本页不打印

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 开发/优化者 | 时间 | 审核人 | 开发类型（新开发/优化） | 更新说明 |
| 黄巧玲 | 2020.07.01 |  | 新开发 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 课程编码 | 适用产品 | 产品版本 | 课程版本ISSUE |
|  | DWS |  | 1.0 |

修订记录

数据仓库建模与数据分析

实验手册



华为技术有限公司

目录

[1 数据仓库建模与数据分析 2](#_Toc44517965)

[1.1 实验介绍 2](#_Toc44517966)

[1.1.1 关于本实验 2](#_Toc44517967)

[1.1.2 实验目的 2](#_Toc44517968)

[1.1.3 实验背景 2](#_Toc44517969)

[1.1.4 实验环境说明 2](#_Toc44517970)

[1.2 实验任务配置 3](#_Toc44517971)

[1.2.1 数据仓库建模 3](#_Toc44517972)

[1.2.2 数据库对象创建 13](#_Toc44517973)

[1.2.3 数据转换 26](#_Toc44517974)

[1.2.4 数据分析 49](#_Toc44517975)

[1.3 实验小结 54](#_Toc44517976)

# 数据仓库建模与数据分析

## 实验介绍

### 关于本实验

数据仓库是为了便于多维分析和多角度展现而将数据按特定的模式进行存储所建立起来的关系型数据库，它的数据基于OLTP源系统。

数据仓库能够对多个异构的数据源进行有效集成，集成后按照主题进行重组，存放大量历史数据，用于支持决策，面向分析型数据处理，它不同于企业现有的面向事务型数据库。

对于数据仓库建模，不同的架构会有不同的建模流程和方式，但都会涉及到维度建模，维度建模以商业用户可理解的方式发布数据，并能够提供更高效的查询性能。

### 实验目的

理解数据仓库建模的步骤。

理解数据从源事务系统到目标系统的转换过程。

掌握复杂查询的基本能力。

### 实验背景

在本次litemall项目中，用户的交易数据短期内存放在华为云数据库RDS中，再定期导入到GaussDB(DWS)数据仓库集群中。通过维度建模来构建数据仓库数据模型，利用Kettle工具实现数据的迁移并在GaussDB(DWS)中完成转换，最后利用历史数据构建复杂查询来对litemall商城进行分析。

### 实验环境说明

本实验基于华为云DWS开展。数据仓库服务（Data Warehouse Service，简称DWS）是一种基于公有云基础架构和平台的在线数据处理数据库，提供即开即用、可扩展且完全托管的分析型数据库服务。DWS兼容标准ANSI SQL 99和SQL 2003，同时兼容PostgreSQL/Oracle数据库生态，为各行业PB级海量大数据分析提供有竞争力的解决方案。DWS可广泛应用于金融、车联网、政企、电商、能源、电信等多个领域，已连续两年入选Gartner发布的数据管理解决方案魔力象限，相比传统数据仓库，性价比提升数倍，具备大规模扩展能力和企业级可靠性。

关于本关卡，实验环境说明如下：

1. 分组情况下，一组公用一套DWS集群，因此需要由其中一位学员（例如组长）完成集群的创建工作，具体创建过程可以参考《鲲鹏应用数据分析与管理实战 - 随堂练习手册》-创建DWS集群部分；

2. 使用Data Studio连接数据库时，首次请先以系统管理员的身份连接postgres数据库，然后创建新的数据库，新的数据库名建议设置为个人姓名简称-demo，例如：create database hql\_demo; 后续连接时，请连接个人创建的数据库，即Data Studio连接界面*数据库*处填写此处创建的库名。具体连接过程参考《鲲鹏应用数据分析与管理实战 - 随堂练习手册》。

## 实验任务配置

### 数据仓库建模

说明：

**1. 本实验中涉及到的DWS服务请勿自行购买，会统一提供。**

Kimball的维度模型设计采用四步方法：选择业务过程、声明粒度、确定维度、确定事实。

开始讨论维度建模设计工作前，必须考虑正确的人选，包括业务代表、业务数据管理人员等等。他们的加入与合作必然会增加最终模型解决用户需求的可能性以及数据管理的合理性。

维度模型的构建是个具有高度动态性且需要迭代产生的过程。最初的准备过程完成后，设计工作将开始处理从总线架构获取的图形化模型，确定设计范围并澄清所提出的事实表及相关维度表的粒度。设计小组将开展针对维度表属性、领域值、来源、关系、数据质量关注点和转换等方面的工作。确定维度后，将建模事实表。建模过程的最后阶段是与感兴趣的伙伴，特别是业务代表们一起对模型进行评审和验证工作，主要目标是建立满足用户需求的模型，检验加载到模型中的数据的可用性，为ETL小组提供最初的源到目标的映射。

维度模型通过一系列设计会议展开，每一次会议将产生更详细、更健壮的按照业务需求反复测试过的设计结果。当模型清楚地满足用户需求后，结束建模过程。通常需要三四周时间完成一次业务过程维度模型的设计，当然需要的时间会随着小组的经验、详细业务需求的可用性、涉及的业务代表或授权负责管理组织数据的人员、数据源的复杂程度、利用现存一致性维度的能力等的差异而存在较大的差异。

为简化整体地操作，本次实验主要聚焦于订单处理相关流程，其他内容可自行研究。

另外关于Litemall项目，说明如下：

1. 未生成优惠券、团购、积分减免等相关数据，因此在建模过程中可不用考虑。

2. 下单人也是收货人，一个用户只有一个地址。

3. 若客户申请退款，则全单退还。

选择业务过程

对业务的整个生命周期进行分析，明确关键业务步骤，从而选择与需求相关的业务过程。业务过程通常使用“行为动词”来表示业务执行的活动。

针对本次Litemall案例，对整体业务进行分析，我们希望能够更好地理解Litemall商城的经营情况，包括商城GMV、营收，商品销量排行，区域销量排行，客户购买行为等等。因此我们重点关注订单系统。

在订单处理上，完整的业务逻辑包含：客户下单🡪生成订单🡪客户付款🡪订单发货🡪物流配送🡪客户收货🡪客户申请退货🡪管理员操作🡪客户退货🡪退款完成。当然也会存在不同的场景，在本实验中，主要以正向订单为例开展。

声明粒度

声明粒度意味着精确定义某个事实表的每一行表示什么。 粒度传递的是与事实表度量有关的细节级别。它回答“如何描述事实表中每个行的内容？”这一问题。粒度由获取业务过程事件的操作型系统的物理实现确定。

针对本次Litemall案例，我们希望通过原子数据来进行不同层面不同角度的分析，希望能够保留最底层最细节的数据来对业务情况进行洞察。最细粒度的数据为订单中某个商品下某种货品的交易信息。

确定维度

详细的粒度说明确定了事实表的主要维度。然后可以将更多维度增加到事实表上，只要这些额外的维度自然地承担主维度合并的某个值。如果附加的维度会产生与粒度不符的其他事实行，则取消该维度或重新考虑粒度声明。

针对本次Litemall案例，主要的维度表包括：日期表、时间表、商品表、用户表、用户地址表。具体字段如下方表格所示。

说明：前期已经说明，未产生优惠券、团购、积分等数据，另外，因为后续分析重点不在商品性质上，为简化整体模型，相关数据并不考虑在内，而在实际过程中，这些应该体现在维度表中。

日期维度

| **Column** | **Datatype** | **Primary Key** | **备注** |
| --- | --- | --- | --- |
| date\_key | integer | Y | 日期键 |
| **full\_date** | **date** |  | 日期 |
| day\_of\_week | tinyint |  | 周中第几天 |
| day\_num\_in\_month | tinyint |  | 月中第几天 |
| day\_num\_overall | smallint |  | 整体中第几“天” |
| day\_name | varchar(9) |  | 星期几 |
| day\_abbrev | char(3) |  | 星期几缩写 |
| weekday\_flag | varchar(10) |  | 工作日标记 |
| week\_num\_in\_year | tinyint |  | 一年中的第几周 |
| week\_num\_overall | smallint |  | 整体中所在“周” |
| **week\_begin\_date** | **date** |  | 当周开始日期 |
| week\_begin\_date\_key | integer |  | 当周开始日期键 |
| month | tinyint |  | 月份 |
| month\_num\_overall | smallint |  | 整体中所在“月份” |
| month\_name | varchar(9) |  | 月份全称 |
| month\_abbrev | char(3) |  | 月份缩写 |
| quarter | tinyint |  | 季度 |
| year | smallint |  | 年 |
| yearmo | int |  | 年月 |
| fiscal\_month | tinyint |  | 财政月份 |
| fiscal\_quarter | tinyint |  | 财政季度 |
| fiscal\_year | smallint |  | 财政年份 |
| last\_day\_in\_month\_flag | varchar(15) |  | 月末标记 |
| same\_day\_year\_ago\_date | date |  | 一年前的当前日 |

时间维度

| **Column** | **Datatype** | **Primary Key** | **备注** |
| --- | --- | --- | --- |
| time\_key | integer | Y | 时间键 |
| hour | integer |  | 时 |
| minute | integer |  | 分 |
| second | integer |  | 秒 |
| t\_am\_pm | char(2) |  | 上下午 |

商品维度

| **Column** | **Datatype** | **Primary Key** | **备注** |
| --- | --- | --- | --- |
| goods\_key | integer | Y | 商品键 |
| goods\_id | integer |  | 商品id |
| goods\_sn | varchar(63) |  | 商品编号 |
| goods\_name | varchar(127) |  | 商品名词 |
| category\_id | integer |  | 商品所属类目id |
| category\_name | varchar(63) |  | 类目名称 |
| category\_keywords | varchar(1023) |  | 类目关键字 |
| category\_desc | varchar(255) |  | 类目广告语介绍 |
| category\_pid | integer |  | 父类目id |
| brand\_id | integer |  | 品牌商id |
| gallery | varchar(1023) |  | 商品宣传图片列表 |
| keywords | varchar(255) |  | 商品关键字 |
| brief | varchar(1023) |  | 商品简介 |
| is\_on\_sale | tinyint |  | 是否上架 |
| sort\_order | smallint |  | 排列序号 |
| pic\_url | varchar(1023) |  | 商品页面商品图片 |
| share\_url | varchar(1023) |  | 商品分享朋友圈图片 |
| is\_new | tinyint |  | 是否新品首发 |
| is\_hot | tinyint |  | 是否人气推荐 |
| unit | varchar(31) |  | 商品单位，例如件、盒 |
| counter\_price | decimal(10,2) |  | 专柜价格 |
| retail\_price | decimal(10,2) |  | 零售价格 |
| detail | text |  | 商品详细介绍 |
| product\_id | integer |  | 商品货品id |
| specifications | varchar(1023) |  | 商品规格值列表 |
| **gp\_add\_date** | **integer** |  | **商品货品创建日期键** |
| **gp\_add\_time** | **integer** |  | **商品货品创建时间键** |
| **gp\_update\_date** | **integer** |  | **商品货品更新日期键** |
| **gp\_update\_time** | **integer** |  | **商品货品更新时间键** |
| gp\_deleted | tinyint |  | 商品货品逻辑删除 |
| category\_icon\_url | varchar(1023) |  | 类目图标 |
| category\_pic\_url | varchar(1023) |  | 类目图片 |
| category\_level | varchar(255) |  | 类目级别 |
| category\_sort\_order | tinyint |  | 类目排列序号 |
| category\_add\_date | integer |  | 类目创建日期 |
| category\_add\_time | integer |  | 类目创建时间 |
| cateogy\_update\_date | integer |  | 类目更新日期 |
| category\_update\_time | integer |  | 类目更新时间 |
| category\_deleted | tinyint |  | 类目逻辑删除 |
| **goods\_add\_date** | **integer** |  | 商品添加日期 |
| **goods\_add\_time** | **integer** |  | 商品添加时间 |
| **goods\_update\_date** | **integer** |  | 商品更新日期 |
| **goods\_update\_time** | **integer** |  | 商品更新时间 |
| goods**\_**deleted | tinyint |  | 商品逻辑删除 |

用户维度

| **Column** | **Datatype** | **Primary Key** | **Primary Key** |
| --- | --- | --- | --- |
| user\_key | integer | Y | 用户键 |
| user\_id | integer |  | 用户id |
| username | varchar(63) |  | 用户名称 |
| password | varchar(63) |  | 用户密码 |
| gender | tinyint |  | 性别 |
| **birthday** | **integer** |  | 生日 |
| **last\_login\_date** | **integer** |  | 最近一次登录日期键 |
| **last\_login\_time** | **integer** |  | 最近一次登录时间键 |
| last\_login\_ip | varchar(63) |  | 最近一次登录ip地址 |
| user\_level | tinyint |  | 用户级别 |
| nickname | varchar(63) |  | 用户昵称或网络名称 |
| mobile | varchar(20) |  | 用户手机号码 |
| avatar | varchar(255) |  | 用户头像图片 |
| weixin\_openid | varchar(63) |  | 微信登录openid |
| session\_key | varchar(100) |  | 微信登录会话key |
| status | tinyint |  | 用户状态 |
| **add\_date** | **integer** |  | 用户创建日期键 |
| **add\_time** | **integer** |  | 用户创建时间键 |
| **update\_date** | **integer** |  | 用户更新日期键 |
| **update\_time** | **integer** |  | 用户更新时间键 |
| deleted | tinyint |  | 用户逻辑删除 |

地址维度

| **Column** | **Datatype** | **Primary Key** | **备注** |
| --- | --- | --- | --- |
| address\_key | integer | Y | 地址键 |
| address\_id | integer |  | 地址id |
| province | varchar(63) |  | 行政区域表的省id |
| city | varchar(63) |  | 行政区域表的市id |
| county | varchar(63) |  | 行政区域表的区县id |
| address\_detail | varchar(127) |  | 详细收货地址 |
| area\_code | char(6) |  | 地区编码 |
| postal\_code | char(6) |  | 邮政编码 |
| is\_default | tinyint |  | 是否默认地址 |
| **add\_date** | **integer** |  | 地址创建日期键 |
| **add\_time** | **integer** |  | 地址创建时间键 |
| **update\_date** | **integer** |  | 地址更新日期键 |
| **update\_time** | **integer** |  | 地址更新时间键 |
| deleted | tinyint |  | 地址逻辑删除 |

确定事实

事实必须与粒度吻合。在声明粒度环节，我们已经确定采用细节的、原子粒度的数据。试想一下下单环节，一个订单可以包含多个商品，结合实际的操作型数据库中的数据，在此我们将订单相关的数据组织为订单表和订单商品表两张事实表。另外，我们需要一张库存表来存储商品的实际库存量数据。由此事实表包含：订单表、订单商品表和库存表。具体的字段如下方表格所示。

订单表

| **Column** | **Datatype** | **Primary Key** | **备注** |
| --- | --- | --- | --- |
| order\_id | integer | Y | 订单id |
| order\_sn | varchar(63) |  | 订单编号 |
| order\_status | smallint |  | 订单状态 |
| **user\_key** | **integer** |  | 用户键 |
| **address\_key** | **integer** |  | 地址键 |
| goods\_price | decimal(10,2) |  | 商品总费用 |
| freight\_price | decimal(10,2) |  | 配送费用 |
| coupon\_price | decimal(10,2) |  | 优惠券减免 |
| integral\_price | decimal(10,2) |  | 用户积分减免 |
| groupon\_price | decimal(10,2) |  | 团购优惠价减免 |
| order\_price | decimal(10,2) |  | 订单费用 |
| actual\_price | decimal(10,2) |  | 实付费用 |
| pay\_id | varchar(63) |  | 微信付款编号 |
| **pay\_date** | **integer** |  | 微信付款日期键 |
| **pay\_time** | **integer** |  | 微信付款时间键 |
| ship\_sn | varchar(63) |  | 发货编号 |
| ship\_channel | varchar(63) |  | 发货快递公司 |
| **ship\_date** | **integer** |  | 发货开始日期键 |
| **ship\_time** | **integer** |  | 发货开始时间键 |
| **confirm\_date** | **integer** |  | 用户确认收货日期键 |
| **confirm\_time** | **integer** |  | 用户确认收货时间键 |
| message | varchar(512) |  | 用户订单留言 |
| comments | smallint |  | 待评价订单商品数量 |
| aftersale\_status | smallint |  | 售后状态 |
| refund\_amount | decimal(10,2) |  | 实际退款金额 |
| refund\_type | varchar(63) |  | 退款方式 |
| refund\_content | varchar(127) |  | 退款备注 |
| **refund\_date** | **integer** |  | 退款日期键 |
| **refund\_time** | **integer** |  | 退款时间键 |
| **end\_date** | **integer** |  | 订单关闭日期键 |
| **end\_time** | **integer** |  | 订单关闭时间键 |
| **add\_date** | **integer** |  | 订单创建日期键 |
| **add\_time** | **integer** |  | 订单创建时间键 |
| **update\_date** | **integer** |  | 订单更新日期键 |
| **update\_time** | **integer** |  | 订单更新时间键 |
| deleted | tinyinit |  | 订单逻辑删除 |

订单商品表

| **Column** | **Datatype** | **Primary Key** | **备注** |
| --- | --- | --- | --- |
| order\_id | integer | Y | 订单id |
| goods\_key | integer | Y | 商品键 |
| number | integer |  | 商品货品的购买数量 |
| price | decimal(10,2) |  | 商品货品的售价 |
| comment\_value | integer |  | 订单商品评论 |
| **add\_date** | **integer** |  | 创建日期键 |
| **add\_time** | **initeger** |  | 创建时间键 |
| **update\_date** | **integer** |  | 更新日期键 |
| **update\_time** | **integer** |  | 更新时间键 |
| deleted | tinyint |  | 逻辑删除 |

库存表

| **Column** | **Datatype** | **Primary Key** | **备注** |
| --- | --- | --- | --- |
| date\_key | integer | Y | 日期键 |
| goods\_key | integer | Y | 商品键 |
| number | integer |  | 商品货品数量 |

