尚硅谷大数据项目之电商数仓（业务数据采集平台）

(作者：尚硅谷大数据研发部)

版本：V5.0

第 1 章 电商业务简介

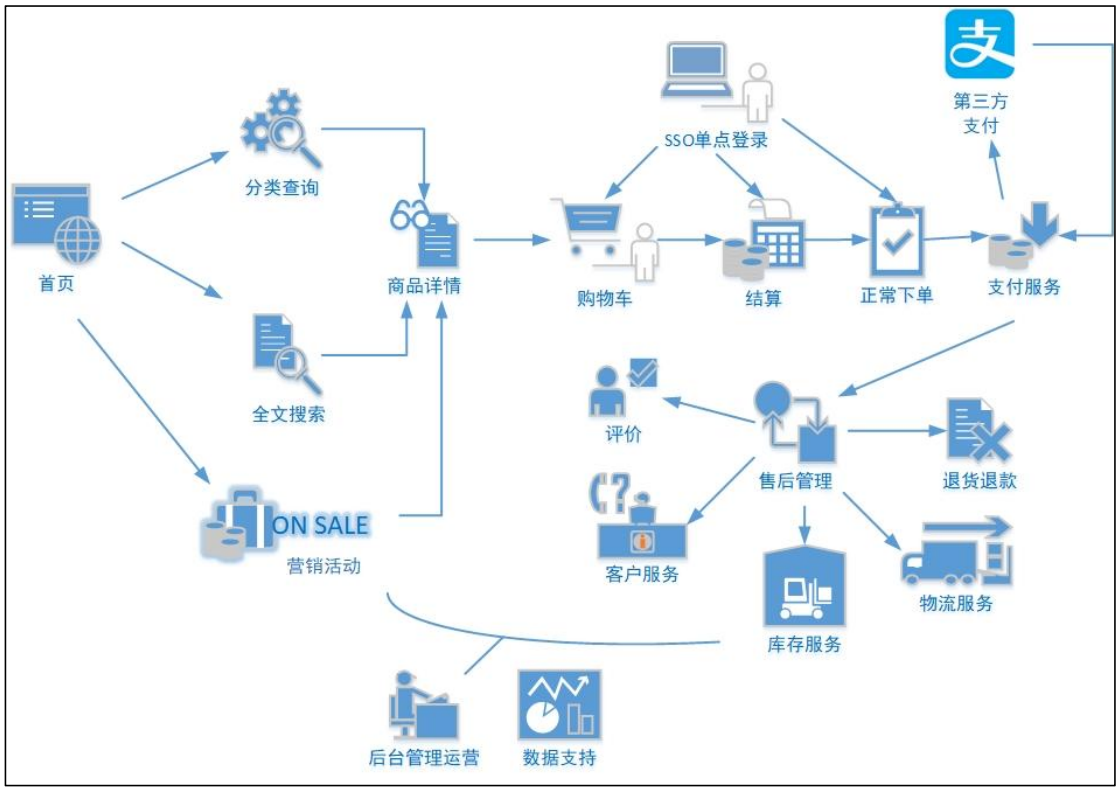
1.1 电商业务流程

电商的业务流程可以以一个普通用户的浏览足迹为例进行说明，用户点开电商首页开始浏览，可能会通过分类查询也可能通过全文搜索寻找自己中意的商品，这些商品无疑都是存储在后台的管理系统中的。

当用户寻找到自己中意的商品，可能会想要购买，将商品添加到购物车后发现需要登录，登录后对商品进行结算，这时候购物车的管理和商品订单信息的生成都会对业务数据库产生影响，会生成相应的订单数据和支付数据。

订单正式生成之后，还会对订单进行跟踪处理，直到订单全部完成。

电商的主要业务流程包括用户前台浏览商品时的商品详情的管理，用户商品加入购物车进行支付时用户个人中心&支付服务的管理，用户支付完成后订单后台服务的管理，这些流程涉及到了十几个甚至几十个业务数据表，甚至更多。



1.2 电商常识

1.2.1 SKU 和 SPU

SKU=Stock Keeping Unit（库存量基本单位）。现在已经被引申为产品统一编号的简称，每种产品均对应有唯一的 SKU 号。

SPU（Standard Product Unit）：是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息集合。

例如：iPhoneX 手机就是 SPU。一台银色、128G 内存的、支持联通网络的 iPhoneX，就是 SKU。

SPU 表示一类商品。同一 SPU 的商品可以共用商品图片、海报、销售属性等。

1.2.2 平台属性和销售属性

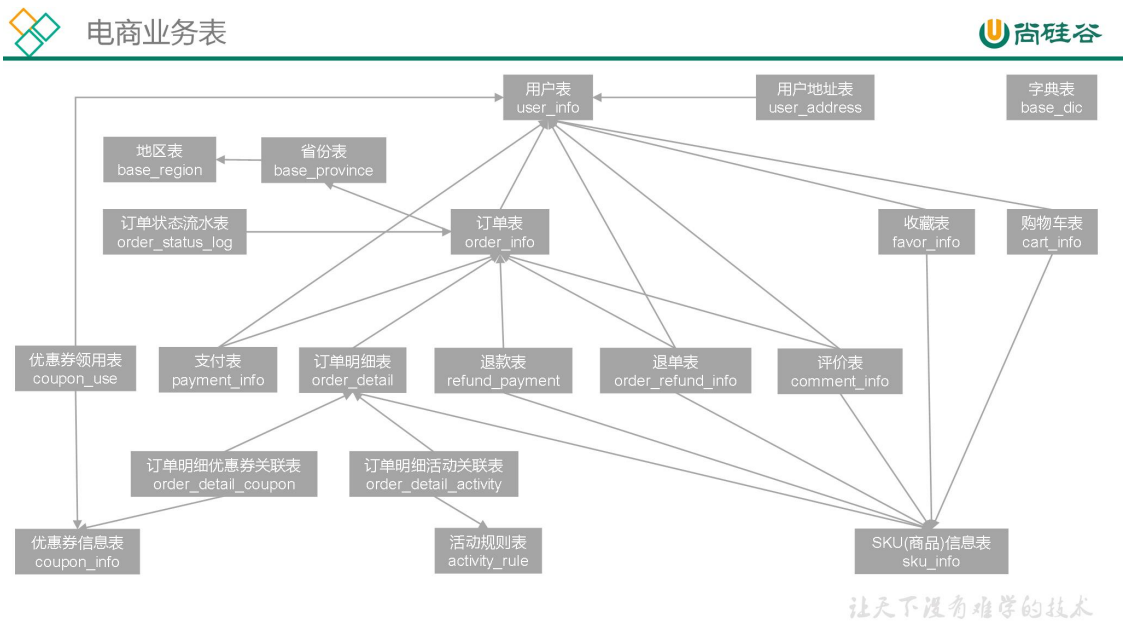
1.平台属性

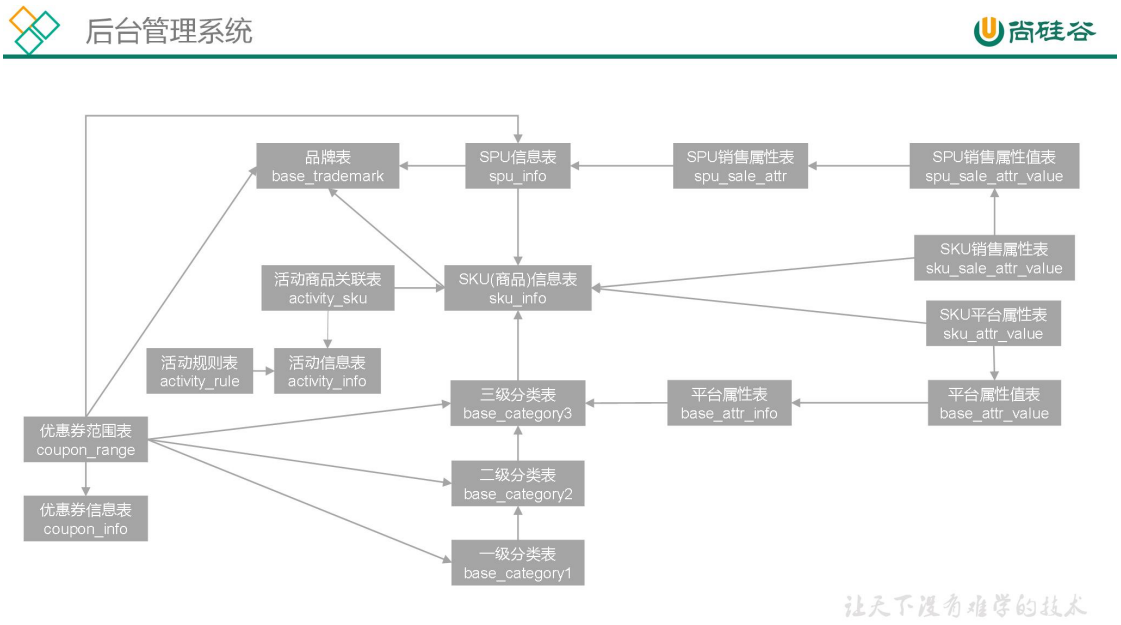
2.销售属性

第 2 章 电商业务数据

2.1 电商系统表结构

以下为本电商数仓系统涉及到的业务数据表结构关系。这 34 个表以订单表、用户表、SKU 商品表、活动表和优惠券表为中心，延伸出了优惠券领用表、支付流水表、活动订单表、订单详情表、订单状态表、商品评论表、编码字典表退单表、SPU 商品表等，用户表提供用户的详细信息，支付流水表提供该订单的支付详情，订单详情表提供订单的商品数量等情况，商品表给订单详情表提供商品的详细信息。本次讲解以此 34 个表为例，实际项目中，业务数据库中表格远远不止这些。





2.1.1 活动信息表（activity_info）

字段名	字段说明
id	活动 id
activity_name	活动名称
activity_type	活动类型（1：满减，2：折扣）
activity_desc	活动描述

start_time	开始时间
end_time	结束时间
create_time	创建时间

2.1.2 活动规则表（activity_rule）

id	编号
activity_id	活动 ID
activity_type	活动类型
condition_amount	满减金额
condition_num	满减件数
benefit_amount	优惠金额
benefit_discount	优惠折扣
benefit_level	优惠级别

2.1.3 活动商品关联表（activity_sku）

字段名	字段说明
id	编号
activity_id	活动 id
sku_id	sku_id
create_time	创建时间

2.1.4 平台属性表（base_attr_info）

字段名	字段说明
id	编号
attr_name	属性名称
category_id	分类 id
category_level	分类层级

2.1.5 平台属性值表（base_attr_value）

字段名	字段说明
id	编号
value_name	属性值名称
attr_id	属性 id

2.1.6 一级分类表（base_category1）

字段名	字段说明
id	编号

name	分类名称

2.1.7 二级分类表（base_category2）

字段名	字段说明
id	编号
name	二级分类名称
category1_id	一级分类编号

2.1.8 三级分类表（base_category3）

字段名	字段说明
id	编号
name	三级分类名称
category2_id	二级分类编号

2.1.9 字典表（base_dic）

字段名	字段说明
dic_code	编号
dic_name	编码名称
parent_code	父编号
create_time	创建日期
operate_time	修改日期

2.1.10 省份表（base_province）

字段名	字段说明
id	id
name	省名称
region_id	大区 id
area_code	行政区位码
iso_code	国际编码
iso_3166_2	ISO3166 编码

2.1.11 地区表（base_region）

字段名	字段说明
id	大区 id
region_name	大区名称

2.1.12 品牌表（base_trademark）

字段名	字段说明
id	编号
tm_name	属性值
logo_url	品牌 logo 的图片路径

2.1.13 购物车表（cart_info）

字段名	字段说明
id	编号
user_id	用户 id
sku_id	skuid
cart_price	放入购物车时价格
sku_num	数量
img_url	图片文件
sku_name	sku 名称 (冗余)
is_checked	
create_time	创建时间
operate_time	修改时间
is_ordered	是否已经下单
order_time	下单时间
source_type	来源类型
source_id	来源编号

2.1.14 评价表（comment_info）

字段名	字段说明
id	编号
user_id	用户 id
nick_name	用户昵称
head_img	
sku_id	skuid
spu_id	商品 id
order_id	订单编号
appraise	评价 1 好评 2 中评 3 差评
comment_txt	评价内容
create_time	创建时间
operate_time	修改时间

