位

```
  private[this] val timeUnitLabels = List(
    DAYS         -> "d day",
    HOURS        -> "h hour",
    MINUTES      -> "min minute",
    SECONDS      -> "s sec second",
    MILLISECONDS -> "ms milli millisecond",
    MICROSECONDS -> "µs micro microsecond",
    NANOSECONDS  -> "ns nano nanosecond"
  )
```

注意事项

time attr 使用 group window的时候不能使用函数。

TUMBLE_START(fun(`timestamp`), INTERVAL ‘1’ hour);
timestamp 失去了时间属性，不能使用TUMBLE

fun(TUMBLE_START(`timestamp`, INTERVAL ‘1’ hour))
fun后，失去了时间属性，和group by字段不能匹配，会判定为主键不完整

flink 维表自动更新

flink 目前使用了look up的方式来自动更新维表，目前只是blink planer支持。

维表的字段更新目前只有jdbc，使用维表的自动更新，需要指定connector.lookup.cache.ttl，和connector.lookup.cache.max-rows2个参数。

hbase也支持维表自动更新，但是没有使用缓存，每次都会查询hbase。

如果使用维表自动更新

流水表需要定义一个proctime字段：

```
create table t(
    ts timestamp
) with(
  -- 表示此字段为进程处理时间。flink会自动填充值。
  'schema.0.proctime' = 'true'
)
```

维表正常定义，定义好lookup相关参数，不需要定义时间参数，在join维表使用如下语法：

```
select a.id from a
left join diw FOR SYSTEM_TIME AS OF a.ts b  on agr.id = b.id
```

join的表名称后面跟：FOR SYSTEM_TIME AS OF a.ts 加表别名称。

维表自动更新的原理

jdbc的JDBCLookupFunction 就是继承了TableFunction。
JDBCTableSource会继承LookupableTableSource，source的getLookupFunction会返回JDBCLookupFunction
blink planer 碰见FOR SYSTEM_TIME AS OF a.ts，会调用getLookupFunction，这是一个表函数，会返回多行。

JDBCLookupFunction 在创建的时候，会创建一个guava cache：private transient Cache<Row, List> cache;
过期时间为ttl设置的值，最大大小为max-rows设置的值。这个cache的key其实是join的全部字段的值，value的值是对应join
字段的值在jdbc中的全部记录。

在eval方法中，传递进来join字段的值，判断cache中是否存在这个记录，如果存在返回。
如果不存在，在jdbc中查找指定join字段值的记录（不会查询全部）, 保存到缓存中，返回。

说明

如果jdbc维表使用了distinct等有状态操作，是无法使用 FOR SYSTEM_TIME AS 语法的，解析会报错。
在paic-jdbc的封装中，可以用connector.read.sub-query参数，传递一个字查询，在这个

子查询中使用
distinct语法，在实现中这个配置做为子查询：

```
return "SELECT " + selectExpressions + " FROM (" + subQuery + ") " +
        quoteIdentifier(tableName) + (conditionFields.length > 0 ? " WHER
E " + fieldExpressions : "");
```

使用方式如下：

```
create table dw_dim_department_source(
) with(
  'connector.read.sub-query' =
    'select
      distinct department_code,
      sec_department_code,
      third_department_code,
      fourth_department_code
    from
      dw_dim_department'

);
```

flink 1.9.1 问题和bug

bug

1. 无法定义 array类型的数据。sql解析失败
2. cast(a as string)，会报错，只能使用varchar
3. decimal类型在sink和source中，会报类型不匹配的错误。

需要注意的的地方

1. flink udf 无法支持嵌套数据类型（row类型）
2. flink sql 区分大小写
3. flink 不会进行自动类型转换。‘1’ * 0.1，会报错，请使用cast强制类型转换。
4. row()生产一个嵌套类型, 只支持写字段名称。写函数，或者加库名称都是不行的
5. map类型的访问，目前只支持： `map['filed']` 这种方式。
6. 数组的下标是从1开始，： `riskGroupInfoList[1]`
7. insert into 不支持部分字段。

8. 如果自己定义了factroy，flink lib 目录下也有factory，java 包需要放到 flink/lib 目录下，否则无法加载自定义的factory。
9. flink kakfa 0.11以上版本和其他版本存在冲突，只能引入一个。

自定义sink/source/factory