优雅的维度模型与多维分析设计实践

邱盛昌-OPPO-技术专家/数据团队主管

DataFunSummit # 2023

oppo DataFun.

个人介绍

联系微信号：QiuChanson

目录 CONTENT

01

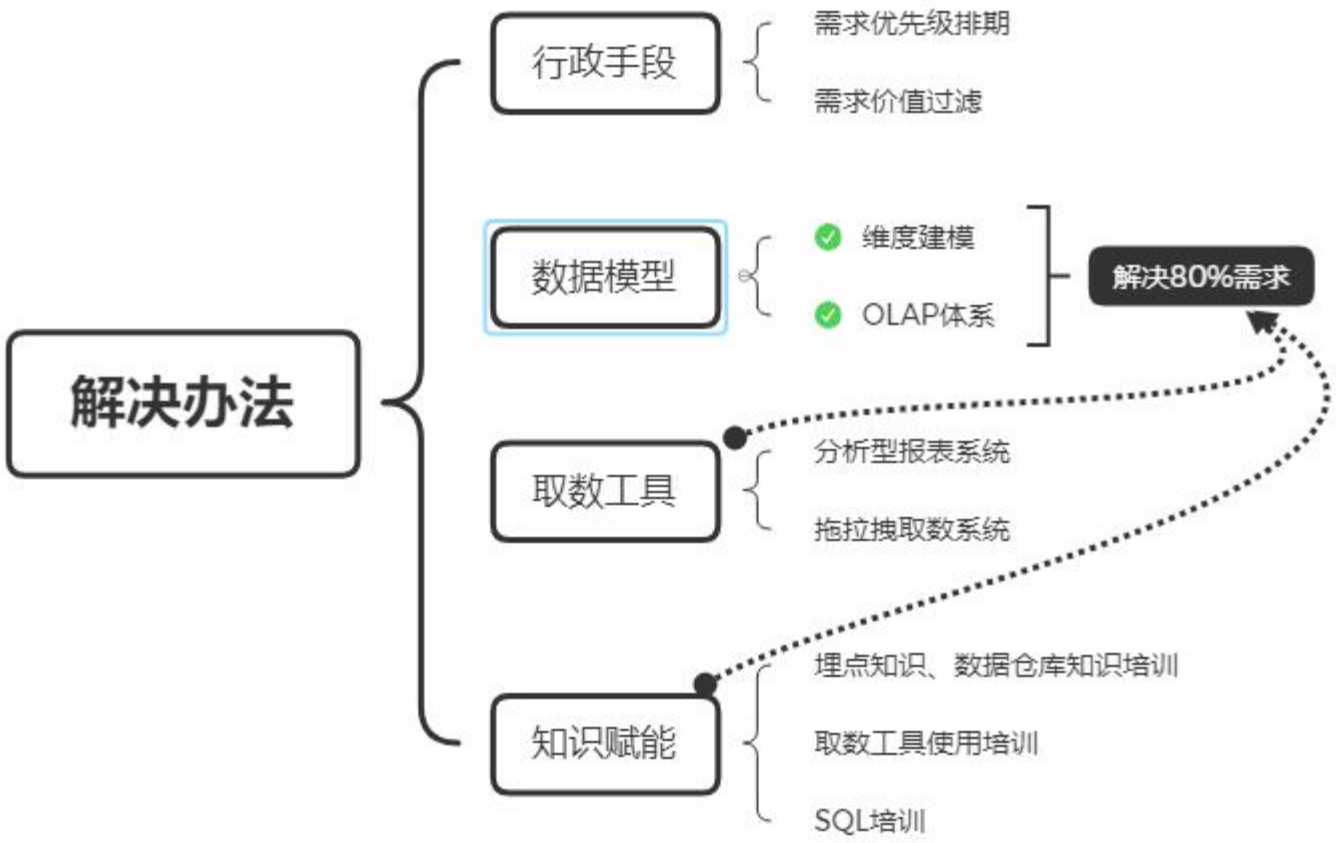
维度模型设计的必要性

DataFunSummit # 2023

DataFun.

