> **Hive系列文章**
>
> Hive表的基本操作
> Hive中的集合数据类型
> Hive动态分区详解
> hive中orc格式表的数据导入
> Java通过jdbc连接hive
> 通过HiveServer2访问Hive
> SpringBoot连接Hive实现自助取数
> hive关联hbase表
> Hive udf 使用方法
> Hive基于UDF进行文本分词
> Hive窗口函数row number的用法
> 数据仓库之拉链表

建模方法论

> 数仓的建模或者分层，其实都是为了更好的去组织、管理、维护数据,所以当你站在更高的维度去看的话，所有的划分都是为了更好的管理。小到JVM 内存区域的划分，JVM 中堆空间的划分(年轻代、老年代、方法区等)，大到国家的省市区的划分，无一例外的都是为了更好的组织管理

1. **访问性能**：能够快速查询所需的数据，减少数据I/O。
2. **数据成本**：减少不必要的数据冗余，实现计算结果数据复用，降低大数据系统中的存储成本和计算成本。
3. **使用效率**：改善用户应用体验，提高使用数据的效率。
4. **数据质量**：改善数据统计口径的不一致性，减少数据计算错误的可能性，提供高质量的、一致的数据访问平台。

> 需要注意的建模其实是和公司的业务、公司的数据量、公司使用的工具、公司数据的使用方式密不可分的，因为模型是概念上的东西，需要理论落地至于落地到什么程度，就取决于公司的现状了

范式建模(关系型数据库)

范式建模法其实是我们在构建数据模型常用的一个方法，该方法的主要由Inmon所提倡，主要解决关系型数据库得数据存储，利用的一种技术层面上的方法，主要用于业务系统，**所以范式建模主要是利用关系型数据库进行数仓建设**

目前，我们在关系型数据库中的建模方法，大部分采用的是三范式建模法。

符合3NF要求的数据库设计，基本上解决了数据冗余过大，插入异常，修改异常，删除异常的问题。

三范式

第一范式

属性值不可再分，说直白点就是一列里面不能包含多个小列，就像下面这样

编号	品名	进货		销售		备注
		数量	单价	数量	单价	

1NF是所有关系型数据库的最基本要求，你在关系型数据库管理系统（RDBMS），例如SQL Server，Oracle，MySQL中创建数据表的时候，如果数据表的设计不符合这个最基本的要求，那么操作一定是不能成功的。也就是说，只要在RDBMS中已经存在的数据表，一定是符合1NF的

第二范式

这里我们先说一下，为什么有了第一范式，还需要第二范式，那是应为第一范式，不能消除重复，存在数据冗余过大，导致插入异常，删除异常，修改异常的问题

学号	姓名	系名	系主任	课名	分数
1022211101	李小明	经济系	王强	高等数学	95
1022211101	李小明	经济系	王强	大学英语	87
1022211101	李小明	经济系	王强	普通化学	76

所以要求每张表都要有一个主键，其它字段(列)**完全依赖主键**，也就是说要求实体的属性**完全依赖于主关键字。**也就是说表只描述一个事实，因为这账号表描述了3个事实，学生、课程、和系

> 例如，如果花名册里只有名字，没有学号，则重名的话会很麻烦。
> 所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系，例如上面的`系主任`和`系名`就是不依赖学号的，所以这里应该单独拆出来

第三范式

所有字段只能**直接依赖主键**，不得依赖于其它字段(非主属性) 消除依赖传递。所谓传递函数依赖指的是如果存在"A-->B-->C"的决定关系，则C传递函数依赖于A。也就是说**表中的字段和主键直接对应不依靠其他中间字段，说白了就是，决定某字段值的必须是主键，而不是一个依赖于主键的其他字段**

范式建模的优缺点

优点

节约存储(尤其是利用数据库进行数仓建设的时候)

