ALIBABA CLOUD

阿里云

实时计算（流计算）
最佳实践

文档版本：20210830

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.电商行业最佳实践 05
1.1. 电商场景实战之最新交易记录获取 05
1.2. 电商场景实战之实时态势感知和订单地理分布 07
1.3. 电商场景实战之营销红包 10
1.4. 电商场景实战之实时PV和UV曲线 16
1.5. 电商场景实战之订单与销量统计 21
1.6. 电商场景实战之多类目成交总额管理 29
2.IoT行业最佳实践 32
2.1. IoT解决方案之多维度传感器数据分析 32
3.视频直播行业最佳实践 37
3.1. 视频直播解决方案之视频核心指标监控 37
3.2. 视频直播解决方案之直播数字化运营 42

1.电商行业最佳实践

1.1. 电商场景实战之最新交易记录获取

本文为您介绍如何利用实时计算获取最新交易记录。

背景

在互联网电商的真实交易记录里，订单交易表里会出现不同时间对一笔订单出现多次操作的记录。例如您修改了购买的数量、退货等一系列的操作，但是商家只想获取有效的交易记录。下文为您介绍如何利用实时计算获取最新交易记录。

操作流程

- 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]
```

参数	说明
`ROW_NUMBER()`	计算行号的OVER窗口函数，行号计算从1开始。
`PARTITION BY col1[, col2..]`	指定分区的列，可选。
`ORDER BY col1 [asc|desc][, col2 [asc|desc]...]`	指定排序的列，可以多列不同排序方向。

- 测试案例
  - 测试数据

id（BIGINT）	TIME（VARCHAR）	VALUE（BIGINT）
1	1517898096	5
1	1517887296	44
1	1517872896	32
2	1517872896	10

- 测试语句

```
create table source_table (
 id     BIGINT,
 `TIME`  VARCHAR,
 `VALUE` BIGINT
)with(
 type='datahub',
 endPoint='yourEndpointURL',
 project='yourProjectName',
 topic='yourTableName',
 accessId='yourAccessId',
 accessKey='yourAccessSecret'
);
CREATE TABLE result_table (
   id   BIGINT,
 `TIME`  VARCHAR,
 `VALUE` BIGINT
) WITH (
   type= 'rds',
   url = 'yourRDSDatabaseURL',
   userName = 'yourDatabaseName',
   password = 'yourDatabasePassword',
   tableName = 'yourTableName'
);
INSERT INTO result_table
SELECT id,`TIME`,`VALUE`
FROM (
   SELECT *,
     ROW_NUMBER() OVER (PARTITION BY id ORDER BY `TIME` desc) AS rownum
   FROM
   source_table
)
WHERE rownum <= 1
;
```

- 测试结果

id（BIGINT）	TIME（VARCHAR）	VALUE（BIGINT）
1	1517898096	5
2	1517872896	10

- 难点解析

```
SELECT id,`TIME`,`VALUE`
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY id ORDER BY `TIME` desc) AS rownum
  FROM
  source_table
)
WHERE rownum <= 1
;
```

在订单表里有多个业务时间处理的订单，如果您只想取最后时间的订单里面的数据，需要使用 `row_number() OVER (PARTITION BY id ORDER BY TIME DESC)` 表示根据ID分组，在分组内部根据 `TIME` 降序排列。此函数计算的值表示每组内部排序后的顺序编号（组内是连续且唯一的），这样我们取的就是最新时间的订单里面的数据。over窗口函数的使用，详细请参见OVER窗口。

DEMO示例以及源代码

根据上文介绍的最新交易记录获取解决方案，为您创建了一个包含完整链路的DEMO示例：

- DataHub作为源表。
- RDS作为结果表。

DEMO代码完整，您可参考示例代码，注册上下游数据，制定自己的解决方案。单击DEMO代码进行下载。

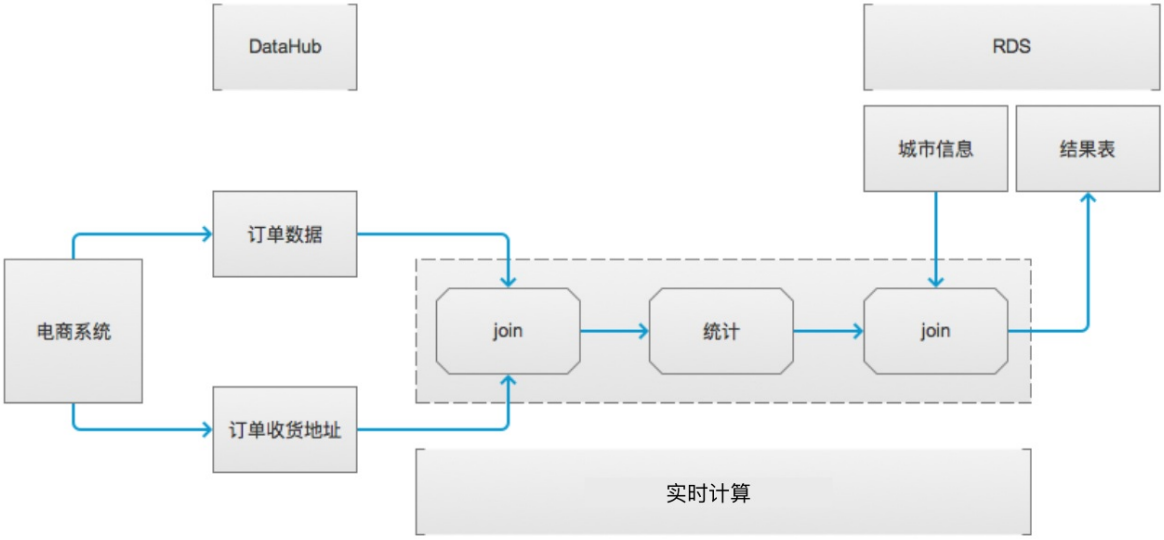
1.2. 电商场景实战之实时态势感知和订单地理分布

本文通过案例为您介绍如何使用实时计算Flink版完成时态势感知和订单地理分布。

背景信息

实时态势感知和订单地理分布有助于企业及时的优化产品品类的分配和发布。以下是某食品电商企业通过实时计算Flink版完成产品的实时态势感知和订单地理分布的案例。

案例



说明 为了聚焦核心逻辑，订单数据格式做了大量精简，只保留了与案例有关的属性。

- 创建数据存储

  在电商系统里，订单与订单地址是分开存储（下单人可以给多个下单地址），所以在订单创建时并没有收货地址，只有在订单提交时才真正的知道收货地址。订单地址里保存的是城市的id（city_id），为了获取地理信息，还需要一张城市表（存储城市地理信息）。目标是按日统计不同地域订单（总销售额）的分布情况。

  - 订单

```
CREATE TABLE source_order (
  id VARCHAR,-- 订单ID。
  seller_id VARCHAR, --卖家ID。
  account_id VARCHAR,--买家ID。
  receive_address_id VARCHAR,--收货地址ID。
  total_price VARCHAR,--订单金额。
  pay_time VARCHAR --订单支付时间。
 ) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='yourProjectName',--您的project。
  topic='yourTopicName',--您的topic。
  roleArn='yourRoleArn',--您的roleArn。
  batchReadSize='500'
);
```

- 订单地址

```
CREATE TABLE source_order_receive_address (
  id VARCHAR,--收货地址ID。
  full_name VARCHAR,--收货人全名。
  mobile_number VARCHAR,--收货人手机号。
  detail_address VARCHAR,--收货详细地址。
  province VARCHAR,--收货省份。
  city_id VARCHAR,--收货城市。
  create_time VARCHAR --创建时间。
 ) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='yourProjectName',--您的project。
  topic='yourTopicName',--您的topic。
  roleArn='yourRoleArn',--您的roleArn。
  batchReadSize='500'
 );
```

- 城市表

```
CREATE TABLE dim_city (
  city_id varchar,
  city_name varchar,--城市名。
  province_id varchar,--所属省份ID。
  zip_code varchar,--邮编。
  lng varchar,--经度。
  lat varchar,--纬度。
  PRIMARY KEY (city_id),
  PERIOD FOR SYSTEM_TIME --定义为维表。
 ) WITH (
  type= 'rds',
  url = 'yourDatabaseURL',--您的数据库URL。
  tableName = 'yourTableName',--您的表名。
  userName = 'yourDatabaseName',--您的用户名。
  password = 'yourDatabasePassword'--您的密码。
 );
```

- 按日统计不同地域订单（总销售额）的分布。

```
CREATE TABLE result_order_city_distribution (
  summary_date varchar,--统计日期。
  city_id bigint,--城市ID。
  city_name varchar,--城市名。
  province_id bigint,--所属省份ID。
  gmv double,--总销售额。
  lng varchar,--经度。
  lat varchar,--纬度。
  primary key (summary_date,city_id)
 ) WITH (
  type= 'rds',
  url = 'yourDatabaseURL',--您的数据库URL。
  tableName = 'yourTableName',--您的表名。
  userName = 'yourDatabaseName',--您的用户名。
  password = 'yourDatabasePassword'--您的密码。
 );
```

- 编辑业务逻辑

