数仓建设规范指南

V 1.0

本文档来自公众号：**五分钟学大数据**

微信扫码关注

目录

一、数据模型架构原则

1. 数仓分层原则

优秀可靠的数仓体系，往往需要清晰的数据分层结构，即要保证数据层的稳定又要屏蔽对下游的影响，并且要避免链路过长。那么问题来了，一直在讲数仓要分层，那数仓分几层最好？

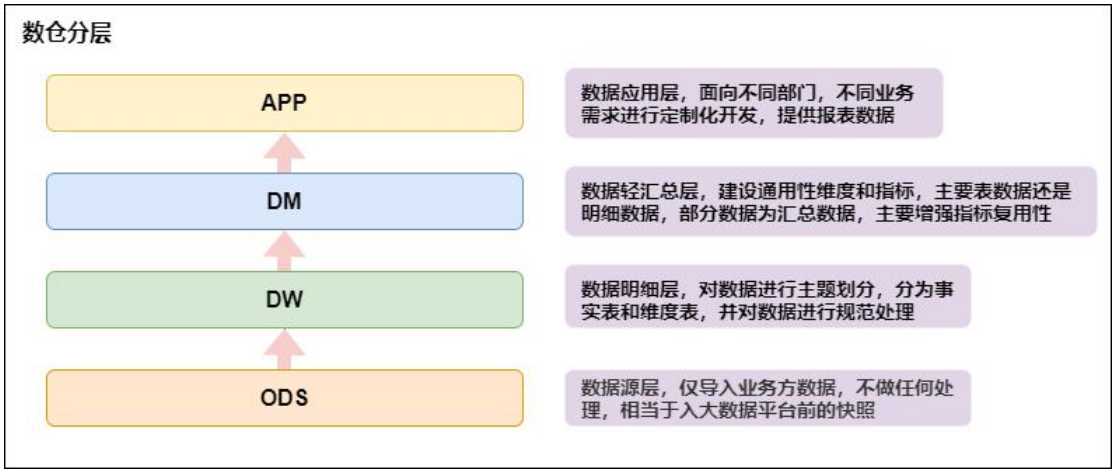
目前市场上主流的分层方式眼花缭乱，不过看事情不能只看表面，还要看到内在的规律，**不能为了分层而分层，没有最好的，只有适合的**。

分层是以解决当前业务快速的数据支撑为目的，为未来抽象出共性的框架并能够赋能给其他业务线，同时为业务发展提供稳定、准确的数据支撑，并能够按照已有的模型为新业务发展提供方向，也就是数据驱动和赋能。

一个好的分层架构，要有以下好处：

1. 清晰数据结构；
2. 数据血缘追踪；
3. 减少重复开发；
4. 数据关系条理化；
5. 屏蔽原始数据的影响。

数仓分层要结合公司业务进行，并且需要清晰明确各层职责，一般采用如下分层结构：



数据分层架构

数仓建模在哪层建设呢？我们以维度建模为例，**建模是在数据源层的下一层进行建设**，在上图中，就是在 DW 层进行数仓建模，所以 **DW 层是数仓建设的核心层**。

下面详细阐述下每层建设规范，和上图的分层稍微有些区别：

1. 数据源层：ODS（Operational Data Store）

ODS 层，是最接近数据源中数据的一层，为了考虑后续可能需要追溯数据问题，因此对于这一层就不建议做过多的数据清洗工作，原封不动地接入原始数据即可，至于数据的去噪、去重、异常值处理等过程可以放在后面的 DWD 层来做。

2. 数据仓库层：DW（Data Warehouse）

数据仓库层是我们在做数据仓库时要核心设计的一层，在这里，从 ODS 层中获得的数据按照主题建立各种数据模型。
DW 层又细分为 DWD（Data Warehouse Detail）层、DWM（Data WareHouse Middle）层和 DWS（Data WareHouse Servce） 层。

1）数据明细层：DWD（Data Warehouse Detail）

该层一般保持和 ODS 层一样的数据粒度，并且提供一定的数据质量保证。**DWD 层要做的就是将数据清理、整合、规范化、脏数据、垃圾数据、规范不一致的、状态定义不一致的、命名不规范的数据都会被处理**。
同时，为了提高数据明细层的易用性，**该层会采用一些维度退化手法，将维度退化至事实表中，减少事实表和维表的关联**。
另外，在该层也会做一部分的数据聚合，将相同主题的数据汇集到一张表中，提高数据的可用性 。

2）数据中间层：DWM（Data WareHouse Middle）

该层会在 DWD 层的数据基础上，数据做轻度的聚合操作，生成一系列的中间表，提升公共指标的复用性，减少重复加工。
直观来讲，**就是对通用的核心维度进行聚合操作，算出相应的统计指标**。
在实际计算中，如果直接从 DWD 或者 ODS 计算出宽表的统计指标，会存在计算量太大并且维度太少的问题，因此一般的做法是，在 DWM 层先计算出多个小的中间表，然后再拼接成一张 DWS 的宽表。由于宽和窄的界限不易界定，也可以去掉 DWM 这一层，只留 DWS 层，将所有的数据再放在 DWS 亦可。