规范化带来的好处是通过**减少数据冗余****提高更新数据的效率，同时保证数据完整性****。

结构清晰，易于理解

缺点

构建比较复杂

查询复杂(需要很多的关联)

不适合在大数据环境下构建(1 查询复杂 2 存储很便宜)

> 由于建模方法限定在关系型数据库之上，在某些时候反而限制了整个数据仓库模型的灵活性，性能等，特别是考虑到数据仓库的底层数据向数据集市的数据进行汇总时，需要进行一定的变通才能满足相应的需求。

为什么要学习范式建模

上游数据源往往是业务数据库，而这些业务数据库采用的实范式建模，所以了解范式建模可以帮助我们去合理的建设数仓

> 如果了解范式建模，从er 模型可以了解到数据架构，例如一个电商系统，从er模型就可以知道哪些涉及到商品的管理、用户的管理、订单管理，拿到这些关系之后，我们就可以更好的进行数仓管理与建
> 数据源的规范定义需要我们了解范式理论，可以更好的和业务系统进行对接

数仓的稀有系统，如报表系统设计的时候也会使用到范式建模

ER实体建模

将事务抽象为"实体"（Entity）、"属性"（Property）、"关系"（Relationship）来**表示数据关联和事物描述**，这种**对数据的抽象建模**通常被称为ER实体关系模型。

实体建模法并不是数据仓库建模中常见的一个方法，它来源于哲学的一个流派。

从哲学的意义上说，客观世界应该是可以细分的，客观世界应该可以分成由一个个实 体，以及实体与实体之间的关系组成。我们在数据仓库的建模过程中完全可以引入这个抽象的方法，**将整个业务也可以划分成一个个的实体，而每个实体之间的 关系，以及针对这些关系的说明就是我们数据建模需要做的工作。**

> 在日常建模中，"实体"用矩形表示，"关系"用菱形，"属性"用椭圆形。ER实体关系模型也称为E-R关系图

> 虽然实体法粗看起来好像有一些抽象，其实理解起来很容易。即我们可以将任何一个业务过程划分成 3 个部分，实体，事件和说明。

描述一个简单的事实：“小明开车去学校上学”。以这个业务事实为例，我们可以把“小明”，“学校”看成是一个实体， “上学”描述的是一个业务过程，我们在这里可以抽象为一个具体“事件”，而“开车去”则可以看成是事件“上学”的一个说明。

应用场景

ER模型是数据库设计的理论基础，当前几乎所有的OLTP系统设计都采用ER模型建模的方式。

Bill Inom提出的数仓理论，推荐采用ER关系模型进行建模。

BI架构提出分层架构，数仓底层ods、dwd也多采用ER关系模型进行设计。

> 由于实体建模法，能够很轻松的实现业务模型的划分，因此，在业务建模阶段和领域概念建模阶段，实体建模法有着广泛的应用。

业务归纳

使用的抽象归纳方法其实很简单，任何业务可以看成 3 个部分：

1. 实体，主要指领域模型中特定的概念主体，指发生业务关系的对象
2. 事件，主要指概念主体之间完成一次业务流程的过程，特指特定的业务过程
3. 说明，主要是针对实体和事件的特殊说明

维度建模

概念和背景

维度模型是数据仓库领域大师Ralph Kimball 所倡导，他的《数据仓库工具箱》，是数据仓库工程领域最流行的数仓建模经典。维度建模以分析决策的需求出发构建模型，构建的数据模型为分析需求服务，因此它重点解决用户如何更快速完成分析需求，同时还有较好的大规模复杂查询的响应性能。

维度建模源自数据集市，主要面向分析场景 **Ralph Kimball 推崇数据集市的集合为数据仓库**，同时也提出了对数据集市的维度建模，将数据仓库中的表划分为事实表、维度表两种类型。