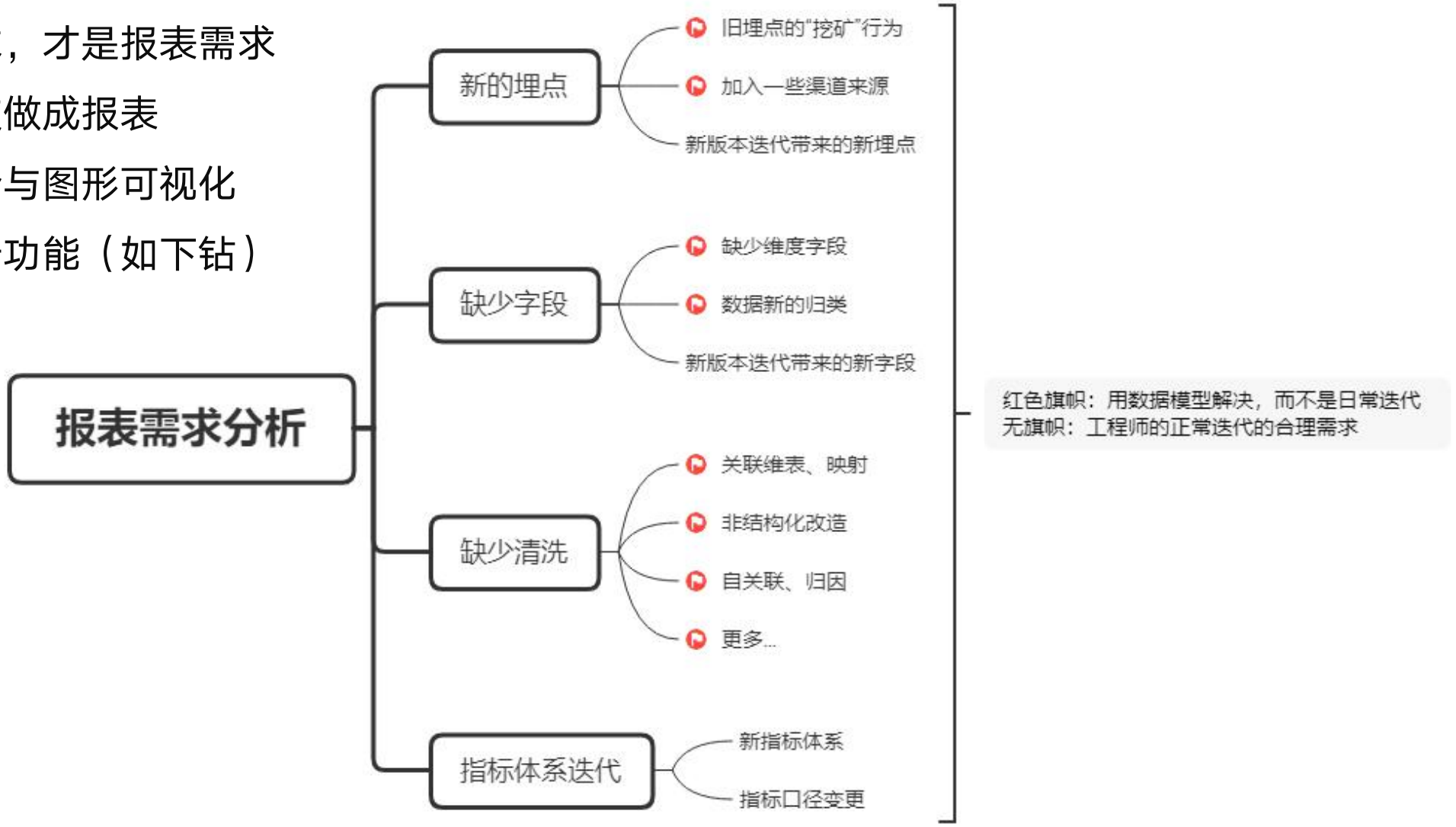
解决巨量的取数需求

- 产生巨量需求的本质：数据建设缺失或者建设方向错误，导致大量用数诉求无法满足



解决无穷无尽的报表需求

- 导致无穷无尽的报表需求原因：通用数据模型设计缺失，面向需求开发无抽象
  - ✓ 首先满足取数需求，才是报表需求
  - ✓ 常用的数据才能被做成报表
  - ✓ 更直观的指标组合与图形可视化
  - ✓ 更方便的报表分析功能（如下钻）



常见的几种错误方向

- 常见错误方向总结
  - ✓ **完全的面向需求开发**，需求要什么字段就做什么，一个需求一张RPT表，到处打听到表就直接取用，逻辑都堆在RPT层
  - ✓ **没有模型表，或者模型抽象不好**，具体表现为：不了解建模知识，SQL/Java能实现就行；对建模有些了解，有中间表但通用性较差，没有系统性的全局建模
  - ✓ **没有OLAP建设概念**，表拆得又多又散，进入无穷迭代陷阱，如：一个事件做一张事实表
  - ✓ **埋点的“挖矿石”行为**，只抽取这个需求要的字段，只做现在要的埋点事件，持续按需求“挖矿”好几年

02

极致分区表的数据仓库架构

DataFunSummit # 2023

数据源处理

- 埋点元数据：事件元数据
  - ✓ 先开发好事件元数据，用于指导主题域划分，模型表的分表和分区等

事件分类ID	事件分类名称	事件ID	事件名称	事件描述	上报数据量	重要程序	负责人
1	曝光	1_1	应用曝光	下载的应用曝光	1200	★★★★	张三
1	曝光	1_2	图片曝光	广告图片曝光	980	★	张三
2	下载	2_1	列表页下载	在列表页点击下载按钮	98	★★★★★	李四
2	下载	2_2	详情页下载	在详情页点击下载按钮	71	★★	李四
3	启动	3_1	主屏幕启动	在桌面启动应用	550	★★★	王五

数据源处理

- 埋点元数据：字段元数据
  - ✓ 字段元数据，用于指导物理表落地建设、字段清洗转换规则等

事件ID	事件名称	字段英文名	字段中文名	字段描述	公共描述	上报数据量	重要程序	负责人
1_1	应用曝光	enter_id	从哪个入口带来的曝光	...	从哪个入口带来的曝光|启动入口	1200	★★★★	张三
1_2	图片曝光	app_id	应用ID	...	应用ID	980	★	张三
2_1	列表页下载	app_id	应用ID	...	应用ID	98	★★★★★	李四
2_2	详情页下载	user_id	用户标识	...	用户标识	71	★★	李四
3_1	主屏幕启动	enter_id	启动入口	...	从哪个入口带来的曝光|启动入口	50	★★★	王五

数据源处理

- 埋点元数据：枚举元数据
  - ✓ 枚举元数据，用于指导码值翻译注释，监控、字段清洗转换规则等

枚举标识	枚举标识名称	枚举码	枚举值	上报数据量	重要程序
app_cat	应用分类	1	视频播放	120	★★
app_cat	应用分类	2	网上购物	110	★★
app_cat	应用分类	3	社交通讯	100	★★
from	渠道来源	101	头条	98	★★
from	渠道来源	102	阿里	90	★★

数据源处理

➢ 补录数据

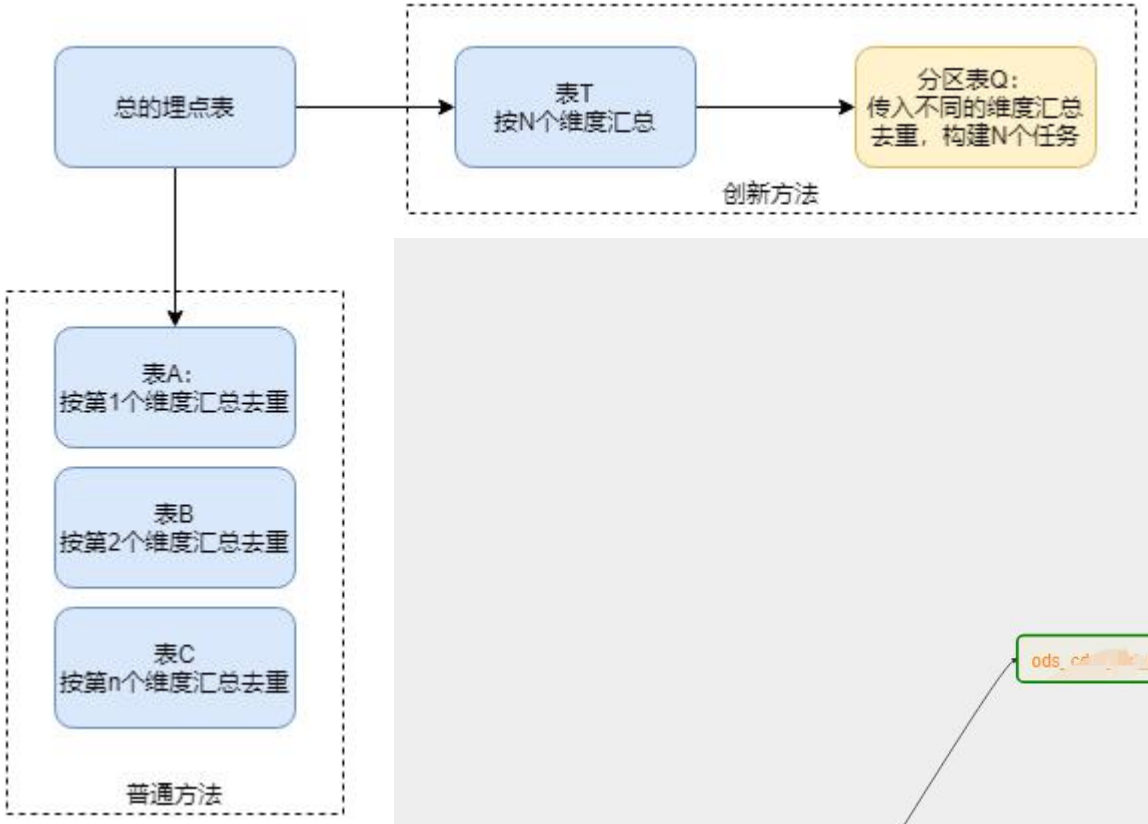
- ✓ 数据仓库建设中，肯定会有许多的补录数据，必须一开始就约束需求人**通过系统录入**，补录系统提供增删改功能，并实现自动导入大数据平台，切记**不可工程师手工导入**
- ✓ 这样处理补录数据的原因
  - 减少沟通成本，提高导入效率，提高工程师的幸福感
  - 保证导入数据的质量，需求人可自助迭代，并可以设置定期提醒补录