2.1.15 优惠券信息表（coupon_info）

字段名	字段说明
id	购物券编号
coupon_name	购物券名称
coupon_type	购物券类型 1 现金券 2 折扣券 3 满减券 4 满件打折券
condition_amount	满额数（3）
condition_num	满件数（4）
activity_id	活动编号
benefit_amount	减金额（1 3）
benefit_discount	折扣（2 4）
create_time	创建时间
range_type	范围类型 1、商品(spuid) 2、品类(三级分类 id) 3、品牌
limit_num	最多领用次数
taken_count	已领用次数
start_time	可以领取的开始日期
end_time	可以领取的结束日期
operate_time	修改时间
expire_time	过期时间
range_desc	范围描述

2.1.16 优惠券优惠范围表（coupon_range）

字段名	字段说明
id	购物券编号
coupon_id	优惠券 id
range_type	范围类型 1、商品(spuid) 2、品类(三级分类 id) 3、品牌
range_id	

2.1.17 优惠券领用表（coupon_use）

字段名	字段说明
id	编号
coupon_id	购物券 ID
user_id	用户 ID
order_id	订单 ID
coupon_status	购物券状态（1：未使用 2：已使用）
get_time	获取时间
using_time	使用时间
used_time	支付时间
expire_time	过期时间

2.1.18 收藏表（favor_info）

字段名	字段说明
id	编号
user_id	用户名称
sku_id	skuid
spu_id	商品 id
is_cancel	是否已取消 0 正常 1 已取消
create_time	创建时间
cancel_time	修改时间

2.1.19 订单明细表（order_detail）

字段名	字段说明
id	编号
order_id	订单编号
sku_id	sku_id
sku_name	sku 名称（冗余)
img_url	图片名称（冗余)
order_price	购买价格(下单时 sku 价格）
sku_num	购买个数
create_time	创建时间
source_type	来源类型
source_id	来源编号
split_total_amount	分摊总金额
split_activity_amount	分摊活动减免金额
split_coupon_amount	分摊优惠券减免金额

2.1.20 订单明细活动关联表（order_detail_activity）

字段名	字段说明
id	编号
order_id	订单 id
order_detail_id	订单明细 id
activity_id	活动 ID
activity_rule_id	活动规则
sku_id	skuID
create_time	获取时间

2.1.21 订单明细优惠券关联表（order_detail_coupon）

字段名	字段说明
id	编号
order_id	订单 id
order_detail_id	订单明细 id
coupon_id	购物券 ID
coupon_use_id	购物券领用 id
sku_id	skuID
create_time	获取时间

2.1.22 订单表(order_info）

字段名	字段说明
id	编号
consignee	收货人
consignee_tel	收件人电话
total_amount	总金额
order_status	订单状态
user_id	用户 id
payment_way	付款方式
delivery_address	送货地址
order_comment	订单备注
out_trade_no	订单交易编号（第三方支付用)
trade_body	订单描述(第三方支付用)
create_time	创建时间
operate_time	操作时间
expire_time	失效时间
process_status	进度状态
tracking_no	物流单编号
parent_order_id	父订单编号
img_url	图片路径
province_id	地区
activity_reduce_amount	促销金额
coupon_reduce_amount	优惠券
original_total_amount	原价金额
freight_fee	运费
freight_fee_reduce	运费减免
refundable_time	可退款日期（签收后 30 天）

2.1.23 退单表（order_refund_info）

字段名	字段说明
id	编号
user_id	用户 id
order_id	订单 id
sku_id	skuid
refund_type	退款类型
refund_num	退货件数
refund_amount	退款金额
refund_reason_type	原因类型
refund_reason_txt	原因内容
refund_status	退款状态（0：待审批 1：已退款）
create_time	创建时间

2.1.24 订单状态流水表（order_status_log）

字段名	字段说明
id	
order_id	
order_status	
operate_time	

2.1.25 支付表（payment_info）

字段名	字段说明
id	编号
out_trade_no	对外业务编号
order_id	订单编号
user_id	
payment_type	支付类型（微信 支付宝）
trade_no	交易编号
total_amount	支付金额
subject	交易内容
payment_status	支付状态
create_time	创建时间
callback_time	回调时间
callback_content	回调信息

2.1.26 退款表（refund_payment）

字段名	字段说明
id	编号
out_trade_no	对外业务编号
order_id	订单编号
sku_id	
payment_type	支付类型（微信 支付宝）
trade_no	交易编号
total_amount	退款金额
subject	交易内容
refund_status	退款状态
create_time	创建时间
callback_time	回调时间
callback_content	回调信息

2.1.27 SKU 平台属性值表（sku_attr_value）

字段名	字段说明
id	编号
attr_id	属性 id（冗余)
value_id	属性值 id
sku_id	skuid
attr_name	属性名称
value_name	属性值名称

2.1.28 SKU 信息表（sku_info）

字段名	字段说明
id	库存 id(itemID)
spu_id	商品 id
price	价格
sku_name	sku 名称
sku_desc	商品规格描述
weight	重量
tm_id	品牌(冗余)
category3_id	三级分类 id（冗余)
sku_default_img	默认显示图片(冗余)
is_sale	是否销售（1：是 0：否）
create_time	创建时间

2.1.29 SKU 销售属性表（sku_sale_attr_value）

字段名	字段说明
id	id
sku_id	库存单元 id
spu_id	spu_id(冗余)
sale_attr_value_id	销售属性值 id
sale_attr_id	
sale_attr_name	
sale_attr_value_name	

2.1.30 SPU 信息表（spu_info）

字段名	字段说明
id	商品 id
spu_name	商品名称
description	商品描述(后台简述）
category3_id	三级分类 id
tm_id	品牌 id

2.1.31 SPU 销售属性表（spu_sale_attr）

字段名	字段说明
id	编号(业务中无关联)
spu_id	商品 id
base_sale_attr_id	销售属性 id
sale_attr_name	销售属性名称(冗余)

2.1.32 SPU 销售属性值表（spu_sale_attr_value）

字段名	字段说明
id	销售属性值编号
spu_id	商品 id
base_sale_attr_id	销售属性 id
sale_attr_value_name	销售属性值名称
sale_attr_name	销售属性名称(冗余)

2.1.33 用户地址表（user_address）

字段名	字段说明
id	编号

user_id	用户 id
province_id	省份 id
user_address	用户地址
consignee	收件人
phone_num	联系方式
is_default	是否是默认

2.1.34 用户信息表（user_info）

字段名	字段说明
id	编号
login_name	用户名称
nick_name	用户昵称
passwd	用户密码
name	用户姓名
phone_num	手机号
email	邮箱
head_img	头像
user_level	用户级别
birthday	用户生日
gender	性别 M 男,F 女
create_time	创建时间
operate_time	修改时间
status	状态

2.2 模拟生成业务数据

2.2.1 MySQL 安装

1）安装包准备

（1）卸载自带的 Mysql-libs（如果之前安装过 mysql，要全都卸载掉）

```
[atguigu@hadoop102 software]$ rpm -qa | grep -i -E mysql\|mariadb
| xargs -n1 sudo rpm -e --nodeps
```

（2）将安装包和 JDBC 驱动上传到/opt/software，共计 6 个

```
01_mysql-community-common-5.7.16-1.el7.x86_64.rpm
02_mysql-community-libs-5.7.16-1.el7.x86_64.rpm
03_mysql-community-libs-compat-5.7.16-1.el7.x86_64.rpm
04_mysql-community-client-5.7.16-1.el7.x86_64.rpm
05_mysql-community-server-5.7.16-1.el7.x86_64.rpm
mysql-connector-java-5.1.27-bin.jar
```

2）安装 MySQL

（1）安装 mysql 依赖

```
[atguigu@hadoop102 software]$ sudo rpm -ivh
01_mysql-community-common-5.7.16-1.el7.x86_64.rpm
```

```
[atguigu@hadoop102 software]$ sudo rpm -ivh 02_mysql-community-libs-5.7.16-1.el7.x86_64.rpm
[atguigu@hadoop102 software]$ sudo rpm -ivh 03_mysql-community-libs-compat-5.7.16-1.el7.x86_64.rpm
```

（2）安装 mysql-client

```
[atguigu@hadoop102 software]$ sudo rpm -ivh 04_mysql-community-client-5.7.16-1.el7.x86_64.rpm
```

（3）安装 mysql-server

```
[atguigu@hadoop102 software]$ sudo rpm -ivh 05_mysql-community-server-5.7.16-1.el7.x86_64.rpm
```

（4）启动 mysql

```
[atguigu@hadoop102 software]$ sudo systemctl start mysqld
```

（5）查看 mysql 密码

```
[atguigu@hadoop102 software]$ sudo cat /var/log/mysqld.log | grep password
```

3）配置 MySQL

配置只要是 root 用户+密码，在任何主机上都能登录 MySQL 数据库。

（1）用刚刚查到的密码进入 mysql（如果报错，给密码加单引号）

```
[atguigu@hadoop102 software]$ mysql -uroot -p'password'
```

（2）设置复杂密码(由于 mysql 密码策略，此密码必须足够复杂)

```
mysql> set password=password("Qs23=zs32");
```

（3）更改 mysql 密码策略

```
mysql> set global validate_password_length=4;
mysql> set global validate_password_policy=0;
```

（4）设置简单好记的密码

```
mysql> set password=password("123456");
```

（5）进入 msyql 库

```
mysql> use mysql
```

（6）查询 user 表

```
mysql> select user, host from user;
```

（7）修改 user 表，把 Host 表内容修改为%

```
mysql> update user set host="%" where user="root";
```

（8）刷新

```
mysql> flush privileges;
```

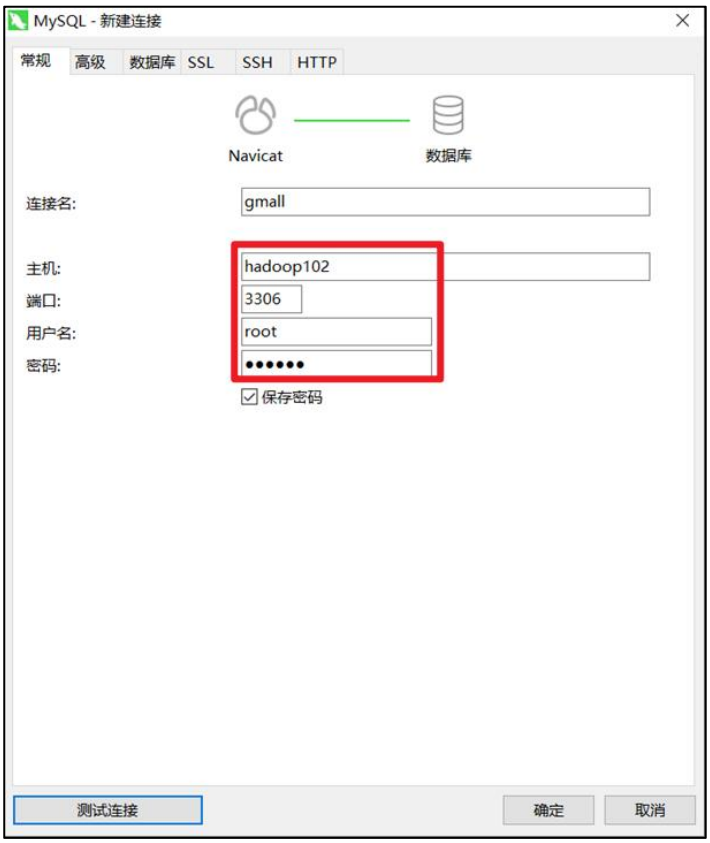
（9）退出

```
mysql> quit;
```

2.2.2 业务数据生成

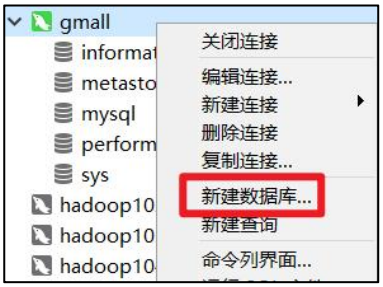
1）连接 MySQL

通过 MySQL 可视化客户端连接数据库。

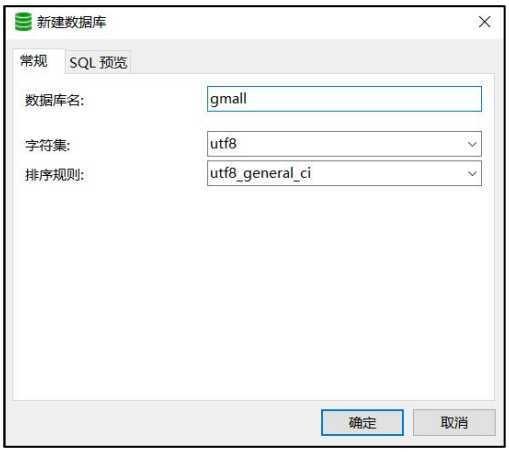


2）建表语句

（1）通过 SQLyog 创建数据库



（2）设置数据库名称为 **gmall**，编码为 **utf-8**，**排序规则**为 utf8_general_ci



（3）导入数据库结构脚本（**gmall.sql**）

注意：完成后，要记得右键，刷新一下对象浏览器，就可以看见数据库中的表了。

3）生成业务数据

（1）在 hadoop102 的/opt/module/目录下创建 db_log 文件夹

```
[atguigu@hadoop102 module]$ mkdir db_log/
```