一般也称之为星型结构建模，有时也加入一些雪花模型在里面。**维度建模是一种面向用户需求的、容易理解的、访问效率高的建模方法**

> 维度模型通常以一种被称为星型模式的方式构建。所谓星型模式，就是以一个事实表为中心，周围环绕着多个维度表。

> 还有一种模式叫做雪花模式，是对维度做进一星型模型做OLAP分析很方便

为什么选择维度建模

适配大数据的处理方式

维度模型的非强范式的，可以更好的利用大数据处理框架的处理能力，避免范式操作的过多关联操作，可以实现高度的并行化。

数据仓库大多数时候是比较适合使用星型模型构建底层数据Hive表，通过大量的冗余来提升查询效率，星型模型对OLAP的分析引擎支持比较友好，这一点在Kylin中比较能体现。

雪花模型在关系型数据库中如MySQL，Oracle中非常常见，尤其像电商的数据库表。

自下而上的建设现状

表已经存在，业务已经开发完毕，需求直接提过来了，这几乎是一个普遍现状，因为很少有公司会提前成立数据部门，让数据部门跟随着业务从头开始一直成长，都是当业务发展到一定的阶段了，想通过数据来提高公司的运营效果

简单的模型 使用简单

这个模型相对来说是比较简单的，简单主要体现在两个方面

1. 维度建模非常直观，紧紧围绕着业务模型，**可以直观的反映出业务模型中的业务问题。**不需要经过特别的抽象处理，即可以完成维度建模。这一点也是维度建模的优势。
2. 星型结构的实现不用考虑很多正规化的因素，设计与实现都比较简单。

分层和建模的关系

明细层的范式模型

明细层采用传统的三范式关系模型。这一层次的数据模型要将业务过程描述清楚，将源数据（即业务系统）中隐含的、有歧义的概念进行清晰化，如活跃用户、VIP用户等。该层次的数据模型追求的目标是灵活地表达业务过程，要保证数据一致性、唯一性、正确性，以尽量少的代价与源数据保持数据同步，同时该层次的数据模型不建议开给不懂技术的业务人员直接使用，因此，采用关系型的三范式模型是最佳的选择。

集市层的维度模型

集市层是按照业务主题、分主题构建出来的、面向特定部门或人员的数据集合，该层次的数据模型会开放给业务人员使用，进行数据挖掘及业务分析。由于业务员多数不懂数据库技术，缺少将业务需求转换为关系型数据结构的逻辑思维，更写不出复杂的SQL语句，因此，越简单的数据模型，越能被他们所接受，因此，这个层次所构建出来的数据模型，要按照业务过程进行组织，每个事实表代表一个独立的业务过程，事实表之间不存在直接的依赖关系，这样业务人员可以很容易地将分析需求对应到事实表上，利用工具或手工写出简单的SQL，将统计数据提取出来进行分析。

模型实现

模型的实现主要指的是在维度建模过程中，需要对维度表和事实表进行关联设计，而这里我们对维度表的设计，就决定了我们最终与事实表关联的之后的形态。也就是说我们可以根据事实表和维度表的关系，又可将常见的模型分为星型模型和雪花型模型

