

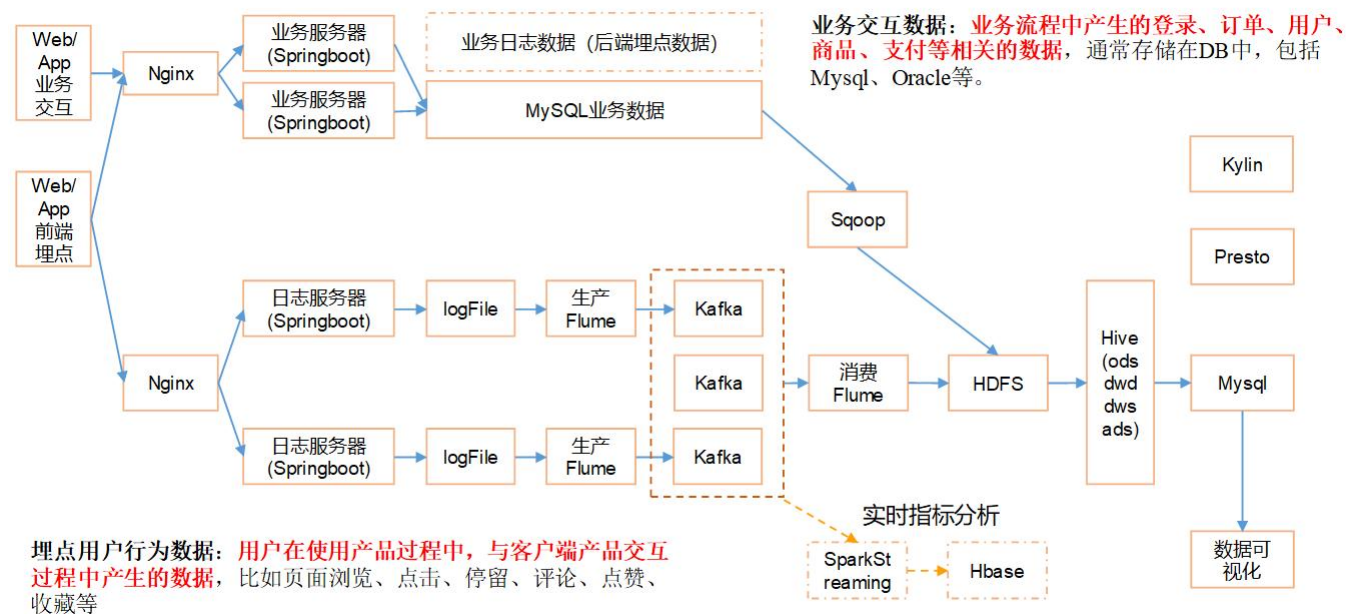
第 1 章 项目架构

1.1 数仓概念

输入系统：埋点产生的用户行为数据、JavaEE 后台产生的业务数据、个别公司有爬虫数据。

输出系统：报表系统、用户画像系统、推荐系统

1.2 系统数据流程设计



1.3 框架版本选型

- 1) Apache: 运维麻烦，组件间兼容性需要自己调研。（一般大厂使用，技术实力雄厚，有专业的运维人员）
- 2) CDH6.3.2: 国内使用最多的版本，但 CM 不开源，但其实对中、小公司使用来说没有影响（建议使用）10000美金一个节点 CDP7.0
- 3) HDP2.7: 开源，可以进行二次开发，但是没有 CDH 稳定，国内使用较少

Apache 框架版本

产品	版本
Hadoop	2.7.2
Flume	1.7.0
Kafka	0.11.0.2
Kafka Manager	1.3.3.22
Hive	1.2.1
Sqoop	1.4.6
Mysql	5.6.24
Azkaban	2.5.0
Java	1.8
Zookeeper	3.4.10
Presto	0.189

CDH 框架版本

产品	版本
Hadoop	2.6.0
Spark	1.6.0
Flume	1.6.0
Hive	1.1.0
Sqoop	1.4.6
Oozie	4.1.0
Zookeeper	3.4.5
Impala	2.9.0

框架选型最好选择最新框架半年前的稳定版。

1.4 服务器选型

服务器使用物理机还是云主机？

1) 机器成本考虑：

(1) 物理机：以 128G 内存，20 核物理 CPU，40 线程，8THDD 和 2TSSD 硬盘，单台报价 4W 出头，惠普品牌。一般物理机寿命 5 年左右。

(2) 云主机，以阿里云为例，差不多相同配置，每年 5W

2) 运维成本考虑：

(1) 物理机：需要有专业的运维人员（1 万*13 个月）、电费（商业用户）、安装空调

(2) 云主机：很多运维工作都由阿里云已经完成，运维相对较轻松

3) 企业选择

(1) 金融有钱公司和阿里没有直接冲突的公司选择阿里云（上海）

(2) 中小公司、为了融资上市，选择阿里云，拉倒融资后买物理机。

(3) 有长期打算，资金比较足，选择物理机。

1.5 集群规模

1) 用户行为数据

(1) 每天日活跃用户 50W，每人一天平均 10 条：50 万*10 条=500 万条

(2) 每条日志 1K 左右：500 万*1K/1024/1024=约 4.8G

(3) ODS 层（LZO）：4.8G 压缩为 0.5G

(4) DWD 层（LZO+parquet）：0.5G 左右

(5) DWS 和 DWT 层属于轻度聚合存储：2.5G 左右

(6) ADS 层忽略不计

(7) 保存 3 副本：3.5*3=10.5G

(8) 半年内不扩容：10.5*180=8190G=约 8T

(9) 预留 20%~30%的空间：8/0.7=约 11.5T

2) kafka 中的数据

太少，忽略不计

3) Flume 中的数据

太少，忽略不计

4) 业务数据

半年内不到 100G

集群规模 12T

5) 服务器

集群规模 12T，实际搭建 48T

选用 32G/8T/10 核 20 线程 6 台 datanode（3 台 Hbase）、（3 台 zk、kafka、flume）
 选用 32G/4T/10 核 20 线程 2 台 namenode（hive、mysql、redis、spark）
 选用 16G/1.2T/6 核心 12 线程 2 台 datanode（ES）
 根据数据规模搭建集群

1	2	3	4	5	6	7	8	9	10
nn	nn	dn	dn	dn	dn	dn	dn	dn	dn
		rm	rm	nm	nm	nm	nm	nm	nm
		nm	nm						
							zk	zk	zk
							kafka	kafka	kafka
							Flume	Flume	flume
		Hbase	Hbase	Hbase					
hive	hive								
mysql	mysql								
spark	spark								
					ES	ES			

- 1) 消耗内存的分开；
- 2) kafka 、zk 、flume 传输数据比较紧密的放在一起；
- 3) 客户端尽量放在一到两台服务器上，方便外部访问；

第 2 章 数仓分层

2.1 数据仓库建模（绝对重点）

2.1.1 建模工具是什么？

PowerDesigner/SQLYog/EZDML

2.1.2 ODS 层

- （1）保持数据原貌不做任何修改，起到备份数据的作用。
- （2）数据采用压缩，减少磁盘存储空间（例如：原始数据 100G，可以压缩到 10G 左右）
- （3）创建分区表，防止后续的全表扫描

2.1.3 DWD 层

DWD 层需构建维度模型，一般采用星型模型，呈现的状态一般为星座模型。

维度建模一般按照以下四个步骤：

选择业务过程→声明粒度→确认维度→确认事实

（1）选择业务过程

在业务系统中，如果业务表过多，挑选我们感兴趣的业务线，比如下单业务，支付业务，退款业务，物流业务，一条业务线对应一张事实表。如果小公司业务表比较少，建议选择所有业务线。

（2）声明粒度

数据粒度指数据仓库的数据中保存数据的细化程度或综合程度的级别。

声明粒度意味着精确定义事实表中的一行数据表示什么，应该尽可能**选择最小粒度**，以此来应各种各样的需求。

典型的粒度声明如下：

订单当中的每个商品项作为下单事实表中的一行，粒度为每次

每周的订单次数作为一行，粒度为每周。

每月的订单次数作为一行，粒度为每月。

如果在 DWD 层粒度就是每周或者每月，那么后续就没有办法统计细粒度的指标了。所有建议采用最小粒度。

（3）确定维度

维度的主要作用是描述业务是事实，主要表示的是“谁，何处，何时”等信息。例如：时间维度、用户维度、地区维度等常见维度。

(4) 确定事实

此处的“事实”一词，指的是业务中的度量值，例如订单金额、下单次数等。

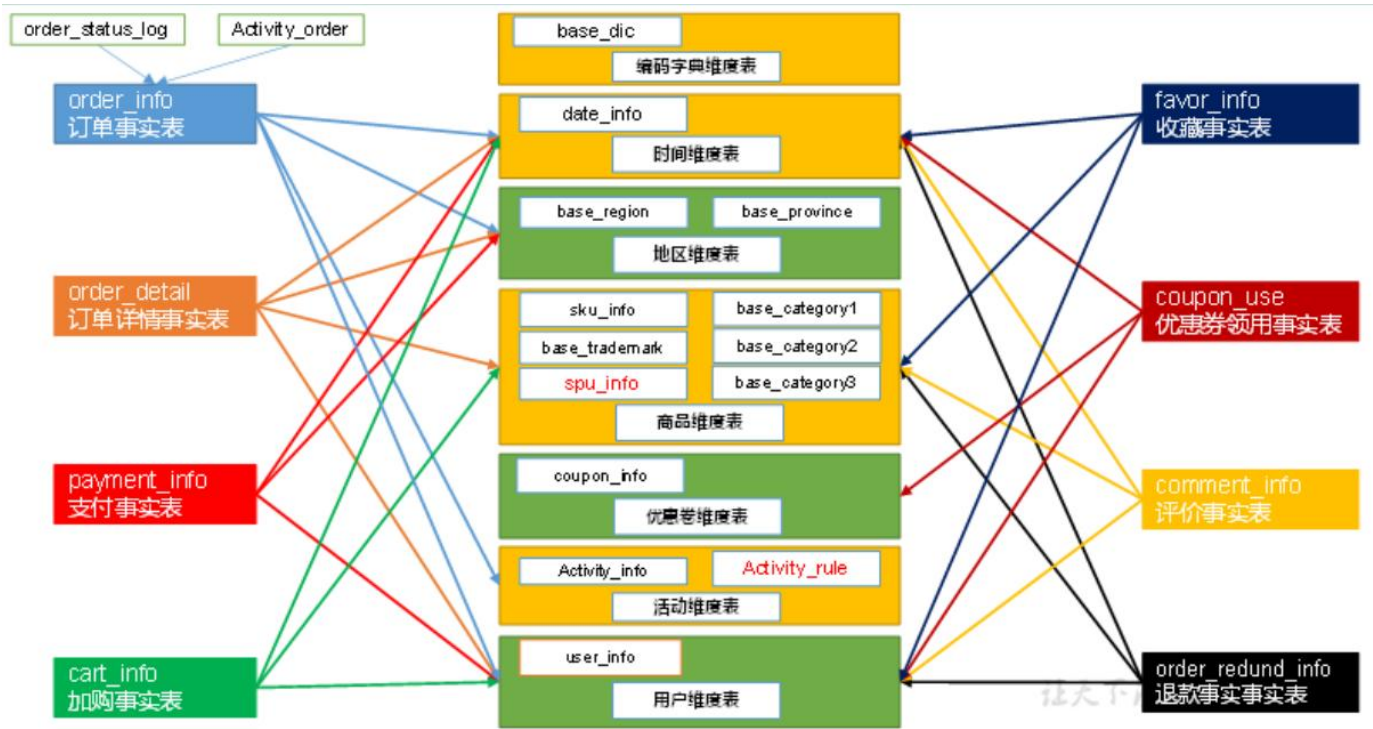
在 DWD 层，以业务过程为建模驱动，基于每个具体业务过程的特点，构建最细粒度的明细层事实表。事实表可做适当的宽表化处理。

通过以上步骤，结合本数仓的业务事实，得出业务总线矩阵表如下表所示。业务总线矩阵的原则，主要是根据维度表和事实表之间的关系，如果两者有关联则使用√标记。

表 业务总线矩阵表

	时间	用户	地区	商品	优惠券	活动	编码	度量值
订单	√	√	√			√		件数/金额
订单详情	√		√	√				件数/金额
支付	√		√					次数/金额
加购	√	√		√				件数/金额
收藏	√	√		√				个数
评价	√	√		√				个数
退款	√	√		√				件数/金额
优惠券领用	√	√			√			个数

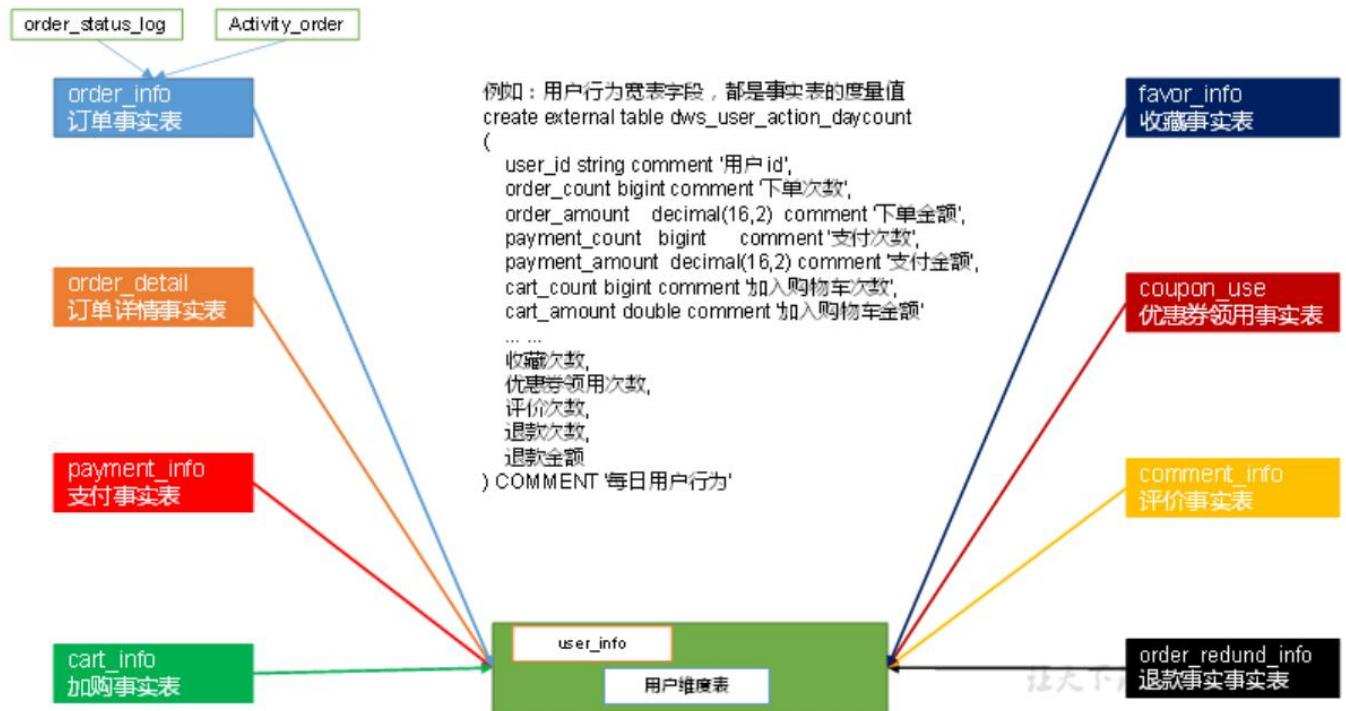
根据维度建模中的星型模型思想，将维度进行退化。例如下图所示：地区表和省份表退化为地区维度表，商品表、品类表、spu 表、商品三级分类、商品二级分类、商品一级分类表退化为商品维度表，活动信息表和活动规则表退化为活动维度表。



至此，数仓的维度建模已经完毕，DWS、DWT 和 ADS 和维度建模已经没有关系了。DWS 和 DWT 都是建宽表，宽表都是按照主题去建。主题相当于观察问题的角度，对应着维度表。

2.1.4 DWS 层

DWS 层统计各个主题对象的当天行为，服务于 DWT 层的主题宽表。如图所示，DWS 层的宽表字段，是站在不同维度的视角去看事实表，重点关注事实表的度量值，通过与之关联的事实表，获得不同的事实表的度量值。

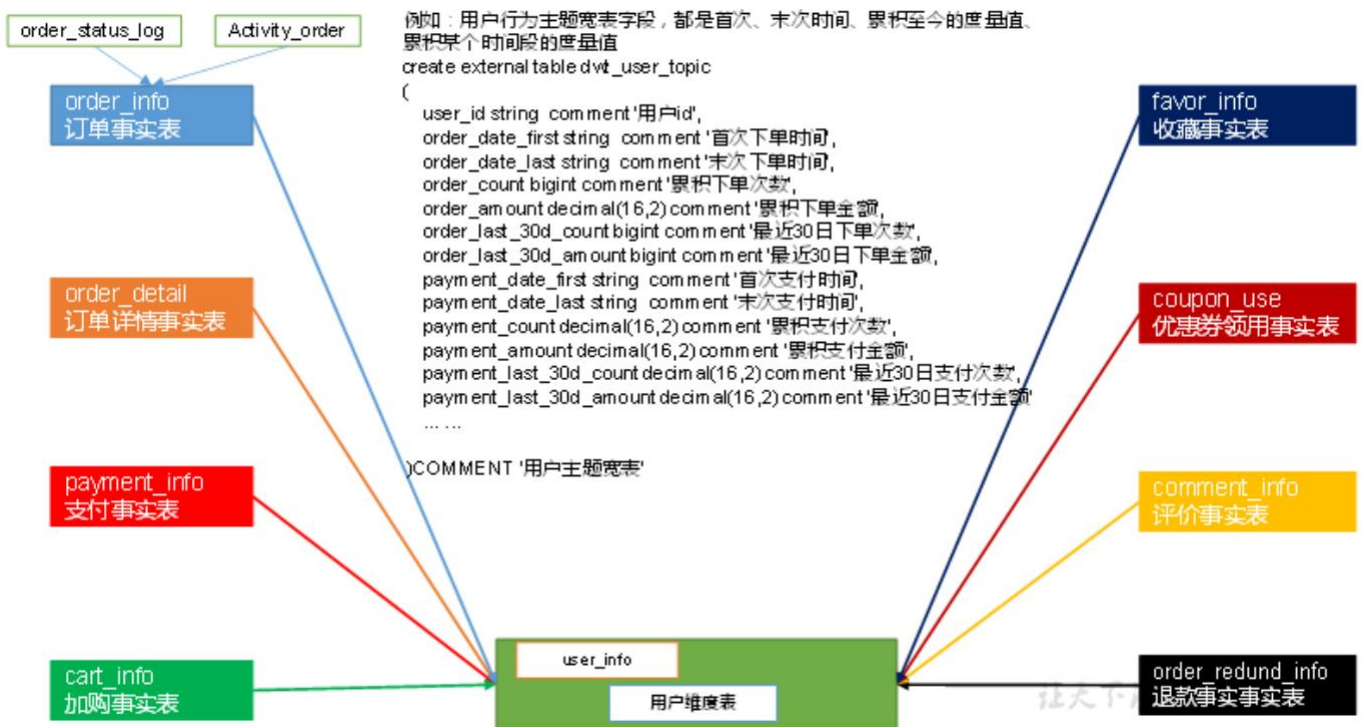


2.1.5 DWT 层

以分析的**主题对象**为建模驱动，基于上层应用和产品的指标需求，构建主题对象的**全量宽表**。

DWT 层主题宽表都记录什么字段？

如图所示，每个维度关联的不同事实表度量值以及首次、末次时间、累积至今的度量值、累积某个时间段的度量值。



2.1.6 ADS 层

分别对设备主题、会员主题、商品主题和营销主题进行指标分析，其中营销主题是用户主题和商品主题的跨主题分析案例

2.2 ODS 层做了哪些事？

- 1) 保持数据原貌，不做任何修改

2) 压缩采用 LZO，压缩比是 100g 数据压缩完 10g 左右。

3) 创建分区表

2.3 DWD 层做了哪些事？

2.3.1 数据清洗

1. 数据清洗

(1) 空值去除

(2) 过滤核心字段无意义的数，比如订单表中订单 id 为 null，支付表中支付 id 为空

(3) 将用户行为宽表和业务表进行数据一致性处理

```
select case when a is null then b else a end as JZR,
```