（2）把 gmall2020-mock-db-2021-11-14.jar 和 application.properties 上传到 hadoop102 的 /opt/module/db_log 路径上。

（3）根据需求修改 application.properties 相关配置

```
logging.level.root=info

spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://hadoop102:3306/gmall?useUni
code=true&characterEncoding=utf-8&useSSL=false&serverTimezone=
GMT%2B8
spring.datasource.username=root
spring.datasource.password=123456

logging.pattern.console=%m%n


mybatis-plus.global-config.db-config.field-strategy=not_null
mybatis.mapperLocations=classpath:mapper/*.xml

#业务日期
mock.date=2020-06-14
#是否重置，首日须设置为 1
mock.clear=1
#是否重置用户，首日须设置为 1
mock.clear.user=1

#生成新用户数量
mock.user.count=200
#男性比例
mock.user.male-rate=20
```

```
#用户数据变化概率
mock.user.update-rate:20

#收藏取消比例
mock.favor.cancel-rate=10
#收藏数量
mock.favor.count=100

#每个用户添加购物车的概率
mock.cart.user-rate=10
#每次每个用户最多添加多少种商品进购物车
mock.cart.max-sku-count=8
#每个商品最多买几个
mock.cart.max-sku-num=3

#购物车来源  用户查询，商品推广，智能推荐，促销活动
mock.cart.source-type-rate=60:20:10:10

#用户下单比例
mock.order.user-rate=30
#用户从购物中购买商品比例
mock.order.sku-rate=50
#是否参加活动
mock.order.join-activity=1
#是否使用购物券
mock.order.use-coupon=1
#购物券领取人数
mock.coupon.user-count=100

#支付比例
mock.payment.rate=70
#支付方式 支付宝：微信 ：银联
mock.payment.payment-type=30:60:10

#评价比例 好：中：差：自动
mock.comment.appraise-rate=30:10:10:50

#退款原因比例：质量问题 商品描述与实际描述不一致 缺货 号码不合适 拍错 不想买了 其他
mock.refund.reason-rate=30:10:20:5:15:5:5

logging.level.com.atguigu.gmall2020.mock.db.mapper=debug
```

（4）并在该目录下执行，如下命令，生成 2020-06-14 日期数据：

```
[atguigu@hadoop102          db_log]$          java          -jar
gmall2020-mock-db-2021-11-14.jar
```