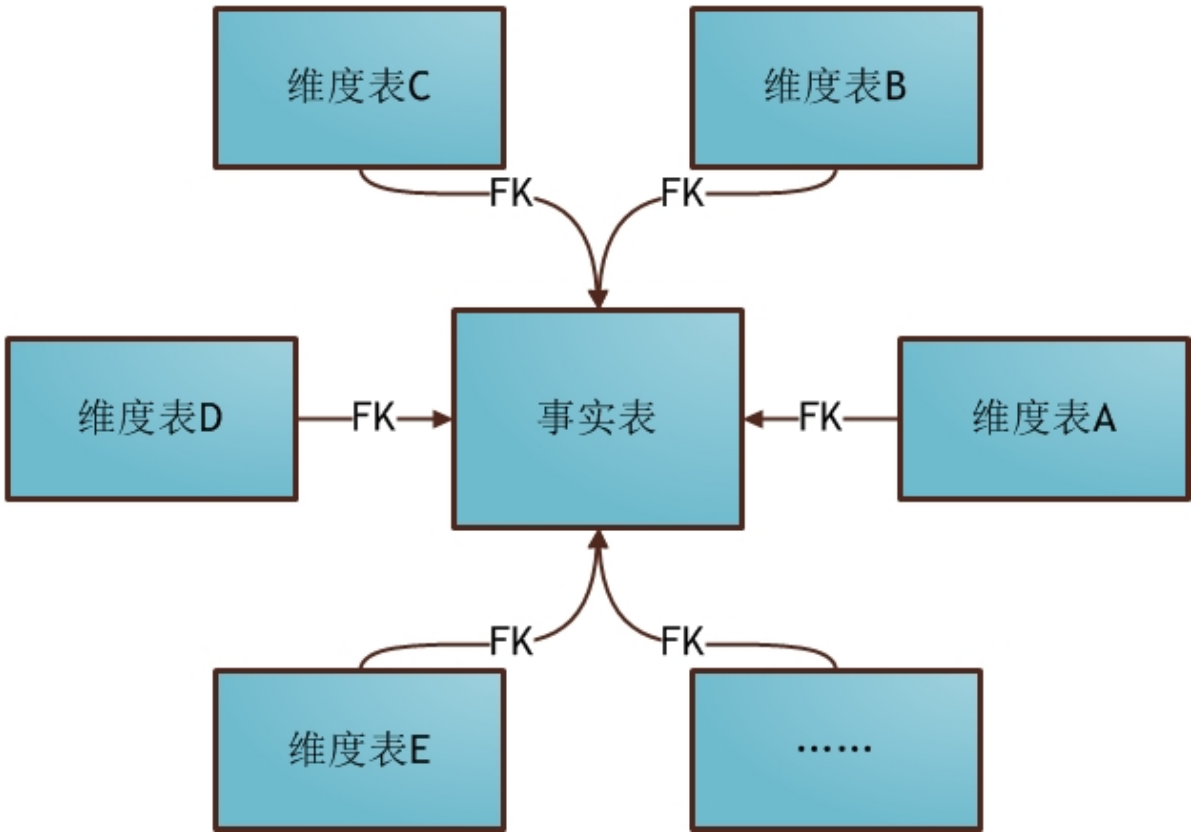
星型模型和雪花模型的**主要区别在于对维度表的拆分，对于雪花模型，维度表的设计更加规范，一般符合3NF**；而**星型模型，一般采用降维的操作，利用冗余来避免模型过于复杂，提高易用性和分析效率。**

星型模型

核心是**一个事实表及多个非正规化描述的维度表**组成，维度表之间是没有关联的，维度表是直接关联到事实表上的，只有当维度表极大，存储空间是个问题时，才考虑雪花型维度，简而言之，最好就用星型维度即可

当所有维表都直接连接到“ 事实表”上时，整个图解就像星星一样，故将该模型称为星型模型

星型架构是一种非正规化的结构，多维数据集的每一个维度都直接与事实表相连接，不存在渐变维度，所以数据有一定的冗余，如在地域维度表中，存在国家 A 省 B 的城市 C 以及国家 A 省 B 的城市 D 两条记录，那么国家 A 和省 B 的信息分别存储了两次，即存在冗余。