3）数据服务层：DWS（Data WareHouse Servce）

DWS 层为公共汇总层，会进行轻度汇总，粒度比明细数据稍粗，基于 DWD 层上的基础数据，**整合汇总成分析某一个主题域的服务数据，一般是宽表**。DWS 层应覆盖 80% 的应用场景。又称数据集市或宽表。

按照业务划分，如主题域流量、订单、用户等，生成字段比较多的宽表，用于提供后续的业务查询，OLAP 分析，数据分发等。

一般来讲，该层的数据表会相对比较少，一张表会涵盖比较多的业务内容，由于其字段较多，因此一般也会称该层的表为宽表。

3. 数据应用层：APP（Application）

在这里，主要是提供给数据产品和数据分析使用的数据，一般会存放在 ES、PostgreSql、Redis 等系统中供线上系统使用，也可能会存在 Hive 或者 Druid 中供数据分析和数据挖掘使用。比如我们经常说的报表数据，一般就放在这里。

4. 维表层（Dimension）

如果维表过多，也可针对维表设计单独一层，维表层主要包含两部分数据：

高基数维度数据：一般是用户资料表、商品资料表类似的资料表。数据量可能是千万级或者上亿级别。

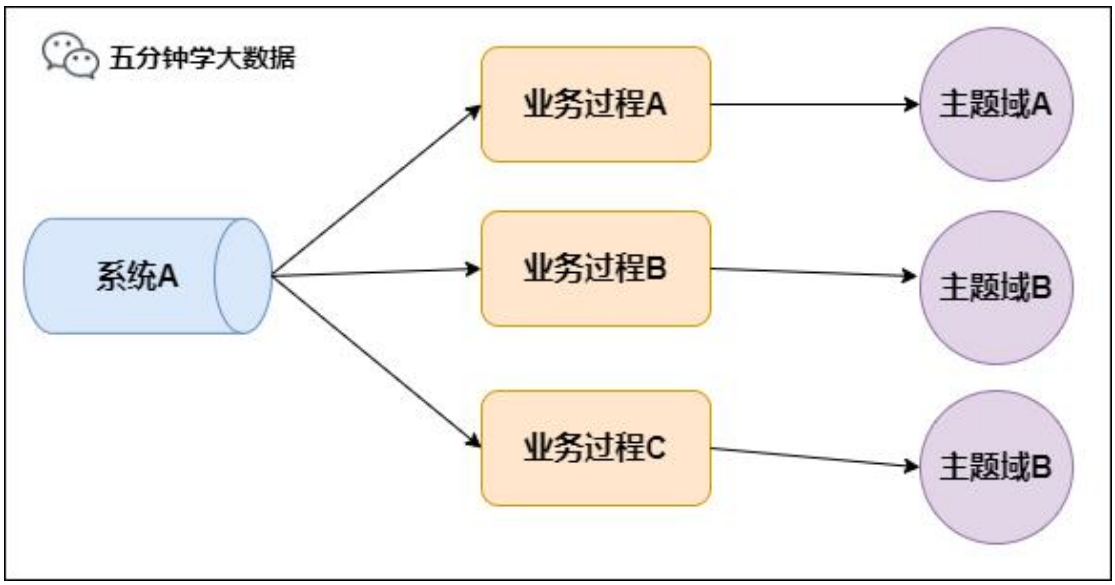
低基数维度数据：一般是配置表，比如枚举值对应的中文含义，或者日期维表。数据量可能是个位数或者几千几万。

2. 主题域划分原则

1）按照业务或业务过程划分

业务容易理解，就是指的功能模块/业务线。

业务过程：指企业的业务活动事件，如下单、支付、退款都是业务过程。不过需要注意的是，一个业务过程是一个不可拆分的行为事件，通俗的讲，业务过程就是企业活动中的事件。



2）按照数据域划分

数据域是指面向业务分析，将业务过程或者维度进行抽象的集合。其中，业务过程可以概括为一个个不可拆分的行为事件，在业务过程下，可以定义指标，维度是指度量的环境，如买家下单事件，买家是维度。为保障整个体系的生命力，数据域是需要抽象提炼，并且长期维护和更新的，但不轻易变动。在划分数据域时，既能涵盖当前所有的业务需求，又能在新业务进入时无影响地被包含进已有的数据域中和扩展新的数据域。

