一、数据模型架构规范

1.数据层次的划分

- **ODS**：Operational Data Store，操作数据层，在结构上其与源系统的增量或者全量数据基本保持一致。它相当于DW数据的一个数据准备区，同时又承担着基础数据的记录以及历史变化。其主要作用是把基础数据引入到DMP。

- **CDM**：Common Data Model，公共维度模型层，又细分为DWD和DWS。它的主要作用是完成数据加工与整合，建立一致性的维度，构建可复用的面向分析和统计的明细事实表，以及汇总公共粒度的指标。
  - **DWD**：Data Warehouse Detail，明细数据层。
  - **DWS**：Data Warehouse Summary，汇总数据层。

- **ADS**：Application Data Service，应用数据层。

2.数据划分及命名空间约定

请根据业务划分数据并约定命名，建议针对业务名称结合数据层次约定相关命名的英文缩写，这样可以给后续数据开发过程中，对项目空间、表、字段等命名作为重要参照。

- 按业务划分：命名时按主要的业务划分，以指导物理模型的划分原则、命名原则及使用的ODS project。
- 按数据域划分：命名时按照CDM层的数据进行数据域划分，以便有效地对数据进行管理，以及指导数据表的命名。例如，"用户"数据的英文可定义为'user'。
- 按业务过程划分：当一个数据域由多个业务过程组成时，命名时可以按业务流程划分。业务过程是从数据分析角度看客观存在的或者抽象的业务行为动作。例如，用户行为数据域中的"登录"这个业务过程的英文缩写可约定命名为'user_login'。

3.数据模型

模型是对现实事物的反映和抽象，能帮助我们更好地了解客观世界。数据模型定义了数据之间关系和结构，使得我们可以有规律地获取想要的数据。例如，在一个超市里，商品的布局都有特定的规范，商品摆放的位置是按照消费者的购买习惯以及人流走向进行摆放的。

1）数据模型的作用

数据模型是在业务需求分析之后，数据仓库工作开始时的第一步。良好的数据模型可以帮助我们更好地存储数据，更有效率地获取数据，保证数据间的一致性。

2）模型设计的基本原则

- 高内聚和低耦合：一个逻辑和物理模型由哪些记录和字段组成，应该遵循最基本的软件设计方法论中的高内聚和低耦合原则。主要从数据业务特性和访问特性两个角度来考虑：将业务相近或者相关的数据、粒度相同数据设计为一个逻辑或者物理模型；将高概率同时访问的数据放一起，将低概率同时访问的数据分开存储。
- 核心模型与扩展模型分离：建立核心模型与扩展模型体系，核心模型包括的字段支持常用核心的业务，扩展模型包括的字段支持个性化或是少量应用的需要。在必须让核心模型与扩展模型做关联时，不能让扩展字段过度侵入核心模型，以免破坏了核心模型的架构简洁性与可维护性。
- 公共处理逻辑下沉及单一：底层公用的处理逻辑应该在数据调度依赖的底层进行封装与实现，不要让公用的处理逻辑暴露给应用层实现，不要让公共逻辑在多处同时存在。
- 成本与性能平衡：适当的数据冗余可换取查询和刷新性能，不宜过度冗余与数据复制。
- 数据可回滚：处理逻辑不变，在不同时间多次运行数据的结果需确定不变。
- 一致性：相同的字段在不同表中的字段名必须相同。
- 命名清晰可理解：表命名规范需清晰、一致，表命名需易于下游的理解和使用。

二、公共规范

1.层次调用约定

应用层应优先调用DW公共层数据，必须存在中间层数据，不允许应用层跨过中间层从ODS层重复加工数据。一方面，中间层团队应该积极了解应用层数据的建设需求，将公用的数据沉淀到公共层，为其他团队提供数据服务；另一方面，应用层团队也需积极配合中间层团队进行持续的数据公共建设的改造。必须避免出现过度的ODS层引用、不合理的数据复制以及子集合冗余。

- ODS层数据不能被应用层任务引用，中间层不能有沉淀的ODS层数据，必须通过CDM层的视图访问。CDM层视图必须使用调度程序进行封装，保持视图的可维护性与可管理性。
- CDM层任务的深度不宜过大（建议不超过10层）。