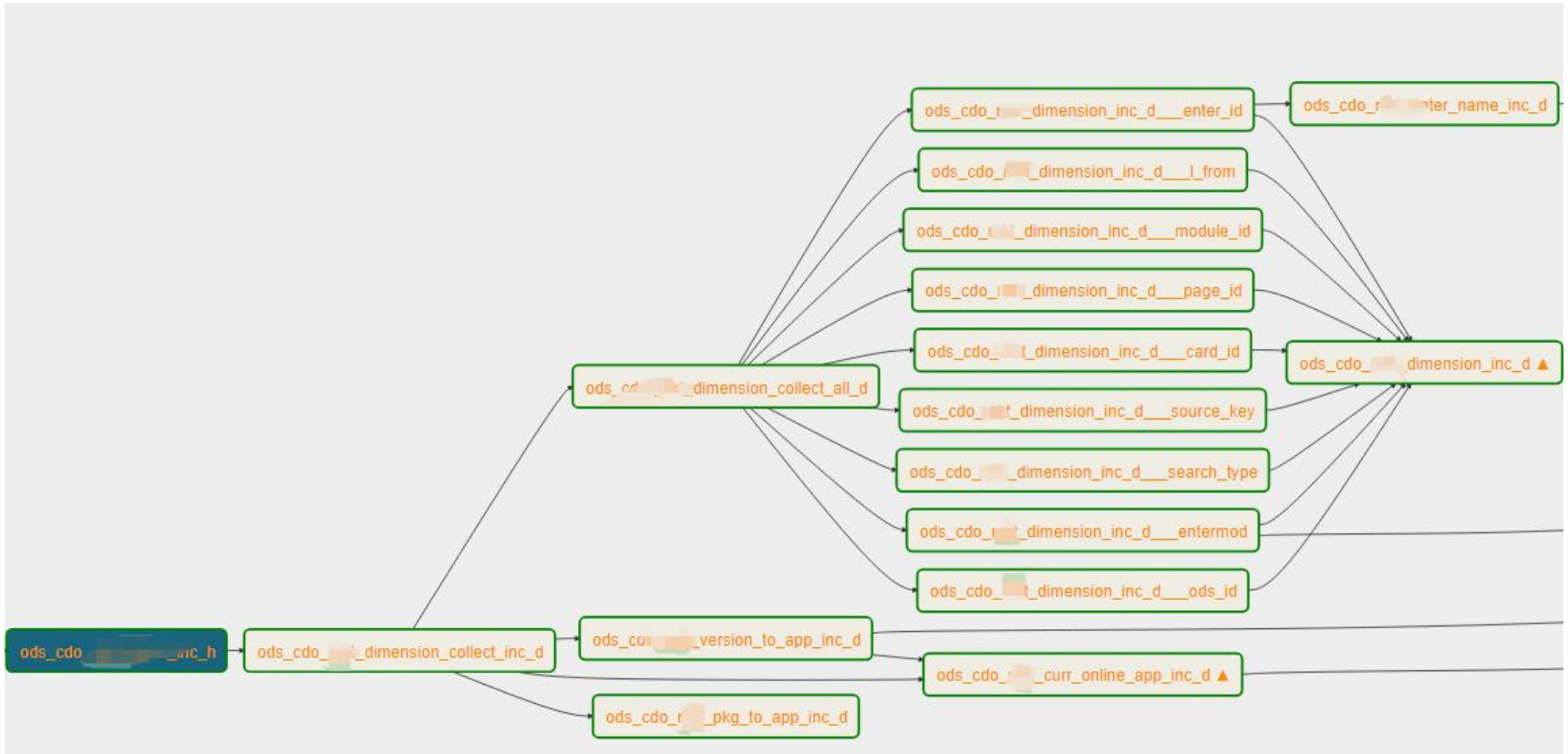
ODS层处理

- 开发原则
  - ✓ 缓存层只保留8天数据，只可以**被一个ODS表引用**
  - ✓ ODS层不可以当作缓存数据使用，直接抽取数据就到ODS
  - ✓ 多**在ODS做join表**，而不是在dwd层做，导致dwd层表变多且通用性降低
  - ✓ 有部分表**不需要进入CDM层**，不需要构建星形模型，则可以在ODS处理直接到应用层

ODS层处理

- 维度穷举实现方法：极致分区表的体现，简洁又省资源





ODS层处理

- 维度穷举的作用
  - ✓ 直接构建维表，在ODS实现，出来的速度非常快
  - ✓ 过滤数据解决数据倾斜问题：

    【表A】 左关联【表B】（表B量大无法广播）ON（c1=c2），改造为：

    【表A】 左关联【表C】（广播）

    【表C】=【表B】内关联【表A.c1的维度穷举表（广播）】

ODS层处理

- 码值翻译
  - ✓ 在ODS层翻译码值，统一全局的口径，不可在应用层处处转换
  - ✓ 命名方式：原字段_name（若原字段以id，code结尾则去掉再拼接，如app_id为app_name）

```
(case
        when q.real_style = '0' then
         '全屏'
        when q.real_style = '1' then
         '半屏'
        when q.real_style = '2' then
         '弹窗'
        when q.real_style = '4' then
         '小半屏'
        else
         q.real_style
      end) real_style_name
```