（5）查看 gmall 数据库，观察是否有 2020-06-14 的数据出现

2.2.3 业务数据梳理工具

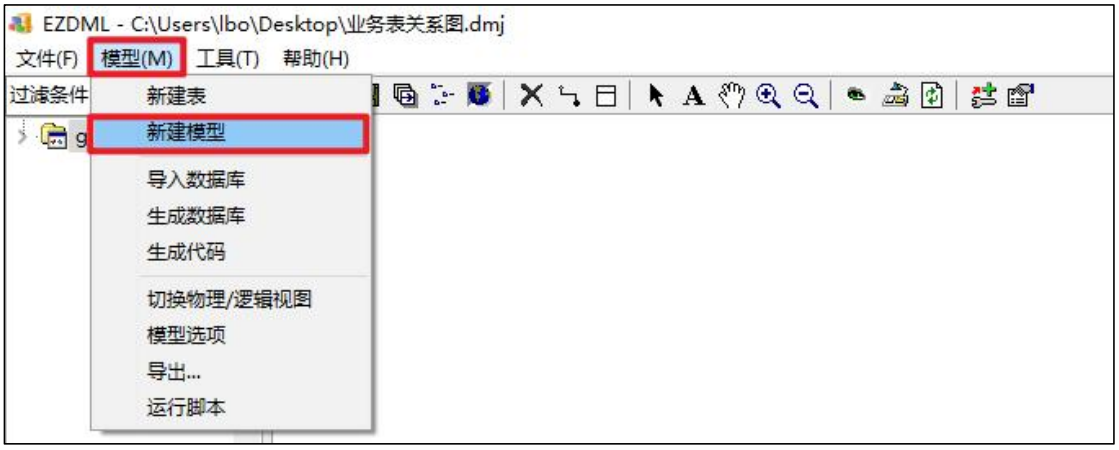
可借助 EZDML 这款数据库设计工具，来辅助我们梳理复杂的业务表关系。

1）下载地址

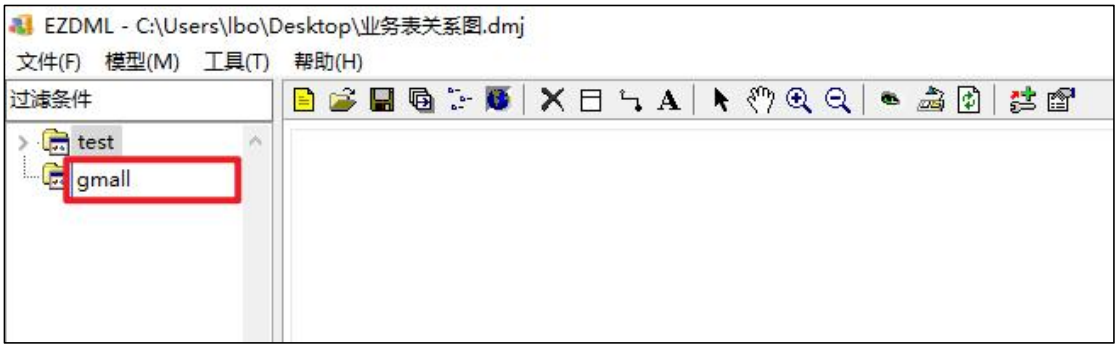
http://www.ezdml.com/download_cn.html

2）使用说明

（1）新建模型

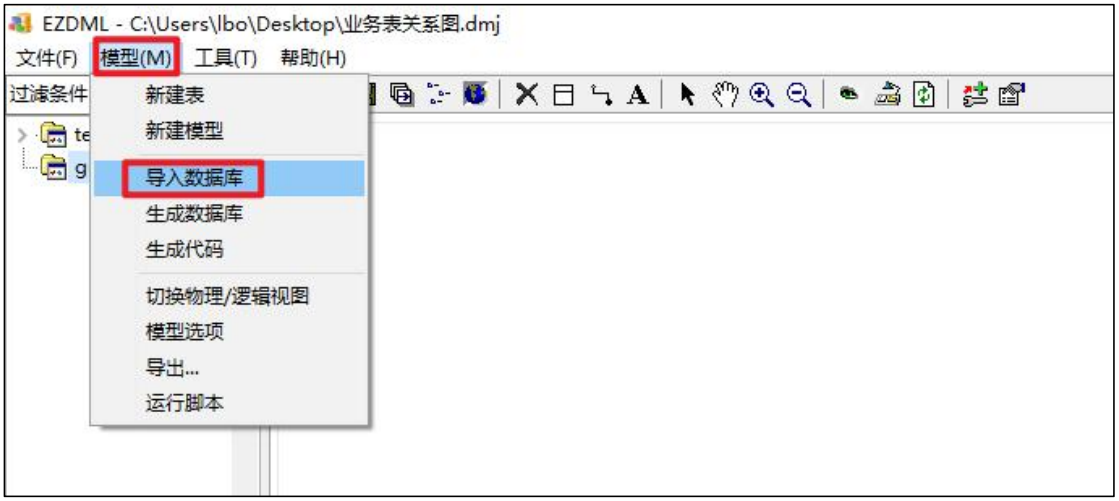


（2）命名模型

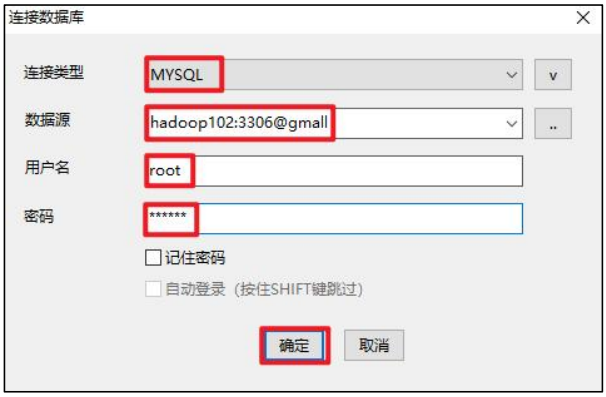


（3）点击图标，选中模型

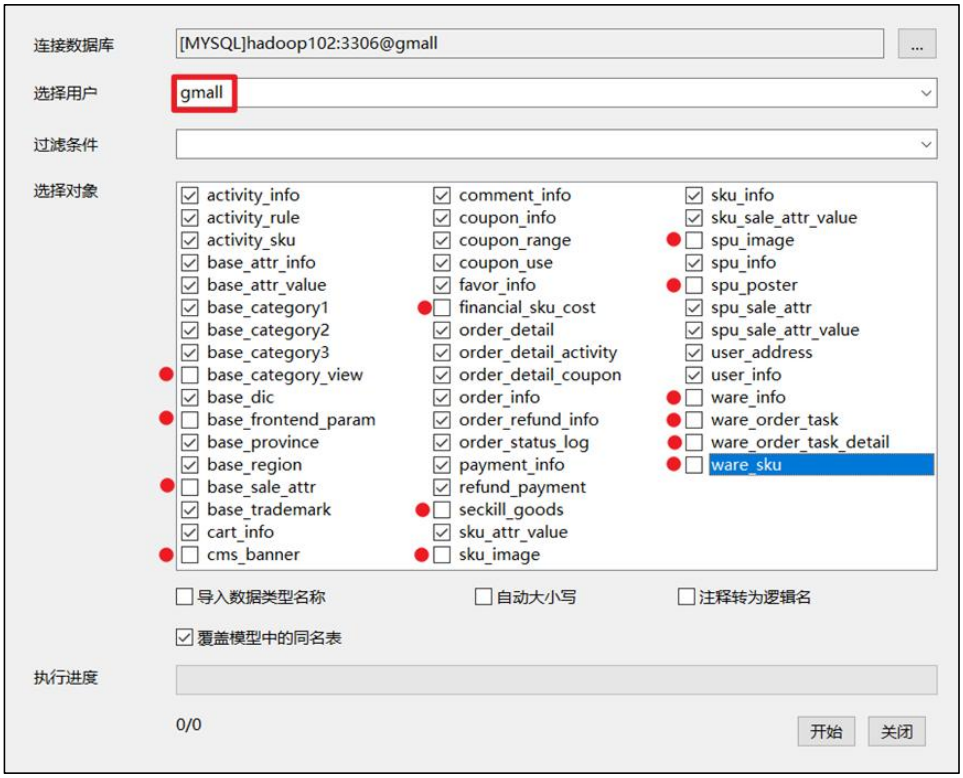
（4）导入数据库



（5）配置数据库连接

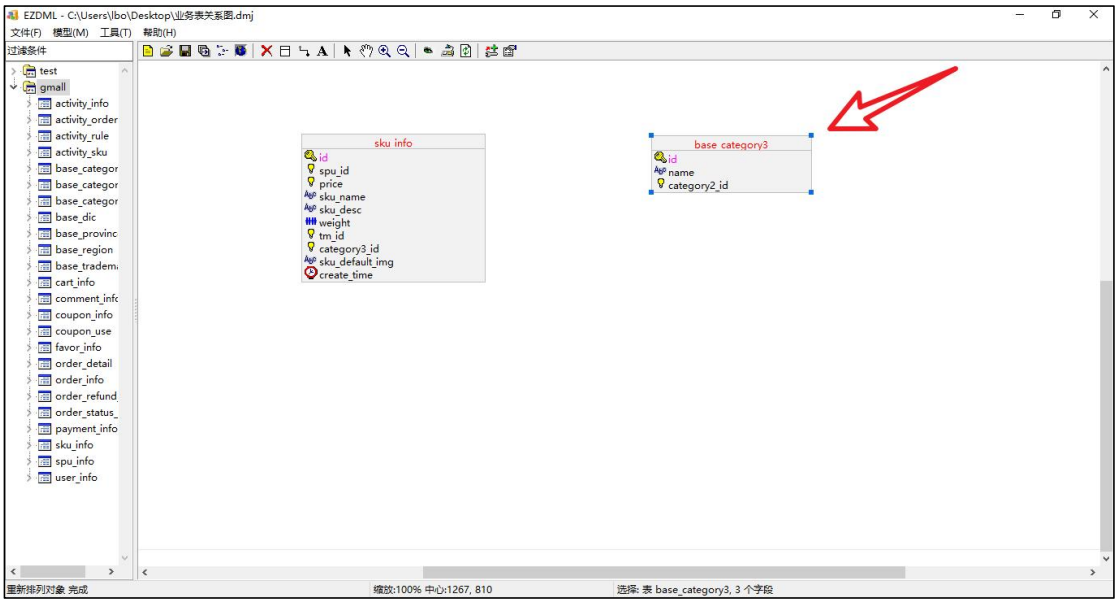


（6）选择导入的表（标注红点的表不需要导入）

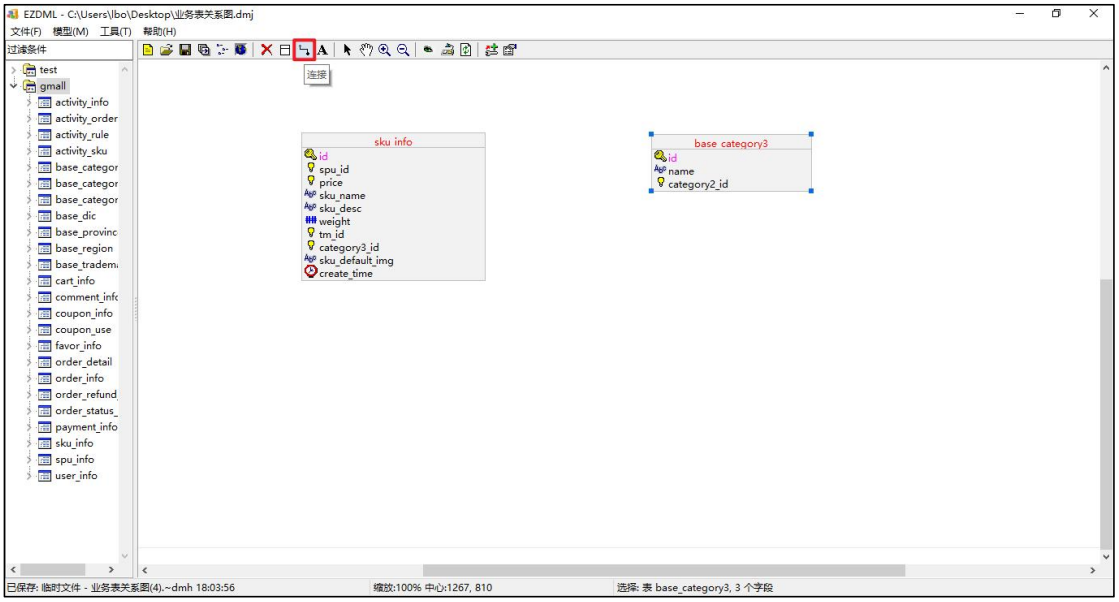


（7）建立表关系

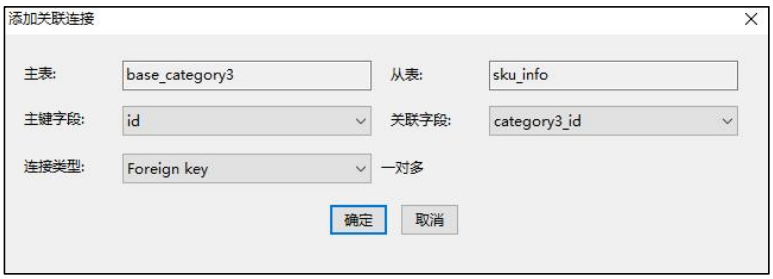
第一步：点击选中主表（主键所在的表）



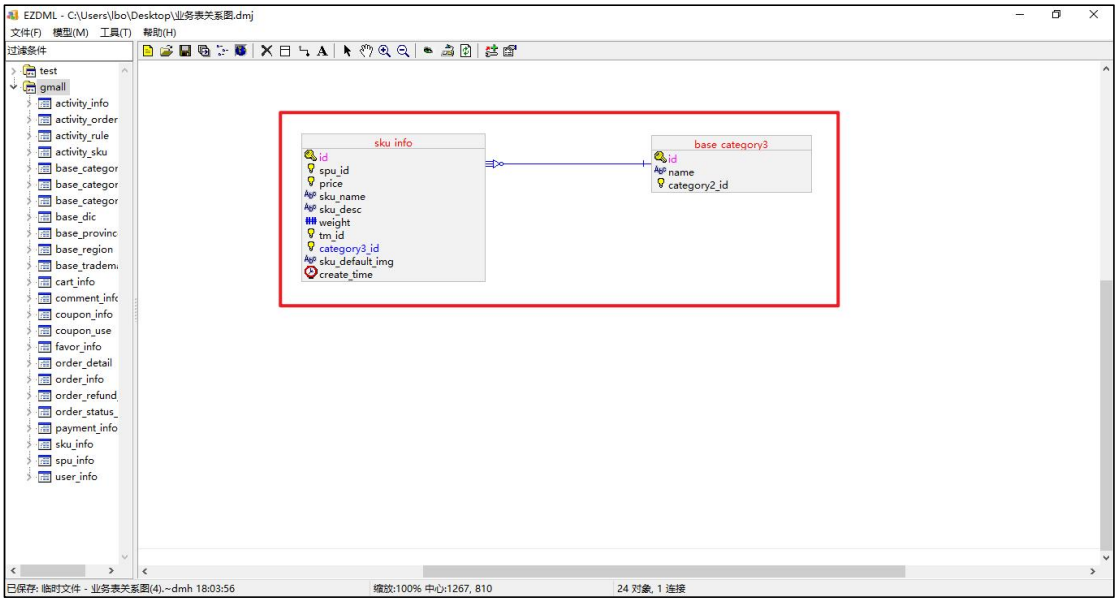
第二步：点击连接按钮



第三步：点击从表，配置连接条件

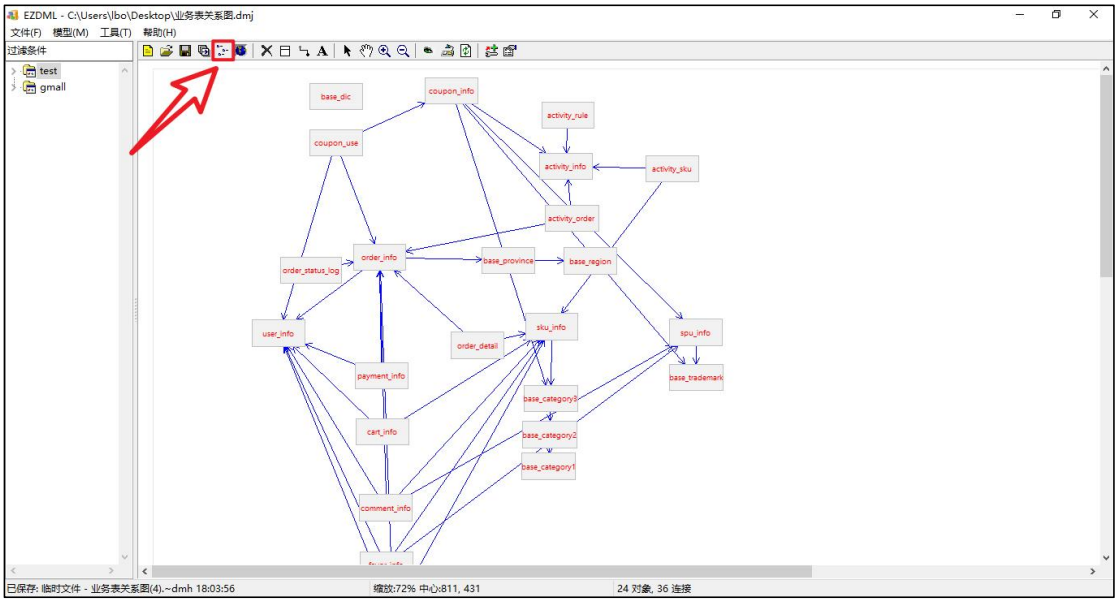


第四步：效果展示



3）使用技巧

（1）缩略图



（2）热键

按住 shift 键，用鼠标点击表，进行多选，可实现批量移动

按住 ctrl 键，用鼠标圈选表，也可进行多选，实现批量移动

第 3 章 业务数据采集模块

3.1 业务数据同步概述

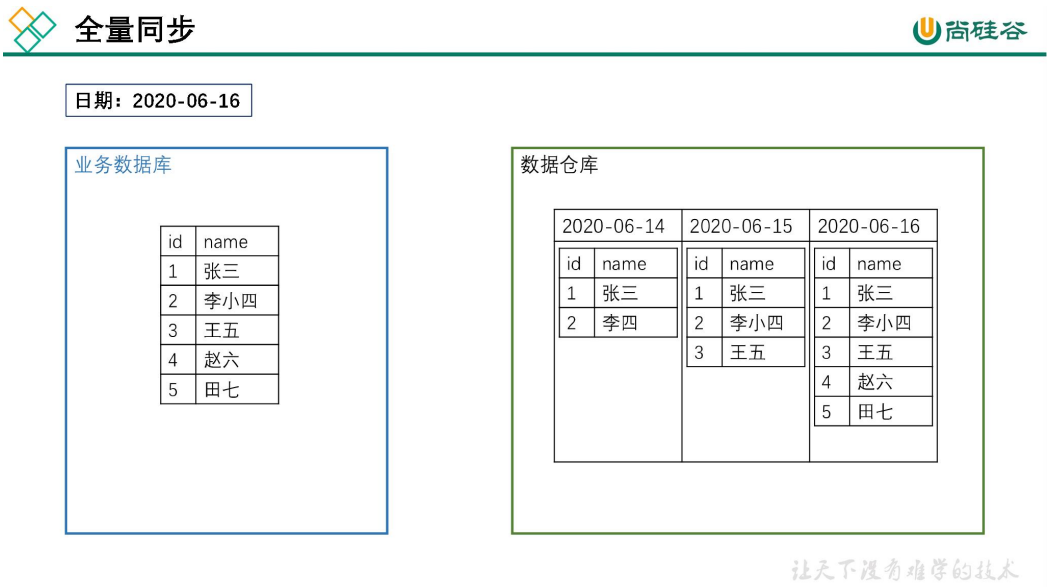
3.1.1 数据同步策略概述

业务数据是数据仓库的重要数据来源，我们需要每日定时从业务数据库中抽取数据，传输到数据仓库中，之后再对数据进行分析统计。

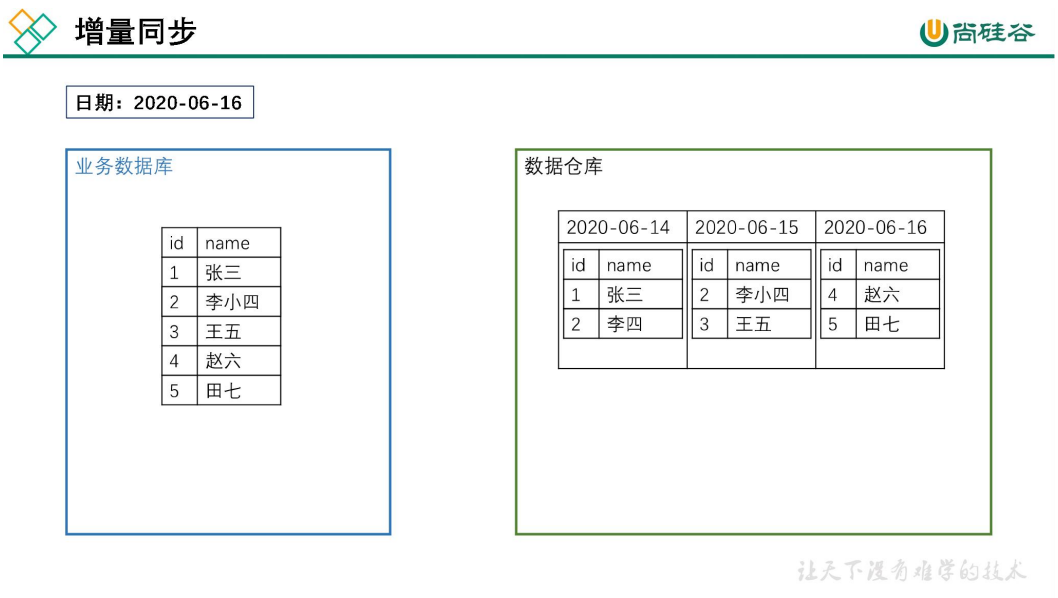
为保证统计结果的正确性，需要保证数据仓库中的数据与业务数据库是同步的，离线数仓的计算周期通常为天，所以数据同步周期也通常为天，即每天同步一次即可。

