

### 项目简介：

苏宁易购智能营销数据建设项目，营销数据数据源主要来自权益曝光(埋点上报)，权益领取，权益核销(优惠记录)，订单数据，除此之外还有很多配置数据。通过数据集成工具和数据开发过程(ETL)实现数据分层，满足上游业务的各类营销数据应用。

### 个人职责：

1. 熟悉公司营销主题域的数仓业务场景，业务架构与逻辑，熟悉开发环境，开发管理规范；
2. 参与将营销系统、权益投放系统等业务数据库中的数据同步至 Hive 数仓 ODS 层中的工作，完成数据接入；
3. 参与部分营销主题域数仓 DWD 层明细事实表的宽表化处理工作，例如权益领取事实表、权益核销事实表；
4. 参与部分权益数据指标的开发工作，例如ROI(投入产出比)、AOV(平均订单价值)、GMV等指标；
5. 参与部分小文件治理工作，整理小文件过多的表，通过参数配置合并小文件，文件个数减少65%；
6. 参与部分数据质量监控(DQC)工作，通过配置进行表的强规则监控及若干指标的弱规则监控等，保障数据质量；
7. 完成对执行时间异常的数仓任务的排查，问题的定位解决，将原本运行时间为32分钟的任务优化到10分钟以内。

## 00. 自我介绍

---

面试官您好，我是xxxxx

本科毕业于XXXX，大数据专业

硕士就读于xxxxx学，统计专业，目前研二

研一期间在计算机科学Top期刊，专家系统应用，发表了一篇有关数据挖掘和深度学习的论文

研一的暑假在南京的苏宁易购实习了3个月，主要担任大数据开发实习生，进组之后主要做的就是苏宁易购智能营销数据建设项目，营销数据数据源主要来自权益曝光(埋点上报)，权益领取，权益核销(优惠记录)，订单数据，除此之外还有很多配置数据。通过数据集成工具和数据开发过程实现数据分层，满足上游业务的各类营销数据应用；

个人负责的内容主要包括：

1. 负责将业务数据库(包括营销系统和权益投放系统的)中的数据同步至 Hive 数仓 ODS 层中，完成数据接入
2. 负责部分【DWD层明细事实表】的宽表化处理
3. 参与部分【权益数据指标】的开发工作，例如ROI(投入产出比)、AOV(平均订单 价值)、GMV等指标
4. 参与部分小文件治理工作、营销主题域的数据质量监控、包括对一些异常任务的排查和解决

## 01. 实习信息

---

### 1.你们组多少人？公司架构什么样子？

部门很多，除了行政类部门以外，跟我们联系比较紧密的主要分业务类和研发类两种部门。大数据开发是独立的一级研发部门，包括平台开发+数仓开发+应用开发，我是属于数仓开发组下的实习生，整个组十几个人，然后每个人可能负责一条或者多条业务线。我的Mentor负责营销这块。

### 2.进公司干了什么？

第一周没有什么开发的需求，去了之后先领电脑装备，然后申请各种权限，第一周的主要工作就是去熟悉一下公司的业务场景，然后看一下公司的开发管理规范等等。

后来就慢慢的参与一些数据的开发工作了，包括对DWD层明细事实表的宽表化处理，还有部分权益数据指标的开发工作。

### 3.一些开发常识

开发工具：公司自研的一体化数据集成平台，里面内嵌了很多模块，可以实现不同的功能

开发语言：Hive SQL（只会Hive SQL）

开发流程：任务上线 => 新建任务 => 代码编写 => 任务提交 => 导师review => 冒烟测试 => 发布上线

开发环境：dev pro

#### 4.日增数据量是多少？

- 权益埋点事实表：日增量5000W+，6G
- 权益领取表：日增量大概在300W+，压缩后大小为百兆级别
- 权益核销表：日增量大概在100W+，压缩后大小为百兆级别

#### 5.公司大致信息？

- 江苏省南京市玄武区苏宁大道1号
- 在新世界花园租的房子，从岗子村地铁站去做4号线，十几分钟就到了

#### 6.开发流程规范，SOP规范？

- 需求分析和追踪，在jira系统发布和追踪
- 数仓分层架构和模型，数据资产平台管理，文档保存在wiki 代码开发是在 数据探查开发平台 进行
- 任务部署是在 调度系统 操作
- 数据质量管理在 数据质量平台，配置DQC和SLA

#### 7.具体开发流程是怎么做的？

具体开发是优先参考以前方案，因为好的模型或者链路，它是有很强的参考性和复用性。

具体的的操作是 查询以前的类似的报表，找指标口径文档，然后去资产管理平台找到这个报表对应的节点ID，去运维平台看报表整体生产链路，参考开发代码。根据技术方法产出的数据模型进行开发。然后是测试上线，配置DQC和SLA。