```
INSERT INTO result_order_city_distribution
SELECT
 d.summary_date,
 cast(d.city_id as BIGINT),
 e.city_name ,
 cast(e.province_id as BIGINT),
 d.gmv,
 e.lng,
 e.lat
from (
 SELECT
 DATE_FORMAT(a.pay_time,'yyyyMMdd') as summary_date ,
 b.city_id as city_id ,
 round(sum(cast(a.total_price as double)),2) as gmv
 from source_order as a
 join source_order_receive_address as b on a.receive_address_id =b.id
 group by DATE_FORMAT(a.pay_time,'yyyyMMdd'),b.city_id
 --双流JOIN，并根据日期和城市ID得到销售额分。
)d join dim_city FOR SYSTEM_TIME AS OF PROCTIME() as e on d.city_id = e.city_id
 -- JOIN维表，补齐城市地理信息，得到最终结果。
;
```

DEMO示例以及源代码

根据上文介绍的实时态势感知和订单地理分布解决方案，阿里云为您创建了一个包含完整链路的DEMO示例，您可以参考DEMO示例，注册上下游数据，制定自己的解决方案。单击DEMO代码进行下载完整示例。使用示例时，上下游存储请注意以下两点：

- DataHub作为源表
- RDS作为结果表

1.3. 电商场景实战之营销红包

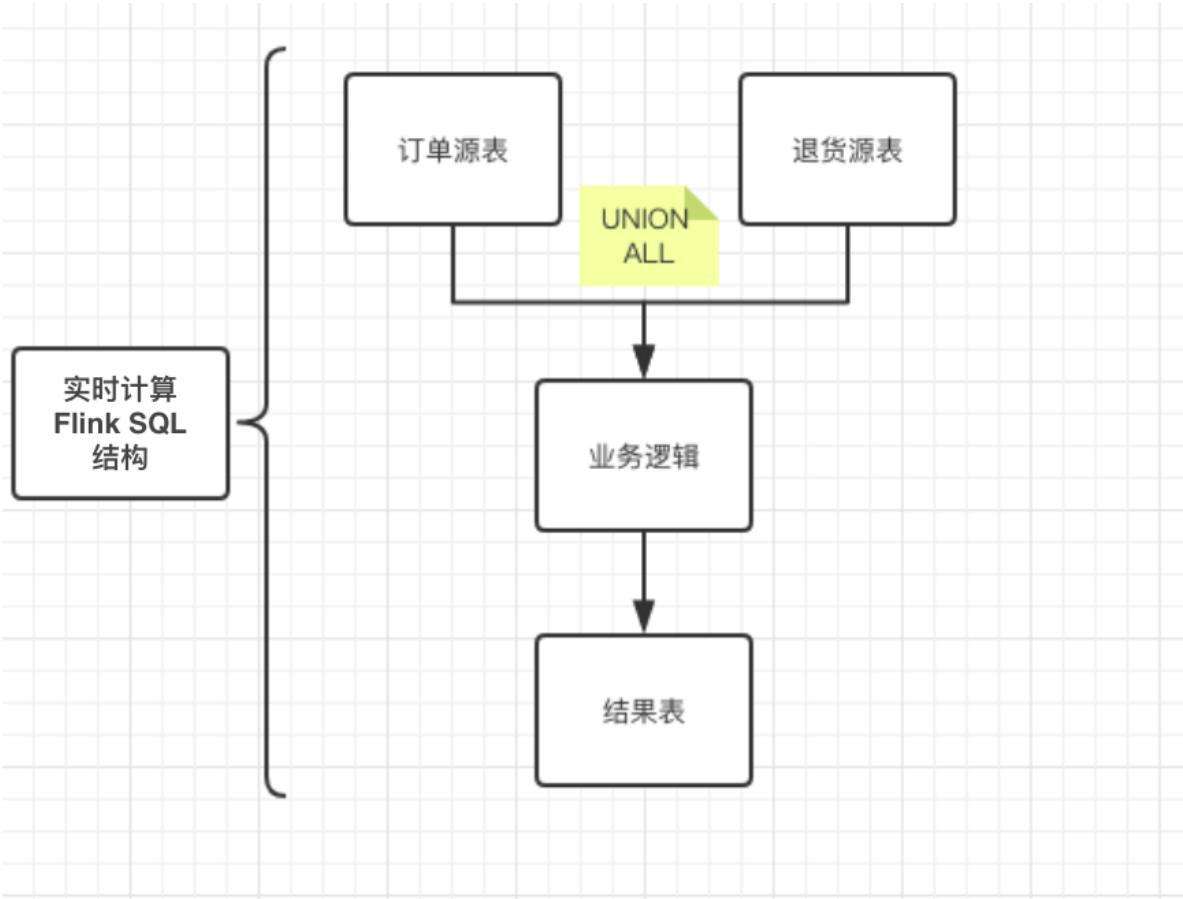
本文为您介绍实时计算如何在营销红包的策略中筛选满足营销红包发放条件的用户。

背景

某商家双十一活动期间的“回血红包”营销方案。用户在消费达到一定金额后，商家为了促进用户继续消费会返给用户一定金额的“回血红包”。实时计算能够为您实时的去监控用户的消费金额，筛选满足营销红包发放条件的用户。

解决方案

- SQL结构图



- 数据源表

- 系统订单源表

> **说明** 为了聚焦核心逻辑，订单数据格式做了大量精简，只保留了与案例有关的属性。

```
CREATE TABLE dwd_tb_trd_pay_ri(
    biz_order_id   VARCHAR, -- '订单ID'
    auction_id     VARCHAR, -- '商品ID'
    auction_title  VARCHAR, -- '商品标题'
    buyer_id       VARCHAR, -- '买家ID'
    buyer_nick     VARCHAR, -- '买家昵称'
    pay_time       VARCHAR, -- '支付时间'
    gmt_create     VARCHAR, -- '创建时间'
    gmt_modified   VARCHAR, -- '修改时间'
    biz_type       VARCHAR, -- '交易类型'
    pay_status     VARCHAR, -- '支付状态'
    `attributes`   VARCHAR, -- '订单标记'
    from_group     VARCHAR, -- '订单来源'
    div_idx_actual_total_fee  DOUBLE  --'成交金额'
) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='yourProjectName',-- '您的project'
  topic='yourTopicName',--'您的topic'
  roleArn='yourRoleArn',-- '您的roleArn'
  batchReadSize='500'
);
```

- 退货源表

> **说明** 为了聚焦核心逻辑，订单数据格式做了大量精简，只保留了与案例有关的属性。

```
CREATE TABLE dwd_tb_trd_rfd_ri(
    biz_order_id   VARCHAR, -- '订单ID'
    auction_id     VARCHAR, -- '商品ID'
    auction_title  VARCHAR, -- '商品标题'
    buyer_id       VARCHAR, -- '买家ID'
    buyer_nick     VARCHAR, -- '买家昵称'
    pay_time       VARCHAR, -- '支付时间'
    gmt_create     VARCHAR, -- '创建时间'
    gmt_modified   VARCHAR, -- '修改时间'
    biz_type       VARCHAR, -- '交易类型'
    pay_status     VARCHAR, -- '支付状态'
    `attributes`   VARCHAR, -- '订单标记'
    from_group     VARCHAR, -- '订单来源'
    refund_fee     DOUBLE   -- '退款金额'
) WITH (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
  project='yourProjectName', --'您的project'
  topic='yourTopicName', --'您的topic'
  roleArn='yourRoleArn', --'您的roleArn'
  batchReadSize='500'
);
```

- 数据结果表

关系型数据库RDS结果表

```
CREATE TABLE tddl_output(
  gmt_create   VARCHAR, --'创建时间'
  gmt_modified VARCHAR, --'修改时间'
  buyer_id     BIGINT, --'买家ID'
  cumulate_amount BIGINT, --'金额'
  effect_time BIGINT, --'支付时间'
  primary key(buyer_id,effect_time)
) WITH (
    type= 'rds',
    url = 'yourDatabaseURL',  --'您的数据库url'
    tableName = 'yourTableName', --'您的表名 '
    userName = 'yourUserName',  --'您的用户名'
    password = 'yourDatabasePassword' --'您的密码'
  );
```

- 业务逻辑

  把订单表和退货表做UNION ALL操作，获取到用户购买的所有商品的信息，统计用户的真实的消费金额和明细。

```
CREATE VIEW  dwd_tb_trd_and_rfd_pay_ri
AS
SELECT
*
FROM(
  (SELECT
    `a`.biz_order_id biz_order_id,
    `a`.auction_id auction_id,
    `a`.auction_title auction_title,
    `a`.buyer_id buyer_id,
    `a`.buyer_nick buyer_nick,
    `a`.pay_time pay_time,
    `a`.gmt_create gmt_create,
    `a`.gmt_modified gmt_modified,
    `a`.biz_type biz_type,
    `a`.pay_status pay_status,
    `a`.`attributes` `attributes`,
    `a`.from_group,
    `a`.div_idx_actual_total_fee div_idx_actual_total_fee
  FROM
    dwd_tb_trd_pay_ri `a`
  )
  UNION ALL
  (
  SELECT
    `b`.biz_order_id biz_order_id,
    `b`.auction_id auction_id,
    `b`.auction_title auction_title,
    `b`.buyer_id buyer_id,
    `b`.buyer_nick buyer_nick,
    `b`.pay_time pay_time,
    `b`.gmt_create gmt_create,
    `b`.gmt_modified gmt_modified,
    `b`.biz_type biz_type,
    `b`.pay_status pay_status,
    `b`.`attributes` `attributes`,
    `b`.from_group,
    `b`.refund_fee div_idx_actual_total_fee --退款取负值
  FROM
    dwd_tb_trd_rfd_ri `b`
  )
);
```

- 去重操作

  在订单表和退货表中可能会存在大量重复的数据。例如商品ID、商品名称等。用MIN函数是为了只取第一次来的参数值，从而过滤掉其他的信息，然后再通过订单号和支付状态做分组取出需要的商品ID、商品名称等。

```
CREATE VIEW filter_hxhb_dwd_tb_trd_and_rfd_pay_ri_distinct AS
SELECT
  biz_order_id biz_order_id,
  pay_status pay_status,
  MIN(auction_id) auction_id,
  MIN(auction_title) auction_title,
  MIN(buyer_id) buyer_id,
  MIN(buyer_nick)  buyer_nick,
  MIN(pay_time)  pay_time,
  MIN(gmt_create)  gmt_create,
  MIN(gmt_modified)  gmt_modified,
  MIN(biz_type)  biz_type,
  MIN(attributes)  attributes,
  MIN(div_idx_actual_total_fee)  div_idx_actual_total_fee
FROM
   dwd_tb_trd_and_rfd_pay_ri
GROUP BY biz_order_id ,pay_status;
```

- 数据汇总所有数据做汇总处理

```
CREATE VIEW ads_tb_trd_and_rfd_pay_ri AS
SELECT
  MIN(gmt_create) gmt_create,  --'创建时间'
  MAX(gmt_modified) gmt_modified,  --'修改时间'
  CAST (buyer_id AS BIGINT) buyer_id,  --'购买ID'
  CAST (date_format(pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMdd') AS BIGINT) as effect_time, --'购买时间'
  SUM(CAST(div_idx_actual_total_fee/100 AS BIGINT)) cumulate_amount  --'总的金额'
FROM
  filter_hxhb_dwd_tb_trd_and_rfd_pay_ri_distinct
GROUP BY
  buyer_id,date_format(pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMdd');
```

Q：为什么取MAX、MIN?

