数据仓库与联机分析处理技术

北京大学信息科学技术学院

童云海

2005年5月

第五章　数据仓库相关技术

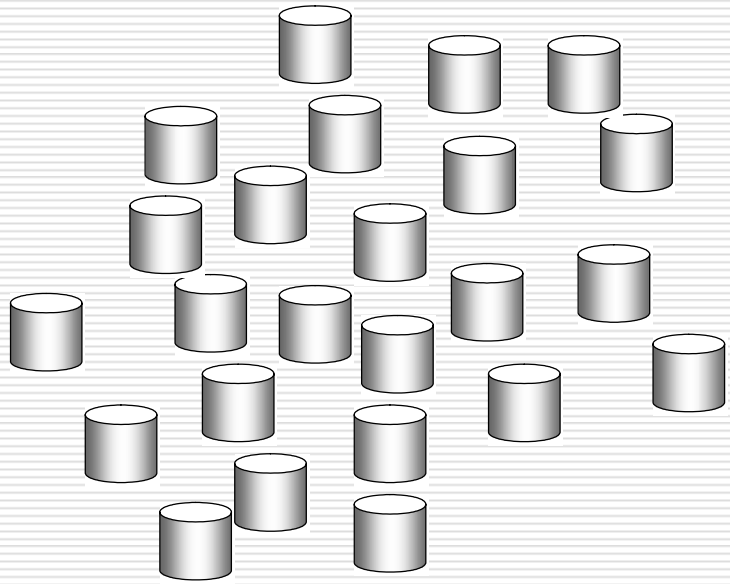
- 数据仓库技术
  - 支持数据仓库技术的基本需求
  - 支持数据仓库技术的其它需求
- 分布式数据仓库

支持数据仓库技术的基本需求

- 管理大量数据的能力
- 能够管理多种介质
- 能够轻松容易地索引和监视数据
- 用各种不同技术接收和传送数据的接口

管理大量数据

- TB和PB级的数量
  - 细节数据、粒度数据
  - 历史数据
- 需要多种大数据量管理办法
  - 通过灵活的寻址能力
  - 通过索引
  - 通过有效的溢出管理把不活跃数据移入到溢出存储器
- 既要考虑管理大数据量的能力和效率，还要考虑存储和处理的代价

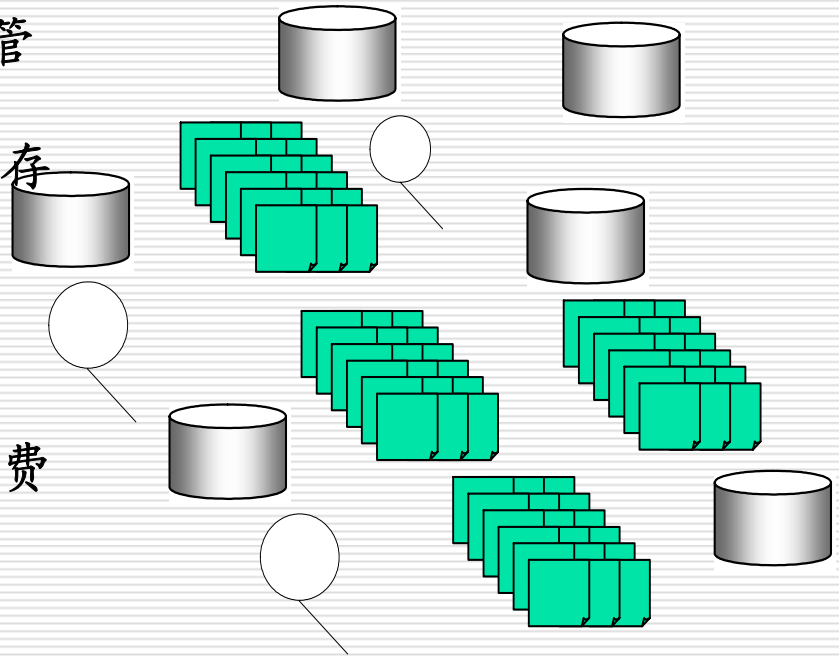


管理多种介质

- 仅仅在直接存取存储设备（DASD）上管理是不够的
- 一个满载的数据仓库应该有多种层次的存储设备

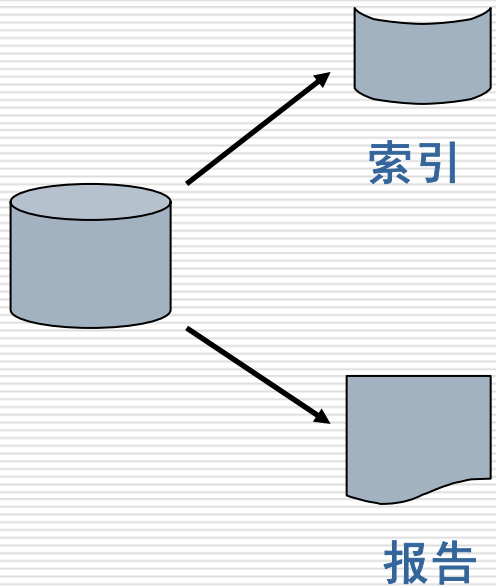
  理由:
  - 数据的容量不同
  - 数据被访问概率不同
- 各种层次级别的存储设备的存取速度和费用情况

主存	非常快	非常贵
扩展内存	非常快	贵
高速缓存	非常快	贵
DASD	快	适中
磁带	不快	不贵
光盘	不慢	不贵
缩微胶片	慢	便宜



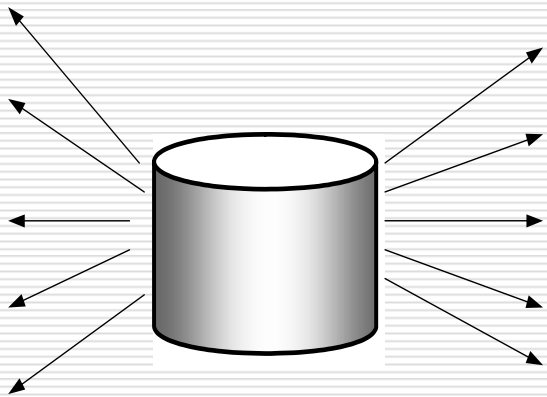
索引/监控数据

- 索引技术：满足快速灵活的数据访问
  - 支持方便的索引，如二级索引、稀疏索引、动态索引、临时索引等。
  - 索引的费用不能太高
- 监视数据：数据能被随意的监视
  - 监视的费用不能太高，过程不能过于复杂，可以随时运行
  - 通过监视数据的使用情况：
    - 考虑数据是否应该重组
    - 考虑索引的建立是否合适
    - 考虑综合数据是否合适
    - 根据数据的增长情况，考虑存储空间的分配



多种技术的接口

- 利用多种技术接收和传送数据
  - 接收：操作型环境、ODS→数据仓库
  - 传送：数据仓库→数据集市、DSS应用、数据挖掘环境等
- 高效、方便的接口，并可以在批处理模式下运行
- 不同技术接口要考虑的因素
  - 数据能否很容易地从一个DBMS传送到另一个DBMS?
  - 数据能否很容易地从一个操作系统传送到另一个操作系统?
  - 数据是否需要在传送过程转换它的基本格式（EBCDIC、ASCII等）



支持数据仓库技术的其它需求

- 数据存放位置控制
- 数据的并行存储/管理
- 元数据管理
- 语言接口
- 数据的有效装载
- 有效利用索引

支持数据仓库技术的其它需求（续）

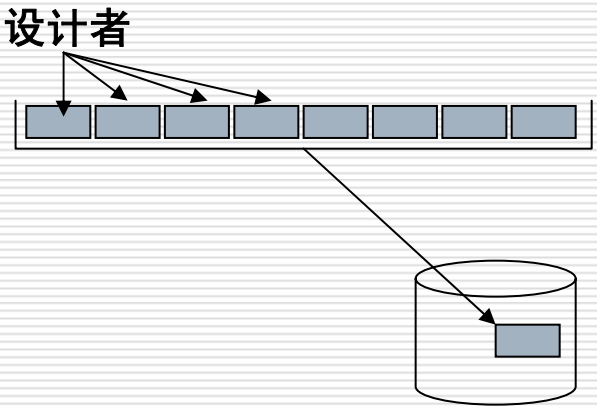
- 数据压缩
- 复合关键字
- 变长数据
- 加锁管理
- 单独索引处理
- 快速恢复

