1. 事实表的类型?

事实表有：事务事实表、周期快照事实表、累积快照事实表、非事实事实表。

事务事实表

事务事实表记录的是事务层面的事实，保存的是最原子的数据，也称“原子事实表”。事务事实表中的数据在事务事件发生后产生，数据的粒度通常是每个事务记录一条记录。

周期快照事实表

以具有规律性的、可预见的时间间隔来记录事实。它统计的是间隔周期内的度量统计，每个时间段一条记录，是在事务事实表之上建立的聚集表。

累积快照事实表

累积快照表记录的不确定的周期的数据。代表的是完全覆盖一个事务或产品的生命周期的时间跨度，通常具有多个日期字段，用来记录整个生命周期中的关键时间点。

非事实型事实表

这个与上面三个有所不同。事实表中通常要保留度量事实和多个维度外键，度量事实是事实表的关键所在。

非事实表中没有这些度量事实，只有多个维度外键。非事实型事实表通常用来跟踪一些事件或说明某些活动的范围。

第一类非事实型事实表是用来跟踪事件的事实表。例如：学生注册事件

第二类非事实型事实表是用来说明某些活动范围的事实表。例如：促销范围事实表。

表 11.7　三种事实表的比较

	事务事实表	周期快照事实表	累积快照事实表
时期/时间	离散事务时间点	以有规律的、可预测的间隔产生快照	用于时间跨度不确定的不断变化的工作流
日期维度	事务日期	快照日期	相关业务过程涉及的多个日期
粒度	每行代表实体的一个事务	每行代表某时间周期的一个实体	每行代表一个实体的生命周期
事实	事务事实	累积事实	相关业务过程事实和时间间隔事实
事实表加载	插入	插入	插入与更新
事实表更新	不更新	不更新	业务过程变更时更新

2. 数仓分层

一般情况下，将数据模型分为3层:

源数据层ODS

存放的是接入的原始数据

经过ETL之后装入本层，大多是按照源头业务系统的分类方式而分类的。为了考虑后续可能追溯数据，因此对这一层不建议做过多的数据清洗工作，原封不动接入元数据即可，至于数据的去噪，去重，异常处理等过程可以放在后面的DW层。

数据仓库层DW

重点设计的数据仓库中间层数据，在这里ODS层获得的数据按照主题建立各种数据模型，DW又细分：

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，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，将维度退化到事实表中，减少事实表和维度表的关联。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。

数据中间层：DWM(Data Warehouse Middle)

在DWD层的数据基础上，对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，直观来说，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。

数据服务层：DWS(Data Warehouse Service)

又称为数据集市或者宽表，按照业务划分，例如流量，订单，用户等，生成字段比较多的宽表，用于后续的业务查询，OLAP分析，数据分析等。

数据应用层APP：面向业务定制的应用数据

主要提供给数据铲平和数据分析使用的数据，一般会放在ES，MYSQL，Redis等系统供线上系统使用，也可以放在Hive中供数据分析和数据挖掘使用。

补充：维表层 Dimension

- 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。

3. 数仓和普通数据库区别

数据库与数据长裤的区别实际讲的是OLTP和OLAP的区别。

OLTP特点如下

联机事务处理OLTP(on-line transaction processing)主要是执行基本的，日常的事务处理，比如数据库记录的增删改查，比如在银行存取一笔款，就是一个事务交易

- 实时性要求高
- 数据量不是很大
- 交易一般是确定的，所以是OLTP是对确定性的数据进行存取(比如存取款都有一个特定的金额)
- 并发现要求高并且严格要求事物的完整性安全性(比如你和你的家人同时间在不同的银行取同一账户的钱)

OLAP特点如下

联机分析处理OLAP(on-line Analytical Processing)是数据仓库系统的主要应用，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果。典型的应用就是复杂的动态报表系统：