3. 数据模型设计原则

1）高内聚、低耦合

即**主题内部高内聚、 不同主题间低耦合**。明细层按照业务过程划分主题，汇总层按照“实体+ 活动”划分不同分析主题，应用层根据应用需求划分不同应用主题。

2）核心模型和扩展模型要分离

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的核心业务，扩展模型包括的字段支持个性化或少量应用的需要，**不能让扩展模型的字段过度侵入核心模型，以免破坏核心模型的架构简洁性与可维护性**。

3）公共处理逻辑下沉及单一

越是底层公用的处理逻辑越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实现，不要让公共逻辑多处同时存在**。

4）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。

5）数据可回滚

处理逻辑不变，在不同时间多次运行数据结果确定不变。

二、数仓公共开发规范

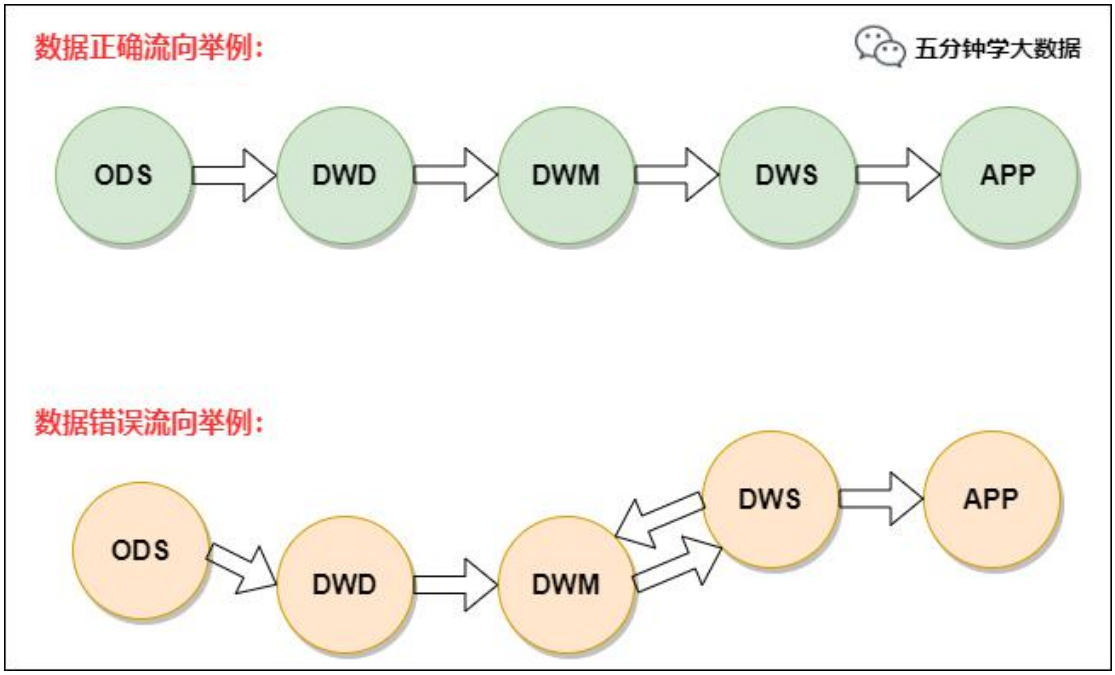
1. 层次调用规范

稳定业务按照标准的数据流向进行开发，即 ODS -> DWD -> DWS -> APP。**非稳定业务**或探索性需求，可以遵循 ODS -> DWD -> APP 或者 ODS -> DWD -> DWM ->APP 两个模型数据流。

在保障了数据链路的合理性之后，也必须保证模型分层引用原则：

- 正常流向：ODS -> DWD -> DWM -> DWS -> APP，当出现 ODS -> DWD -> DWS -> APP 这种关系时，说明主题域未覆盖全。应将 DWD 数据落到 DWM 中，对于使用频度非常低的表允许 DWD -> DWS。
- 尽量避免出现 DWS 宽表中使用 DWD 又使用（该 DWD 所归属主题域）DWM 的表。
- 同一主题域内对于 DWM 生成 DWM 的表，原则上要尽量避免，否则会影响 ETL 的效率。
- DWM、DWS 和 APP 中禁止直接使用 ODS 的表， ODS 的表只能被 DWD 引用。
- **禁止出现反向依赖**，例如 DWM 的表依赖 DWS 的表。

举例：



2. 数据类型规范

需统一规定不同的数据的数据类型，严格按照规定的数据类型执行：

1. 金额：double 或使用 decimal(28,6) 控制精度等，明确单位是分还是元。
2. 字符串：string。
3. id 类：bigint。
4. 时间：string。
5. 状态：string

3. 数据冗余规范

宽表的冗余字段要确保：

1. 冗余字段要使用高频，下游 3 个或以上使用。
2. 冗余字段引入不应造成本身数据产生过多的延后。
3. 冗余字段和已有字段的重复率不应过大，原则上不应超过 60%，如需要可以选择 join 或原表拓展。