### 数据库对象创建

**注意：**

**1. 每个需要截图的地方均在步骤前会说明，请结合作业模板提交最终的作业。**

**2. Data Studio的使用可以参考《随堂练习手册》。**

数据库由各种数据库对象组成，其中表是数据库中最常用的数据库对象。在关系数据库中，表用来表示数据和数据之间的联系，关系(relation)即表示表。表的设计原则主要包括：

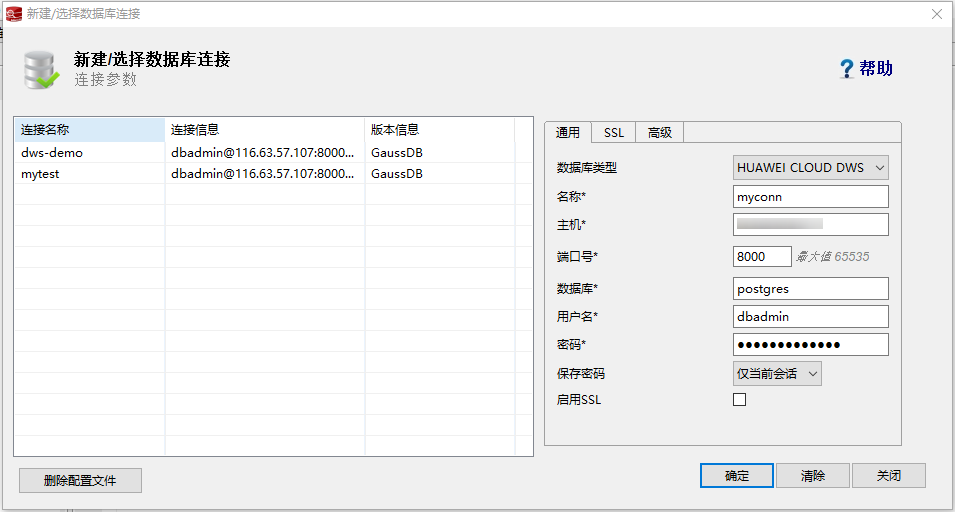
* 数据在各个节点尽量均匀分布，避免数据倾斜。
* 存储方式
  + 行存：增删改操作较多的场景。查询返回记录少，返回全部属性。
  + 列存：查询复杂，涉及多表关联，分组等操作。
* 分布方式
  + 哈希表：表数据量较大
  + 复制表：表数据量较小；维度表
* 减少查询扫描数据量。通过分区表可以将扫描限制在某些分区，减少数据量。
  + 将具体明显区间性的字段进行分区，如日期。
  + 分区上边界值定义为MAXVALUE，避免数据溢出。
* 分布列选择原则
  + 作为分布列的字段取值尽量离散。
  + 尽量不选择在查询中存在形如 col = 10001 过滤条件的字段。
  + 选择查询中关联条件作为分布列。

#### 创建数据库和模式

此处需要截图，截图内容：创建的个人数据库及两个模式。

使用Data Studio连接postgres数据库

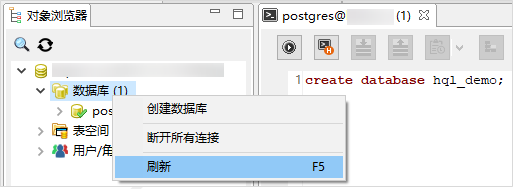
在Data Studio.exe上右键，选择“以管理员身份运行”。在连接界面输入相关参数：

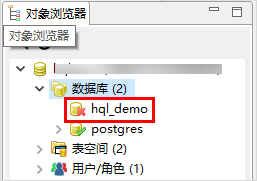


创建数据库，数据库名建议为个人姓名\_demo。

CREATE DATABASE hql\_demo;

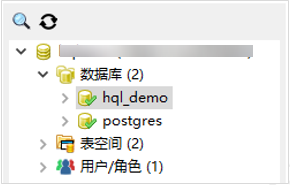
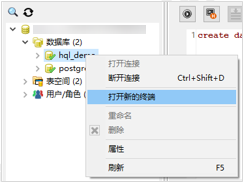
鼠标右键数据库，刷新数据库。





双击个人新建的数据库，建立连接。鼠标右键数据库，打开新的终端。



后续再次连接时，在连接界面的数据库处填入个人创建的数据库名即可。

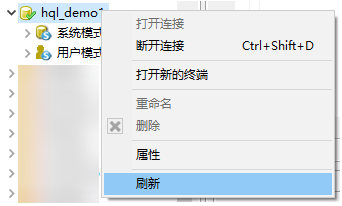


创建模式：original和target。

CREATE SCHEMA original;

CREATE SCHEMA target;

在个人创建的数据库上右键-“刷新”，查看新创建的模式。

模式original主要用于保存源事务型数据库中的数据。在模式original下建立的表与源事务系统中的表结构保持一致。

模式target下创建的表主要为依据维度建模所构建的维度表和事实表。

GaussDB(DWS)支持不同模式之间的互访（管理员用户具有最高权限）。

#### 创建源表

此步骤主要用于保存源事务型系统的所有数据，为后期建模做好数据准备。

源表主要创建在模式original下。

**此处需要截图，截图内容：original模式下所创建的表，具体可以参考“查看创建的表”步骤。**

创建表original\_goods

DROP TABLE IF EXISTS original.original\_goods;

CREATE TABLE original.original\_goods (

id int NOT NULL,

goods\_sn varchar(63) NOT NULL,

name varchar(127) NOT NULL,

category\_id int,

brand\_id int,

gallery varchar(1023),

keywords varchar(255),

brief varchar(1023),

is\_on\_sale tinyint,

sort\_order smallint,

pic\_url varchar(1023),

share\_url varchar(1023),

is\_new tinyint,

is\_hot tinyint,

unit varchar(31),

counter\_price decimal(10,2),

retail\_price decimal(10,2),

detail text,

add\_time date,

update\_time date,

deleted tinyint,

PRIMARY KEY (id)

);

返回结果：

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 132 ms

[INFO] 执行成功...

[NOTICE] table "original\_goods" does not exist, skipping

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "original\_goods\_pkey" for table "original\_goods"

创建表original\_goods\_product

DROP TABLE IF EXISTS original.original\_goods\_product;

CREATE TABLE original.original\_goods\_product (

id int NOT NULL,

goods\_id int NOT NULL,

specifications varchar(1023) NOT NULL,

price decimal(10,2) NOT NULL,

number int NOT NULL,

url varchar(125),

add\_time date,

update\_time date,

deleted tinyint,

PRIMARY KEY (id)

);

返回结果：

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 147 ms

[INFO] 执行成功...

[NOTICE] table "original\_goods\_product" does not exist, skipping

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "original\_goods\_product\_pkey" for table "original\_goods\_product"

创建original\_order

DROP TABLE IF EXISTS original.original\_order;

CREATE TABLE original.original\_order (

id int NOT NULL,

user\_id int NOT NULL,

order\_sn varchar(63) NOT NULL,

order\_status smallint NOT NULL,

aftersale\_status smallint,

consignee varchar(63) NOT NULL,

mobile varchar(63) NOT NULL,

address varchar(127) NOT NULL,

message varchar(512),

goods\_price decimal(10,2) NOT NULL,

freight\_price decimal(10,2) NOT NULL,

coupon\_price decimal(10,2) NOT NULL,

integral\_price decimal(10,2) NOT NULL,

groupon\_price decimal(10,2) NOT NULL,

order\_price decimal(10,2) NOT NULL,

actual\_price decimal(10,2) NOT NULL,

pay\_id varchar(63),

pay\_time date,

ship\_sn varchar(63),

ship\_channel varchar(63),

ship\_time date,

refund\_amount decimal(10,2),

refund\_type varchar(63),

refund\_content varchar(127),

refund\_time date,

confirm\_time date,

comments smallint,

end\_time date,

add\_time date,

update\_time date,

deleted tinyint,

PRIMARY KEY (id)

);

返回结果：

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 97 ms

[INFO] 执行成功...

[NOTICE] table "original\_order" does not exist, skipping

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "original\_order\_pkey" for table "original\_order"

创建original\_order\_goods

DROP TABLE IF EXISTS original.original\_order\_goods;

CREATE TABLE original.original\_order\_goods (

id int NOT NULL,

order\_id int NOT NULL,

goods\_id int NOT NULL,

goods\_name varchar(127) NOT NULL,

goods\_sn varchar(63) NOT NULL,

product\_id int NOT NULL,

number smallint NOT NULL,

price decimal(10,2) NOT NULL,

specifications varchar(1023) NOT NULL,

pic\_url varchar(1023) NOT NULL,

comment int,

add\_time date,

update\_time date,

deleted tinyint,

PRIMARY KEY (id)

);

返回结果：

[NOTICE] table "original\_order\_goods" does not exist, skipping

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 102 ms

[INFO] 执行成功...

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "original\_order\_goods\_pkey" for table "original\_order\_goods"

创建original\_address

DROP TABLE IF EXISTS original.original\_address;

CREATE TABLE original.original\_address (

id int NOT NULL,

name varchar(63) NOT NULL,

user\_id int NOT NULL,

province varchar(63) NOT NULL,

city varchar(63) NOT NULL,

county varchar(63) NOT NULL,

address\_detail varchar(127) NOT NULL,

area\_code char(6),

postal\_code char(6),

tel varchar(20) NOT NULL,

is\_default tinyint NOT NULL,

add\_time date,

update\_time date,

deleted tinyint,

PRIMARY KEY (id)

);

返回结果：

[NOTICE] table "original\_address" does not exist, skipping

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 113 ms

[INFO] 执行成功...

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "original\_address\_pkey" for table "original\_address"

创建original\_user

DROP TABLE IF EXISTS original.original\_user;

CREATE TABLE original.original\_user (

id int NOT NULL,

username varchar(63) NOT NULL,

password varchar(63) NOT NULL,

gender tinyint NOT NULL,

birthday date,

last\_login\_time date,

last\_login\_ip varchar(63) NOT NULL,

user\_level tinyint,

nickname varchar(63),

mobile varchar(20) NOT NULL,

avatar varchar(255),

weixin\_openid varchar(63),

session\_key varchar(100),

status tinyint NOT NULL,

add\_time date,

update\_time date,

deleted tinyint,

PRIMARY KEY (id)

);

返回结果：

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 100 ms

[INFO] 执行成功...

[NOTICE] table "original\_user" does not exist, skipping

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "original\_user\_pkey" for table "original\_user"

创建表original\_category

DROP TABLE IF EXISTS original.original\_category;

CREATE TABLE original.original\_category (

id int NOT NULL,

name varchar(63) NOT NULL,

keywords varchar(1023),

cdesc varchar(255),

pid int NOT NULL,

icon\_url varchar(255),

pic\_url varchar(1023),

level varchar(255),

sort\_order tinyint,

add\_time date,

update\_time date,

deleted tinyint,

PRIMARY KEY (id)

);

返回结果：

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 116 ms

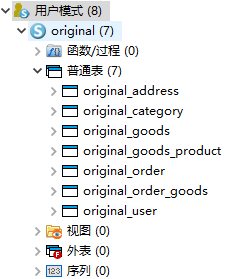
[INFO] 执行成功...

[NOTICE] table "original\_category" does not exist, skipping

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "original\_category\_pkey" for table "original\_category"

查看创建的表

在Data Studio左侧”用户模式“上右键，选择“刷新”后，展开original模式。可以看到刚刚创建的表。



#### 创建目标表

目标表（维度表和事实表）主要创建在模式target下。

**此处需要截图，截图内容：target模式下所创建的表和序列，具体可以参考“查看创建的表”步骤。**

创建日期维度表

DROP TABLE IF EXISTS target.date\_dimension;

CREATE TABLE target.date\_dimension (

date\_key integer NOT NULL,

full\_date date,

day\_of\_week tinyint,

day\_num\_in\_month tinyint,

day\_num\_overall smallint,

day\_name varchar(9),

day\_abbrev char(3),

weekday\_flag varchar(10),

week\_num\_in\_year tinyint,

week\_num\_overall smallint,

week\_begin\_date date,

week\_begin\_date\_key integer,

month tinyint,

month\_num\_overall smallint,

month\_name varchar(9),

month\_abbrev char(3),

quarter tinyint,

year smallint,

yearmo int,

fiscal\_month tinyint,

fiscal\_quarter tinyint,

fiscal\_year smallint,

last\_day\_in\_month\_flag varchar(15),

same\_day\_year\_ago\_date date,

PRIMARY KEY (date\_key)

)

DISTRIBUTE BY REPLICATION;

返回结果：

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 85 ms

[INFO] 执行成功...

[NOTICE] table "date\_dimension" does not exist, skipping

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "date\_dimension\_pkey" for table "date\_dimension"

创建时间维度表

DROP TABLE IF EXISTS target.time\_dimension;

CREATE TABLE target.time\_dimension (

time\_key integer NOT NULL,

hour integer,

minute integer,

second integer,

t\_am\_pm char(2),

PRIMARY KEY (time\_key)

)

DISTRIBUTE BY REPLICATION;

返回结果：

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 109 ms

[INFO] 执行成功...

[NOTICE] table "time\_dimension" does not exist, skipping

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "time\_dimension\_pkey" for table "time\_dimension"

创建商品维度表

DROP TABLE IF EXISTS target.goods\_dimension;

CREATE TABLE target.goods\_dimension (

goods\_key serial,

goods\_id integer,

goods\_sn varchar(63),

goods\_name varchar(127),

category\_id integer,

category\_name varchar(63),

category\_keywords varchar(1023),

category\_desc varchar(255),

category\_pid integer,

brand\_id integer,

gallery varchar(1023),

keywords varchar(255),

brief varchar(1023),

is\_on\_sale tinyint,

sort\_order smallint,

pic\_url varchar(1023),

share\_url varchar(1023),

is\_new tinyint,

is\_hot tinyint,

unit varchar(31),

counter\_price decimal(10, 2),

retail\_price decimal(10,2),

detail text,

product\_id integer,

specifications varchar(1023),

gp\_add\_date integer,

gp\_add\_time integer,

gp\_update\_date integer,

gp\_update\_time integer,

gp\_deleted tinyint,

category\_icon\_url varchar(1023),

category\_pic\_url varchar(1023),

category\_level varchar(255),

category\_sort\_order tinyint,

category\_add\_date integer,

category\_add\_time integer,

category\_update\_date integer,

category\_update\_time integer,

category\_deleted tinyint,

goods\_add\_date integer,

goods\_add\_time integer,

goods\_update\_date integer,

goods\_update\_time integer,

goods\_deleted tinyint,

PRIMARY KEY (goods\_key)

)

DISTRIBUTE BY REPLICATION;

返回结果：

[NOTICE] table "goods\_dimension" does not exist, skipping

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 338 ms

[INFO] 执行成功...

[NOTICE] CREATE TABLE will create implicit sequence "goods\_dimension\_goods\_key\_seq" for serial column "goods\_dimension.goods\_key"

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "goods\_dimension\_pkey" for table "goods\_dimension"

创建用户维度表

DROP TABLE IF EXISTS target.user\_dimension;

CREATE TABLE target.user\_dimension (

user\_key serial,

user\_id integer NOT NULL,

username varchar(63),

password varchar(63),

gender tinyint,

birthday integer,

last\_login\_date integer,

last\_login\_time integer,

last\_login\_ip varchar(63),

user\_level tinyint,

nickname varchar(63),

mobile varchar(20),

avatar varchar(255),

weixin\_openid varchar(63),

session\_key varchar(100),

status tinyint,

add\_date integer,

add\_time integer,

update\_date integer,

update\_time integer,

deleted tinyint,

PRIMARY KEY (user\_key)

)

DISTRIBUTE BY REPLICATION;

[NOTICE] table "user\_dimension" does not exist, skipping

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 301 ms

[INFO] 执行成功...

[NOTICE] CREATE TABLE will create implicit sequence "user\_dimension\_user\_key\_seq" for serial column "user\_dimension.user\_key"

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "user\_dimension\_pkey" for table "user\_dimension"

创建地址维度表

DROP TABLE IF EXISTS target.address\_dimension;

CREATE TABLE target.address\_dimension (

address\_key serial,

address\_id integer NOT NULL,

province varchar(63),

city varchar(63),

county varchar(63),

address\_detail varchar(127),

area\_code char(6),

postal\_code char(6),

is\_default tinyint,

add\_date integer,

add\_time integer,

update\_date integer,

update\_time integer,

deleted tinyint,

PRIMARY KEY (address\_key)

)

DISTRIBUTE BY REPLICATION;

返回结果：

[NOTICE] table "address\_dimension" does not exist, skipping

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 308 ms

[INFO] 执行成功...

[NOTICE] CREATE TABLE will create implicit sequence "address\_dimension\_address\_key\_seq" for serial column "address\_dimension.address\_key"

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "address\_dimension\_pkey" for table "address\_dimension"

创建订单表

DROP TABLE IF EXISTS target.orders;

CREATE TABLE target.orders (

order\_id integer NOT NULL,

order\_sn varchar(63),

order\_status smallint,

user\_key integer,

address\_key integer,

goods\_price decimal(10, 2),

freight\_price decimal(10, 2),

coupon\_price decimal(10, 2),

integral\_price decimal(10, 2),

groupon\_price decimal(10, 2),

order\_price decimal(10, 2),

actual\_price decimal(10, 2),

pay\_id varchar(63),

pay\_date integer,

pay\_time integer,

ship\_sn varchar(63),

ship\_channel varchar(63),

ship\_date integer,

ship\_time integer,

confirm\_date integer,

confirm\_time integer,

message varchar(512),

comments smallint,

aftersale\_status smallint,

refund\_amount decimal(10, 2),

refund\_type varchar(63),

refund\_content varchar(127),

refund\_date integer,

refund\_time integer,

end\_date integer,

end\_time integer,

add\_date integer,

add\_time integer,

update\_date integer,

update\_time integer,

deleted tinyint,

PRIMARY KEY (order\_id)

)

DISTRIBUTE BY HASH (order\_id)

PARTITION BY RANGE (order\_id)

(

PARTITION p1 VALUES LESS THAN(20000),

PARTITION p2 VALUES LESS THAN(40000),

PARTITION p3 VALUES LESS THAN(60000),

PARTITION p4 VALUES LESS THAN(80000),

PARTITION p5 VALUES LESS THAN(MAXVALUE)

);

返回结果：

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 170 ms

[INFO] 执行成功...