- 实时性要求不是很高，很多应用的顶多是每天更新一下数据
- 数据量大，因为OLAP支持的是动态查询，所以用户也许要通过很多数据的统计后才能得到想要的信息，例如时间序列分析等，所以处理的数据量很大
- 因为重点在于决策支持，所以查询一般是动态的，也就是说允许用户随时提出查询的要求，所以在OLAP中通过一个重要概念维来搭建一个动态查询的平台或技术，供用户自己去决定需要知道什么信息

简单来说，OLTP就是我们常说的关系型数据库，即记录即时的增删改查就是我们常用的，这是数据库的基础，TPCC(Transaction Processing Performance Council)属于此类

OLAP即联机分析处理，是数据仓库的核心部分。所谓数据仓库是对于大量已经由OLTP形成的数据的一种分析型的数据库，用于处理商业智能，决策支持等重要的决策信息。数据仓库是在数据库应用到一定程度后对历史数据的加工与分析，读取较多，更新较少，TPCH属于此类，对于OLAP，列存储模式比通常的行存储模式可能更具有优势

OLAP不应该对OLTP产生任何影响，(理想情况下)OLTP应该完全感觉不到OLAP的存在

	OLTP	OLAP
用户	操作人员,底层管理人员	决策人员,高级管理人员
功能	日常操作处理	分析决策
DB设计	面向应用	面向主题
数据	当前的,最新的细节的,二维的分立的	历史的,聚集的,多维的,集成的,统一的
存取	读写数十条记录	读上百万条记录
工作单位	简单的事务	复杂的查询
用户数	上千个	上百万个
DB大小	100MB-GB	100GB-TB
时间要求	具有实时性	对时间的要求不严格
主要应用	数据库	数据仓库

4. 星型模型和雪花模型的区别

- 事实表：一般是用户行为产生的数据，数据量比较大
- 维度表：一般是一些属性信息，用户信息表，产品信息表等。这些属性信息不经常变动
- 数据仓库模型：星型模型和雪花模型

星型模型：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。特点: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，不存在渐变维表，所以数据有一定的冗余，如在地域维度表，存在国家A省B的城市C和国家A省B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。