...

```
from A
```

2. 清洗的手段

Sql、mr、rdd、kettle、Python（项目中采用 sql 进行清除）

3. 清洗掉多少数据算合理

1 万条数据清洗掉 1 条

2.3.2 数据脱敏

对手机号、身份证号等敏感数据脱敏

2.3.3 维度退化

对业务数据传过来的表进行维度退化和降维。（商品一级二级三级、省市县、年月日）

2.3.4 压缩 LZO

2.3.5 列式存储 parquet

2.4 DWS 层做了哪些事？

2.4.1 DWS 层有 3-5 张宽表（处理 100-200 个指标、70% 以上的需求）

具体宽表名称：用户行为宽表，用户购买商品明细行为宽表，商品宽表，购物车宽表，物流宽表、登录注册、售后等。

2.4.2 哪个宽表最宽？大概有多少个字段？

最宽的是用户行为宽表。大概有 60-100 个字段

2.4.3 具体用户行为宽表字段名称

评论、打赏、收藏、关注--商品、关注--人、点赞、分享、好价爆料、文章发布、活跃、签到、补签卡、幸运屋、礼品、金币、电商点击、gmv

```
CREATE TABLE `app_usr_interact`(  
  `stat_dt` date COMMENT '互动日期',  
  `user_id` string COMMENT '用户 id',  
  `nickname` string COMMENT '用户昵称',  
  `register_date` string COMMENT '注册日期',  
  `register_from` string COMMENT '注册来源',  
  `remark` string COMMENT '细分渠道',  
  `province` string COMMENT '注册省份',  
  `pl_cnt` bigint COMMENT '评论次数',  
  `ds_cnt` bigint COMMENT '打赏次数',  
  `sc_add` bigint COMMENT '添加收藏',  
  `sc_cancel` bigint COMMENT '取消收藏',  
  `gzg_add` bigint COMMENT '关注商品',  
  `gzg_cancel` bigint COMMENT '取消关注商品',  
  `gzp_add` bigint COMMENT '关注人',
```



```

`gzp_cancel` bigint COMMENT '取消关注人',
`buzhi_cnt` bigint COMMENT '点不值次数',
`zhi_cnt` bigint COMMENT '点值次数',
`zan_cnt` bigint COMMENT '点赞次数',
`share_cnts` bigint COMMENT '分享次数',
`bl_cnt` bigint COMMENT '爆料数',
`fb_cnt` bigint COMMENT '好价发布数',
`online_cnt` bigint COMMENT '活跃次数',
`checkin_cnt` bigint COMMENT '签到次数',
`fix_checkin` bigint COMMENT '补签次数',
`house_point` bigint COMMENT '幸运屋金币抽奖次数',
`house_gold` bigint COMMENT '幸运屋积分抽奖次数',
`pack_cnt` bigint COMMENT '礼品兑换次数',
`gold_add` bigint COMMENT '获取金币',
`gold_cancel` bigint COMMENT '支出金币',
`surplus_gold` bigint COMMENT '剩余金币',
`event` bigint COMMENT '电商点击次数',
`gmv_amount` bigint COMMENT 'gmv',
`gmv_sales` bigint COMMENT '订单数')
PARTITIONED BY ( `dt` string)

```

2.5 ADS 层分析过哪些指标

2.5.1 分析过的指标（一分钟至少说出 30 个指标）

日活、月活、周活、留存、留存率、新增（日、周、年）、转化率、流失、回流、七天内连续 3 天登录（点赞、收藏、评价、购买、加购、下单、活动）、连续 3 周（月）登录、GMV、复购率、复购率排行、点赞、评论、收藏、领优惠价人数、使用优惠价、沉默、值不值得买、退款人数、退款率 topn 热门商品

产品经理最关心的：留转、复活



2.5.2 留转复活指标

(1) 活跃

日活: 100 万 ; 月活: 是日活的 2-3 倍 300 万

总注册的用户多少? 1000 万-3000 万之间

(2) GMV (网站成交金额)

GMV: 每天 10 万订单 (50 - 100 元) 500 万-1000 万

10%-20% 100 万-200 万 (人员: 程序员、人事、行政、财务、房租、收电费)

(3) 复购率

某日常商品复购: (手纸、面膜、牙膏) 10%-20%

电脑、显示器、手表 1%

(4) 转化率

商品详情 => 加购物车 => 下单 => 支付

5%-10%

60-70%

90%-95%

(5) 留存率

1/2/3、周留存、月留存

搞活动: 10-20%

2.6 ADS 层手写指标

2.6.1 如何分析用户活跃?

在启动日志中统计不同设备 id 出现次数。

2.6.2 如何分析用户新增?

用活跃用户表 left join 用户新增表, 用户新增表中 mid 为空的即为用户新增。

2.6.3 如何分析用户 1 天留存?

留存用户=前一天新增 join 今天活跃

用户留存率=留存用户/前一天新增

2.6.4 如何分析沉默用户?

(登录时间为 7 天前, 且只出现过一次)

按照设备 id 对日活表分组, 登录次数为 1, 且是在一周前登录。

2.6.5 如何分析本周回流用户?

本周活跃 left join 本周新增 left join 上周活跃, 且本周新增 id 和上周活跃 id 都为 null

2.6.6 如何分析流失用户?

(登录时间为 7 天前)

按照设备 id 对日活表分组, 且七天内没有登录过。

2.6.7 如何分析最近连续 3 周活跃用户数?

按照设备 id 对周活进行分组, 统计次数大于 3 次。

2.6.8 如何分析最近七天内连续三天活跃用户数?

1) 查询出最近 7 天的活跃用户, 并对用户活跃日期进行排名

2) 计算用户活跃日期及排名之间的差值

3) 对同用户及差值分组, 统计差值个数

4) 将差值相同个数大于等于 3 的数据取出, 然后去重(去的是什么重???), 即为连续 3 天及以上活跃的用户
7 天连续收藏、点赞、购买、加购、付款、浏览、商品点击、退货

1 个月连续 7 天

连续两周:

2.7 分析过最难的指标

2.7.1 最近连续 3 周活跃用户

最近3周连续活跃的用户: 通常是周一对前3周的数据做统计, 该数据一周计算一次。

1) 创建表

```
drop table if exists ads_continuity_wk_count;
create external table ads_continuity_wk_count(
  `dt` string COMMENT '统计日期, 一般用结束周周日日期, 如果每天计算一次, 可用当天日期',
  `wk_dt` string COMMENT '持续时间',
  `continuity_count` bigint
)
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_continuity_wk_count';
```

	前一周	前二周	前三周
	100	101	101
	101	102	104
	102		

2) 导入数据

```
insert into table ads_continuity_wk_count
select
  '2019-02-20',
  concat(date_add(next_day('2019-02-20', 'MO'), -7*3), '_', date_add(next_day('2019-02-20', 'MO'), -1)),
  count(*)
from
(
  select mid_id
  from dws_uv_detail_wk
  where wk_dt >= concat(date_add(next_day('2019-02-20', 'MO'), -7*3), '_', date_add(next_day('2019-02-20', 'MO'), -7*2-1))
  and wk_dt <= concat(date_add(next_day('2019-02-20', 'MO'), -7), '_', date_add(next_day('2019-02-20', 'MO'), -1))
  group by mid_id
  having count(*) = 3
) t1;
```

按照设备id对每周活跃表分组

统计次数等于3

100	1
101	3
102	2
104	1

让天下没有难学的技术

2.7.2 最近 7 天连续 3 天活跃用户数

1) 创建表

```
drop table if exists ads_continuity_uv_count;
create external table ads_continuity_uv_count(
  `dt` string COMMENT '统计日期',
  `wk_dt` string COMMENT '最近7天日期',
  `continuity_count` bigint
) COMMENT '连续活跃设备数'
row format delimited fields terminated by '\t'
location '/warehouse/gmall/ads/ads_continuity_uv_count';
```

减-			
mid_id=1	2019-02-06 1	2019-02-05	-2019-02-05-3- 2019-02-06 3
	2019-02-07 2	2019-02-05	
	2019-02-08 3	2019-02-05	
	2019-02-10 4	2019-02-06	
	2019-02-11 5	2019-02-06	
	2019-02-12 6	2019-02-06	
mid_id=2	2019-02-10 1		
mid_id=3	2019-02-16 1	2019-02-15	-2019-02-15-2
	2019-02-17 2	2019-02-15	

2) 导入数据

```
insert into table ads_continuity_uv_count
select
  '2019-02-12',
  concat(date_add('2019-02-12',-6),'_', '2019-02-12'),
  count(*)
from
(
  select mid_id
  from
  (
    select mid_id,
      date_sub(dt,rank) date_dif
    from
    (
      select
        mid_id,
        dt,
        rank() over(partition by mid_id order by dt) rank
      from dws_uv_detail_day
      where dt>=date_add('2019-02-12',-6) and dt<='2019-02-12'
    )t1
    )t2
    group by mid_id,date_dif
    having count(*)>=3
  )t3
  group by mid_id
)t4;
```

2、计算用户活跃日期及排名之间的差值

1、查询出最近7天的活跃用户，并对用户活跃日期进行排名

3、对同用户及差值分组，统计差值个数

4、将差值相同个数大于等于3的数据取出

5、对数据去重 让天下没有难学的技术

第 3 章 生产经验—业务

3.1 电商常识

3.1.1 SKU 和 SPU

SKU：一台银色、128G 内存的、支持联通网络的 iPhoneX

SPU：iPhoneX

Tm_id：品牌 Id 苹果，包括 IPHONE，耳机，mac 等

3.1.2 订单表跟订单详情表区别？

订单表的订单状态会变化，订单详情表不会，因为没有订单状态。

订单表记录 user_id，订单 id 订单编号，订单的总金额 order_status，支付方式，订单状态等。

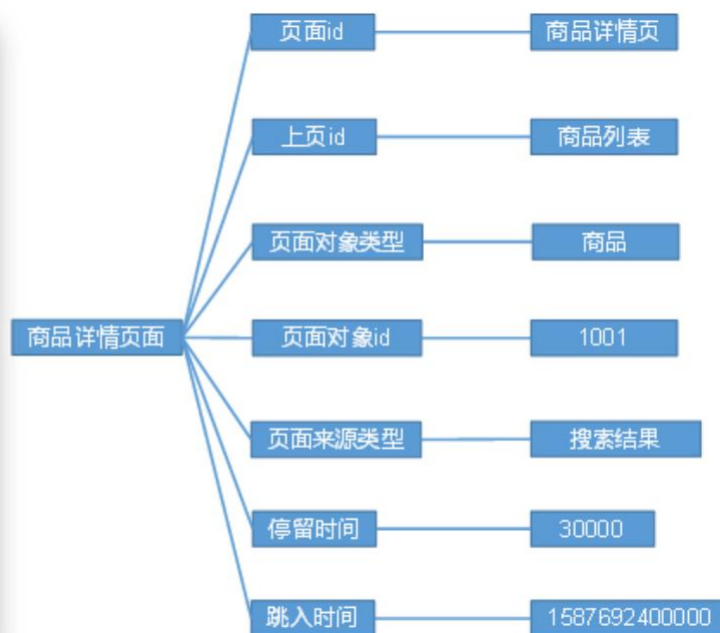
订单详情表记录 user_id，商品 sku_id，具体的商品信息（商品名称 sku_name，价格 order_price，数量 sku_num）

3.2 埋点行为数据基本格式(基本字段)

我们要收集和分析的数据主要包括页面数据、事件数据、曝光数据、启动数据和错误数据。

3.2.1 页面

页面数据主要记录一个页面的用户访问情况，包括访问时间、停留时间、页面路径等信息。



所有页面 id 如下

```

home("首页"),
category("分类页"),
discovery("发现页"),
top_n("热门排行"),
favor("收藏页"),
search("搜索页"),
good_list("商品列表页"),
good_detail("商品详情"),
good_spec("商品规格"),
comment("评价"),
comment_done("评价完成"),
comment_list("评价列表"),
cart("购物车"),
trade("下单结算"),
payment("支付页面"),
payment_done("支付完成"),
orders_all("全部订单"),
orders_unpaid("订单待支付"),
orders_undelivered("订单待发货"),
orders_unreceipted("订单待收货"),
orders_wait_comment("订单待评价"),
mine("我的"),
activity("活动"),
login("登录"),
register("注册");
  
```

所有页面对象类型如下:

```

sku_id("商品 skuld"),
keyword("搜索关键词"),
sku_ids("多个商品 skuld"),
activity_id("活动 id"),
coupon_id("购物券 id");
  
```

所有来源类型如下:

```

promotion("商品推广"),
  
```

```
recommend("算法推荐商品"),
query("查询结果商品"),
activity("促销活动");
```

3.2.2 事件

事件数据主要记录应用内一个具体操作行为，包括操作类型、操作对象、操作对象描述等信息。



所有动作类型如下：

```
favor_add("添加收藏"),
favor_cancel("取消收藏"),
cart_add("添加购物车"),
cart_remove("删除购物车"),
cart_add_num("增加购物车商品数量"),
cart_minus_num("减少购物车商品数量"),
trade_add_address("增加收货地址"),
get_coupon("领取优惠券");
```

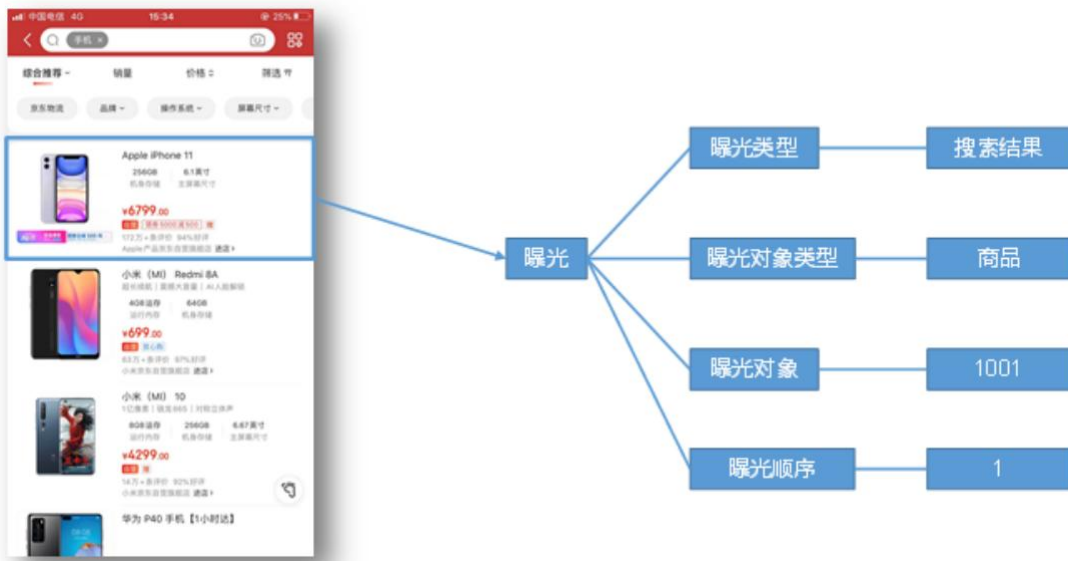
注：对于下单、支付等业务数据，可从业务数据库获取。

所有动作目标类型如下：

```
sku_id("商品"),
coupon_id("购物券");
```

4.2.3 曝光

曝光数据主要记录页面所曝光的内容，包括曝光对象，曝光类型等信息。



所有曝光类型如下：

```
promotion("商品推广"),
recommend("算法推荐商品"),
query("查询结果商品"),
activity("促销活动");
```

所有曝光对象类型如下：

```
sku_id("商品 skuld"),
activity_id("活动 id");
```

4.2.4 启动

启动数据记录应用的启动信息。



所有启动入口类型如下：

```
icon("图标"),
notification("通知"),
install("安装后启动");
```

4.2.5 错误

错误数据记录应用使用过程中的错误信息，包括错误编号及错误信息。

4.2.6 埋点数据日志格式

我们的日志结构大致可分为两类，一是普通页面埋点日志，二是启动日志。

普通页面日志结构如下，每条日志包含了，当前页面的**页面信息**，所有**事件（动作）**、所有**曝光信息**以及**错误信息**。除此之外，还包含了一系列**公共信息**，包括设备信息，地理位置，应用信息等，即下边的 **common** 字段。

```
{
  "common": {
    "ar": "230000",      -- 公共信息
    "ba": "iPhone",     -- 地区编码
    "ch": "Appstore",   -- 手机品牌
    "md": "iPhone 8",   -- 渠道
    "mid": "YXfhjAYH6As2z9Iq", -- 手机型号
    "os": "iOS 13.2.9", -- 设备 id
    "uid": "485",       -- 操作系统
    "vc": "v2.1.134"    -- 会员 id
  },
  "actions": [          -- app 版本号
    {
      "action_id": "favor_add", -- 动作 id
      "item": "3",             -- 目标 id
      "item_type": "sku_id",   -- 目标类型
    }
  ]
}
```

```

    "ts": 1585744376605      --动作时间戳
  }
],
"displays": [
  {
    "displayType": "query",    -- 曝光类型
    "item": "3",              -- 曝光对象 id
    "item_type": "sku_id",     -- 曝光对象类型
    "order": 1                --出现顺序
  },
  {
    "displayType": "promotion",
    "item": "6",
    "item_type": "sku_id",
    "order": 2
  },
  {
    "displayType": "promotion",
    "item": "9",
    "item_type": "sku_id",
    "order": 3
  },
  {
    "displayType": "recommend",
    "item": "6",
    "item_type": "sku_id",
    "order": 4
  },
  {
    "displayType": "query ",
    "item": "6",
    "item_type": "sku_id",
    "order": 5
  }
],
"page": {                    --页面信息
  "during_time": 7648,       -- 持续时间毫秒
  "item": "3",               -- 目标 id
  "item_type": "sku_id",     -- 目标类型
  "last_page_id": "login",   -- 上页类型
  "page_id": "good_detail",  -- 页面 ID
  "sourceType": "promotion"  -- 来源类型
},
"err": {                    --错误
  "error_code": "1234",      --错误码
  "msg": "*****"           --错误信息
},
"ts": 1585744374423      --跳入时间戳
}