4. NULL 字段处理规范

- 对于维度字段，需设置为-1
- 对于指标字段，需设置为 0

5. 指标口径规范

保证主题域内，指标口径一致，无歧义。

通过数据分层，提供统一的数据出口，统一对外输出的数据口径，避免同一指标不同口径的情况发生。

1）指标梳理

指标口径的不一致使得数据使用的成本极高，经常出现口径打架、反复核对数据的问题。在数据治理中，我们将需求梳理到的所有指标进行进一步梳理，明确其口径，如果存在两个指标名称相同，但口径不一致，先判断是否是进行合并，如需要同时存在，那么在命名上必须能够区分开。

2）指标管理

指标管理分为原子指标维护和派生指标维护。

原子指标：

- 选择原子指标的归属产线、业务板块、数据域、业务过程
- 选择原子指标的统计数据来源于该业务过程下的原始数据源
- 录入原子指标的英文名称、中文名称、概述
- 填写指标函数
- 系统根据指标函数自动生成原子指标的定义表达式
- 系统根据指标定义表达式以及数据源表生成原子指标 SQL

派生指标：

- 在原子指标的基础之上选择了一些维度或者修饰限定词。

6. 数据表处理规范

1）增量表

新增数据，增量数据是上次导出之后的新数据。

1. 记录每次增加的量，而不是总量；
2. 增量表，只报变化量，无变化不用报；

3. 每天一个分区。

2）全量表

每天的所有的最新状态的数据。

1. 全量表，有无变化，都要报；
2. 每次上报的数据都是所有的数据（变化的 + 没有变化的）；
3. 只有一个分区。

3）快照表

按日分区，记录截止数据日期的全量数据。

1. 快照表，有无变化，都要报；
2. 每次上报的数据都是所有的数据（变化的 + 没有变化的）；
3. 一天一个分区。

4）拉链表

记录截止数据日期的全量数据。

1. 记录一个事物从开始，一直到当前状态的所有变化的信息；
2. 拉链表每次上报的都是历史记录的最终状态，是记录在当前时刻的历史总量；
3. 当前记录存的是当前时间之前的所有历史记录的最后变化量（总量）；
4. 只有一个分区。

7. 表的生命周期管理

这部分主要是要通过对历史数据的等级划分与对表类型的划分生成相应的生命周期管理矩阵。

1）历史数据等级划分

主要将历史数据划分 P0、P1、P2、P3 四个等级，其具体定义如下：

- P0 ： 非常重要的主题域数据和非常重要的应用数据，具有不可恢复性，如交易、日志、集团 KPI 数据、 IPO 关联表。
- P1 ： 重要的业务数据和重要的应用数据，具有不可恢复性，如重要的业务产品数据。
- P2 ：重要的业务数据和重要的应用数据，具有可恢复性，如交易线 ETL 产生的中间过程数据。
- P3 ：不重要的业务数据和不重要的应用数据，具有可恢复性，如某些 SNS 产品报表。

2）表类型划分

1. **事件型流水表（增量表）**

事件型流水表（增量表）指数据无重复或者无主键数据，如日志。

2. **事件型镜像表（增量表）**

事件型镜像表（增量表）指业务过程性数据，有主键，但是对于同样主键的属性会发生缓慢变化，如交易、订单状态与时间会根据业务发生变更。

3. **维表**

维表包括维度与维度属性数据，如用户表、商品表。

4. **Merge 全量表**

Merge 全量表包括业务过程性数据或者维表数据。由于数据本身有新增的或者发生状态变更，对于同样主键的数据可能会保留多份，因此可以对这些数据根据主键进行 Merge 操作，主键对应的属性只会保留最新状态，历史状态保留在前一天分区 中。例如，用户表、交易表等都可以进行 Merge 操作。

5. **ETL 临时表**

ETL 临时表是指 ETL 处理过程中产生的临时表数据，一般不建议保留，最多 7 天。

6. **TT 临时数据**

TT 拉取的数据和 DbSync 产生的临时数据最终会流转到 DS 层，ODS 层数据作为原始数据保留下来，从而使得 TT&DbSync 上游数据成为临时数据。这类数据不建议保留很长时间，生命周期默认设置为 93 天，可以根据实际情况适当减少保留天数。