- 原则上一个计算刷新任务只允许一个输出表，特殊情况除外。
- 如果多个任务刷新输出一个表（不同任务插入不同的分区），调度上需要建立一个依赖多个刷新任务的虚拟任务，通常下游应该依赖此虚拟任务。
- CDM汇总层应优先调用CDM明细层。在调用可累加类指标计算时，CDM汇总层尽量优先调用已经产出的粗粒度汇总层，以避免大量汇总都直接从海量的明细数据层计算。
- CDM明细层累计快照事实表优先调用CDM事务型事实表，以保持数据的一致性产出。
- 避免应用层过度引用和依赖CDM层明细数据，需有针对性地建设好CDM公共汇总层。

2.数据类型规范

ODS层的数据类型应基于源系统数据类型转换。如源数据为MySQL时的转换规则如下：

MySQL数据类型	DMP数据类型
TINYINT	TINYINT
SMALLINT/MEDIUMINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL	DECIMAL
CHAR/VARCHAR	STRING
LONGTEXT/TEXT	STRING
DATE/TIMESTAMP/TIME/YEAR	STRING
DATETIME	DATETIME

CDM数据公共层如果是引用ODS层数据，则默认使用ODS层字段的数据类型。其衍生加工数据字段按以下标准执行：

- 金额类及其它小数点数据使用DOUBLE类型。
- 字符类数据使用STRING类型。
- ID类和整形数值使用BIGINT类型。
- 时间类型数据使用STRING类型（如果有特殊的格式强制要求，可以选择性使用DATETIME类型）。
- 状态使用STRING类型。

3.公共字段定义规范

数据统计日期的分区字段按以下标准：

- 按天分区：dt(YYYYMMDD)
- 按小时分区：hh(00-23)
- 按分钟：mi (00-59)
- is_{业务}：表示布尔型数据字段。"Y","N"表示，不允许出现空值域。
- 原则上不需要冗余分区字段。

4.数据冗余

1）相关数据冗余

一个表做宽表冗余维度属性时，应该遵循以下建议准则：

- 冗余字段与表中其它字段高频率（大于3个下游应用SQL）同时访问。
- 冗余字段的引入不应造成其本身的刷新完成时间产生过多后延。
- 公共层数据不允许字段重复率大于60%的相同粒度数据表冗余，可以选择在原表基础上拓宽或者在下游应用中通过JOIN方式实现。

2）子集合冗余

当需要从一个集合中冗余一部分记录作为另外一张表存在时，可以优先考虑子分区方式，但多级子分区不应超过（5级）。只有以下情况才考虑冗余：

- 子类型表有较多（大于10）个字段，而父类型表并不存在。
- 子集合的过滤条件会被多次（大于5次）应用。

5.数据拆分

数据的水平和垂直拆分是按照访问热度分布和数据表"非空或者0值"的数据值在行列二维空间上分布情况进行划分的。

- 在物理上划分核心模型和扩展模型，将其字段进行垂直划分。

- 将访问相关度较高的列在一个表存储，将访问相关度较低的字段分开存储。
- 将经常用到的where条件按记录行进行水平切分或者冗余；水平切分可以考虑二级分区手段，以避免多余的数据复制与冗余。
- 将出现大量空值和零值的统计汇总表，依据其空值和零值分布状况可以做适当的水平和垂直切分，以减少存储和下游的扫描数据量。

6.空值处理原则

- 汇总类指标的空值：空值处理，填充为零，当前DMP基于列存储的压缩技术不会由于填充大量空值导致存储成本上升。
- 维度属性值为空：在汇总到对应维度上时，对于无法对应的统计事实，记录行会填充为-99（未知），对应维表会出现一条-99（未知）的记录。

7.数据仓库中表格的命名规范

如下表所示：

数据层次	****	周期/数据范围	****
公用维度	dim	日快照	d
dm层	dm	增量	i
ods层	ods	周	w
dwd层	dwd	拉链	l
dws层	dws	非分区全量	f

三、数据存储及生命周期管理规范

数据表类型	存储方式	最长存储保留策略
ODS流水型全量表	按天分区	不可再生情况下永久保存。日志（非常大:如一天数据大于100G）数据保留24个月。自主设置是否保留历史月初数据。自主设置是否保留特殊日期数据。
ODS镜像型全量表	按天分区	重要的业务表及需要保留历史的表视情况保存。ODS全量表的默认生命周期管理为保留2天，并采用ds=max_pt(tablename)方式进行数据访问。
ODS增量表	按天分区	有对应的全量表则最多保留最近14天分区数据。无全量表的增量数据需永久保留。
ODS ETL过程临时表	按天分区	最多保留最近7天分区
BDSync非去重数据	按天分区	由应用通过中间层做保留历史处理，默认ODS层不保留历史。