数据的同步策略有**全量同步**和**增量同步**。

全量同步，就是每天都将业务数据库中的全部数据同步一份到数据仓库，这是保证两侧数据同步的最简单的方式。



增量同步，就是每天只将业务数据中的新增及变化数据同步到数据仓库。采用每日增量同步的表，通常需要在首日先进行一次全量同步。



3.1.2 数据同步策略选择

两种策略都能保证数据仓库和业务数据库的数据同步，那应该如何选择呢？下面对两种策略进行简要对比。

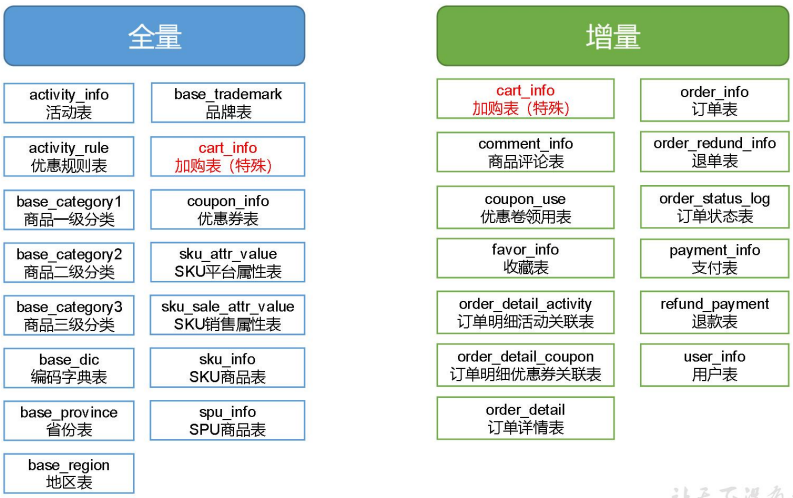
同步策略	优点	缺点
全量同步	逻辑简单	在某些情况下效率较低。例如某张表数据量较大，但是每天数据的变化比例很低，若对其采用每日全量同步，则会重复同步和存储大量相同的数据。
增量同步	效率高，无需同步和存储重复数据	逻辑复杂，需要将每日的新增及变化数据同原来的数据进行整合，才能使用

根据上述对比，可以得出以下结论：

若业务表数据量比较大，且每天数据变化的比例比较低，这时应采用增量同步，否则可采用全量同步。

下图为各表同步策略：

各表同步策略



注：由于后续数仓建模需要，cart_info 需进行全量同步和增量同步，此处暂不解释，后续章节会作出解释。

3.1.3 数据同步工具概述

数据同步工具种类繁多，大致可分为两类，一类是以 DataX、Sqoop 为代表的基于 Select 查询的离线、批量同步工具，另一类是以 Maxwell、Canal 为代表的基于数据库数据变更日志（例如 MySQL 的 binlog，其会实时记录所有的 insert、update 以及 delete 操作）的实时流式同步工具。

全量同步通常使用 DataX、Sqoop 等基于查询的离线同步工具。而增量同步既可以使用 DataX、Sqoop 等工具，也可使用 Maxwell、Canal 等工具，下面对增量同步不同方案进行简要对比。

增量同步方案	DataX/Sqoop	Maxwell/Canal
对数据库的要求	原理是基于查询，故若想通过 select 查询获取新增及变化数据，就要求数据表中存在 create_time、update_time 等字段，然后根据这些字段获取变更数据。	要求数据库记录变更操作，例如 MySQL 需开启 binlog。
数据的中间状态	由于是离线批量同步，故若一条数据在一天中变化多次，该方案只能获取最后一个状态，中间状态无法获取。	由于是实时获取所有的数据变更操作，所以可以获取变更数据的所有中间状态。

本项目中，全量同步采用 DataX，增量同步采用 Maxwell。

3.1.4 数据同步工具部署

1）DataX

尚硅谷大数据技术之DataX.docx

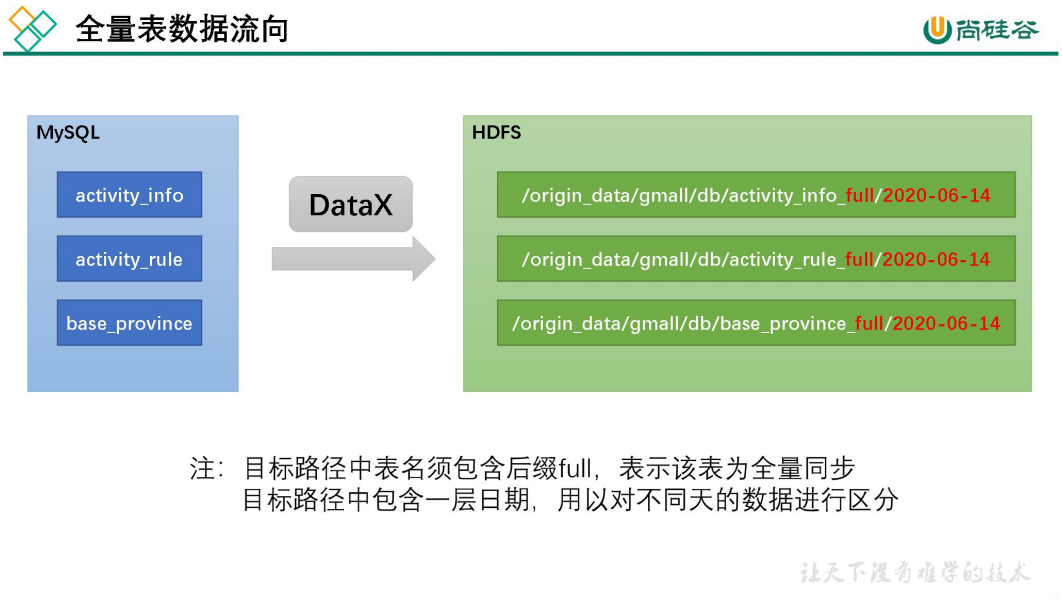
2）Maxwell

尚硅谷大数据技术之Maxwell.docx

3.2 全量表数据同步

3.2.1 数据通道

全量表数据由 DataX 从 MySQL 业务数据库直接同步到 HDFS，具体数据流向如下图所示。



3.2.2 DataX 配置文件

我们需要为每张全量表编写一个 DataX 的 json 配置文件，此处以 activity_info 为例，配置文件内容如下：

```
{
    "job": {
        "content": [
```

```
{
    "reader": {
        "name": "mysqlreader",
        "parameter": {
            "column": [
                "id",
                "activity_name",
                "activity_type",
                "activity_desc",
                "start_time",
                "end_time",
                "create_time"
            ],
            "connection": [
                {
                    "jdbcUrl": [
"jdbc:mysql://hadoop102:3306/gmall"
                    ],
                    "table": [
                        "activity_info"
                    ]
                }
            ],
            "password": "123456",
            "splitPk": "",
            "username": "root"
        }
    },
    "writer": {
        "name": "hdfswriter",
        "parameter": {
            "column": [
                {
                    "name": "id",
                    "type": "bigint"
                },
                {
                    "name": "activity_name",
                    "type": "string"
                },
                {
                    "name": "activity_type",
                    "type": "string"
                },
                {
                    "name": "activity_desc",
                    "type": "string"
                },
                {
                    "name": "start_time",
                    "type": "string"
                },
                {
                    "name": "end_time",
                    "type": "string"
                },
                {
```

```
                            "name": "create_time",
                            "type": "string"
                        }
                    ],
                    "compress": "gzip",
                    "defaultFS": "hdfs://hadoop102:8020",
                    "fieldDelimiter": "\t",
                    "fileName": "activity_info",
                    "fileType": "text",
                    "path": "${targetdir}",
                    "writeMode": "append"
                }
            }
        }
    ],
    "setting": {
        "speed": {
            "channel": 1
        }
    }
  }
}
```

注：由于目标路径包含一层日期，用于对不同天的数据加以区分，故 path 参数并未写死，需在提交任务时通过参数动态传入，参数名称为 **targetdir**。

3.2.3 DataX 配置文件生成脚本

方便起见，此处提供了 DataX 配置文件批量生成脚本，脚本内容及使用方式如下。

1）在~/bin 目录下创建 gen_import_config.py 脚本

```
[atguigu@hadoop102 bin]$ vim ~/bin/gen_import_config.py
```

脚本内容如下

```
# coding=utf-8
import json
import getopt
import os
import sys
import MySQLdb

#MySQL 相关配置，需根据实际情况作出修改
mysql_host = "hadoop102"
mysql_port = "3306"
mysql_user = "root"
mysql_passwd = "123456"

#HDFS NameNode 相关配置，需根据实际情况作出修改
hdfs_nn_host = "hadoop102"
hdfs_nn_port = "8020"

#生成配置文件的目标路径，可根据实际情况作出修改
output_path = "/opt/module/datax/job/import"

#获取 mysql 连接
def get_connection():
```

```
    return MySQLdb.connect(host=mysql_host, port=int(mysql_port),
user=mysql_user, passwd=mysql_passwd)

#获取表格的元数据  包含列名和数据类型
def get_mysql_meta(database, table):
    connection = get_connection()
    cursor = connection.cursor()
    sql      =      "SELECT      COLUMN_NAME,DATA_TYPE      from
information_schema.COLUMNS     WHERE     TABLE_SCHEMA=%s     AND
TABLE_NAME=%s ORDER BY ORDINAL_POSITION"
    cursor.execute(sql, [database, table])
    fetchall = cursor.fetchall()
    cursor.close()
    connection.close()
    return fetchall

#获取 mysql 表的列名
def get_mysql_columns(database, table):
    return map(lambda x: x[0], get_mysql_meta(database, table))

#将获取的元数据中 mysql 的数据类型转换为 hive 的数据类型  写入到
hdfswriter 中
def get_hive_columns(database, table):
    def type_mapping(mysql_type):
        mappings = {
            "bigint": "bigint",
            "int": "bigint",
            "smallint": "bigint",
            "tinyint": "bigint",
            "decimal": "string",
            "double": "double",
            "float": "float",
            "binary": "string",
            "char": "string",
            "varchar": "string",
            "datetime": "string",
            "time": "string",
            "timestamp": "string",
            "date": "string",
            "text": "string"
        }
        return mappings[mysql_type]

    meta = get_mysql_meta(database, table)
    return     map(lambda     x:     {"name":     x[0],     "type":
type_mapping(x[1].lower())}, meta)

#生成 json 文件
def generate_json(source_database, source_table):
    job = {
        "job": {
            "setting": {
                "speed": {
                    "channel": 3
                },
                "errorLimit": {
                    "record": 0,
```

```
                "percentage": 0.02
            }
        },
        "content": [{
            "reader": {
                "name": "mysqlreader",
                "parameter": {
                    "username": mysql_user,
                    "password": mysql_passwd,
                    "column": get_mysql_columns(source_database,
source_table),
                    "splitPk": "",
                    "connection": [{
                        "table": [source_table],
                        "jdbcUrl": ["jdbc:mysql://" + mysql_host
+ ":" + mysql_port + "/" + source_database]
                    }]
                }
            },
            "writer": {
                "name": "hdfswriter",
                "parameter": {
                    "defaultFS": "hdfs://" + hdfs_nn_host + ":"
+ hdfs_nn_port,
                    "fileType": "text",
                    "path": "${targetdir}",
                    "fileName": source_table,
                    "column": get_hive_columns(source_database,
source_table),
                    "writeMode": "append",
                    "fieldDelimiter": "\t",
                    "compress": "gzip"
                }
            }
        }]
    }
}
if not os.path.exists(output_path):
    os.makedirs(output_path)
with                          open(os.path.join(output_path,
".".join([source_database, source_table, "json"])), "w") as f:
    json.dump(job, f)


def main(args):
    source_database = ""
    source_table = ""

    options,    arguments    =    getopt.getopt(args,    '-d:-t:',
['sourcedb=', 'sourcetbl='])
    for opt_name, opt_value in options:
        if opt_name in ('-d', '--sourcedb'):
            source_database = opt_value
        if opt_name in ('-t', '--sourcetbl'):
            source_table = opt_value

    generate_json(source_database, source_table)
```

```
if __name__ == '__main__':
    main(sys.argv[1:])
```