```

启动日志结构相对简单，主要包含公共信息，启动信息和错误信息。

```

{
  "common": {
    "ar": "370000",
    "ba": "Honor",
    "ch": "wandoujia",
    "md": "Honor 20s",

```



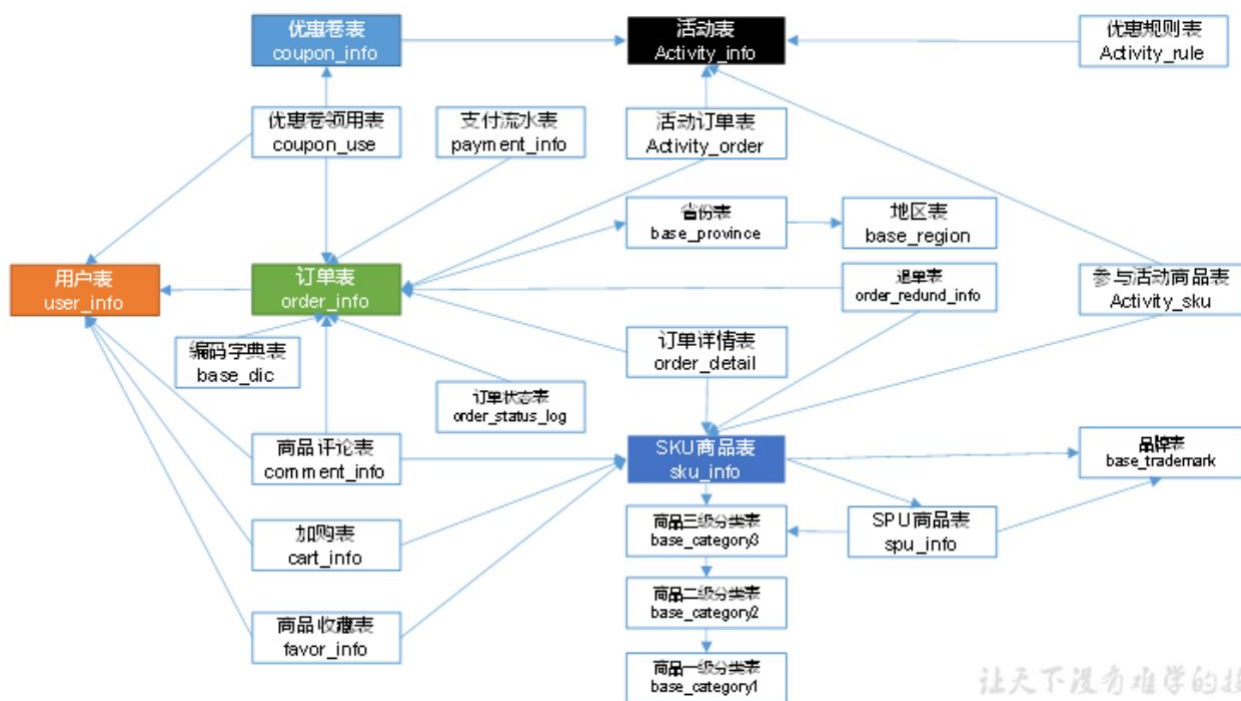
```

    "mid": "eQF5boERMJFOujcp",
    "os": "Android 11.0",
    "uid": "76",
    "vc": "v2.1.134"
  },
  "start": {
    "entry": "icon",          --icon 手机图标  notice 通知  install 安装后启动
    "loading_time": 18803,    --启动加载时间
    "open_ad_id": 7,          --广告页 ID
    "open_ad_ms": 3449,       -- 广告总共播放时间
    "open_ad_skip_ms": 1989   -- 用户跳过广告时点
  },
  "err": {
    "error_code": "1234",     --错误码
    "msg": "*****"          --错误信息
  },
  "ts": 1585744304000
}

```

3.3 电商业务流程

- 1) 记住表与表之间的关系
- 2) 每个表记住 2-3 个字段



3.4 维度表和事实表（重点）

3.4.1 维度表

维度表：一般是对事实的**描述信息**。每一张维表对应现实世界中的一个对象或者概念。例如：用户、商品、日期、地区等。

3.4.2 事实表

事实表中的每行数据代表一个业务事件（下单、支付、退款、评价等）。“事实”这个术语表示的是业务事件的**度量值（可统计次数、个数、件数、金额等）**，例如，订单事件中的下单金额。

每一个事实表的行包括：具有可加性的数值型的度量值、与维表相连接的外键、通常具有两个和两个以上的外键、外键之间表示维表之间多对多的关系。

1) 事务型事实表

以**每个事务或事件为单位**，例如一个销售订单记录，一笔支付记录等，作为事实表里的一行数据。一旦事务被提交，事实表数据被插入，数据就不再进行更改，其更新方式为增量更新。

2) 周期型快照事实表

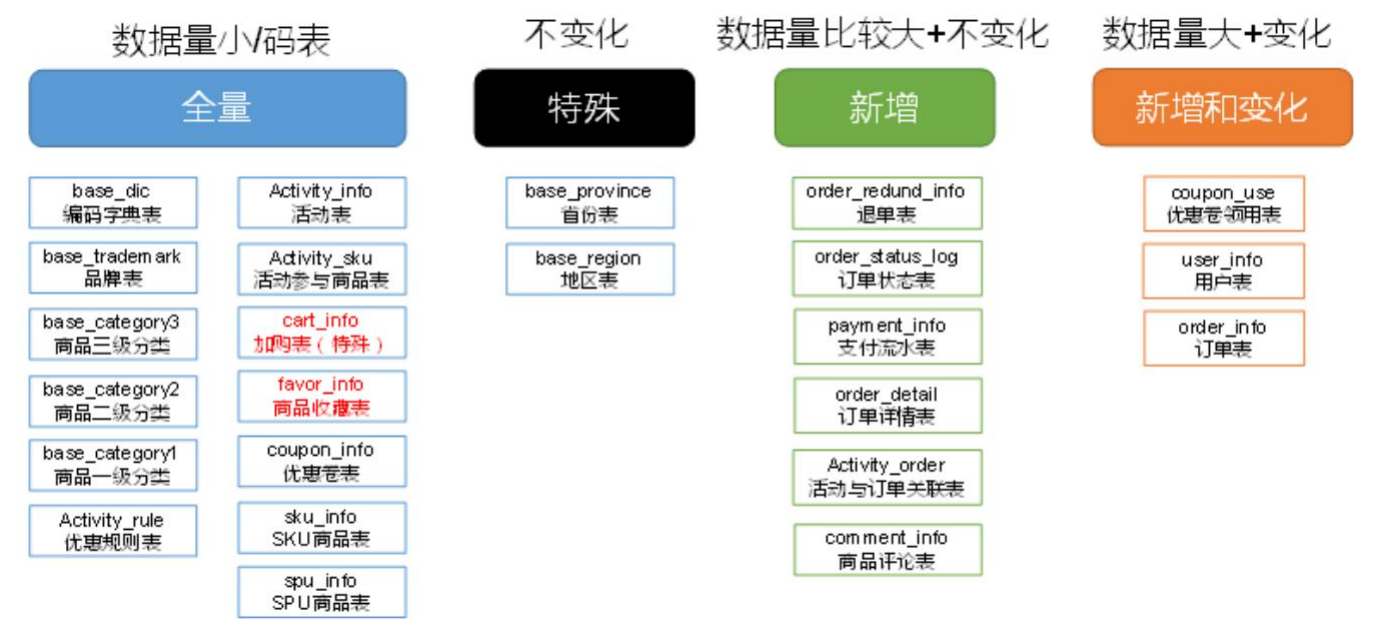
周期型快照事实表中**不会保留所有数据，只保留固定时间间隔的数据**，例如每天或者每月的销售额，或每月的账户余额等。

3) 累积型快照事实表

累计快照事实表用于跟踪业务事实的变化。例如，数据仓库中可能需要累积或者存储订单从下订单开始，到订单商品被打包、运输、和签收的各个业务阶段的时间点数据来跟踪订单声明周期的进展情况。当这个业务过程进行时，事实表的记录也要不断更新。

订单 id	用户 id	下单时间	打包时间	发货时间	签收时间	订单金额
		3-8	3-8	3-9	3-10	

3.5 同步策略（重点）



实体表，维度表统称维度表，每日全量或者每月（更长时间）全量

事务型事实表：每日增量

周期型事实表：拉链表

全量同步策略：

每日全量，导入完整数据到 hive 的分区表，就是每天存储一份完整数据，作为一个分区。适用于表数据量不大，且每日都有新数据插入，也会有旧数据修改的场景。

例如：编码字典表，且每天即会有数据插入，也会有旧数据的修改的场景。

增量同步策略：

每日增量，就是每天存储一份增量数据，作为一个分区。适用于数据量大，且每天只会有新数据插入的场景。

例如：退单表，订单状况表，支付流水表，订单详情表，活动与订单关联表，商品评论表。

新增及变化策略：

每日新增及变化，就是存储创建时间和操作时间都是今天的数据。使用场景：表的数据量大，既有新增，又会有变化。

例如：用户表（更适合用拉链表来实现），订单表，优惠券领用表。

拉链表：缓慢变化表。到一个特定时间，修改字段。

特殊策略：

只需要导入一次。（只能有一个固定值，比如：省份，地区表等字典表）

四种表

全量表：存储完整的数据，数据量不大的表（可能有变化）。
 增量表：存储新增加的数据，数据没有修改，但是会追加的情况。
 新增及变化表：存储新增加的数据和变化的数据，表的数据量比较大。
 特殊表：只需要存储一次。

3.6 关系型数据库范式理论

1NF：属性不可再分割（例如不能存在 5 台电脑的属性，坏处：表都没法用）
 2NF：不能存在部分函数依赖（例如主键（学号+课名）-->成绩，姓名，但学号-->姓名，所以姓名部分依赖于主键（学号+课名），所以要去除，坏处：数据冗余）
 3NF：不能存在传递函数依赖（学号-->宿舍种类-->价钱，坏处：数据冗余和增删异常）
 MySQL 关系模型：关系模型主要应用与 OLTP 系统中，为了保证数据的一致性以及避免冗余，所以大部分业务系统的表都是遵循第三范式的。
 Hive 维度模型：维度模型主要应用于 OLAP 系统中，因为关系模型虽然冗余少，但是在大规模数据，跨表分析统计查询过程中，会造成多表关联，这会大大降低执行效率。
 所以 HIVE 把相关各种表整理成两种：事实表和维度表两种。所有维度表围绕着事实表进行解释。

3.7 数据模型

雪花模型、星型模型和星座模型
 （在维度建模的基础上又分为三种模型：星型模型、雪花模型、星座模型。）
 星型模型（一级维度表），雪花（多级维度），星座模型（星型模型+多个事实表）

3.8 拉链表（重点）

拉链表处理的业务场景：主要处理缓慢变化维的业务场景。（用户表、订单表）

1）初始的拉链表（dwd_order_info_his）2019-01-01

订单 Id	状态	生效开始日期	生效结束日期
1	待支付	2019-01-01	9999-99-99
2	待支付	2019-01-01	9999-99-99
3	已支付	2019-01-01	9999-99-99

2）订单变化表（dwd_order_info）2019-01-02

订单 Id	状态
2	已支付
4	待支付
5	已支付

3）临时拉链表（dwd_order_info_his_tmp）2019-01-02

订单 Id	状态	生效开始日期	生效结束日期
1	待支付	2019-01-01	9999-99-99
2	待支付	2019-01-01	2019-01-01
2	已支付	2019-01-02	9999-99-99
3	已支付	2019-01-01	9999-99-99
4	待支付	2019-01-02	9999-99-99
5	已支付	2019-01-02	9999-99-99

```
insert overwrite table dwd_order_info_his_tmp
select * from
(
    select
        id,
        total_amount,
        order_status,
        user_id,
        payment_way,
        out_trade_no,
        create_time,
        operate_time,
        '2019-01-02' start_date,
        '9999-99-99' end_date
    from dwd_order_info where dt='2019-01-02'

    union all
    select
        oh.id,
        oh.total_amount,
        oh.order_status,
        oh.user_id,
        oh.payment_way,
        oh.out_trade_no,
        oh.create_time,
        oh.operate_time,
        oh.start_date,
        if(oi.id is null ,oh.end_date, date_add(oi.dt,-1)) end_date
    from dwd_order_info_his oh left join
    (
        select * from dwd_order_info
        where dt='2019-01-02'
    ) oi
    on oh.id=oi.id and oh.end_date='9999-99-99'
)his
order by his.id, start_date;
```

3.9 即席查询数据仓库

对比项目	Druid	Kylin	Presto	Impala	Spark SQL	ES
亚秒级响应	Y	Y	N	N	N	N
百亿数据集	Y	Y	Y	Y	Y	Y
SQL支持	N (开发中)	Y	Y	Y	Y	N
离线	Y	Y	Y	Y	Y	Y
实时	Y	N (开发中)	N	N	N	Y
精确去重	N	Y	Y	Y	Y	N
多表Join	N	Y	Y	Y	Y	N
JDBC for BI	N	Y	Y	Y	Y	N

4) **Spark SQL**: 基于Spark平台上的一个OLAP框架, 基本思路是**增加机器来并行计算**, 从而提高查询速度。

5) **ES**: 最大的特点是使用了倒排索引解决索引问题。根据研究, ES在数据获取和聚集用的资源比在Druid高。

1) **Druid**: 是一个实时处理时序数据的OLAP数据库, 因为它的索引首先按照时间分片, 查询的时候也是**按照时间线去路由索引**。

2) **Kylin**: 核心是Cube, Cube是一种预计算技术, 基本思路是预先对数据作多维索引, 查询时只扫描索引而不访问原始数据从而提速。

3) **Presto**: 它没有使用MapReduce, 大部分场景下比Hive快一个数量级, 其中的关键是**所有的处理都在内存中完成**。

4) **Impala**: **基于内存运算, 速度快, 支持的数据源没有Presto多。**

6) **框架选型**:

(1) 从超大数据的查询效率来看:

Druid > Kylin > Presto > Spark SQL

(2) 从支持的数据源种类来讲:

Presto > Spark SQL > Kylin > Druid

让天下没有难学的技术

Kylin: T+1

Impala: CDH

Presto: Apache 版本框架

3.10 数据仓库每天跑多少张表, 大概什么时候运行, 运行多久?

基本一个项目建一个库, 表格个数为初始的原始数据表格加上统计结果表格的总数。(一般 70-100 张表格)
 用户行为 5 张; 业务数据 23 张表 =>ods 24 =>dwd=>20 张=>dws 6 张宽表=>dwt6 张宽表=>ads=>30 张 =>86 张
 每天 0: 30 开始运行。=> sqoop 40-50 分钟: 1 点 20: => 5-6 个小时运行完指标
 所有离线数据报表控制在 8 小时之内
 大数据实时处理部分控制在 5 分钟之内。(分钟级别、秒级别)
 如果是实时推荐系统, 需要秒级响应

3.11 活动的话, 数据量会增加多少? 怎么解决?

日活增加 50%, GMV 增加多少。(留转、复活)情人节, 促销手纸。
 集群资源都留有预量。11.11, 6.18, 数据量过大, 提前动态增加服务器。

3.12 并发峰值多少? 大概哪个时间点?

高峰期晚上 7-12 点。Kafka 里面 20m/s 2 万/s 并发峰值在 1-2 万人

3.13 数仓中使用的哪种文件存储格式

常用的包括: textFile, rcFile, ORC, Parquet, 一般企业里使用 ORC 或者 Parquet, 因为是列式存储, 且压缩比非常高, 所以相比于 textFile, 查询速度快, 占用硬盘空间少

3.14 哪张表最费时间, 有没有优化

用户行为宽表, 数据量过大。数据倾斜的相关优化手段。(hadoop、hive、spark)

3.15 用什么工具做权限管理

Ranger 或 Sentry (用户认证 kerberos (张三、李四、王五) =>表级别权限 (张三、李四)、字段级别权限 (李四))

3.16 数仓当中数据多久删除一次

- 1) 部分公司永久不删
- 2) 有一年、两年“删除”一次的，这里面说的删除是，先将超时数据压缩下载到单独安装的磁盘上。然后删除集群上数据。 很少有公司不备份数据，直接删除的。

第 4 章 生产经验--测试上线相关

4.1 测试相关

4.1.1 公司有多少台测试服务器？

测试服务器一般三台

4.1.2 测试环境什么样？

有钱的公司和生产环境电脑配置一样。

一般公司测试环境的配置是生产的一半

4.1.3 测试数据哪来的？

一部分自己写 Java 程序自己造（更灵活），一部分从生产环境上取一部分（更真实）。

4.1.4 如何保证写的 sql 正确性

需要造一些特定的测试数据，测试。

从生产环境抓取一部分数据，数据有多少你是知道的，运算完毕应该符合你的预期。

离线数据和实时数据分析的结果比较。（日活 1 万 实时 10100），倾向取离线。

先在 **mysql** 的业务库里面把结果计算出来；在给你在 **ads** 层计算的结果进行比较；

4.1.5 测试之后如何上线？

大公司：上线的时候，将脚本打包，提交 **git**。先发邮件抄送经理和总监，运维。运维负责上线。

小公司：跟项目经理说一下，项目经理技术把关，项目经理通过了就可以上线了。风险意识。

4.2 项目实际工作流程

以下是活跃用户需求的整体开发流程。

产品经理负责收集需求：需求来源与客户反馈、老板的意见

第 1 步：确定指标的业务口径

由产品经理主导，找到提出该指标的运营负责人沟通。首先要问清楚**指标是怎么定义的**，比如活跃用户是指启动过 APP 的用户。设备 id 还是用户 id。

邮件/需求文档->不要口头

第 2 步：需求评审

由产品经理主导设计原型，对于活跃主题，我们最终要展示的是**最近 n 天的活跃用户数变化趋势**，效果如下图所示。此处大数据开发工程师、后端开发工程师、前端开发工程师一同参与，一起说明整个功能的价值和详细的操作流程，确保大家理解的一致。



第 3 步：大数据开发

大数据开发工程师，通过数据同步的工具如 **Flume**、**Sqoop** 等将数据同步到 **ODS** 层，然后就是一层一层的通过 **SQL** 计算到 **DWD**、**DWS** 层，最后形成可为应用直接服务的数据填充到 **ADS** 层。

第4步：后端开发

后端工程师负责，为大数据工程师提供业务数据接口；
同时还负责读取 ADS 层分析后，写入 MySQL 中的数据。

第5步：前端开发

前端工程师负责，前端埋点。
对分析后的结果数据进行可视化展示。

第6步：联调

此时大数据开发工程师、前端开发工程师、后端开发工程师都要参与进来。此时会要求大数据开发工程师基于历史的数据执行计算任务，大数据开发工程师承担数据准确性的校验。前后端解决用户操作的相关 BUG 保证不出现低级的问题完成自测。