7. 普通全量表

很多小业务数据或者产品数据，BI 一般是直接全量拉取，这种方式效率快，对存储压力也不是很大，而且表保留很长时间，可以根据历史数据等级确定保留策略。

通过上述历史数据等级划分与表类型划分，生成相应的生命周期管理矩阵，如下表所示：

公众号：五分钟学大数据		P0	P1	P2	P3
ODS层	事件型流水表（增量表）	永久保留	3年	365天	180天
	事件型流水表（增量表）	永久保留	3年	365天	180天
	维表（全量表）	33天+极限存储	33天+极限存储	33天+极限存储	33天+极限存储
	Merge 全量表	2天	2天	2天	2天
	普通全量表	3年	3年	3年	3年
	新同步全量表	3天	3天	3天	3天
DWD层	事件型流水表（增量表）	永久保留	3年	365天	180天
	事件型流水表（增量表）	永久保留	3年	365天	180天
	维表（全量表）	33天+极限存储	33天+极限存储	33天+极限存储	33天+极限存储
	普通全量表	3年	365天	365天	180天
DWS层	各粒度数据	永久保留	3年	3年	3年
临时存储区	ETL临时表	7天	3天	3天	3天
	TT临时表	7天	7天	7天	7天
应用层	运营报表	永久保留	——	——	——
	对外数据	7年	——	——	——
	内部产品	3年	——	——	——
本表依据来源《大数据之路》					

三、数仓各层开发规范

1. ODS 层设计规范

同步规范：

1. 一个系统源表只允许同步一次；
2. 全量初始化同步和增量同步处理逻辑要清晰；
3. 以统计日期和时间进行分区存储；
4. 目标表字段在源表不存在时要自动填充处理。

表分类与生命周期：

1. ods 流水全量表：

- 不可再生的永久保存；
- 日志可按留存要求；
- 按需设置保留特殊日期数据；
- 按需设置保留特殊月份数据；

2. ods 镜像型全量表：

- 推荐按天存储；
- 对历史变化进行保留；
- 最新数据存储在最大分区；
- 历史数据按需保留；

3. ods 增量数据：

- 推荐按天存储；
- 有对应全量表的，建议只保留 14 天数据；
- 无对应全量表的，永久保留；

4. ods 的 etl 过程中的临时表：

- 推荐按需保留；
- 最多保留 7 天；
- 建议用完即删，下次使用再生成；

5. BDSync 非去重数据：

- 通过中间层保留，默认用完即删，不建议保留。

数据质量：

1. 全量表必须配置唯一性字段标识；

2. 对分区空数据进行监控；
3. 对枚举类型字段，进行枚举值变化和分布监控；
4. ods 表数据量级和记录数做环比监控；
5. ods 全表都必须要有注释；

2. 公共维度层设计规范

1）设计准则

1. 一致性

共维度在不同的物理表中的字段名称、数据类型、数据内容必须保持一致（历史原因不一致，要做好版本控制）

2. 维度的组合与拆分

- **组合原则**：

将维度与关联性强的字段进行组合，一起查询，一起展示，两个维度必须具有天然的关系，如：商品的基本属性和所属品牌。
无相关性：如一些使用频率较小的杂项维度，可以构建一个集合杂项维度的特殊属性。
行为维度：经过计算的度量，但下游当维度处理，例：点击量 0-1000,100-1000 等，可以做聚合分类。

- **拆分与冗余**：

针对重要性，业务相关性、源、使用频率等可分为核心表、扩展表。
数据记录较大的维度，可以适当冗余一些子集。

2）存储及生命周期管理

建议按天分区。

1. 3 个月内最大访问跨度<=4 天时，建议保留最近 7 天分区；
2. 3 个月内最大访问跨度<=12 天时，建议保留最近 15 天分区；

3. 3 个月内最大访问跨度<=30 天时，建议保留最近 33 天分区；
4. 3 个月内最大访问跨度<=90 天时，建议保留最近 120 天分区；
5. 3 个月内最大访问跨度<=180 天时，建议保留最近 240 天分区；
6. 3 个月内最大访问跨度<=300 天时，建议保留最近 400 天分区；

3. DWD 明细层设计规范

1）存储及生命周期管理

建议按天分区。

1. 3 个月内最大访问跨度<=4 天时，建议保留最近 7 天分区；
2. 3 个月内最大访问跨度<=12 天时，建议保留最近 15 天分区；
3. 3 个月内最大访问跨度<=30 天时，建议保留最近 33 天分区；
4. 3 个月内最大访问跨度<=90 天时，建议保留最近 120 天分区；
5. 3 个月内最大访问跨度<=180 天时，建议保留最近 240 天分区；
6. 3 个月内最大访问跨度<=300 天时，建议保留最近 400 天分区；

2）事务型事实表设计准则

- 基于数据应用需求的分析设计事务型事实表，结合下游较大的针对某个业务过程和分析指标需求，可考虑基于某个事件过程构建事务型实时表；
- 一般选用事件的发生日期或时间作为分区字段，便于扫描和裁剪；
- 冗余子集原则，有利于降低后续 IO 开销；
- 明细层事实表维度退化，减少后续使用 join 成本。

3）周期快照事实表

- 周期快照事实表中的每行汇总了发生在某一标准周期，如某一天、某周、某月的多个度量事件。
- 粒度是周期性的，不是个体的事务。
- 通常包含许多事实，因为任何与事实表粒度一致的度量事件都是被允许的。