[NOTICE] table "orders" does not exist, skipping

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "orders\_pkey" for table "orders"

创建订单商品表

DROP TABLE IF EXISTS target.order\_goods;

CREATE TABLE target.order\_goods (

order\_id integer,

goods\_key integer,

number integer,

price decimal(10, 2),

comment\_value integer,

add\_date integer,

add\_time integer,

update\_date integer,

update\_time integer,

deleted tinyint,

PRIMARY KEY (order\_id, goods\_key)

)

DISTRIBUTE BY HASH(order\_id, goods\_key);

返回结果：

[NOTICE] table "order\_goods" does not exist, skipping

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 116 ms

[INFO] 执行成功...

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "order\_goods\_pkey" for table "order\_goods"

创建库存表

DROP TABLE IF EXISTS target.inventory;

CREATE TABLE target.inventory (

date\_key integer NOT NULL,

goods\_key integer NOT NULL,

number integer,

PRIMARY KEY (date\_key, goods\_key)

)

DISTRIBUTE BY HASH(date\_key, goods\_key);

返回结果：

[INFO] 操作影响的记录行数 : 0

[INFO] 执行时间 : 69 ms

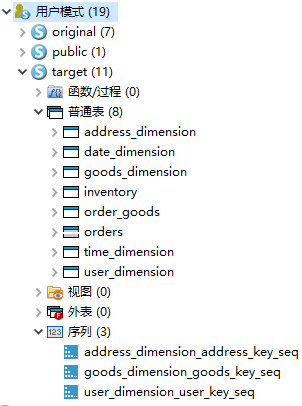
[INFO] 执行成功...

[NOTICE] table "inventory" does not exist, skipping

[NOTICE] CREATE TABLE / PRIMARY KEY will create implicit index "inventory\_pkey" for table "inventory"

查看创建的表

在Data Studio左侧”用户模式“上右键，选择“刷新”后，展开target模式。可以看到刚刚创建的表以及序列。



### 数据转换

#### 准备工作

说明：

1. 此处以Windows系统为例。

2. **本次提供的安装包已经将相关的驱动文件添加至相应的目录下**，**无需个人下载，直接使用Kettle即可**。1.2.3.1小节的内容主要用于学习参考。

3. 因关卡二的操作不同，本实验的结果与示例截图可能略有差异，请以个人最终结果为准。

下载Kettle。

Kettle是一款国外开源的ETL工具，可以在Windows、Linux、Unix上运行，数据抽取高效稳定。

在以下链接：<https://sourceforge.net/projects/pentaho/files/Data%20Integration/> 下载Kettle安装包。





下载之后完成解压即可。

下载DWS JDBC驱动

在DWS集群“连接管理”界面下载JDBC驱动，或者由授课老师提供该驱动。



将下载的JDBC驱动dws\_8.1.x\_jdbc\_driver.zip包解压，并将其中的gsjdbc4.jar拷贝到\data-integration\lib目录下。

下载MySQL JDBC驱动

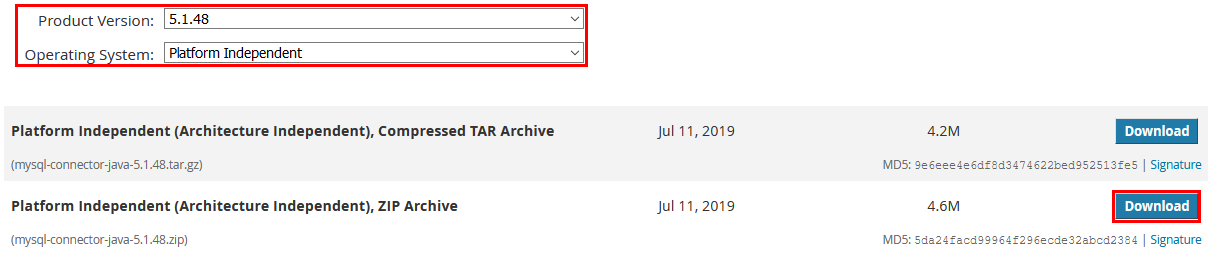
在以下链接中：<https://downloads.mysql.com/archives/c-j/> 下载和电脑版本对应的MySQL JDBC驱动包。

此处：

Product Version可以选择5.1.48（RDS for MySQL数据库引擎为5.7版本）；

Operation System选择Platform Independent

点击download。



将下载的JDBC驱动解压，并将mysql-connector-java-5.1.48.jar拷贝到\data-integration\lib目录下。

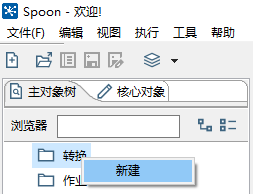
#### 创建数据库连接

**此处需要截图，截图内容：正确连接到数据库rdsformysql的测试提示信息以及正确连接到数据库dws的测试提示信息。**

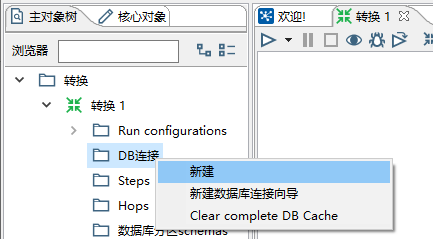
新建转换

在data-integration文件夹中双击Spoon.bat。

在“主对象树”-“转换”上右键，点击“新建”。



建立rdsformysql连接



在弹出来的数据库连接界面：

连接名称：rdsformysql

连接类型：MySQL

连接方式：Native(JDBC)

主机名称：RDS for MySQL公网IP，本实验手册中为：121.36.99.22，请替换为个人购买的RDS for MySQL公网IP

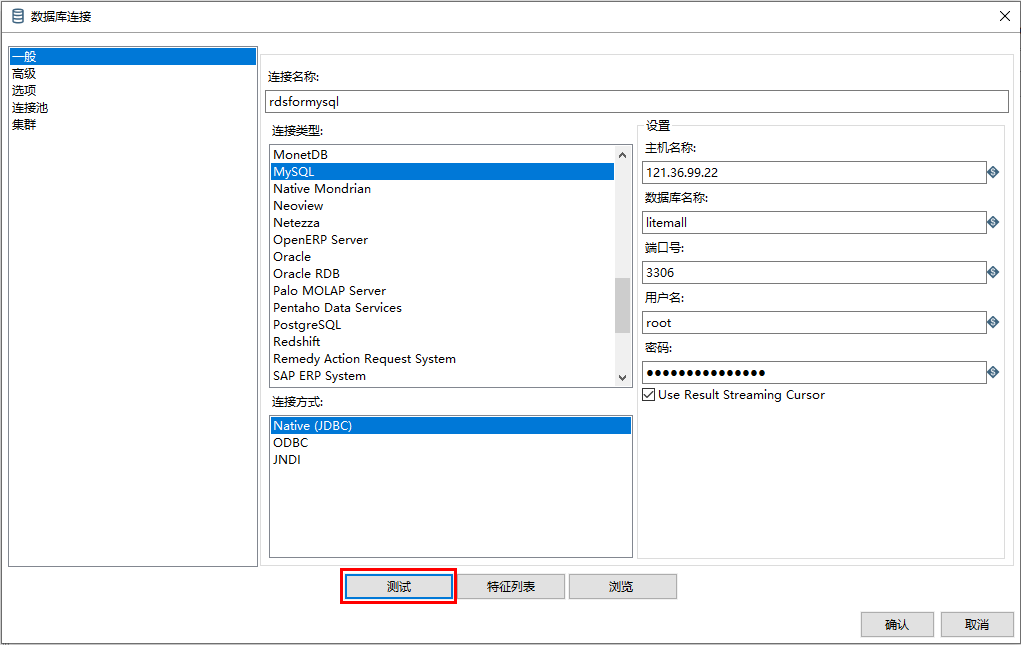
数据库名称：litemall

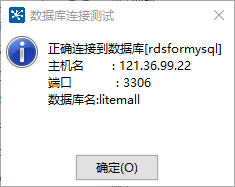
端口号：3306

用户名：root

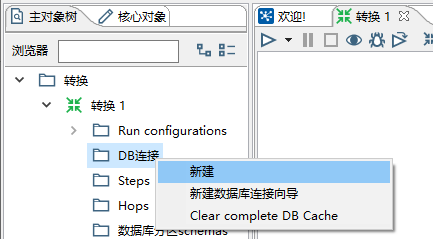
密码：root用户密码（为购买服务过程中设置的密码）

填写完之后点击“测试”，提示：正确连接到数据库。





建立dws连接



在弹出来的数据库连接界面：

连接名称：dws

连接类型：Generic database

连接方式：Native(JDBC)

自定义连接URL：jdbc:postgresql://*dws-demo*.dws.huaweicloud.com:8000/*个人创建的数据库名*，例如内网：jdbc:postgresql://*dws-demo*.dws.huaweicloud.com:8000/hql\_demo

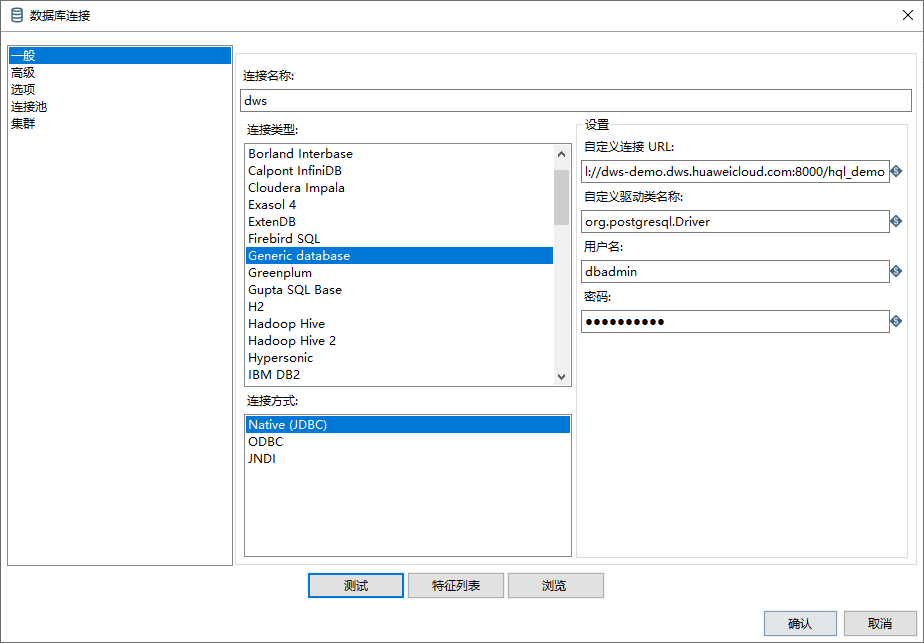
例如公网：jdbc:postgresql://124.71.228.33:8000/hql\_demo

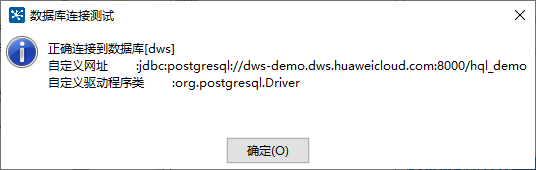
自定义驱动类名称：org.postgresql.Driver

用户名：dbadmin

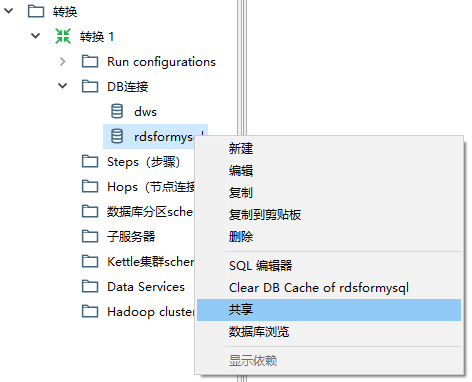
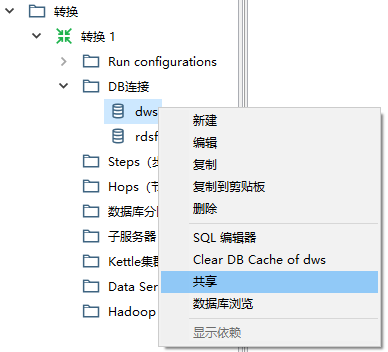
密码：dbadmin用户密码（为购买服务过程中设置的密码）

填写完之后点击“测试”，提示：正确连接到数据库。

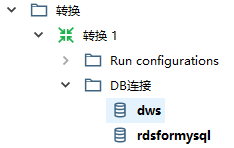




共享数据库连接



设置完共享之后可以发现，刚刚建立的数据库连接变为黑色粗体字。



保存转换并重启spoon.bat

保存该“转换”，命名为connection（此命名只是举例）。保存后关闭spoon.bat并重新打开使得共享数据库连接生效。

#### 利用Kettle完成数据迁移

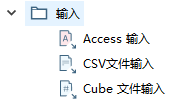
**此处需要截图，截图内容：每个转换的“步骤度量”信息。**

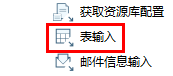
说明：此处创建了7个转换，完成7张表的数据迁移工作。当然在一个转换中完成7张表的数据迁移也可以，只是这样的话，如果某一步错误，相应的改动会比较大。

完成original\_goods数据导入

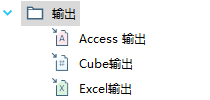
新建一个转换，名为goods。

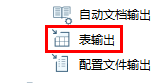
在“核心对象”-“输入”，双击“表输入”。



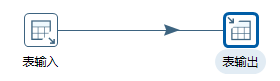


在“核心对象”-“输出”，双击“表输出”。





会建立如下转换。

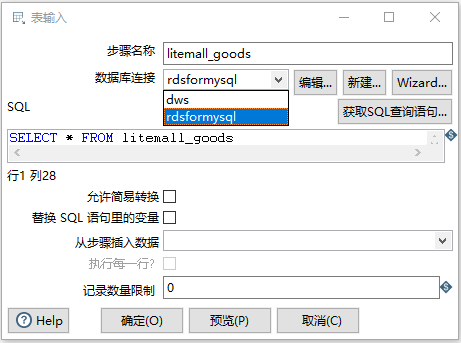


双击“表输入”，在弹出来的界面中：

步骤名称：litemall\_goods

数据库连接：rdsformysql（此时可以看到之前创建的数据库连接）

SQL: SELECT \* FROM litemall\_goods



参数配置完后可以点击“预览”查看相关数据。



最后点击“确定”。

双击“表输出”，在弹出来的界面中：

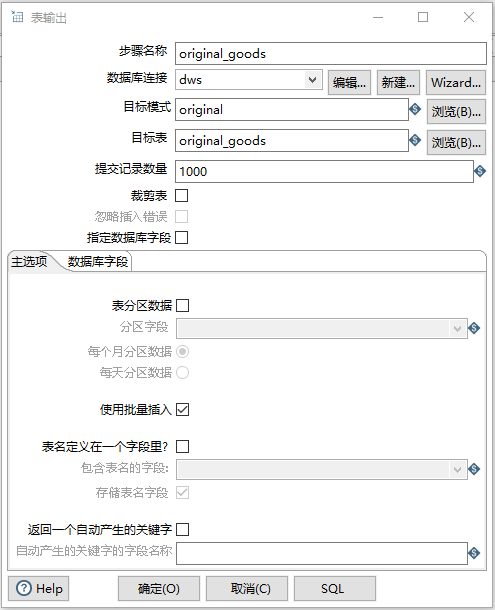
步骤名称：original\_goods

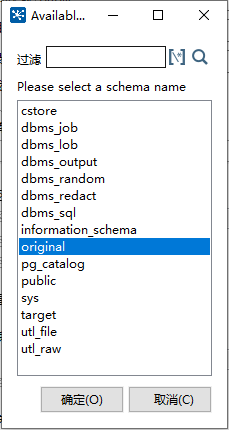
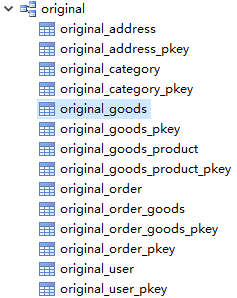
数据库连接：dws

模式：点击“浏览”，选择“original”

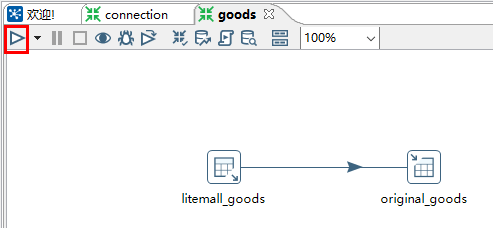
目标表：点击“浏览”，在“original”模式下选择“original\_goods”

其他保持默认即可，最后点击“确定”。

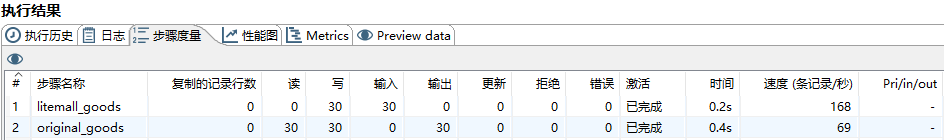


保存该转换后点击左上角运行图标，在弹出的界面中点击“启动”。



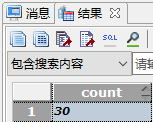
在“执行结果”-“步骤度量”中可以看到执行的记录数及用时情况。



在Data Studio中，执行以下命令：

SELECT COUNT(\*) FROM original.original\_goods;