第7步：测试

测试工程师对整个大数据系统进行测试。测试的手段包括，边界值、等价类等。

提交测试异常的软件有：禅道、bugzilla（测试人员记录测试问题 1.0，输入是什么，结果是什么，跟预期不一样->需要开发人员解释，是一个 bug，下一个版本解决 1.1->测试人员再测试。测试 1.1ok->测试经理关闭 bug）

1 周开发写代码 => 2 周测试时间

第8步：上线

运维工程师会配合我们的前后端开发工程师更新最新的版本到服务器。此时产品经理要找到该指标的负责人长期跟进指标的准确性。重要的指标还要每过一个周期内部再次验证，从而保证数据的准确性。

4.3 项目中实现一个需求大概多长时间

刚入职第一个需求大概需要 7 天左右。

对业务熟悉后，平均一天一个需求。

影响时间的因素：测试服务器购买获取环境准备、对业务熟悉、开会讨论需求、表的权限申请、测试等。新员工培训（公司规章制度、代码规范）

4.4 项目在 3 年内迭代次数，每一个项目具体是如何迭代的。公司版本迭代多久一次，

迭代到哪个版本

瀑布式开发、敏捷开发

差不多一个月会迭代一次。每月都有节日（元旦、春节、情人节、3.8 妇女节、端午节、618、国庆、中秋、1111/6.1/5.1、生日、周末）新产品、新区域

就产品或我们提出优化需求，然后评估时间。每周我们都会开会做下周计划和本周总结。（日报、周报、月报、季度报、年报）需求 1 周的时间，周三一定完成。周四周五（帮同事写代码、自己学习工作额外的技术）

有时候也会去预研一些新技术。Flink hudi

5.1.2

5 是大版本号：必须是重大升级

1：一般是核心模块变动

2：一般版本变化

4.5 项目开发中每天做什么事

1) 新需求（活动、优化、新产品、新市场）。 60%

2) 故障分析：数仓的任何步骤出现问题，需要查看问题，比如日活，月活下降或快速上升等。20%

3) 新技术的预研（比如 flink、数仓建模、数据质量、元数据管理）10%

4) 其临时任务 10%

5) 晨会-> 10 点做操-> 讨论中午吃什么-> 12 点出去吃 1 点-> 睡到 2 点-> 3 点茶歇水果-> 晚上吃啥-> 吃加班餐-> 开会-> 晚上 6 点吃饭-> 7 点开始干活-10 点-> 11 点

4.6 实时项目数据计算

4.6.1 跑实时任务，怎么分配内存和 CPU 资源

128m 数据对应 1g 内存

1 个 Kafka 分区对应 1 个 CPU

4.6.2 跑实时任务，每天数据量多少？

用户行为：实时任务用到了用户行为多少张表（20g） 100g/5 张表

业务数据：实时任务用到了业务数据多少张表(34m) 1g/30 张表

活动、风控、销售、流量

第 5 章 生产经验—技术

5.1 可视化报表工具

Echarts（百度开源）、kibana（开源）、Tableau（功能强大的收费软件）、Superset（功能一般免费）、QuickBI（阿里云收费的离线）、DataV（阿里云收费的实时）

5.2 集群监控工具

Zabbix+ Grafana

5.3 项目中遇到的问题怎么解决的（重点*****）

Shell 中 flume 停止脚本

Hadoop 宕机

Hadoop 解决数据倾斜方法

集群资源分配参数（项目中遇到的问题）

HDFS 小文件处理

Hadoop 优化

Flume 挂掉

Flume 优化

Kafka 挂掉

Kafka 丢失

Kafka 数据重复

Kafka 消息数据积压

Kafka 优化

Kafka 单条日志传输大小

自定义 UDF、UDTF 函数

Hive 优化

Hive 解决数据倾斜方法

7 天内连续 3 次活跃

分摊

Sqoop 空值、一致性、数据倾斜

Azkaban 任务挂了怎么办？

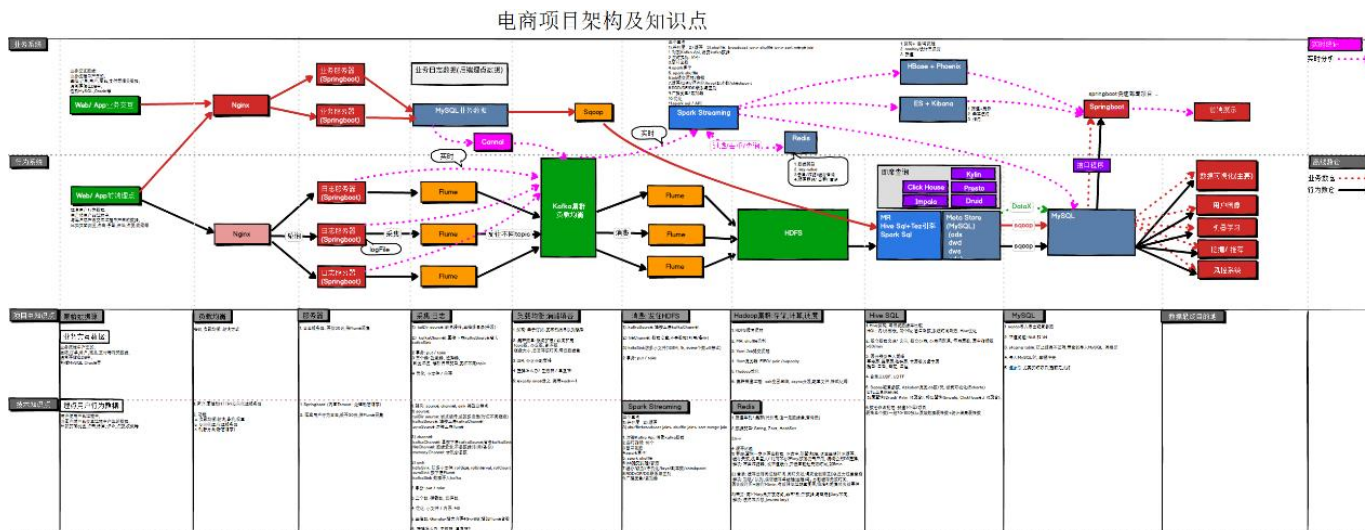
Azkaban 故障报警

Spark 数据倾斜

Spark 优化

SparkStreaming 精确一次性消费

5.4 Linux+Shell+Hadoop+ZK+Flume+kafka+Hive+Sqoop+Azkaban 那些事



第 6 章 生产经验—热点问题

6.1 元数据管理 (Atlas 血缘系统)

依赖关系能够做到：表级别和字段级别

用处：作业执行失败，评估他的影响范围。 主要用于表比较多的公司。

版本问题：

0.84 版本：2019-06-21

2.0 版本：2019-05-13

框架版本：

Apache 0.84 2.0

CDH 2.0

6.2 数据质量监控 (Griffin)

6.2.1 监控原则

1) 单表数据量监控

一张表的记录数在一个已知的范围内，或者上下浮动不会超过某个阈值

- SQL 结果：var 数据量 = select count (*) from 表 where 时间等过滤条件
- 报警触发条件设置：如果数据量不在[数值下限, 数值上限], 则触发报警
- 同比增加：如果((本周的数据量 - 上周的数据量)/上周的数据量*100)不在 [比例下线, 比例上限], 则触发报警
- 环比增加：如果((今天的数据量 - 昨天的数据量)/昨天的数据量*100)不在 [比例下线, 比例上限], 则触发报警
- 报警触发条件设置一定要有。如果没有配置的阈值，不能做监控

日活、周活、月活、留存（日/周/月）、转化率（日、周、月）GMV（日、周、月）
复购率（日/周/月） 30%

2) 单表空值检测

某个字段为空的记录数在一个范围内，或者占总量的百分比在某个阈值范围内

- 目标字段：选择要监控的字段，不能选“无”
- SQL 结果：var 异常数据量 = select count(*) from 表 where 目标字段 is null
- 单次检测：如果(异常数据量)不在[数值下限, 数值上限]，则触发报警

3) 单表重复值检测

一个或多个字段是否满足某些规则

- 目标字段：第一步先正常统计条数；select count(*) from 表；
- 第二步，去重统计；select count(*) from 表 group by 某个字段
- 第一步的值和第二步不的值做减法，看是否在上下线阈值之内
- 单次检测：如果(异常数据量)不在[数值下限, 数值上限]，则触发报警

4) 单表值域检测

一个或多个字段没有重复记录

- 目标字段：选择要监控的字段，支持多选
- 检测规则：填写“目标字段”要满足的条件。其中\$1 表示第一个目标字段，\$2 表示第二个目标字段，以此类推。上图中的“检测规则”经过渲染后变为“delivery_fee = delivery_fee_base+delivery_fee_extra”
- 阈值配置与“空值检测”相同

5) 跨表数据量对比

主要针对同步流程，监控两张表的数据量是否一致

- SQL 结果：count(本表) - count(关联表)
- 阈值配置与“空值检测”相同

6.2.2 数据质量实现

附：质量管理

6.3 权限管理（Ranger）

附 2：权限管理

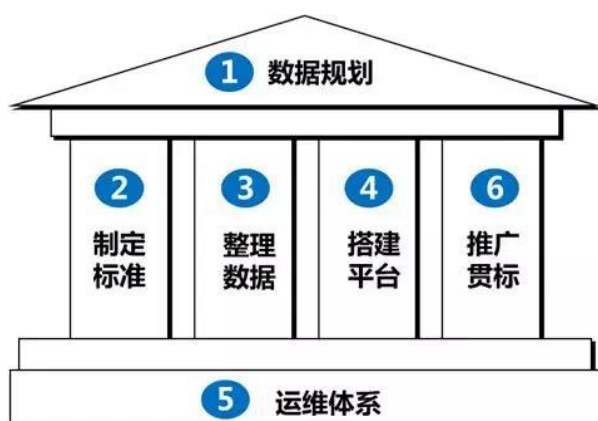
6.4 数据治理

包括：数据质量管理、元数据管理、权限管理（ranger sentry）。

CDH cloudmanager->sentry； HDP ambari=>ranger

数据治理是一个复杂的系统工程，涉及到企业和单位多个领域，既要做好顶层设计，又要解决好统一标准、统一流程、统一管理体系等问题，同时也要解决好数据采集、数据清洗、数据对接和应用集成等相关问题。

数据治理实施要点主要包含数据规划、制定数据标准、整理数据、搭建数据管理工具、构建运维体系及推广贯标六大部分，其中数据规划是纲领、制定数据标准是基础、整理数据是过程、搭建数据管理工具是技术手段、构建运维体系是前提，推广贯标是持续保障。



6.5 数据中台

<https://mp.weixin.qq.com/s/nXI0nSSOnetelCIA7dming>

6.5.1 什么是中台？

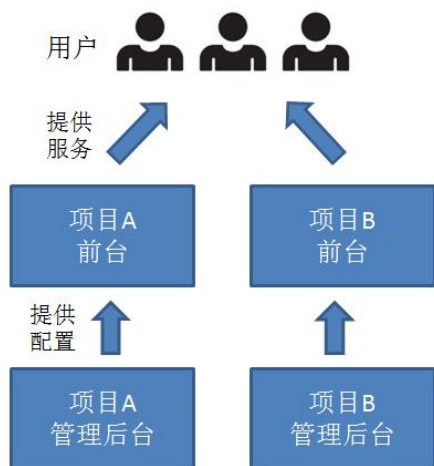
在传统 IT 企业，项目的物理结构是什么样的呢？无论项目内部的如何复杂，都可分为“前台”和“后台”这两部分。

什么是前台？

首先，这里所说的“前台”和“前端”并不是一回事。所谓前台即包括各种和用户直接交互的界面，比如 web 页面，手机 app；也包括服务端各种实时响应用户请求的业务逻辑，比如商品查询、订单系统等等。

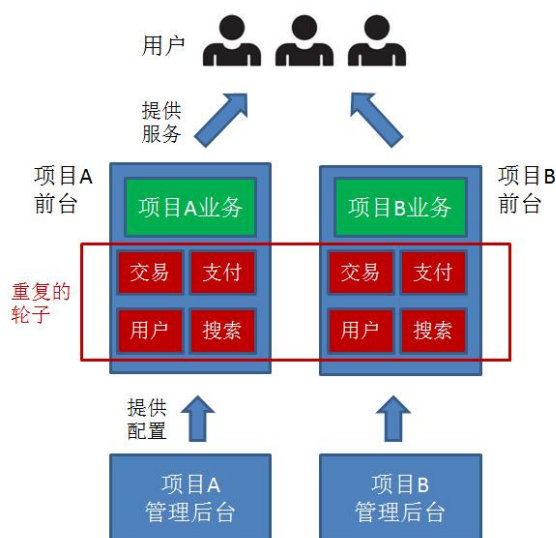
什么是后台？

后台并不直接面向用户，而是面向运营人员的配置管理系统，比如商品管理、物流管理、结算管理。后台为前台提供了一些简单的配置。



6.5.2 传统项目痛点

痛点：重复造轮子。



6.5.3 各家中台

1) SuperCell 公司



2) 阿里巴巴提出了“大中台，小前台”的战略

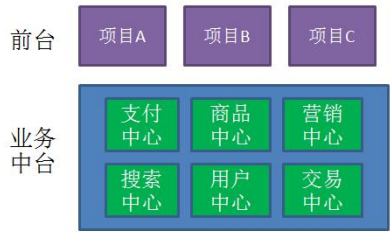


3) 华为提出了“平台炮火支撑精兵作战”的战略



6.5.4 中台具体划分

1) 业务中台



2) 技术中台



3) 数据中台



4) 算法中台



6.5.5 中台使用场景

1) 从 0 到 1 的阶段，没有必要搭建中台。

从 0 到 1 的创业型公司，首要目的是生存下去，以最快的速度打造出产品，证明自身的市场价值。

这个时候，让项目野蛮生长才是最好的选择。如果不慌不忙地先去搭建中台，恐怕中台还没搭建好，公司早就饿死了。

2) 从 1 到 N 的阶段，适合搭建中台。

当企业有了一定规模，产品得到了市场的认可，这时候公司的首要目的不再是活下去，而是活的更好。

这个时候，趁着项目复杂度还不是特别高，可以考虑把各项目的通用部分下沉，组建中台，以方便后续新项目的尝试和旧项目的迭代。

3) 从 N 到 N+1 的阶段，搭建中台势在必行。

当企业已经有了很大的规模，各种产品、服务、部门错综复杂，这时候做架构调整会比较痛苦。

但是长痛不如短痛，为了项目的长期发展，还是需要尽早调整架构，实现平台化，以免日后越来越难以维护。

6.6 数据湖

数据湖（Data Lake）是一个存储企业的各种各样原始数据的大型仓库，其中的数据可供存取、处理、分析及传输。 **hudi**

目前，Hadoop 是最常用的部署数据湖的技术，所以很多人会觉得数据湖就是 Hadoop 集群。数据湖是一个概念，而 Hadoop 是用于实现这个概念的技术。



数据仓库	数据湖
主要处理历史的、结构化的数据，而且这些数据必须与数据仓库事先定义的模式吻合。	能处理所有类型的数据，如结构化数据，非结构化数据，半结构化数据等，数据的类型依赖于数据源系统的原始数据格式。非结构化数据（语音、图片、视频等）
数据仓库分析的指标都是产品经理提前规定好的。按需分析数据。（日活、新增、留存、转化率）	根据海量的数据，挖掘出规律，反应给运营部门。 拥有非常强的计算能力用于处理数据。数据挖掘

6.7 埋点

免费的埋点：上课演示。

收费的埋点：神策 <https://mp.weixin.qq.com/s/Xp3-alWF4XHvKDP9rNWCoQ>

目前主流的埋点方式，有代码埋点（前端/后端）、可视化埋点、全埋点三种。

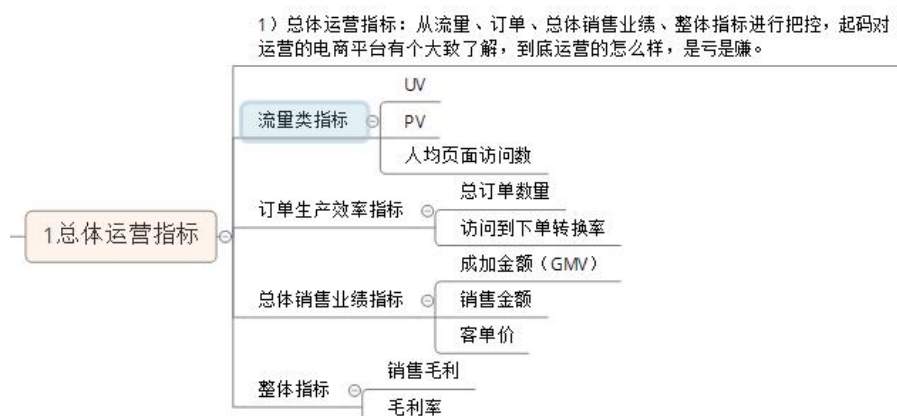
代码埋点是通过调用埋点 SDK 函数，在需要埋点的业务逻辑功能位置调用接口，上报埋点数据。例如，我们对页面中的某个按钮埋点后，当这个按钮被点击时，可以在这个按钮对应的 OnClick 函数里面调用 SDK 提供的数据发送接口，来发送数据。

可视化埋点只需要研发人员集成采集 SDK，不需要写埋点代码，业务人员就可以通过访问分析平台的“圈选”功能，来“圈”出需要对用户行为进行捕捉的控件，并对该事件进行命名。圈选完毕后，这些配置会同步到各个用户的终端上，由采集 SDK 按照圈选的配置自动进行用户行为数据的采集和发送。

全埋点是通过在产品中嵌入 SDK，前端自动采集页面上的全部用户行为事件，上报埋点数据，相当于做了一个统一的埋点。然后再通过界面配置哪些数据需要在系统里面进行分析。

6.8 电商运营经验

6.8.1 电商 8 类基本指标



总体运营指标：从流量、订单、总体销售业绩、整体指标进行把控，起码对运营的电商平台有个大致了解，到底运营的怎么样，是亏是赚。

流量类指标：UV、PV、人均页面访问数

订单生产效率指标：总订单数量、访问到下单转化率

总体销售业绩指标：GMV、销售金额、客单价

整体指标：销售毛利、毛利率

2) 站流量指标：即对访问你网站的访客进行分析，基于这些数据可以对网页进行改进，以及对访客的行为进行分析等等。



3) 销售转化指标：分析从下单到支付整个过程的数据，帮助你提升商品转化率。也可以对一些频繁异常的数据展开分析。



4) 客户价值指标：这里主要就是分析客户的价值，可以建立RFM价值模型，找出那些有价值的客户，精准营销等等。



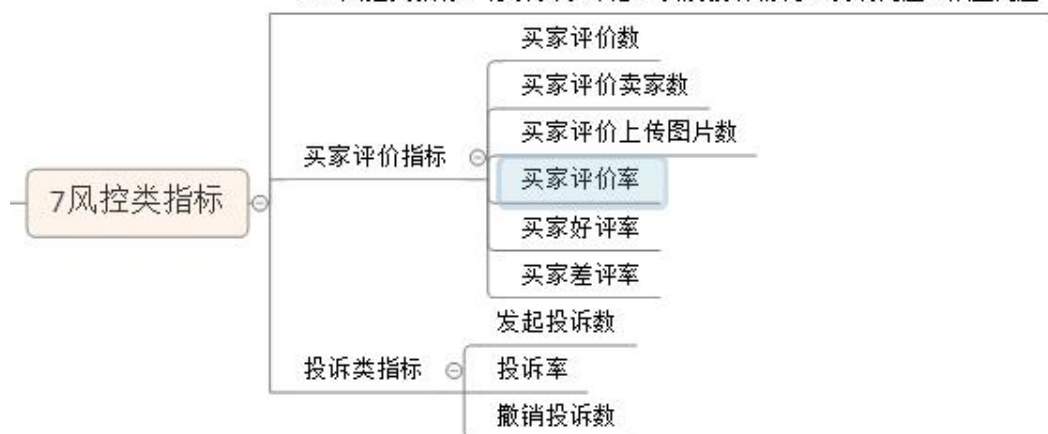
5) 商品类指标：主要分析商品的种类，那些商品卖得好，库存情况，以及可以建立关联模型，分析那些商品同时销售的几率比较高，而进行捆绑销售，有点像啤酒喝尿布的故事。