数据存放位置控制

- 允许设计者/开发者对数据存放位置进行控制

  控制到物理的块/页一级

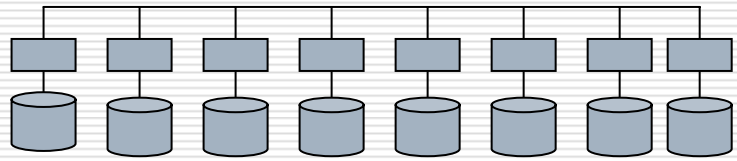
- 设计者/开发者能经常调整数据的物理存储位置，使之适合其用途，提高资源使用率



数据存放位置的控制

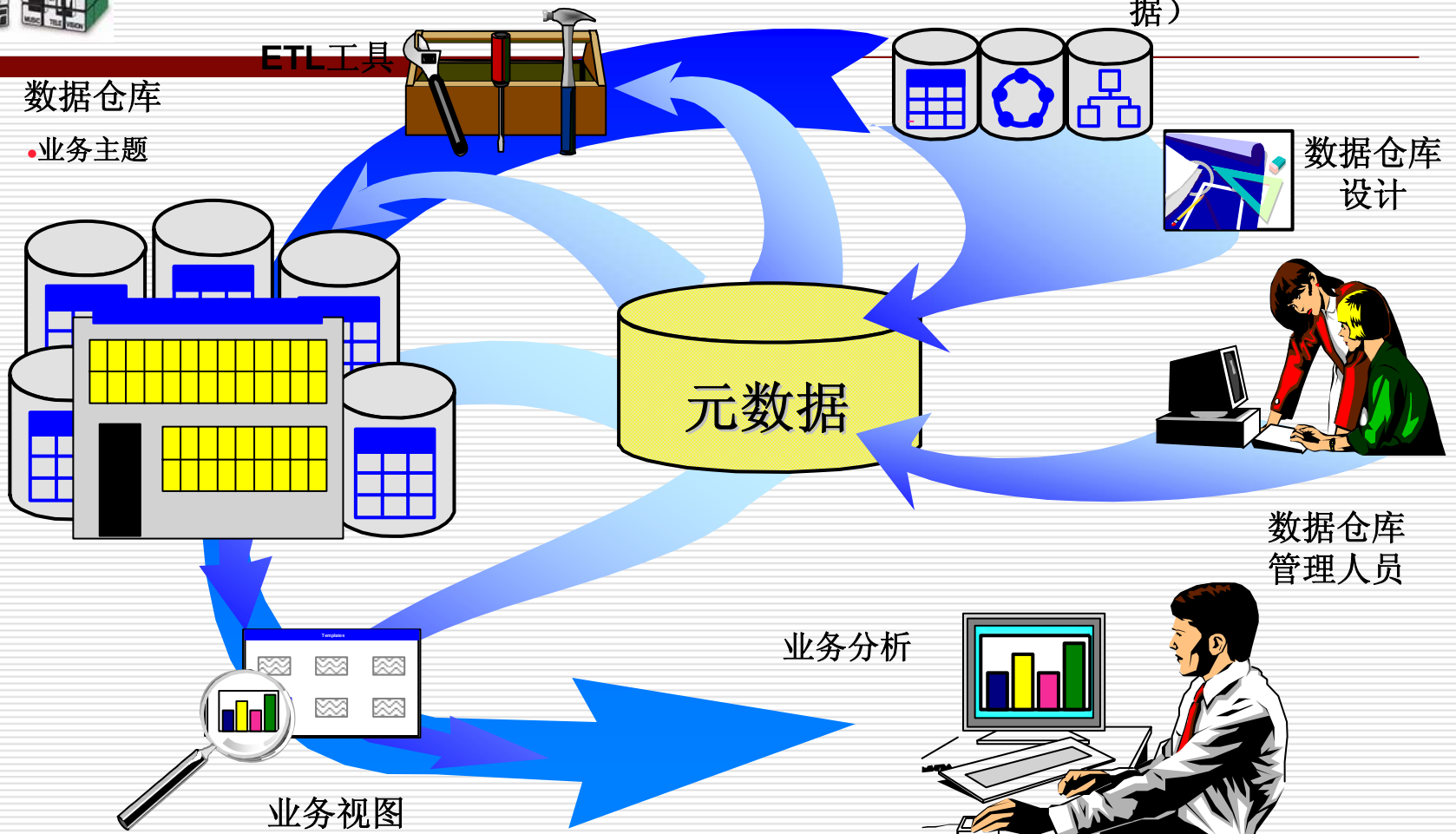
数据的并行存储/管理

- 并行存储和管理，可大大提高性能
- 性能的提高与数据所分布的物理设备的多少有关



并行管理数据

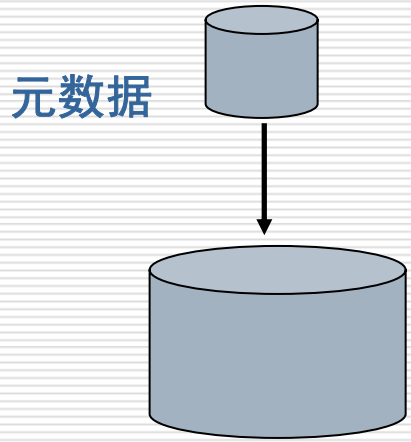
元数据管理



元数据的提取、修改和共享
贯穿整个数据仓库建设和使用过程中

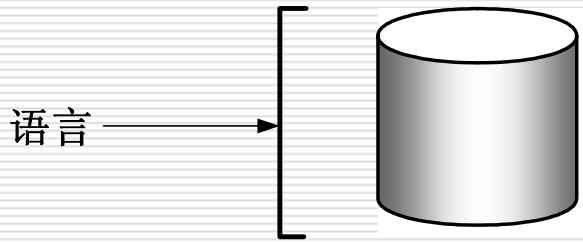
元数据管理（续）

- 元数据管理
  - 支持用户能访问到准确和最新的元数据
  - 典型的元数据包括:
    - 数据仓库的表结构
    - 数据仓库的表属性
    - 数据仓库的源数据（记录系统）
    - 从记录系统到数据仓库的映射
    - 数据模型的说明
    - 抽取日志
    - 数据的定义和描述
    - 数据单元之间的关系



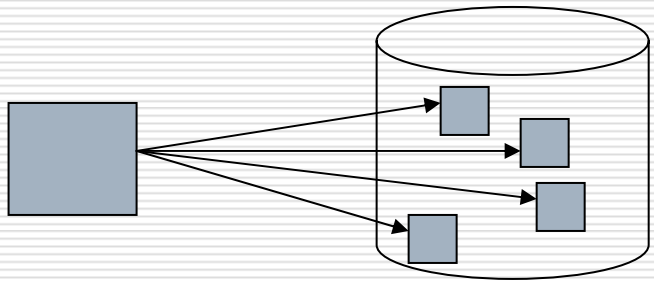
语言接口

- 需要有非常丰富的语言说明
- 访问数据仓库的语言应该是高效、易于操作和健壮的
- 能够一次访问一组数据
- 能够一次访问一条记录
- 能够支持一个或多个索引
- 有SQL接口
- 能够插入、删除、更新数据



数据的有效装载

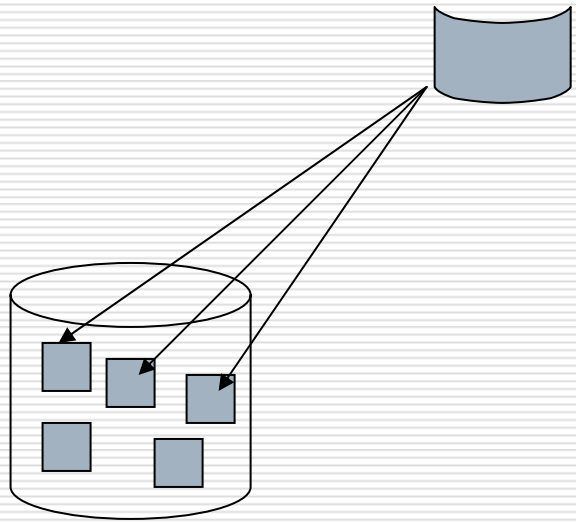
- 数据装入有两种基本方式
  - 通过一个语言接口一次装入一条记录
  - 通过一种工具一次全部批量装入，通过工具装入比较快
- 索引装入
  - 装入数据的同时，高效的装入索引
  - 为了平衡工作负载，数据索引的装入可以在数据装入后进行
- 并行装载
  - 被装载的数据被分成几个工作流，每个工作流独立运行，提高了装载速度
- 装载前进行缓冲处理
  - 当转换和集成的复杂性很高时，把被抽取的数据在转换、装载前集中放入缓冲区，在缓冲区中进行数据合并、编辑和汇总，然后再转换、装载



高效的数据装入

有效利用索引

- 不仅支持新索引的创建和装入，而且能高效的访问这些索引
- 高效访问索引的方法
  - 将部分或全部索引装入内存
  - 对索引项进行压缩
  - 用位映像的方法
  - 用多级索引
  - 创建选择索引或范围索引



高效的索引技术

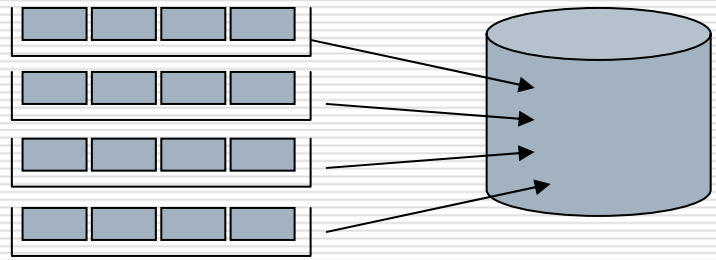
数据压缩

- 管理大量的数据的重要手段，减少存储空间

  数据仓库中数据的稳定性，有利于数据压缩技术的使用

- 数据以压缩形式存储时，程序员可以充分发挥I/O资源的效率
- 解压缩需要一定的开销，但这个开销不是I/O资源的开销，而是CPU的开销

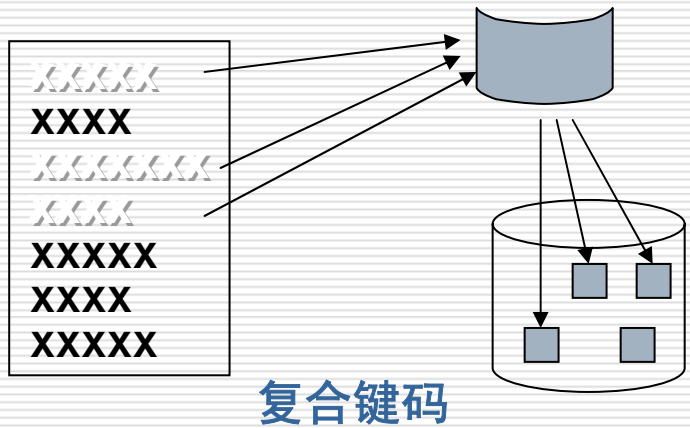
  在数据仓库环境中，I/O资源比CPU资源少得多



数据压缩技术

复合关键字

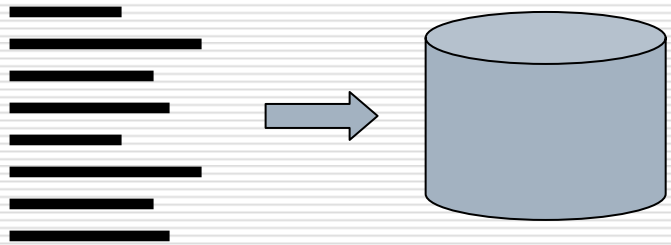
- 支持复合关键字，主要因为数据仓库中的数据均包含时间特性，以及数据中经常出现码/外码关系



变长数据

- 数据仓库数据的多样性，对数据的变长结构的支持是必须的
- 有效管理变长数据的能力

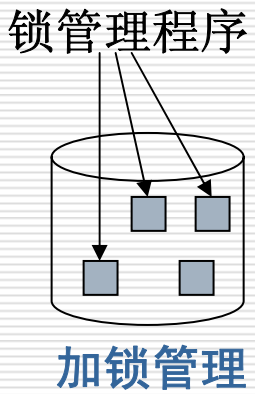
  数据仓库中，变长数据很稳定（即不更新），因而不会产生固有的性能问题



支持变长数据

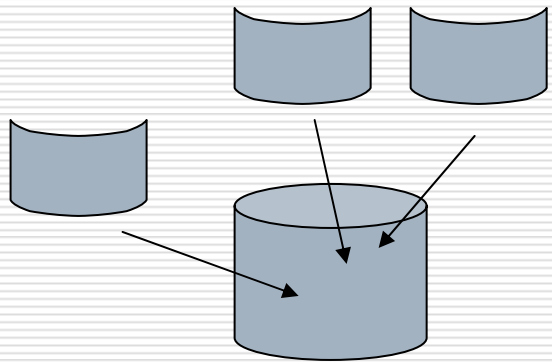
加锁管理

- 加锁管理程序需要消耗相当的资源，其目的是为了确保没有两个用户同时更新同一记录
- 数据仓库中，没有更新操作，为了节省资源，需要有选择地将加锁管理程序打开或关闭，能够在程序员级显式控制锁管理程序