```
MIN(gmt_create) gmt_create, --'创建时间'
MAX(gmt_modified) gmt_modified, --'修改时间'
```

A：MIN(gmt_create)是指的订单的第一笔订单的创建时间，MAX(gmt_modified)是指的最后一比订单的时间。根据订单的业务逻辑您可以用MAX和MIN来取相应的值。

> **说明** 如果时间字段不是BIGINT类型的可以用相应的内置函数做转换，详情请参见内置函数。

- 数据入库

  把统计好的数据插入RDS结果，再由其他的推送软件把相应的红包数发放给满足条件的用户。

```
INSERT INTO tddl_output
SELECT
  gmt_create,
  gmt_modified,
  buyer_id,
  cumulate_amount,
  effect_time
from ads_tb_trd_and_rfd_pay_ri
where cumulate_amount>0;
```

Q：为什么要取总的金额大于0的数？

```
cumulate_amount>0
```

A：是为了过滤掉在上一步做UNION ALL的退货商品的金额。

总结

Q：在大量的订单和退货单中怎样才能清洗出我们所需要的数据？

A：在真实的购买记录中会存在大量的购买量和退货量，用UNION ALL把两张或者是多张表合并成一张大表，然后再通过具体的业务逻辑去重、聚合操作。最后把用户所有的真实订单交易金额写入存储中为后续的红包推送做准备。

DEMO示例以及源代码

根据上文介绍的营销红包解决方案，为您创建了一个包含完整链路的DEMO示例。

- DataHub作为源表
- RDS作为结果表

DEMO代码完整，您可参考示例代码，注册上下游数据，制定自己的营销红包解决方案。单击DEMO代码进行下载。

1.4. 电商场景实战之实时PV和UV曲线

本文以实时计算合作伙伴格格家的案例为例，为您介绍如何使用实时计算制作实时PV和UV曲线图。

背景

随着新零售的概念慢慢崛起，互联网电商行业竞争越来越激烈。实时数据信息对于电商行业尤为重要，例如实时地统计网页PV和UV的总量。

案例

- 业务架构图

- 业务流程
  i. 通过数据总线（DataHub）提供的SDK，同步Binlog日志数据到DataHub。
  ii. 阿里云实时计算订阅DataHub的数据进行实时计算。
  iii. 插入实时数据到云数据库RDS。
  iv. 通过阿里云DataV数据可视化或其他大屏展示结果数据。
- 准备工作 日志源表

字段	数据类型	说明
account_id	VARCHAR	用户ID
client_ip	VARCHAR	客户端IP
client_info	VACHAR	设备机型信息
platform	VARHCAR	系统版本信息
imei	VARCHAR	设备唯一标识
version	BIGINT	版本号
action	BIGINT	页面跳转描述
gpm	VARCHAR	埋点链路
c_time	VARCHAR	请求时间
target_type	VARCHAR	目标类型
target_id	VARCHAR	目标ID
udata	VARCHAR	扩展信息

字段	数据类型	说明
session_id	VARHCAR	会话ID
product_id_chain	VARHCAR	商品ID串
cart_product_id_chain	VARCHAR	加购商品ID
tag	VARCHAR	特殊标记
position	VARCHAR	位置信息
network	VARCHAR	网络使用情况
p_dt	VARCHAR	时间分区
p_platform	VARCHAR	系统版本信息

数据库RDS结果表

字段	数据类型	说明
summary_date	BIGINT	统计日期
summary_min	VARCHAR	统计分钟
pv	BIGINT	单击量
uv	BIGINT	访客量 说明 一天内同个访客多次访问网站计为一个UV。
currenttime	TIMESTAMP	当前时间

- 业务逻辑

```
--数据的订单源表。
CREATE TABLE source_ods_fact_log_track_action (
  account_id          VARCHAR,--用户ID。
  client_ip           VARCHAR,--客户端IP。
  client_info         VARCHAR,--设备机型信息。
  platform            VARCHAR,--系统版本信息。
  imei                VARCHAR,--设备唯一标识。
  `version`           VARCHAR,--版本号。
  `action`            VARCHAR,--页面跳转描述。
  gpm                 VARCHAR,--埋点链路。
  c_time              VARCHAR,--请求时间。
  target_type         VARCHAR,--目标类型。
  target_id           VARCHAR,--目标ID。
  udata               VARCHAR,--扩展信息，JSON格式。
  session_id          VARCHAR,--会话ID。
  product_id_chain    VARCHAR,--商品ID串
```

```
  product_id_chain          VARCHAR,--商品ID串。
  cart_product_id_chain        VARCHAR,--加购商品ID。
  tag                  VARCHAR,--特殊标记。
  `position`                VARCHAR,--位置信息。
  network                 VARCHAR,--网络使用情况。
  p_dt                 VARCHAR,--时间分区天。
  p_platform               VARCHAR --系统版本信息。
) WITH (
  type='datahub',
  endPoint='yourEndpointURL',
  project='yourProjectName',
  topic='yourTopicName',
  accessId='yourAccessId',
  accessKey='yourAccessSecret',
  batchReadSize='1000'
);
CREATE TABLE result_cps_total_summary_pvuv_min (
  summary_date         bigint,--统计日期。
  summary_min         varchar,--统计分钟。
  pv              bigint,--单击量。
  uv              bigint,--一天内同个访客多次访问计算为一个UV。
  currenttime          timestamp,--当前时间。
  primary key (summary_date,summary_min)
) WITH (
  type= 'rds',
  url = 'yourRDSDatabaseURL',
  userName = 'yourDatabaseUserName',
  password = 'yourDatabasePassword',
  tableName = 'yourTableName'
);
CREATE VIEW result_cps_total_summary_pvuv_min_01 AS
select
cast(p_dt as bigint) as summary_date --时间分区。
,count(client_ip) as pv --客户端的IP。
,count(distinct client_ip) as uv  --客户端去重。
,cast(max(c_time ) as TIMESTAMP)  as c_time  --请求的时间。
from source_ods_fact_log_track_action
group by p_dt;
INSERT into  result_cps_total_summary_pvuv_min
select
a.summary_date, --时间分区。
cast(DATE_FORMAT(c_time,'HH:mm')  as varchar) as summary_min, --取出小时分钟级别的时间。
a.pv,
a.uv,
CURRENT_TIMESTAMP  as currenttime  --当前时间。
from result_cps_total_summary_pvuv_min_01 AS a
;
```

- 难点解析

  为了方便理解结构化代码和代码维护，我们推荐使用View（数据视图概念）把业务逻辑拆分成两个模块。

- 模块一

```
CREATE VIEW result_cps_total_summary_pvuv_min_01 AS
select
cast(p_dt as bigint) as summary_date, --时间分区。
count(client_ip) as pv, --客户端的IP。
count(distinct client_ip) as uv, --客户端去重。
cast(max(c_time) as TIMESTAMP) as c_time  --请求的时间。
from source_ods_fact_log_track_action
group by p_dt;
```

  - PV是客户访问网站后的单击次数，UV是去重的客户IP数。
  - cast(max(c_time) as TIMESTAMP)为最后请求时间。
  - 以p_dt作为时间分区，单位为天，以max(c_time)作为访问网站的截止时间，向数据库插入一条PV和UV。

结果

p_dt	pv	uv	max(c_time)
2017-12-12	1000	100	2017-12-12 9:00:00
2017-12-12	1500	120	2017-12-12 9:01:00
2017-12-12	2200	200	2017-12-12 9:02:00
2017-12-12	3300	320	2017-12-12 9:03:00

- 模块二