注：

（1）安装 Python Mysql 驱动

由于需要使用 Python 访问 Mysql 数据库，故需安装驱动，命令如下：

```
[atguigu@hadoop102 bin]$ sudo yum install -y MySQL-python
```

（2）脚本使用说明

```
python gen_import_config.py -d database -t table
```

通过-d 传入数据库名，-t 传入表名，执行上述命令即可生成该表的 DataX 同步配置文件。

2）在~/bin 目录下创建 gen_import_config.sh 脚本

```
[atguigu@hadoop102 bin]$ vim ~/bin/gen_import_config.sh
```

脚本内容如下

```
#!/bin/bash

python ~/bin/gen_import_config.py -d gmall -t activity_info
python ~/bin/gen_import_config.py -d gmall -t activity_rule
python ~/bin/gen_import_config.py -d gmall -t base_category1
python ~/bin/gen_import_config.py -d gmall -t base_category2
python ~/bin/gen_import_config.py -d gmall -t base_category3
python ~/bin/gen_import_config.py -d gmall -t base_dic
python ~/bin/gen_import_config.py -d gmall -t base_province
python ~/bin/gen_import_config.py -d gmall -t base_region
python ~/bin/gen_import_config.py -d gmall -t base_trademark
python ~/bin/gen_import_config.py -d gmall -t cart_info
python ~/bin/gen_import_config.py -d gmall -t coupon_info
python ~/bin/gen_import_config.py -d gmall -t sku_attr_value
python ~/bin/gen_import_config.py -d gmall -t sku_info
python ~/bin/gen_import_config.py -d gmall -t sku_sale_attr_value
python ~/bin/gen_import_config.py -d gmall -t spu_info
```

3）为 gen_import_config.sh 脚本增加执行权限

```
[atguigu@hadoop102 bin]$ chmod +x ~/bin/gen_import_config.sh
```

4）执行 gen_import_config.sh 脚本，生成配置文件

```
[atguigu@hadoop102 bin]$ gen_import_config.sh
```

5）观察生成的配置文件

```
[atguigu@hadoop102 bin]$ ll /opt/module/datax/job/import/
总用量 60
-rw-rw-r-- 1 atguigu atguigu  957 10月 15 22:17 gmall.activity_info.json
-rw-rw-r-- 1 atguigu atguigu 1049 10月 15 22:17 gmall.activity_rule.json
-rw-rw-r-- 1 atguigu atguigu  651 10月 15 22:17 gmall.base_category1.json
-rw-rw-r-- 1 atguigu atguigu  711 10月 15 22:17 gmall.base_category2.json
-rw-rw-r-- 1 atguigu atguigu  711 10月 15 22:17 gmall.base_category3.json
-rw-rw-r-- 1 atguigu atguigu  835 10月 15 22:17 gmall.base_dic.json
-rw-rw-r-- 1 atguigu atguigu  865 10月 15 22:17 gmall.base_province.json
-rw-rw-r-- 1 atguigu atguigu  659 10月 15 22:17 gmall.base_region.json
-rw-rw-r-- 1 atguigu atguigu  709 10月 15 22:17 gmall.base_trademark.json
-rw-rw-r-- 1 atguigu atguigu 1301 10月 15 22:17 gmall.cart_info.json
-rw-rw-r-- 1 atguigu atguigu 1545 10月 15 22:17 gmall.coupon_info.json
-rw-rw-r-- 1 atguigu atguigu  867 10月 15 22:17 gmall.sku_attr_value.json
```

```
-rw-rw-r-- 1 atguigu atguigu 1121 10月 15 22:17 gmall.sku_info.json
-rw-rw-r-- 1 atguigu atguigu  985 10月 15 22:17 gmall.sku_sale_attr_value.json
-rw-rw-r-- 1 atguigu atguigu  811 10月 15 22:17 gmall.spu_info.json
```

3.2.4 测试生成的 DataX 配置文件

以 activity_info 为例，测试用脚本生成的配置文件是否可用。

1）创建目标路径

由于 DataX 同步任务要求目标路径提前存在，故需手动创建路径，当前 activity_info 表的目标路径应为/origin_data/gmall/db/activity_info_full/2020-06-14。

```
[atguigu@hadoop102 bin]$ hadoop fs -mkdir /origin_data/gmall/db/activity_info_full/2020-06-14
```

2）执行 DataX 同步命令

```
[atguigu@hadoop102 bin]$ python /opt/module/datax/bin/datax.py -p"-Dtargetdir=/origin_data/gmall/db/activity_info_full/2020-06-14" /opt/module/datax/job/import/gmall.activity_info.json
```

3）观察同步结果

观察 HFDS 目标路径是否出现数据。

3.2.5 全量表数据同步脚本

为方便使用以及后续的任务调度，此处编写一个全量表数据同步脚本。

1）在~/bin 目录创建 mysql_to_hdfs_full.sh

```
[atguigu@hadoop102 bin]$ vim ~/bin/mysql_to_hdfs_full.sh
```

脚本内容如下

```
#!/bin/bash

DATAX_HOME=/opt/module/datax

# 如果传入日期则 do_date 等于传入的日期，否则等于前一天日期
if [ -n "$2" ] ;then
    do_date=$2
else
    do_date=`date -d "-1 day" +%F`
fi

#处理目标路径，此处的处理逻辑是，如果目标路径不存在，则创建；若存在，则清空，目的是保证同步任务可重复执行
handle_targetdir() {
  hadoop fs -test -e $1
  if [[ $? -eq 1 ]]; then
    echo "路径$1 不存在，正在创建......"
    hadoop fs -mkdir -p $1
  else
    echo "路径$1 已经存在"
    fs_count=$(hadoop fs -count $1)
    content_size=$(echo $fs_count | awk '{print $3}')
    if [[ $content_size -eq 0 ]]; then
```

```
    echo "路径$1 为空"
  else
    echo "路径$1 不为空，正在清空......"
    hadoop fs -rm -r -f $1/*
  fi
 fi
}

#数据同步
import_data() {
 datax_config=$1
 target_dir=$2

 handle_targetdir $target_dir
 python  $DATAX_HOME/bin/datax.py  -p"-Dtargetdir=$target_dir"
$datax_config
}

case $1 in
"activity_info")
 import_data
/opt/module/datax/job/import/gmall.activity_info.json
/origin_data/gmall/db/activity_info_full/$do_date
 ;;
"activity_rule")
 import_data
/opt/module/datax/job/import/gmall.activity_rule.json
/origin_data/gmall/db/activity_rule_full/$do_date
 ;;
"base_category1")
 import_data
/opt/module/datax/job/import/gmall.base_category1.json
/origin_data/gmall/db/base_category1_full/$do_date
 ;;
"base_category2")
 import_data
/opt/module/datax/job/import/gmall.base_category2.json
/origin_data/gmall/db/base_category2_full/$do_date
 ;;
"base_category3")
 import_data
/opt/module/datax/job/import/gmall.base_category3.json
/origin_data/gmall/db/base_category3_full/$do_date
 ;;
"base_dic")
 import_data  /opt/module/datax/job/import/gmall.base_dic.json
/origin_data/gmall/db/base_dic_full/$do_date
 ;;
"base_province")
 import_data
/opt/module/datax/job/import/gmall.base_province.json
/origin_data/gmall/db/base_province_full/$do_date
 ;;
"base_region")
 import_data
/opt/module/datax/job/import/gmall.base_region.json
/origin_data/gmall/db/base_region_full/$do_date
```

```
  ;;
"base_trademark")
  import_data
/opt/module/datax/job/import/gmall.base_trademark.json
/origin_data/gmall/db/base_trademark_full/$do_date
  ;;
"cart_info")
  import_data /opt/module/datax/job/import/gmall.cart_info.json
/origin_data/gmall/db/cart_info_full/$do_date
  ;;
"coupon_info")
  import_data
/opt/module/datax/job/import/gmall.coupon_info.json
/origin_data/gmall/db/coupon_info_full/$do_date
  ;;
"sku_attr_value")
  import_data
/opt/module/datax/job/import/gmall.sku_attr_value.json
/origin_data/gmall/db/sku_attr_value_full/$do_date
  ;;
"sku_info")
  import_data  /opt/module/datax/job/import/gmall.sku_info.json
/origin_data/gmall/db/sku_info_full/$do_date
  ;;
"sku_sale_attr_value")
  import_data
/opt/module/datax/job/import/gmall.sku_sale_attr_value.json
/origin_data/gmall/db/sku_sale_attr_value_full/$do_date
  ;;
"spu_info")
  import_data  /opt/module/datax/job/import/gmall.spu_info.json
/origin_data/gmall/db/spu_info_full/$do_date
  ;;
"all")
  import_data
/opt/module/datax/job/import/gmall.activity_info.json
/origin_data/gmall/db/activity_info_full/$do_date
  import_data
/opt/module/datax/job/import/gmall.activity_rule.json
/origin_data/gmall/db/activity_rule_full/$do_date
  import_data
/opt/module/datax/job/import/gmall.base_category1.json
/origin_data/gmall/db/base_category1_full/$do_date
  import_data
/opt/module/datax/job/import/gmall.base_category2.json
/origin_data/gmall/db/base_category2_full/$do_date
  import_data
/opt/module/datax/job/import/gmall.base_category3.json
/origin_data/gmall/db/base_category3_full/$do_date
  import_data  /opt/module/datax/job/import/gmall.base_dic.json
/origin_data/gmall/db/base_dic_full/$do_date
  import_data
/opt/module/datax/job/import/gmall.base_province.json
/origin_data/gmall/db/base_province_full/$do_date
  import_data
/opt/module/datax/job/import/gmall.base_region.json
/origin_data/gmall/db/base_region_full/$do_date
  import_data
```

```
/opt/module/datax/job/import/gmall.base_trademark.json
/origin_data/gmall/db/base_trademark_full/$do_date
  import_data /opt/module/datax/job/import/gmall.cart_info.json
/origin_data/gmall/db/cart_info_full/$do_date
  import_data
/opt/module/datax/job/import/gmall.coupon_info.json
/origin_data/gmall/db/coupon_info_full/$do_date
  import_data
/opt/module/datax/job/import/gmall.sku_attr_value.json
/origin_data/gmall/db/sku_attr_value_full/$do_date
  import_data  /opt/module/datax/job/import/gmall.sku_info.json
/origin_data/gmall/db/sku_info_full/$do_date
  import_data
/opt/module/datax/job/import/gmall.sku_sale_attr_value.json
/origin_data/gmall/db/sku_sale_attr_value_full/$do_date
  import_data  /opt/module/datax/job/import/gmall.spu_info.json
/origin_data/gmall/db/spu_info_full/$do_date
  ;;
esac
```

2）为 mysql_to_hdfs_full.sh 增加执行权限

```
[atguigu@hadoop102 bin]$ chmod +x ~/bin/mysql_to_hdfs_full.sh
```