#### 8.公司集群规模

- 集群规模：问了运维，百台规模，具体不方便透漏，每台64core + 256G + 12 8T 硬盘
- 任务情况：每天大概有1000多个左右的app在跑

## 1.熟悉业务场景、业务架构、数仓架构、开发管理规范

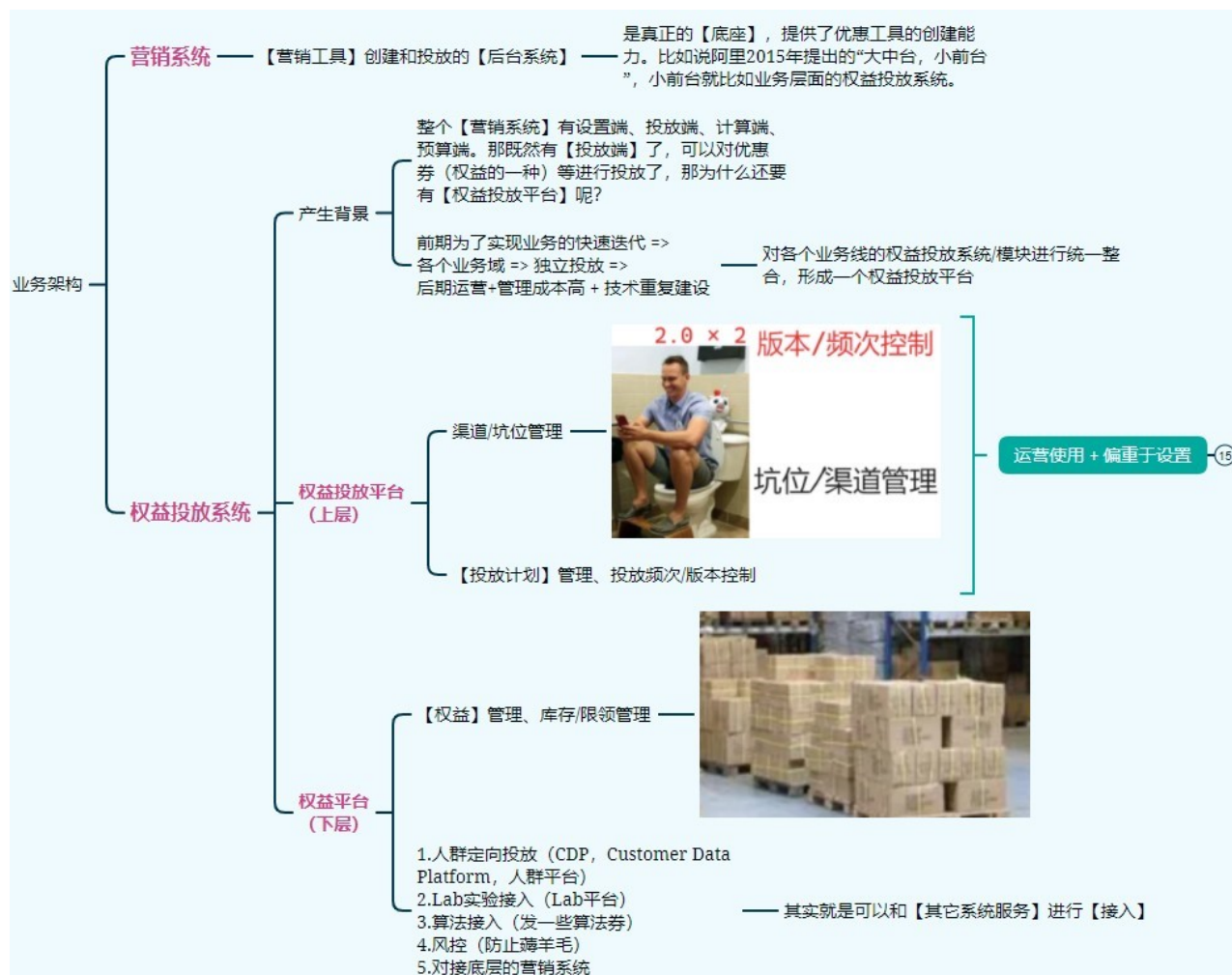
---

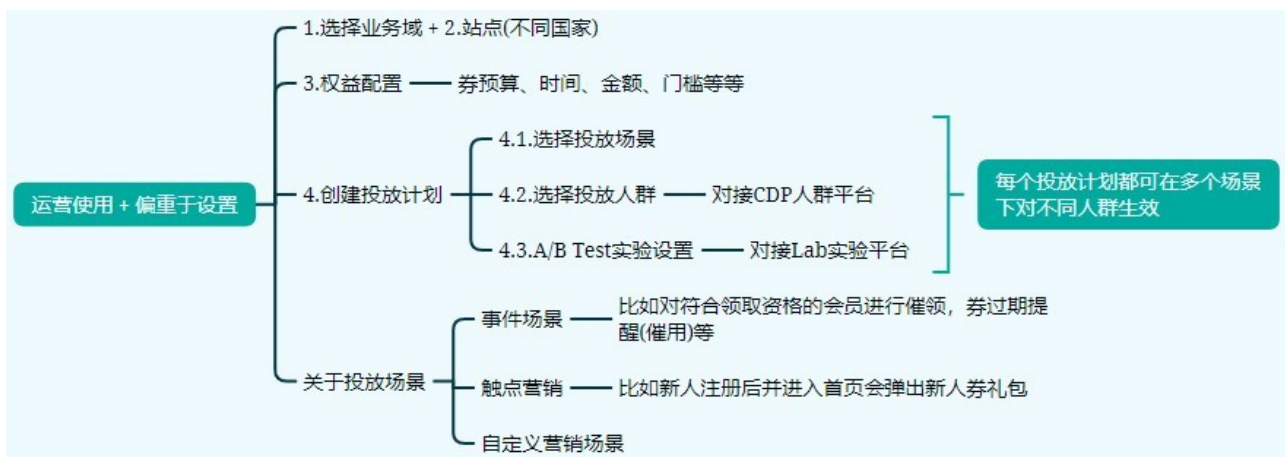
## 1. 营销业务场景

- 首先，整个营销业务是建立在苏宁的电商业务之上的。
- 在苏宁易购APP上面有很多页面，包括落地页、首页、PDP(商品详情)、搜索、商品列表、下单页面、购物车页面都可以去发券。包括领券中心，也可以去领取一些不同品类的优惠券。
- 最后，用户领券之后，可能就会去在平台上去消费，从而达成交易。
- 总结：
  - 宏观：营销业务 => 苏宁电商业务
  - 具体：各个发券页面（可以从用户视角来看）+ 领券中心
  - 用户：用户领券消费，达成交易

## 2. 业务架构

整个业务架构分为营销系统和权益投放系统，营销系统是营销工具创建和投放的后台系统，权益投放系统可以分为权益平台和权益投放平台，权益平台主要负责对接各个外部系统，包括营销系统，权益投放平台主要负责券的投放，主要面向运营人员。





### 3. 数仓架构

- 数据全链路：数据源(业务数据库MySQL、埋点数据、订单数据、配置数据(权益 配置、投放配置)) => ODS => 数据明细层(DWD明细事实表 + DIM维度表) => DWS数据汇总层 => ADS数据应用层
- 分层介绍：
  - ods层主要包括营销模板活动表(券、非券)、圈品表、黑名单表、领取表、营销快照表、预算规则表、预算实例表、权益配置信息表、权益(券模板)表、权益领取表
  - dim层主要以营销活动表(券、非券)(来自ods层)为主表建模，把创建【券模板】的维度和属性信息都落到dim层，包括权益维度表、券维度表、投放计划表、投放场景表、券池
  - dwd层是基于ods层，基于权益从发放到核销的业务过程(曝光、领取、核销)建模，把dim表中的主要分析维度(权益维度、券维度、投放计划、投放场景)下沉到dwd层，方便下游依赖方进行查询和分析
  - dws基于业务分析视角(运营、产品/管理层)，进行各类维度的各项指标(曝光、领取、核销)聚合汇总，形成营销数据集市，分析维度例如：权益类型、权益投放场景、国家、event\_id、purpose\_id。
    - 整个dws层有两种汇总粒度，分别针对不同人员(运营、产品/管理层)，针对运营人员一般就是券粒度进行汇总，针对产品/管理层一般就是营销工具粒度进行汇总
  - ads层就是业务应用层，引用dws层的数据进行加工组装，形成不同的ads层表，支撑上游业务。

### 4. 开发管理规范

大致有：

- 层次调用：


- 禁止逆向调用。 优先使用公共层，一个计算任务只允许一个输出表dws 汇总层优先调用 dwd 明细层，dwd明细层累计快照事实表优先调用 dwd 事务型事实表，保持数据的一致性产出。
- 命名规范：
  - 主要参照wiki提供词根规范表，分区后缀表，还有主题域简称表。
  - 然后表的命名在每层都不一样，比如 dwd 是 层级 + 数据域英文缩写+业务内容简写+分区类型，dws在业务内容后面再加个 时间粒度。
- 代码开发：
  - 禁止使用select \* 操作
  - 代码中需要添加必要的注释，增加代码可读性
  - SQL代码禁止单独使用Join语法，Join应当显式标明是inner join、left join、left semi join的类型
  - A left join B 时，如果B表存在限制条件，例如需要 B.day = xxx，则需要使用子查询，不推荐使用 A left join B where B.day = xxx