```
INSERT into  result_cps_total_summary_pvuv_min
select
a.summary_date, --时间分区，天单位为。
cast(DATE_FORMAT(c_time,'HH:mm')  as varchar) as summary_min, --取出小时分钟级别的时间。
a.pv,
a.uv,
CURRENT_TIMESTAMP  as currenttime  --当前时间。
from result_cps_total_summary_pvuv_min_01 AS a;
```

把模块一的数据精确到小时分钟级别取出后，根据数据得出PV和UV的增长曲线。

示例及源代码

根据上文介绍的PV和UV曲线解决方案，阿里云为您创建了一个包含完整链路的DEMO示例，您可以参考DEMO示例，注册上下游存储，从而得出您的PV和UV曲线图。单击示例代码下载完整示，使用示例时，上下游存储请注意以下两点：

- DataHub作为源表。
- RDS作为结果表。

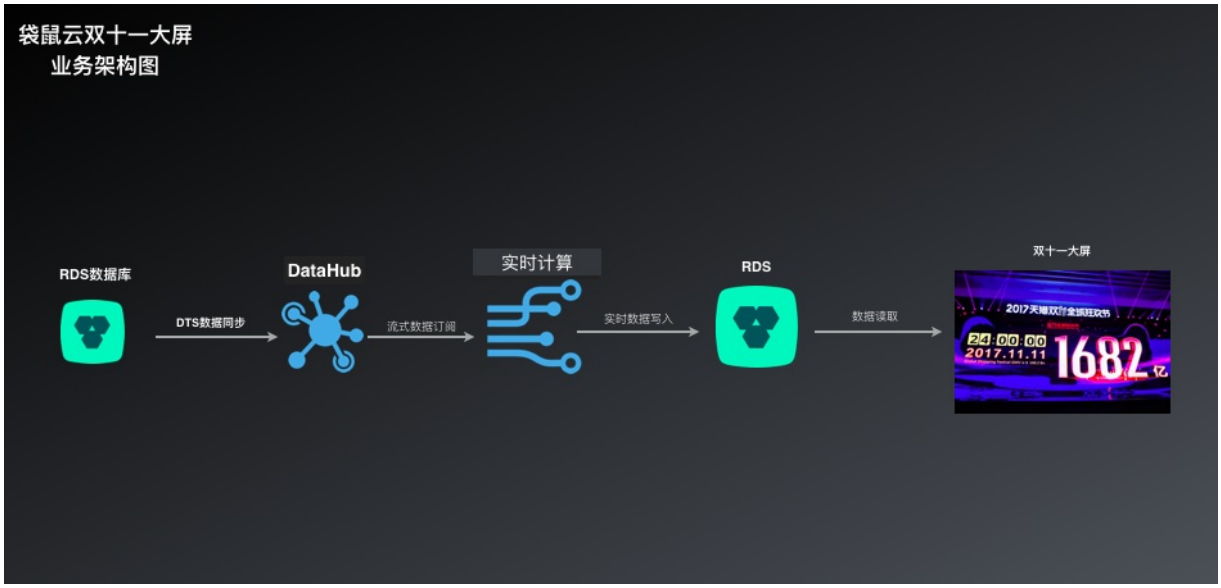
1.5. 电商场景实战之订单与销量统计

本文通过案例为您介绍如何使用实时计算完成订单与销量的统计。

背景信息

以下案例是实时计算的合作伙伴袋鼠云通过阿里云实时计算来完成电商订单管理的案例。

业务架构图



业务流程：

1. 使用数据传输服务DTS把您的数据同步到大数据总线（DataHub）。具体步骤请参见MySQL到DataHub数据实时同步。
2. 阿里云实时计算订阅大数据总线（DataHub）的数据进行实时计算。
3. 将实时数据插入到RDS的云数据库。
4. 通过阿里云的DataV或者是其他的大屏完成数据展示。

准备工作

将RDS MySQL产生的增量数据实时同步到DataHub中的Topic。由RDS经过DTS数据同步到大数据总线（DataHub）Schema表的信息。

具体步骤请参见MySQL到DataHub数据实时同步。

orders_real源表

字段名	数据类型	详情
dts_ordercodeofsys	VARCHAR	订单编号
dts_paytime	BIGINT	订单付款时间
dts_deliveredtime	VARCHAR	订单发货时间
dts_storecode	VARCHAR	店铺编号
dts_warehousecode	VARCHAR	仓库code
dts_cancelled	BIGINT	是否取消
dts_delivered	BIGINT	是否发货
dts_receivercity	VARCHAR	收货人城市
dts_receiverprovince	VARCHAR	收货人省份

字段名	数据类型	详情
dts_record_id	VARCHAR	记录ID
dts_operation_flag	VARCHAR	操作Flag
dts_instance_id	VARCHAR	数据库instanceId
dts_db_name	VARCHAR	数据库名
dts_table_name	VARCHAR	数据表
dts_utc_timestamp	VARCHAR	更新时间
dts_before_flag	VARCHAR	变更前标识
dts_after_flag	VARCHAR	变更后标识

orderdetail_real源表

字段名	数据类型	详情
dts_ordercodeofsys	VARCHAR	订单编号
dts_skuname	VARCHAR	商品名字
dts_skucode	VARCHAR	商品编号
dts_quantity	BIGINT	数量
dts_dividedamount	DOUBLE	发货金额
dts_salechanneldividedamount	DOUBLE	渠道销售金额
dts_initialcost	DOUBLE	成本
dts_record_id	VARCHAR	记录ID
dts_operation_flag	VARCHAR	操作Flag
dts_instance_id	VARCHAR	数据库instanceId
dts_db_name	VARCHAR	数据库名字
dts_table_name	VARCHAR	表名
dts_utc_timestamp	VARCHAR	更新时间
dts_before_flag	VARCHAR	变更前标识
dts_after_flag	VARCHAR	变更后标识

编写业务逻辑

```
--数据的订单源表。
create table orders_real(
 dts_ordercodeofsys varchar,
 dts_paytime bigint,
 dts_deliveredtime varchar,
 dts_storecode varchar,
 dts_warehousecode varchar,
 dts_cancelled bigint,
 dts_delivered bigint,
 dts_receivercity varchar,
 dts_receiverprovince varchar,
 dts_record_id varchar,
 dts_operation_flag varchar,
 dts_instance_id varchar,
 dts_db_name varchar,
 dts_table_name varchar,
 dts_utc_timestamp varchar,
 dts_before_flag varchar,
 dts_after_flag varchar
) with (
 type='datahub',
 endPoint='http://dh-cn-****.com',
 project='your',
 topic='表名',
 accessId='您的ID',
 accessKey='您的KEY'
);
create table orderdetail_real(
 dts_ordercodeofsys varchar,
 dts_skuname varchar,
 dts_skucode varchar,
 dts_quantity bigint,
 dts_dividedamount double,
 dts_salechanneldividedamount double,
 dts_initialcost double,
 dts_record_id varchar,
 dts_operation_flag varchar,
 dts_instance_id varchar,
 dts_db_name varchar,
 dts_table_name varchar,
 dts_utc_timestamp varchar,
 dts_before_flag varchar,
 dts_after_flag varchar
) with (
 type='datahub',
 endPoint='http://dh-cn-****.com',
 project='yourPorjectName',
 topic='yourTableName',
 accessId='yourAccessId',
 accessKey='yourAccessSecret'
);
create table ads_all_count_amount(
 bill_date varchar,--下单时间。
 bill_count bigint,--总的订单总数。
```

```
  qty bigint,--总的销售量。
  primary key (bill_date)
) with (
  type='rds',
  url='jdbc:mysql://rm-XXXX.mysql.rds.aXXXXcs.com:3306/XXXX',
  tableName='yourDatabaseTableName',
  userName='yourDatabaseAccount',
  password='yourDatabasePassword'
);
--订单源表，最新交易时间的商品编号。
CREATE VIEW new_paytime AS
  SELECT
  dts_ordercodeofsys,
  MAX(dts_paytime) AS dts_paytime
FROM orders_real
  GROUP BY dts_ordercodeofsys ;
--订单详情表，有效订单的订单编码、商品名称、商品编号、数量的信息。
CREATE VIEW new_orderdetail AS
SELECT
  dts_ordercodeofsys,
  dts_skuname,
  dts_skucode,
  CASE WHEN dts_operation_flag='U'
  AND dts_before_flag='Y'
  AND dts_after_flag='N' THEN -1*dts_quantity
  WHEN dts_operation_flag='U'
  AND dts_before_flag='N'
  AND dts_after_flag='Y' THEN dts_quantity
  WHEN dts_operation_flag='D' THEN -1*dts_quantity
  ELSE dts_quantity
  END AS dts_quantity
FROM
  orderdetail_real;
--订单总单数，总销售量。
INSERT INTO ads_all_count_amount
  SELECT
  FROM_UNIXTIME(cast(a.dts_paytime/1000000 AS bigint),'yyyyMMdd') AS bill_date,
  COUNT(DISTINCT a.dts_ordercodeofsys) AS bill_count,
  SUM(b.dts_quantity) AS qty
from
  new_paytime a
  join
  new_orderdetail b
  ON a.dts_ordercodeofsys=b.dts_ordercodeofsys
  GROUP BY
  FROM_UNIXTIME(CAST(a.dts_paytime/1000000 AS bigint),'yyyyMMdd');
```

难点解析

为了方便您理解结构化代码和代码维护，推荐使用View（数据视图概念）将业务逻辑拆分为三个模块。

- 模块一：根据订单编号进行分组

同一个编号订单会有多次业务操作（例如下单、付款、发货），并在Binlog日志中形成多条同一订单编号的订单流水记录。使用 `MAX(dts_paytime)` 获取同一编号的最后一次操作数据库最终付款交易时间。

```
CREATE VIEW new_paytime AS
SELECT
 dts_ordercodeofsys,
 MAX(dts_paytime) AS dts_paytime
FROM orders_real
GROUP BY dts_ordercodeofsys;
```

- 模块二：生成有效订单的信息