3）测试同步脚本

```
[atguigu@hadoop102 bin]$ mysql_to_hdfs_full.sh all 2020-06-14
```

4）检查同步结果

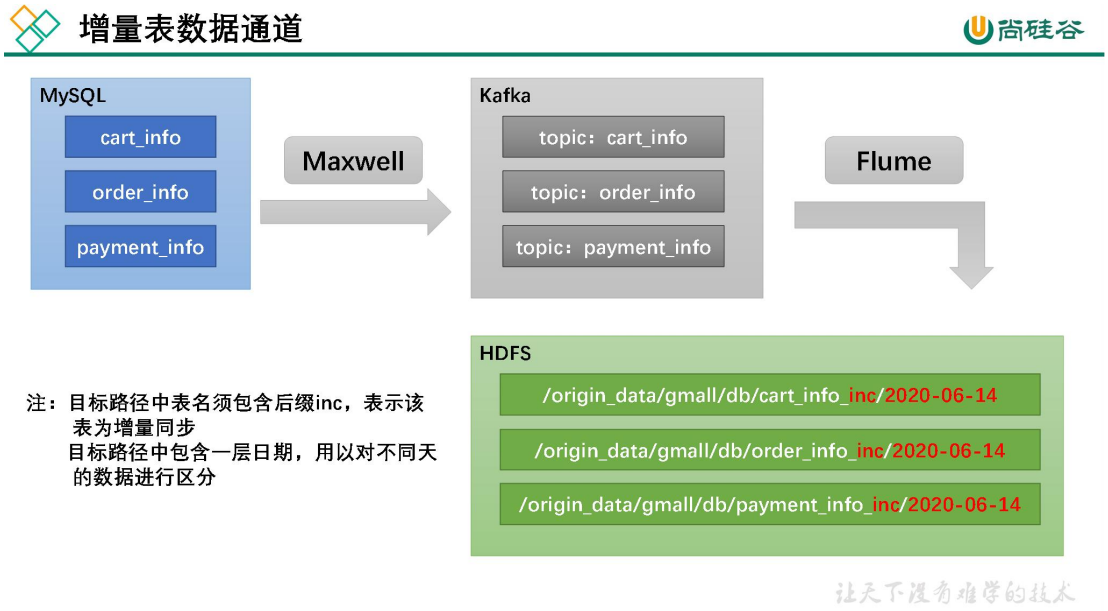
查看 HDFS 目表路径是否出现全量表数据，全量表共 15 张。

3.2.6 全量表同步总结

全量表同步逻辑比较简单，只需每日执行全量表数据同步脚本 mysql_to_hdfs_full.sh 即可。

3.3 增量表数据同步

3.3.1 数据通道



3.3.2 Maxwell 配置

按照规划，有 cart_info、comment_info 等共计 13 张表需进行增量同步，默认情况下，Maxwell 会同步 binlog 中的所有表的数据变更记录，因此我们需要对 Maxwell 进行配置，另其只同步这特定的 13 张表。

另外，为方便下游使用数据，还需对 Maxwell 进行配置，另其将不同表的数据发往不同的 Kafka Topic。Maxwell 最终配置如下：

1）修改 Maxwell 配置文件 config.properties

```
[atguigu@hadoop102 maxwell]$ vim /opt/module/maxwell/config.properties
```

2）全部配置参数如下

```
log_level=info

producer=kafka
kafka.bootstrap.servers=hadoop102:9092,hadoop103:9092

#kafka topic 动态配置
kafka_topic=%{table}
# mysql login info
host=hadoop102
user=maxwell
password=maxwell
jdbc_options=useSSL=false&serverTimezone=Asia/Shanghai

#表过滤，只同步特定的 13 张表
```

```
filter=
include:gmall.cart_info,include:gmall.comment_info,include:gmall.coupon_use,include:gmall.favor_info,include:gmall.order_detail,include:gmall.order_detail_activity,include:gmall.order_detail_coupon,include:gmall.order_info,include:gmall.order_refund_info,include:gmall.order_status_log,include:gmall.payment_info,include:gmall.refund_payment,include:gmall.user_info
```

3）重新启动 Maxwell

```
[atguigu@hadoop102 bin]$ mxw.sh restart
```

4）通道测试

（1）启动 Zookeeper 以及 Kafka 集群

（2）启动一个 Kafka Console Consumer，消费任一 topic 数据

```
[atguigu@hadoop103 kafka]$ bin/kafka-console-consumer.sh --bootstrap-server hadoop102:9092 --topic cart_info
```

（3）生成模拟数据

```
[atguigu@hadoop102 bin]$ cd /opt/module/db_log/
[atguigu@hadoop102 db_log]$ java -jar gmall2020-mock-db-2021-11-14.jar
```

（4）观察 Kafka 消费者是否能消费到数据

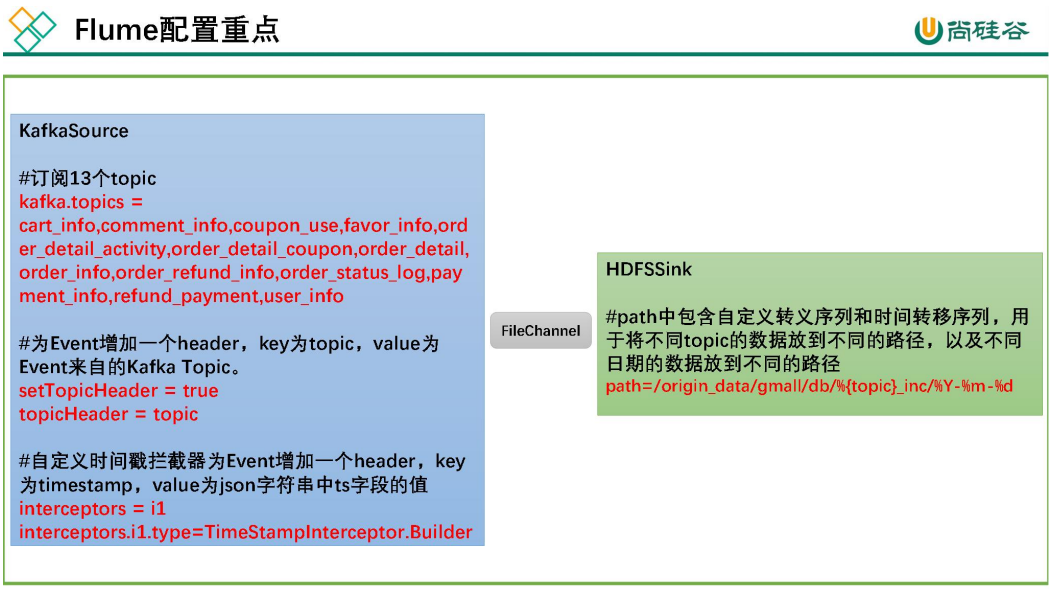
```
{"database":"gmall","table":"cart_info","type":"update","ts":1592270938,"xid":13090,"xoffset":1573,"data":{"id":100924,"user_id":"93","sku_id":16,"cart_price":4488.00,"sku_num":1,"img_url":"http://47.93.148.192:8080/group1/M00/00/02/rBHu8l-sklaALrngAAHGDqdpFtU741.jpg","sku_name":"华为 HUAWEI P40 麒麟 990 5G SoC 芯片 5000 万超感知徕卡三摄 30 倍数字变焦 8GB+128GB 亮黑色全网通 5G 手机","is_checked":null,"create_time":"2020-06-14 09:28:57","operate_time":null,"is_ordered":1,"order_time":"2021-10-17 09:28:58","source_type":"2401","source_id":null},"old":{"is_ordered":0,"order_time":null}}
```

3.3.3 Flume 配置

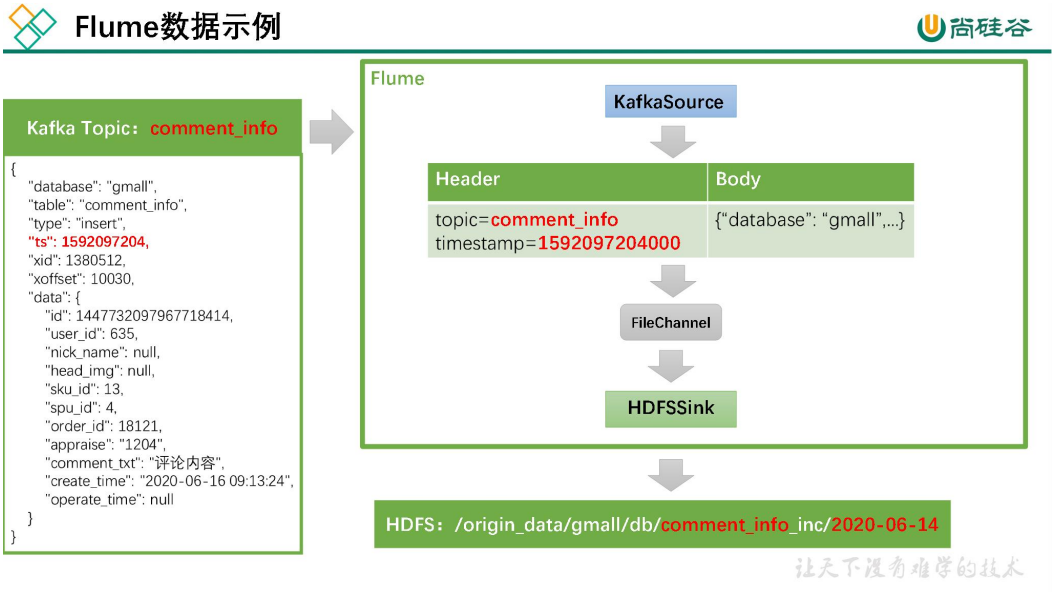
1）Flume 配置概述

Flume 需要将 Kafka 中各 topic 的数据传输到 HDFS，故其需选用 KafkaSource 以及 HDFSSink，Channe 选用 FileChanne。

需要注意的是，KafkaSource 需订阅 Kafka 中的 13 个 topic，HDFSSink 需要将不同 topic 的数据写到不同的路径，并且路径中应当包含一层日期，用于区分每天的数据。关键配置如下：



具体数据示例如下：



2）Flume 配置实操

（1）创建 Flume 配置文件

在 hadoop104 节点的 Flume 的 job 目录下创建 kafka_to_hdfs_db.conf

```
[atguigu@hadoop104 flume]$ mkdir job
[atguigu@hadoop104 flume]$ vim job/kafka_to_hdfs_db.conf
```

（2）配置文件内容如下

```
a1.sources = r1
a1.channels = c1
a1.sinks = k1

a1.sources.r1.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.r1.batchSize = 5000
a1.sources.r1.batchDurationMillis = 2000
```

```
a1.sources.r1.kafka.bootstrap.servers =
hadoop102:9092,hadoop103:9092
a1.sources.r1.kafka.topics =
cart_info,comment_info,coupon_use,favor_info,order_detail_acti
vity,order_detail_coupon,order_detail,order_info,order_refund_
info,order_status_log,payment_info,refund_payment,user_info
a1.sources.r1.kafka.consumer.group.id = flume
a1.sources.r1.setTopicHeader = true
a1.sources.r1.topicHeader = topic
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type =
com.atguigu.flume.interceptor.db.TimestampInterceptor$Builder


a1.channels.c1.type = file
a1.channels.c1.checkpointDir =
/opt/module/flume/checkpoint/behavior2
a1.channels.c1.dataDirs = /opt/module/flume/data/behavior2/
a1.channels.c1.maxFileSize = 2146435071
a1.channels.c1.capacity = 1123456
a1.channels.c1.keep-alive = 6

## sink1
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path =
/origin_data/gmall/db/%{topic}_inc/%Y-%m-%d
a1.sinks.k1.hdfs.filePrefix = db
a1.sinks.k1.hdfs.round = false


a1.sinks.k1.hdfs.rollInterval = 10
a1.sinks.k1.hdfs.rollSize = 134217728
a1.sinks.k1.hdfs.rollCount = 0


a1.sinks.k1.hdfs.fileType = CompressedStream
a1.sinks.k1.hdfs.codeC = gzip

## 拼装
a1.sources.r1.channels = c1
a1.sinks.k1.channel= c1
```

（3）编写 Flume 拦截器

①新建一个 Maven 项目，并在 pom.xml 文件中加入如下配置

```
<dependencies>
    <dependency>
        <groupId>org.apache.flume</groupId>
        <artifactId>flume-ng-core</artifactId>
        <version>1.9.0</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
        <version>1.2.62</version>
```