CDM层处理

- 开发原则
  - ✓ **事实表：**只有需要统一建模才能有cdm，绝不是中间表的概念，是建模的概念（不是为了落一个中间层方便共用逻辑，而是真正全局的构建星形模型的表）
  - ✓ **维表：**不是所有表都有资格成为dim表，很多表没有通过抽象与建模也就是个ods，一定要考虑一致性维度！属于一个维的内容不要有多张表

CDM层处理

- 过滤数据处理
  - ✓ 在模型中过滤数据对模型有巨大伤害，新手非常容易犯此错误，举例：
    - country='CN' --表中所有数据都是CN的，但这个过滤相当于埋了一个雷，随时会爆
    - user_is is not null --看似没用的数据，但可能给你带上枷锁，因为要对数，必须处处带上
    - cheat_flag=0 --作弊数据过滤掉，且不说作弊会误判，此条件将会让你查问题异常困难

过滤数据属于个性化逻辑，应该在应用层中处理，模型中切记不可以做任何过滤

CDM层处理

- 公共逻辑处理
  - ✓ 当一个逻辑在应用层中出现3次及以上，无论这个逻辑多么小，都应该压在CDM层中处理，举例：
    - 解开数组或者jason，['app_id'] --看似极小的逻辑，如果你要对app_id做规则，所有地方都要改
    - 截断字符，substr(str,1,2) --同上，此字段如果在应用层处理，一旦有变化会极其麻烦
    - count(1) --非常简单的逻辑，但此处统一了口径，如果在应用层处处count(1)，维护成本会非常大
    - 更多...

应用层一般只做合并、关联或个性化逻辑处理，不做公共逻辑，此条规则一般人很难做到

CDM层处理

➢ 分区表实现方法

✓ 建表

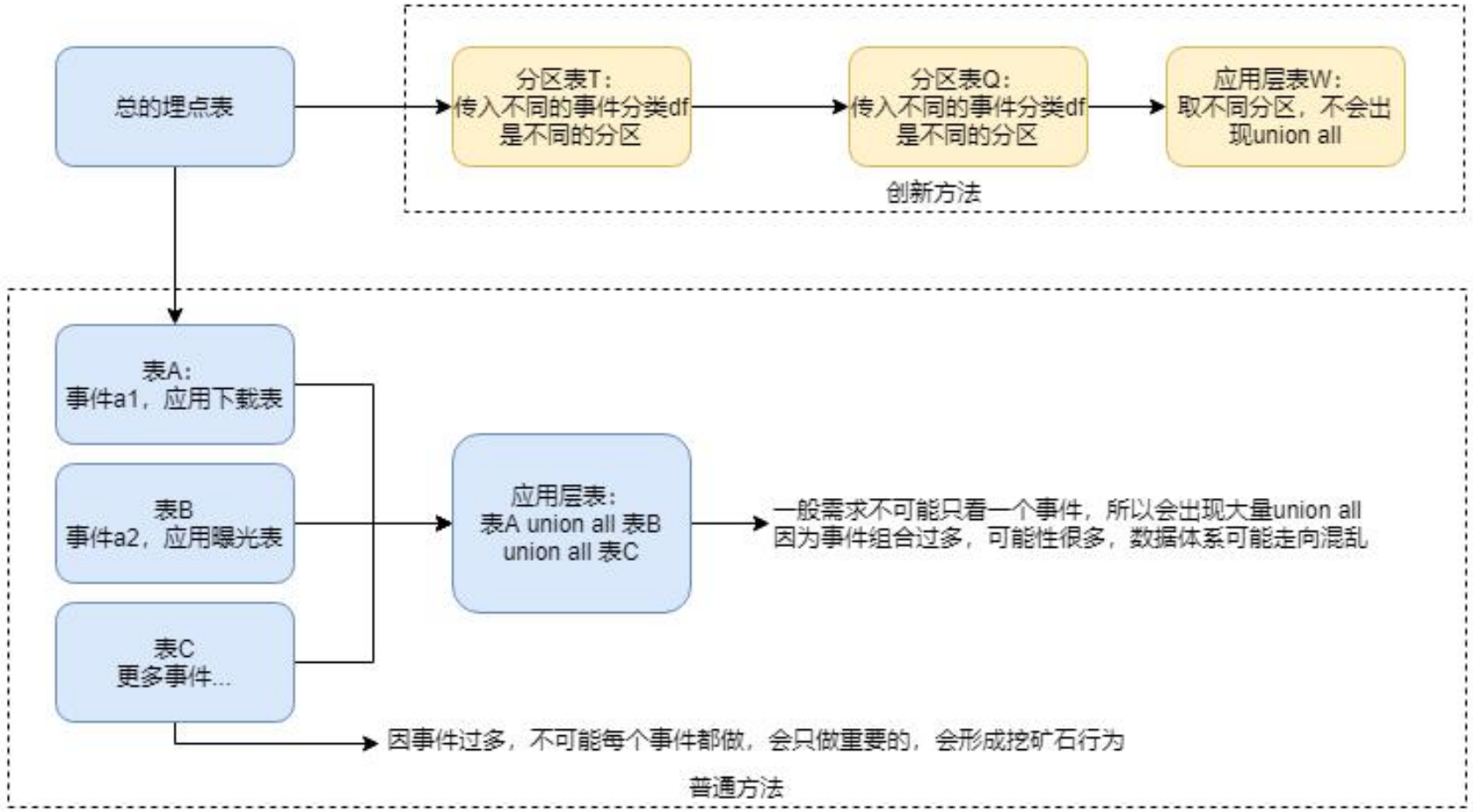
```
`dt` string comment '分区：日期yyyymmdd',
`df` string comment '分区：事件分类'
```

✓ 代码写法

```
category_code="${vi_category_code}"
if [ -z "${category_code}" ];then
 category_code="others"
 condition="and df in ('others')";
else
 condition="and df in ('${category_code}')"
fi
insert overwrite table cdo_xxx_dw.ods_cdo_xxx_xxxx_wide_inc_d
            partition (dt='${v_day}',df='${category_code}')
select *
```