数据质量规范

- 每个ODS全量表必须配置唯一性字段标识。
- 每个ODS全量表必须做分区空数据监控。
- 建议对重要表的重要枚举类型字段做枚举值变化及枚举值分布监控。
- 建议对ODS表的数据量及数据记录数设置上周同比无变化监控，用于监控源系统是否下线或者已迁移。
- 只有有监控要求的表才创建数据质量管控层，应由DMP的数据质量配置完成。
- 每个ODS层全表都必须要有注释。

四、CDM公共维度层设计规范

1.设计准则

1）一致性维度规范

公共层的维度表中相同维度属性在不同物理表中的字段名称、数据类型、数据内容必须保持一致。除了以下特例：

- 在不同的实际物理表中，如果由于维度角色的差异，需要使用其他的名称，其名称也必须是规范的维度属性的别名。比如：定义一个标准的用户ID时，如果在一个表中，分别要表示正式用户ID，访客ID，那么设计规范阶段就预先对用户ID分别定义正式用户ID和访客ID。
- 如果由于历史原因，在暂时不一致的情况下，必须在规范的维度定义一个标准维度属性，不同的物理名也必须是来自标准维度属性的别名。

2）维度的组合与拆分

①组合原则

- 将维度所描述业务相关性强的字段在一个物理维表实现，相关性一般指：经常需要一起查询、报表展现，比如商品基本属性和所属品牌；两个维度属性间是否存在天然的关系等。
- 无相关性的维度可以适当考虑杂项维度，比如交易，可以构建一个交易杂项维度收集交易的特殊标记属性、业务分类等信息。也可以将杂项维度退化在事实表中处理，不过容易造成事实表相对庞大，加工处理较为复杂。
- 所谓的行为维度是经过汇总计算的指标，在下游的应用使用时将其当维度处理。如果有需要，度量指标可以作为行为维度冗余到维度表中。

②拆分与冗余

- 对于维度属性过多，涉及源较多的维度表，可以做适当拆分。
  - 比如用户表，建议拆分为核心表和扩展表。核心表相对字段较少，刷新产出时间较早，优先使用。扩展表字段较多，且可以冗余核心表部分字段，刷新产出时间较晚，适合数据分析人员使用。
  - 根据维度属性的业务不相关性，将相关度不大的维度属性拆分为多个物理表存储。

- 数据记录数较大的维度表，可以适当冗余一些子集合，以减少下游扫描数据量：
  - 比如用户表，可以根据当天是否有行为，产出一个有活跃行为的相关维表，以减少应用的数据扫描量。
  - 可根据所属业务扫描数据范围大小的不同，进行适当子集合冗余。

2.表命名规范

命名规则：{project_name}.dim{业务/pub}{维度定义}[_{自定义命名标签}]，所谓的pub是类似与具体业务无关，各个业务部都可以共用，例如时间维度。

3.数据存储及生命周期管理规范

CDM公共维度层的表的类型为维度表，存储方式为按天分区。

最长存储保留策略：

模型设计者根据自身业务需求设置表的生命周期管理。您可依据3个月内的最大需要访问的跨度设置保留策略，具

体计算方式如下。

- 当3个月内的最大访问跨度小于或等于4天时，建议将保留天数设为7天。
- 当3个月内的最大访问跨度小于或等于12天时，建议将保留天数设为15天。
- 当3个月内的最大访问跨度小于或等于30天时， 建议将保留天数设为33天。
- 当3个月内的最大访问跨度小于或等于90天时，建议将保留天数设为93天。
- 当3个月内的最大访问跨度小于或等于180天时， 建议将保留天数设为183天。
- 当3个月内的最大访问跨度小于或等于365天时，建议将保留天数设为368天。

五、CDM明细层设计规范

1.表命名规范

命名规则：{project_name}.dwd{业务缩写/pub}{数据域缩写}{业务过程缩写}[{自定义表命名标签缩写}]{刷新周期标识}{单分区增量全量标识}。