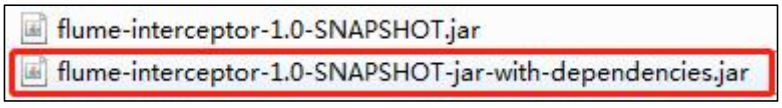
```
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <configuration>
                <descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
            </configuration>
            <executions>
                <execution>
                    <id>make-assembly</id>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

②在 com.atguigu.flume.interceptor.db 包下创建 TimestampInterceptor 类

```
package com.atguigu.flume.interceptor.db;

import com.alibaba.fastjson.JSONObject;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.interceptor.Interceptor;

import java.nio.charset.StandardCharsets;
import java.util.List;
import java.util.Map;

public class TimestampInterceptor implements Interceptor {
    @Override
    public void initialize() {

    }

    @Override
    public Event intercept(Event event) {

        Map<String, String> headers = event.getHeaders();
        String log = new String(event.getBody(),
```

```
StandardCharsets.UTF_8);

        JSONObject jsonObject = JSONObject.parseObject(log);

        Long ts = jsonObject.getLong("ts");

        //Maxwell 输出的数据中的 ts 字段时间戳单位为秒，Flume HDFSSink
要求单位为毫秒
        String timeMills = String.valueOf(ts * 1000);

        headers.put("timestamp", timeMills);

        return event;

    }

    @Override
    public List<Event> intercept(List<Event> events) {

        for (Event event : events) {
            intercept(event);
        }

        return events;
    }

    @Override
    public void close() {

    }

    public static class Builder implements Interceptor.Builder {


        @Override
        public Interceptor build() {
            return new TimestampInterceptor();
        }

        @Override
        public void configure(Context context) {

        }
    }
}
```

③重新打包



④将打好的包放入到 hadoop104 的/opt/module/flume/lib 文件夹下

```
[atguigu@hadoop102 lib]$ ls | grep interceptor
flume-interceptor-1.0-SNAPSHOT-jar-with-dependencies.jar
```

3）通道测试

（1）启动 Zookeeper、Kafka 集群

（2）启动 hadoop104 的 Flume

```
[atguigu@hadoop104 flume]$ bin/flume-ng agent -n a1 -c conf/ -f job/kafka_to_hdfs_db.conf -Dflume.root.logger=info,console
```

（3）生成模拟数据

```
[atguigu@hadoop102 bin]$ cd /opt/module/db_log/
[atguigu@hadoop102 db_log]$ java -jar gmall2020-mock-db-2021-11-14.jar
```

（4）观察 HDFS 上的目标路径是否有数据出现

若 HDFS 上的目标路径已有增量表的数据出现了，就证明数据通道已经打通。

（5）数据目标路径的日期说明

仔细观察，会发现目标路径中的日期，并非模拟数据的业务日期，而是当前日期。这是由于 Maxwell 输出的 JSON 字符串中的 ts 字段的值，是数据的变动日期。而真实场景下，数据的业务日期与变动日期应当是一致的。

此处为了模拟真实环境，对 Maxwell 源码进行了改动，增加了一个参数 mock_date，该参数的作用就是指定 Maxwell 输出 JSON 字符串的 ts 时间戳的日期，接下来进行测试。

①修改 Maxwell 配置文件 config.properties，增加 mock_date 参数，如下

```
# 该日期须和 /opt/module/db_log/application.properties 中的 mock.date 参数保持一致
mock_date=2020-06-14
```

注：该参数仅供学习使用，**修改该参数后重启 Maxwell 才可生效**。

②重启 Maxwell

```
[atguigu@hadoop102 bin]$ mxw.sh restart
```

③重新生成模拟数据

```
[atguigu@hadoop102 bin]$ cd /opt/module/db_log/
[atguigu@hadoop102 db_log]$ java -jar gmall2020-mock-db-2021-11-14.jar
```

④观察 HDFS 目标路径日期是否正常

4）编写 Flume 启停脚本

为方便使用，此处编写一个 Flume 的启停脚本

1）在 hadoop102 节点的/home/atguigu/bin 目录下创建脚本 f3.sh

```
[atguigu@hadoop102 bin]$ vim f3.sh
```

在脚本中填写如下内容

```
#!/bin/bash

case $1 in
```

```
"start")
       echo " --------启动 hadoop104 业务数据 flume-------"
       ssh hadoop104 "nohup /opt/module/flume/bin/flume-ng agent
-n         a1         -c         /opt/module/flume/conf         -f
/opt/module/flume/job/kafka_to_hdfs_db.conf >/dev/null 2>&1 &"
;;
"stop")

       echo " --------停止 hadoop104 业务数据 flume-------"
       ssh hadoop104 "ps -ef | grep kafka_to_hdfs_db.conf | grep
-v grep |awk '{print \$2}' | xargs -n1 kill"
;;
esac
```

2）增加脚本执行权限

```
[atguigu@hadoop102 bin]$ chmod +x f3.sh
```

3）f3 启动

```
[atguigu@hadoop102 module]$ f3.sh start
```

4）f3 停止

```
[atguigu@hadoop102 module]$ f3.sh stop
```

3.3.4 增量表首日全量同步

通常情况下，增量表需要在首日进行一次全量同步，后续每日再进行增量同步，首日全量同步可以使用 Maxwell 的 bootstrap 功能，方便起见，下面编写一个增量表首日全量同步脚本。

1）在~/bin 目录创建 mysql_to_kafka_inc_init.sh

```
[atguigu@hadoop102 bin]$ vim mysql_to_kafka_inc_init.sh
```

脚本内容如下

```
#!/bin/bash

# 该脚本的作用是初始化所有的增量表，只需执行一次

MAXWELL_HOME=/opt/module/maxwell

import_data() {
    $MAXWELL_HOME/bin/maxwell-bootstrap --database gmall --table $1
--config $MAXWELL_HOME/config.properties
}

case $1 in
"cart_info")
  import_data cart_info
  ;;
"comment_info")
  import_data comment_info
  ;;
"coupon_use")
  import_data coupon_use
  ;;
"favor_info")
  import_data favor_info
```

```
  ;;
"order_detail")
  import_data order_detail
  ;;
"order_detail_activity")
  import_data order_detail_activity
  ;;
"order_detail_coupon")
  import_data order_detail_coupon
  ;;
"order_info")
  import_data order_info
  ;;
"order_refund_info")
  import_data order_refund_info
  ;;
"order_status_log")
  import_data order_status_log
  ;;
"payment_info")
  import_data payment_info
  ;;
"refund_payment")
  import_data refund_payment
  ;;
"user_info")
  import_data user_info
  ;;
"all")
  import_data cart_info
  import_data comment_info
  import_data coupon_use
  import_data favor_info
  import_data order_detail
  import_data order_detail_activity
  import_data order_detail_coupon
  import_data order_info
  import_data order_refund_info
  import_data order_status_log
  import_data payment_info
  import_data refund_payment
  import_data user_info
  ;;
esac
```

2）为 mysql_to_kafka_inc_init.sh 增加执行权限

```
[atguigu@hadoop102 bin]$ chmod +x ~/bin/mysql_to_kafka_inc_init.sh
```

3）测试同步脚本

（1）清理历史数据

为方便查看结果，现将 HDFS 上之前同步的增量表数据删除

```
[atguigu@hadoop102 ~]$ hadoop fs -ls /origin_data/gmall/db | grep _inc | awk '{print $8}' | xargs hadoop fs -rm -r -f
```

（2）执行同步脚本

```
[atguigu@hadoop102 bin]$ mysql_to_kafka_inc_init.sh all
```

4）检查同步结果

观察 HDFS 上是否重新出现增量表数据。

3.3.5 增量表同步总结

增量表同步，需要在首日进行一次全量同步，后续每日才是增量同步。首日进行全量同步时，需先启动数据通道，包括 Maxwell、Kafka、Flume，然后执行增量表首日同步脚本 mysql_to_kafka_inc_init.sh 进行同步。后续每日只需保证采集通道正常运行即可，Maxwell 便会实时将变动数据发往 Kafka。

第 4 章 数仓环境准备

4.1 Hive 安装部署

1）把 apache-hive-3.1.2-bin.tar.gz 上传到 linux 的/opt/software 目录下

2）解压 apache-hive-3.1.2-bin.tar.gz 到/opt/module/目录下面

```
[atguigu@hadoop102 software]$ tar -zxvf /opt/software/apache-hive-3.1.2-bin.tar.gz -C /opt/module/
```

3）修改 apache-hive-3.1.2-bin.tar.gz 的名称为 hive

```
[atguigu@hadoop102 software]$ mv /opt/module/apache-hive-3.1.2-bin/ /opt/module/hive
```

4）修改/etc/profile.d/my_env.sh，添加环境变量

```
[atguigu@hadoop102 software]$ sudo vim /etc/profile.d/my_env.sh
```

添加内容

```
#HIVE_HOME
export HIVE_HOME=/opt/module/hive
export PATH=$PATH:$HIVE_HOME/bin
```

重启 Xshell 对话框或者 source 一下 /etc/profile.d/my_env.sh 文件，使环境变量生效

```
[atguigu@hadoop102 software]$ source /etc/profile.d/my_env.sh
```

5）解决日志 Jar 包冲突，进入/opt/module/hive/lib 目录

```
[atguigu@hadoop102 lib]$ mv log4j-slf4j-impl-2.10.0.jar log4j-slf4j-impl-2.10.0.jar.bak
```

4.2 Hive 元数据配置到 MySQL

4.2.1 拷贝驱动

将 MySQL 的 JDBC 驱动拷贝到 Hive 的 lib 目录下

```
[atguigu@hadoop102 lib]$ cp /opt/software/mysql-connector-java-5.1.27.jar /opt/module/hive/lib/
```

4.2.2 配置 MySQL 作为元数据存储

在$HIVE_HOME/conf 目录下新建 hive-site.xml 文件

```
[atguigu@hadoop102 conf]$ vim hive-site.xml
```

添加如下内容

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://hadoop102:3306/metastore?useSSL=false</val
ue>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.jdbc.Driver</value>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>root</value>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>123456</value>
    </property>

    <property>
        <name>hive.metastore.warehouse.dir</name>
        <value>/user/hive/warehouse</value>
    </property>

    <property>
        <name>hive.metastore.schema.verification</name>
        <value>false</value>
    </property>

    <property>
    <name>hive.server2.thrift.port</name>
    <value>10000</value>
    </property>

    <property>
        <name>hive.server2.thrift.bind.host</name>
        <value>hadoop102</value>
    </property>

    <property>

<name>hive.metastore.event.db.notification.api.auth</name>
        <value>false</value>
    </property>

    <property>
        <name>hive.cli.print.header</name>
        <value>true</value>
    </property>
```

```
    <property>
        <name>hive.cli.print.current.db</name>
        <value>true</value>
    </property>
</configuration>
```

4.3 启动 Hive

4.3.1 初始化元数据库

1）登陆 MySQL

```
[atguigu@hadoop102 conf]$ mysql -uroot -p123456
```

2）新建 Hive 元数据库

```
mysql> create database metastore;
mysql> quit;
```

3）初始化 Hive 元数据库

```
[atguigu@hadoop102 conf]$ schematool -initSchema -dbType mysql
-verbose
```

4.3.2 启动 hive 客户端

1）启动 Hive 客户端

```
[atguigu@hadoop102 hive]$ bin/hive
```

2）查看一下数据库

```
hive (default)> show databases;
OK
database_name
default
```

4.4 修改元数据库字符集

Hive 元数据库的字符集默认为 Latin1，由于其不支持中文字符，故若建表语句中包含中文注释，会出现乱码现象。如需解决乱码问题，须做以下修改。

1）修改 Hive 元数据库中存储注释的字段的字符集为 utf-8

（1）字段注释

```
mysql> alter table COLUMNS_V2 modify column COMMENT varchar(256)
character set utf8;
```

（2）表注释

```
mysql> alter table TABLE_PARAMS modify column PARAM_VALUE
mediumtext character set utf8;
```

2）修改 hive-site.xml 中 JDBC URL，如下

```
    <property>
        <name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://hadoop102:3306/metastore?useSSL=false&amp;
useUnicode=true&amp;characterEncoding=UTF-8</value>
    </property>
```