4）累积快照事实表

- 多个业务过程联合分析而构建的事实表，如采购单的流转环节。
- 用于分析事件时间和时间之间的间隔周期。
- 少量的且当前事务型不支持的，如关闭、发货等相关的统计。

4. DWS 公共汇总层设计规范

数据仓库的性能是数据仓库建设是否成功的重要标准之一。**聚集**主要是通过**汇总明细粒度数据**来获得改进查询性能的效果。通过访问聚集数据，可以减少数据库在响应查询时必须执行的工作量，能够快速响应用户的查询，同时有利于减少不同用访问明细数据带来的结果不一致问题。

1）聚集的基本原则

- **一致性**。聚集表必须提供与查询明细粒度数据一致的查询结果。
- **避免单一表设计**。不要在同一个表中存储不同层次的聚集数据。
- **聚集粒度可不同**。聚集并不需要保持与原始明细粒度数据一样的粒度，聚集只关心所需要查询的维度。

2）聚集的基本步骤

第一步：确定聚集维度

在原始明细模型中会存在多个描述事实的维度，如日期、商品类别、卖家等，这时候需要确定根据什么维度聚集，如果只关心商品的交易额情况，那么就可以根据商品维度聚集数据。

第二步：确定一致性上钻

这时候要关心是按月汇总还是按天汇总，是按照商品汇总还是按照类目汇总，如果按照类目汇总，还需要关心是按照大类汇总还是小类汇总。当然，我们要做的只是了解用户需要什么，然后按照他们想要的进行聚集。

第三步：确定聚集事实

在原始明细模型中可能会有多个事实的度量，比如在交易中有交易额、交易数量等，这时候要明确是按照交易额汇总还是按照成交数量汇总。

3）公共汇总层设计原则

除了聚集基本的原则外，公共汇总层还必须遵循以下原则：

- **数据公用性**。汇总的聚集会有第三者使用吗？基于某个维度的聚集是不是经常用于数据分析中？如果答案是肯定的，那么就有必要把明细数据经过汇总沉淀到聚集表中。
- **不跨数据域**。数据域是在较高层次上对数据进行分类聚集的抽象。如以业务
- **区分统计周期**。在表的命名上要能说明数据的统计周期，如 _Id 表示最近 1 天，_td 表示截至当天，_nd 表示最近 N 天。

四、数仓命名规范

1. 词根设计规范

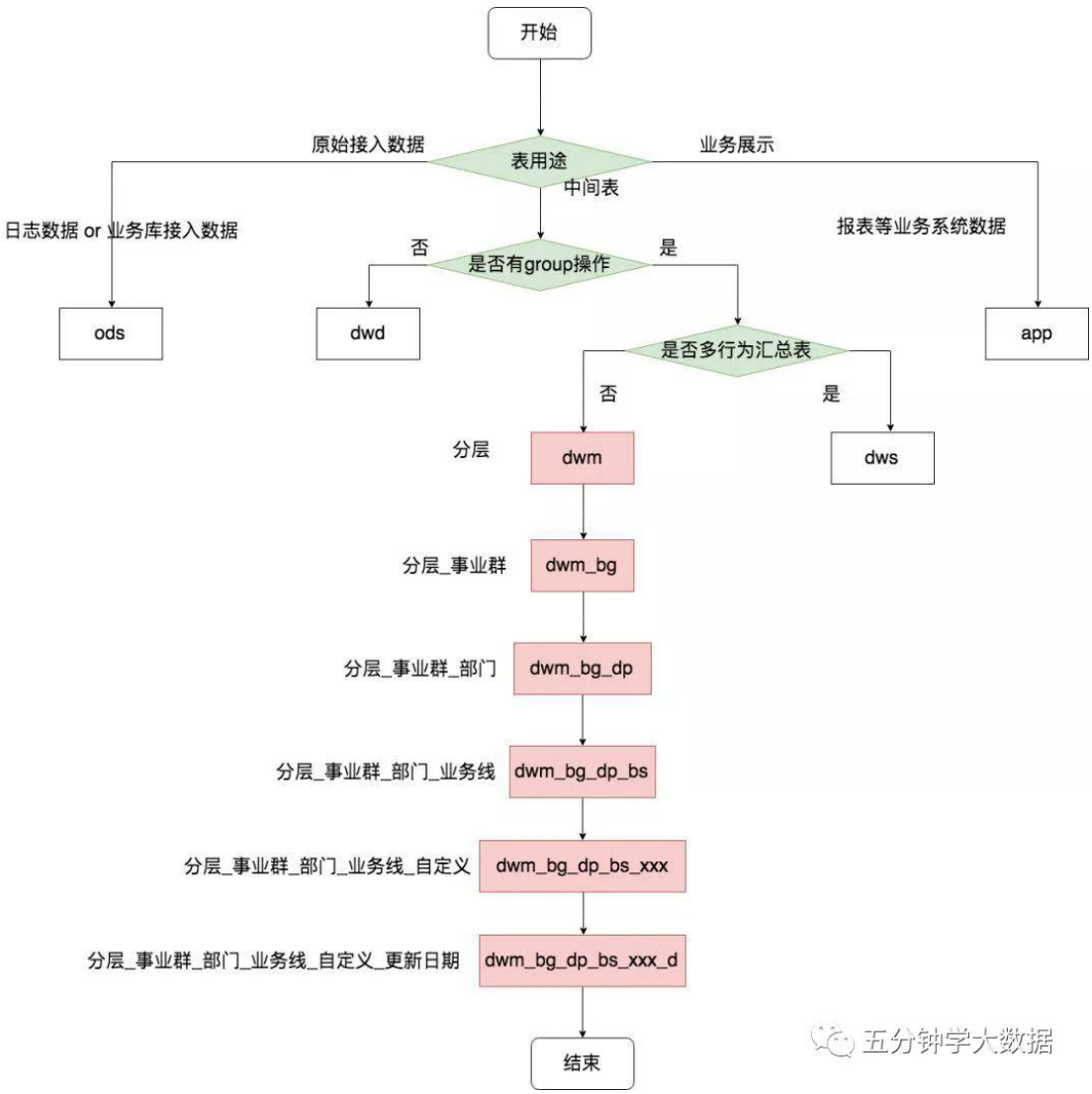
词根属于数仓建设中的规范，属于元数据管理的范畴，现在把这个划到数据治理的一部分。完整的数仓建设是包含数据治理的，只是现在谈到数仓偏向于数据建模， 而谈到数据治理，更多的是关于数据规范、数据管理。