说明：

- pub表示数据包括多个业务的数据。
- 单分区增量全量标识：i表示增量，f表示全量。

2.数据存储及生命周期管理规范

CDM明细层的表的类型为事实表，存储方式为按天分区。

事务型事实表一般永久保存。周期性快照事实表根据业务需求设置生命周期管理。您可依据3个月内的最大需要访问的跨度设置保留策略，具体计算方式如下：

- 当3个月内的最大访问跨度小于或等于4天时，建议将保留天数设为7天。
- 当3个月内的最大访问跨度小于或等于12天时，建议将保留天数设为15天。
- 当3个月内的最大访问跨度小于或等于30天时， 建议将保留天数设为33天。
- 当3个月内的最大访问跨度小于或等于90天时，建议将保留天数设为93天。
- 当3个月内的最大访问跨度小于或等于180天时， 建议将保留天数设为183天。
- 当3个月内的最大访问跨度小于或等于365天时，建议将保留天数设为368天。

3.事务型事实表设计准则

事务型事实表主要用于分析行为与追踪事件。事务事实表获取业务过程中的事件或者行为细节，然后通过事实与维度之间关联，可以非常方便地统计各种事件相关的度量，比如浏览UV，搜索次数等等。

- 基于数据应用需求的分析设计事务型事实表，如果下游存在较大的针对某个业务过程事件的分析指标需求，可以考虑基于某一个事件过程构建事务型事实表。
- 事务型事实表一般选用事件发生日期或时间作为分区字段，这种分区方式可以方便下游的作业数据扫描执行分区裁剪。
- 明细层事实表的冗余子集的原则能有利于降低上层数据访问的IO开销。
- 明细层事实表维度退化到事实表原则能有利于减少上层数据访问的JOIN成本。

4.周期快照型事实表

周期快照型事实表主要用于分析状态型或者存量型事实，快照是指以预定的时间间隔来采样状态度量。

5.累计快照事实表

累计快照事实表是基于多个业务过程联合分析从而构建的事实表，如采购单的流转环节等。

累计快照事实表主要用于分析事件之间的时间间隔与周期，比如用交易的支付与发货之间的间隔，来分析发货速度，或在支付和退款环节分析支付退款率等等；同时也可以用于帮助分析一些少量的、且对刷新时间不是非常敏感的指标统计，比如，在当前事务型事实表不支持，且只有少量的统计指标时，需分析交易的关闭和发货，就可以基于累计快照事实表进行计算。

六、CDM汇总层设计规范

1.命名规范

命名规则：{project_name}.dws{业务缩写/pub}{数据域缩写}{数据粒度缩写}[{自定义表命名标签缩写}]{统计时间周期范围缩写}{刷新周期标识}{单分区增量全量标识}。

说明：

- 关于统计时间周期范围缩写，在缺省情况下，离线计算应该包括最近一天（1d）,最近N天（nd）和历史截至当天(td)三个表，如果出现nd的表的字段过多，需要拆分时，只允许以一个统计周期单元作为原子拆分，即一个统计周期拆分一个表，比如最近7天(_1w)拆分一个表；不允许拆分出来的一个表存储多个统计周期的。
- 对于{刷新周期标识}和{单分区增量全量标识}在汇总层不做强制要求。单分区增量全量标识：i：表示增量，f表示全量。
- 对于小时表不管是按天刷新还是按小时刷新, 都用_hh 来表示。
- 对于分钟表不管是按天刷新还是按小时刷新,都用_mm来表示。

2.数据存储及生命周期管理规范

CDM汇总层的表的类型为事实表，存储方式为按天分区。

事务型事实表一般永久保存。周期性快照事实表根据业务需求设置生命周期管理。您可依据3个月内的最大需要访问的跨度设置保留策略，具体计算方式如下：

- 当3个月内的最大访问跨度小于或等于4天时，建议将保留天数设为7天。
- 当3个月内的最大访问跨度小于或等于12天时，建议将保留天数设为15天。
- 当3个月内的最大访问跨度小于或等于30天时， 建议将保留天数设为33天。
- 当3个月内的最大访问跨度小于或等于90天时，建议将保留天数设为93天。
- 当3个月内的最大访问跨度小于或等于180天时， 建议将保留天数设为183天。
- 当3个月内的最大访问跨度小于或等于365天时，建议将保留天数设为368天。