## 2.参与数据同步工作

---

### 1.负责哪些数据的同步工作？

这一部分我做的主要工作就是：从营销系统中同步营销领取表(buyer\_resource)数据，从权益投放系统中同步权益领取数据(plutus\_lottery\_record\_history)，直接在Web页面上配置同步任务，配置完成之后通过调度系统发布上线，同步步骤：

- 先添加业务数据库的数据源
- 再添加Hive的数据源
- 创建数据同步任务，并对任务进行一些配置，包括创建Reader、创建Writer、字段映射、接着可能需要配置一些，比如**1.并发度**，**2.失败了怎么重试**，**3.失败多少条需要报警**等等
- 创建完任务之后， 就是将任务发布上线
- ：同样的同步任务，只需要配置一次，然后去发布上线，然后去调度就可以了，后面出问题的话，先下线，然后去修改即可，修改完成再去上线。

### 2.营销业务的数据源有哪些？

- 1.业务数据库（营销系统、权益投放系统）
- 2.Web/APP 前端埋点用户行为数据（权益曝光、用户点击领取、用户点击关闭等）
- 3.交易域中的数据（订单表、订单子表(仅包含用户使用优惠券产生的订单)）



- 交易域中的数据：关于交易域中的数据，可以理解为【用户产生交易行为时】，在落订单的时候，会去交易子订单中查看哪些子订单用了营销信息，然后把营销数据落到业务数据库中，但是交易这一块不是我们负责的，我们负责的主要就是领取。
- Web/APP 前端埋点用户行为数据：曝光这一块也不需要我们负责，埋点数据不需要我们进行同步，可以理解为集团每天都会去落一份大的日志表，我们只需要从日志表中，通过我们的埋点参数，去解析我们要用的数据即可：(当然这部分也不是我负责的，我主要负责对领取相关的数据进行同步)
  - 我们会收集以下事件的信息：
    - 用户进入权益活动页面 => 初始化 => 请求曝光资格(判断用户是否有资格参与当前权益活动)
    - => 【曝光(核心)】
    - => 【用户点击领取/用户关闭页面(核心)】
  - 包括用户的一些信息：
    - 用户id、设备id， session\_id(会话)，日志产生时间

### 3. 订单表是怎么同步的？

增量合并全量同步，订单表的历史数据量非常大，几十亿条，肯定是增量同步，但是我们增量同步之后会和昨天的全量数据进行一个合并形成当天的全量表，因为需要满足一些查询历史订单的需求。

### 4. 你知道的数据同步方式有哪些？

- 全量同步：每天都将业务数据库中的全部数据同步一份到数据仓库，建议一般对于小数据量可以选择全量同步，而对于大数据量的数据则选择增量同步
  - 有两种全量同步的方式：全量覆盖 / 分区全量表
    - 全量覆盖：简单粗暴的进行覆盖，可能对于一些维度表，我们会选择全量覆盖同步，因为历史维度对分析是没有用的。但如果历史变化的维度对分析也是有价值的，我们可以结合具体的业务场景，采用增加新行、增加新属性、快照存储、历史拉链存储等方式进行处理。
    - 分区全量表（又称为快照表）：每一天都将业务数据库中的全量数据同步到最新的分区中，我们可以获取历史某一天的全量数据，也可以获取当前最新的全量数据。
- 增量同步：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。这样每日增量同步过来的数据都放在增量分区表的每日分区内，可以保存数据的历史变更状态，【不过要取历史全量数据的数仓最新状态数据对于增量分区表的话就会取数麻烦些（这里刚好是滴滴问到我的地方！卧槽）】。比如要取全量订单记录当前最新状态，需要先

扫描增量分区表历史全部分区得到全量订单数据，再取每笔订单数仓最新的那条状态记录。

- （好处坏处）离线来讲：增量的好处就是【只需要同步很少的数据量】，但如果要取历史全量数据的数仓最新状态数据对于增量分区表的话就会取数麻烦些，因为要扫描所有的分区数据。
- 离线增量和实时增量：
  - 离线增量：一般是做 T+1 的处理方式，即次日会同步前一日的所有的所需要的业务数据
  - 实时增量：一般是实时监测，只要所需要的业务数据发生变化，就将该数据立马同步到数仓中
- 增量合并全量同步：其实它本质上也属于全量同步中的【分区全量表】的一种，只不过做了优化，就是我们不需要每次都从业务库中拉取【历史全量数据】到新的分区内，而是从业务库中仅仅拉取增量数据，然后和数仓中前一天的全量数据进行union all+去重，最后写入到数仓新的全量数据分区内即可得到新的历史全量数据！
  - 优点：在最新分区表里1.即满足了数仓的全量最新快照数据，2.同时也降低了对业务源数据库的压力负载，并且每次也不需要同步全量数据
  - 具体实现：先使用union all将增量分区表和全量分区表进行合并，再对结果使用 row\_number() over (partition by 去重主键 order by 时间字段 desc) as rk 来打上序号，最后嵌套一层子查询使用where rk = 1 过滤出最新的数据即可！
  - 目前在行业中离线数据接入大致也是这种思路：增量合并全量

## 5.说一下数据库日志解析同步？

背景：大多数主流数据库都已经实现了使用日志文件（**binlog**）进行系统恢复，因为日志文件信息足够丰富，而且数据格式也很稳定，完全可以【通过解析日志文件获取发生变更的数据】（**CDC: Change Database Capture**）。

【Binlog可以记录数据库中所有的操作，包括增、删、改等操作】

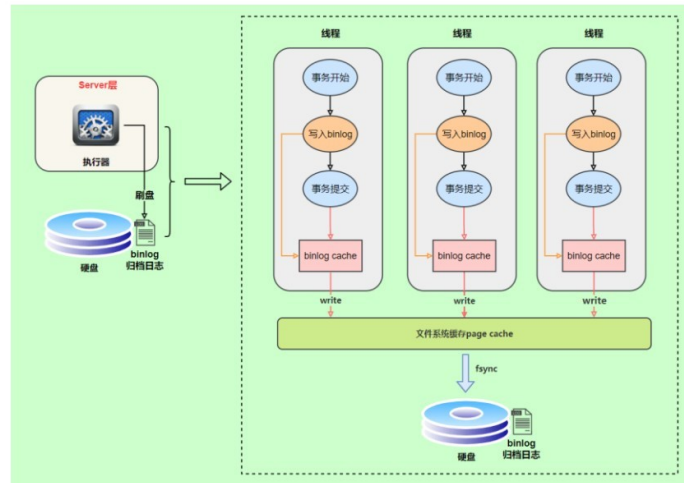
【实时同步的话，类似的更多使用Flink CDC来进行】

应用：因此有了这一机制，我们可以通过监控该文件，通过该文件的变动从而满足全量或者增量数据同步的需求，比如 **mysql**，一般是通过解析 **binlog** 日志方式来获取增量的数据更新，并通过消息订阅模式来实现业务数据库数据的实时同步（当然对于文件的实时同步可以使用Flume）