返回结果：



说明表litemall\_goods中的数据已经迁移成功。

完成original\_goods\_product数据导入

新建一个转换，名为goods\_product。配置如下：

表输入：

步骤名称：litemall\_goods\_product

数据库连接：rdsformysql

SQL：SELECT \* FROM litemall\_goods\_product

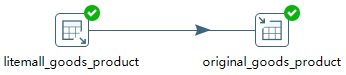
表输出：

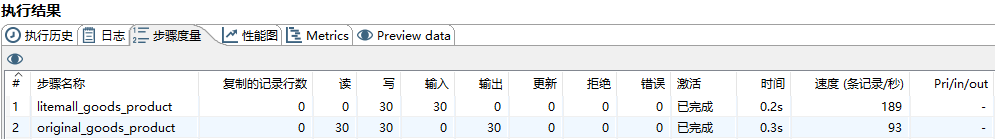
步骤名称：original\_goods\_product

数据库连接：dws

目标模式：original

目标表：original\_goods\_product

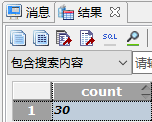




在Data Studio中，执行以下命令：

SELECT COUNT(\*) FROM original.original\_goods\_product;

返回结果：



说明表litemall\_goods\_product中的数据已经迁移成功。

完成original\_order数据导入

新建一个转换，名为order。配置如下：

表输入：

步骤名称：litemall\_order

数据库连接：rdsformysql

SQL：SELECT \* FROM litemall\_order

表输出：

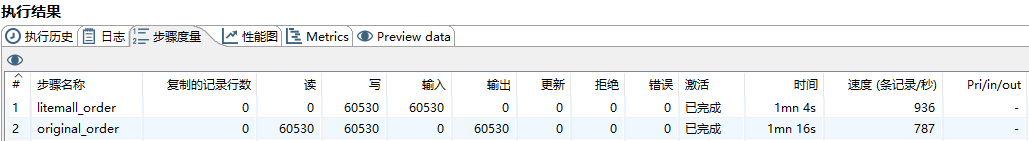
步骤名称：original\_order

数据库连接：dws

目标模式：original

目标表：original\_order





完成original\_order\_goods数据导入

新建一个转换，名为order\_goods。配置如下：

表输入：

步骤名称：litemall\_order\_goods

数据库连接：rdsformysql

SQL：SELECT \* FROM litemall\_order\_goods

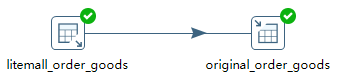
表输出：

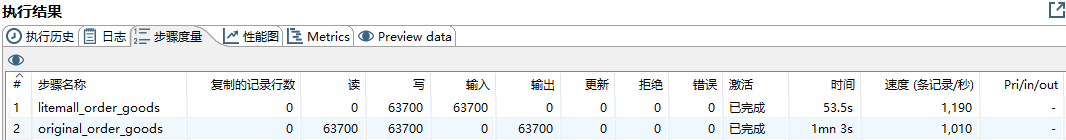
步骤名称：original\_order\_goods

数据库连接：dws

目标模式：original

目标表：original\_order\_goods





完成original\_address数据导入

新建一个转换，名为address。配置如下：

表输入：

步骤名称：litemall\_address

数据库连接：rdsformysql

SQL：SELECT \* FROM litemall\_address

表输出：

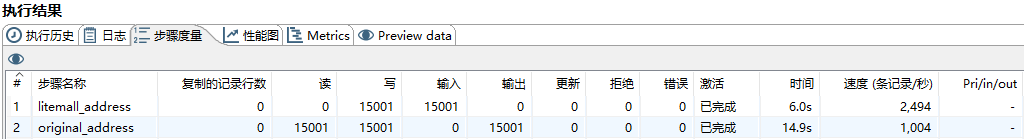
步骤名称：original\_address

数据库连接：dws

目标模式：original

目标表：original\_address





完成original\_user数据导入

新建一个转换，名为user。配置如下：

表输入：

步骤名称：litemall\_user

数据库连接：rdsformysql

SQL：SELECT \* FROM litemall\_user

表输出：

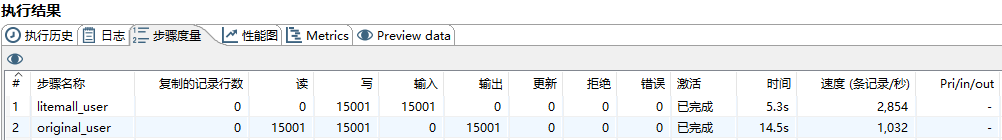
步骤名称：original\_user

数据库连接：dws

目标模式：original

目标表：original\_user





完成original\_category数据导入

新建一个转换，名为category。配置如下：

表输入：

步骤名称：litemall\_category

数据库连接：rdsformysql

SQL：SELECT \* FROM litemall\_category

表输出：

步骤名称：original\_category

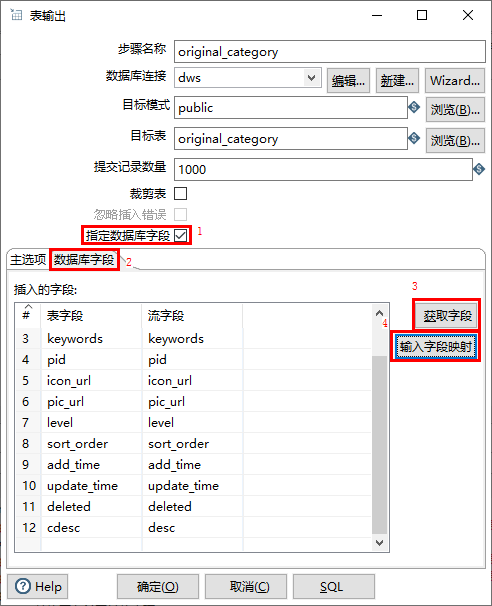
数据库连接：dws

目标模式：original

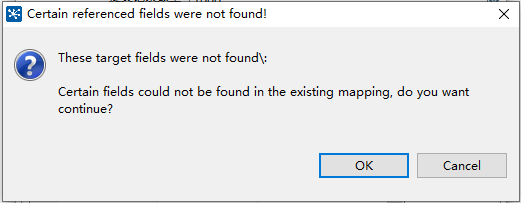
目标表：original\_category

**此处需要注意，对于此步骤，额外需要做如下操作：**

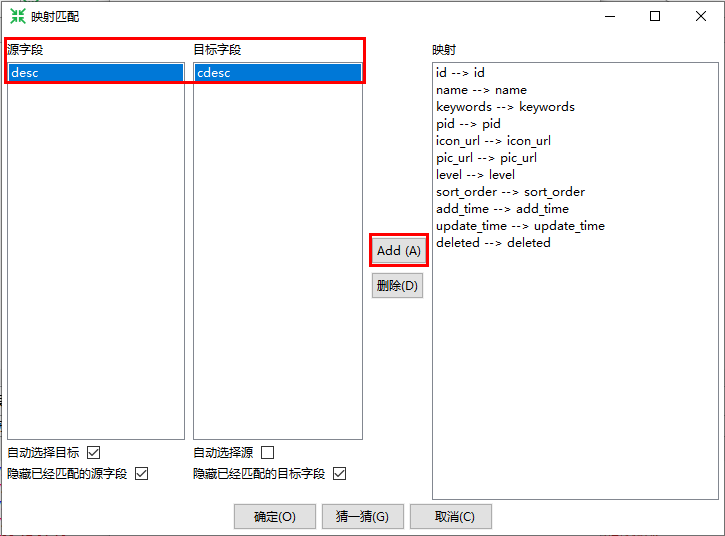
勾选“指定数据库字段”，在“数据库字段”下点击“获取字段”。



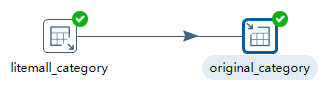
点击“输入字段映射”，在弹出的提示框中点击“OK”。

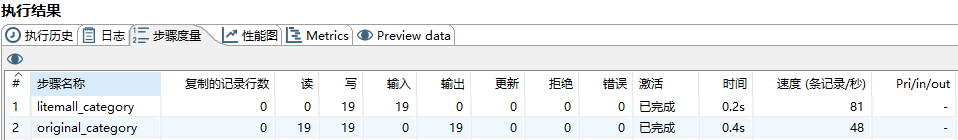


选择“desc”，点击“Add”，点击“确定”。（因为desc在DWS中是关键字，所以litemall\_category中的desc字段，更改其字段名为cdesc）



最后点击“确定”。





#### 在DWS中完成转换

此处需要截图，截图内容：

说明：

1. 先将date\_dimension.txt和time\_dimension.txt两个文件下载到本地，**进行步骤2之前务必先完成date\_dimension.txt和time\_dimension.txt的导入工作。**

2. 1.2.3.4小节的内容主要是利用SQL语句实现事实表和维度表的转换。

使用Data Studio完成date\_dimension和time\_dimension的导入

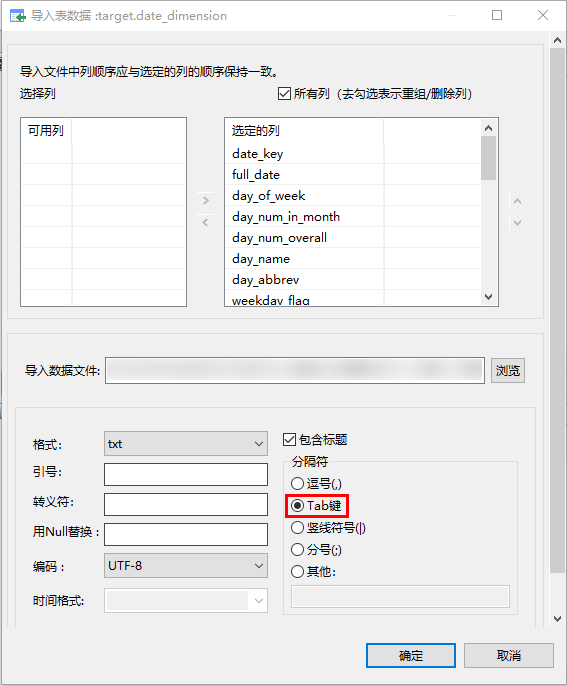
# 导入日期表date\_dimension

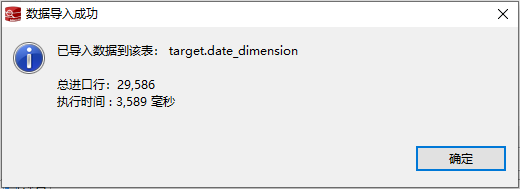
在target模式下找到date\_dimension表，右键 - 导入表数据。

点击浏览，找到date\_dimension.txt文件。

分隔符选择Tab键。

点击确定完成导入。





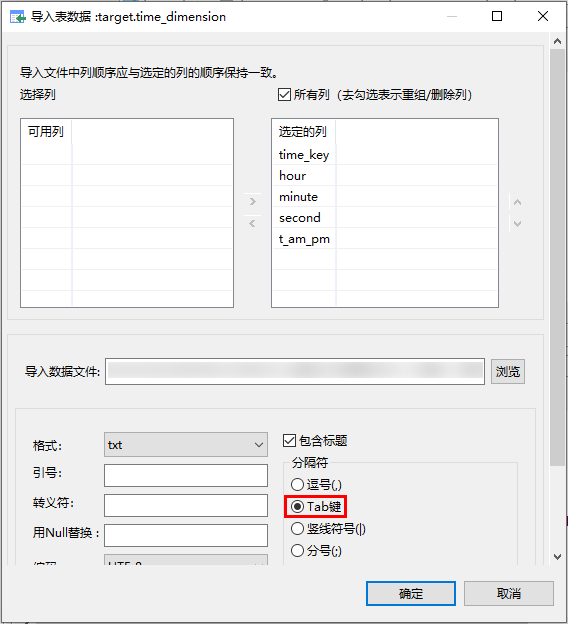
# 导入时间表time\_dimension

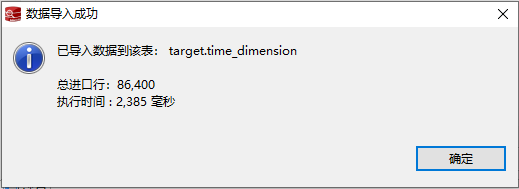
在Data Studio中找到time\_dimension表，右键 - 导入表数据。

点击浏览，找到time\_dimension.txt文件。

分隔符选择Tab键。

点击确定完成导入。





完成表goods\_dimension

INSERT INTO target.goods\_dimension

(goods\_id,

goods\_sn,

goods\_name,

category\_id,

category\_name,

category\_keywords,

category\_desc,

category\_pid,

brand\_id,

gallery,

keywords,

brief,

is\_on\_sale,

sort\_order,

pic\_url,

share\_url,

is\_new,

is\_hot,

unit,

counter\_price,

retail\_price,

detail,

product\_id,

specifications,

gp\_add\_date,

gp\_add\_time,

gp\_update\_date,

gp\_update\_time,

gp\_deleted,

category\_icon\_url,

category\_pic\_url,

category\_level,

category\_sort\_order,

category\_add\_date,

category\_add\_time,

category\_update\_date,

category\_update\_time,

category\_deleted,

goods\_add\_date,

goods\_add\_time,

goods\_update\_date,

goods\_update\_time,

goods\_deleted)

SELECT

og.id AS goods\_id,

og.goods\_sn,

og.name AS goods\_name,

category\_id,

oc.name AS category\_name,

oc.keywords AS category\_keywords,

oc.cdesc AS category\_desc,

oc.pid AS category\_pid,

og.brand\_id,

og.gallery,

og.keywords,

og.brief,

og.is\_on\_sale,

og.sort\_order,

og.pic\_url,

og.share\_url,

og.is\_new,

og.is\_hot,

og.unit,

og.counter\_price,

og.retail\_price,

og.detail,

ogp.id AS product\_id,

ogp.specifications,

to\_char(ogp.add\_time,'YYYYMMDD') :: INTEGER AS gp\_add\_date,

0 AS gp\_add\_time,

to\_char(ogp.update\_time, 'YYYYMMDD') :: INTEGER AS gp\_update\_date,

0 AS gp\_update\_time,

ogp.deleted,

oc.icon\_url AS category\_icon\_url,

oc.pic\_url AS category\_pic\_url,

oc.level AS category\_level,

oc.sort\_order AS category\_sort\_order,

to\_char(oc.add\_time, 'YYYYMMDD') :: INTEGER AS category\_add\_date,

0 AS category\_add\_time,

to\_char(oc.update\_time, 'YYYYMMDD') :: INTEGER AS category\_update\_date,

0 AS category\_update\_time,

oc.deleted AS category\_deleted,

to\_char(og.add\_time, 'YYYYMMDD') :: INTEGER AS goods\_add\_date,

0 AS goods\_add\_time,

to\_char(og.update\_time, 'YYYYMMDD') :: INTEGER AS goods\_update\_date,

0 AS goods\_update\_time,

og.deleted AS goods\_deleted

FROM original.original\_goods og,

original.original\_goods\_product ogp,

original.original\_category oc

WHERE og.category\_id = oc.id AND og.id = ogp.goods\_id

ORDER BY og.id, ogp.id;

完成表user\_dimension

INSERT INTO target.user\_dimension

(user\_id,

username,

password,

gender,

birthday,

last\_login\_date,

last\_login\_time,

last\_login\_ip,

user\_level,

nickname,

mobile,

avatar,

weixin\_openid,

session\_key,

status,

add\_date,

add\_time,

update\_date,

update\_time,

deleted)

SELECT id AS user\_id,

username,

password,

gender,

to\_char(birthday, 'YYYYMMDD') :: INTEGER AS birthday,

to\_char(last\_login\_time, 'YYYYMMDD') :: INTEGER AS last\_login\_date,

0 AS last\_login\_time,

last\_login\_ip,

user\_level,

nickname,

mobile,

avatar,

weixin\_openid,

session\_key,

status,

to\_char(add\_time, 'YYYYMMDD') :: INTEGER AS add\_date,

0 AS add\_time,

to\_char(update\_time, 'YYYYMMDD') :: INTEGER AS update\_date,

0 AS update\_time,

deleted

FROM original.original\_user

ORDER BY id;

完成表address\_dimension

INSERT INTO target.address\_dimension

(address\_id,

province,

city,

county,

address\_detail,

area\_code,

postal\_code,

is\_default,

add\_date,

add\_time,

update\_date,

update\_time,

deleted)

SELECT id AS address\_id,

province,

city,

county,

address\_detail,

area\_code,

postal\_code,

is\_default,

to\_char(add\_time, 'YYYYMMDD') :: INTEGER AS add\_date,

0 AS add\_time,

to\_char(update\_time, 'YYYYMMDD') :: INTEGER AS update\_date,

0 AS update\_time,

deleted

FROM original.original\_address

ORDER BY id;

完成表orders

INSERT INTO target.orders

(order\_id,

order\_sn,

order\_status,

user\_key,

address\_key,

goods\_price,

freight\_price,

coupon\_price,

integral\_price,

groupon\_price,

order\_price,

actual\_price,

pay\_id,

pay\_date,

pay\_time,

ship\_sn,

ship\_channel,

ship\_date,

ship\_time,

confirm\_date,

confirm\_time,

message,

comments,

aftersale\_status,

refund\_amount,

refund\_type,

refund\_content,

refund\_date,

refund\_time,

end\_date,

end\_time,

add\_date,

add\_time,

update\_date,

update\_time,

deleted)

select order\_id,

order\_sn,

order\_status,

user\_key,

address\_key,

goods\_price,

freight\_price,

coupon\_price,

integral\_price,

groupon\_price,

order\_price,

actual\_price,

pay\_id,

pay\_date,

pay\_time,

ship\_sn,

ship\_channel,

ship\_date,

ship\_time,

confirm\_date,

confirm\_time,

message,

comments,

aftersale\_status,

refund\_amount,

refund\_type,

refund\_content,

refund\_date,

refund\_time,

end\_date,

end\_time,

add\_date,

add\_time,

update\_date,

update\_time,

deleted

from (SELECT oo.id AS order\_id,

order\_sn,

order\_status,

ud.user\_key,

ad.address\_key,

goods\_price,

freight\_price,

coupon\_price,

integral\_price,

groupon\_price,

order\_price,

actual\_price,

pay\_id,

to\_char(pay\_time, 'YYYYMMDD') :: INTEGER AS pay\_date,

0 AS pay\_time,

ship\_sn,

ship\_channel,

to\_char(ship\_time, 'YYYYMMDD') :: INTEGER AS ship\_date,

0 AS ship\_time,

to\_char(confirm\_time, 'YYYYMMDD') :: INTEGER AS confirm\_date,

0 AS confirm\_time,

message,

comments,

aftersale\_status,

refund\_amount,

refund\_type,

refund\_content,

to\_char(refund\_time, 'YYYYMMDD') :: INTEGER AS refund\_date,

0 AS refund\_time,

to\_char(end\_time, 'YYYYMMDD') :: INTEGER AS end\_date,

0 AS end\_time,

to\_char(oo.add\_time, 'YYYYMMDD') :: INTEGER AS add\_date,

0 AS add\_time,

to\_char(oo.update\_time, 'YYYYMMDD') :: INTEGER AS update\_date,

0 AS update\_time,

oo.deleted,

row\_number() over(partition by oo.id) num

FROM original.original\_order oo

LEFT JOIN original.original\_address oa ON oo.user\_id = oa.user\_id

LEFT JOIN target.address\_dimension ad ON oa.id = ad.address\_id

LEFT JOIN target.user\_dimension ud ON ud.user\_id = oo.user\_id ) a

where a.num = 1

order by order\_id;