CDM层处理

- 分区表实现方法：极致分区表的体现（注意只是一个主题域下的表）
  - ✓ 创新方法：只有一个表，一个脚本，一套逻辑，多个调度任务

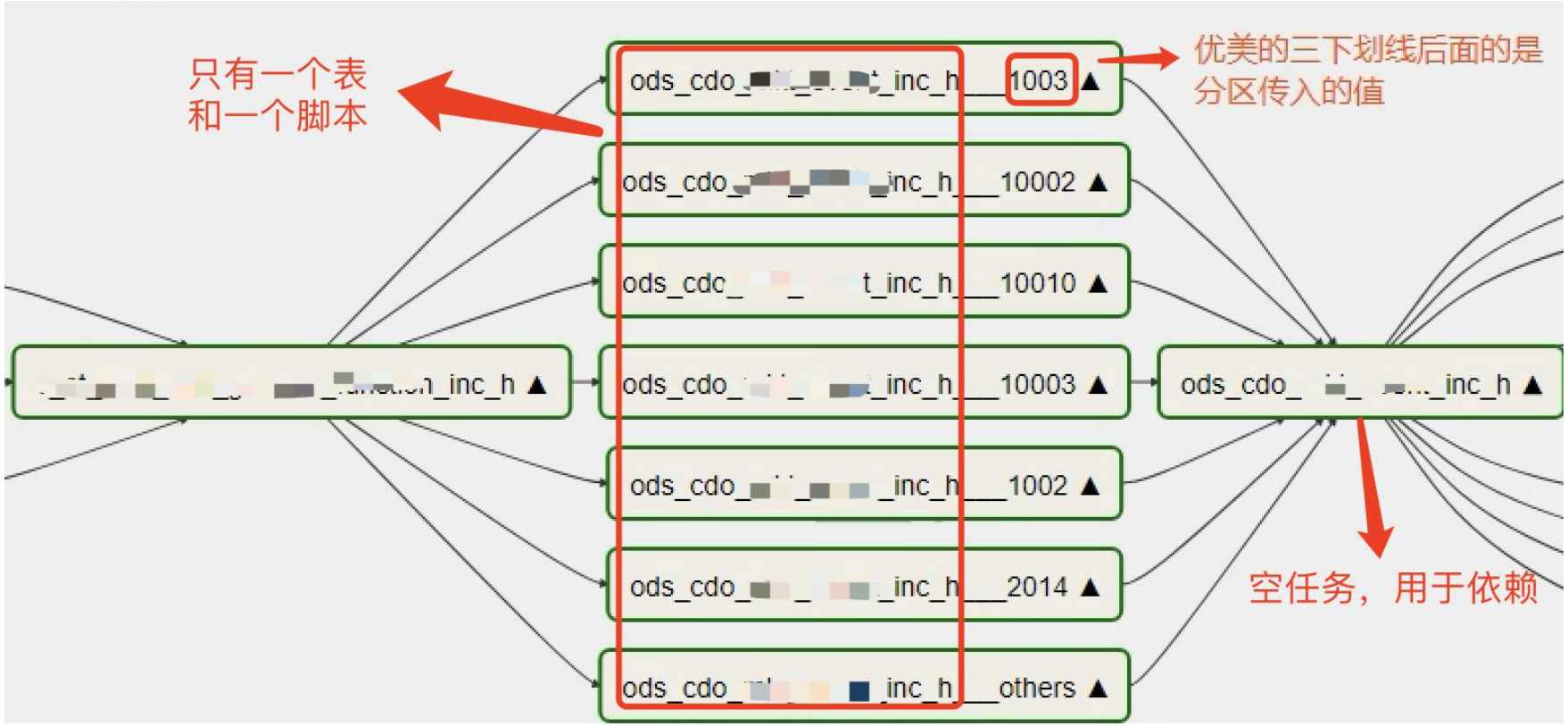


ETL架构

➢ 分区表调度方法

✓ 优点：

- 可维护性好
- 不容易出错
- 无重复代码拷贝
- 分df并行跑，无性能瓶颈
- 表少，脚本少
- 表好用，无大量union
- 只关联一次维表
- 省资源，无重复存储
- 省资源，无重复计算
- 简洁，容易依赖
- 调度无等待，谁先好先跑