只涉及索引的处理

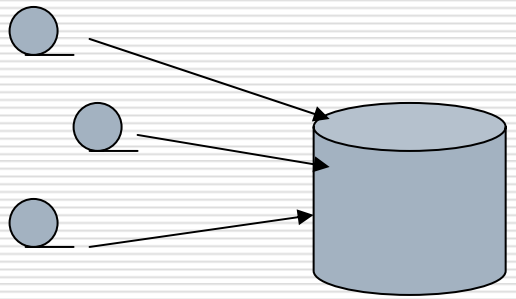
- 只通过索引就可以满足某些请求时，可以只利用这些索引提供相应服务，提高效率



进行单独索引处理

快速恢复

- 可以从非直接存取设备中快速恢复数据仓库表，从而能够节约大量开支
- 如果没有能从二级存储设备上快速恢复的能力，通常的做法是将DASD的数目增加一倍
- 系统需要以一种自动的模式去侦测发生的错误；需要有创建诊断工具的能力，用以判断哪些数据已经被破坏



数据的快速、完全地恢复

其它的技术特征

- 传统数据库管理系统中的部分技术，在数据仓库系统中只起到很小的作用
  - 事务集成性
  - 高速缓存
  - 行/页级的锁定
  - 参照完整性
  - 数据视图

数据仓库专用DBMS

- 数据仓库专用数据库管理系统是特别为数据仓库和决策支持而优化设计的
- 数据仓库专用数据库管理系统与普通数据库管理系统之间的比较

名称	数据仓库专用DBMS	传统DBMS
处理类型	装载和访问，不更新	事务处理，更新，开销巨大
数据量	TB或PB	少得多
数据管理	不需要自由空间	需要自由空间，可能占50％
索引	可以有多种索引	有限数量的索引
组织数据	针对DSS访问和分析进行优化	针对事务访问进行优化

改变DBMS技术

- DBMS技术变化的原因
  - 目前可用的DBMS技术，在数据仓库首次载入数据时并不一定适合
  - 数据仓库的大小已经增长到一定的程度，需要采用新的技术途径
  - 现有数据仓库的DBMS技术已经不能满足当前的需要
  - 需要不时地对基本的DBMS选择进行审查

改变DBMS技术（续）

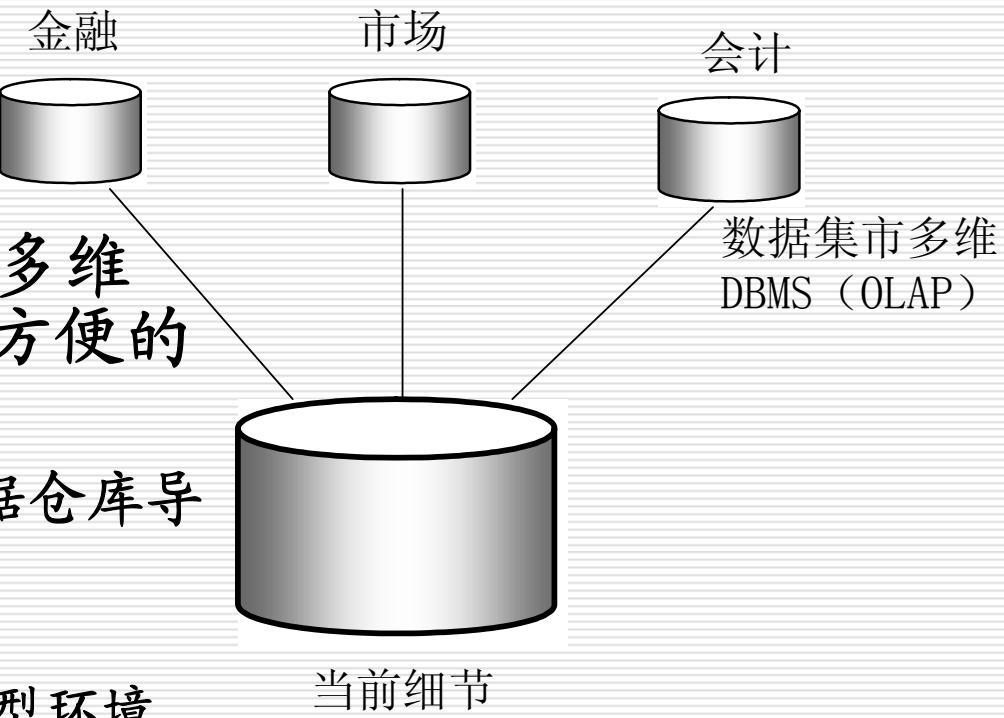
- 决定采用新的DBMS技术需要重点考虑的问题：
  - 新的DBMS技术能否满足可预知的需求？
  - 从旧的DBMS技术向新的DBMS技术的转变应该怎样去做？
  - 转换程序应该如何改变？

多维DBMS和数据仓库

- 多维DBMS提供了一种信息系统结构
- 该结构可以使企业灵活地访问数据

  切片、切块、上钻、下钻
- 多维DBMS为终端用户提供了灵活性和控制功能
- 非常适合DSS环境

多维DBMS和数据仓库（续）



- 数据仓库中的细节数据为多维DBMS提供了非常健壮和方便的数据源
  - 定期刷新，数据定期从数据仓库导入到多维DBMS中
  - 数据已是集成好的

    多维DBMS不再需要从操作型环境抽取和集成数据
  - 最低级别上保存数据

    为低级别的分析提供基础数据

数据仓库与多维DBMS的区别

一般来说，多维DBMS技术并不用于数据仓库，数据仓库中的最重要的特性也不是多维DBMS技术特性

- 数据仓库有大量的数据；多维DBMS中的数据至少要少一个数量级
- 数据仓库只适于少量的灵活访问；而多维DBMS适合大量的，不可预测的数据访问和分析
- 数据仓库内存储了很长时间范围内的数据—从5年到10年；而多维DBMS中只存储较短时间范围内的数据
- 数据仓库只允许分析人员以受限的形式访问；而多维DBMS允许自由的访问
- 多维DBMS和数据仓库有着互补关系，但数据仓库并不建立在多维DBMS之上

数据仓库与多维DBMS的互补

- 数据仓库可以为非常细节的数据提供基础，多维DBMS提供不同层次的综合数据
- 数据仓库和多维DBMS相结合，使得分析者可以对多维DBMS中不同层次的数据进行钻取，而且还可以向下钻取到数据仓库
- 汇总的信息在多维DBMS中计算和聚集后被存储在数据仓库中，这样，汇总数据在数据仓库中能比在多维DBMS中存储更长的时间
- 多维DBMS存放中等时间长度的数据（一般为12—15个月），数据仓库存放5至10年的数据，因此多维DBMS减少了数据的存储代价，数据仓库成为多维DBMS的数据源
- 多维DBMS的两种数据基础
  - 多维DBMS建立在关系模型上—关系型技术基础
  - 多维DBMS的数据存储在多维立方体内—“立方体”（“CUBE”)技术基础

基于关系型技术的多维DBMS

- 优点
  - 能支持大量数据
  - 能支持数据的动态连接
  - 已被证实是有效的技术
  - 能够支持通用的数据更新处理
  - 如果不知道数据的使用模式，选择关系型结构比较好
- 缺点
  - 性能上不是最佳的
  - 不能对访问处理作完整的优化

基于“CUBE”技术的多维DBMS

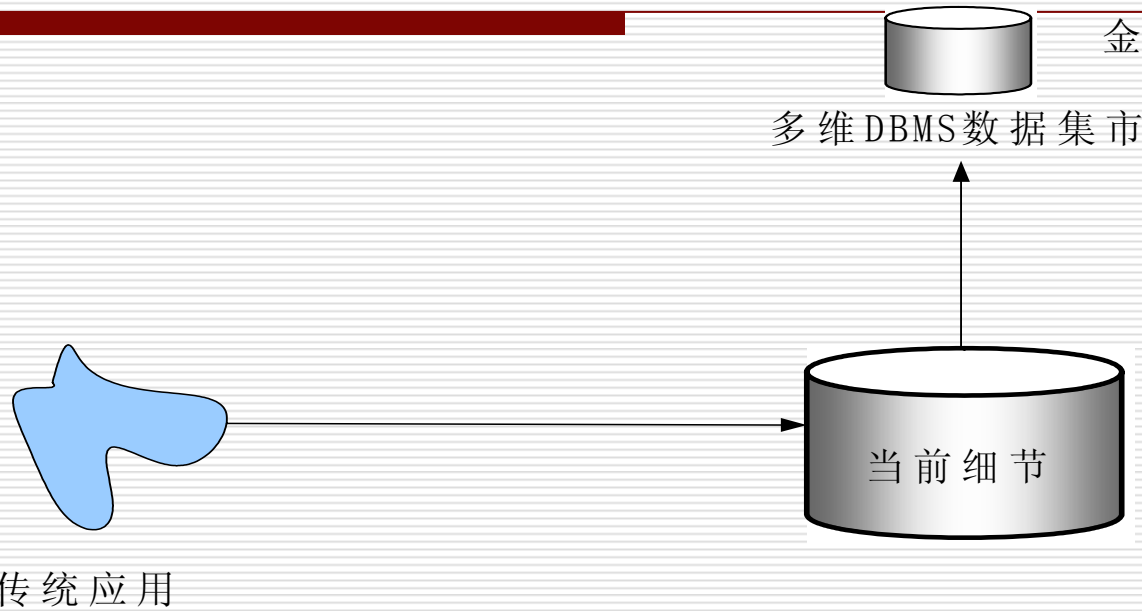
- 优点
  - 在性能上最适合于DSS处理
  - 能够对数据进行优化以满足快速存取要求
  - 如果已知数据访问模式，则数据的结构可以优化
  - 能够方便地进行“切片和切块”
  - 可以用许多方法进行检测
- 缺点
  - 无法处理像标准关系模式那么多的数据
  - 不支持通用的更新处理
  - 装载的时间很长
  - 如果对数据访问的路径不被数据设计所支持的话，这种结构就显得不灵活
  - 对数据的动态连接的支持是有问题的

数据直接从历史环境装入多维DBMS

一般情况，数据仓库作为流入多维**DBMS**数据的基础，选择细节数据的子集给多维**DBMS**，在其中，对数据进行汇总和聚集，但有观点认为：多维**DBMS**不需要数据仓库作为它的数据基础

- 数据直接从旧的、历史环境中得到
- 直接、容易实现
- 数据没有被集成，并且容易形成“蜘蛛网”