6) 市场营销活动指标，主要监控某次活动给电商网站带来的效果，以及监控广告的投放指标。



7) 风控类指标：分析卖家评论，以及投诉情况，发现问题，改正问题

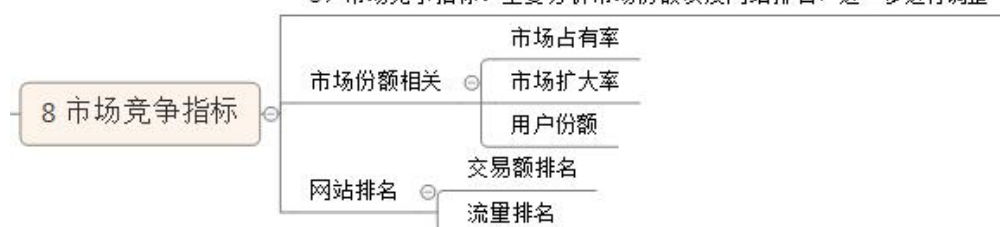


8) 市场竞争指标：主要分析市场份额以及网站排名，进一步进行调整

市场份额相关：市场占有率、市场扩大率、用户份额

网站排名：交易额排名、流量排名

8) 市场竞争指标：主要分析市场份额以及网站排名，进一步进行调整



6.8.2 直播指标

首页：

新增设备	
新增用户	
日活	登陆
	游客
设备启动次数	
平均使用时长	
平台实时在线人数	
平台实时开播数量	
平台充值金额（近 3 日）	
支出	用户提现成功+主播提现成功
	腾讯云支出
主播总工资	主播直播工资
	主播绩效工资
净收益	

用户：

新增设备	
平均使用时长	
设备启动次数	
用户总数	
新增用户	

用户活跃	新用户
	老用户
	登陆
用户留存	新增用户留存
	活跃用户留存
用户登陆	PC 登陆用户数量
	安卓登陆用户数量
	苹果登陆用户数量
用户等级	当天各个等级得用户数量
用户忠诚	
用户流式	
用户回流	
用户设备型号	

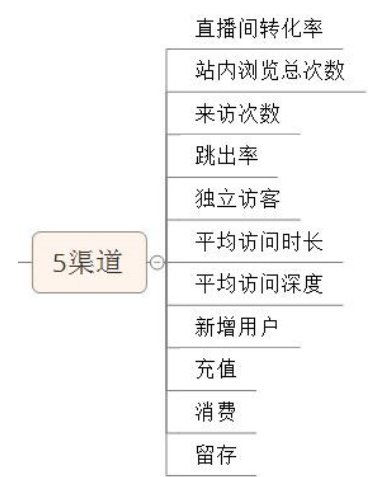
礼物:

各项礼物送出数量	
礼物平台销售额	普通礼物
	转盘礼物
	幸运礼物
	道剧礼物
礼物平台消耗额	销售额-道剧礼物（送该礼物，用户领取得米币）
	消耗额-幸运礼物（送该礼物，用户有几率获得米币）
各项礼物平台收益额	
各项礼物送出占比	
礼物返现	礼物送出总价值
	用户返现
免费礼物统计	免费礼物总人数
	免费礼物总数

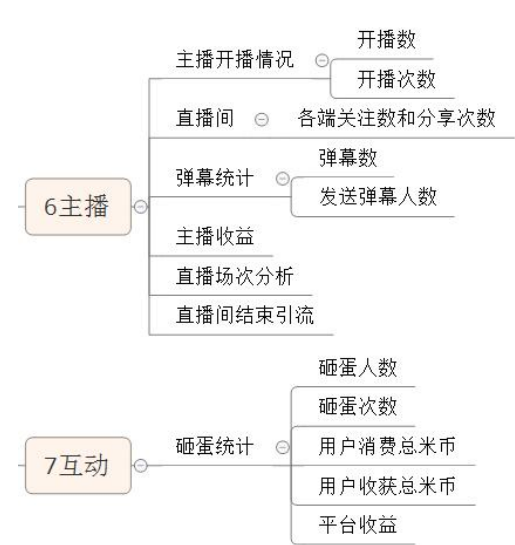
资金:

充值统计	充值金额
	充值笔数
	充值人数
	人均充值
	新增充值
	充值总金额
消费分析	礼物消费额
	礼物收获额
	礼物消费人数
	砸蛋消费额
	砸蛋收入额
	砸蛋参与人数
平台余币	
平台白名单	
用户提现和红包	用户提现金额
	领取红包金额

	用户总提现金额
支出额	
直播工资支出额	
绩效工资支出额	
主播工资总支出	
阿里云支出	



渠道：



附 1：质量管理

第 1 章 数据质量

1.1 概述

数据质量的高低代表了该数据满足数据消费者期望的程度，这种程度基于他们对数据的使用预期，只有达到数据的使用预期才能给予管理层正确的决策参考。数据质量管理作为数据仓库的一个重要模块，主要可以分为数据的健康标准化、监控和保障。

1.2. 数据质量标准分类

- ① 数据完整性: 数据不存在大量的缺失值、不缺少某一日期/部门/地点等部分维度的数据，同时在 ETL 过程当中应保证数据的完整不丢失。验证数据时总数应符合正常规律时间推移，记录数总数的增长符合正常的趋势。
- ② 数据一致性: 数仓各层的数据，应与上一层保持数据一致，最终经过数据清洗转化（ETL）的宽表/指标能和数据源保持一致。

1.3 数据质量管理解决方案

本文通过 Shell 命令和 Hive 脚本的方式，通过验证增量数据的记录数、全表空值记录数、全表记录数是否在合理的范围之内，以及验证数据来源表和目标表一致性，确定当日的数据是否符合健康标准，达到数据质量的监控与管理。

第 2 章 ODS 层数据校验

2.1 数据校验通用脚本

通过 shell 脚本调用 hive，检验当日分区增加的记录数量和全表记录数量是否在合理的范围之内，同时检验关键字段为空的记录的记录数量。

- 1) 创建数据检查脚本文件夹，用于存放数据校验 shell 脚本

```
[atguigu@hadoop102 module]$ mkdir -p data_check/sh
[atguigu@hadoop102 sh]$ pwd
/opt/module/data_check/sh
```

- 2) 在 Hive 中创建表数据质量校验记录表，记录数据校验的各个指标：

```
[atguigu@atguigu data_check]$ hive
```

创建数据库，用于存放数据质量校验的结果数据：

```
hive (default)> create database datacheck;
```

创建数据表，用于存放 ods 层的数据检验结果：

```
hive (datacheck)> create table datacheck.table_count_add_standard(
    data_date string comment '数据时间分区 dt',
    database_name string comment '库名',
    table_name string comment '表名',
    table_type string comment '表类型（全量/增量）',
    add_count bigint comment '当日增量数据的记录数',
    null_count bigint comment '表空值记录数',
    total_count bigint comment '全表记录数'
);
hive (datacheck)>quit;
```

3) 在路径/opt/module/data_check/sh 下创建数据检验增量表通用 shell 脚本

```
[atguigu@hadoop102 sh]$ vim increment_data_check_public.sh
```

在脚本中编写如下内容：

```
#!/bin/bash
# 增量数据所在的日期分区
do_date=$1
# 校验数据的表名
table_name=$2
# 需要校验空值的列名，以逗号','隔开
null_column=$3
# 初始化 SQL 查询语句
null_where_sql_str=""
# 将空值检验字符串切成列名数组
array=(${null_column//,/ })
# 遍历数组，拼接空值查询条件
for(( i=0;i<${#array[@]};i++)) do
    if [ ${i} -eq 0 ];then
        null_where_sql_str=" where ${array[i]} is null "
    else
        null_where_sql_str="$null_where_sql_str or ${array[i]} is null "
    fi
done;
# 执行当日增量数据记录数量 SQL 查询语句
add_count_query_result=`hive -e "select count(*) from gmall.${table_name} where dt='${do_date}'`
# 取出当日增量数据记录数量
add_count=${add_count_query_result:3}
# 执行当日全表数据记录数量 SQL 查询语句
total_count_query_result=`hive -e "select count(*) from gmall.${table_name}"`
# 取出当日全量数据记录数量
total_count=${total_count_query_result:3}
# 执行全表空值数据记录数量 SQL 查询语句
table_null_query_result=`hive -e "select count(*) from gmall.${table_name} $null_where_sql_str"`
# 取出全表空值数据记录数量
null_count=${table_null_query_result:3}
# 将所有数据检验结果插入到表中
hive -e "insert into datacheck.table_count_add_standard values('${do_date}','gmall','${table_name}','increment_table',${add_count},${null_count},${total_count})"
```

脚本参数注释：

第一个参数：传入时间分区参数(dt)

第二个参数：需要进行数据校验的表名(table_name)

第三个参数：需要判断是否为空值的字段名称用逗号'，'，'隔开，例如： col1,col2,col3

给脚本/opt/module/data_check/sh/increment_data_check_public.sh 赋权限：

```
[atguigu@hadoop102 sh]$ chmod 777 increment_data_check_public.sh
```

脚本执行示例：

```
[atguigu@hadoop102 sh]$ ./increment_data_check_public.sh 2020-06-14 ods_activity_rule id,activity_id
```

4) 在路径/opt/module/data_check/sh 下创建数据检验全量表通用 shell 脚本

```
[atguigu@hadoop102 sh]$ vim total_data_check_public.sh
```

在脚本中编写如下内容：

```
#!/bin/bash
# 增量数据所在的日期分区
do_date=$1
# 校验数据的表名
table_name=$2
```

```

# 需要校验空值的列名，以逗号',' 隔开
null_column=$3
# 将空值检验字符串切成列名数组
null_where_sql_str="
# 遍历数组，拼接空值查询条件
array=(${null_column//,/ })
# 遍历数组，拼接空值查询条件
for(( i=0;i<${#array[@]};i++) do
    if [ $i -eq 0 ];then
        null_where_sql_str=" where ${array[i]} is null "
    else
        null_where_sql_str="$null_where_sql_str or ${array[i]} is null "
    fi
done;
# 执行当日全表数据记录数量 SQL 查询语句 table_count_query_result=`hive -e "select count(*) from gmall.$table_name"`
# 取出当日全量数据记录数量
table_count=${table_count_query_result:3}
# 执行全表空值数据记录数量 SQL 查询语句
table_null_query_result=`hive -e "select count(*) from gmall.$table_name $null_where_sql_str"`
# 取出全表空值数据记录数量
null_count=${table_null_query_result:3}
# 将所有数据检验结果插入到表中
hive -e "insert into datacheck.table_count_add_standard values('${do_date}','gmall','$table_name','total_table',null,$null_count,'$table_count')"
```

脚本参数注释：

第一个参数：传入数据校验日期(dt)

第二个参数：需要进行数据校验的表名(table_name)

第三个参数：需要判断是否为空值的字段名称用逗号',' 隔开，例如： col1,col2,col3

给脚本/opt/module/data_check/sh/total_data_check_public.sh 赋权限：

```
[atguigu@hadoop102 sh]$ chmod 777 total_data_check_public.sh
```

脚本执行示例：

```
[atguigu@hadoop102 sh]$ ./total_data_check_public.sh 2020-06-14 ods_activity_rule id,activity_id
```

2.2 ODS 层各表检验

1. 涉及表

增量检查

- (1) 订单详情表 (ods_order_detail)
- (2) 用户表 (ods_user_info)
- (3) 支付流水表 (ods_payment_info)
- (4) 订单状态表 (ods_order_status_log)
- (5) 商品评论表 (ods_comment_info)
- (6) 退单表 (ods_order_refund_info)
- (7) 活动订单关联表 (ods_activity_order)

全量检查

- (1) 订单表 (ods_order_info)
- (2) SKU 商品表 (ods_sku_info)
- (3) 商品一级分类表 (ods_base_category1)
- (4) 商品二级分类表 (ods_base_category2)
- (5) 商品三级分类表 (ods_base_category3)
- (6) 品牌表 (ods_base_trademark)
- (7) SPU 商品表 (ods_spu_info)
- (8) 加购表 (ods_cart_info)

- (9) 商品收藏表 (ods_favor_info)
- (10) 优惠券领用表 (ods_coupon_use)
- (11) 优惠券表 (ods_coupon_info)
- (12) 活动表 (ods_activity_info)
- (13) 优惠规则表 (ods_activity_rule)
- (14) 编码字典表 (ods_base_dic)

2. ODS 层数据检查脚本

1) 在路径/opt/module/data_check/sh 下创建 ODS 层数据检查脚本

```
[atguigu@atguigu sh]$ pwd
/opt/module/data_check/sh
[atguigu@atguigu sh]$ vim ods_data_check.sh
```

在脚本中编写如下内容：

```
#!/bin/bash
data_date=$1

# 增量检查
# 订单详情表
/opt/module/data_check/sh/increment_data_check_public.sh $data_date ods_order_detail id,order_id,user_id,sku_id,sku_name,order_price,sku_num,create_time
# 用户表
/opt/module/data_check/sh/increment_data_check_public.sh $data_date ods_user_info id,name,birthday,gender,email,user_level,create_time,operate_time
# 支付流水表
/opt/module/data_check/sh/increment_data_check_public.sh $data_date ods_payment_info id,out_trade_no,order_id,user_id,alipay_trade_no,total_amount,subject,payment_type,payment_time
# 订单状态表
/opt/module/data_check/sh/increment_data_check_public.sh $data_date ods_order_status_log id,order_id,order_status,operate_time
# 商品评论表
/opt/module/data_check/sh/increment_data_check_public.sh $data_date ods_comment_info id,user_id,sku_id,spu_id,order_id,appraise,create_time
# 退单表
/opt/module/data_check/sh/increment_data_check_public.sh $data_date ods_order_refund_info id,user_id,order_id,sku_id,refund_type,refund_num,refund_amount,refund_reason_type,create_time
# 活动订单关联表
/opt/module/data_check/sh/increment_data_check_public.sh $data_date ods_activity_order id,activity_id,order_id,create_time
# 全量检查
# 订单表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_order_info id,final_total_amount,order_status,user_id,out_trade_no,create_time,operate_time,province_id,benefit_reduce_amount,original_total_amount,feight_fee
# SKU 商品表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_sku_info id,spu_id,price,sku_name,sku_desc,weight,tm_id,category3_id,create_time
# 商品一级分类表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_base_category1 id,name
# 商品二级分类表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_base_category2 id,name,category1_id
# 商品三级分类表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_base_category3 id,name,category2_id
# 品牌表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_base_trademark tm_id,tm_name
# SPU 商品表
```

```

/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_spu_info id,sku_name,category3_id,tm_id
# 加购表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_cart_info id,user_id,sku_id,cart_price,sku_num,sku_name,create_time,operate_time,is_ordered,order_time
# 商品收藏表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_favor_info id,user_id,sku_id,spu_id,is_cancel,create_time,cancel_time
# 优惠券领用表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_coupon_use id,coupon_id,user_id,order_id,coupon_status,get_time,using_time,used_time
# 优惠券表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_coupon_info id,coupon_name,coupon_type,condition_amount,condition_num,activity_id,benefit_amount,benefit_discount,create_time,range_type,spu_id,tm_id,category3_id,limit_num,operate_time,expire_time
# 活动表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_activity_info id,activity_name,activity_type,start_time,end_time,create_time
# 优惠规则表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_activity_rule id,activity_id,condition_amount,condition_num,benefit_amount,benefit_discount,benefit_level
# 编码字典表
/opt/module/data_check/sh/total_data_check_public.sh $data_date ods_base_dic dic_code,dic_name,parent_code,create_time,operate_time
2) 给脚本/opt/module/data_check/sh/ods_data_check.sh 赋权限:
[atguigu@atguigu sh]$ chmod 777 ods_data_check.sh

```

第3章 DWD 层数据校验

3.1 数据校验通用脚本

1) 创建数据表，用于存放 dwd 层的数据检验结果：

```

hive (datacheck)> create table datacheck.dwd_table_data_check(
    data_date string comment '数据时间分区 dt',
    database_name string comment '库名',
    source_table_name string comment '数据源表表名',
    source_column string comment '数据源表字段名',
    target_table_name string comment '数据目标表表名',
    target_column string comment '数据目标表字段名',
    consistent_data_count bigint comment '全表数据一致记录数',
    source_table_count bigint comment '数据源表全表记录数',
    target_table_count bigint comment '数据目标表全表记录数'
);
hive (datacheck)>quit;

```

2) 在路径/opt/module/data_check/sh 下创建 dwd 层数据一致性检验通用 shell 脚本

```
[atguigu@hadoop102 sh]$ vim table_consistent_check_public.sh
```

在脚本中编写如下内容：

```

#!/bin/bash
# 增量数据所在的日期分区
do_date=$1
# 校验数据源表的表名
source_table_name=$2
# 检验数据源表的字段（与目标表顺序一致才能对比两个字段）
source_column=$3
# 检验数据目标表的表名
target_table_name=$4
# 检验数据目标表的字段（与源表顺序一致才能对比两个字段）