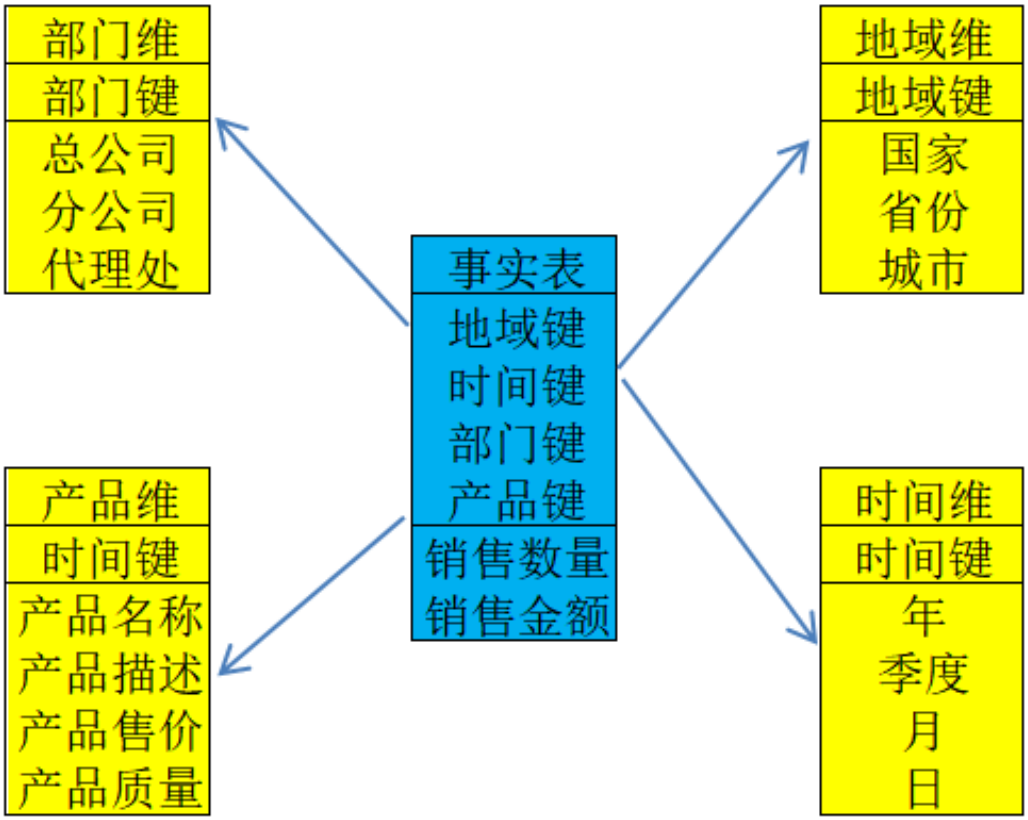


图:销售数据仓库中的星型模型

雪花模型：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，其图解就像多个雪花连接在一起，故称为雪花型。雪花模型是对星型模型的扩展。它对星型模型的维表进一步层次化，原有的各维表可能被扩展为小的事实表。如下图，将地域维表又分解为国家省份城市等维表。它的优点是通过最大限度地减少数据存储量及联合较小的维表来改善查询性能。雪花型结构去除了数据冗余。

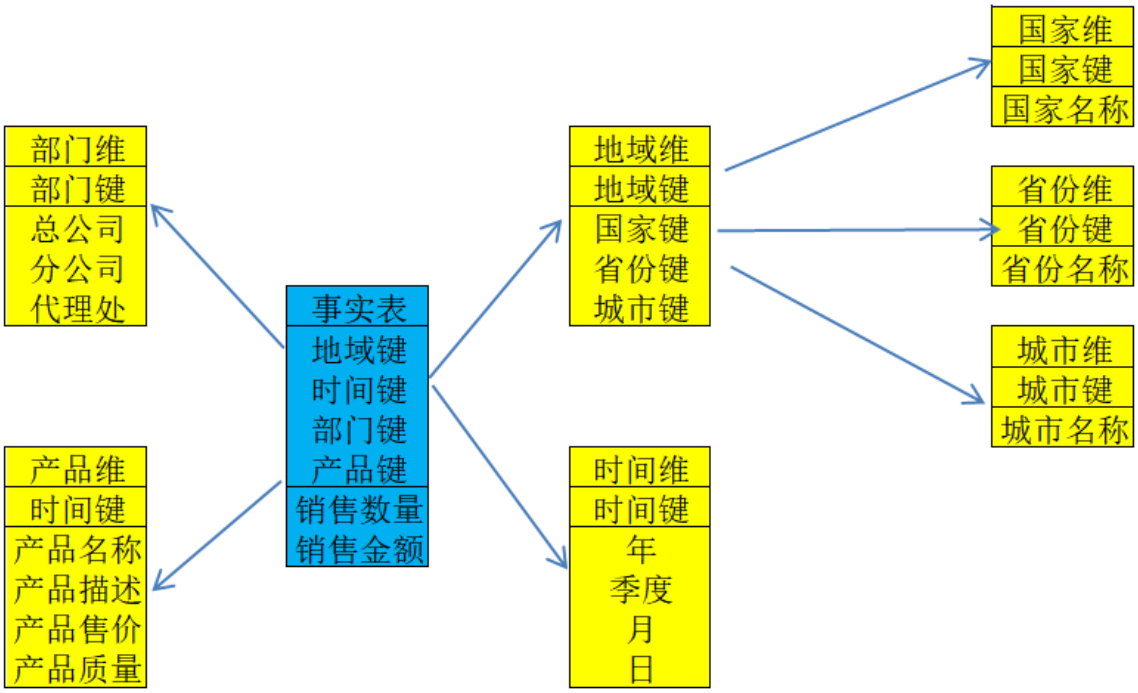


图:销售数据仓库中的雪花模型

星型模型因为数据的冗余所以很多统计查询不需要做外部地连接，因此一般情况下效率比雪花模型高。星型结构不用考虑很多正规化地因素，设计与实现都比较简单。雪花模型由于去除了冗余，有些统计就需要通过表连接才能产生，所以效率不一定有星型模型高。正规化也是一种比较复杂地过程，相应地数据库结构设计，数据地ETL以及后期地维护都要复杂一些。因此在冗余可以接受地前提下，实际运用中星型模型使用更多，也更有效。