表命名，其实在很大程度上是对元数据描述的一种体现，表命名规范越完善，我们能从表名获取到的信息就越多。比如：一部分业务是关于货架的，英文名是：rack，rack 就是一个词根，那我们就在所有的表、字段等用到的地方都叫 rack，不要叫成 别的什么。这就是词根的作用，用来统一命名，表达同一个含义。

指标体系中有很多“率”的指标，都可以拆解成 XXX+率，率可以叫 rate，那我们所有的指标都叫做 XXX+rate。

词根：可以用来统一表名、字段名、主题域名等等。

举例： 以流程图的方式来展示，更加直观和易懂，本图侧重 dwm 层表的命名 规范，其余命名是类似的道理：



第一个判断条件是该表的用途，是中间表、原始日志还是业务展示用的表 如果该表被判断为中间表，就会走入下一个判断条件：表是否有 group 操作 通过是否有 group 操作来判断该表该划分在 dwd 层还是 dwm 和 dws 层 如果不是 dwd 层，则需要判断该表是否是多个行为的汇总表（即宽表） 最后再分别填上事业群、部门、业务线、自定义名称和更新频率等信息即可。

分层：表的使用范围

事业群和部门：生产该表或者该数据的团队

业务线：表明该数据是哪个产品或者业务线相关

主题域：分析问题的角度，对象实体

自定义：一般会尽可能多描述该表的信息，比如活跃表、留存表等

更新周期：比如说天级还是月级更新

数仓表的命名规范如下：

1. 数仓层次：

公用维度：dim

DM 层：dm

ODS 层：ods

DWD 层：dwd

DWS 层：dws

2. 周期/数据范围：

日快照：d

增量：i

全量：f

周：w

拉链表：l

非分区全量表：a

2. 表命名规范

1）常规表

常规表是我们需要固化的表，是正式使用的表，是目前一段时间内需要去维护去完善的表。

规范：分层前缀[dwd|dws|ads]_部门_业务域_主题域_XXX_更新周期|数据范围

业务域、主题域我们都可以用词根的方式枚举清楚，不断完善。

更新周期主要的是时间粒度、日、月、年、周等。

2）中间表

中间表一般出现在 Job 中，是 Job 中临时存储的中间数据的表，中间表的作 用域只限于当前 Job 执行过程中，Job 一旦执行完成，该中间表的使命就完 成了，是可以删除的（按照自己公司的场景自由选择，以前公司会保留几天 的中间表数据，用来排查问题）。