通过解析Binlog可以实现数据的增量+全量+实时同步



Mysql 的 binlog 一般的产生如下：



数据库日志解析同步的优缺点：

优点：

1. 日志文件信息足够丰富，数据格式稳定，通过解析日志文件满足增量数据同步的需求。
2. 通过网络协议，确保数据文件的正确接收，提供网络传输冗余，保证数据的完整性。
3. 实现了实时与准实时同步的能力，延迟控制在毫秒级别。

缺点：

1. 当数据更新量超出系统处理峰值，会导致数据延迟。
2. 投入较大，需要在源数据库和目标数据库之前部署一个实时数据同步系统【Flink CDC可以实现】。
3. 数据漂移和遗漏。

### 3.DWD明细事实表的宽表化

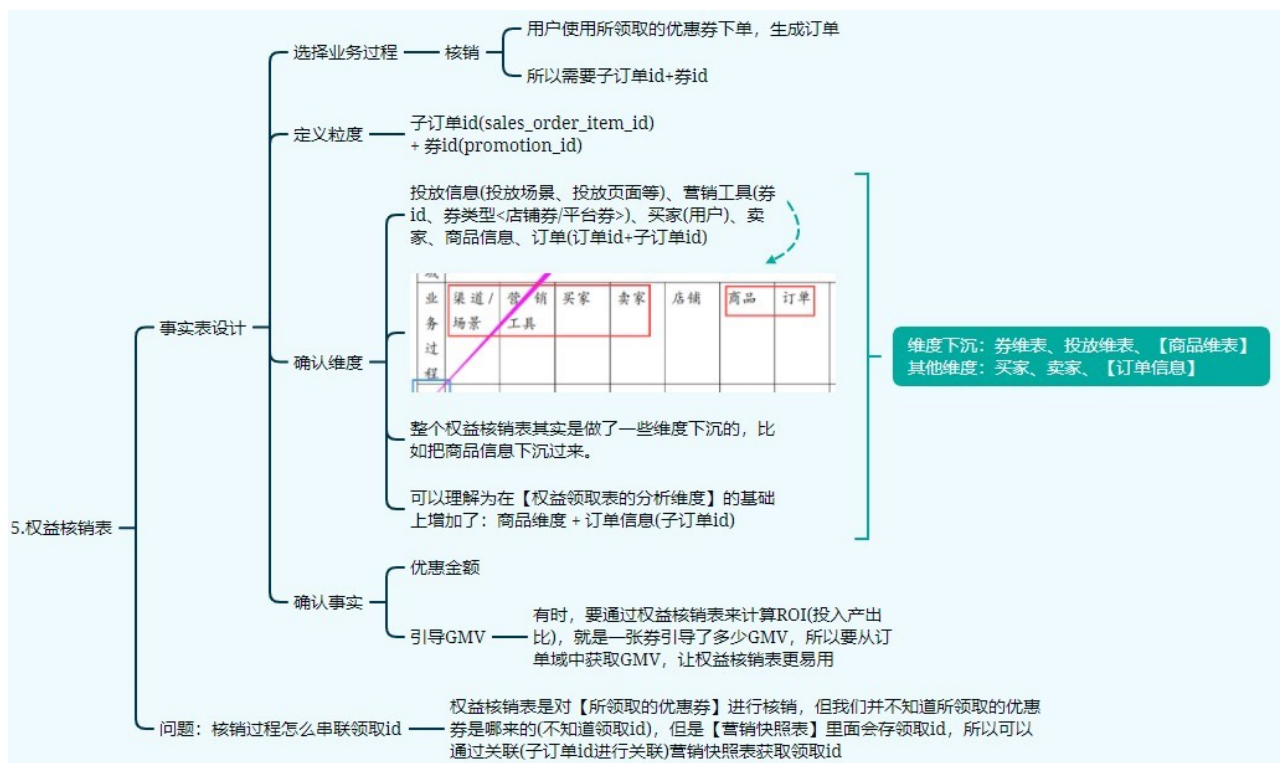
#### 1.明细事实表的宽表化怎么做的？有哪些表？

主要就是对权益领取和权益核销的这两张事实表进行宽表化处理，其实就是将DIM层中的一些主要的分析维度，比如说券维度、权益维度、投放计划维度跟投放场景维度，下沉到DWD层中的事实表中，比如说权益领取表，权益核销表。方便下游进行统计分析嘛。

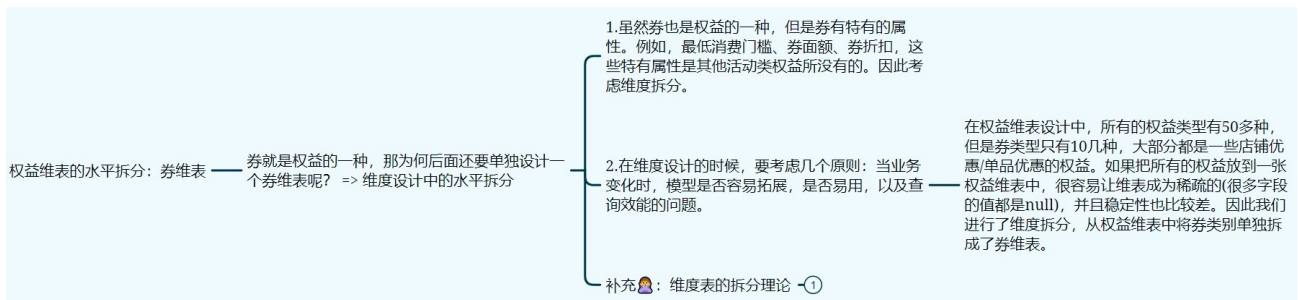
- 其实就是直接关联Join的一个操作，通过id

- 宽表不是我设计的，是组里其他人设计的，Mentor直接给我安排的任务，况且宽表关键点不在于怎么实现，而是怎么设计能够覆盖大部分的查询需求

其中，权益领取表和权益核销表就是按照维度建模的过程来建设的：



## 2. 怎么理解你提到的分析维度中的权益维度和券维度？



关于维度表的水平拆分理论：

### 什么情况下需要进行维度拆分？

当一张维度表中1.包含多个类别,2.加工逻辑十分困难,3.有部分**维度属性**可以单独处理或者不常用时可以考虑将维度拆分。

无论是维表**是分还是合**，都需要从以下角度权衡：

当业务变化时，模型是否容易扩展，是否易用，以及查询效能问题。

通常来说，拆分方法有以下几种：

(1)水平拆分（数据层面）：例如航旅的商品和普通商品，都属于商品。那我们就可以提取**航旅和普通商品的公共属性**去建立一个【商品维表】，然后建立一个子维度表【航旅商品维表】，用来存放航旅商品的特有属性。