完成表order\_goods

INSERT INTO target.order\_goods(

order\_id,

goods\_key,

number,

price,

comment\_value,

add\_date,

add\_time,

update\_date,

update\_time,

deleted

)

SELECT oog.order\_id,

gd.goods\_key,

oog.number,

oog.price,

oog.comment,

to\_char(oog.add\_time, 'YYYYMMDD') :: INTEGER AS add\_date,

0 AS add\_time,

to\_char(oog.update\_time, 'YYYYMMDD') :: INTEGER AS update\_date,

0 AS update\_time,

oog.deleted

FROM original.original\_order\_goods oog

LEFT JOIN target.goods\_dimension gd ON oog.product\_id = gd.product\_id

ORDER BY order\_id,goods\_key;

完成表inventory

INSERT INTO target.inventory

(date\_key, goods\_key, number)

(WITH cur\_inv AS(

SELECT to\_char(inv.update\_time, 'YYYYMMDD') :: INTEGER AS update\_date,

inv.product\_id AS product\_id,

inv.number

from (SELECT update\_time,

id AS product\_id,

number,

ROW\_NUMBER() OVER(PARTITION BY id ORDER BY update\_time DESC) AS update\_inv

FROM original.original\_goods\_product ) inv

where inv.update\_inv = 1)

SELECT ci.update\_date, gd.goods\_key, ci.number

FROM cur\_inv ci

LEFT JOIN target.goods\_dimension gd on ci.product\_id = gd.product\_id

ORDER BY goods\_key);

### 数据分析

**此处需要截图，截图内容：补全的SQL语句以及结果截图（结果截图，SQL语句直接粘贴即可）。**

该部分直接在DWS集群中进行分析。

GMV(Gross Merchandise Volume = 销售额 + 取消订单金额 + 拒收订单金额 + 退货订单金额)相关分析。**提示：只要产生订单就纳入GMV的计算，所以不用专门看哪些订单被取消了，哪些订单被拒收了，直接对所有订单进行统计即可。**

# 查看2020年3月各省GMV，降序输出

SELECT ad.province AS province, SUM(o.actual\_price) AS GMV

FROM target.orders o,

target.address\_dimension ad,

target.date\_dimension dd

WHERE o.address\_key = ad.address\_key

AND o.add\_date = dd.date\_key

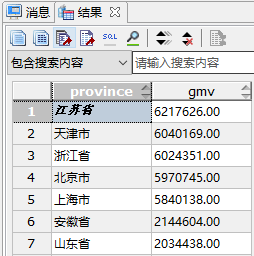
AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

ORDER BY SUM(o.actual\_price) DESC;

结果：



# 查看全国各省市GMV，时间范围为年初到今天（此处考虑到4个直辖市的信息与其他各省的信息不在一个字段，所以将查询结果做合并处理）

SELECT ad.province AS province,

ad.city AS city,

SUM(o.actual\_price) AS GMV

FROM target.orders o,

target.address\_dimension ad,

target.date\_dimension dd

WHERE o.address\_key = ad.address\_key

AND o.add\_date = dd.date\_key

AND to\_date(dd.full\_date) BETWEEN to\_date('2020/1/1')

AND current\_date

AND province not in('北京市', '上海市', '天津市', '重庆市')

GROUP BY ad.province, ad.city

UNION ALL

SELECT ad.province AS province,

ad.county AS city,

SUM(o.actual\_price) AS GMV

FROM target.orders o,

target.address\_dimension ad,

target.date\_dimension dd

WHERE o.address\_key = ad.address\_key

AND o.add\_date = dd.date\_key

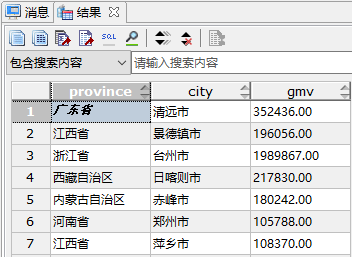
AND to\_date(dd.full\_date) BETWEEN to\_date('2020/1/1')

AND current\_date

AND province in('北京市', '上海市', '天津市', '重庆市')

GROUP BY ad.province, ad.county;

结果：



# 查看每周的总体GMV

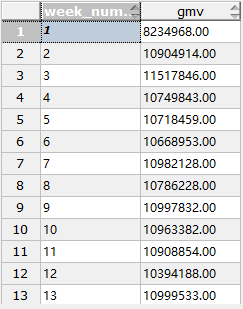
SELECT dd.week\_num\_in\_year, SUM(o.actual\_price) AS GMV

FROM target.orders o, target.date\_dimension dd

WHERE o.add\_date = dd.date\_key

GROUP BY dd.week\_num\_in\_year

ORDER BY dd.week\_num\_in\_year;



用户相关分析

# 查看全国各省市今年购买过的用户数，降序输出

SELECT ad.province, ad.city, COUNT(DISTINCT o.user\_key) AS totaluser

FROM target.orders o,

target.address\_dimension ad,

target.date\_dimension dd

WHERE o.add\_date = dd.date\_key

AND o.address\_key = ad.address\_key

AND dd.year = 2020

GROUP BY ad.province, ad.city

ORDER BY COUNT(DISTINCT o.user\_key) DESC;



# 查看用户消费金额（不考虑退货），降序输出

SELECT ud.username,

gender,

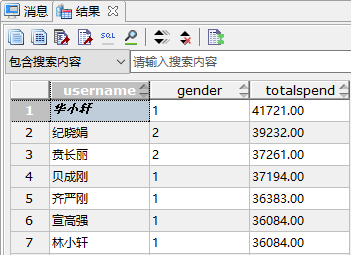
SUM(o.actual\_price) AS totalspend

FROM target.orders o, target.user\_dimension ud

WHERE o.user\_key = ud.user\_key

GROUP BY ud.username, ud.weixin\_openid, gender

ORDER BY SUM(o.actual\_price) DESC;



商品相关分析

# 查看各商品的总体销量（不考虑退货），降序输出

SELECT gd.goods\_name,

SUM(og.number) AS totalnum,

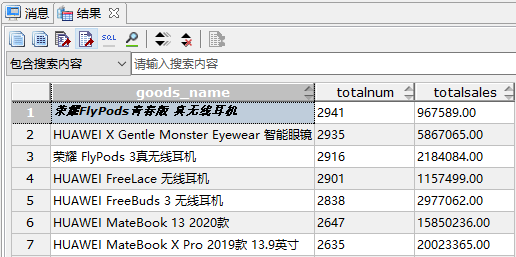
SUM(og.price \* og.number) AS totalsales

FROM target.order\_goods og

LEFT JOIN target.goods\_dimension gd ON og.goods\_key = gd.goods\_key

GROUP BY gd.goods\_name, gd.category\_name

ORDER BY SUM(og.number) DESC;



# 查看各商品的月度销量（不考虑退货）

SELECT gd.goods\_name, dd.month, SUM(og.number) AS monthnum

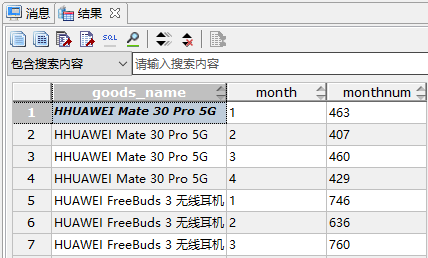
FROM target.order\_goods og

LEFT JOIN target.goods\_dimension gd ON og.goods\_key = gd.goods\_key

LEFT JOIN target.date\_dimension dd ON og.add\_date = dd.date\_key

GROUP BY gd.goods\_name,dd.month

ORDER BY gd.goods\_name,dd.month;



# 查看销量最高的前三个类目（仅从数量上考虑销量）

SELECT gd.category\_name, SUM(og.number) AS sum

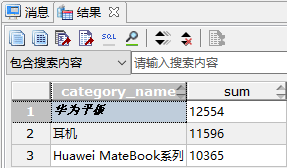
FROM target.order\_goods og

LEFT JOIN target.goods\_dimension gd ON og.goods\_key = gd.goods\_key

GROUP BY gd.category\_name

ORDER BY SUM(og.number) DESC

LIMIT 3;



## 实验小结

本实验主要完成了数据仓库建模并在DWS中实现表的创建，借助Kettle工具实现数据的迁移并在DWS中完成表的转换，最后基于历史数据实现对商城经营效益的分析。

# 附录

## 说明

此部分内容主要利用python实现ETL的功能。如果有兴趣的同学可以尝试以下内容。

相关说明：

1. 在public模式下创建相关的表（不加其他模式名的情况下，默认是创建在public模式下）；

2. 提前准备好python的环境（建议安装anaconda），安装psycopg2包，可以参考练习手册- 8 Python基本操作简介-数据库编程

### 数据库对象创建

创建日期维度表

DROP TABLE IF EXISTS date\_dimension;

CREATE TABLE Date\_Dimension (

date\_key integer NOT NULL,

full\_date date,

day\_of\_week tinyint,

day\_num\_in\_month tinyint,

day\_num\_overall smallint,

day\_name varchar(9),

day\_abbrev char(3),

weekday\_flag varchar(10),

week\_num\_in\_year tinyint,

week\_num\_overall smallint,

week\_begin\_date date,

week\_begin\_date\_key integer,

month tinyint,

month\_num\_overall smallint,

month\_name varchar(9),

month\_abbrev char(3),

quarter tinyint,

year smallint,

yearmo int,

fiscal\_month tinyint,

fiscal\_quarter tinyint,

fiscal\_year smallint,

last\_day\_in\_month\_flag varchar(15),

same\_day\_year\_ago\_date date,

PRIMARY KEY (date\_key)

)

DISTRIBUTE BY REPLICATION;

创建时间维度表

DROP TABLE IF EXISTS time\_dimension;

CREATE TABLE Time\_Dimension (

time\_key integer NOT NULL,

hour integer,

minute integer,

second integer,

t\_am\_pm char(2),

PRIMARY KEY (time\_key)

)

DISTRIBUTE BY REPLICATION;

创建商品维度表

DROP TABLE IF EXISTS goods\_dimension;

CREATE TABLE Goods\_Dimension (

goods\_key int NOT NULL,

goods\_id integer,

goods\_sn varchar(63),

goods\_name varchar(127),

category\_id integer,

category\_name varchar(63),

category\_keywords varchar(1023),

category\_desc varchar(255),

category\_pid integer,

brand\_id integer,

gallery varchar(1023),

keywords varchar(255),

brief varchar(1024),

is\_on\_sale tinyint,

sort\_order smallint,

pic\_url varchar(255),

share\_url varchar(1024),

is\_new tinyint,

is\_hot tinyint,

unit varchar(31),

counter\_price decimal(10, 2),

retail\_price decimal(10,2),

detail text,

product\_id integer,

specifications varchar(1023),

gp\_add\_date integer,

gp\_add\_time integer,

gp\_update\_date integer,

gp\_update\_time integer,

gp\_deleted tinyint,

category\_icon\_url varchar(1024),

category\_pic\_url varchar(1024),

category\_level varchar(255),

category\_sort\_order tinyint,

category\_add\_date integer,

category\_add\_time integer,

category\_update\_date integer,

category\_update\_time integer,

category\_deleted tinyint,

goods\_add\_date integer,

goods\_add\_time integer,

goods\_update\_date integer,

goods\_update\_time integer,

goods\_deleted tinyint,

PRIMARY KEY (goods\_key)

)

DISTRIBUTE BY REPLICATION;

创建用户维度表

DROP TABLE IF EXISTS user\_dimension;

CREATE TABLE User\_Dimension (

user\_key integer NOT NULL,

user\_id integer NOT NULL,

username varchar(63),

password varchar(63),

gender tinyint,

birthday integer,

last\_login\_date integer,

last\_login\_time integer,

last\_login\_ip varchar(63),

user\_level tinyint,

nickname varchar(63),

mobile varchar(20),

avatar varchar(255),

weixin\_openid varchar(63),

session\_key varchar(100),

status tinyint,

add\_date integer,

add\_time integer,

update\_date integer,

update\_time integer,

deleted tinyint,

PRIMARY KEY (user\_key)

)

DISTRIBUTE BY REPLICATION;

创建客户地址维度

DROP TABLE IF EXISTS Address\_Dimension;

CREATE TABLE Address\_Dimension (

address\_key integer NOT NULL,

address\_id integer NOT NULL,

province varchar(63),

city varchar(63),

county varchar(63),

address\_detail varchar(127),

area\_code char(6),

postal\_code char(6),

is\_default tinyint,

add\_date integer,

add\_time integer,

update\_date integer,

update\_time integer,

deleted tinyint,

PRIMARY KEY (address\_key)

)

DISTRIBUTE BY REPLICATION;

创建订单表

DROP TABLE IF EXISTS orders;

CREATE TABLE orders (

order\_id integer NOT NULL,

order\_sn varchar(63),

order\_status smallint,

user\_key integer,

address\_key integer,

goods\_price decimal(10, 2),

freight\_price decimal(10, 2),

coupon\_price decimal(10, 2),

integral\_price decimal(10, 2),

groupon\_price decimal(10, 2),

order\_price decimal(10, 2),

actual\_price decimal(10, 2),

pay\_id varchar(63),

pay\_date integer,

pay\_time integer,

ship\_sn varchar(63),

ship\_channel varchar(63),

ship\_date integer,

ship\_time integer,

confirm\_date integer,

confirm\_time integer,

message varchar(512),

comments smallint,

aftersale\_status smallint,

refund\_amount decimal(10, 2),

refund\_type varchar(63),

refund\_content varchar(127),

refund\_date integer,

refund\_time integer,

end\_date integer,

end\_time integer,

add\_date integer,

add\_time integer,

update\_date integer,

update\_time integer,

deleted tinyint,

PRIMARY KEY (order\_id)

)

DISTRIBUTE BY HASH (order\_id)

PARTITION BY RANGE (order\_id)

(

PARTITION p1 VALUES LESS THAN(20000),

PARTITION p2 VALUES LESS THAN(40000),

PARTITION p3 VALUES LESS THAN(60000),

PARTITION p4 VALUES LESS THAN(80000),

PARTITION p5 VALUES LESS THAN(MAXVALUE)

);

创建订单商品表

DROP TABLE IF EXISTS order\_goods;

CREATE TABLE order\_goods (

order\_id integer,

goods\_key integer,

number integer,

price decimal(10, 2),

comment\_value integer,

add\_date integer,

add\_time integer,

update\_date integer,

update\_time integer,

deleted tinyint,

PRIMARY KEY (goods\_key, order\_id)

)

DISTRIBUTE BY HASH(order\_id, goods\_key);

创建库存表

DROP TABLE IF EXISTS Inventory;

CREATE TABLE Inventory (

date\_key integer NOT NULL,

goods\_key integer NOT NULL,

number integer,

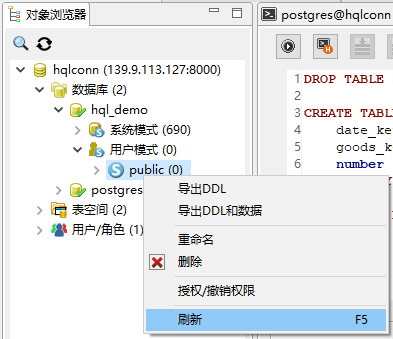
PRIMARY KEY (date\_key, goods\_key)

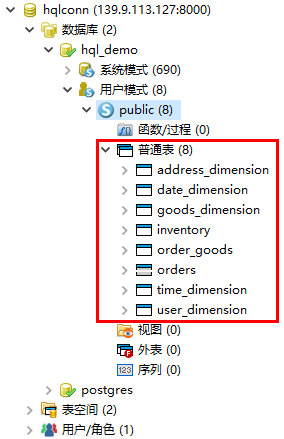
)

DISTRIBUTE BY HASH(date\_key, goods\_key);

查看创建的表

在Data Studio中，展开个人创建的数据库，例如本实验中的hql\_demo，在模式public上右键，点击“刷新”。





### 数据转换

说明：

1. 此部分会协助使用python完成数据的抽取-转换-导入（ETL）过程，因此需要提前安装好python相关环境。

2. 为简化操作，此部分已经将代码进行了封装，包含6个.py文件，但实验手册中仍然保留了详细代码，如有兴趣可以自行研究。

3. 实际过程中，其实可以直接使用python将转换后的数据导入到DWS中，但由于数据量比较多，耗时较长，采用的方案为：数据转换后先导出为csv文件，再在Data Studio中导入（具体过程可以参考date\_dimension表和time\_dimension表的数据导入）。