规范：mid_table_name_[0~9|dim]

table_name 是我们任务中目标表的名字，通常来说一个任务只有一个目标表。这里加上表名，是为了防止自由发挥的时候表名冲突，而末尾大家可以选择自由

发挥，起一些有意义的名字，或者简单粗暴，使用数字代替，各有优劣吧，谨慎选择。

通常会遇到需要补全维度的表，这里使用 dim 结尾。

如果要保留历史的中间表，可以加上日期或者时间戳。

3）临时表

临时表是临时测试的表，是临时使用一次的表，就是暂时保存下数据看看，后续一般不再使用的表，是可以随时删除的表。

规范：tmp_xxx

只要加上 tmp 开头即可，其他名字随意，注意 tmp 开头的表不要用来实际使用，只是测试验证而已。

4）维度表

维度表是基于底层数据，抽象出来的描述类的表。维度表可以自动从底层表抽象出来，也可以手工来维护。

规范：dim_xxx

维度表，统一以 dim 开头，后面加上，对该指标的描述。

5）手工表

手工表是手工维护的表，手工初始化一次之后，一般不会自动改变，后面变更，也是手工来维护。

一般来说，手工的数据粒度是偏细的，所以暂时统一放在 dwd 层，后面如果有目标值或者其他类型手工数据，再根据实际情况分层。

规范：dwd_业务域_manual_xxx

手工表，增加特殊的主题域，manual，表示手工维护表。

3. 指标命名规范

1）公共规则

- 所有单词小写
- 单词之间下划线分割（反例：appName 或 AppName）
- 可读性优于长度（词根，避免出现同一个指标，命名一致性）
- 禁止使用 sql 关键字，如字段名与关键字冲突时 +col
- 数量字段后缀 _cnt 等标识...
- 金额字段后缀 _price 标识
- 天分区使用字段 dt，格式统一（yyyymmdd 或 yyyy-mm-dd）
- 小时分区使用字段 hh，范围（00-23）
- 分钟分区使用字段 mi，范围（00-59）
- 布尔类型标识：is_{业务}，不允许出现空值

2）指标命名规范

结合指标的特性以及词根管理规范，将指标进行结构化处理。

1. 基础指标词根，即所有指标必须包含以下基础词根：

基础指标词根	英文全称	Hive数据类型	MySQL数据类型	长度	精度	词根	样例
数量	count	Bigint	Bigint	10	0	cnt	
金额类	amout	Decimal	Decimal	20	4	amt	
比率/占比	ratio	Decimal	Decimal	10	4	ratio	0.9818
.....	.....	.....	.....			.....	

2. 业务修饰词，用于描述业务场景的词汇，例如 trade-交易。

3.日期修饰词，用于修饰业务发生的时间区间。

日期类型	全称	词根	备注
日	daily	d	
周	weekly	w	
.....	.....	.....	

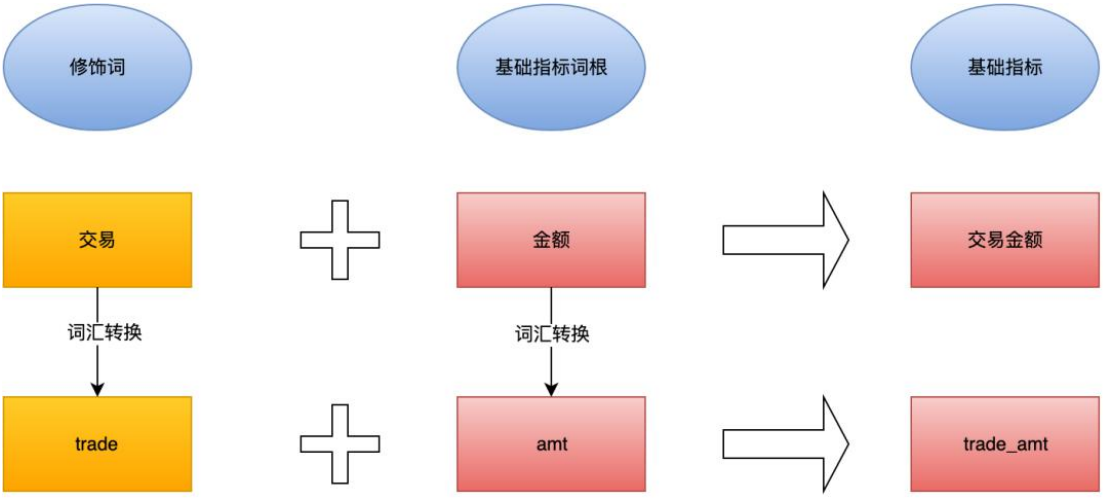
4.聚合修饰词，对结果进行聚集操作。

聚合类型	全称	词根	备注
平均	average	avg	
周累计	wtd	wtd	本周一截止到当天累计
.....	.....	.....	

5.基础指标，单一的业务修饰词+基础指标词根构建基础指标 ，例如：交易金额-trade_amt。

6.派生指标，多修饰词+基础指标词根构建派生指标。派生指标继承基础指标的特性，例如：安装门店数量-install_poi_cnt。

7.普通指标命名规范，与字段命名规范一致，由词汇转换即可以。



参考

本文档规范依据来源参考：

1. 《大数据之路：阿里巴巴大数据实践》
2. 《数仓工具箱：维度建模权威指南》
3. 《OneData 建设：美团 SaaS 数仓建设》

最后