```
--订单详情表，有效订单的订单编码、商品名称、商品编号、数量的信息。
CREATE VIEW new_orderdetail AS
SELECT
  dts_ordercodeofsys,
  dts_skuname,
  dts_skucode,
CASE WHEN dts_operation_flag='U'
    AND dts_before_flag='Y'
    AND dts_after_flag='N' THEN -1*dts_quantity
  WHEN dts_operation_flag='U'
    AND dts_before_flag='N'
    AND dts_after_flag='Y' THEN dts_quantity
  WHEN dts_operation_flag='D' THEN -1*dts_quantity
  ELSE dts_quantity
  END AS dts_quantity
FROM orderdetail_real;
```

数据库日志会获取所有的数据记录的变更，而每个订单是有状态的，如下表所示。

字段名	数据类型	详情
dts_record_id	VARCHAR	记录ID。增量日志的唯一标识，唯一递增。如果变更类型为Update，那么增量更新会被拆分成2条，一条Insert，一条Delete。这两条记录具有相同的 `record_id` 。
dts_operation_flag	VARCHAR	标示这条增量日志的操作类型，取值为： ○ I：Insert ○ D：Delete ○ U：Update
dts_instance_id	VARCHAR	数据库 `instanceId` 。这条增量日志所对应的数据库的Server ID。
dts_db_name	VARCHAR	这条增量更新日志更新的表所在的数据库库名。
dts_table_name	VARCHAR	这条增量更新日志更新的表名。
dts_utc_timestamp	VARCHAR	这条增量日志的操作时间戳，为这个更新操作记录Binlog的时间戳。时间戳为UTC时间。

字段名	数据类型	详情
dts_before_flag	VARCHAR	表示这条增量日志后面带的各个Column值是否为更新前的值。取值包括：`Y` 和 `N` 。当后面的Column为更新前的值时，`dts_before_flag=Y` ，当后面的Column值为更新后的值时，`dts_before_flag=N` 。
dts_after_flag	VARCHAR	表示这条增量日志后面带的各个Column值是否为更新后的值。取值包括：`Y` 和 `N` 。当后面的Column为更新前的值时，`dts_after_flag=N` ，当后面的Column值为更新后的值时，`dts_after_flag=Y` 。

对于不同的操作类型，增量日志中的 `dts_before_flag` 和 `dts_after_flag` 定义如下：

i. 操作类型为Insert

所有Column值为新插入的记录值，即为更新后的值，所以 `dts_before_flag=N` ， `dts_after_flag=Y` 。

dts_record_id	dts_instance_id	dts_db_name	dts_table_name	dts_operation_flag	dts_utc_timestamp	dts_before_flag	dts_after_flag	dts_col1	....	dts_colN
1	234	db1	sbtest1	I	1476258462	N	Y	1	.....	JustInsert

ii. 操作类型为Update

Update操作被拆为2条增量日志。这两条增量日志的 `dts_record_id` ， `dts_operation_flag` 及 `dts_utc_timestamp` 相同。第一条日志记录更新前的值，所以 `dts_before_flag=Y` ， `dts_after_flag=N` 。第二条日志记录了更新后的值，所以 `dts_before_flag=N` ， `dts_after_flag=Y` 。

dts_record_id	dts_instance_id	dts_db_name	dts_table_name	dts_operation_flag	dts_utc_timestamp	dts_before_flag	dts_after_flag	dts_col1	....	dts_colN
2	234	db1	sbtest1	I	1476258463	Y	N	1	.....	JustInsert
2	234	db1	sbtest1	I	1476258463	N	Y	1	.....	JustUpdate

iii. 操作类型为Delete

所有Column值为被删除的记录值，即为更新前的值。所以 `dts_before_flag=Y` ， `dts_after_flag=N` 。

dts_record_id	dts_instance_id	dts_db_name	dts_table_name	dts_operation_flag	dts_utc_timestamp	dts_before_flag	dts_after_flag	dts_col1	....	dts_colN
3	234	db1	sbtest1	D	1476258464	Y	N	1	.....	JustUpdate

- 模块三：统计总订单数和销售额

```
SELECT
  from_unixtime(CAST(a.dts_paytime/1000000 AS bigint),'yyyyMMdd') AS bill_date,
  COUNT(DISTINCTa.dts_ordercodeofsys) AS bill_count,
  SUM(b.dts_quantity) AS qty
from
  new_paytime as a
join
  new_orderdetail as b
ON
  a.dts_ordercodeofsys=b.dts_ordercodeofsys
GROUP BY
from_unixtime(CAST(a.dts_paytime/1000000 AS bigint),'yyyyMMdd');
```

DEMO示例以及源代码

根据上文介绍的订单与销量统计解决方案，为您创建了一个包含完整链路的DEMO示例，如下所示：

- DataHub作为源表。
- RDS作为结果表。

DEMO代码完整，您可参考示例代码，注册上下游数据，制定自己的订单与销量统计解决方案。单击DEMO代码进行下载。

常见问题

- Q：模块二中，如何判断有效交易订单？

  A：首先是要满足 `dts_operation_flag=U` 或者 `dts_operation_flag=I` ，然后 `dts_before_flag` 代表的是变更前订单状态， `dts_after_flag` 是变更后订单状态。所以有效交易订单如下。

```
dts_operation_flag='U'
   AND dts_before_flag='N'
   AND dts_after_flag='Y' THEN dts_quantity
```

- Q：模块二中，为什么 `THEN -1*dts_quantity` ？

  A：订单的取消或者是交易没有成功，在总的销量里也会记录。为了保证总销量的正确性，所以把没有成交的订单数量设为负数，在计算总的销量会减去这个数量。

- Q：模块三中，为什么订单源表和订单详情需要进行JOIN操作？

  A： `new_paytime` 查出的是最新交易的时间的所有的订单编号。 `new_orderdetail` 查询的是所有有效订单的订单编码、商品名称、商品编号、数量的信息。两张表JOIN是为了方便用户来统计订单总数和总的销量。

1.6. 电商场景实战之多类目成交总额管理

本文通过案例为您介绍如何使用实时计算完成数据的实时处理、多类目的管理。

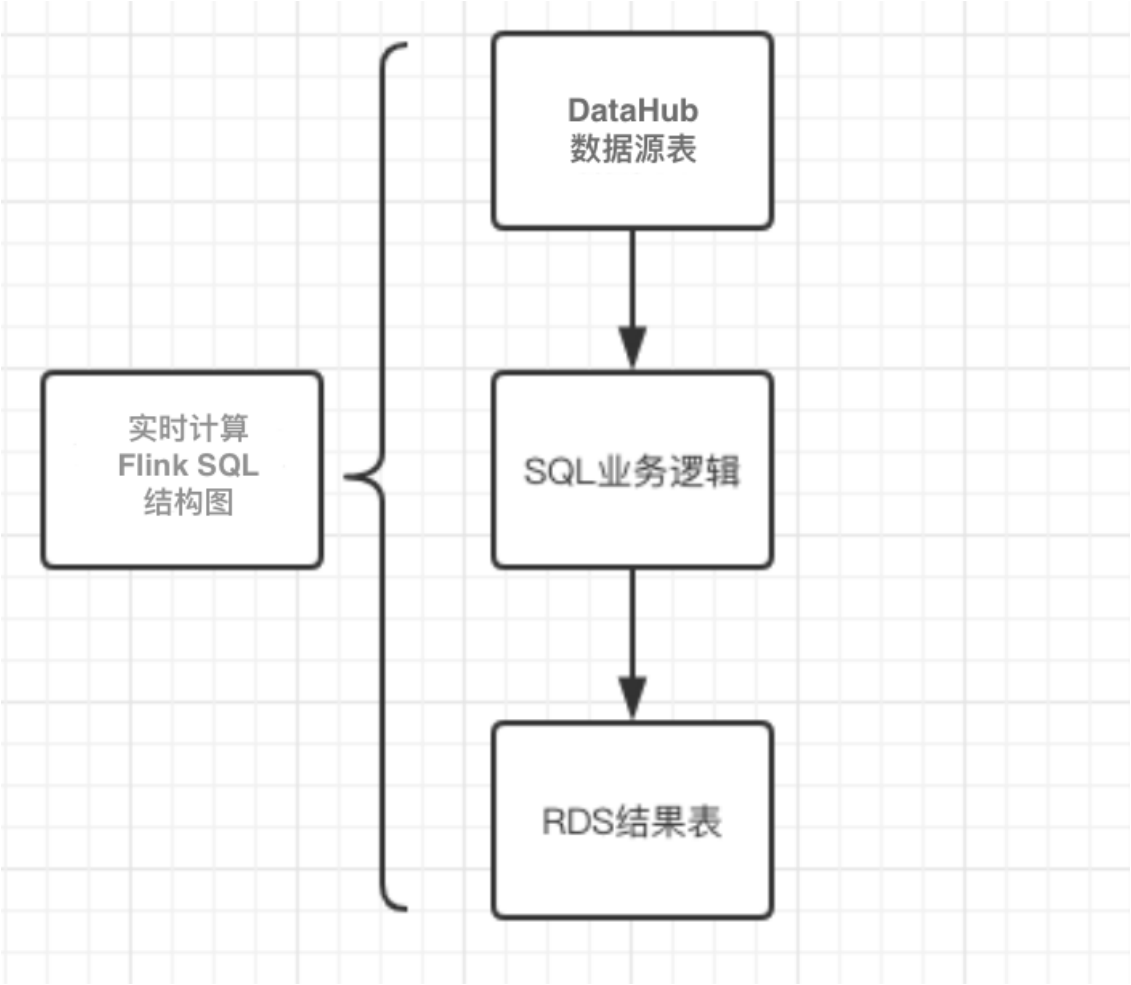
背景信息

以下是天猫工程师利用实时计算实现实时多类目成交总额管理的案例。

多类目成交总额管理案例

> **说明** 为了聚焦核心逻辑，订单数据格式做了大量精简，只保留了与案例有关的属性。

SQL结构图



操作步骤如下：

1. 创建数据总线DataHub源表

   系统订单是实时产生的，源表语句如下。