```

```

target_column=$5

# 初始化 SQL 查询语句
join_on_sql_str=""
# 将检验数据源表的字段切成列名数组
source_column_array=(${source_column//,/ })
# 将检验数据目标表的字段切成列名数组
target_column_array=(${target_column//,/ })

# 遍历数组，拼接表关联条件，输入字段全部关联
for(( i=0;i<${#source_column_array[@]};i++)) do
    if [ $i -eq 0 ];then
        join_on_sql_str=" on $source_table_name.${source_column_array[i]}=$target_table_name.${target_column_array[i]} "
    else
        join_on_sql_str="$join_on_sql_str and $source_table_name.${source_column_array[i]}=$target_table_name.${target_column_array[i]} "
    fi
done;

echo "-----ods-dwd 一致性检查-----"
# 执行数据源表和目标表关联查询 SQL 语句，查询数据一致的条数
consistent_data_query_result=`hive -e "select count(*) from gmall.$source_table_name join gmall.$target_table_name $join_on_sql_str"`
# 取出全表查询数据一致的条数
consistent_data_count=${consistent_data_query_result:3}

echo "-----ods 层记录条数-----"
# 执行查询数据源表的记录条数
source_table_query_result=`hive -e "select count(*) from gmall.$source_table_name"`
# 取出全表数据源表的记录条数
source_table_count=${source_table_query_result:3}

echo "-----dwd 层记录条数-----"
# 执行查询数据目标表的记录条数
target_table_query_result=`hive -e "select count(*) from gmall.$target_table_name"`
# 取出全表数据目标表的记录条数
target_table_count=${target_table_query_result:3}

# 将所有数据检验结果插入到表中
hive -e "insert into datacheck.dwd_table_data_check values('$do_date','gmall','$source_table_name','$source_column','$target_table_name','$target_column','$consistent_data_count','$source_table_count','$target_table_count')"
脚本参数注释：
第 1 个参数：传入时间分区参数( dt )
第 2 个参数：需要进行数据校验源表的表名(source_table_name )
第 3 个参数：需要校验的数据源表字段名称用逗号'，'，'隔开，例如： col1,col2,col3
第 4 个参数：需要进行数据校验目标表的表名(target_table_name )
第 5 个参数：需要校验的数据目标表字段名称用逗号'，'，'隔开，例如： col1,col2,col3
给脚本/opt/module/data_check/sh/increment_data_check_public.sh 赋权限：
[atguigu@hadoop102 sh]$ chmod 777 table_consistent_check_public.sh
脚本执行示例：
[atguigu@hadoop102 sh]$ ./table_consistent_check_public.sh 2020-06-14 ods_base_province name dwd_dim_base_province province_name

```

3.2 DWD 层各表检验

1. 业务数据表数据检验

① 优惠券信息表

数据源表: ods_coupon_info

源表字段: id,coupon_name,coupon_type,condition_amount,condition_num,activity_id,benefit_amount,benefit_discount,create_time,range_type,spu_id,tm_id,category3_id,limit_num,operate_time,expire_time

数据目标表: dwd_dim_coupon_info

目标表字段: id,coupon_name,coupon_type,condition_amount,condition_num,activity_id,benefit_amount,benefit_discount,create_time,range_type,spu_id,tm_id,category3_id,limit_num,operate_time,expire_time

分区: dt='2020-06-14'

数据检查脚本:

```
[atguigu@hadoop102 sh]$ pwd
```

```
/opt/module/data_check/sh
```

```
[atguigu@hadoop102 sh]$ /opt/module/data_check/sh/table_consistent_check_public.sh 2020-06-14 ods_coupon_info id,coupon_name,coupon_type,condition_amount,condition_num,activity_id,benefit_amount,benefit_discount,create_time,range_type,spu_id,tm_id,category3_id,limit_num,operate_time,expire_time dwd_dim_coupon_info id,coupon_name,coupon_type,condition_amount,condition_num,activity_id,benefit_amount,benefit_discount,create_time,range_type,spu_id,tm_id,category3_id,limit_num,operate_time,expire_time
```

② 订单明细事实表

数据源表: ods_order_detail

源表字段: id,order_id,user_id,sku_id,sku_name,order_price,sku_num,create_time

数据目标表: dwd_fact_order_detail

目标表字段: id,order_id,user_id,sku_id,sku_name,order_price,sku_num,create_time

分区: dt='2020-06-14'

数据检查脚本:

```
[atguigu@hadoop102 sh]$ pwd
```

```
/opt/module/data_check/sh
```

```
[atguigu@hadoop102 sh]$ /opt/module/data_check/sh/table_consistent_check_public.sh 2020-06-14 ods_order_detail id,order_id,user_id,sku_id,sku_name,order_price,sku_num,create_time dwd_fact_order_detail id,order_id,user_id,sku_id,sku_name,order_price,sku_num,create_time
```

③ 支付事实表

数据源表: ods_payment_info

源表字段: id,out_trade_no,order_id,user_id,alipay_trade_no,total_amount,subject,payment_type,payment_time

数据目标表: dwd_fact_payment_info

目标表字段: id,out_trade_no,order_id,user_id,alipay_trade_no,payment_amount,subject,payment_type,payment_time

分区: dt='2020-06-14'

数据检查脚本:

```
[atguigu@hadoop102 sh]$ pwd
```

```
/opt/module/data_check/sh
```

```
[atguigu@hadoop102 sh]$ /opt/module/data_check/sh/table_consistent_check_public.sh 2020-06-14 ods_payment_info id,out_trade_no,order_id,user_id,alipay_trade_no,total_amount,subject,payment_type,payment_time dwd_fact_payment_info id,out_trade_no,order_id,user_id,alipay_trade_no,payment_amount,subject,payment_type,payment_time
```

④ 退款事实表

数据源表: ods_order_refund_info

源表字段: id,user_id,order_id,sku_id,refund_type,refund_num,refund_amount,refund_reason_type,create_time

数据目标表: dwd_fact_order_refund_info

目标表字段: id,user_id,order_id,sku_id,refund_type,refund_num,refund_amount,refund_reason_type,create_time

分区: dt='2020-06-14'

数据检查脚本:

```
[atguigu@hadoop102 sh]$ pwd
```

```
/opt/module/data_check/sh
```

```
[atguigu@hadoop102 sh]$ /opt/module/data_check/sh/table_consistent_check_public.sh 2020-06-14 ods_order_refund_info id,user_id,order_id,sku_id,refund_type,refund_num,refund_amount,refund_reason_type,create_time dwd_fact_order_refund_info id,user_id,order_id,sku_id,refund_type,refund_num,refund_amount,refund_reason_type,create_time
```

⑤ 评价事实表

数据源表: ods_comment_info

源表字段: id,user_id,sku_id,spu_id,order_id,appraise,create_time

数据目标表: dwd_fact_comment_info

目标表字段: id,user_id,sku_id,spu_id,order_id,appraise,create_time

分区: dt='2020-06-14'

数据检查脚本:

```
[atguigu@hadoop102 sh]$ pwd
```

```
/opt/module/data_check/sh
```

```
[atguigu@hadoop102 sh]$ /opt/module/data_check/sh/table_consistent_check_public.sh 2020-06-14 ods_comment_info id,user_id,sku_id,spu_id,order_id,appraise,create_time dwd_fact_comment_info id,user_id,sku_id,spu_id,order_id,appraise,create_time
```

⑥ 加购事实表

数据源表: ods_cart_info

源表字段: id,user_id,sku_id,card_price,sku_num,sku_name,create_time,operate_time,is_ordered,order_time

数据目标表: dwd_fact_cart_info

目标表字段: id,user_id,sku_id,card_price,sku_num,sku_name,create_time,operate_time,is_ordered,order_time

分区: dt='2020-06-14'

数据检查脚本:

```
[atguigu@hadoop102 sh]$ pwd
```

```
/opt/module/data_check/sh
```

```
[atguigu@hadoop102 sh]$ /opt/module/data_check/sh/table_consistent_check_public.sh 2020-06-14 ods_cart_info id,user_id,sku_id,card_price,sku_num,sku_name,create_time,operate_time,is_ordered,order_time dwd_fact_cart_info id,user_id,sku_id,card_price,sku_num,sku_name,create_time,operate_time,is_ordered,order_time
```

⑦ 收藏事实表

数据源表: ods_favor_info

源表字段: id,user_id,sku_id,spu_id,is_cancel,create_time,create_time

数据目标表: dwd_fact_favor_info

目标表字段: id,user_id,sku_id,spu_id,is_cancel,create_time,create_time

分区: dt='2020-06-14'

数据检查脚本:

```
[atguigu@hadoop102 sh]$ pwd
```

```
/opt/module/data_check/sh
```

```
[atguigu@hadoop102 sh]$ /opt/module/data_check/sh/table_consistent_check_public.sh 2020-06-14 ods_favor_info id,user_id,sku_id,spu_id,is_cancel,create_time,create_time dwd_fact_favor_info id,user_id,sku_id,spu_id,is_cancel,create_time,create_time
```

2. DWD 层数据检查脚本

1) 在路径/opt/module/data_check/sh 下创建 dwd 层数据一致性检验 shell 脚本

```
[atguigu@hadoop102 sh]$ vim dwd_data_check.sh
```

2) 在脚本中编写如下内容:

```
#!/bin/bash
```

```
# 数据所在的日期分区
```

```
do_date=$1
```

```
/opt/module/data_check/sh/table_consistent_check_public.sh $do_date ods_coupon_info id,coupon_name,coupon_type,condition_amount,condition_num,activity_id,benefit_amount,benefit_discount,create_time,range_type,spu_id,tm_id,category3_id,limit_num,operate_time,expire_time dwd_dim_coupon_info id,coupon_name,coupon_type,condition_amount,condition_num,activity_id,benefit_amount,benefit_discount,create_time,range_type,spu_id,tm_id,category3_id,limit_num,operate_time,expire_time
```

```
/opt/module/data_check/sh/table_consistent_check_public.sh $do_date ods_order_detail id,order_id,user_id,sku_id,sku_name,order_price,sku_num,create_time dwd_fact_order_detail id,order_id,user_id,sku_id,sku_name,order_price,sku_num,create_time
```

```
/opt/module/data_check/sh/table_consistent_check_public.sh $do_date ods_payment_info id,out_trade_no,order
```

```

_id,user_id,alipay_trade_no,total_amount,subject,payment_type,payment_time dwd_fact_payment_info id,out_trade_no,order_id,user_id,alipay_trade_no,payment_amount,subject,payment_type,payment_time

/opt/module/data_check/sh/table_consistent_check_public.sh $do_date ods_order_refund_info id,user_id,order_id,sku_id,refund_type,refund_num,refund_amount,refund_reason_type,create_time dwd_fact_order_refund_info id,user_id,order_id,sku_id,refund_type,refund_num,refund_amount,refund_reason_type,create_time

/opt/module/data_check/sh/table_consistent_check_public.sh $do_date ods_comment_info id,user_id,sku_id,spu_id,order_id,appraise,create_time dwd_fact_comment_info id,user_id,sku_id,spu_id,order_id,appraise,create_time

/opt/module/data_check/sh/table_consistent_check_public.sh $do_date ods_cart_info id,user_id,sku_id,cart_price,sku_num,sku_name,create_time,operate_time,is_ordered,order_time dwd_fact_cart_info id,user_id,sku_id,cart_price,sku_num,sku_name,create_time,operate_time,is_ordered,order_time

/opt/module/data_check/sh/table_consistent_check_public.sh $do_date ods_favor_info id,user_id,sku_id,spu_id,is_cancel,create_time,cancel_time dwd_fact_favor_info id,user_id,sku_id,spu_id,is_cancel,create_time,cancel_time
脚本参数注释：
第 1 个参数：传入时间分区参数( dt )
3) 给脚本/opt/module/data_check/sh/dwd_data_check.sh 赋权限：
[atguigu@hadoop102 sh]$ chmod 777 dwd_data_check.sh
脚本执行示例：
[atguigu@hadoop102 sh]$ ./dwd_data_check.sh 2020-06-15

```

第 4 章 DWS 层数据校验

4.1 DWS 层数据质量校验

由于 DWS 层数据质量无法从一致性进行判断，只能通过表记录数以及空值记录数等维度判断，所以在 DWS 将检验当日宽表记录数量是否在合理的范围之内，同时检验关键字段为空的记录的记录数量。

4.2 DWS 层数据校验脚本

1. 数据校验涉及表

- 每日会员行为
- 每日商品行为
- 每日活动行为
- 每日地区行为

2. 数据质量校验脚本

1) 在路径/opt/module/data_check/sh 下创建 ODS 层数据检查脚本

```

[atguigu@atguigu sh]$ pwd
/opt/module/data_check/sh
[atguigu@atguigu sh]$ vim dws_data_check.sh

```

在脚本中编写如下内容：

```

#!/bin/bash
data_date=$1

# 每日会员行为
/opt/module/data_check/sh/total_data_check_public.sh $data_date dws_user_action_daycount user_id,login_count,cart_count,order_count

# 每日商品行为
/opt/module/data_check/sh/total_data_check_public.sh $data_date dws_sku_action_daycount sku_id,order_count,order_num,order_amount,payment_count,payment_num,payment_amount,refund_count

# 每日活动行为
/opt/module/data_check/sh/total_data_check_public.sh $data_date dws_activity_info_daycount id,activity_name,activity_type,start_time,end_time,create_time,display_count

```

```
# 每日地区统计
/opt/module/data_check/sh/total_data_check_public.sh $data_date dws_area_stats_daycount id,province_name,area_code,iso_code,region_id,region_name,login_count,order_count,order_amount,payment_count,payment_amount

2) 给脚本/opt/module/data_check/sh/dws_data_check.sh 赋权限:
[atguigu@atguigu sh]$ chmod 777 dws_data_check.sh

3) 执行/opt/module/data_check/sh/dws_data_check.sh, 检验 2020-06-14 当日宽表数据质量。
[atguigu@atguigu sh]$ ./dws_data_check.sh 2020-16-14
```

第 5 章 DWT 层数据校验

5.1 DWT 层数据质量校验方法

在宽表阶段数据已经经过了一定的判断、过滤和变换等操作，因此在 DWT 层也将检验当日宽表记录数量是否在合理的范围之内，同时检验关键字段为空的记录的记录数量。

5.2 宽表校验脚本

1. 数据校验涉及表

- 设备主题宽表
- 会员主题宽表
- 商品主题宽表
- 活动主题宽表
- 地区主题宽表

2. 数据质量校验脚本

1) 在路径/opt/module/data_check/sh 下创建 ODS 层数据检查脚本

```
[atguigu@atguigu sh]$ pwd
/opt/module/data_check/sh
[atguigu@atguigu sh]$ vim dwt_data_check.sh
```

在脚本中编写如下内容：

```
#!/bin/bash
data_date=$1

# 全量检查
# 设备主题宽表
/opt/module/data_check/sh/total_data_check_public.sh $data_date dwt_uv_topic mid_id,brand,model,login_date_first,login_date_last,login_day_count,login_count

# 会员主题宽表
/opt/module/data_check/sh/total_data_check_public.sh $data_date dwt_user_topic user_id,login_date_first,login_date_last,login_count,login_last_30d_count,order_date_first,order_date_last,order_count,order_amount,order_last_30d_count,order_last_30d_amount,payment_date_first,payment_date_last,payment_count,payment_amount,payment_last_30d_count,payment_last_30d_amount

# 商品主题宽表
/opt/module/data_check/sh/total_data_check_public.sh $data_date dwt_sku_topic sku_id,spu_id,order_last_30d_count,order_last_30d_num,order_last_30d_amount,order_count,order_num,order_amount

# 活动主题宽表
/opt/module/data_check/sh/total_data_check_public.sh $data_date dwt_activity_topic id,activity_name,activity_type,start_time,end_time,create_time,display_day_count,order_day_count,order_day_amount,payment_day_count,payment_day_amount,display_count,order_count,order_amount,payment_count,payment_amount

# 地区主题宽表
/opt/module/data_check/sh/total_data_check_public.sh $data_date id,province_name,area_code,iso_code,region_id,region_name,login_day_count,login_last_30d_count,order_day_count,order_day_amount,order_last_30d_count,o
```

```
rder_last_30d_amount,payment_day_count,payment_day_amount,payment_last_30d_count,payment_last_30d_ament
```

2) 给脚本/opt/module/data_check/sh/dwt_data_check.sh 赋权限:

```
[atguigu@atguigu sh]$ chmod 777 dwt_data_check.sh
```

3) 执行/opt/module/data_check/sh/dwt_data_check.sh, 检验 2020-06-15 当日宽表数据质量。

```
[atguigu@atguigu sh]$ ./dwt_data_check.sh 2020-16-15
```

第 6 章 ADS 层数据校验

数据仓库中 ADS 层数据是经过高度聚合计算的具体指标, 因此无法从技术层面进行判断数据是否健康。需要通过校对对各个指标的数值是否在合理的范围之内进行校验, 进行定制化数据校验。由于 ADS 层涉及的需求无法一一涉及, 因此在这针对一个需求进行分析。

1) 在 Hive 中创建表数据质量校验记录表, 记录数据校验的各个指标:

```
[atguigu@atguigu data_check]$ hive
```

创建数据库, 用于存放数据质量校验的结果数据:

```
hive (default)> create database datacheck;
```

创建数据表, 用于存放 ods 层的数据检验结果:

```
hive (datacheck)> create table datacheck.ads_table_data_check(
  data_date string comment '数据时间分区 dt',
  database_name string comment '库名',
  table_name string comment '表名',
  column_name string comment '指标名',
  healthy_value string comment '该指标合理值',
  now_value bigint comment '该指标当前值',
  is_healthy bigint comment '该指标是否合理: 1 合理/0 不合理'
```

```
);
```

```
hive (datacheck)>quit;
```

3) 在路径/opt/module/data_check/sh 下创建数据检验增量表通用 shell 脚本

```
[atguigu@hadoop102 sh]$ vim ads_data_check.sh
```

在脚本中编写如下内容:

```
#!/bin/bash
```

```
# 增量数据所在的日期分区
```

```
do_date=$1
```

```
hive -e "insert into datacheck.ads_table_data_check select
  temp.data_date,
  temp.database_name,
  temp.table_name,
  temp.column_name,
  temp.healthy_value,
  temp.new_mid_count,
  temp.is_healthy
```

```
from (
```

```
  select
```

```
    "$do_date" as data_date,
    "gmall" as database_name,
    "ads_new_mid_count" as table_name,
    "new_mid_count" as column_name,
    "大于 3" as healthy_value,
    new_mid_count,
    if(new_mid_count>3,1,0) as is_healthy
```

```
  from gmall.ads_new_mid_count
```

```
) as temp
```

```
"
```

脚本参数注释:

第一个参数：传入时间分区参数(dt)

给脚本/opt/module/data_check/sh/ads_data_check.sh 赋权限：

```
[atguigu@hadoop102 sh]$ chmod 777 ads_data_check.sh
```

脚本执行示例：

```
[atguigu@hadoop102 sh]$
```

```
./ads_data_check.sh 2020-06-14
```

第 7 章 数据质量之 Griffin

由于 Griffin 有着较为严重的版本依赖，因此无法在最新版本的数据仓库架构中兼容进去。但若是使用 2.x 版本的 Spark 和 hadoop，可通过如下案例进行 Griffin 的数据质量监控。文档中同时附上了 DWD 层与 ODS 层使用 Griffin 进行一致性数据检验的章节。文档如下：



质量监控之Griffin.
docx

附 2：权限管理监控

第 1 章 Ranger 概述

1.1 什么是 Ranger

Apache Ranger 是一个用来在 Hadoop 平台上进行监控，启用服务，以及全方位数据安全访问管理的安全框架。

Ranger 的愿景是在 Apache Hadoop 生态系统中提供全面的安全管理。随着企业业务的拓展，企业可能在多用户环境中运行多个工作任务，这就要求 Hadoop 内的数据安全性需要扩展为同时支持多种不同的需求进行数据访问，同时还需要提供一个可以对安全策略进行集中管理，配置和监控用户访问的框架。Ranger 由此产生！