5. 拉链表

拉链表是针对数据仓库设计中表存储数据的方式而定义的，顾名思义，所谓拉链，就是记录历史。记录一个事物从开始，一直到当前状态的所有变化的信息

拉链表的使用场景

在数据仓库的数据模型设计过程中，经常会遇到下面这种表的设计

- 有一些表的数据量很大，比如一张用户表，大约10亿条记录，50个字段，这种表即使使用ORC压缩，单张表的存储也会超过100G，在hdfs使用双备份或者三备份就更大了
- 表中的部分字段会被update更新操作，如用户联系方式，产品的描述信息，订单的状态等
- 需要查看某一时间点或者时间段的历史快照信息，比如查看某一订单在历史某一时间点的状态
- 表中的记录变化的比例和频率不是很大，比如，总共有10亿的用户，每天新增和发生变化的有200万左右，变化的比例占的很小

对于这种表的设计，有几种方案可选

- 方案一：每天只留最新的一份，比如我们每天用datax抽取最新的一份全量数据到Hive中
- 方案二：每天保留一份全量的切片数据
- 方案三：使用拉链表

使用拉链表的原因

对于方案一，实现起来很简单，每天删除前一天的数据，重新抽一份最新的。优点很明显，节省空间，一些普通的使用也很方便，不用在选择表的时候加一个时间分区。缺点同样很明显，没有历史数据，想翻旧账只能通过其他方式，比如从流水表里抽。

对于方案二，每天一份全量的切片是一种比较稳妥的方案，而且历史数据也在。缺点就是存储空间占用太大太大，如果对这边表每天都保留一份全量，那么每次全量中会保存很多不变的信息，对存储是极大的浪费。

对于方案三，拉链表在使用上基本兼顾了我们的需求。首先在空间上做了一个取舍，虽说不像方案一那样占用量那么小，但是它每日的增量可能只有方案二的千分之一甚至是万分之一。它能满足方案二所能满足的需求，既能获取最新数据，也能添加筛选条件获取历史数据，所以我们还是很有必要使用拉链表。

6. 数据漂移如何解决

什么是数据漂移?

通常是指ods表的同一个业务日期数据中包含了前一天或后一天凌晨附近的数据或者丢失当天变更的数据，这种现象就叫做漂移，且在大部分公司中都会遇到的场景。

如何解决数据漂移问题?

通常有两种解决方案：

1. 多获取后一天的数据，保障数据只多不少
2. 通过多个时间戳字段来限制时间获取相对准确的数据

第一种方案比较暴力，这里不做过多解释，主要来讲解一下第二种解决方案。（首先这种解决方案在大数据之路这本书有体现）。

第一种方案里，时间戳字段分为四类：

1. 数据库表中用来标识**数据记录更新时间**的时间戳字段（假设这类字段叫 modified time ）。
2. 数据库**日志中**用来标识**数据记录更新时间**的时间戳字段·（假设这类字段叫 log_time）。
3. 数据库表中用来记录**具体业务过程发生时间**的时间戳字段 （假设这类字段叫 proc_time）。
4. 标识数据记录**被抽取到时间**的时间戳字段（假设这类字段extract time）。

理论上这几个时间应该是一致的，但往往会出现差异，造成的原因可能为：

1. 数据抽取需要一定的时间，extract_time往往晚于前三个时间。
2. 业务系统手动改动数据并未更新modfied_time。
3. 网络或系统压力问题，log_time或modified_time晚于proc_time。