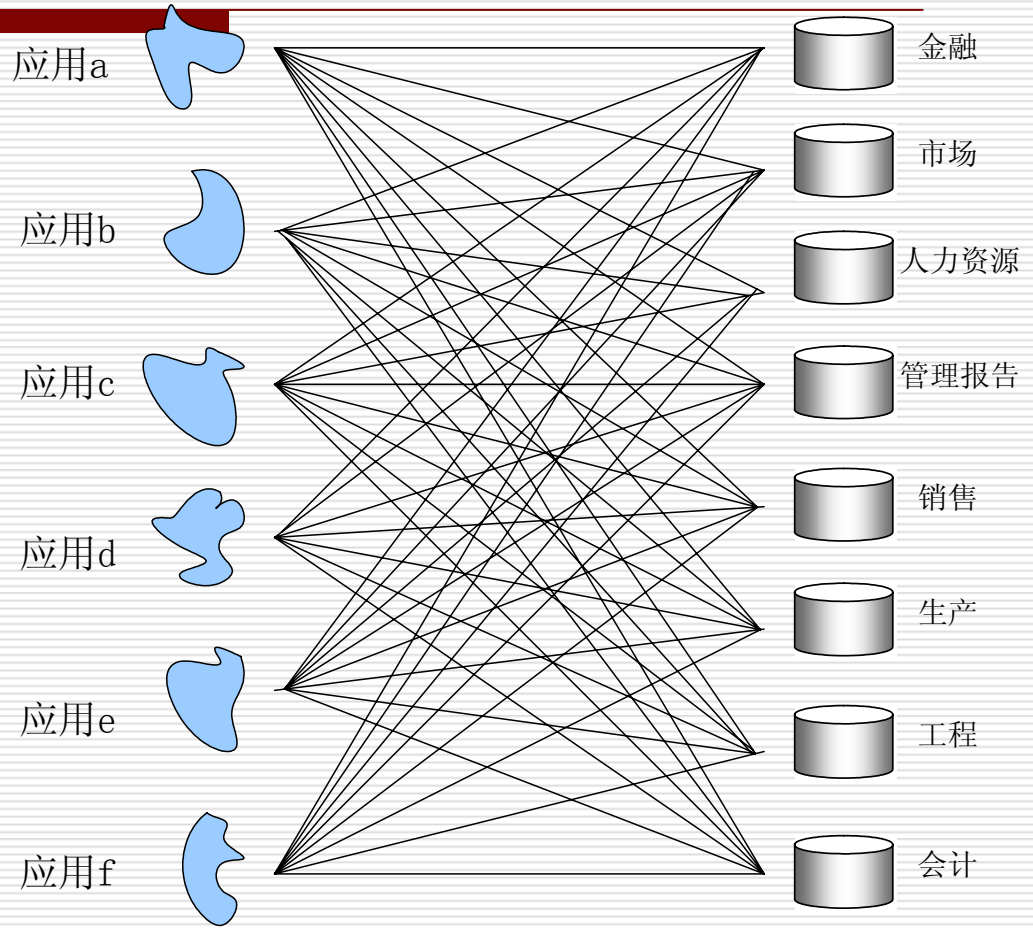
数据从数据仓库装入多维DBMS



- 用数据仓库当前细节级的数据为多维DBMS环境提供数据
- 数据仓库中的数据已进行了集成和转换
- 建立数据仓库需要投入精力，但是是值得的
- 长期来看，这种方式具有更高的性价比

数据直接从历史环境装入带来的问题(I)

- 有许多的应用和许多的数据集市，每对之间都需要一个接口，回避当前细节数据的后果是产生无法管理的“蜘蛛网”



众多的多维DBMS都要直接而独立地从历史应用中获得数据

数据直接从历史环境装入带来的问题(II)

- 抽取数据所需进行的开发量是巨大的
  - 每一个不同部门的多维DBMS都需要开发一套适合它的抽取程序
  - 抽取处理过程有大量的重叠
  - 浪费的开发工作量很大
- 当多维DBMS是从数据仓库抽取数据时，只要一套集成和转换的程序就够了

数据直接从历史环境装入带来的问题(III)

- 抽取数据没有集成基础
  - 每一个不同部门的多维DBMS都需要从不同的应用中集成自己的数据
  - 不同部门之间集成相同数据的方法不同
  - 最终没有单一集成的确定的数据源
- 数据仓库本来就是一个单一的、集成的、确定的数据源

数据直接从历史环境装入带来的问题(IV)

- 为了维护所进行的开发工作量是巨大的
  - 一个旧地历史应用的改变就会影响许多抽取程序
  - 在有抽取程序的地方都要做相应的改动
- 数据仓库抽取数据由于只需要很少的程序来处理传统环境和数据仓库的接口，所以应用中的改变产生的影响很小

数据直接从历史环境装入带来的问题(V)

- 需要消耗大量的硬件资源
  - 对每一个部门的每一个抽取处理，同样的历史数据都要顺序地重复传送
- 数据仓库中，为了刷新数据仓库中的数据，传送数据只需要传送一次

数据直接从历史环境装入带来的问题(VI)

- 从历史环境中将数据直接导入多维DBMS环境中的复杂度使得无法对元数据进行有效管理和控制
- 数据仓库中可对元数据进行直接的捕捉和管理

数据直接从历史环境装入带来的问题(VII)

- 缺乏数据的一致性
  - 当不同部门存在意见上的分歧时，由于各自有自己的多维DBMS，冲突难以解决
- 数据仓库中，冲突的解决是自然并且很容易的

数据直接从历史环境装入带来的问题(VIII)

- 每次构建一个新的多维DBMS，必须根据历史环境建立，需要大量的工作
- 数据仓库中新建一个多维DBMS环境将会快速而容易

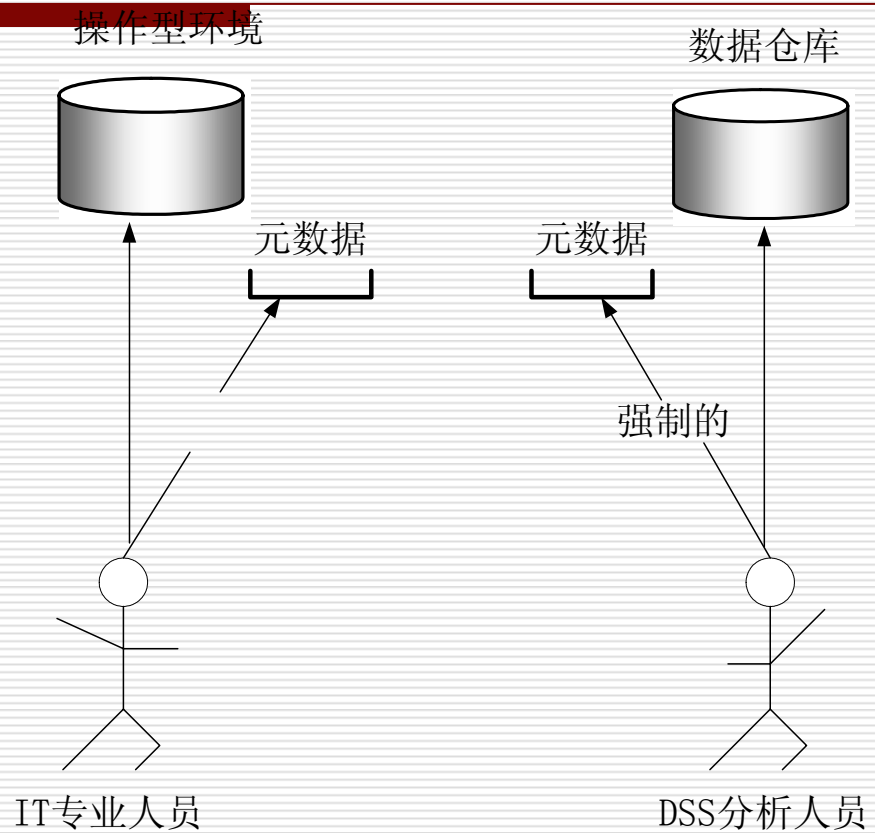
因此，从长期观点出发，采用数据仓库地方法，会大大降低总费用

在多种存储介质上构建数据仓库

- DASD环境：能够进行在线、交互式处理
- 大容量存储环境：磁带处理或其它大容量存储环境
- 逻辑上，两种环境结合在一起构成了一个统一的数据仓库
- 物理上，两种环境具有很大的区别
- 支持DASD环境的底层技术和支持大容量存储环境的底层技术是不同的，因此，采用混合技术是自然和正常的
- 将数据仓库的不同部分分散存储在不同厂商的平台上是不可取的（除非是分布式数据仓库）

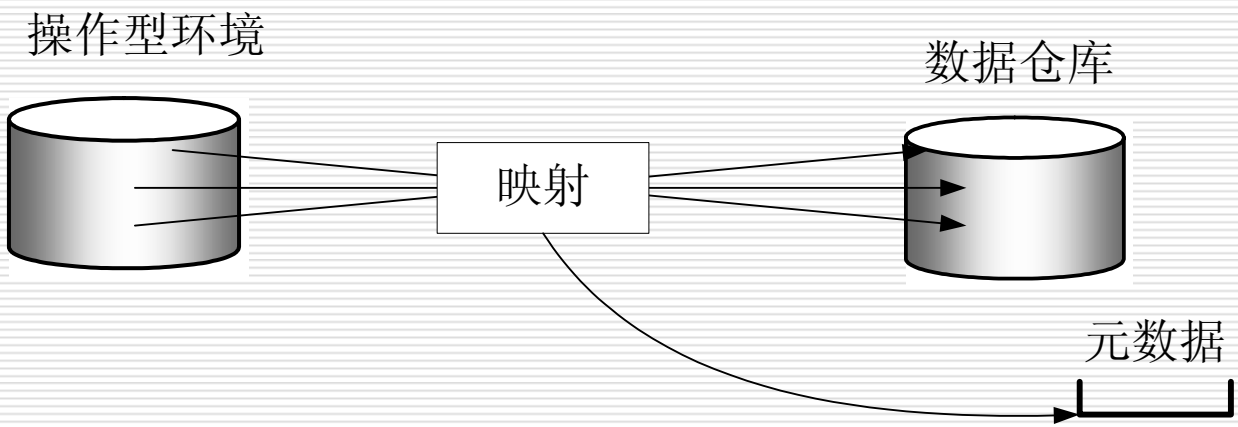
操作型和数据仓库环境元数据比较（I）

- 操作型环境中
  - 元数据的重要性与文档相同，往往成为事后补记
  - 操作型数据由IT专业人员使用，他们会在系统中找到自己去的地方
- 数据仓库环境中
  - 元数据很重要
  - 数据仓库数据由DSS分析者使用，他们需要尽量多的帮助