Ranger 的官网：<https://ranger.apache.org/>

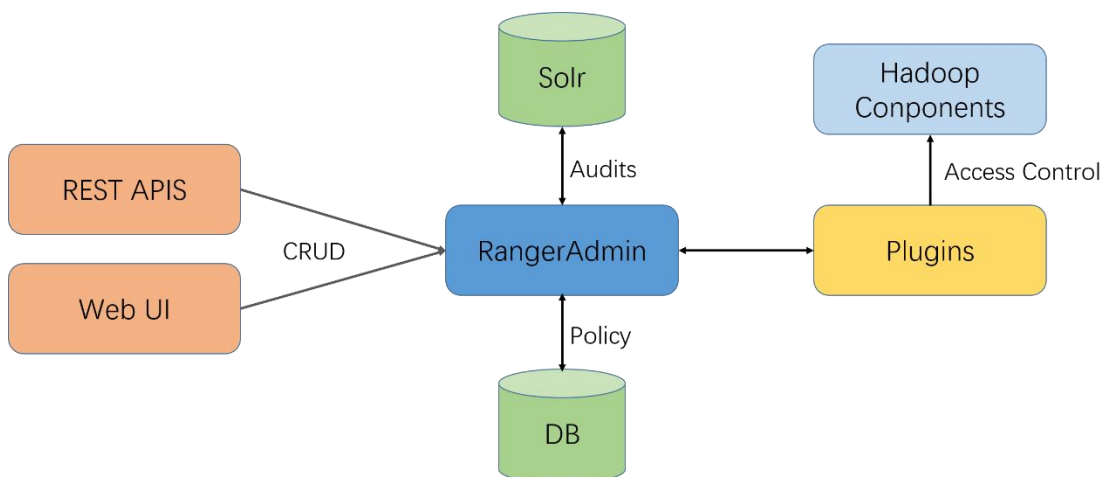
1.2 Ranger 的目标

- 允许用户使用 UI 或 REST API 对所有和安全相关的任务进行集中化的管理
- 允许用户使用一个管理工具对操作 Hadoop 体系中的组件和工具的行为进行细粒度的授权
- 支持 Hadoop 体系中各个组件的授权认证标准
- 增强了对不同业务场景需求的授权方法支持，例如基于角色的授权或基于属性的授权
- 支持对 Hadoop 组件所有涉及安全的审计行为的集中化管理

1.3 Ranger 支持的框架

- Apache Hadoop
- Apache Hive
- Apache HBase
- Apache Storm
- Apache Knox
- Apache Solr
- Apache Kafka
- YARN
- NIFI

1.4 Ranger 的架构



1.5 Ranger 的工作原理

Ranger 的核心是 Web 应用程序，也成为 RangerAdmin 模块，此模块由管理策略，审计日志和报告等三部分组成。

管理员角色的用户可以通过 RangerAdmin 提供的 web 界面或 REST APIS 来定制安全策略。这些策略会由 Ranger 提供的轻量级的针对不同 Hadoop 体系中组件的插件来执行。插件会在 Hadoop 的不同组件的核心进程启动后，启动对应的插件进程来进行安全管理！

第 2 章 Ranger 的安装

2.1 环境准备

Ranger2.0 要求对应的 Hadoop 为 3.x 以上，Hive 为 3.x 以上版本，JDK 为 1.8 以上版本！

2.2 安装 RangerAdmin

2.2.1 数据库环境准备

(1) 登录 MySQL

```
[atguigu@hadoop102 ~]$ mysql -uroot -p000000
```

(2) 在 MySQL 数据库中创建 Ranger 存储数据的数据库

```
mysql> create database ranger;
```

(3) 更改 mysql 密码策略，为了可以采用比较简单的密码

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

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

(4) 创建用户

```
mysql> grant all privileges on ranger.* to ranger@'%' identified by 'ranger';
```

2.2.2 安装 RangerAdmin

(1) 在 hadoop102 的 /opt/module 路径上创建一个 ranger

```
[atguigu@hadoop102 module]$ mkdir ranger
```

(2) 解压软件

```
[atguigu@hadoop102 software]$ tar -zxvf ranger-2.0.0-admin.tar.gz -C /opt/module/ranger
```

(3) 进入 /opt/module/ranger/ranger-2.0.0-admin 路径，对 install.properties 配置

```
[atguigu@hadoop102 ranger-2.0.0-admin]$ vim install.properties
```

修改以下配置内容：

```
#mysql 驱动
```

```
SQL_CONNECTOR_JAR=/opt/software/mysql-connector-java-5.1.48.jar
```

```
#mysql 的主机名和 root 用户的用户名密码
```

```
db_root_user=root
```

```
db_root_password=000000
```

```
db_host=hadoop102
```

```
#ranger 需要的数据库名和用户信息，和 2.2.1 创建的信息要一一对应
db_name=ranger
db_user=ranger
db_password=ranger
#其他 ranger admin 需要的用户密码
rangerAdmin_password=atguigu123
rangerTagsync_password=atguigu123
rangerUsersync_password=atguigu123
keyadmin_password=atguigu123
#ranger 存储审计日志的路径，默认为 solr，这里为了方便暂不设置
audit_store=
#策略管理器的 url,rangeradmin 安装在哪台机器，主机名就为对应的主机名
policymgr_external_url=http://hadoop102:6080
#启动 ranger admin 进程的 linux 用户信息
unix_user=atguigu
unix_user_pwd=atguigu
unix_group=atguigu
#hadoop 的配置文件目录
hadoop_conf=/opt/module/hadoop-3.1.3/etc/hadoop
```

(4) 之后切换到 root 用户，执行安装

```
[root@hadoop102 ranger-2.0.0-admin]# ./setup.sh
```

出现以下信息，说明安装完成

```
2020-04-30 13:58:18,051 [I] Ranger all admins default password change request processed successfully..
Installation of Ranger PolicyManager Web Application is completed.
```

(5) 创建 ranger 的配置文件软连接到 web 应用下

```
[root@hadoop102 ranger-2.0.0-admin]# ./set_globals.sh
```

usermod: 无改变

```
[2020/04/30 13:58:47]: [I] Soft linking /etc/ranger/admin/conf to ews/webapp/WEB-INF/classes/conf
```

2.2.3 启动 RangerAdmin

(1) 配置 RangerAdmin web 应用的配置信息

```
[root@hadoop102 ranger-2.0.0-admin]# cd /etc/ranger/admin/conf/
[root@hadoop102 conf]# vim ranger-admin-site.xml
```

```
<property>
  <name>ranger.jpa.jdbc.password</name>
  <value>ranger</value>
  <description />
</property>

<property>
  <name>ranger.service.host</name>
  <value>hadoop102</value>
</property>
```

(2) 启动 ranger

```
[root@hadoop102 conf]# ranger-admin start
Starting Apache Ranger Admin Service
Apache Ranger Admin Service with pid 7058 has started.
```

ranger-admin 在安装时已经配设置为开机自启动，因此之后无需再手动启动！

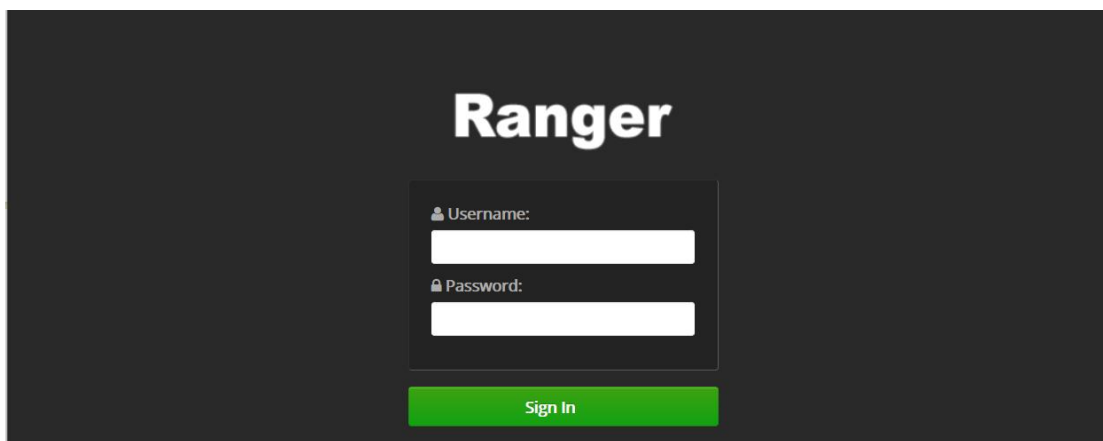
(3) 查看启动后的进程

```
[root@hadoop102 ranger-2.0.0-usersync]# jps
7058 EmbeddedServer
8132 Jps
```

(4) 停止 ranger

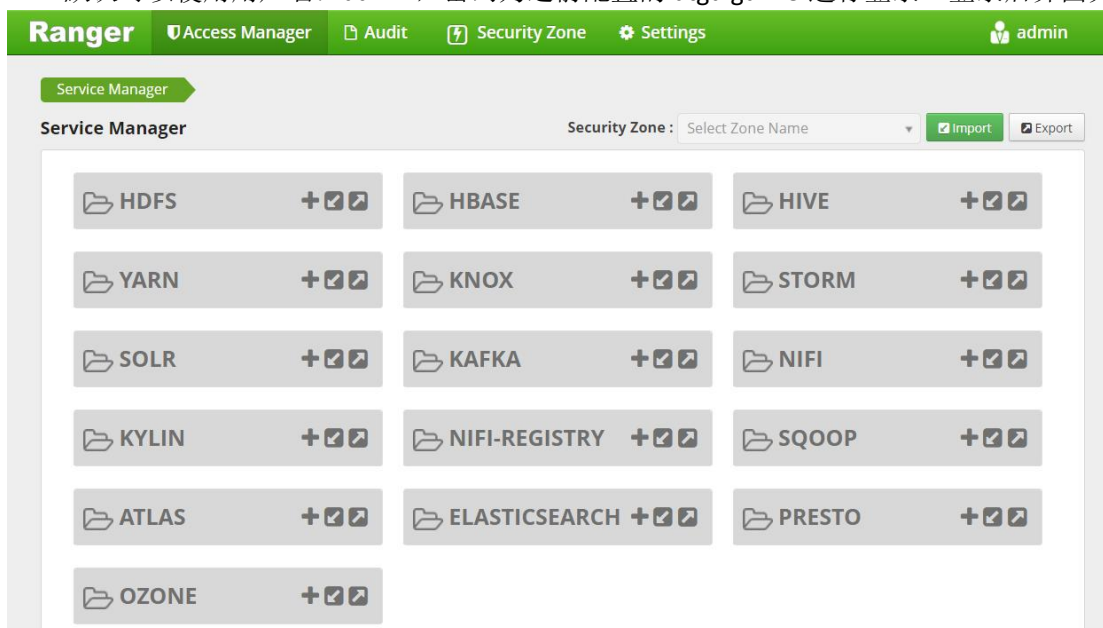
```
[root@hadoop102 conf]# ranger-admin stop
```

之后可以通过访问 <http://hadoop102:6080>，如出现以下界面，说明 ranger-admin 启动完成！



2.2.4 登录管理员用户

默认可以使用用户名：admin，密码为之前配置的 atguigu123 进行登录！登录后界面如下：



第3章 安装 RangerUsersync

3.1 RangerUsersync 简介

RangerUsersync 作为 Ranger 提供的一个管理模块，可以将 Linux 机器上的用户和组信息同步到 RangerAdmin 的数据库中进行管理！

3.2 RangerUsersync 安装

(1) 解压软件

```
[root@hadoop102 software]# tar -zxvf ranger-2.0.0-usersync.tar.gz -C /opt/module/ranger/
```

(2) 配置软件

```
[root@hadoop102 ranger-2.0.0-usersync]# vim install.properties
```

修改以下配置信息

#rangeradmin 的 url

POLICY_MGR_URL =<http://hadoop102:6080>

#同步间隔时间，单位(分钟)

SYNC_INTERVAL = 1

#运行此进程的 linux 用户

unix_user=[atguigu](#)

unix_group=[atguigu](#)

#rangerUserSync 的用户密码，参考 rangeradmin 中 install.properties 的配置

rangerUsersync_password=[atguigu123](#)

#hadoop 的配置文件目录
hadoop_conf=/opt/module/hadoop-3.1.3/etc/hadoop

(3) 使用 root 用户进行安装

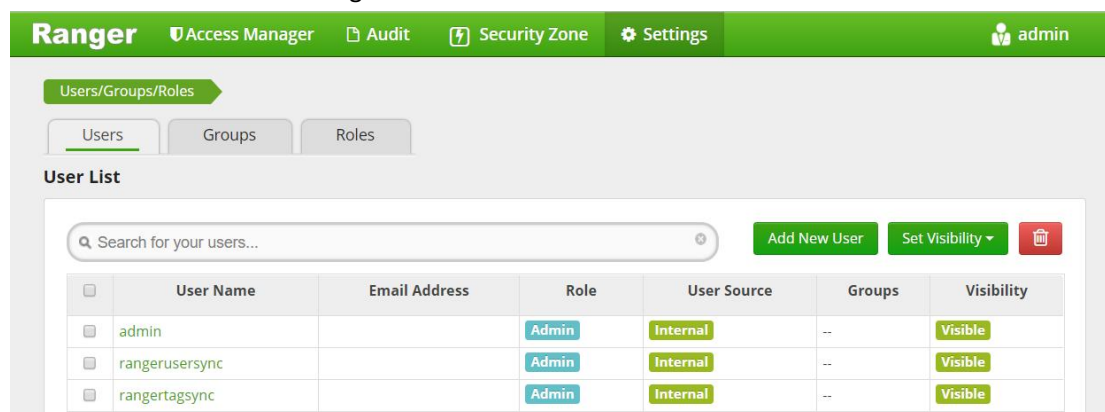
```
[root@hadoop102 ranger-2.0.0-usersync]# ./setup.sh
```

出现以下信息，说明安装完成

```
ranger.usersync.policymgr.password has been successfully created.  
Provider jceks://file/etc/ranger/usersync/conf/rangerusersync.jceks was updated.  
[I] Successfully updated password of rangerusersync user
```

3.3 RangerUsersync 启动

(1) 启动之前，在 ranger admin 的 web-UI 界面，查看用户信息如下：



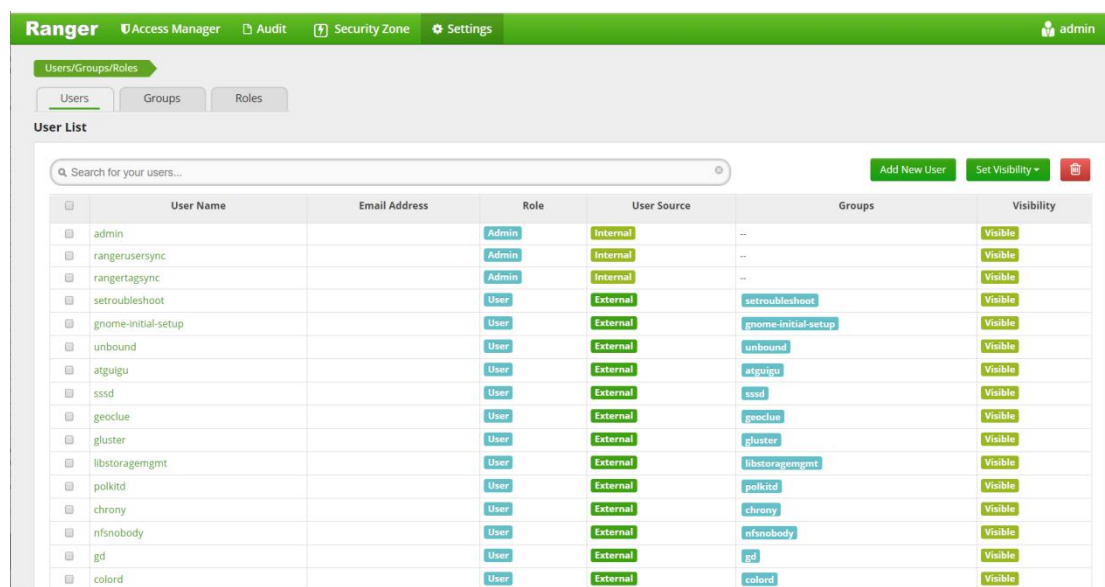
	User Name	Email Address	Role	User Source	Groups	Visibility
<input type="checkbox"/>	admin		Admin	Internal	--	Visible
<input type="checkbox"/>	rangerusersync		Admin	Internal	--	Visible
<input type="checkbox"/>	rangertagsync		Admin	Internal	--	Visible

(2) 使用 root 用户启动

```
[root@hadoop102 ranger-2.0.0-usersync]# ranger-usersync start
```

```
Starting Apache Ranger Usersync Service  
Apache Ranger Usersync Service with pid 7510 has started.
```

(3) 启动后，再次查看用户信息：



	User Name	Email Address	Role	User Source	Groups	Visibility
<input type="checkbox"/>	admin		Admin	Internal	--	Visible
<input type="checkbox"/>	rangerusersync		Admin	Internal	--	Visible
<input type="checkbox"/>	rangertagsync		Admin	Internal	--	Visible
<input type="checkbox"/>	setroubleshoot		User	External	setroubleshoot	Visible
<input type="checkbox"/>	gnome-initial-setup		User	External	gnome-initial-setup	Visible
<input type="checkbox"/>	unbound		User	External	unbound	Visible
<input type="checkbox"/>	atguigu		User	External	atguigu	Visible
<input type="checkbox"/>	sssd		User	External	sssd	Visible
<input type="checkbox"/>	geoclue		User	External	geoclue	Visible
<input type="checkbox"/>	gluster		User	External	gluster	Visible
<input type="checkbox"/>	libstoragemgmt		User	External	libstoragemgmt	Visible
<input type="checkbox"/>	polkitd		User	External	polkitd	Visible
<input type="checkbox"/>	chrony		User	External	chrony	Visible
<input type="checkbox"/>	nfsnobody		User	External	nfsnobody	Visible
<input type="checkbox"/>	gd		User	External	gd	Visible
<input type="checkbox"/>	colord		User	External	colord	Visible

说明 ranger-usersync 工作正常！

ranger-usersync 服务也是开机自启动的，因此之后不需要手动启动！

第 4 章 安装 Ranger Hive-plugin

4.1 Ranger Hive-plugin 简介

Ranger Hive-plugin 是 Ranger 对 hive 进行权限管理的插件。Ranger Hive-plugin 只能对使用 jdbc 方式访问 hive 的请求进行权限管理，hive-cli 并不受限制！

4.2 Ranger Hive-plugin 安装

(1) 解压软件

```
[root@hadoop102 software]# tar -zxvf ranger-2.0.0-hive-plugin.tar.gz -C /opt/module/ranger/
```

(2) 配置软件

```
[root@hadoop102 ranger-2.0.0-hive-plugin]# vim install.properties
```

修改以下内容

#策略管理器的 url 地址

POLICY_MGR_URL=**http://hadoop102:6080**

#组件名称可以自定义

REPOSITORY_NAME=**hivedev**

#hive 的安装目录

COMPONENT_INSTALL_DIR_NAME=**/opt/module/hive**

#hive 组件的启动用户

CUSTOM_USER=**atguigu**

#hive 组件启动用户所属组

CUSTOM_GROUP=**atguigu**

(3) 将 hive 的配置文件作为软连接安装到 Ranger Hive-plugin 目录下

```
[root@hadoop102 ranger-2.0.0-hive-plugin]# ln -s /opt/module/hive/conf/ conf
```