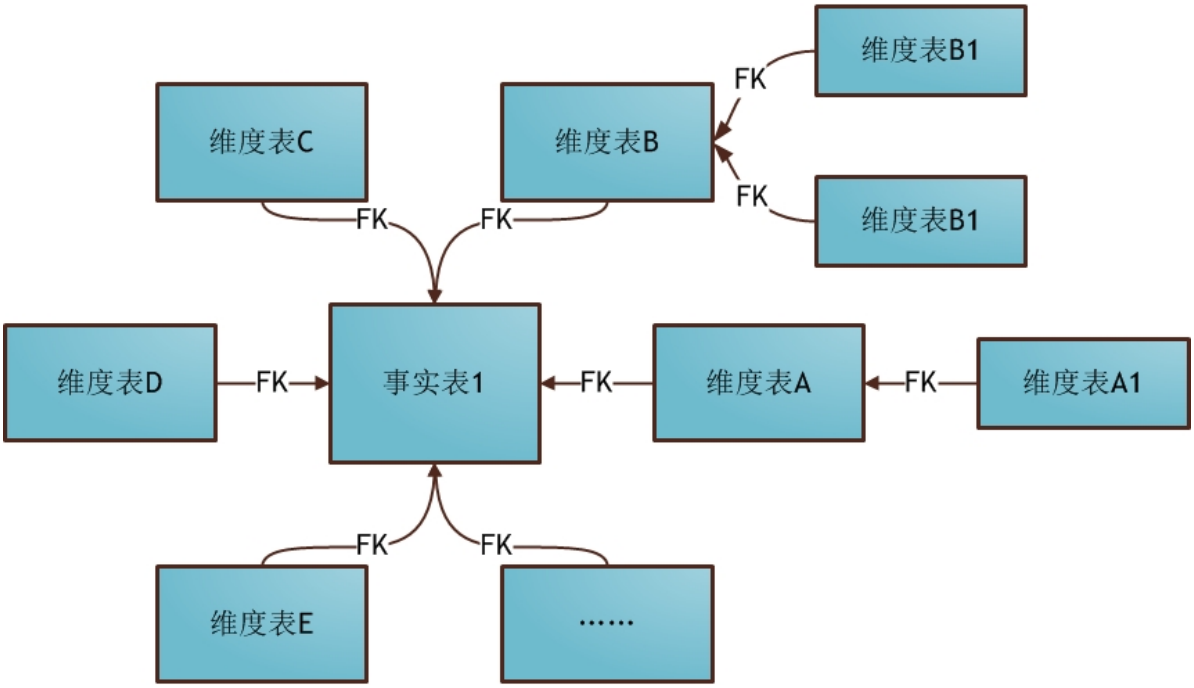


雪花模型

星形模式中的维表相对雪花模式来说要大，而且不满足规范化设计。雪花模型相当于将星形模式的大维表拆分成小维表，满足了规范化设计。然而这种模式在实际应用中很少见，因为这样做会导致开发难度增大，而数据冗余问题在数据仓库里并不严重

可以认为雪花模型是星型模型的一个扩展，每个维度表可以继续向外扩展，连接多个子维度。

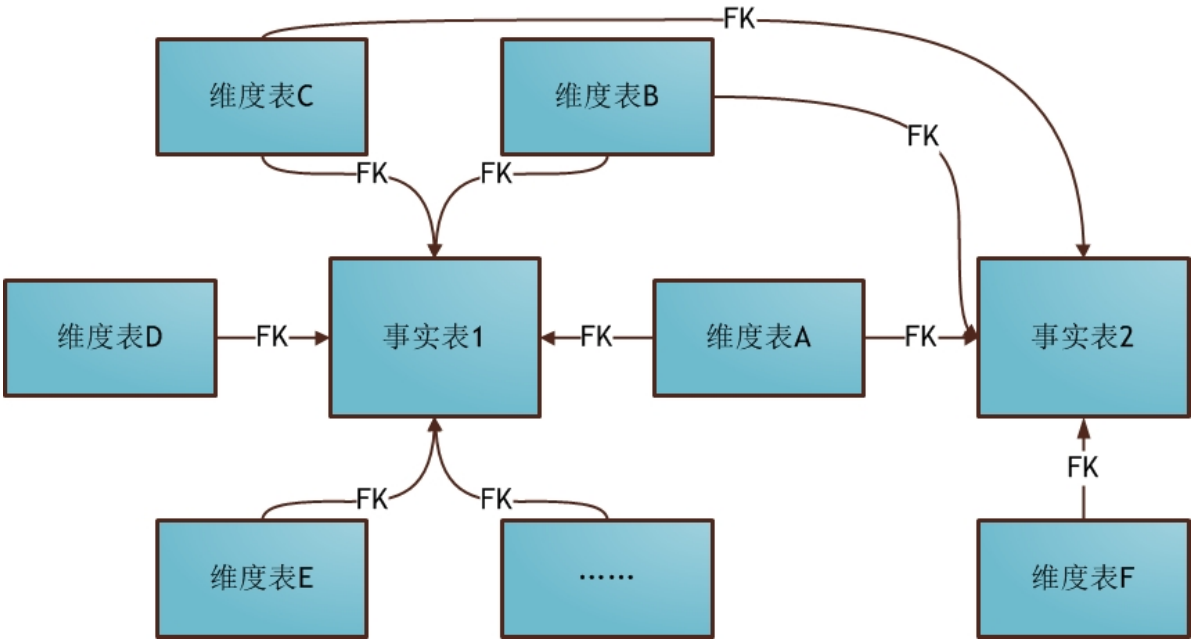
当有一个或多个维表没有直接连接到事实表上，而是通过其他维表连接到事实表上时，其图解就像多个雪花连接在一起，故称雪花模型