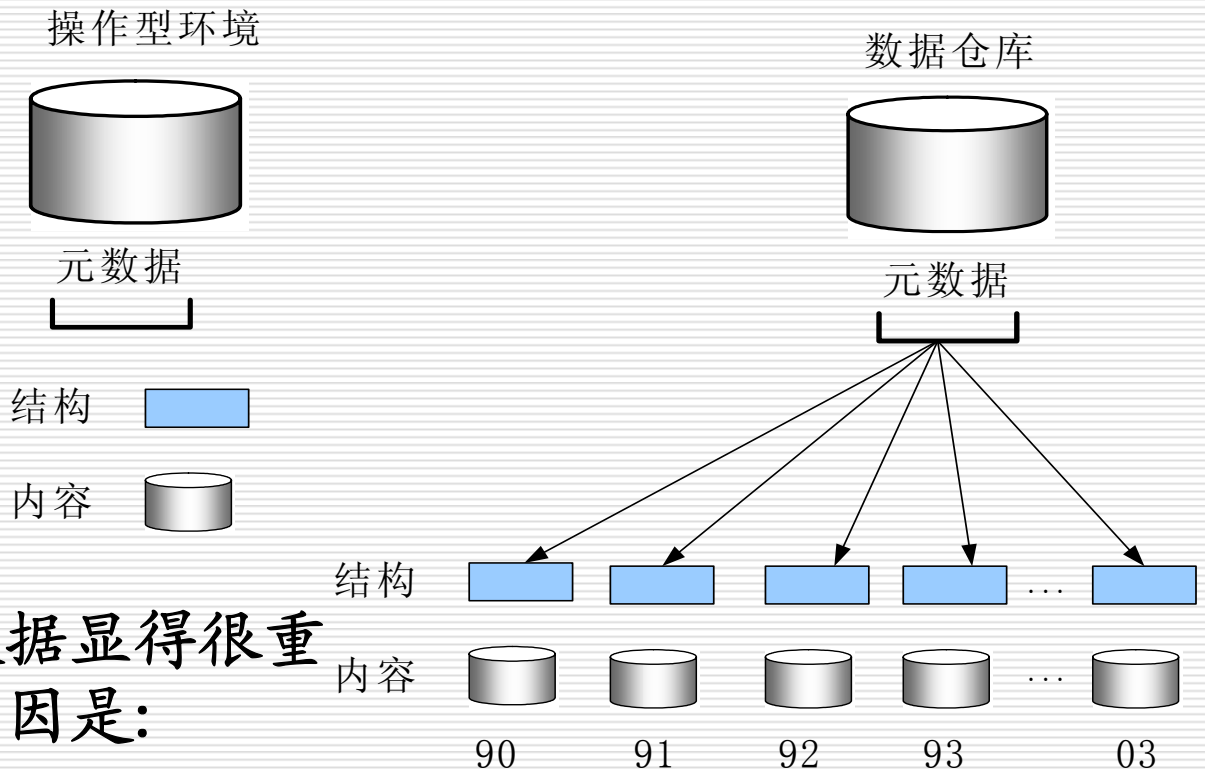
IT专业人员偶尔使用元数据，DSS分析人员经常使用元数据并作为分析的第一步

操作型和数据仓库环境元数据比较（II）



- 操作型环境和数据仓库环境之间的映射是使元数据显得很重要的另一个主要原因：没有这种映射，对接口进行控制是非常困难的
  - 数据从操作型环境传入数据仓库时要经历转换
    - 会发生转换、过滤、汇总、结构改变等
    - 对转换必须进行跟踪，元数据进行这项工作
  - 元数据支持了向下钻取，数据转换的记录描述了怎样从数据仓库钻取到操作型环境的源数据

操作型和数据仓库环境元数据比较（III）



数据仓库元数据显得很重要的另一个原因是:

- 数据仓库中包含很长一段时间的数据，因此必须管理多种数据结构/定义。操作型环境在任一时刻一般只有唯一的一个正确的数据定义

上下文和内容

- 操作型环境主要关心企业的当前数据上，例如:此刻帐目的余额是多少、存货有多少等
- 数据仓库的一个重要特征就是能够对随时间变化的数据进行存储、管理和访问
- 上下文维—由于足够长的时间范围是数据仓库本身的一个特征
- 如果随着时间的变化数据的内容毫无附加信息，那么内容本身也就非常难于解释和难于令人置信的
- 随着时间的变化，把上下文加到数据的内容上，内容和上下文都变得非常明了
- 为了能随着时间的变化解释和理解信息，需要一个全新的上下文维
- 信息的内容固然重要，但为了理解随时间变化的信息，上下文和内容具有同等的重要性

上下文和内容：举例

- 管理者从数据仓库得到一份1995年的报表，同时又要了一份1990年的报表，95年财政报告显示收入为$50,000,000，而90年显示为$10,000。管理者断言这些报表肯定存在问题，他说任何帐户或科目都不可能在过去5年时间内就增长这么多
- 数据仓库工程师向管理者指出，还有一些相关因素没有在报表中体现出来：
  - 1990年和1995年的数据是从不同来源得到的
  - 1990年和1995年的产品定义不同
  - 1990年和1995年有不同的市场范围
  - 1990年和1995年有不同的计算方法，如针对贬值问题
  - 许多不同的外部因素，如在通货膨胀、税款、经济预测方面的差异
- 当向管理者解释了报表的上下文情况之后，这种内容在相当程度上可以接受

上下文信息的三种类型（I）

- 简单上下文信息
- 复杂上下文信息
- 外部上下文信息
- 简单上下文信息—与数据本身的基本结构有关
  - 数据的结构
  - 数据的编码
  - 数据的命名习惯
  - 描述数据的度量
    - 数据量有多少
    - 数据增长速度
    - 数据的哪一部分在增长
    - 数据的使用情况
- 简单上下文信息用字典、目录、系统监视器等进行管理

上下文信息的三种类型（II）

- 复杂上下文信息—描述的是和简单上下文信息相同的数据，但从一个不同侧面进行描述，如：
  - 产品定义
  - 市场范围
  - 定价
  - 包装
  - 组织结构
  - 配送
- 复杂上下文信息在理解和解释跨时间段信息方面有着非常重要的作用

上下文信息的三种类型（III）

- 外部上下文信息—处于企业之外，在理解随时间变化的信息方面起重要作用，如:
  - 经济预测
    - 通货膨胀
    - 金融
    - 税务
    - 经济增长
  - 政治信息
  - 竞争信息
  - 技术进展
  - 用户人数的统计变动
- 指出了企业运转和竞争中所处的大环境中的任何事情

捕获和管理上下文信息

- 复杂上下文信息和外部上下文信息由于是非结构化信息，显得非常杂乱无章，并且上下文信息变化很快，所以很难捕获与确定，很难系统化
- 上下文信息的管理历史
  - 数据字典、知识库、目录和索引库都是用来管理简单上下文信息的尝试
  - 但这种方式存在以下局限:
    - 信息管理的目的是针对信息系统开发者，而不是最终用户
    - 这些上下文信息管理的尝试都是被动的，很多人回避使用这些上下文信息管理工具
    - 这些上下文信息管理的计划在很多情况下都会被从开发计划中删除掉
    - 这些上下文信息管理的尝试仅局限于简单上下文信息，并没有尝试去捕获或管理外部和复杂上下文信息

数据仓库刷新

- 数据仓库建好后，运作和维护费用很高
- 数据量的增长速度很快，与数据仓库运作有关的最大的、最不可预知的开销是根据历史数据的周期性刷新
- 刷新方法
  - 直接读取历史数据
  - 捕捉历史环境中正在被修改的数据
- 直接读取历史数据
  - 开销非常大
    - 在读取过程中，历史DBMS必须是在线和活动的，为刷新数据仓库需要将时间窗口扩大
    - 相同的历史数据没必要地被传送了好几次

    当只需1％的历史数据时，却需要100％地扫描整个历史文件

数据仓库刷新（续）

- 捕捉历史环境正在被修改的数据
  - 优点
    - 不需要对历史环境中的表进行整表扫描
    - 捕捉的数据可以以脱机方式处理
  - 两种技术
    - 数据复制
    - 变化数据捕获

数据复制

- 要捕获的数据在修改之前标识出来，当改变发生时，就能将数据捕获
- 需要设置一个触发器来捕获数据的更新活动
- 优点
  - 可以有选择的控制捕获处理的过程，只有需要捕获的数据才会被捕获到
  - 数据的格式“整洁”、定义完善，被捕获的数据的内容与结构都具有很好的文档说明易于被程序员理解
- 缺点
  - 产生许多额外的I/O操作
  - 由于数据仓库总在变化的特性，系统要不断的注意控制捕获过程的参数和触发器的定义
  - 所消耗的I/O都是在系统高性能运行时进行的

变化数据捕获（CDC）（I）

- 使用日志磁带来捕获和确定在在线过程中的变化，这种方法要读取日志或日志磁带，但读取日志磁带存在以下障碍:
  - 日志磁带包含许多无关数据
  - 日志磁带格式难于理解
  - 日志磁带包括跨区记录
  - 日志磁带通常包含的是数据的地址而并非它的值
  - 日志磁带反映了DBMS的特征，并随DBMS的不同而有很大不同
- 优点
  - 高效率，日志磁带的CDC不需要附加I/O操作（因为不管是否用于数据仓库刷新，日志磁带总是要写的）
  - 日志磁带会捕获所有的数据更新操作

变化数据捕获（CDC）（II）

- CDC的另一种方法是：当数据发生变化时，从DBMS的缓冲区提出已改变的数据
- 优点
  - 数据改变能得到立即的反映，无需读日志磁带，并节约从数据发生改变到改变被反映到数据仓库这段时间
  - 能够以非常高的速度处理大量的数据捕获
- 缺点
  - 需要更多的在线资源，包括系统软件对数据改变的敏感性，这种方法会给性能带来一定的冲击

刷新技术的发展进程

- 首先，从历史数据库直接读取数据
- 然后，尝试数据复制方法
- 最后，CDC作为刷新的主要方法
- 发现
  - 一些文件需要直接读取
  - 另一些文件适合于用复制方法
  - 但CDC是一种长期的、最终的数据仓库刷新方法

测试问题

- 传统操作型环境设置两个并行环境—一个用于生产，一个用于测试
- 数据仓库环境和传统的生产环境存在不同，因为在很多情况下测试环境完全不需要，同时也很难找到这样的测试环境：
  - 数据仓库是如此之大，公司很难测试
  - 数据仓库的开发生命周期是反复的，对大多数情况，程序是以启发式的模式来运行的，如果程序员在数据仓库环境做错了，在这种环境下，程序员只需要简单的重做一遍就可以了

第三章　数据仓库相关技术

3.1 数据仓库技术

3.2 分布式数据仓库

3.2.1 分布式数据仓库概述

3.2.2 分布式数据仓库的建立

集中式数据仓库环境比较流行

- 数据仓库中的数据是全企业范围集成的，而且只有总部才会使用集成的数据
- 公司运作的商务模式是集中式的
- 数据仓库中的数据量非常大，将数据集中存储在一个地方是较为妥当的
- 即使数据能被集成，但是若将它们分布于多个局部站点，那么存取这些数据也是很麻烦的