```
CREATE TABLE source_order (
id VARCHAR,-- 商品ID。
buyer_id VARCHAR, --买家ID。
site VARCHAR,--商品类别。
pay_time VARCHAR,--订单支付时间。
buy_amount DOUBLE, --订单金额。
wap VARCHAR--购买方式。
) WITH (
type='datahub',
endPoint='http://dh-cn-hangzhou.aliyun-inc.com',
project='yourProjectName',--DataHub中Poject的名称。
topic='yourTopicName',--DataHub Poject中Topic的名称。
roleArn='yourRoleArn',--指定角色的roleArn。
batchReadSize='500'
);
```

2. 创建RDS结果表

   RDS结果表语句如下。

```
CREATE TABLE ads_site_block_trd_pay_ri (
id VARCHAR,
site VARCHAR,
data_time VARCHAR,
all_alipay BIGINT,
all_ord_cnt BIGINT,
primary key(id,site,data_time)
) WITH (
type= 'rds',
url = 'yourRDSDatabaseURL',--RDS数据库URL。
tableName = 'yourDatabaseTableName',--RDS数据库中的表名。
userName = 'yourDatabaseUserName',--登录数据库的用户名。
password = 'yourDatabasePassword'--登录数据库的密码。
);
```

3. 创建一个View视图

   根据商品ID、商品类别、买家ID和订单支付时间做分组操作。

```
CREATE VIEW tmp_ads_site_block_trd_pay_ri AS
SELECT
id id,
mod(HASH_CODE(`a`.buyer_id),4096) hash_id,
site site,
date_format(`a`.pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMddHH') data_time,
SUM(cast(buy_amount as bigint)) all_alipay,
count(distinct `a`.buyer_id) all_ord_cnt
FROM
source_order `a`
GROUP BY id , site , mod(HASH_CODE(`a`.buyer_id),4096) , date_format(`a`.pay_time , 'yyyy-MM-dd H
H:mm:ss' , 'yyyyMMddHH') ;
```

4. 数据聚合后插入RDS结果表

   把分组好的数据聚合后插入RDS结果表中，语句如下。

```
INSERT INTO ads_site_block_trd_pay_ri
SELECT
id,
site,
`a`.data_time,
CAST(sum(all_alipay) AS BIGINT) as all_alipay,
CAST(sum(all_ord_cnt) AS BIGINT) as all_ord_cnt
FROM
tmp_ads_site_block_trd_pay_ri `a`
GROUP BY id , site , `a`.data_time ;
```

常见问题

Q：如何解决数据倾斜？

A：假如您的ID数据非常大，根据您的ID进行分组计算可能会造成机器热点从而导致数据倾斜，计算性能会很差。

```
mod(HASH_CODE(`a`.buyer_id),4096) hash_id
```

使用HASH_CODE这个离散函数来分离数据热点，然后使用MOD函数对哈希值进行分组操作。这样做的优点是规范每个节点数据的数量，避免大量数据的堆积导致数据倾斜（4096指的是分组的数量，可以根据数据的大小进行分组操作）。HASH_CODE函数详情请参见HASH_CODE。

Q：您得到的UV量是否准确？

A：直接使用 `count(distinct.buver id)` 会出现计算不准的问题。请使用 `GROUP BY mod(HASH_CODE(a.buyer_id),4096)` ，把相同 `buyer_id` 过滤去重后，再进行 `SUM` 操作，就可以避免这样的错误产生。

```
CREATE VIEW tmp_ads_site_block_trd_pay_ri AS
 SELECT
  id id,
  mod(HASH_CODE(`a`.buyer_id),4096) hash_id,
  site site,
  date_format(`a`.pay_time , 'yyyy-MM-dd HH:mm:ss' , 'yyyyMMddHH') data_time,
  SUM(cast(buy_amount as bigint)) all_qty_cnt,
  count(distinct `a`.buyer_id) all_ord_cnt
   FROM
 source_order `a`
  GROUP BY id , site , mod(HASH_CODE(`a`.buyer_id),4096) , date_format(`a`.pay_time , 'yyyy-MM-dd HH:m
m:ss' , 'yyyyMMddHH') ;
```

DEMO示例以及源代码

根据上文介绍的成交总额管理解决方案，为您创建了一个包含完整链路的DEMO代码，您可以参考示例代码，注册上下游数据，制定您的多类目成交总额管理解决方案。使用示例时，您需要使用DataHub作为源表提供流数据，RDS作为结果表来存储结果数据。

2.IoT行业最佳实践

2.1. IoT解决方案之多维度传感器数据分析

本文通过案例为您介绍如何使用实时计算完成多维度传感器数据分析。

背景说明

在经济全球化的浪潮中，工业制造商面临的竞争日趋激烈。汽车、航空、高新技术、食品与饮料、纺织和制药行业生产商需要进行创新，但更换现有基础设施是一项艰巨的任务。传统系统和设备已使用了几十年，维护费用高昂，但更换系统和设备成本巨大，而且还有生产放缓且质量得不到保障的风险。

同时，安全风险比以往更为严重，流程自动化的需求也更为紧迫。由于配备机械臂、装配线、包装机械等众多活动部件，远程应用必须能够无缝进行设备部署、更新、故障转移以及处理寿命终止事项。因此，制造业需要高可靠性和高可用性的系统，才能保障业务安全可靠地实时运行。

所有新一代系统必须能够捕捉、分析工业设施产生的巨量数据，并及时对这些数据做出响应。为了更好地发展，制造商需要进行优化升级。使用阿里云实时计算系统，搭配使用阿里云IoT套件，可以帮助用户实时分析和诊断工业设备的运行状况，实时检测运行故障，实时预测制品良率。

下面以一个工业制造设备使用实时计算分析大量工业传感器传入数据，实时进行数据清洗和归纳、实时监控设备关键指标、实时将数据清洗并写入在线OLAP系统为例，为您介绍如何使用实时计算完成多维度传感器数据分析。

业务描述

该工业客户拥有1千多台设备，分布在不同城市的多个厂区，每个设备上有10个不同种类传感器，这些传感器，大概每5秒采集并上传一份数据到日志服务（Log/SLS），每个采集点格式如下。

s_id	s_value	s_ts
传感器ID	传感器当前值	发送时间

同时，上述传感器分布在多个设备、多个厂区，用户在RDS上记录传感器、设备、厂区的分布维表，表格如下：

s_id	s_type	device_id	factory_id
传感器ID	传感器监控类型	设备ID	厂区ID

上述信息存放在RDS上，用户希望传感器上传的数据能够和上述数据关联，并将传感器数据按照设备归类每1分钟打平为一张宽表，表格如下：

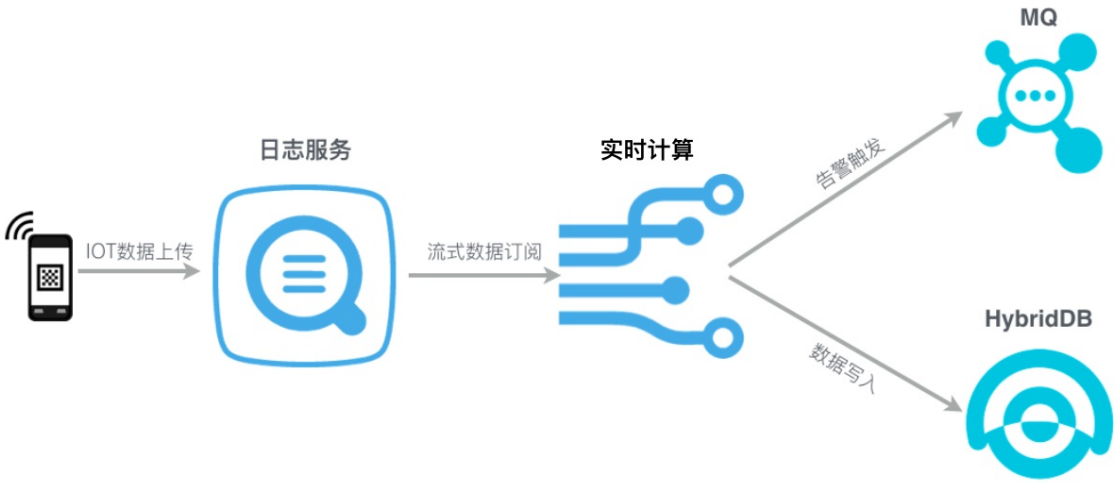
ts	device_id	factory_id	device_temp	device_pres
时间	设备ID	工厂ID	设备温度	设备压力

为了简化不必要的逻辑，我们假定仅有两种类型的监控传感器，即温度和压力，以方便后续的计算，后续计算逻辑如下：

1. 筛选温度大于80的设备，并向下游触发告警。用户选择使用MQ作为消息触发源，也就是实时计算将温度大于80的设备过滤并投递给MQ，触发下游的用户定义的告警系统。
2. 将数据写出到在线OLAP系统中，这里用户选择了PetaData。下游用户开发一整套BI系统对接PetaData

进行多维度展示。

整体架构如下：



要点说明

- 如何打平为一张宽表？

  通常IoT的数据都是一个传感器上传一个维度的数据，这样的数据并不利于后续数据加工分析。使用阿里云实时计算可以将数据按照一定窗口汇聚，并根据传感器不同维度进行数据筛选，打平为一张宽表。

- 为什么要选择MQ作为告警消息源？

  理论上，实时计算可以支持将结果写出到任意系统，但在告警、通知等场景下，实时计算仍然推荐将结果数据投递到MQ之类的消息存储，这样能更好地保证告警准确性。

代码说明

经过传感器上传的数据进入Log，当行数据格式如下。