七、表设计规范

DMP中不同类型计算任务的操作对象（输入、输出）都是表。表设计是否合理将影响存储和计算的性能，进而影响到存储和计算的计费。

1.表设计主要目标

1）降低存储成本

合理的表设计可以降低数据分层设计上的冗余存储，减少中间表的数据量大小。对表数据的生命周期进行正确地管理，也能够直接降低存储的数据量及存储成本。

2)降低计算成本

规范化的表设计可以帮助使用者优化数据的读取，从而减少计算过程中的冗余读写和计算，提升计算性能，降低计算成本。

3)降低维护复杂度

规范化的表分层设计能够直接体现业务的特点。例如，在规范化设计表的同时对数据通道中的数据采集方式进行优化，可以减少分布式系统中小文件的问题，降低表和分区维护的数量等复杂度。

2.表设计的影响

1）表设计所影响的操作：表创建、入数据、表更新、表删除、表管理。

2）导入数据场景（区分要做实时数据采集还是离线批量数据写入）：

- 导入即查询与计算
- 多次导入，定时查询与计算
- 导入后生成中间表进行计算

说明：

- 合理的表设计和数据集成周期管理能够降低数据在存储期间的成本。
- DMP优先计算批量数据集成库并按业务逻辑进行计算，例如按照分区进行计算。
- 导入后立即查询与计算，需要考虑每次导入的数据量，减少流式小量数据导入。
- 不合理的数据导入及存储（小文件）会影响整体的存储性能、计算性能、运维稳定性。

3.表设计步骤

- 确定所属项目空间，依据业务过程规划表类型，分析数据层次。
- 定义表描述，进行权限定义与Owner定义。
- 依据数据量、数据集成特点定义分区表或非分区表。
- 定义字段或分区字段。
- 进行表创建、表转换。
- 明确导入数据场景的相关因素（包括批量数据写入、流式数据写入、条式数据插入）。
- 定义表和分区数据生命周期。

说明：

- 创建完表后，您可以依据业务变化修改表的schema，例如设置生命周期：RangeClustering。
- 在表设计阶段，需要特别注意区分数据的场景（批量数据写入、流式数据写入、周期性条式数据插入）。
- 合理使用非分区表和分区表。建议采用分区表来设计日志表、事实表，原始采集表等，并按照时间进行分区。
- 注意各种表和分区的限制条件。

4.表数据存储规范

1)按数据分层规范数据的生命周期

- 源表**ODS**层：每天从业务系统同步过来的数据，全部保留，生命周期定义永久保存。当下游数据受损时，可以从ODS恢复数据。若ODS每天同步过来的是全量表，则可以通过全表拉链的方式来压缩存储。
- 数据仓库（基础）层：　至少保留一份完整的全量数据（不必像ODS那样存储冗余的全量表）。您可以通过拆表或者做分区来提升性能。
- 数据集市层：数据将被按需保留1~3年。数据集市的数据比较容易生成，所以无需保留久远的历史数据。

2)按数据的变更和历史规范数据的保存

- 客户属性、产品属性不断在变更。将这些属性的历史变化情况记录下来，以便追溯某个时点的值。
- 在事实表里冗余维表的字段，即把"事件发生时"的各种维度属性值与该事件绑定起来。使用者无需关联多张表就可以使用数据。此方式仅可应用于数据应用层。
- 用拉链表或者日快照的形式，记录维表的变化情况。这使得数据结构变得灵活、易于扩展，数据一致性得到了增强，数据加工者可以更加方便地管理数据。此方式仅可应用于数据基础层。

5.数据导入通道与表设计

通道类型：

- **Datahub**：规划写入的分区与写入流量之间的关系，做到每64M进行一次commit。
- 数据集成或**DataX**：规划写入的表分区的频率，做到每64M进行一次commit，以免commit空目录。
- **DTS**：规划写入的表存量分区与增量分区的关系，设置commit频率。
- **Console**（**Run SQL or Tunnel upload**）：需要避免高频小数据量文件的插入或者上传。
- **SDK**执行**SQL**的**insert into**语句：对表或者分区上传时需要注意在插入到分区后使用merge语句进行小文件整理操作，以免对一个分区或者非分区表插入多次。

说明：

- DMP导入数据的通道只能是Tunnel SDK或执行SQL的insert into语句，请避免流式插入。
- 以上各通道本身均由自身逻辑进行流式数据写入、批量数据写入、周期调度写入。
- 当使用数据通道写入表或分区时，需将一次写入的数据量控制在合理范围，例如64M以上。