通常都是根据以上的某几个字段来切分ODS表，这就产生了数据漂移。具体场景如下：

1. 根据extract_time进行同步。
2. 根据modified_time进行限制同步， 在实际生产中这种情况最常见，但是往往会发生不更新 modified time 而导致的数据遗漏，或者凌晨时间产生的数据记录漂移到后天 。由于网络或者系统压力问题， log_time 会晚 proc_time ，从而导致凌晨时间产生的数据记录漂移到后一天。
3. 根据proc_time来限制，会违背ods和业务库保持一致的原则，因为仅仅根据proc_time来限制，会遗漏很多其他过程的变化。

第二种解决方案：

1. 首先通过log_time多同步前一天最后15分钟和后一天凌晨开始15分钟的数据，然后用modified_time过滤非当天的数据，这样确保数据不会因为系统问题被遗漏。
2. 然后根据log_time获取后一天15分钟的数据，基于这部分数据，按照主键根据log_time做升序排序，那么第一条数据也就是最接近当天记录变化的。
3. 最后将前两步的数据做全外连接，通过限制业务时间proc_time来获取想要的数据。

7. 维度建模和范式建模的区别

通常数据建模有以下几个流程：

1. 概念建模：即通常先将业务划分多个主题。
2. 逻辑建模：即定义各种实体、属性和关系。
3. 物理建模：设计数据对象的物理实现，比如表字段类型、命名等。

那么范式建模，即3NF模型具有以下特点：

1. 原子性，**即数据不可分割。**
2. 基于第一个条件，实体属性完全依赖于主键，不能存在仅依赖主关键字一部分属性。**即不能存在部分依赖**。
3. 基于第二个条件，任何非主属性不依赖于其他非主属性。**即消除传递依赖。**

基于以上三个特点，3NF的最终目的就是为了降低数据冗余，保障数据一致性；同时也有了数据关联逻辑复杂的缺点。

而维度建模是面向分析场景的，主要关注点在于快速、灵活，能够提供大规模的数据响应。

常用的维度模型类型主要有：

1. 星型模型：即由一个事实表和一组维度表组成，每个维表都有一个维度作为主键。事实表居中，多个维表呈辐射状分布在四周，并与事实表关联，形成一个星型结构。
2. 雪花模型：在星型模型的基础上，基于范式理论进一步层次化，将某些维表扩展成事实表，最终形成雪花状结构
3. 星系模型：基于多个事实表，共享一些维度表。

8. 数据仓库和数据库的区别?

从目标、用途、设计来说：

- 数据库是面向事物处理的，数据是由日常的业务产生的，常更新；数据仓库是面向主题的，数据来源多样，经过一定的规则转换得到，用来分析。
- 数据库一般用来存储当前事务性数据，如交易数据；数据仓库一般存储的历史数据。
- 数据库的设计一般是符合三范式的，有最大的精确度和最小的冗余度，有利于数据的插入；数据仓库的设计一般不符合三范式，有利于查询。

9. 元数据的理解?

狭义来讲就是用来描述数据的数据。

广义来看，除了业务逻辑直接读写处理的业务数据，所有其他用来维护整个系统运转所需要的数据，都可以较为元数据。

定义：元数据metadata是关于数据的数据。在数仓系统中，元数据可以帮助数据仓库管理员和数据仓库开发人员方便的找到他们所关心的数据；元数据是描述数据仓库内部数据的结构和建立方法的数据。按照用途可分为：技术元数据、业务元数据。

技术元数据

存储关于数据仓库技术细节的数据，用于开发和管理数据仓库使用的数据。

- 数据仓库结构的描述，包括数据模式、视图、维、层次结构和导出数据的定义，以及数据集市的位置和内容
- 业务系统、数据仓库和数据集市的体系结构和模式

- 由操作环境到数据仓库环境的映射，包括元数据和他们的内容、数据提取、转换规则和数据刷新规则、权限等。

业务元数据

从业务角度描述了数据仓库中的数据，他提供了介于使用者和实际系统之间的语义层，使不懂计算机技术的业务人员也能读懂数仓中的数据。

- 企业概念模型：表示企业数据模型的高层信息。整个企业业务概念和相互关系。以这个企业模型为基础，不懂sql的人也能做到心中有数
- 多维数据模型。告诉业务分析人员在数据集市中有哪些维、维的类别、数据立方体以及数据集市中的聚合规则。
- 业务概念模型和物理数据之间的依赖。业务视图和实际数仓的表、字段、维的对应关系也应该在元数据知识库中有所体现。

10. 元数据管理系统?