```
{
  "sid": "t_xxsfdsad",
  "s_value": "85.5",
  "s_ts": "1515228763"
}
```

定义Log源表为s_sensor_data，结构如下。

```
CREATE TABLE s_sensor_data (
  s_id   VARCHAR,
  s_value VARCHAR,
  s_ts   VARCHAR,
  ts     AS CAST(FROM_UNIXTIME(CAST(s_ts AS BIGINT)) AS TIMESTAMP),
   WATERMARK FOR ts AS withOffset(ts, 10000)
) WITH (
  TYPE='sls',
  endPoint ='http://cn-hangzhou-corp.sls.aliyuncs.com',
   accessId ='xxxxxxxxxxx',
   accessKey ='xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
   project ='ali-cloud-streamtest',
   logStore ='stream-test',
);
```

定义传感器和设备关联RDS维表为d_sensor_device_data，结构如下。

```
CREATE TABLE d_sensor_device_data (
  s_id   VARCHAR,
  s_type   VARCHAR,
  device_id BIGINT,
  factory_id BIGINT,
  PERIOD FOR SYSTEM_TIME,
  PRIMARY KEY(s_id)
) WITH (
  TYPE='RDS',
   url='',
   tableName='test4',
   userName='test',
   password='XXXXXX'
);
```

定义触发告警逻辑MQ表为r_monitor_data，结构如下。

```
CREATE TABLE r_monitor_data (
  ts   VARCHAR,
  device_id   BIGINT,
  factory_id   BIGINT,
  device_TEMP   DOUBLE,
  device_PRES DOUBLE
) WITH (
  TYPE='MQ'
);
```

定义存储结果数据的HybridDB表定义为r_device_data，结构如下。

```
CREATE TABLE r_device_data (
  ts   VARCHAR,
  device_id BIGINT,
  factory_id BIGINT,
  device_temp   DOUBLE,
  device_pres DOUBLE,
  PRIMARY KEY(ts, device_id)
) WITH (
  TYPE='HybridDB'
);
```

先考虑将传感器数据按分钟级别进行汇总，打平为一个宽表。为了方便后续代码维护，请使用结构化代码和View。

```
--先获取每个传感器对应的设备、厂区。
CREATE VIEW v_sensor_device_data
AS
SELECT
  s.ts,
  s.s_id,
  s.s_value,
  d.s_type,
  d.device_id,
  d.factory_id
FROM
  s_sensor_data s
JOIN
  d_sensor_device_data FOR SYSTEM_TIME AS OF PROCTIME() as d
ON
  s.s_id = d.s_id;
--打平为一张宽表。
CREATE VIEW v_device_data
AS
SELECT
  --使用滚窗的起始时间作为该条记录的时间。
  CAST(TUMBLE_START(v.ts, INTERVAL '1' MINUTE) AS VARCHAR) as ts,
  v.device_id,
  v.factory_id,
  CAST(SUM(IF(v.s_type = 'TEMP', v.s_value, 0)) AS DOUBLE)/CAST(SUM(IF(v.s_type = 'TEMP', 1, 0)) AS DOUB
LE) device_temp, --这里用于计算这一分钟的温度平均值
  CAST(SUM(IF(v.s_type = 'PRES', v.s_value, 0)) AS DOUBLE)/CAST(SUM(IF(v.s_type = 'PRES', 1, 0)) AS DOUBL
E) device_pres --这里用于计算这一分钟的压力平均值
FROM
  v_sensor_device_data v
GROUP BY
  TUMBLE(v.ts, INTERVAL '1' MINUTE), v.device_id, v.factory_id;
```

上述是计算的核心逻辑，统计一分钟内温度和压力的平均值，作为这一分钟的温度值、压力值。由于使用的是Tumbling Window，也就意味着在每分钟结束时产出一份数据。接下来将数据过滤写出到MQ和HybridDB。

Demo示例以及源代码

根据上文介绍的IoT解决方案，为您创建了一个包含完整链路的Demo示例。

```
--过滤温度大于80摄氏度的传感器，并写出到MQ触发告警。
INSERT INTO r_monitor_data
SELECT
  ts,
  device_id,
  factory_id,
  device_temp,
  device_pres
FROM
  v_device_data
WHERE
  device_temp > 80.0;
--将数据写出到HybridDB，用于后续的分析。
INSERT INTO r_device_data
SELECT
  ts,
  device_id,
  factory_id,
  device_temp,
  device_pres
FROM
  v_device_data;
```

您可以参考Demo代码，注册上下游数据，制定自己的IoT解决方案。使用示例时，上下游存储请注意以下几点：

- 选择Logtail采集ECS上的文本信息作为源表。
- RDS作为维表。
- MQ和HybridDB作为结果表。

3.视频直播行业最佳实践

3.1. 视频直播解决方案之视频核心指标监控

本文通过案例为您介绍如何使用实时计算完成视频核心指标监控。

背景信息

随着互联网络技术的发展，网络直播受到越来越多人的关注，特别是视频直播生态链。通过网络信号，我们在线收看体育赛事、重大活动和新闻等，真正地体验直播的乐趣。

在体验为王的时代，任何体验不佳均会导致客户群体大规模的流失。对于网站直播平台而言，需要重点关注以下几点：

- 用户（主播和粉丝）的使用体验，重点关注系统指标包括音视频卡顿率、延迟率、丢包等情况。
- 直播平台具有时效性，平台方更需要实时了解系统周边问题，并先于用户发现并定位问题和故障。
- 平台运营方还应该及时跟踪和了解网站整体的客户运营情况，以及视频爆款等情况。

下面我们以某个视频直播平台方为例，讲解如何使用实时计算监控系统稳定性和平台运营情况。

业务详情

为了最大程度活跃用户社群，覆盖更多场景直播频道，最终实现平台方盈利，业务通常是这样实现的：

- 同一个视频直播网站有多个主播。
- 每个主播向一个频道内的用户进行广播。
- 用户可以看到当前频道内的主播视频，并听到其声音。
- 主播可以与频道内的多个用户进行私聊。

业务流程

1. 主播和粉丝各自持有直播App软件，该App软件每10秒向服务器打点。
2. 服务器接收到打点信息会输出到本地磁盘，阿里云日志服务采集打点数据。
3. 实时计算订阅该日志服务。
4. 实时计算客户端视频播放情况。

业务目标

针对客户端App的监控，获取以下指标：

- 房间故障，包括卡顿、丢帧、音视频不同步等。
- 分地域统计数据端到端延迟平均情况。
- 统计实时整体卡顿率（出现卡顿的在线用户数/在线总用户数*100%，通过此指标可以衡量当前卡顿影响的人群范围）。

- 统计人均卡顿次数（在线卡顿总次数/在线用户数，通过此指标可以从卡顿频次上衡量整体的卡顿严重程度）。

最终我们希望将上述数据实时计算并写入阿里云关系型数据库RDS（Relational Database Service），并提供在线数据报表展示、大屏展示、甚至部分监控告警。

数据格式

客户端App向服务器打点日志格式如下。

字段	含义
ip	用户端IP
agent	端类型
roomid	房间号
userid	用户ID
abytes	音频码率
afcnt	音频帧数
adrop	音频丢帧数量
afts	音频时间戳
alat	音频帧端到端延迟
vbytes	视频码率
vfcnt	视频帧数
vdrop	视频丢帧数量
vfts	视频时间戳
vlat	视频帧端到端延迟
ublock	上行卡顿次数
dblock	下行卡顿次数
timestamp	打点时间戳
region	地域

日志服务内部是半结构化存储，上述数据将展现出来格式如下。

```
{
  "ip": "ip",
  "agent": "agent",
  "roomid": "123456789",
  "userid": "123456789",
  "abytes": "123456",
  "afcnt": "34",
  "adrop": "3",
  "afts": "1515922566",
  "alat": "123",
  "vbytes": "123456",
  "vfcnt": "34",
  "vdrop": "4",
  "vfts": "1515922566",
  "vlat": "123",
  "ublock": "1",
  "dblock": "2",
  "timestamp": "15151922566",
  "region": "hangzhou"
}
```

SQL编写

- 数据清洗

  首先在实时计算中声明源表。

```
CREATE TABLE app_heartbeat_stream_source (
  ip VARCHAR,
  agent VARCHAR,
  roomid VARCHAR,
  userid VARCHAR,
  abytes VARCHAR,
  afcnt VARCHAR,
  adrop VARCHAR,
  afts VARCHAR,
  alat VARCHAR,
  vbytes VARCHAR,
  vfcnt VARCHAR,
  vdrop VARCHAR,
  vfts VARCHAR,
  vlat VARCHAR,
  ublock VARCHAR,
  dblock VARCHAR,
  `timestamp` VARCHAR,
  app_ts AS TO_TIMESTAMP(CAST(`timestamp` AS BIGINT)), --定义生成WATERMARK的字段。
  WATERMARK FOR app_ts AS withOffset(app_ts, 10000) --WATERMARK比数据时间线性增加10秒。
) WITH (
  type ='sls',
  endPoint ='http://cn-hangzhou-corp.sls.aliyuncs.com',
  accessId ='xxxxxxxxxxx',
  accessKey ='xxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  project ='xxxx',
  logStore ='app_heartbeat_stream_source',
);
```

源端业务为考虑业务方便，将所有的数据均处理为VARCHAR类型。为了方便后续处理，我们将上述源表进行数据清洗，清洗目标为：

i. 格式转换。将部分VARCHAR类型转为BIGINT。

ii. 业务补充。例如，填写地域相关信息。