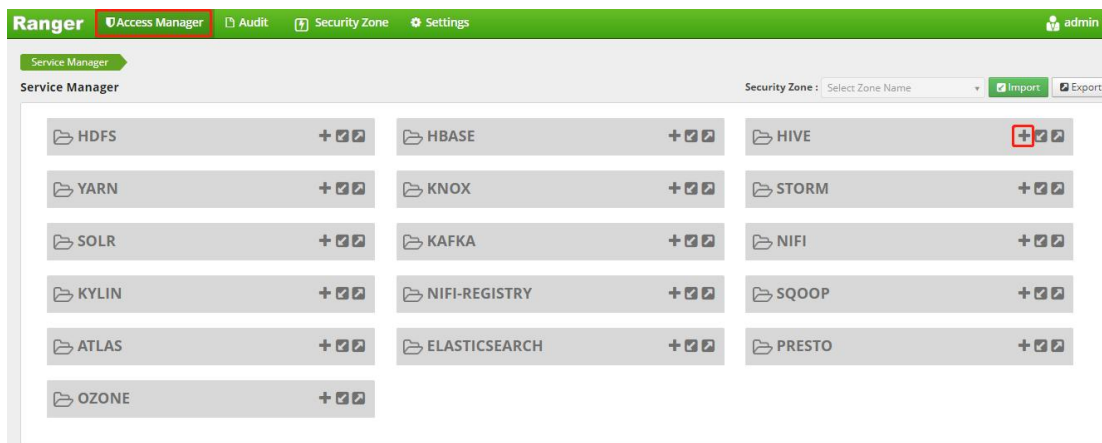
(4) 使用 root 用户启动 Ranger Hive-plugin

```
[root@hadoop102 ranger-2.0.0-hive-plugin]# ./enable-hive-plugin.sh
```

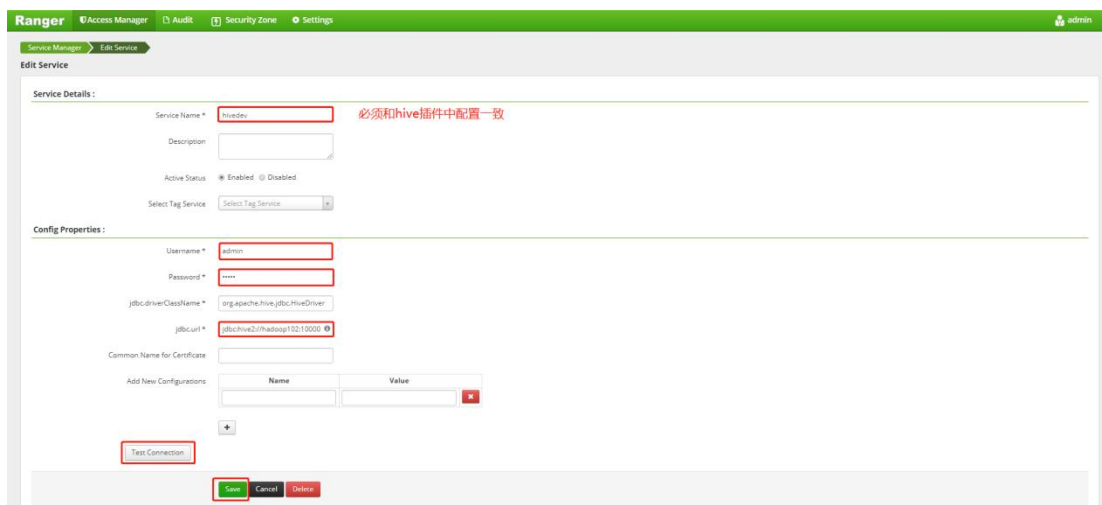
之后需要重启 hive 才能生效！

4.3 在 ranger admin 上配置 hive 插件

(1) Access Manager/hive



(2) 配置服务详情



注意：一定要点击 Save 后再执行后续的测试。

4.4 测试连接 hiveserver2

(1) 启动 hiveserver2

```
[atguigu@hadoop102 hive]$ hive --service metastore &
[atguigu@hadoop102 hive]$ hiveserver2
```

(2) 测试插件是否可以连接 hiveserver2

The screenshot shows the 'Edit Service' form in the Ranger Service Manager. The 'Service Details' section includes fields for 'Service Name' (hivedev), 'Description', 'Active Status' (Enabled), and 'Select Tag Service'. The 'Config Properties' section includes fields for 'Username' (admin), 'Password', 'jdbc.driverClassName' (org.apache.hive.jdbc.HiveDriver), and 'jdbc.url' (jdbc:hive2://hadoop102:10000). There is a 'Test Connection' button at the bottom left of the form.

(3) 出现以下提示说明连接成功！

Connected Successfully.

OK

第5章 使用 Ranger 对 Hive 进行权限管理

5.1 权限控制初体验

(1) 查看默认的访问策略，此时只有 admin 用户拥有对所有库、表和函数的访问权限

The screenshot shows the 'Hivedev Policies' page in the Ranger Service Manager. It displays a table of policies for the 'hivedev' service. The table has columns for Policy ID, Policy Name, Policy Labels, Status, Audit Logging, Roles, Groups, Users, and Action. There are four policies listed, all with a status of 'Enabled' and assigned to the 'admin' user.

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
1	all - hiveservice	--	Enabled	Enabled	--	--	admin	[View] [Edit] [Delete]
2	all - database, table, column	--	Enabled	Enabled	--	--	admin	[View] [Edit] [Delete]
3	all - database, udf	--	Enabled	Enabled	--	--	admin	[View] [Edit] [Delete]
4	all - url	--	Enabled	Enabled	--	--	admin	[View] [Edit] [Delete]

(2) 验证：使用 atguigu 用户尝试进行登录，登录成功后，执行查询语句

```
[atguigu@hadoop102 ~]$ beeline
```

```
beeline> !connect 'jdbc:hive2://hadoop102:10000'
```

```
Connecting to jdbc:hive2://hadoop102:10000
```

```
Enter username for jdbc:hive2://hadoop102:10000: atguigu
```

```
Enter password for jdbc:hive2://hadoop102:10000: *****
```

```
Connected to: Apache Hive (version 3.1.2)
```

```
Driver: Hive JDBC (version 3.1.2)
```

```
Transaction isolation: TRANSACTION_REPEATABLE_READ
```

```
0: jdbc:hive2://hadoop102:10000> show tables;
```

```
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied: user [atguigu] does not have [USE] privilege on [default] (state=42000,code=40000)
```

```
0: jdbc:hive2://hadoop102:10000>
```

(3) 之后使用 admin 用户进行登录，可以完成 Hive 的所有操作。

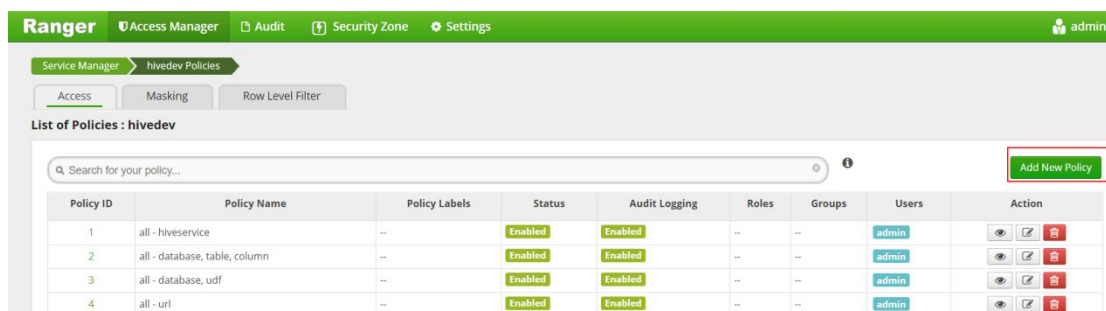

```
[atguigu@hadoop102 ~]$ beeline

beeline> !connect 'jdbc:hive2://hadoop102:10000'
Connecting to jdbc:hive2://hadoop102:10000
Enter username for jdbc:hive2://hadoop102:10000: admin
Enter password for jdbc:hive2://hadoop102:10000: ****
Connected to: Apache Hive (version 3.1.2)
Driver: Hive JDBC (version 3.1.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://hadoop102:10000> show tables;
+-----+
| tab_name |
+-----+
| student |
| student1 |
+-----+
```

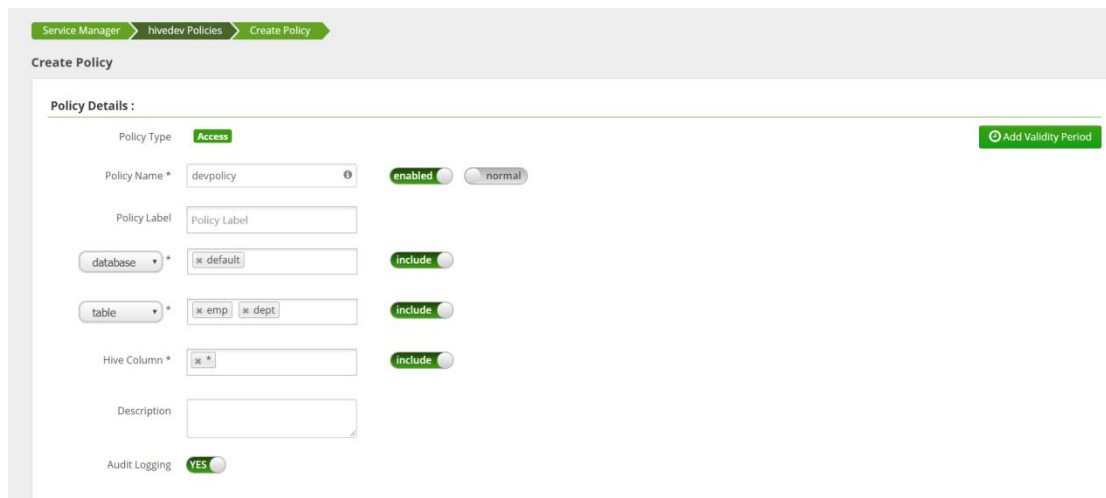
5.2 为用户配置权限

例如为 atguigu 用户配置 default 库 emp 和 dept 表的所有列的读权限，为 jack 用户配置 default 库 emp 和 dept 表的所有列的读写权限。

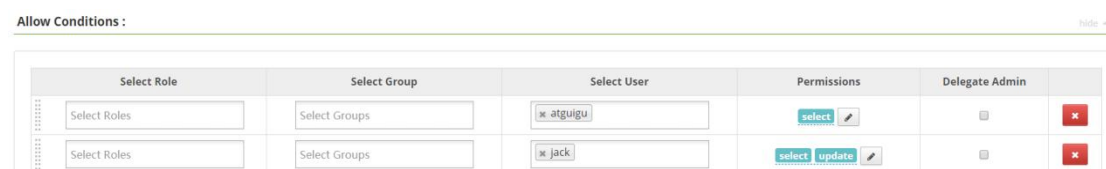
(1) 点击 Add New Policy 按钮



(2) 填写策略名称，以及此策略设计的库、表、列等信息



(3) 填写设计此策略的允许的用户权限



(4) 之后点击 Add 添加按钮，发现在面板上已经添加完成

Service Manager

hivedev Policies

Access

Masking

Row Level Filter

List of Policies : hivedev

Search for your policy...

Add New Policy

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
1	all - hiveservice	--	Enabled	Enabled	--	--	admin	
2	all - database, table, column	--	Enabled	Enabled	--	--	admin	
3	all - database, udf	--	Enabled	Enabled	--	--	admin	
4	all - url	--	Enabled	Enabled	--	--	admin	
5	devpolicy	--	Enabled	Enabled	--	--	atguigu jack	

（5）测试：beeline 无需重新连接 hiveserver2，再次执行查询，发现 atguigu 用户已经可以进行查询，但是只能查询自己有权限查询的表信息

```
0: jdbc:hive2://hadoop103:10000> show tables;
INFO : Compiling command(queryId=atguigu_20200430155544_0af189ed-9eea-420c-83d8-fba3cb173921): show tables
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retrial = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:tab_name, type:string, comment:from deserializer)], properties:null)
INFO : Completed compiling command(queryId=atguigu_20200430155544_0af189ed-9eea-420c-83d8-fba3cb173921); Time taken: 0.018 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=atguigu_20200430155544_0af189ed-9eea-420c-83d8-fba3cb173921): show tables
INFO : Starting task [Stage-0:DDL] in serial mode
INFO : Completed executing command(queryId=atguigu_20200430155544_0af189ed-9eea-420c-83d8-fba3cb173921); Time taken: 0.014 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-----+
| tab_name |
+-----+
| dept     |
| emp      |
+-----+
```

（6）对以下两个表，有读权限，没有写权限

```
0: jdbc:hive2://hadoop103:10000> select * from dept;
INFO : Compiling command(queryId=atguigu_20200430160050_cc76d406-4c82-4fec-8f21-cc61f5366821): select * from dept
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Semantic Analysis Completed (retrial = false)
INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:dept.deptno, type:int, comment:null), FieldSchema(name:dept.dname, type:string, comment:null), FieldSchema(name:dept.loc, type:int, comment:null)], properties:null)
INFO : Completed compiling command(queryId=atguigu_20200430160050_cc76d406-4c82-4fec-8f21-cc61f5366821); Time taken: 0.209 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=atguigu_20200430160050_cc76d406-4c82-4fec-8f21-cc61f5366821): select * from dept
INFO : Completed executing command(queryId=atguigu_20200430160050_cc76d406-4c82-4fec-8f21-cc61f5366821); Time taken: 0.001 seconds
INFO : OK
INFO : Concurrency mode is disabled, not creating a lock manager
+-----+
| dept.deptno | dept.dname | dept.loc |
+-----+
| 10          | ACCOUNTING | 1700     |
| 20          | RESEARCH  | 1800     |
| 30          | SALES      | 1900     |
| 40          | OPERATIONS | 1700     |
+-----+
4 rows selected (0.287 seconds)
0: jdbc:hive2://hadoop103:10000> insert into table dept values(50,'SECURITY',1800);
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied: user [atguigu] does not have [UPDATE] privilege on [default/dept] (state=42000,code=40000)
```

（7）再次测试 jack 用户，尝试向 dept 表写入数据后查询

```
0: jdbc:hive2://hadoop102:10000> insert into table dept values(50,'SECURITY',1800);
0: jdbc:hive2://hadoop102:10000> select * from dept;
```

dept.deptno	dept.dname	dept.loc
50	SECURITY	1800
10	ACCOUNTING	1700
20	RESEARCH	1800
30	SALES	1900
40	OPERATIONS	1700

5.3 脱敏操作

通过脱敏操作可以限制用户对某一列的访问，将敏感数据不暴露给用户！

案例：指定 atguigu 用户在查询 emp 表时，对 hiredate 的年月部分脱敏！

首先需要保证用户对指定的列有访问权限，可以参考 5.2 进行配置！

（1）点击 Masing 标签，再点击 Add New Policy

Service Manager

hivedev Policies

Access

Masking

Row Level Filter

List of Policies : hivedev

Q

Search for your policy...

（2）指定表和列

Please ensure that users/groups listed in this policy have access to the column via an **Access Policy**. This policy does not implicitly grant access to the column.

Policy Details:

Policy Type: **Masking** ➕ Add Validity Period

Policy ID: **8**

Policy Name: empMasks enabled normal

Policy Label:

Hive Database:

Hive Table:

Hive Column:

Description:

Audit Logging: **YES**

(3) 指定用户和脱敏操作

Hive Column:

Description:

Audit Logging: **YES**

Mask Conditions:

Select Role: Select Group: Select User: Access Types:

Select Masking Option:

- ☒ Redact
- ☐ Partial mask: show last 4
- ☐ Partial mask: show first 4
- ☐ Hash
- ☐ Nullify
- ☐ Unmasked (retain original value)
- ☐ Date: show only year
- ☒ Custom

Custom:

(4) 之后点击 save 按钮！那么只有 atguigu 用户在查询时，会触发此策略！

emp.empno	emp.ename	emp.job	emp.mgr	emp.hiredate	emp.sal	emp.comm	emp.deptno
7369	SMITH	CLERK	7902	*****17	800.0	NULL	20
7499	ALLEN	SALESMAN	7698	*****20	1600.0	300.0	30
7521	WARD	SALESMAN	7698	*****22	1250.0	500.0	30
7566	JONES	MANAGER	7839	*****2	2975.0	NULL	20
7654	MARTIN	SALESMAN	7698	*****28	1250.0	1400.0	30
7698	BLAKE	MANAGER	7839	*****1	2850.0	NULL	30
7782	CLARK	MANAGER	7839	*****9	2450.0	NULL	10
7788	SCOTT	ANALYST	7566	*****19	3000.0	NULL	20
7839	KING	PRESIDENT	NULL	*****17	5000.0	NULL	10
7844	TURNER	SALESMAN	7698	*****8	1500.0	0.0	30
7876	ADAMS	CLERK	7788	*****23	1100.0	NULL	20
7900	JAMES	CLERK	7698	*****3	950.0	NULL	30
7902	FORD	ANALYST	7566	*****3	3000.0	NULL	20
7934	MILLER	CLERK	7782	*****23	1300.0	NULL	10

14 rows selected (0.332 seconds)

5.4 行级别过滤

通过行级别过滤可以将表中的数据进行条件过滤后再暴露给用户！

例如：atguigu 用户只允许查询 emp 表中 job 类型为 SALESMAN 的用户信息。

同理，行级别过滤也要求用户对指定表有 access 权限！参考 5.2 的配置！

(1) 选择 Row Level Filter 标签，点击 Add New Policy:

Service Manager > hivedev Policies

Access Masking **Row Level Filter**

List of Policies: hivedev

Search for your policy...

➕ Add New Policy

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
No Policies found!								

(2) 选择对应的库和表:

Service Manager

hivedev/ Policies

Create Policy

Create Policy

Please ensure that users/groups listed in this policy have access to the table via an Access Policy. This policy does not implicitly grant access to the table.

Policy Details :

Policy Type

Row Level Filter

Add Validity Period

Policy Name *

atguigu_salesman

enabled

normal

Policy Label

Policy Label

Hive Database *

default

Hive Table *

emp

Description

Audit Logging

YES

(3) 添加过滤规则和用户

Row Filter Conditions :

Select Role

Select Group

Select User

Access Types

Row Level Filter

Select Roles

Select Groups

atguigu

select

job=SALESMAN

(4) 之后点击 add 按钮！验证。

emp.empno	emp.ename	emp.job	emp.mgr	emp.hiredate	emp.sal	emp.comm	emp.deptno
7499	ALLEN	SALESMAN	7698	*****20	1600.0	300.0	30
7521	WARD	SALESMAN	7698	*****22	1250.0	500.0	30
7654	MARTIN	SALESMAN	7698	*****28	1250.0	1400.0	30
7844	TURNER	SALESMAN	7698	*****8	1500.0	0.0	30

第 6 章 官网其他权限配置

更多配置，可以参考官网介绍：
<https://cwiki.apache.org/confluence/display/RANGER/Row-level+filtering+and+column-masking+using+Apache+Ranger+policies+in+Apache+Hive>