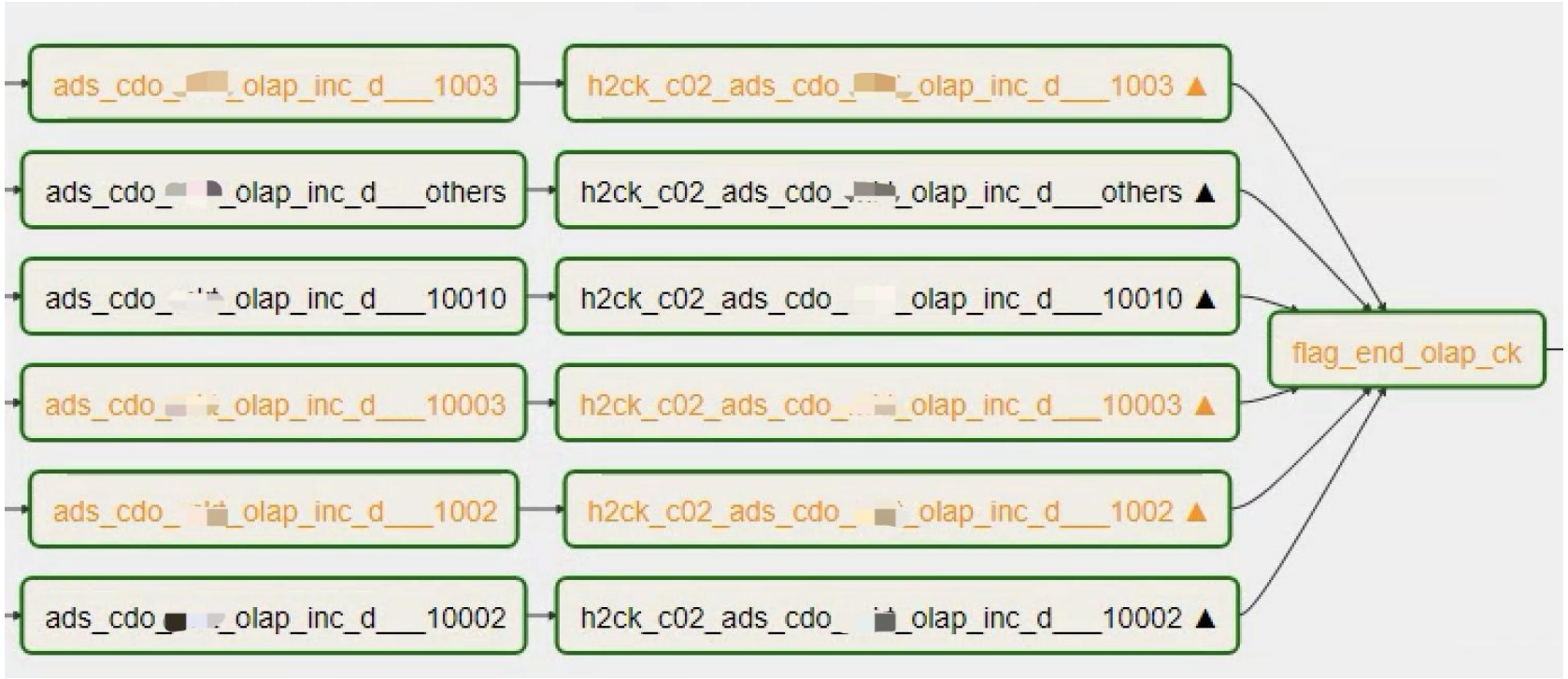


ETL架构

➢ 分区的OLAP报表实现方法

✓ 优点：

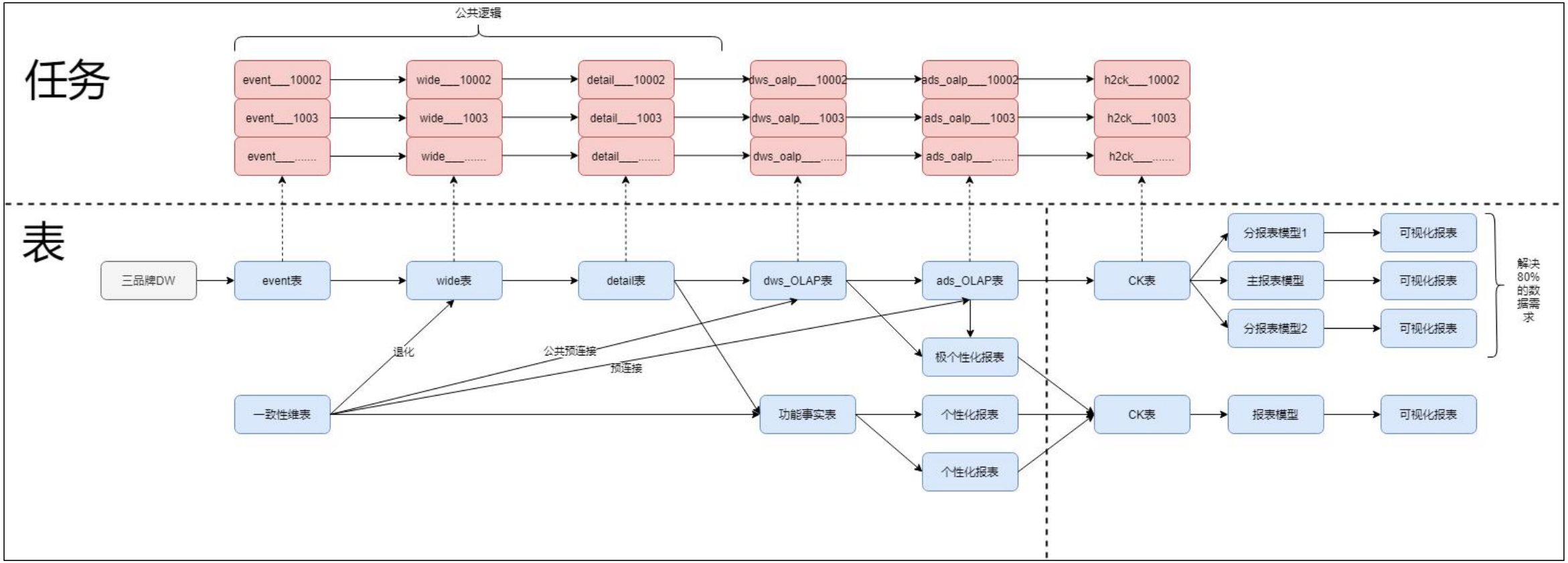
- 只需要维护一张ads层的表
- 推送按分区，分区越多越快
- CK只有一张表，一张报表
- 一张报表配出n张报表
- 报表无等待，谁快可先用



ETL架构

➢ 整体架构

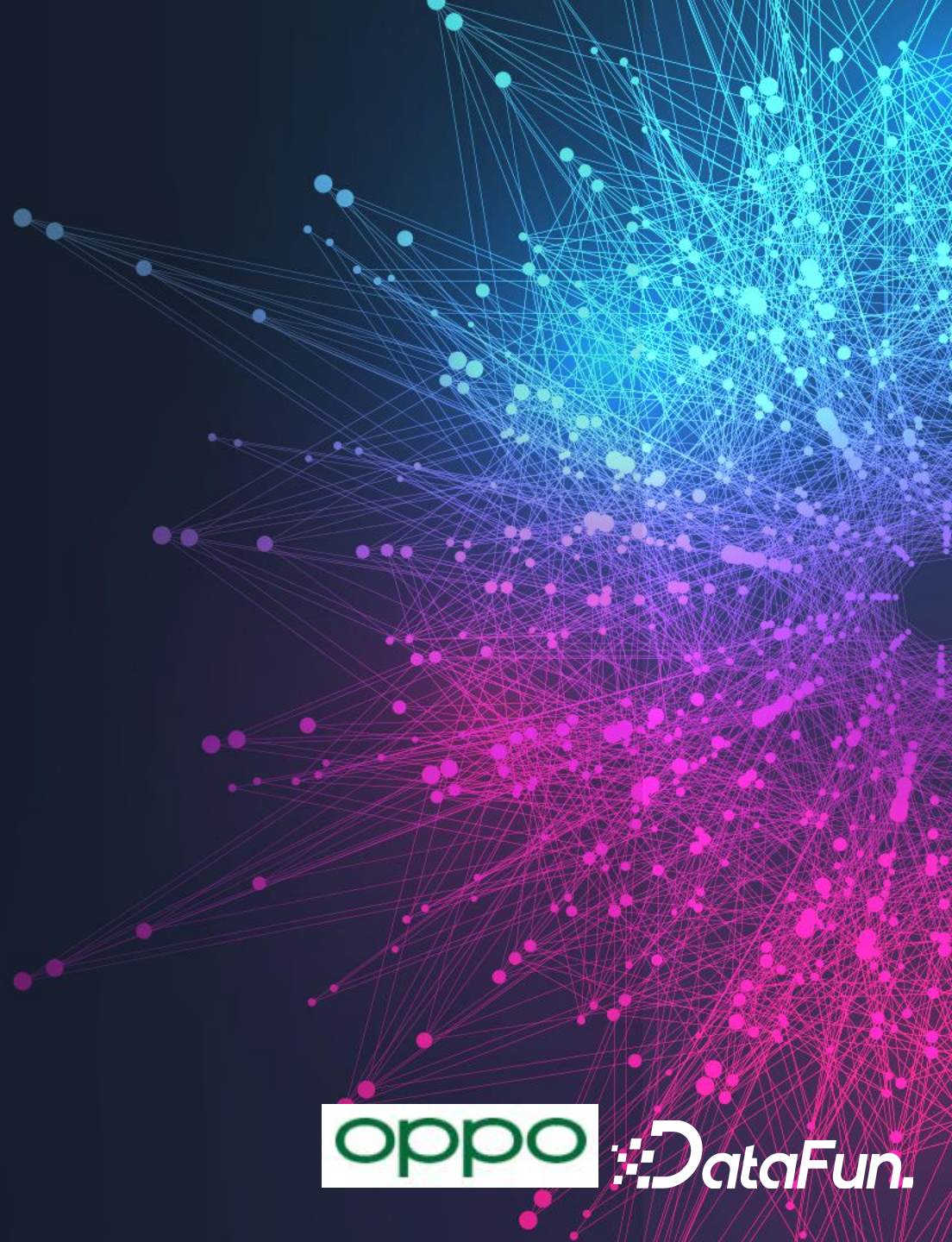
- ✓ 把分区表运用到出神入化，从ods→dwd→dws→ads→h2ck一路**全是分区表**，对于分区表查询，超快超省钱
- ✓ 所有事件都加入建模，且新事件也**自动进到模型**，在后继的CDM层的流程中一起处理公共逻辑
- ✓ 极其优美的三个下划线代表着不同的分区，避免**依赖选择**困难