(2)垂直拆分（维度属性层面）：当某些维度属性的来源表**产出时间较早**，而某些维度属性的**产出时间较晚**；或者某些维度属性的**热度高、使用频繁**，而某些维度属性的**热度低、较少使用**，都可以使用**主从表**进行垂直拆分。

## 4. 参与权益数据指标开发工作

### 1. 整个营销域的数据指标体系是什么样的？

首先，整个指标体系是离不开【分析维度】的：

- 国家/站点  
券类型(平台券/店铺券)+券id  
权益类型+权益子类型  
投放场景+场景id  
投放坑位+坑位id  
event\_id(在大促中/还是日常运营活动投的)  
purpose\_id(投放的是新用户/流失用户)

接着就是我们的【指标(ADS层中的那张表)】：

- 指标的话可以从我们整个的业务过程来进行划分，券的曝光、领取、核销

- 曝光PV、曝光UV  
领取UV、领取张数、领取金额  
核销UV、核销张数、核销金额
- 包括我们可以去看整个业务过程的一个漏斗率：
  - 曝光到领取的漏斗
  - 领取到核销的漏斗
- 一些汇总类的指标：
  - 总订单数(3个角度：用户下订单数量、SKU级别订单数量、卖家级别订单数量)
  - 总引导GMV
  - 总DAU(每日活跃用户)
- 一些比例型的指标（核销）：
  - 核销UV占订单UV的比例
  - 核销的GMV占总GMV的比例
- 我们业务最为关心的就是【券的经效和成本】，那我们就可以去统计一些指标：
  - ROI、CIR
  - CPB、CPO、CPCO、CPSO
  - AOV

**投入产出比** ROI : Guided GMV/Spending 引导GMV/核销金额  
CIR : Spending/Guided GMV 核销金额/引导GMV

CPB(Buyers) : Spending/Guided Buyers 核销金额/引导买家数  
CPO(SKU Orders) : Spending/SKU Orders 核销金额/SKU订单数  
CPCO(CheckOut Orders) : Spending/Checkout Orders 核销金额/买家下订单数  
CPSP(Seller Orders) : Spending/Seller Orders 核销金额/卖家订单数

**AOV: 每个订单的平均 GMV**

AOV : Guided GMV/SKU Orders 引导GMV/SKU订单数  
AOV : Guided GMV/Checkout Orders 引导GMV/买家下订单数  
AOV : Guided GMV/Seller Orders 引导GMV/卖家订单数  
ARPU : Guided GMV/Guided Buyers 引导GMV/引导买家数

## 2.你负责过哪些指标？怎么计算的？

- $ROI = \text{Guided GMV} / \text{Spending} = \text{引导GMV} / \text{核销金额}$
- $AOV = \text{每个订单的平均GMV} = \text{引导GMV} / \text{订单数}$   
订单数有三个维度：
  - SKU Orders: SKU订单数
  - Checkout Orders: 买家下订单数
  - Seller Orders: 卖家订单数
- $GMV = \text{总的GMV} = \text{用户实际支付} + \text{平台/店铺总补贴} + \text{实际运费} + \text{平台补贴运费}$
- 其实问题1中提到的指标都可以说是自己开发的...

Tips:

- 可以理解为这些指标，包括GMV啊等等，都是基于前一天的数据进行计算的。比如 ROI/CIR/GMV等等，都是基于前一天所有的数据进行计算的。除了这些指标，包括一些核销、领取等等都是看的当天的数据！比如今天是2024/2/22，我们计算的就是2024/2/21的数据！
- 上面这些指标都是分开去统计的，比如说领取，我们只会去看当天的领取，核销也是看的当天的核销，而不是仅仅去看当天领取的核销，所以我们统计的当天的核销数据，可能很久之前就领好了！就是不会去看这种严格的漏斗，但其实差不多的，因为很多人都是领取了就核销，所以从长期来看这么一个整体的趋势是不会变的！

## 5.小文件治理

---

<https://blog.csdn.net/zhaomengszu/article/details/124520732>

预备三问：

### 1. 为什么大数据开发要关注小文件？

- 首先大数据开发是小文件的制造者
- 资深大数据开发需要企业中参与数据治理，小文件治理是重要的组成部分，大数据开发是小文件治理的参与者
- 资深大数据开发，技术专家往往需要具有大数据全局统筹的能力

### 2. 什么是小文件？

- 实际公司生产中对于小文件的大小没有统一的定义，一般公司集群的 `blocksize` 的大小在 128/256 两者居多，小文件肯定是要远小于 `blocksize` 的文件。一般公司小文件的大小定义如 1Mb, 8Mb, 甚至 16Mb, 32Mb 更大。
- 日常生产中 HDFS 上小文件的产生是一个很正常的事情，有些甚至是不可避免的，比如 jar、xml 配置文件、tmp 临时文件、流式任务等都是小文件的组成部分。
- 小文件的产生，更多的是因为集群设置不合理导致的

### 3. 小文件产生的途径有哪些？

- 数据源本身就包含众多小文件
- 数据增量导入，导入频繁的话也可能产生很多小文件。
- 使用动态分区插入数据时，产生大量的小文件（比如使用动态分区时，由于分区字段设置的不合理(多级分区)，导致产生非常多的文件夹，每个文件夹里面的文件可能非常小）



- Map任务中，Map的个数过多，每个MapTask都会产生一个文件；MapReduce任务中，**Reduce**数量设置过多，导致产生很多小文件，因为reduce的个数和输出文件是对应的
- 分桶表

## 1.集群小文件的危害

总结来看，小文件首先影响着整个集群的存户，不仅对 **namenode** 有影响，对 **datanode** 也有影响。其次，小文件对 计算性能 浪费较大，同时会拖慢任务执行时长。

### 1.小文件对 **namenode** 的影响

- 【每个文件和目录】都对应元数据的一个记录，小文件数量庞大会导致元数据存储压力增加。
- 同样的一个文件，不同形态的存储方式，所占用的**namenode**内存相差是非常大的。
  - 比如一个大小为192MB的文件，Block大小为128MB，可以存储为2个Block(128MB+64MB)，如果每个Block存储3个副本的话，一共就有6个Block文件。
  - 但如果将这个文件切分为192个1MB的小文件，整体文件大小也是192MB，但会存储为192个Block，如果每个Block存储3个副本的话，一共就有576个Block文件。这就占用了大量的namenode内存空间。

### 2.小文件对 **datanode** 的影响

- 大量的小文件，意味着数据寻址需要花费很多时间，花费在【寻址】的时间比【文件读取写入】的时间更多。这种就违背了 **blocksize** 大小设计的初衷（实践显示最佳效果是：寻址时间仅占传输时间的 1%）