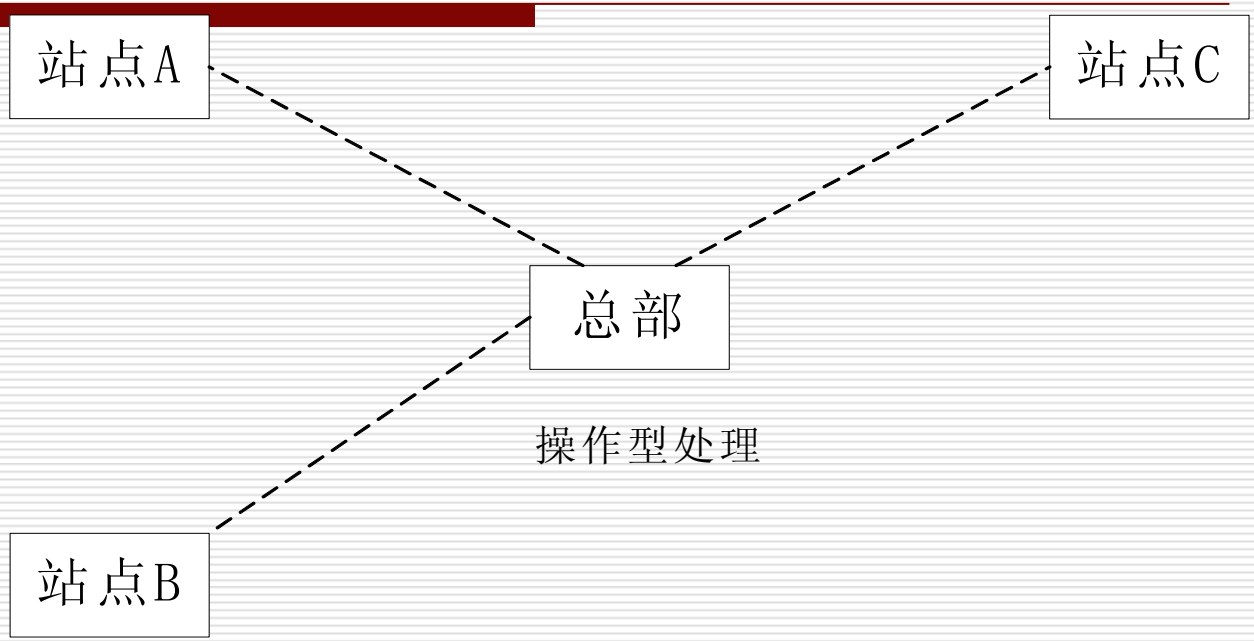
分布式数据仓库的类型

- 业务上分布的数据仓库
  - 业务是在不同地域或不同的生产线上进行的
  - 局部数据仓库：在远程站点上提供和处理数据
  - 全局数据仓库：提供在整个业务范围集成后的数据
- 技术上分布的数据仓库
  - 由多个处理器对数据仓库环境的大量数据进行处理
  - 逻辑上只有一个数据仓库，但从物理上看，存在着许多有紧密联系但分布在不同的处理器上的数据仓库
- 独立演进的分布式数据库
  - 数据仓库环境是以一种未统一规划方式建立起来的，首先建立某一个数据仓库，然后再独立的建立下一个数据仓库
  - 由于政策和机构上的差异，不同的数据仓库建立方式也不同，缺乏协调性

局部数据仓库和全局数据仓库

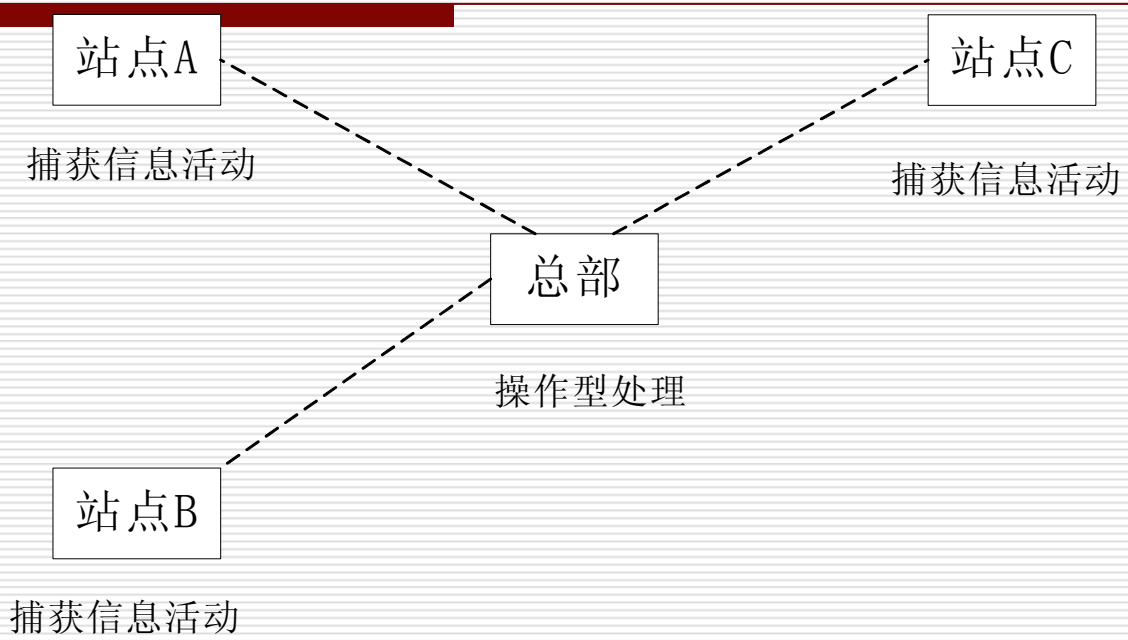
- 一个企业的业务遍及世界各地时，总部和分支机构都需要信息，数据以集中式和分布式两种方式存在
  - 中心数据仓库负责采集数据，同时可以满足总部对企业信息需求
  - 分布在不同国家的各个分支机构，仍然有建立数据仓库的需要
- 当一个大企业有许多不同的业务时，需要有局部/全局分布式数据仓库，不同业务的数据仓库将为一个企业集中式数据仓库提供 支持
  - 有的业务需要在企业层面上集成，如财务;
  - 不同范围内的业务可能存在着相当大的业务集成，如客户、产品、销售等

企业业务处理拓扑结构（1）



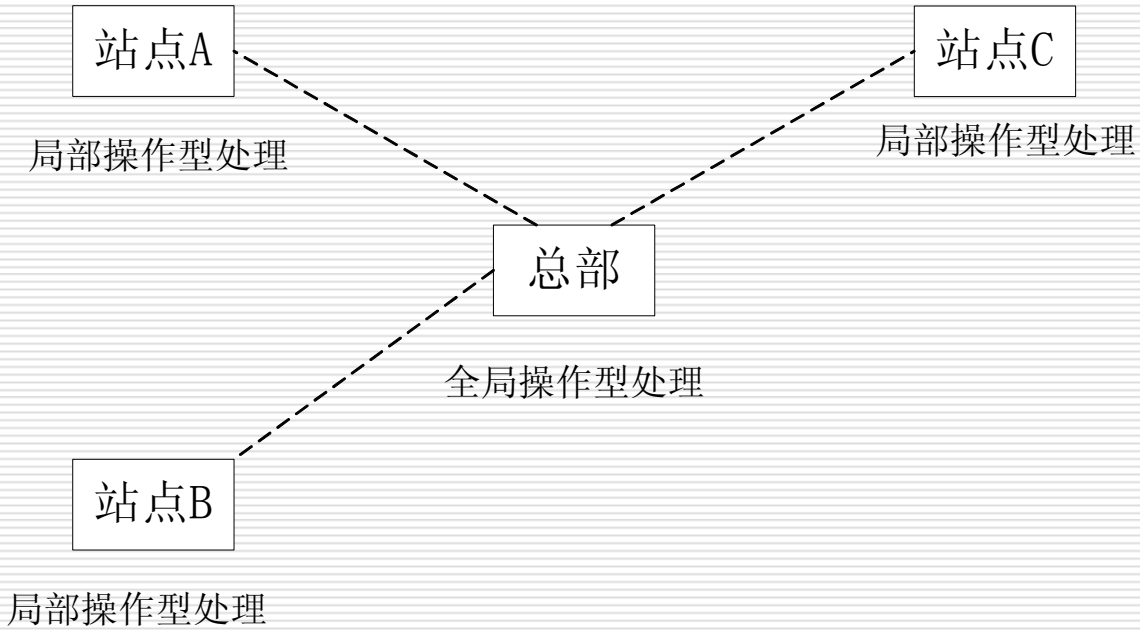
- 企业总部负责处理所有业务，分支机构层上业务处理非常简单，可能只是一些哑终端，没有必要建立分布式数据仓库环境

企业业务处理拓扑结构（2）



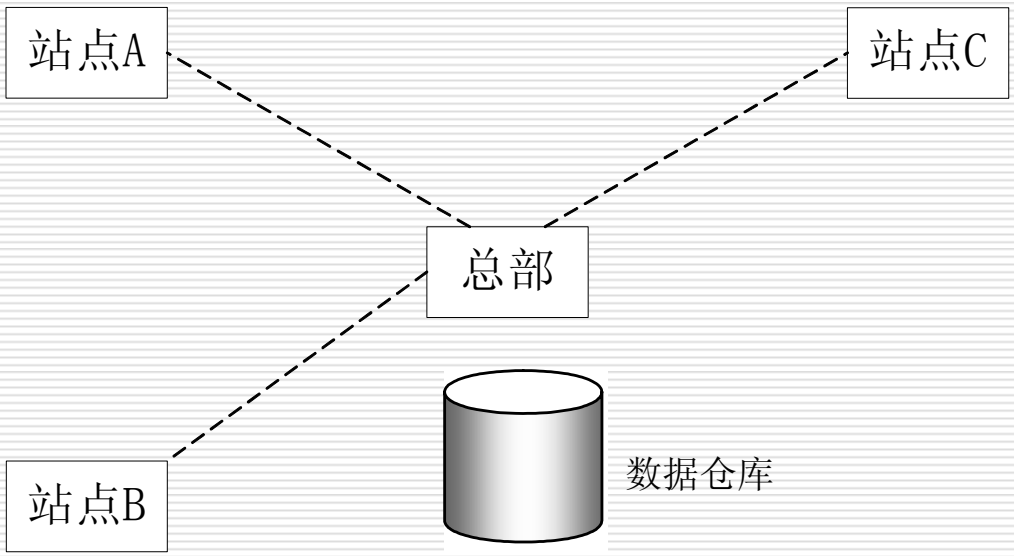
- 分支机构层出现简单数据和事务的获取，并把数据传送到总部进行进一步处理，但没有出现大量业务，分支机构层所做的决策不需要数据仓库，因此，也没必要建立分布式数据仓库环境

企业业务处理拓扑结构（3）



- 相当多的处理是在分支机构层进行的，就操作型处理而言，分支机构层是自主的，仅偶然或某些特定的处理需要将数据和业务活动发送到总部处理，总部存有一份集中的公司财务平衡表，对这类企业来说，采用某种形式的分布式数据仓库是必要的。