03

优雅的维度模型设计

DataFunSummit # 2023

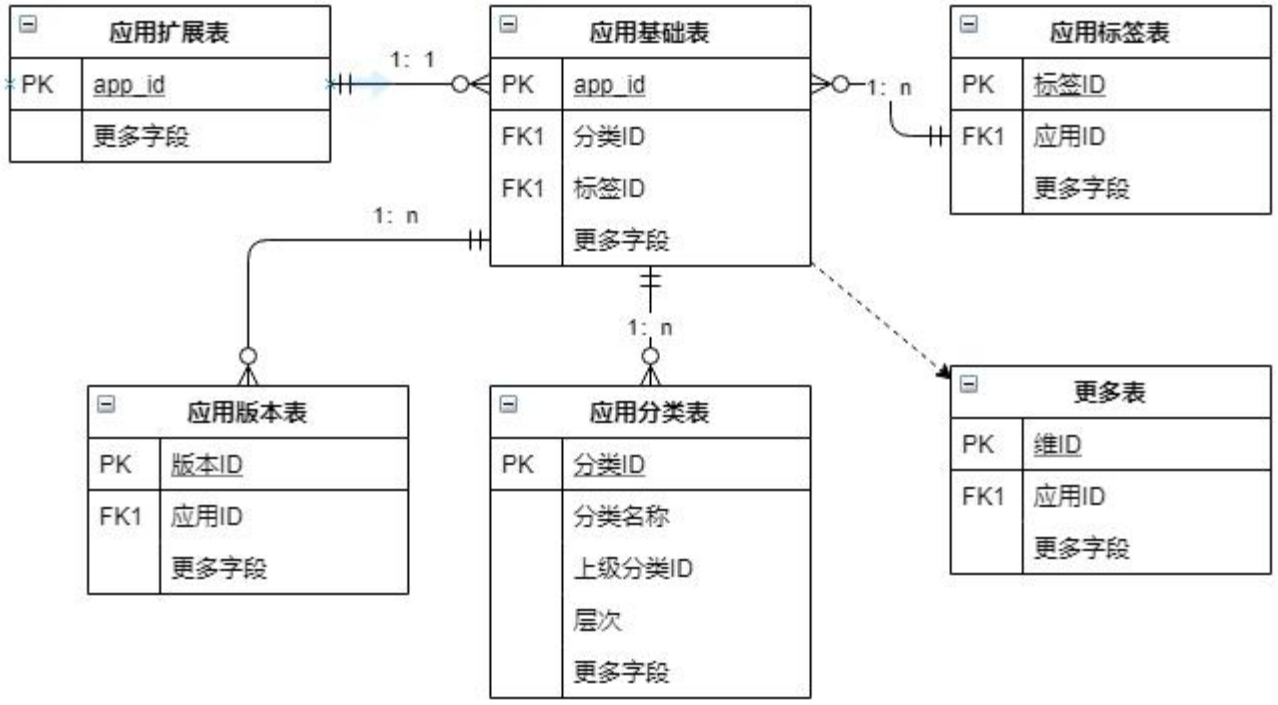


维表设计

- 数据源表的抽象
  - ✓ 抽取涉及维度的**所有表所有字段**，构造这些表的ER关系图，从ER图抽象出维表
  - ✓ 错误示例：
    - 抽一张Mysql表就是一张维表，从不考虑整合，还是ER模型
    - dim_app，dim_app_v2，dim_ap_v99 --同一个维有多张表
    - dim_model，dim_model_detail，--受源系统影响，分段式设计
  - ✓ 抽取原则
    - 抽取这个表的所有字段，不要只挑当前需求要的字段
    - 抽取表，一个表一个脚本一个任务，不可以一个脚本抽取多个表
    - 维表保留历史状态，可以采用快照表的方式保留
    - 抽取完做校验，防止空跑，不然修复成本会非常大

维表设计

- 数据源表的抽象
  - ✓ 尽量做成单一主键的表（但不采用kimball的代理键方式）如下图抽象：
    - 应用维（含应用分类）
    - 应用标签表维（微型维）
    - 应用分类维表（一二三级分类）



维表设计

➢ 维表建设重要原则

- ✓ 维表尽可能宽，不管冗余，哪怕超过1000个字段
  - 码值全映射出来
  - 非结构化数据全解析出来，如数组，json，map，截断
  - 所有自定义分类在此定义，如把应用归为头条系，腾讯系
  - 能在维表沉淀的公共逻辑，不要把逻辑写在事实表中，在维表中统一口径代价更低
- ✓ 强制保证**主键唯一**，直接在维表代码中做row_number处理

明细表设计

- 一个主题下只做一个明细表（分区表）
  - ✓ 以一个APP上报的所有埋点为例：
    - 分区df：一般以事件分类为准，建议不要直接取事件ID，长尾影响数据分布
    - 所有事件理论上全部进明细表，数据量大且冷门的、因成本问题的根据情况来确定

分区：日期dt	分区：数据标识df	事件6+	头部字段100+	业务字段600+	明细map保留
2011-01-01	1	1_1...	...	...	...
2023-11-12	2	1_2...	...	...	...