### 3. 小文件对计算的影响

- 文件越多，意味着越多的Block，以及更多的节点分布
- 对于MapReduce任务而言，Map的个数取决于切片个数。我们会对每一个文件单独进行切片，文件越多，切片个数就越多，需要开启的MapTask个数就越多。这就会产生更多任务相关的开销。
- 另外，虽然可以开启Map任务前的小文件合并，但是这个过程需要从不同的小文件节点上1.建立连接，2.进行数据读取，3.网络I/O，然后进行合并，这样也会增加资源的消耗和计算的时间。  
 （说白了就是将存储在各个节点上的小文件都拉取到同一个节点上进行合并，这个拉取的过程也非常消耗资源）

## 2.小文件参数治理

优化分区方式:

- 如果小文件是由于【动态分区设置不合理】导致的(多级分区),可以考虑优化分区方式,比如减少分区层级
- 但很多时候由于业务需要,分区方式无法更改

设置计算引擎读取数据时的合并参数: map前合并 + 输出合并

- ```
-- 两套参数, 小文件治理

-- 开启 map 前小文件合并, 官方默认开启
set hive.input.format =
org.apache.hadoop.hive.q1.io.CombineHiveInputFormat;

-- 输出合并 : Map/MapReduce输出
set hive.merge.mapfiles = true;      -- 开启 map-only 任务的小文件合并
set hive.merge.mapredfiles = true;  -- 开启 map-reduce 任务的小文件合并
-- 表示当一个MR作业的平均输出文件大小小于这个数字时, Hive将启动一个额外的 map-reduce 作业,将输出文件合并为更大的文件
set hive.merge.smallfiles.avgsize=32000000;
-- 控制合并后的文件大小, 一般和block大小保持一致
hive.merge.size.per.task=25600000;

-- 控制map输入的大小
set mapred.max.split.size=256000000;      -- 设置切片最大值
set mapred.min.split.size=32000000;      -- 设置切片最小值
set mapred.min.split.size.per.node=32000000; -- 设置切片最小值
set mapred.min.split.size.per.rack=32000000; -- 设置切片最小值
set mapreduce.job.queueName=root.paas_test ;
```

```
-- 如果我们设置的切片最小值是8MB，前面设置的
hive.merge.smallfiles.avgsize为32MB，那么当第一个MapReduce输出 时
，即使文件的平均大小小于32MB(例如10MB、15MB、30MB)，也不会开启新的
MapReduce去进行小文件合并

-- 因为第二个MapReduce进行小文件合并的原理其实就是使用
CombineHiveInputFormat进行合并，但是CombineHiveInputFormat的合并 其
实真正受到mapred.min.split.size参数的控制，也就是切片的最小值，只有 当
文件大小小于切片最小值时，才会去进行小文件合并

-- 前面提到，我们设置的切片最小值是8MB，而输出的文件大小为10MB、15MB、
30MB，均大于切片最小值，所以就不必进行合并了

-- 所以要将mapred.min.split.size/size.per.node/size.per.rack的
大小设置为32MB，此时输出的文件小于切片最小值，才会继续进行小文件合并
```

小文件合并成大文件：

- 对于分区表而言，最有效的方式就是将单个分区(文件夹)下包含的文件进行合并。
- 如果数据存储格式为ORC：运行下面这个命令(可以连续运行)，可以对指定分区进行小文件合并

```
alter table tableName_orc partition
(date="20200101") concatenate;
-- 使用concatenate命令合并小文件时不能指定合并后的文件数量，但
可以多次执行该命令。
```

- 如果数据存储格式为非ORC：运行下面这个命令，可以将小文件个数合并到指定的N个以内

```
insert overwrite table ${table_name} partition(...)
select *
from ${table_name}
distribute by ${column_name} (rand()*N);
```

## 6.数据质量监控(DQC)

---

## 1.你配置过哪些监控规则？

首先，有时间的话，理解为所有的表，或者大部分的表都要去配置DQC，尤其有很多下游去用的时候，必须去配置DQC。就拿DWD层中的一些宽表（权益领取表）举例，他有3个必须去配置的规则：

- 行数等于0
- 还是权益领取表，它的主键是否唯一
- 然后比如说权益领取表，是按照日期+国家分区的，那我们要确定国家分区，他的枚举值是不是等于6个

包括一些常识性的规则和业务性的规则：

- 常识性的规则，比如一个比例型指标，它的百分比肯定要设置在一个合理的范围内；
- 业务性规则，比如我们一些领取型的券，它的金额不能是负数，不能超过**50美金** (因为不可能一单优惠50美金)

拓展表达：

就是说数据质量虽然可以通过配置一些DQC规则来保证，但我们也不会去配置过多规则，因为规则配置太多的话，其实有时会它会影响到这个【表产出的性能】。所以我们会根据一些实际情况来去配置一些规则。

## 2.什么是DQC？

定义：DQC，Data Quality Check，数据质量检查。通过【配置】DQC的数据质量校验规则，可以实现在数据处理过程中进行自动的数据质量监控。DQC可以监控【数据质量】并报警，但是它不会对数据产出进行处理，需要报警接收人判断如何处理。  
(助记：通过配置规则 => 可以实现 => DQC监控+报警)

核心作用：对核心的维度、指标数据进行监控。例如主键是否唯一，量级是否符合预期，维度是否做了非空处理，指标字段【数据类型】和【精度】是否有问题等等。



补充：DQC数据监控规则有强规则和弱规则：

- 强规则：一旦触发报警就会阻断任务的执行（将任务置为失败状态，使下游任务不会被触发执行）。
- 弱规则：只报警但不阻断任务的执行。

3.说一下数据质量的四大特性

- 1.完整性：数据的记录和信息是否完整，是否存在缺失的情况。数据的缺失主要包括记录的缺失和记录中某个字段信息的缺失，两者都会造成统计结果不准确。
  - 举例：比如交易中每天的订单数为100万条，如果突然有一天变成了1万条，那么很可能就是数据记录缺失了。对于记录中某个字段信息的缺失，比如订单中的商品ID，这些字段的空值个数肯定是 0，一旦大于 0 就是出现了字段信息的缺失。
- 2.准确性：数据的记录和信息是否准确，是否存在异常或者错误的信息。
  - 举例：比如说某个订单中的金额是负的。
- 3.一致性：一致性一般体现在跨度很大的数据仓库体系中，比如苏宁易购数据仓库，内部有很多业务数据仓库分支，对于同一份数据，必须保证一致性。
  - 举例：例如用户 ID，在不同的业务数据库中，必须都是同一种类型，长度也需要保持一致。



- 4. 及时性：保障数据能够及时产出，比如数据是否在规定的时间内更新。
  - 举例：可以配置SLA基线，如果任务延迟产出，会在基线前30分钟进行告警
- 5. 助记：
  - 完整 => 记录和信息是否完整 => 缺失
  - 准确 => 记录和信息是否准确 => 异常/错误的信息
  - 一致 => 体现在很大的数据仓库体系中 => 举例苏宁易购
  - 及时 => 数据能够及时产出 => 是否在规定的时间内更新