中心数据仓库

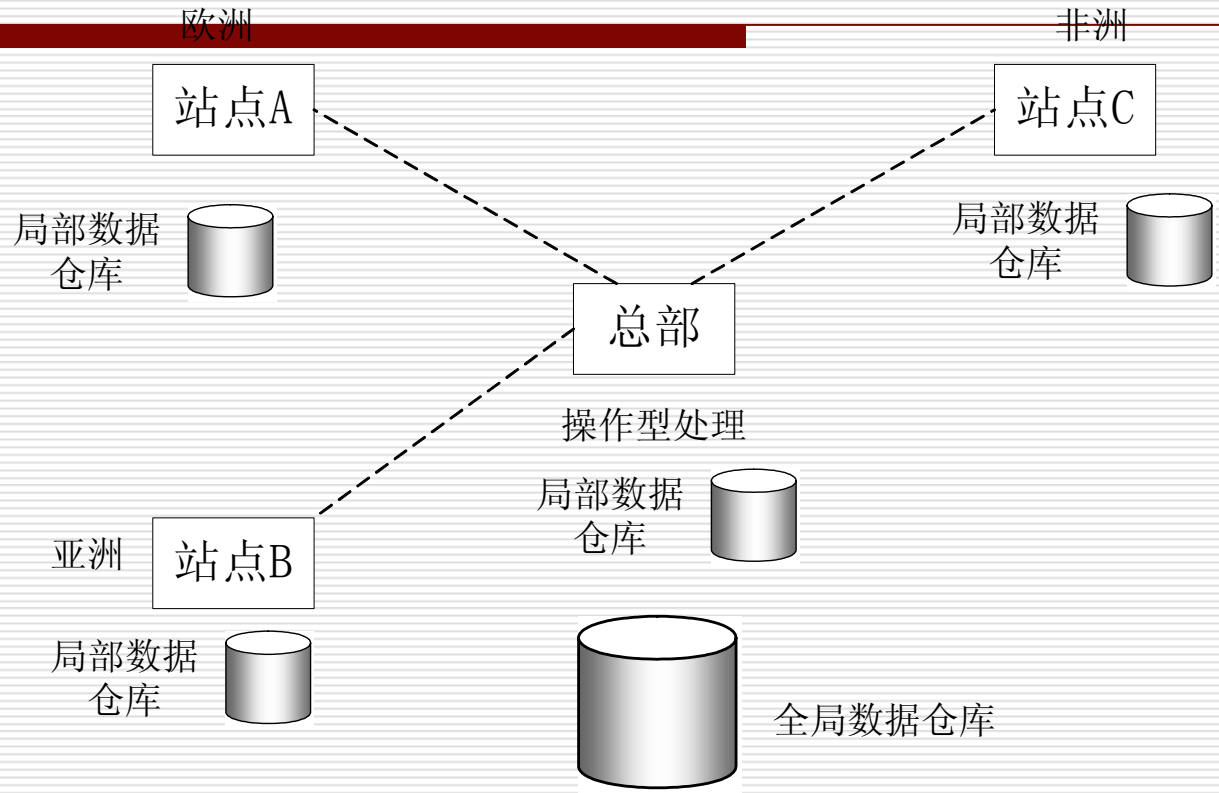


- 分支机构自主权不大，也没有大量的业务处理，这样的企业大都拥有一个中央控制和中央存储的数据仓库

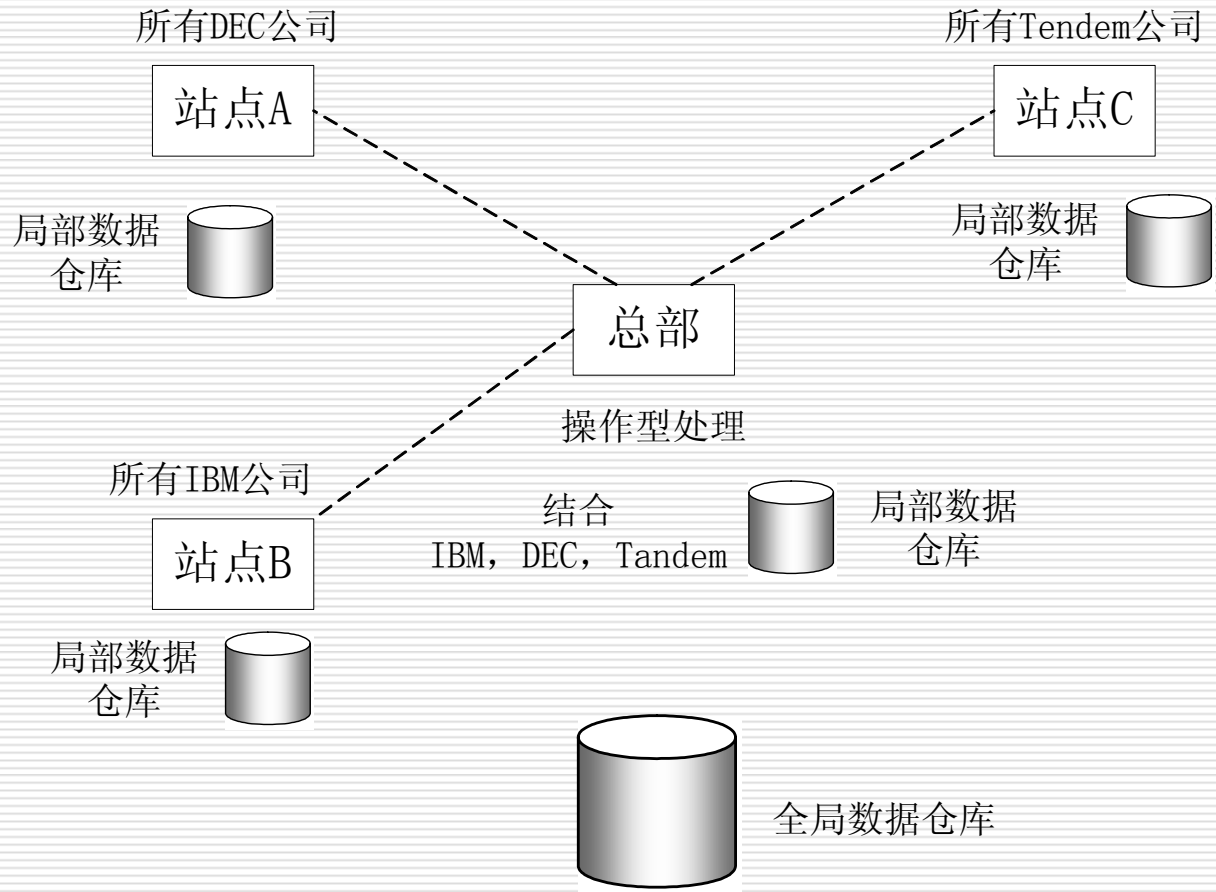
局部数据仓库

- 局部数据仓库是数据仓库的一种形式，只包含对局部层有意义的数据
- 局部数据仓库根据不同地区的分支机构或不同的技术联营组织创建的
- 局部数据仓库除了存储的数据是局部的外具有其他任何数据仓库的功能
- 局部数据仓库包含的是在局部站点上的历史的和集成的数据
- 局部数据仓库间的数据或数据结构不必要协调一致