```
CREATE VIEW view_app_heartbeat_stream AS
SELECT
  ip,
  agent,
  CAST(roomid AS BIGINT),
  CAST(userid AS BIGINT),
  CAST(abytes AS BIGINT),
  CAST(afcnt AS BIGINT),
  CAST(adrop AS BIGINT),
  CAST(afts AS BIGINT),
  CAST(alat AS BIGINT),
  CAST(vbytes AS BIGINT),
  CAST(vfcnt AS BIGINT),
  CAST(vdrop AS BIGINT),
  CAST(vfts AS BIGINT),
  CAST(vlat AS BIGINT),
  CAST(ublock AS BIGINT),
  CAST(dblock AS BIGINT),
  app_ts,
  region
FROM
  app_heartbeat_stream_source;
```

- 房间故障统计

  统计房间故障，故障包括卡顿、丢帧、音视频不同步信息，使用10分钟一个窗口进行统计。

```
CREATE VIEW room_error_statistics_10min AS
SELECT
  CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
  roomid,
  SUM(ublock) as ublock, --统计10分钟内上行卡顿次数。
  SUM(dblock) as dblock, --统计10分钟内下行卡顿次数。
  SUM(adrop) as adrop, --统计10分钟内音频丢包次数。
  SUM(vdrop) as vdrop, --统计10分钟内视频丢包次数。
  SUM(alat) as alat, --统计10分钟内音频延迟。
  SUM(vlat) as vlat, --统计10分钟内音频延迟。
FROM
  view_app_heartbeat_stream
GROUP BY
  TUMBLE(app_ts, INTERVAL '10' MINUTE), roomid;
```

- 分地域统计延迟情况

  分地域统计数据端到端延迟平均情况，每10分钟统计音频、视频平均延迟情况。

```
CREATE VIEW region_lat_statistics_10min AS
SELECT
  CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
  region,
  SUM(alat)/COUNT(alat) as alat,
  SUM(vlat)/COUNT(vlat) as vlat,
FROM
  view_app_heartbeat_stream
GROUP BY
  TUMBLE(app_ts, INTERVAL '10' MINUTE), region;
```

- 实时整体卡顿率

  统计实时整体卡顿率，即出现卡顿的在线用户数/在线总用户数*100%，通过此指标可以衡量当前卡顿影响的人群范围。

```
CREATE VIEW block_total_statistics_10min AS
SELECT
  CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
  SUM(IF(ublock <> 0 OR dblock <> 0, 1, 0)) / CAST(COUNT(DISTINCT userid) AS DOUBLE) as block_rate, --C
OUNT(DISTINCT) 需要在1.4.4版本以上Blink才能支持
FROM
  view_app_heartbeat_stream
GROUP BY
  TUMBLE(app_ts, INTERVAL '10' MINUTE);
```

- 统计人均卡顿次数

  统计人均卡顿次数，即在线卡顿总次数/在线用户数，通过此指标可以从卡顿频次上衡量整体的卡顿严重程度。

```
CREATE VIEW block_peruser_statistics_10min AS
SELECT
  CAST(TUMBLE_START(app_ts, INTERVAL '10' MINUTE) as VARCHAR) as app_ts,
  SUM(ublock+dblock) / CAST(COUNT(DISTINCT userid) AS DOUBLE) as block_peruser, --COUNT(DISTINCT
) 需要在1.4.4版本以上Blink才能支持
FROM
  view_app_heartbeat_stream
GROUP BY
  TUMBLE(app_ts, INTERVAL '10' MINUTE);
```

Demo示例以及源代码

根据上文介绍的视频直播解决方案之视频核心指标监控，阿里云为您创建了一个包含完整链路的Demo示例，参考示例代码，您可注册上下游数据，制定自己的视频核心指标监控的解决方案。单击下载示例。使用示例时，您需要选择CSV文件上传至DataHub作为源表，RDS作为结果表来存储结果数据。

3.2. 视频直播解决方案之直播数字化运营

本文通过案例为您介绍如何使用实时计算完成直播数字化运营的视频直播解决方案。

直播数字化运营业务

本文主要关注的是直播数据化运营业务。为您介绍使用实时计算完成对当前直播间运营情况实时的监控，包括热门视频、用户走势等等。

解决方案

- 业务目标
  - 全站观看直播总人数以及走势。
  - 房间直播总人数以及走势。
  - 热门直播房间及主播Top10，分类目主播Top10。
- 数据格式

  参考上面的数据格式，使用App打点日志作为原始数据进行统计。

  客户端App向服务器打点日志格式如下。

字段	含义
ip	用户端IP
agent	端类型
roomid	房间号
userid	用户ID
abytes	音频码率
afcnt	音频帧数
adrop	音频丢帧数量
afts	音频时间戳
alat	音频帧端到端延迟
vbytes	视频码率
vfcnt	视频帧数
vdrop	视频丢帧数量
vfts	视频时间戳
vlat	视频帧端到端延迟
ublock	上行卡顿次数
dblock	下行卡顿次数
timestamp	打点时间戳
region	地区

日志服务内部是半结构化存储，上述数据将展现出来格式如下。

```
{
  "ip": "ip",
  "agent": "agent",
  "roomid": "123456789",
  "userid": "123456789",
  "abytes": "123456",
  "afcnt": "34",
  "adrop": "3",
  "afts": "1515922566",
  "alat": "123",
  "vbytes": "123456",
  "vfcnt": "34",
  "vdrop": "4",
  "vfts": "1515922566",
  "vlat": "123",
  "ublock": "1",
  "dblock": "2",
    "timestamp": "1515922566",
  "region": "hangzhou"
}
```

- SQL编写
  - 全站总人数及走势

    使用每分钟一个窗口统计全站观看人数走势，走势最近的1分钟计算结果就是当前总人数。

```
CREATE VIEW view_app_total_visit_1min AS
SELECT
  CAST(TUMBLE_START(app_ts, INTERVAL '1' MINUTE) as VARCHAR) as app_ts,
  COUNT(DISTINCT userid) as app_total_user_cnt
FROM
  view_app_heartbeat_stream
GROUP BY
  TUMBLE(app_ts, INTERVAL '1' MINUTE);
```

  - 房间直播总人数以及走势

    同样，使用每分钟一个窗口统计房间内观看人数走势，走势最近的1分钟计算结果就是当前房间人数。

```
CREATE VIEW view_app_room_visit_1min AS
SELECT
  CAST(TUMBLE_START(app_ts, INTERVAL '1' MINUTE) as VARCHAR) as app_ts,
  roomid as room_id,
  COUNT(DISTINCT userid) as app_room_user_cnt
FROM
  view_app_heartbeat_stream
GROUP BY
  TUMBLE(app_ts, INTERVAL '1' MINUTE), roomid;
```

- 热门直播房间排名

  统计热门直播实际上为首页推荐用户使用，把热门的房间放在首页推广能够获取更多流量和单击。

```
CREATE VIEW view_app_room_visit_top10 AS
SELECT
 app_ts,
 room_id,
 app_room_user_cnt,
 rangking
FROM
(
  SELECT
    app_ts,
    room_id,
    app_room_user_cnt,
    ROW_NUMBER() OVER (PARTITION BY 1 ORDER BY app_room_user_cnt desc) AS ranking
  FROM
    view_app_room_visit_1min
) WHERE ranking <= 10;
```

- 分类目统计热门直播房间排行

  为了最大程度活跃用户社群，覆盖更多场景直播频道，最终实现平台方盈利，通常同一个视频直播网站会开设大量的不同频道，以迎合各类不同人群的观看需求。例如，淘宝直播就会有美妆、男装、汽车、健身等多个不同类目。

  类目和房间关系表是在RDS存储的映射表。

```
CREATE TABLE dim_category_room (
  id   BIGINT,
  category_id BIGINT,
  category_name VARCHAR,
  room_id   BIGINT
   PRIMARY KEY (room_id),
   PERIOD FOR SYSTEM_TIME --定义为维表。
 ) WITH (
   type= 'rds',
   url = 'xxxx',--您的数据库url。
   tableName = 'xxx',--您的表名。
   userName = 'xxx',--您的用户名。
   password = 'xxx'--您的密码。
 );
```

  针对不同的房间，关联类目表，并分类统计各自的排名情况。

```
CREATE VIEW view_app_room_visit_1min AS
SELECT
  CAST(TUMBLE_START(app_ts, INTERVAL '1' MINUTE) as VARCHAR) as app_ts,
  roomid as room_id,
  COUNT(DISTINCT userid) as app_room_user_cnt
FROM
  view_app_heartbeat_stream
GROUP BY
  TUMBLE(app_ts, INTERVAL '1' MINUTE), roomid;
--关联类目表。
CREATE VIEW view_app_category_visit_1min AS
SELECT
  r.app_ts,
  r.room_id,
  d.category_id,
  d.category_name,
  r.app_room_user_cnt
FROM
  view_app_room_visit_1min r
JOIN
  dim_category_room d
ON
  r.room_id = d.room_id;
--计算不同类目下房间号排名。
CREATE VIEW view_app_category_visit_top10 AS
SELECT
 app_ts,
 category_id,
 category_name,
 app_room_user_cnt,
 rangking
FROM
(
  SELECT
    app_ts,
    room_id,
    category_id,
    category_name,
    app_room_user_cnt,
    ROW_NUMBER() OVER (PARTITION BY category_id ORDER BY app_room_user_cnt desc) AS ranking
  FROM
    view_app_category_visit_1min
) WHERE ranking <= 10;
```

Demo示例以及源代码

阿里云为您创建了一个包含完整链路的Demo示例，您可参考示例代码，注册上下游数据，制定自己的直播数字化运营的解决方案。在使用示例时，上下游存储请注意以下几点：

- 选择CSV文件上传至DataHub作为源表。
- RDS作为维表。
- RDS作为结果表。