## 4. 你们公司是怎么配置SLA的？

苏宁易购的话，平台化做的还可以，公司有一个数据集成平台，里面嵌入了一个数据治理管理平台，配置SLA的时候一般就是在这个平台上配置的。

首先整个监控规则是：

1. 按照模板进行配置的，我们可以根据需求选择系统内置的规则模板，包括【表级】内置规则模板和【字段级】内置规则模板。
2. 对规则进行设定：
  1. 首先需要选择规则的类型，包括动态阈值、波动率和数值型。
  2. 然后定义规则影响，包括强规则和弱规则。强规则就是会对下游任务进行熔断，弱规则不会，所以如果是一些严重的问题，就应该设置为强规则。
3. 接着就会生成规则，我们需要选择将规则应用到哪里？即选择需要校验的表或字段，然后选择需要校验的表分区。
4. 最后是试跑，包括对规则进行触发，然后检查规则是否配置正常。

## 7. 任务排查与解决

---

（开发用的Hive SQL，所以更熟悉Hive的优化）

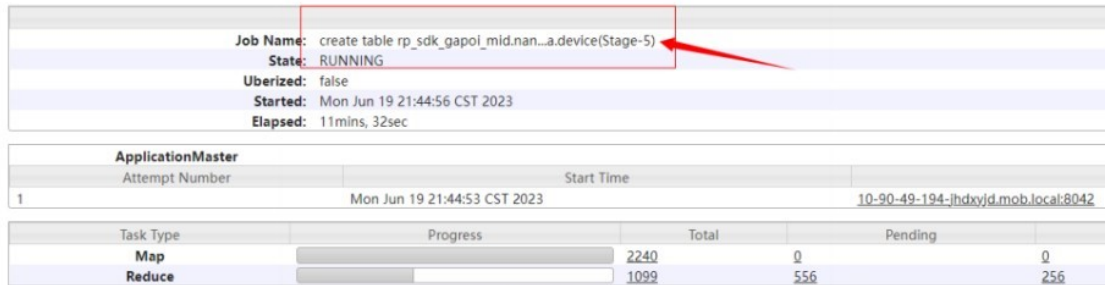
### 1. 你是怎么排查出有问题的任务的？

首先，一般的任务可能30min之内可以跑完，但如果有些任务30min之内还没有跑完，那我们就可以考虑对该任务进行优化，或者说是不是出现了数据倾斜问题。

接着就是问题的排查：

- 1. 定位Stage
- 2. 查看执行计划

- 3. 定位关键字
- 4.数据探查
- 1.定位**Stage**: 看任务执行慢在哪个 stage, 一般 Hive 默认的 jobname 名称会带上stage 阶段, 如下通过 jobname 看到任务卡住的为 Stage-5, 如果 jobname是自定义的, 那么需要通过 task 执行日志来确定 stage



|           |                                                       |
|-----------|-------------------------------------------------------|
| Job Name: | create table rp_sdk_gapoi_mid.nan...a.device(Stage-5) |
| State:    | RUNNING                                               |
| Uberized: | false                                                 |
| Started:  | Mon Jun 19 21:44:56 CST 2023                          |
| Elapsed:  | 11mins, 32sec                                         |

| ApplicationMaster |                              |
|-------------------|------------------------------|
| Attempt Number    | Start Time                   |
| 1                 | Mon Jun 19 21:44:53 CST 2023 |

| Task Type | Progress | Total | Pending |
|-----------|----------|-------|---------|
| Map       | 2240     | 0     | 0       |
| Reduce    | 1099     | 556   | 256     |

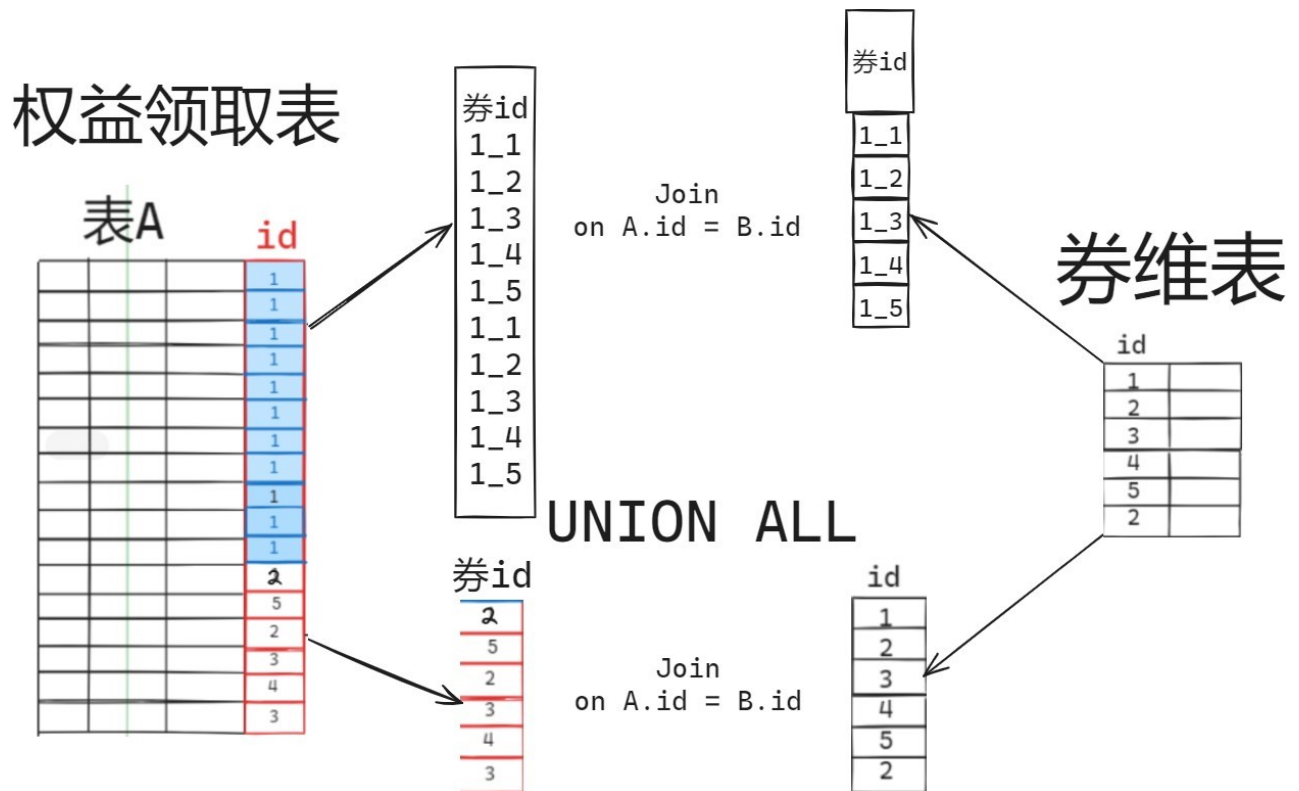
- 2.查看执行计划: 确定了执行阶段, 查看任务执行计划, 再通过表的别名, 关键字等判断出在哪个阶段产生了数据倾斜
- 3.定位关键字: 数据倾斜一般只有 join, group by, count(distinct), 找这些关键字
- 4.数据探查: 探查表数据, 关键列名等