# 方式一：直接执行脚本（推荐）

使用Data Studio完成date\_dimension和time\_dimension的导入

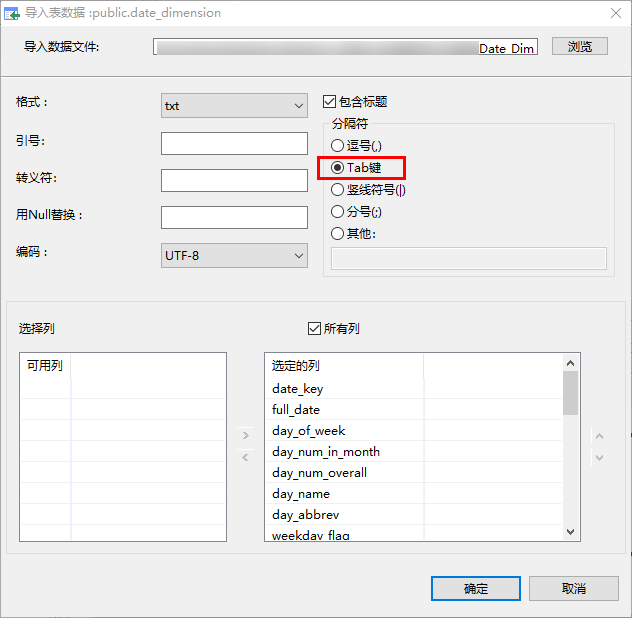
# 导入日期表date\_dimension

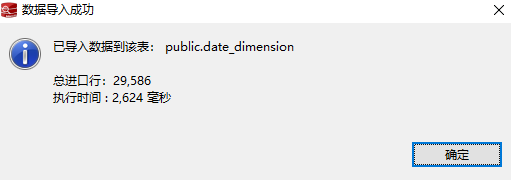
在Data Studio中找到date\_dimension表，右键 - 导入表数据。

点击浏览，找到date\_dimension.txt文件。

分隔符选择Tab键。

点击确定完成导入。





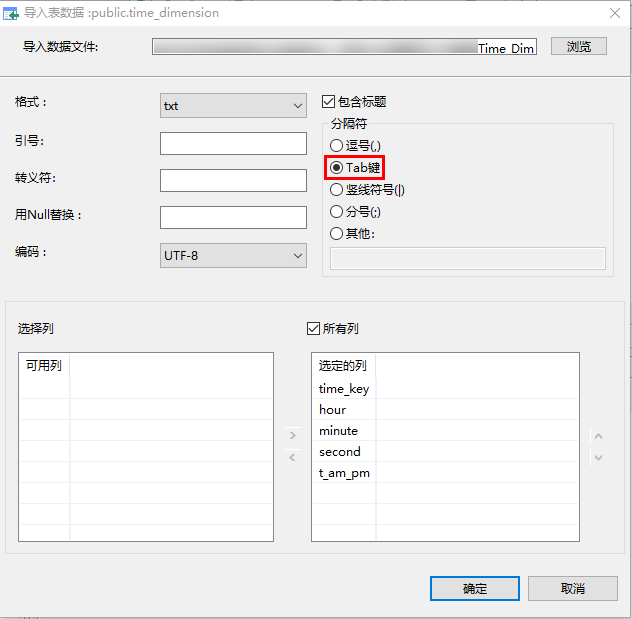
# 导入时间表time\_dimension

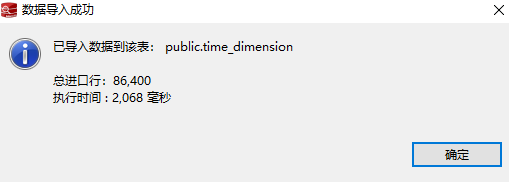
在Data Studio中找到time\_dimension表，右键 - 导入表数据。

点击浏览，找到time\_dimension.txt文件。

分隔符选择Tab键。

点击确定完成导入。





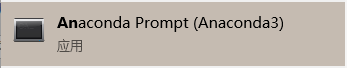
接下来用python完成其他表数据的转换。

准备工作：

准备好ETL脚本文件：例如本实验中，将ETL脚本文件存放在D盘根目录：D:\ETL\_part。

准备好RDS公网IP，端口号，root用户密码，DWS公网IP，端口号，dbadmin用户密码。

打开Anaconda Prompt。



**将1.py-6.py依次执行一遍，切记每次执行后要先将数据利用Data Studio导入到DWS中。**

语法格式：

python xx.py ip\_mysql port\_mysql database\_name\_mysql username\_mysql password\_mysql ip\_DWS port\_DWS database\_name\_DWS username\_DWS password\_DWS

注意需要替换为本次实验中的RDS及DWS相关信息。

实现goods\_dimension的数据转换

例如，在Anaconda Prompt中输入：

python D:\\ETL\_part\\1.py 121.36.99.22 3306 litemall root U2000server\_123 119.3.233.224 8000 hql\_demo dbadmin Dws@123456

返回结果：

Transforming gp\_add\_date now

Transforming gp\_update\_date now

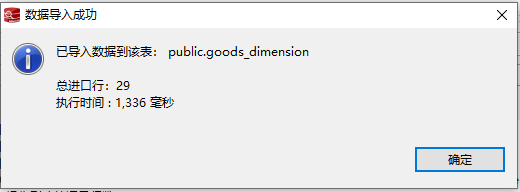
Transforming category\_add\_date now

Transforming category\_update\_date now

Transforming goods\_add\_date now

Transforming goods\_update\_date now

在D盘根目录下找到刚刚生成的数据文件goods\_dimension.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



实现address\_dimension的数据转换

例如，在Anaconda Prompt中输入：

python D:\\ETL\_part\\2.py 121.36.99.22 3306 litemall root U2000server\_123 119.3.233.224 8000 hql\_demo dbadmin Dws@123456

返回结果：

Transforming add\_date now

已经执行完1000行，还剩7002行

已经执行完2000行，还剩6002行

已经执行完3000行，还剩5002行

已经执行完4000行，还剩4002行

已经执行完5000行，还剩3002行

已经执行完6000行，还剩2002行

已经执行完7000行，还剩1002行

已经执行完8000行，还剩2行

Transforming update\_date now

已经执行完1000行，还剩7002行

已经执行完2000行，还剩6002行

已经执行完3000行，还剩5002行

已经执行完4000行，还剩4002行

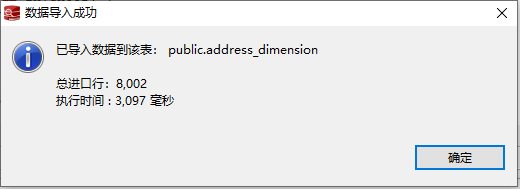
已经执行完5000行，还剩3002行

已经执行完6000行，还剩2002行

已经执行完7000行，还剩1002行

已经执行完8000行，还剩2行

在D盘根目录下找到刚刚生成的数据文件address\_dimension.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



实现user\_dimension的数据转换

例如，在Anaconda Prompt中输入：

python D:\\ETL\_part\\3.py 121.36.99.22 3306 litemall root U2000server\_123 119.3.233.224 8000 hql\_demo dbadmin Dws@123456

Transforming add\_date now

已经执行完1000行，还剩7002行

已经执行完2000行，还剩6002行

已经执行完3000行，还剩5002行

已经执行完4000行，还剩4002行

已经执行完5000行，还剩3002行

已经执行完6000行，还剩2002行

已经执行完7000行，还剩1002行

已经执行完8000行，还剩2行

Transforming update\_date now

已经执行完1000行，还剩7002行

已经执行完2000行，还剩6002行

已经执行完3000行，还剩5002行

已经执行完4000行，还剩4002行

已经执行完5000行，还剩3002行

已经执行完6000行，还剩2002行

已经执行完7000行，还剩1002行

已经执行完8000行，还剩2行

Transforming birthday now

已经执行完1000行，还剩7002行

已经执行完2000行，还剩6002行

已经执行完3000行，还剩5002行

已经执行完4000行，还剩4002行

已经执行完5000行，还剩3002行

已经执行完6000行，还剩2002行

已经执行完7000行，还剩1002行

已经执行完8000行，还剩2行

Transforming last\_login\_date now

已经执行完1000行，还剩7002行

已经执行完2000行，还剩6002行

已经执行完3000行，还剩5002行

已经执行完4000行，还剩4002行

已经执行完5000行，还剩3002行

已经执行完6000行，还剩2002行

已经执行完7000行，还剩1002行

已经执行完8000行，还剩2行

Transforming add\_date now

已经执行完1000行，还剩7002行

已经执行完2000行，还剩6002行

已经执行完3000行，还剩5002行

已经执行完4000行，还剩4002行

已经执行完5000行，还剩3002行

已经执行完6000行，还剩2002行

已经执行完7000行，还剩1002行

已经执行完8000行，还剩2行

Transforming update\_date now

已经执行完1000行，还剩7002行

已经执行完2000行，还剩6002行

已经执行完3000行，还剩5002行

已经执行完4000行，还剩4002行

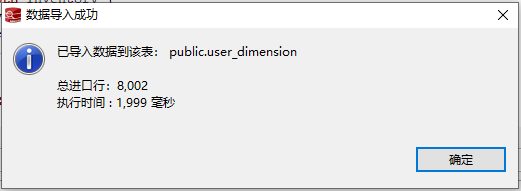
已经执行完5000行，还剩3002行

已经执行完6000行，还剩2002行

已经执行完7000行，还剩1002行

已经执行完8000行，还剩2行

在D盘根目录下找到刚刚生成的数据文件user\_dimension.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



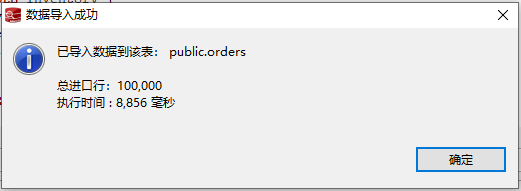
实现orders的数据转换

例如，在Anaconda Prompt中输入：

python D:\\ETL\_part\\4.py 121.36.99.22 3306 litemall root U2000server\_123 119.3.233.224 8000 hql\_demo dbadmin Dws@123456

因数据量略大，转换的时间较长且返回结果较多，这里不再详细贴出。

在D盘根目录下找到刚刚生成的数据文件orders.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



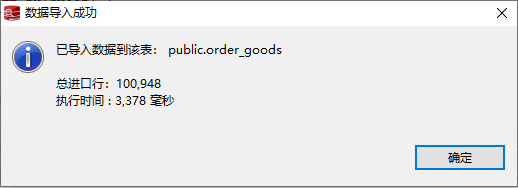
实现order\_goods的数据转换

例如，在Anaconda Prompt中输入：

python D:\\ETL\_part\\5.py 121.36.99.22 3306 litemall root U2000server\_123 119.3.233.224 8000 hql\_demo dbadmin Dws@123456

因数据量略大，转换的时间较长且返回结果较多，这里不再详细贴出。

在D盘根目录下找到刚刚生成的数据文件order\_goods.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。

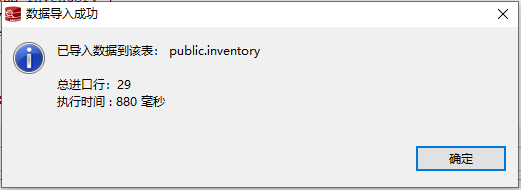


实现inventory的数据转换

例如，在Anaconda Prompt中输入：

python D:\\ETL\_part\\6.py 121.36.99.22 3306 litemall root U2000server\_123 119.3.233.224 8000 hql\_demo dbadmin Dws@123456

在D盘根目录下找到刚刚生成的数据文件inventory.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



# 方式二：自行拷贝代码执行转换（需要注意python当中的缩进）

使用Data Studio完成date\_dimension和time\_dimension的导入

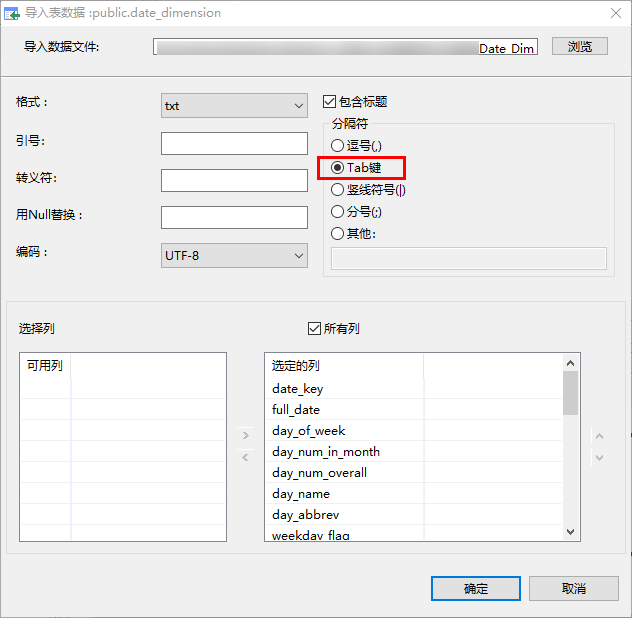
# 导入日期表date\_dimension

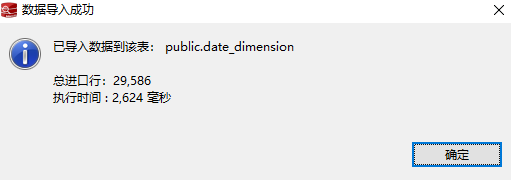
在Data Studio中找到date\_dimension表，右键 - 导入表数据。

点击浏览，找到date\_dimension.txt文件。

分隔符选择Tab键。

点击确定完成导入。





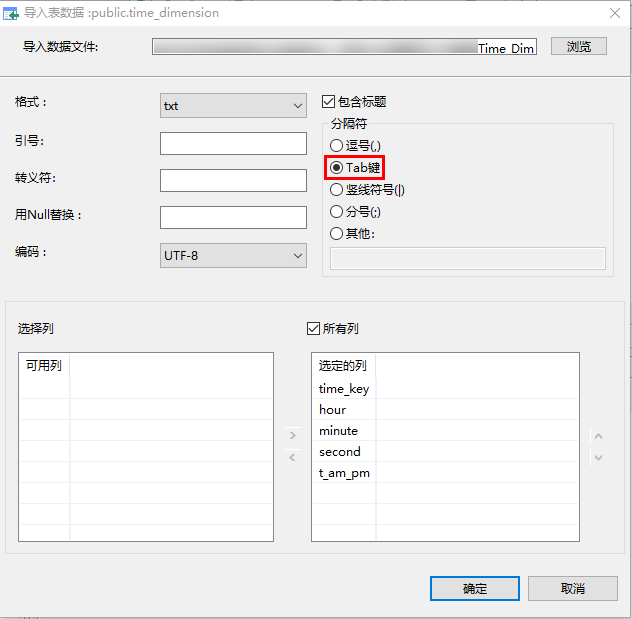
# 导入时间表time\_dimension

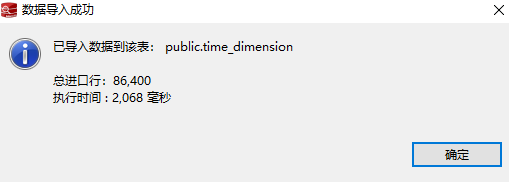
在Data Studio中找到time\_dimension表，右键 - 导入表数据。

点击浏览，找到time\_dimension.txt文件。

分隔符选择Tab键。

点击确定完成导入。





实现goods\_dimension的数据转换

在python中输入：

import psycopg2

import pymysql

import numpy as np

import pandas as pd

import re

import datetime

# psycopg2 和pymysql 的安装请见《随堂练习手册》：8.2.14-数据库编程 的内容

def get\_table(ip,sql,port=3306,database='litemall',user='root',password='\*\*\*\*\*\*'):

'''

连接数据库，操作相应的sql语句来进行查询。此处的*ip, port, database, user, password*为连接RDS的相关信息，请根据个人设置进行相应修改。

'''

if type(port)==int:

connection = pymysql.connect(host=ip, port=port, database=database, user=user, password=password, charset='utf8')

elif type(port)==str:

connection = psycopg2.connect(host=ip, port=port, database=database, user=user, password=password)

connection.set\_client\_encoding('utf-8')

cursor = connection.cursor()

cursor.execute(sql)

rows=cursor.fetchall()

cursor.close()

df = pd.DataFrame(list(rows),index=range(1,len(rows)+1))

connection.close()

return df

def find\_main\_key(dataframe,ndarray,column\_name,main\_key\_name):

'''

将时间戳转化为对应的主键

'''

main\_key = []

for i,ele in enumerate(ndarray):

if type(ele)==pd.\_libs.tslibs.timestamps.Timestamp:

mainkey = dataframe[dataframe[column\_name]==ele.floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

elif type(ele)==str or type(ele)==datetime.date:

mainkey = dataframe[dataframe[column\_name]==pd.to\_datetime(ele).floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

else:

#print('Unexpected Error!')

main\_key.append(ele)

if (i+1) % 1000 == 0:#每隔1000行输出一个提示信息

print('已经执行完%d行，还剩%d行'%(i+1,len(ndarray)-i-1))

return np.array(main\_key)

sql1 = '''SELECT lg.id AS goods\_id, lg.goods\_sn, lg.name AS goods\_name, category\_id, lc.name AS category\_name

, lc.keywords AS category\_keywords, lc.desc AS category\_desc, lc.pid AS category\_pid, lg.brand\_id, lg.gallery

, lg.keywords, lg.brief, lg.is\_on\_sale, lg.sort\_order, lg.pic\_url

, lg.share\_url, lg.is\_new, lg.is\_hot, lg.unit, lg.counter\_price

, lg.retail\_price, lg.detail, lgp.id AS product\_id, lgp.specifications, lgp.add\_time AS gp\_add\_time

, lgp.update\_time AS gp\_update\_time, lgp.deleted, lc.icon\_url AS category\_icon\_url, lc.pic\_url AS category\_pic\_url, lc.level AS category\_level

, lc.sort\_order AS category\_sort\_order, lc.add\_time AS category\_add\_time, lc.update\_time AS category\_update\_time, lc.deleted AS category\_deleted, lg.add\_time AS goods\_add\_time

, lg.update\_time AS goods\_update\_time, lg.deleted AS goods\_deleted

FROM litemall\_goods lg, litemall\_goods\_product lgp, litemall\_category lc

WHERE lg.category\_id = lc.id

AND lg.id = lgp.goods\_id;

'''

# Extracting：

df = get\_table('121.36.99.22',sql1) #调用get\_table函数，执行相应的sql语句完成查表操作。**此处121.36.99.22为RDS公网ip，请修改为个人相应的ip信息。**