明细表设计

➢ 维度退化

- ✓ 热门维表的**热门字段**在明细表全部关联到明细表中
- ✓ 哪些维度退化极其考验工程师的设计功底，不可贪多，不能太少
- ✓ 例子说明
  - 明细表中只有应用版本ID；有应用ID、名称；有应用ID、名称、包名；应用维表热门字段；应用维表所有字段，代表不同等级维度退化设计
  - **错误方向：**维度还在map/json中的如['app_id']，无法做集成化处理
  - **错误方向：**明细表中只有id，没有中文名，如city_id 没有city_name，每次使用都去关联
  - **错误方向：**只存应用版本ID，SKU_ID等不常用字段，每次使用都要通过桥接表找应用ID，商品ID

明细表设计

➢ 二维表化设计

- ✓ 明细表是一个标准的二维表，字段有原子性，所有非结构化的、二维表化的做法都会带来麻烦
- ✓ 后端工程师可能更愿意使用json，map，切记不可将这个习惯带到数据领域
- ✓ 需要改造的例子
  - 码表翻译，如字段sex_flag里的值：0，1，2，99，加sex_flag_name字段，存映射中文：男，女，其他，未知
  - map全解开，如[a:1,b:1]应该解开为两个字段a，b
  - json全解开，与map类似，以前听说过一个纯json的数据仓库，所有逻辑用java实现
  - 聚合上报数据，如'a,b,c,d'，根据情况拆解为4条记录

汇总表设计

➢ OLAP表设计

✓ 必要先做一张OLAP的dws表，大量非个性化的指标在这里实现

分区：日期dt	分区：数据标识df	事件6+	头部字段100+	业务字段600+	公共指标50+
2011-01-01	1	1_1...	...	...	...
2023-11-12	2	1_2...	...	...	...

✓ 必要做一张从OLAP的dws表出的ads层表，作为OLAP报表的底层表

分区：日期dt	分区：数据标识df	事件6+	筛选后的维度500+	公共指标50+
2011-01-01	1	1_1...	...	...
2023-11-12	2	1_2...	...	...

汇总表设计

➢ 周期快照事实表设计

- ✓ 周期快照比较常见的是用户画像的实现，就是给用户打标签，实现的逻辑一般比较复杂
- ✓ 用户标签最好抽象成**周期快照事实表**，不要放在原子事实表中，不然有可能造成复用度下降
- ✓ 所有与这个周期快照相关的，最后都应该**合并在一个表中**，不要过于分散（根据产出时间适当调整）

分区：日期dt	用户标识	年龄	性别	是否高价值用户	总游戏时长	更多字段
2011-01-01	1	...	...	...	...	...
2023-11-12	2	...	...	...	...	...

汇总表设计

➢ 留存事实表设计

- ✓ 留存事实表是运营最为常用的事实表之一，一般不会做到OLAP表，形式上是一个梯形
- ✓ 留存事实表是一个错开日期自关联过程，可以抽象为五要素：用户标签（维度）、前行为时间，前行为、后行为时间，后行为（前行为与后行为可相同，后行为时间与指标可合并，如打标：是否1，3，7，14，30天留存）
- ✓ 如果成本允许 ，建议**将用户ID也带到报表中**，可以配置出千变万化的留存看板，如下例子：

分区：日期dt	用户ID	用户标签	前行为时间	前行为指标	后行为时间	后行为指标
2011-01-01	1	如：机型	...	...	...	...
2023-11-12	2	如：机型	...	...	...	...

汇总表设计

➢ 归因事实表设计

- ✓ 归因事实表是运营最为常用的事实表之一，一般不会做到OLAP表，形式上是一个漏斗
- ✓ 比如一个典型归因的过程：启动→曝光→下载→打开→访问页面→下单
- ✓ 归因的各种方法：参数归因、**离线字段匹配归因**（有损）、实时数据流中染色打标
- ✓ 归因事实表非常复杂，计算量巨大，容易产生性能问题，很考验工程师的技术水平
- ✓ 归因路径组合较多，需要与业务充分沟通，选择确定好场景，在dws层**预先计算**出来
- ✓ 合并事实表与归因事实表区别：

04

万能的多维分析模型与报表

DataFunSummit # 2023

DataFun.

Mysql与Clickhouse的区别

- Mysql表的问题
  - ✓ 大数据量报表查询缓慢，使得开发者报表拆得太细做得过多，**程序分散，维护困难，人效低**
  - ✓ Mysql数据库整体容量太小，经常满，经常要清理和迁移，**运维成本大**
  - ✓ 大部分人不会使用分区表，推数粗暴导致推送**主从延迟严重**，被迫读写都使用主库
  - ✓ 与Hadoop集群的Sqoop工具搭配使用时权限管理非常困难，经常报**权限问题**

Mysql与Clickhouse的区别

- Clickhouse解决的问题
  - ✓ 快!
  - ✓ 可搭建空间非常大的集群
  - ✓ 无主从延迟烦恼，无权限烦恼，**吞吐量大**，推送数据快
  - ✓ 99%的报表都可以使用**CK做为载体**，稳定性好，无卡死，锁之类的问题
  - ✓ 天然支持TTL生命周期，从规范上限定所有表都有TTL，**自动实现数据治理**，防止先污染再治理
  - ✓ 支持**近似计算**，使用uniq(imei)替代count(distinct imei)，获取性能优势

多维报表模型

- 多维报表模型

多维分析报表

➢ 多维分析报表

- ✓ 提供一个报表解决80%+的数据获取问题
- ✓ 70%的报表可以从一张报表配置衍生出来，不需要开发代码，不需要发版本，运营也可以实现配置
- ✓ 大量报表的底层表都是OALP表，只需要维护一张表
- ✓ 基于Clickhouse直接自助查询CK表，可以秒出，效率提升几百倍
- ✓ 为平衡成本，保证历史数据的获取，可以制定多版本：

感谢观看