## 2.你遇到过哪些异常任务? 你是如何解决的?

遇到过一个数据倾斜的案例, 具体而言:

- 因为我负责过部分DWD层中明细事实表的宽表化处理, 比如说权益领取表。其实宽表化处理的操作很简单, 就是通过id去和DIM层中的一些维度表进行关联, 然后获取对应维表中的一些属性字段, 可以理解成把这些属性字段冗余到事实表中, 做一个宽表化处理的工作。
- 当时处理时, 需要将【券维度】下沉到【权益领取表】中, 通过【券id】进行关联
- 但是我们的权益领取表其实是做的全量表(增量同步全量更新), 因为有些业务场景可能需要去【分析某些用户历史的领券记录】。然后整个权益领取表中, 可能权益有50多种, 但是券的话不到10种, 而且有些券可能很早之前就在发了, 而且效果很好, 领的人也很多, 那就会产生非常多的权益领取记录, 并且这些权益领取记录的券id都是一样的。那如果此时通过券id和券维表进行关联的话, 就会出现数据倾斜问题, 而且这种问题非常常见!
- 总结:
  - DWD层宽表化处理+使用id关联DIM层中的维度表
  - 具体例子: 将券维表下沉到权益领取表中
  - 权益领取表 => 全量表 => 某些券是热点券(原因) => 和券维表关联 => 数据倾斜 => 非常常见

解决方案:



- 比如表A的id字段是倾斜的(id=1的记录远大于其它id), 可以先把id=1的记录(倾斜字段)筛选出来作为表C(临时表), 然后在id字段上加一个随机数后缀, 然后表B膨胀(union all拼接)相应的倍数并与C相关联, 将表A无倾斜的数据和原来的表B相关联, 最后union all两次关联的结果

- ```
select *
from
(
    select concat(id, '_', FLOOR(RAND() * 5)+1)
    from A
    where id = 1
) as t1
left join
(
    select concat(id, '_', 1) from B where id = 1
    union all
    select concat(id, '_', 2) from B where id = 1
    union all
    select concat(id, '_', 3) from B where id = 1
    union all
    select concat(id, '_', 4) from B where id = 1
    union all
    select concat(id, '_', 5) from B where id = 1
) as t2
```

```
on t1.id = t2.id

union all

select
from A as t1 left join B as t2
on A.id = B.id
where A.id != 1

-- 总结:
select A.id from A join B on A.id = B.id where A.id != 1
union all
select A.id from A join B on A.id = B.id where A.id = 1 and
B.id = 1;
```

### 3.夜间任务失败，可能晚于出数时间，怎么处理？

第一时间：预计晚于最晚出数时间，第一时间通知下游应用方（运营、产品），让他们通知到他们的业务方。然后告诉他们预期什么时候处理完，要把这些评估出来。

失败任务的处理：

- 首先判断是不是资源的问题，资源问题导致自己的任务抢不到资源，然后一直等待，最后被系统Kill掉
- 还是说任务本身逻辑有问题，可能是代码Bug等等

## 8.常规问题

---

### 1.何时入职？

- 随时都可以，因为论文很早就发出去了，导师没有什么在校学习要求
- 但是三月中旬有个开题答辩可能要请几天假回学校，您看可以吗？

## 2.实习多久？

实习时长我这边都OK的，实习时间越久能学到的东西越多嘛，我也想有一段相对稳定的实习经历，把学习到的内容具体的实践一下。

## 9.简历深挖/拓展

---

### 1.如果交易链路会涉及到营销的数据，会应用你的模型吗？

在交易的链路中，我们生成订单的时候，它会去子订单中查找优惠记录，然后把这部分有优惠记录的子订单数据单独落下来，这部分数据是营销系统要用的。所以可以理解为：营销会去引用交易域的数据(子订单优惠记录)。

但是从中间CDM层来看，我们的数仓建设需要符合一个【高内聚，低耦合】的规范，即使交易和营销关联密切，但是各自的CDM层都是独立建设的，不会有交叉和引用。

### 2.整个数仓建设的完善吗？有没有迭代工作？或者说有没有对历史模型的优化工作？

首先，整个数仓一定是有迭代工作的，主要有以下几个层面：

- 业务视角：首先，业务在发展，我们的数仓模型很难把业务的所有过程都兼容进来
  - 随着业务的发展可能要加一些表啊或者字段啊等等，去满足下游业务的需要
- 模型设计本身：其次，数仓模型也取决于设计者的经验，所以肯定也要对数仓模型本身进行持续优化
  - 比如可以说一下DIM层中对权益维表进行水平拆分的例子
- 数据治理：当然，可能一些数据治理的项目(团队)，要求我们对数仓进行优化
  - 包括表的命名的规范、引用调用的规范等等

### 3.ODS层中的表是内部表还是外部表？

（可以先讲内部表和外部表的概念，再去说具体的应用）

从业务系统中同步到ODS层中的数据，这种需要保存的，一般用的都是内部表。而且一般由Hive管理的，不涉及与其他系统共享的数据，我们一般用的都是内部表。



定义：由external创建的表即为外部表，未用external创建的表即为内部表

区别：

- 数据管理：内部表由Hive自身管理，外部表由HDFS管理
- 数据存储：内部表存储在HDFS上的默认位置：/user/hive/warehouse，外部表存储位置自己指定
- 数据删除：内部表元数据和存储数据都被删除，外部表仅删除元数据，HDFS上的文件不会被删除

应用场景：主要从误删恢复(外部表好恢复，因为可以共享副本)和共享(外部表可以共享)角度来考虑

- 每天采集的埋点日志，在存储的时候建议使用外部表
- 抽取过来的业务数据，其实用 外部表 / 内部表 问题都不大，就算被误删，恢复起来也是很快的；但我们用的一般都是内部表，管理起来更方便。
- 统计分析的时候用到的中间表，结果表可以使用内部表，因为这些数据不需要共享，使用内部表更为合适

#### 4.大表和大表关联，出现数据倾斜？比如两个事实表

（遇到问题之后，先定位问题，再去根据具体的问题思考解决方案）

大表和大表进行Join，出现了数据倾斜：

- 首先可以检查有没有做SQL本身的优化，比如有没有分区裁剪，有没有这种减少后续Shuffle数据量的操作
- 其次，大表关联大表，我们是不能做一些广播的，这个时候可能需要对两个大表数据进行探查，看看JoinKey是否出现倾斜呢？
- 如果有出现倾斜的Key，那就把这些出现倾斜的Key单独拿出来，然后通过加随机数打散，再去和另一张表关联。关联之后再去和另一部分进行合并。