元数据管理往往容易被忽视，但是元数据管理是不可或缺的。一方面元数据为数据需求方提供了完整的数仓使用文档，帮助他们能自主快速的获取数据；另一方面数仓团队可以从日常的数据解释中解脱出来，无论是对后期的迭代更新还是维护，都有很大的好处。元数据管理可以让数据仓库的应用和维护更加的高效。

元数据管理功能

- 数据地图：以拓扑图的形式对数据系统的各类数据实体、数据处理过程元数据进行分层次的图形化展示，并通过不同层次的图形展现。
- 元数据分析：血缘分析、影响分析、实体关联分析、实体差异分析、指标一致性分析。
- 辅助应用优化：结合元数据分析功能，可以对数据系统的应用进行优化。
- 辅助安全管理：采用合理的安全管理机制来保障系统的数据安全；对数据系统的数据访问和功能使用进行有效监控。
- 基于元数据的开发管理：通过元数据管理系统规范日常开发的工作流程。

元数据管理标准

- 对于相对简单的环境，按照通用的元数据管理标准建立一个集中式的元数据知识库。
- 对于比较复杂的环境，分别建立各部分的元数据管理系统，形成分布式元数据知识库，然后通过建立标准的元数据交换格式，实现元数据的集成管理。

11. 数仓如何确定主题域?

主题

主题是在较高层次上将数据进行综合、归类和分析利用的一个抽象概念，每一个主题基本对应一个宏观的分析领域。在逻辑意义上，它是对企业中某一宏观分析领域所涉及的分析对象。

面向主题的数据组织方式，就是在较高层次上对分析对象数据的一个完整并且一致的描述，能刻画各个分析对象所涉及的企业各项数据，以及数据之间的联系。

主题是根据分析的要求来确定的。

主题域

从数据角度看（集合论）

主题语通常是联系较为紧密的数据主题的集合。可以根据业务的关注点，将这些数据主题划分到不同的主题域。主题域的确定由最终用户和数仓设计人员共同完成。

从需要建设的数仓主题看（边界论）

主题域是对某个主题进行分析后确定的主题的边界。

数仓建设过程中，需要对主题进行分析，确定主题所涉及到的表、字段、维度等界限。

确定主题内容

数仓主题定义好以后，数仓中的逻辑模型也就基本成形了，需要在主题的逻辑关系中列出属性和系统相关行为。此阶段需要定义好数据仓库的存储结构，向主题模型中添加所需要的信息和能充分代表主题的属性组。