星座模型

前面介绍的两种维度建模方法都是多维表对应单事实表，但在很多时候维度空间内的事实表不止一个，而一个维表也可能被多个事实表用到。在业务发展后期，绝大部分维度建模都采用的是星座模式。

可以认为是多个事实表的关联或者是星型模型的关联，其实到了业务发展后期都是星座模型



应用场景

维度建模**是面向分析场景而生**，针对分析场景构建数仓模型，重点关注快速、灵活的解决分析需求，同时能够提供大规模数据的快速响应性能。

针对性强，主要应用于数据仓库构建和OLAP引擎底层数据模型

优点

方便使用，模型简单

适合大数据下的处理操作(其实就是shuffle)

适合OLAP操作(上钻下钻)

维度建模非常直观，紧紧围绕着业务模型，可以直观的反映出业务模型中的业务问题。不需要经过特别的抽象处理，即可以完成维度建模。

可扩展，维度模型是可扩展的。由于维度模型允许数据冗余，因此当向一个维度表或事实表中添加字段时，不会像关系模型那样产生巨大的影响，带来的结果就是更容易容纳不可预料的新增数据。

缺点

数据冗余，维度补全后造成的数据浪费

灵活性差，维度变化造成的数据更新量大(例如刷数据的时候，需要刷大量的表)

与典型的范式理论差异很大，如数据不一致，比如用户发起购买行为的时候的数据，和我们维度表里面存放的数据不一致

> 既然如此为什么还要使用范式建模呢，其实和我们使用的工具有关系

由于在构建星型模式之前需要进行大量的数据预处理，因此会导致大量的数据处理工作。而且，当业务发生变化，需要重新进行维度的定义时，往往需要重新进行维度数据的预处理。而在这些与处理过程中，往往会导致大量的数据冗余。

如果只是依靠单纯的维度建模，不能保证数据来源的一致性和准确性，**而且在数据仓库的底层，不是特别适用于维度建模的方法。**

维度建模的领域主要适用与数据集市层，它的最大的作用其实是为了解决数据仓库建模中的性能问题。维度建模很难能够提供一个完整地描述真实业务实体之间的复杂关系的抽象方法

总结

上述的这些方法都有自己的优点和局限性，在创建自己的数据仓库模型的时候，可以参考使用上述的三种数据仓库得建模方法，在各个不同阶段采用不同的方法，从而能够保证整个数据仓库建模的质量。

方法论仅仅停留在理论层面上，落地实现的才真正决定了数仓设计的好坏，当然再好的方法，只有在合适的阶段使用，才有意义，才能发挥它最大的价值