6.分区设计与逻辑存储的对应

一张表里有很多个一级分区，每个一级分区都会按时间存储二级分区，每个二级分区都会存储所有的列。

- 请设置分区的数量上限。
- 请避免每个分区中只存少量数据。
- 分区的条件设置应以方便数据的查询和计算为前提。
- 避免每个分区中出现多次的数据写入。

7.表和分区设计基本规则

1）所有的表、字段名要使用统一的命名规范：

- 命名应能区分该表的业务类型。
- 命名应能区分该表是"事实表"或"维度表"、"日志表"、"极限存储表"（待发布的功能）。
- 命名应能区分该表的实体信息。

2）不同表中具有相同业务含义的字段要定义成统一的数据类型，避免不必要的类型转换。

3）分区设计及使用规则：

- 支持新增分区，不支持新增分区字段。
- 单表支持分区数量为6万。
- 对于多级分区的表，如果想添加新的分区，则必须指明全部的分区值。
- 不支持修改分区列的列名，只能修改分区列对应的值。修改多级分区的一个或者多个分区值时，多级分区的每一级的分区值都必须写上。

8.分区设计

1）分区字段和普通字段的选择

通过分区字段，您可以划分数据扫描范围，更加方便地管理数据。

您在可以在创建表时设置普通字段和分区字段。通常，普通字段可以被理解为数据文件的数据，而分区字段可以被理解为文件系统的目录。表的存储空间的占用主要是普通字段的空间占用。

分区列虽不直接存储数据，但如同文件系统里的目录，可以方便您管理数据。例如，在计算时若指定具体的分区，则计算过程中只需查询对应分区，从而减少计算输入量。

分区表的分区列的级数不能超过6级，即底层存储数据的目录层数不能超过6层。您应为分区表设置合适的生命周期。当部分数据的生命周期与其它数据不同时，您可以通过细粒度分区实现对部分数据的管理。

说明：

- 设置分区字段时，您可以从数据管理和常用的数据扫描方面考虑，来选择对应的字段。
- 不具备规律或类型数量大于10000且不经常作为查询条件的字段，应被设置成普通字段。
- 分区字段定义依据

2）按优先级高低排序

- 分区列的选择应充分考虑时间因素，尽量避免对于存量分区进行更新。
- 如果有多个事实表（不包括维度表）进行join，应将查询条件where范围的列作为分区列。
- 选择group by或distinct包含的列作为分区列。
- 选择值分布均匀的列，而不要选择分区倾斜的列作为分区列。
- 常用SQL语句中若经常包含某列的等值或in的查询条件，则选择该列作为分区列。例如：

```
select … from** table where id =123 and** ….;
```

3）分区个数定义依据

- 时间分区：建议按天或月进行分区。如果按小时进行分区，则二级分区的平均数量不应超过8个。
- 地域分区：若对省、市、县进行分区，则应考虑进行多级分区。23个省，5个自治区，4个直辖市，2个特别行政区，50个地区（州、盟），661个市（其中直辖市4个、地级市283个、县级市374个），1636个县（自治县、旗、自治旗、特区和林区），按照最细粒度县进行分区后，不应再按照更细粒度小时进行分区。

- 单分区与多级分区：在单分区下，建议每次提交64M数据。如果为多级分区，则需保证每个最细粒度级分区下的二级分区的数据都遵循单分区个数规则。
- 单表分区：单表分区数（包括下级分区）不能超过6万。

3）分区数量和数据量建议

- 在计算的时候可以使用分区裁剪是分区的优势。
- 建议单个分区中数据量不要太大。
- 应尽量避免分区数据倾斜，避免单个表不同分区的数据量差异超过100万。
- 做分区设计时应合理规划分区个数，较细粒度的分区在跨分区扫描时会影响到SQL的执行性能。
- 单个分区中数据量较大的情况下，DMP执行任务时会做分片处理不影响分区裁剪的优势。
- 单个分区中文件数较多时，会影响DMP Instance数量，造成资源浪费和SQL性能的影响。
- 采用多级分区，先按日期分区，然后按交易类型分区。
- 拆表，一种交易类型独立成一张表，然后每张表按日期分区。
- 维度表不做分区。

作者丨白枭
来源丨https://blog.csdn.net/github_36444580/article/details/117921306