#以下是一些正则化表达式的内容，主要是用来提取sql语句中的字段信息的；

#这一部分内容不是重点，有困难的学员可以直接复制粘贴相应的字段，保存到c这个列表中。

c = re.search('(SELECT[\s\S]\*FROM)',sql1).groups()[0]

c = re.search('(SELECT[\s\S]\*FROM)',sql1).groups()[0]

c = re.search('(AS[\s\S]\*FROM)',c).groups()[0]

c = re.sub('(la.[\w]\* AS)','',c)

c = re.sub('(lg.[\w]\* AS)','',c)

c = re.sub('(lc.[\w]\* AS)','',c)

c = re.sub('(lgp.[\w]\* AS)','',c)

c = re.sub('(lo.[\w]\* AS)','',c)

c = c.replace('AS','').replace('FROM','').replace('\n','').replace('\t','')\

.replace(' ','').replace('lo.','').replace('la.','').replace('time','date')\

.replace('lg.','').replace('lc.','').replace('lgp.','').split(',')

df.columns = c

#调用 get\_table 函数，查询 Date\_Dimension 到df2中：

sql\_datedim = 'select date\_key, full\_date from date\_dimension;'

df2 = get\_table('116.63.57.107',sql\_datedim,port='8000',database='hqltest',user='dbadmin', password='\*\*\*\*\*\*') #**此处的*ip=116.63.57.107, port=8000, database=hqltest, user=dbadmin, password=\*\*\*\*\*\**为连接DWS的相关信息，请根据个人设置进行相应修改。**

df2.columns=['date key','full date']

#Transforming：调用find\_main\_key方法完成转化工作，比如把 2020-04-01 0:00:00这样的时间转化为 20200401 这样的主键。

c1 = []

date\_index = []

date\_list = []

for i,ele in enumerate(c):

if ele[-4:]=='date' or ele[-3:]=='day':

c1.append(ele)

date\_index.append(i)

date\_list.append(find\_main\_key(df2,df[ele],'full date','date key'))

date\_list = np.array(date\_list)

#把新的数据添加到原始表中：

df.iloc[:,date\_index] = date\_list.T

df.iloc[:,date\_index[0]]=date\_list.T[:,0]

#修改一下字段名称，同时添加新的列名:

c\_new = c.copy()

i = 0

for index in date\_index:

c\_new[index] = c\_new[index].replace('\_date','\_time')

for index in date\_index:

if c[index][-3:] != 'day':

c\_new.insert(index+i,c[index])

i+=1

#把新的列用0填充:

df = df.reindex(columns=c\_new, fill\_value=0)

# Loading - 把数据导入到DWS中：先将数据导出成csv文件。

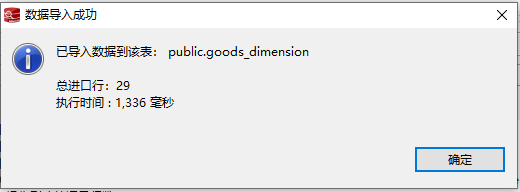
# t = np.concatenate((df.index.values.reshape(df.index.values.shape[0],1),df.values),1)

df.to\_csv('D:\\goods\_dimension.csv',index\_label='goods\_key',encoding='utf-8-sig')

--完毕—

# 使用Data Studio导入数据到DWS中。

在D盘根目录下找到刚刚生成的数据文件goods\_dimension.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



实现address\_dimension的数据转换

import psycopg2

import pymysql

import numpy as np

import pandas as pd

import re

import datetime

def get\_table(ip,sql,port=3306,database='litemall',user='root',password='\*\*\*\*\*\*'):

'''

连接数据库，操作相应的sql语句来进行查询。此处的*ip, port, database, user, password*为连接RDS的相关信息，请根据个人设置进行相应修改。

'''

if type(port)==int:

connection = pymysql.connect(host=ip, port=port, database=database, user=user, password=password, charset='utf8')

elif type(port)==str:

connection = psycopg2.connect(host=ip, port=port, database=database, user=user, password=password)

connection.set\_client\_encoding('utf-8')

cursor = connection.cursor()

cursor.execute(sql)

rows=cursor.fetchall()

cursor.close()

df = pd.DataFrame(list(rows),index=range(1,len(rows)+1))

connection.close()

return df

def find\_main\_key(dataframe,ndarray,column\_name,main\_key\_name):

'''

将时间戳转化为对应的主键

'''

main\_key = []

for ele in ndarray:

if type(ele)==pd.\_libs.tslibs.timestamps.Timestamp:

mainkey = dataframe[dataframe[column\_name]==ele.floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

elif type(ele)==str or type(ele)==datetime.date:

mainkey = dataframe[dataframe[column\_name]==pd.to\_datetime(ele).floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

else:

#print('Unexpected Error!')

main\_key.append(ele)

return np.array(main\_key)

sql2 = '''SELECT id AS address\_id, province, city, county, address\_detail

, area\_code, postal\_code, is\_default, add\_time, update\_time

, deleted

FROM litemall\_address

'''

#Extracting：

df = get\_table('121.36.99.22',sql2) #调用get\_table函数，执行相应的sql语句完成查表操作。**此处121.36.99.22为RDS公网ip，请修改为个人相应的ip信息。**

#以下是一些正则化表达式的内容，主要是用来提取sql语句中的字段信息的；

#这一部分内容不是重点，有困难的学员可以直接复制粘贴相应的字段，保存到c这个列表中。

c = re.search('(SELECT[\s\S]\*FROM)',sql2).groups()[0]

c = re.search('(SELECT[\s\S]\*FROM)',sql2).groups()[0]

c = re.search('(AS[\s\S]\*FROM)',c).groups()[0]

c = re.sub('(la.[\w]\* AS)','',c)

c = re.sub('(lg.[\w]\* AS)','',c)

c = re.sub('(lc.[\w]\* AS)','',c)

c = re.sub('(lgp.[\w]\* AS)','',c)

c = re.sub('(lo.[\w]\* AS)','',c)

c = c.replace('AS','').replace('FROM','').replace('\n','').replace('\t','')\

.replace(' ','').replace('lo.','').replace('la.','').replace('time','date')\

.replace('lg.','').replace('lc.','').replace('lgp.','').split(',')

df.columns = c

#调用 get\_table 函数，查询 Date\_Dimension 到df2中：

sql\_datedim = 'select date\_key, full\_date from date\_dimension;'

df2 = get\_table('116.63.57.107',sql\_datedim,port='8000',database='hqltest',user='dbadmin', password='\*\*\*\*\*\*') #**此处的*ip=116.63.57.107, port=8000, database=hqltest, user=dbadmin, password=\*\*\*\*\*\**为连接DWS的相关信息，请根据个人设置进行相应修改。**

df2.columns=['date key','full date']

#Transforming：调用find\_main\_key方法完成转化工作，比如把 2020-04-01 0:00:00这样的时间转化为 20200401 这样的主键。

c1 = []

date\_index = []

date\_list = []

for i,ele in enumerate(c):

if ele[-4:]=='date' or ele[-3:]=='day':

c1.append(ele)

date\_index.append(i)

date\_list.append(find\_main\_key(df2,df[ele],'full date','date key'))

date\_list = np.array(date\_list)

#把新的数据添加到原始表中：

df.iloc[:,date\_index] = date\_list.T

df.iloc[:,date\_index[0]]=date\_list.T[:,0]

#修改一下字段名称，同时添加新的列名:

c\_new = c.copy()

i = 0

for index in date\_index:

c\_new[index] = c\_new[index].replace('\_date','\_time')

for index in date\_index:

if c[index][-3:] != 'day':

c\_new.insert(index+i,c[index])

i+=1

#把新的列用0填充:

df = df.reindex(columns=c\_new, fill\_value=0)

#Loading-把数据导入到DWS中：先将数据导出成csv文件。

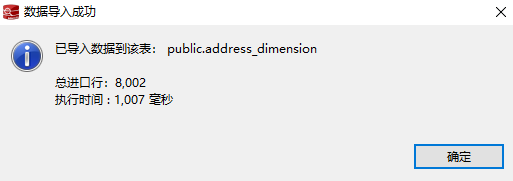
# t = np.concatenate((df.index.values.reshape(df.index.values.shape[0],1),df.values),1)

df.to\_csv('D:\\address\_dimension.csv',index\_label='address\_key',encoding='utf-8-sig')

--完毕—

# 使用Data Studio导入数据到DWS中。

在D盘根目录下找到刚刚生成的数据文件address\_dimension.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



实现user\_dimension的数据转换

import psycopg2

import pymysql

import numpy as np

import pandas as pd

import re

import datetime

def get\_table(ip,sql,port=3306,database='litemall',user='root',password='\*\*\*\*\*\*'):

'''

连接数据库，操作相应的sql语句来进行查询。此处的*ip, port, database, user, password*为连接RDS的相关信息，请根据个人设置进行相应修改。

'''

if type(port)==int:

connection = pymysql.connect(host=ip, port=port, database=database, user=user, password=password, charset='utf8')

elif type(port)==str:

connection = psycopg2.connect(host=ip, port=port, database=database, user=user, password=password)

connection.set\_client\_encoding('utf-8')

cursor = connection.cursor()

cursor.execute(sql)

rows=cursor.fetchall()

cursor.close()

df = pd.DataFrame(list(rows),index=range(1,len(rows)+1))

connection.close()

return df

def find\_main\_key(dataframe,ndarray,column\_name,main\_key\_name):

'''

将时间戳转化为对应的主键

'''

main\_key = []

for ele in ndarray:

if type(ele)==pd.\_libs.tslibs.timestamps.Timestamp:

mainkey = dataframe[dataframe[column\_name]==ele.floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

elif type(ele)==str or type(ele)==datetime.date:

mainkey = dataframe[dataframe[column\_name]==pd.to\_datetime(ele).floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

else:

#print('Unexpected Error!')

main\_key.append(ele)

return np.array(main\_key)

sql3='''SELECT id AS user\_id, username, password, gender, birthday

, last\_login\_time, last\_login\_ip, user\_level, nickname, mobile

, avatar, weixin\_openid, session\_key, status, add\_time

, update\_time, deleted

FROM litemall\_user;

'''

#Extracting：

df = get\_table('121.36.99.22',sql3) #调用get\_table函数，执行相应的sql语句完成查表操作。**此处121.36.99.22为RDS公网ip，请修改为个人相应的ip信息。**

#以下是一些正则化表达式的内容，主要是用来提取sql语句中的字段信息的；

#这一部分内容不是重点，有困难的学员可以直接复制粘贴相应的字段，保存到c这个列表中。

c = re.search('(SELECT[\s\S]\*FROM)',sql3).groups()[0]

c = re.search('(SELECT[\s\S]\*FROM)',sql3).groups()[0]

c = re.search('(AS[\s\S]\*FROM)',c).groups()[0]

c = re.sub('(la.[\w]\* AS)','',c)

c = re.sub('(lg.[\w]\* AS)','',c)

c = re.sub('(lc.[\w]\* AS)','',c)

c = re.sub('(lgp.[\w]\* AS)','',c)

c = re.sub('(lo.[\w]\* AS)','',c)

c = c.replace('AS','').replace('FROM','').replace('\n','').replace('\t','')\

.replace(' ','').replace('lo.','').replace('la.','').replace('time','date')\

.replace('lg.','').replace('lc.','').replace('lgp.','').split(',')

df.columns = c

#调用 get\_table 函数，查询 Date\_Dimension 到df2中：

sql\_datedim = 'select date\_key, full\_date from date\_dimension;'

df2 = get\_table('116.63.57.107',sql\_datedim,port='8000',database='hqltest',user='dbadmin', password='\*\*\*\*\*\*') #**此处的ip=116.63.57.107, port=8000, database=hqltest, user=dbadmin, password=\*\*\*\*\*\*为连接DWS的相关信息，请根据个人设置进行相应修改。**

df2.columns=['date key','full date']

#Transforming：调用find\_main\_key方法完成转化工作，比如把 2020-04-01 0:00:00这样的时间转化为 20200401 这样的主键。

c1 = []

date\_index = []

date\_list = []

for i,ele in enumerate(c):

if ele[-4:]=='date' or ele[-3:]=='day':

c1.append(ele)

date\_index.append(i)

date\_list.append(find\_main\_key(df2,df[ele],'full date','date key'))

date\_list = np.array(date\_list)

#把新的数据添加到原始表中：

df.iloc[:,date\_index] = date\_list.T

df.iloc[:,date\_index[0]]=date\_list.T[:,0]

#修改一下字段名称，同时添加新的列名:

c\_new = c.copy()

i = 0

for index in date\_index:

c\_new[index] = c\_new[index].replace('\_date','\_time')

for index in date\_index:

if c[index][-3:] != 'day':

c\_new.insert(index+i,c[index])

i+=1

#把新的列用0填充:

df = df.reindex(columns=c\_new, fill\_value=0)

#Loading-把数据导入到DWS中：先将数据导出成csv文件。

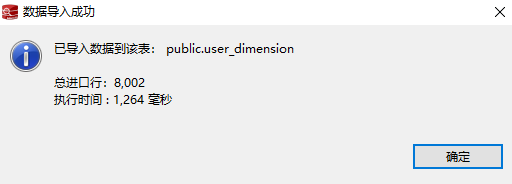
# t = np.concatenate((df.index.values.reshape(df.index.values.shape[0],1),df.values),1)

df.to\_csv('D:\\user\_dimension.csv',index\_label='user\_key',encoding='utf-8-sig')

--完毕--

# 使用Data Studio导入数据到DWS中。

在D盘根目录下找到刚刚生成的数据文件user\_dimension.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



实现orders的数据转换

import psycopg2

import pymysql

import numpy as np

import pandas as pd

import re

import datetime

def get\_table(ip,sql,port=3306,database='litemall',user='root',password='\*\*\*\*\*\*'):

'''

连接数据库，操作相应的sql语句来进行查询。此处的*ip, port, database, user, password*为连接RDS的相关信息，请根据个人设置进行相应修改。

'''

if type(port)==int:

connection = pymysql.connect(host=ip, port=port, database=database, user=user, password=password, charset='utf8')

elif type(port)==str:

connection = psycopg2.connect(host=ip, port=port, database=database, user=user, password=password)

connection.set\_client\_encoding('utf-8')

cursor = connection.cursor()

cursor.execute(sql)

rows=cursor.fetchall()

cursor.close()

df = pd.DataFrame(list(rows),index=range(1,len(rows)+1))

connection.close()

return df

def find\_main\_key(dataframe,ndarray,column\_name,main\_key\_name):

'''

将时间戳转化为对应的主键

'''

main\_key = []

for i,ele in enumerate(ndarray):

if type(ele)==pd.\_libs.tslibs.timestamps.Timestamp:

mainkey = dataframe[dataframe[column\_name]==ele.floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

elif type(ele)==str or type(ele)==datetime.date:

mainkey = dataframe[dataframe[column\_name]==pd.to\_datetime(ele).floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

else:

#print('Unexpected Error!')

main\_key.append(ele)

if (i+1) % 1000 == 0:#每隔1000行输出一个提示信息

print('已经执行完%d行，还剩%d行'%(i+1,len(ndarray)-i-1))

return np.array(main\_key)

sql4 = '''SELECT lo.id AS order\_id, order\_sn, order\_status, lo.user\_id, la.id AS address\_id

, goods\_price, freight\_price, coupon\_price, integral\_price, groupon\_price

, order\_price, actual\_price, pay\_id, pay\_time, ship\_sn

, ship\_channel, ship\_time, confirm\_time, message, comments

, aftersale\_status, refund\_amount, refund\_type, refund\_content, refund\_time

, end\_time, lo.add\_time, lo.update\_time, lo.deleted

FROM litemall\_order lo, litemall\_address la

WHERE lo.user\_id = la.user\_id'''

#Extracting：

df = get\_table('121.36.99.22',sql4) #调用get\_table函数，执行相应的sql语句完成查表操作。**此处121.36.99.22为RDS公网ip，请修改为个人相应的ip信息。**

#以下是一些正则化表达式的内容，主要是用来提取sql语句中的字段信息的；

#这一部分内容不是重点，有困难的学员可以直接复制粘贴相应的字段，保存到c这个列表中。

c = re.search('(SELECT[\s\S]\*FROM)',sql4).groups()[0]

c = re.search('(AS[\s\S]\*FROM)',c).groups()[0]

c = re.sub('(la.[\w]\* AS)','',c)

c = re.sub('(lg.[\w]\* AS)','',c)

c = re.sub('(lc.[\w]\* AS)','',c)

c = re.sub('(lgp.[\w]\* AS)','',c)

c = re.sub('(lo.[\w]\* AS)','',c)

c = c.replace('AS','').replace('FROM','').replace('\n','').replace('\t','')\

.replace(' ','').replace('lo.','').replace('la.','').replace('time','date')\

.replace('lg.','').replace('lc.','').replace('lgp.','').split(',')

df.columns = c

#调用 get\_table 函数，查询 Date\_Dimension 到df2中：

sql\_datedim = 'select date\_key, full\_date from date\_dimension;'

df2 = get\_table('116.63.57.107',sql\_datedim,port='8000',database='hqltest',user='dbadmin', password='\*\*\*\*\*\*') #**此处的*ip=116.63.57.107, port=8000, database=hqltest, user=dbadmin, password=\*\*\*\*\*\**为连接DWS的相关信息，请根据个人设置进行相应修改。**

df2.columns=['date key','full date']

#Transforming：调用find\_main\_key方法完成转化工作，比如把 2020-04-01 0:00:00这样的时间转化为 20200401 这样的主键。

c1 = []

date\_index = []

date\_list = []

for i,ele in enumerate(c):

if ele[-4:]=='date' or ele[-3:]=='day':

c1.append(ele)

date\_index.append(i)

date\_list.append(find\_main\_key(df2,df[ele],'full date','date key'))

date\_list = np.array(date\_list)

#把新的数据添加到原始表中：

df.iloc[:,date\_index] = date\_list.T

df.iloc[:,date\_index[0]]=date\_list.T[:,0]