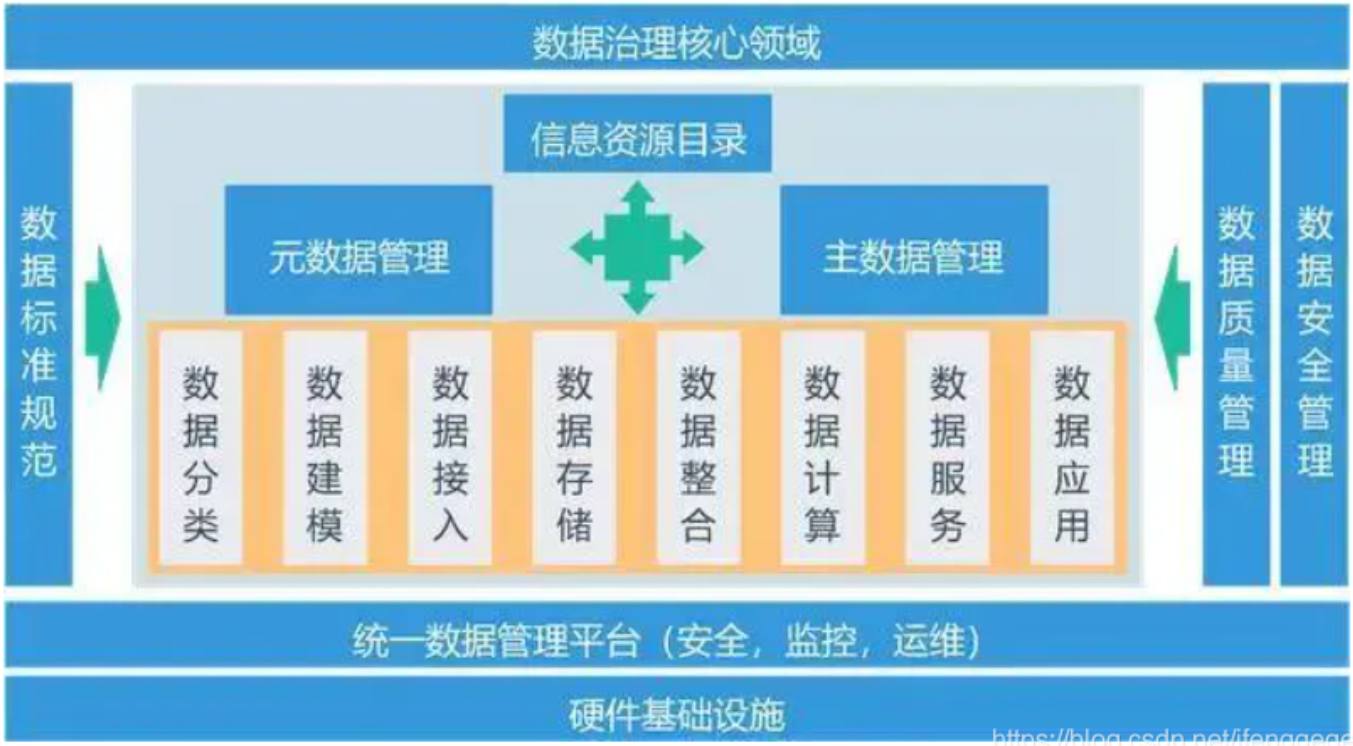
12. 如何控制数据质量?

- 校验机制：每天进行数据量的比对 select count(*)，早发现，早修复。
- 数据内容的比对，抽样比对。
- 复盘、每月做一次全量。

13. 如何做数据治理?

数据治理不仅需要完善的保障机制，还需要理解具体的治理内容，比如数据应该怎么进行规范，元数据该怎么来管理，每个过程需要那些系统或者工具来配合?

数据治理领域包括但不限于以下内容：数据标准、元数据、数据模型、数据分布、数据存储、数据交换、数据声明周期管理、数据质量、数据安全以及数据共享服务。



14. 数据质量管理

数据质量管理是对数据从计划、获取、存储、共享、维护、应用、消亡生命周期的每个阶段里可能引发的数据质量问题，进行识别、度量、监控、预警等，通过改善了提高组织的管理水平使数据质量进一步提高。

数据质量管理是一个集方法论、技术、业务和管理为一体的解决方案。放过有效的数据质量控制手段，进行数据的管理和控制，消除数据质量问题，从而提高企业数据变现的能力。

会遇到的数据质量问题：数据真实性、数据准确性、数据一致性、数据完整性、数据唯一性、数据关联性、数据及时性。

15. 数仓架构为什么要分层?

- 分层可以清晰数据结构，使用时更好的定位和理解。
- 方便追踪数据的血缘关系。
- 规范数据分层，可以开发一些通用的中间层数据，能够减少极大的重复计算。
- 把复杂问题简单化。
- 屏蔽原始数据的异常，不必改一次业务就重新接入数据。

注：资料来源于网络。