#修改一下字段名称，同时添加新的列名:

c\_new = c.copy()

i = 0

for index in date\_index:

c\_new[index] = c\_new[index].replace('\_date','\_time')

for index in date\_index:

if c[index][-3:] != 'day':

c\_new.insert(index+i,c[index])

i+=1

#把新的列用0填充:

df = df.reindex(columns=c\_new, fill\_value=0)

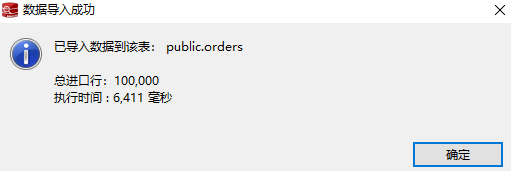
#Loading-把表添加到DWS中：先将数据导出成csv文件。

df.to\_csv('D:\\orders.csv',index=False,encoding='utf-8-sig')

--完毕--

# 使用Data Studio导入数据到DWS中。

在D盘根目录下找到刚刚生成的数据文件orders.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



实现order\_goods的数据转换

import psycopg2

import pymysql

import numpy as np

import pandas as pd

import re

import datetime

def get\_table(ip,sql,port=3306,database='litemall',user='root',password='\*\*\*\*\*\*'):

'''

连接数据库，操作相应的sql语句来进行查询。此处的*ip, port, database, user, password*为连接RDS的相关信息，请根据个人设置进行相应修改。

'''

if type(port)==int:

connection = pymysql.connect(host=ip, port=port, database=database, user=user, password=password, charset='utf8')

elif type(port)==str:

connection = psycopg2.connect(host=ip, port=port, database=database, user=user, password=password)

connection.set\_client\_encoding('utf-8')

cursor = connection.cursor()

cursor.execute(sql)

rows=cursor.fetchall()

cursor.close()

df = pd.DataFrame(list(rows),index=range(1,len(rows)+1))

connection.close()

return df

def find\_main\_key(dataframe,ndarray,column\_name,main\_key\_name):

'''

将时间戳转化为对应的主键

'''

main\_key = []

for ele in ndarray:

if type(ele)==pd.\_libs.tslibs.timestamps.Timestamp:

mainkey = dataframe[dataframe[column\_name]==ele.floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

elif type(ele)==str or type(ele)==datetime.date:

mainkey = dataframe[dataframe[column\_name]==pd.to\_datetime(ele).floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

else:

#print('Unexpected Error!')

main\_key.append(ele)

return np.array(main\_key)

def find\_main\_key2(dataframe,ndarray,column\_name,main\_key\_name):

'''

将值转化为对应的主键

'''

main\_key=[]

for ele in ndarray:

con = dataframe.loc[dataframe[column\_name]==ele,main\_key\_name].values

if len(con)==0:

print(ele)

main\_key.append(con[0])

#main\_key = [dataframe.loc[dataframe[column\_name]==ele,main\_key\_name].values[0] for ele in ndarray]

return np.array(main\_key)

sql5='''SELECT order\_id, product\_id, number, price, comment

, add\_time, update\_time, deleted

FROM litemall\_order\_goods

'''

#Extracting：

df = get\_table('121.36.99.22',sql5)#调用get\_table函数，执行相应的sql语句完成查表操作。**此处121.36.99.22为RDS公网ip，请修改为个人相应的ip信息。**

#以下是一些正则化表达式的内容，主要是用来提取sql语句中的字段信息的；

#这一部分内容不是重点，有困难的学员可以直接复制粘贴相应的字段，保存到c这个列表中。

c = re.search('(SELECT[\s\S]\*FROM)',sql5).groups()[0]

c = re.sub('(la.[\w]\* AS)','',c)

c = re.sub('(lg.[\w]\* AS)','',c)

c = re.sub('(lc.[\w]\* AS)','',c)

c = re.sub('(lgp.[\w]\* AS)','',c)

c = re.sub('(lo.[\w]\* AS)','',c)

c = c.replace('AS','').replace('FROM','').replace('\n','').replace('\t','')\

.replace(' ','').replace('lo.','').replace('la.','').replace('time','date')\

.replace('lg.','').replace('lc.','').replace('lgp.','').replace('SELECT','').split(',')

df.columns = c

#从DWS里读取goods\_dimension里的goods\_key 和 product\_id，以便通过product\_id关联goods\_key：

df3 = get\_table('116.63.57.107','SELECT goods\_key, product\_id FROM goods\_dimension',port='8000',database='hqltest',user='dbadmin', password='\*\*\*\*\*\*') #**此处的*ip=116.63.57.107, port=8000, database=hqltest, user=dbadmin, password=\*\*\*\*\*\**为连接DWS的相关信息，请根据个人设置进行相应修改。**

fields = 'goods\_key,product\_id'

df3.columns = fields.split(',')

#通过product\_id关联goods\_key：

goods\_key = find\_main\_key2(df3,df['product\_id'],'product\_id','goods\_key')

#把product\_id替换为goods\_key：

df['product\_id']=goods\_key

c[c.index('product\_id')]='goods\_key'

c[c.index('comment')]='comment\_value'

#修改字段名称：

df.columns = c

#调用 get\_table 函数，查询 Date\_Dimension 到df2中：

sql\_datedim = 'select date\_key, full\_date from date\_dimension;'

df2 = get\_table('116.63.57.107',sql\_datedim,port='8000',database='hqltest',user='dbadmin', password='\*\*\*\*\*\*') #**此处的*ip=116.63.57.107, port=8000, database=hqltest, user=dbadmin, password=\*\*\*\*\*\**为连接DWS的相关信息，请根据个人设置进行相应修改。**

df2.columns=['date key','full date']

#Transforming：调用find\_main\_key方法完成转化工作，比如把 2020-04-01 0:00:00这样的时间转化为 20200401 这样的主键。

c1 = []

date\_index = []

date\_list = []

for i,ele in enumerate(c):

if ele[-4:]=='date' or ele[-3:]=='day':

c1.append(ele)

date\_index.append(i)

date\_list.append(find\_main\_key(df2,df[ele],'full date','date key'))

date\_list = np.array(date\_list)

#把新的数据添加到原始表中：

df.iloc[:,date\_index] = date\_list.T

df.iloc[:,date\_index[0]]=date\_list.T[:,0]

#修改一下字段名称，同时添加新的列名:

c\_new = c.copy()

i = 0

for index in date\_index:

c\_new[index] = c\_new[index].replace('\_date','\_time')

for index in date\_index:

if c[index][-3:] != 'day':

c\_new.insert(index+i,c[index])

i+=1

#把新的列用0填充:

df = df.reindex(columns=c\_new, fill\_value=0)

#order\_id和goods\_key是新表的主键，数据库不允许主键重复，因此删除重复的数据，把购买量number修改为2：

duplicated\_index = df.iloc[:,:2].duplicated()

duplicated\_data = df.iloc[duplicated\_index.values,:2]

dff = df.drop\_duplicates(subset=['order\_id','goods\_key'], keep='first', inplace=False)

for ele in duplicated\_data.values[:,0]:

dff.loc[dff['order\_id']==ele,'number']=2

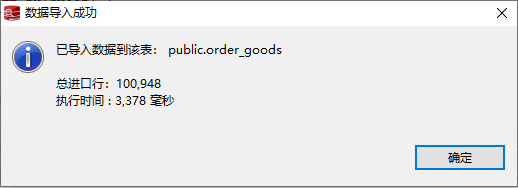
#Loading-把表添加到DWS中：先将数据导出成csv文件。

dff.to\_csv('D:\\order\_goods.csv',index=False,encoding='utf-8-sig')

--完--

# 使用Data Studio导入数据到DWS中。

在D盘根目录下找到刚刚生成的数据文件order\_goods.csv，在Data Studio中完成导入即可，**注意，导入时分隔符选择“逗号”**。



实现inventory的数据转换

import psycopg2

import pymysql

import numpy as np

import pandas as pd

import datetime

def get\_table(ip,sql,port=3306,database='litemall',user='root',password='\*\*\*\*\*\*'):

'''

连接数据库，操作相应的sql语句来进行查询。此处的*ip, port, database, user, password*为连接RDS的相关信息，请根据个人设置进行相应修改。

'''

if type(port)==int:

connection = pymysql.connect(host=ip, port=port, database=database, user=user, password=password, charset='utf8')

elif type(port)==str:

connection = psycopg2.connect(host=ip, port=port, database=database, user=user, password=password)

connection.set\_client\_encoding('utf-8')

cursor = connection.cursor()

cursor.execute(sql)

rows=cursor.fetchall()

cursor.close()

df = pd.DataFrame(list(rows),index=range(1,len(rows)+1))

connection.close()

return df

def find\_main\_key(dataframe,ndarray,column\_name,main\_key\_name):

'''

将时间戳转化为对应的主键

'''

main\_key = []

for ele in ndarray:

if type(ele)==pd.\_libs.tslibs.timestamps.Timestamp:

mainkey = dataframe[dataframe[column\_name]==ele.floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

elif type(ele)==str or type(ele)==datetime.date:

mainkey = dataframe[dataframe[column\_name]==pd.to\_datetime(ele).floor('d')][main\_key\_name].values

main\_key.append(mainkey[0])

else:

#print('Unexpected Error!')

main\_key.append(ele)

return np.array(main\_key)

def find\_main\_key2(dataframe,ndarray,column\_name,main\_key\_name):

'''

将值转化为对应的主键

'''

main\_key = [dataframe.loc[dataframe[column\_name]==ele,main\_key\_name].values for ele in ndarray]

return np.array(main\_key)

sql6 = '''SELECT update\_time, id AS product\_id, number

FROM litemall\_goods\_product

'''

#Extracting：

df = get\_table('121.36.99.22',sql6)#调用get\_table函数，执行相应的sql语句完成查表操作。**此处121.36.99.22为RDS公网ip，请修改为个人相应的ip信息。**

#指定字段名称：

c = ['update\_time','product\_id','number']

df.columns = c

#从DWS里读取goods\_dimension里的goods\_key 和 product\_id，以便通过product\_id关联goods\_key：

df3 = get\_table('116.63.57.107','SELECT goods\_key, product\_id FROM goods\_dimension',port='8000',database='hqltest',user='dbadmin', password='\*\*\*\*\*\*') #**此处的*ip=116.63.57.107, port=8000, database=hqltest, user=dbadmin, password=\*\*\*\*\*\**为连接DWS的相关信息，请根据个人设置进行相应修改。**

fields = 'goods\_key,product\_id'

df3.columns = fields.split(',')

#通过product\_id关联goods\_key：

goods\_key = find\_main\_key2(df3,df['product\_id'],'product\_id','goods\_key')

df['product\_id']=goods\_key

#调用 get\_table 函数，查询 Date\_Dimension 到df2中：

sql\_datedim = 'select date\_key, full\_date from date\_dimension;'

df2 = get\_table('116.63.57.107',sql\_datedim,port='8000',database='hqltest',user='dbadmin', password='\*\*\*\*\*\*') #**此处的*ip=116.63.57.107, port=8000, database=hqltest, user=dbadmin, password=\*\*\*\*\*\**为连接DWS的相关信息，请根据个人设置进行相应修改。**

df2.columns=['date key','full date']

#Transforming：调用find\_main\_key方法完成转化工作，比如把 2020-04-01 0:00:00这样的时间转化为 20200401 这样的主键。

df['update\_time'] = find\_main\_key(df2,df['update\_time'],'full date','date key')

#修改字段名称：

c[c.index('product\_id')]='goods\_key'

c[c.index('update\_time')]='date\_key'

df.columns = c

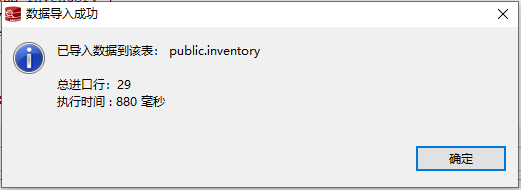
#Loading-把表添加到DWS中：先将数据导出成csv文件。

df.to\_csv('D:\\inventory.csv',index=False,encoding='utf-8-sig')

--完--

# 使用Data Studio导入数据到DWS中。

在D盘根目录下找到刚刚生成的数据文件inventory.csv，在Data Studio中完成导入即可**注意，导入时分隔符选择“逗号”**。



### 数据分析

此处，直接在DWS集群中进行分析。说明，以下语句中，下划线处已做挖空处理，需要自行填入。

GMV (Gross Merchandise Volume = 销售额 + 取消订单金额 + 拒收订单金额 + 退货订单金额) 相关分析。提示：只要产生过订单就算是GMV的一部分，所以直接统计所有的订单即可，不用单独考虑取消、拒收、退货等。

# 查看2020年3月各省GMV，降序输出

SELECT ad.province AS province, SUM(o.actual\_price) AS GMV

FROM orders o, address\_dimension ad, date\_dimension dd

WHERE o.address\_key = ad.address\_key

AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

ORDER BY SUM(o.actual\_price) DESC;

结果：



# 查看全国各省市GMV，时间范围为年初到今天（此处考虑到4个直辖市的信息与其他各省的信息不在一个字段，所以将查询结果做合并处理）

SELECT ad.province AS province, ad.city AS city, SUM(o.actual\_price) AS GMV

FROM orders o, address\_dimension ad, date\_dimension dd

WHERE o.address\_key = ad.address\_key

AND o.add\_date = dd.date\_key

AND to\_date(dd.full\_date) BETWEEN to\_date('2020/1/1') AND current\_date

AND province not in ('北京市', '上海市', '天津市', '重庆市')

GROUP BY ad.province, ad.city

UNION ALL

SELECT ad.province AS province, ad.county AS city, SUM(o.actual\_price) AS GMV

FROM orders o, address\_dimension ad, date\_dimension dd

WHERE o.address\_key = ad.address\_key

AND o.add\_date = dd.date\_key

AND to\_date(dd.full\_date) BETWEEN to\_date('2020/1/1') AND current\_date

AND province in ('北京市', '上海市', '天津市', '重庆市')

GROUP BY ad.province, ad.county;

结果：



# 查看每周的总体GMV

SELECT dd.week\_num\_in\_year, SUM(o.actual\_price) AS GMV

FROM orders o, date\_dimension dd

WHERE o.add\_date = dd.date\_key

GROUP BY dd.week\_num\_in\_year

ORDER BY dd.week\_num\_in\_year;

结果：



用户相关分析

# 查看全国各省市今年购买过的用户数，降序输出

SELECT ad.province, ad.city, COUNT(DISTINCT o.user\_key) AS totaluser

FROM orders o, address\_dimension ad, date\_dimension dd

WHERE o.add\_date = dd.date\_key

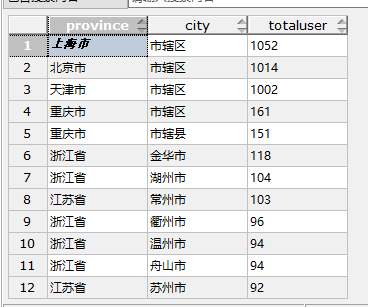
AND o.address\_key = ad.address\_key

AND dd.year = 2020

GROUP BY ad.province, ad.city

ORDER BY COUNT(DISTINCT o.user\_key) DESC;

结果：



# 查看用户消费金额（不考虑退货），降序输出

SELECT ud.user\_key, ud.username, gender, SUM(o. actual\_price) AS totalspend

FROM orders o, user\_dimension ud

WHERE o.user\_key = ud.user\_key

GROUP BY ud.user\_key, ud.username, gender

ORDER BY sum(o.actual\_price) DESC;

结果：



商品相关分析

# 查看各商品的总体销量（不考虑退货），降序输出

SELECT gd.goods\_sn, gd.goods\_name, SUM(og.number) AS totalnum, SUM(og.price\*og.number) AS totalsales

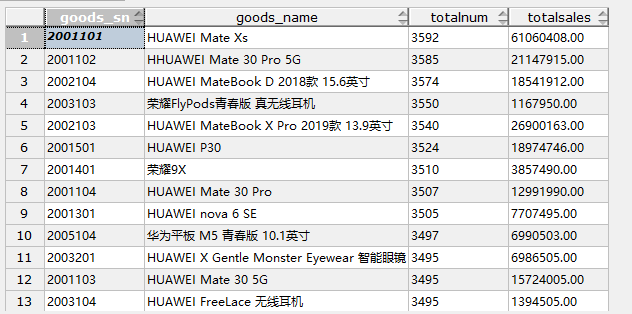
FROM order\_goods og

LEFT JOIN goods\_dimension gd ON og.goods\_key = gd.goods\_key

GROUP BY gd.goods\_sn, gd.goods\_name, gd.category\_name

ORDER BY SUM(og.number) DESC;

结果：



# 查看各商品的月度销量（不考虑退货）

SELECT gd.goods\_sn, gd.goods\_name, dd.month, SUM(og.number) AS monthnum

FROM order\_goods og

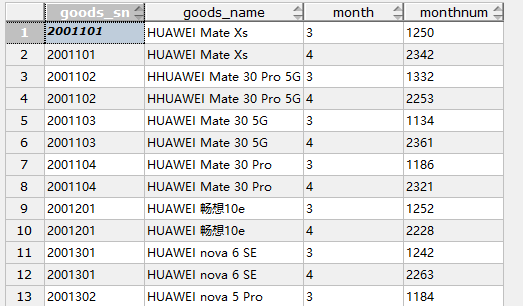
LEFT JOIN goods\_dimension gd ON og.goods\_key = gd.goods\_key

LEFT JOIN date\_dimension dd ON og.add\_date = dd.date\_key

GROUP BY gd.goods\_sn, gd.goods\_name, dd.month

ORDER BY gd.goods\_sn, dd.month;

结果：



# 查看销量最高的前三个类目（仅从数量上考虑销量）

SELECT gd.category\_name, SUM(og.number) AS sum

FROM order\_goods og

LEFT JOIN goods\_dimension gd ON og.goods\_key = gd.goods\_key

GROUP BY gd.category\_name

ORDER BY SUM(og.number) DESC

LIMIT 3;

结果：