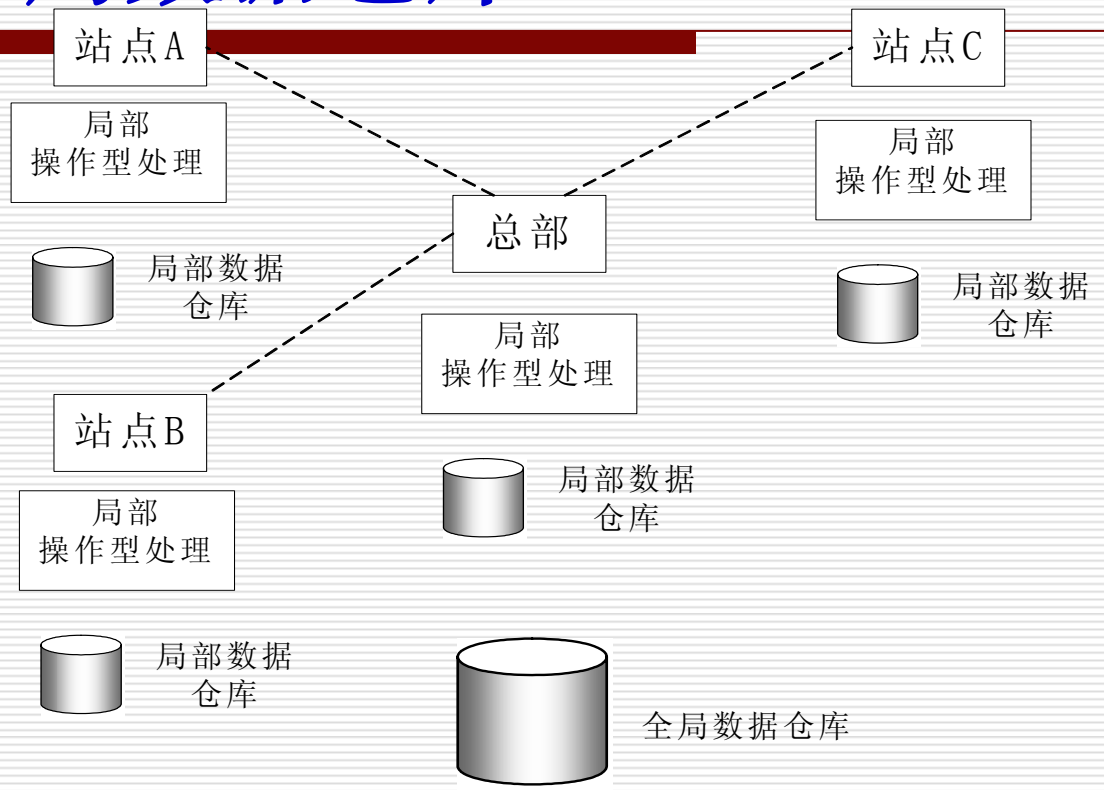
根据不同地区创建的局部数据仓库



根据不同技术联营创建的局部数据仓库

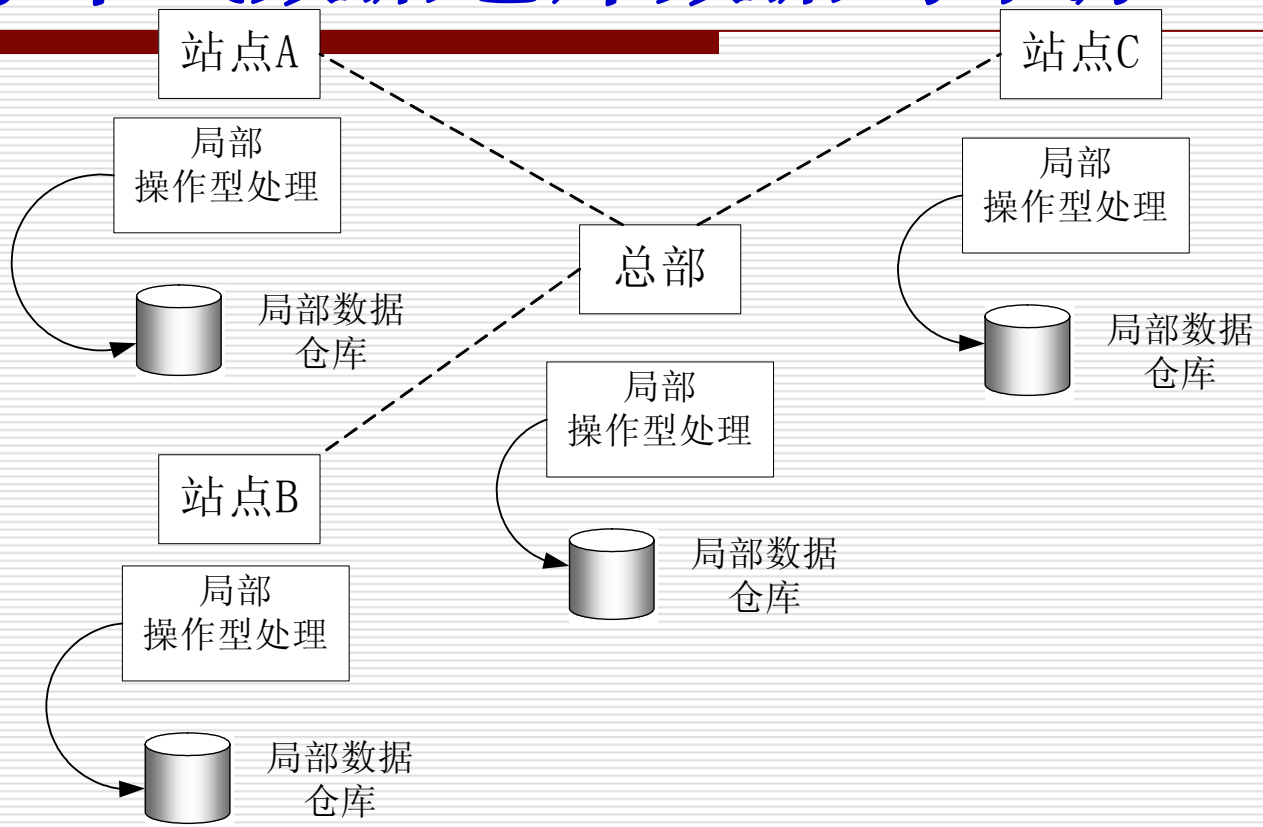


全局数据仓库



- 全局数据仓库涉及整个企业或组织
- 全局数据仓库的范围是企业级集成的业务数据，也包含历史数据

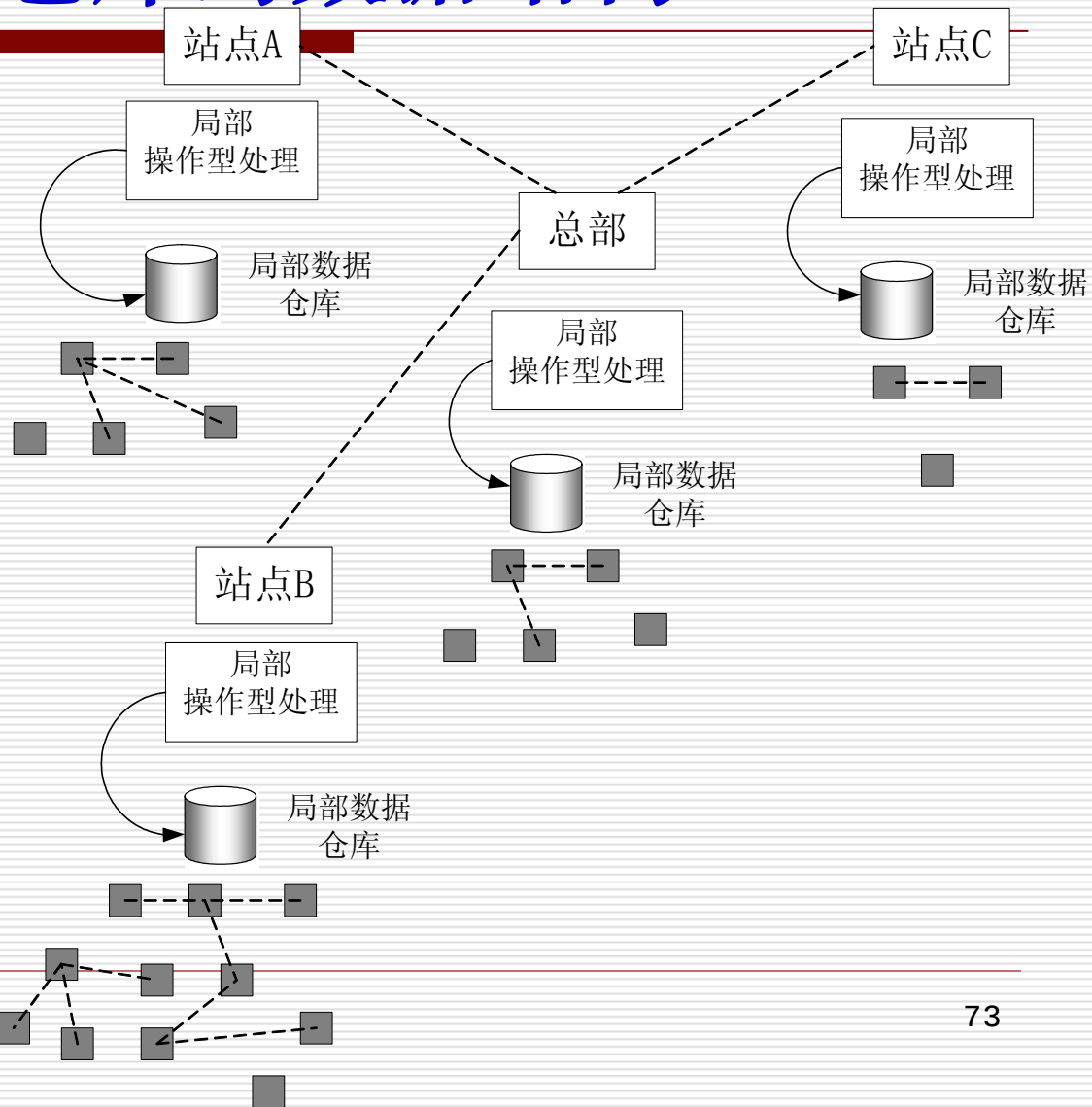
分布式数据仓库数据的来源



- 局部数据仓库数据来源于相应的操作型系统
- 全局数据仓库数据来源通常是局部数据仓库，有时，操作型数据可以直接进入全局数据仓库

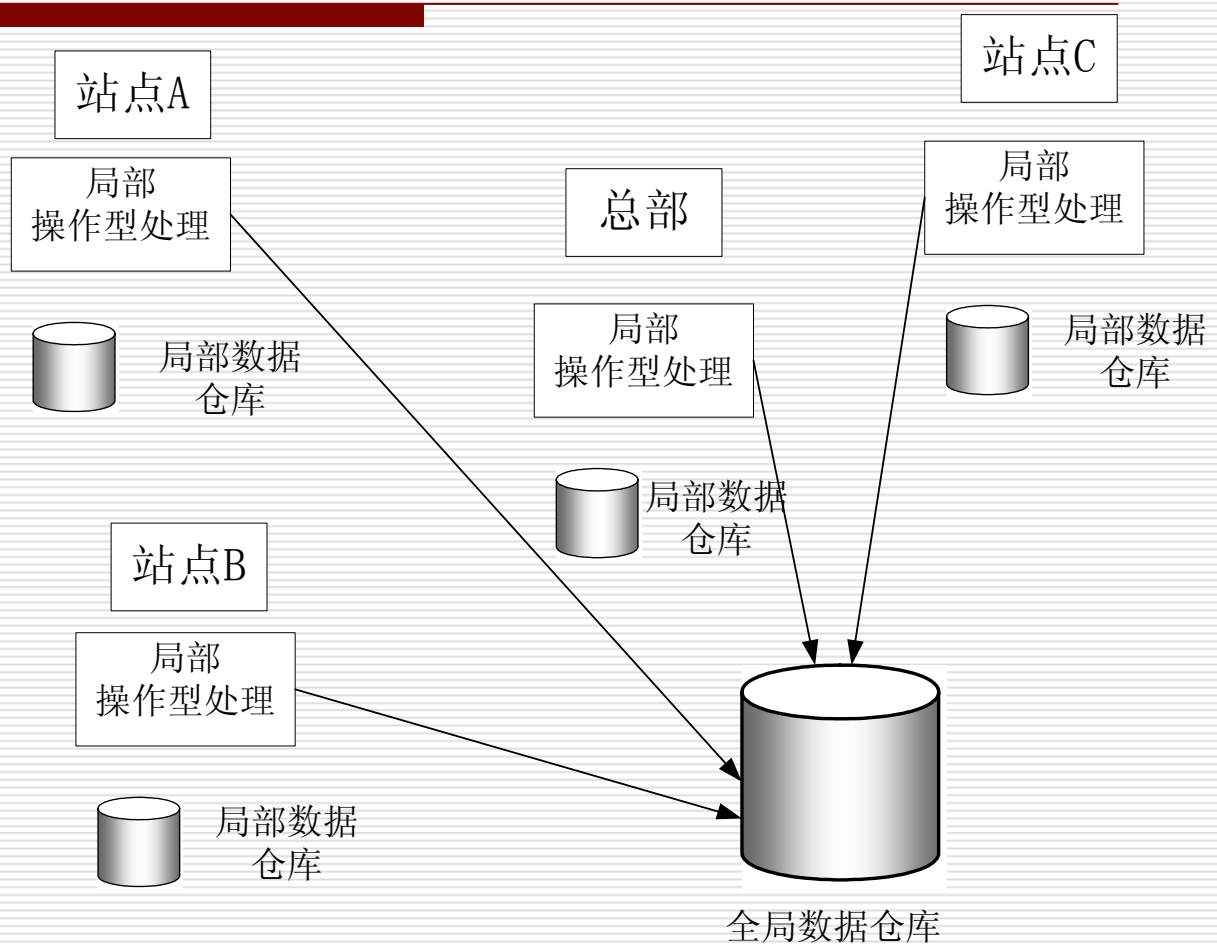
局部数据仓库的数据结构

- 局部数据仓库都有自己独特的数据和结构
- 局部数据仓库之间无论什么数据、处理过程或定义都不用协调



全局数据仓库数据来自于局部操作型系统的情况

- 企业内不同站点的数据有可能存在自然重叠，最好将这些数据存放在全局数据仓库中
- 全局数据仓库包含的是企业级公共和集成的数据，公有数据可能包含财务信息、客户信息、零售商信息等
- 数据可能同时存在两种数据仓库中，而且当数据被导入到全局数据仓库时需要数据转换，例如：币制转换、度量制转换

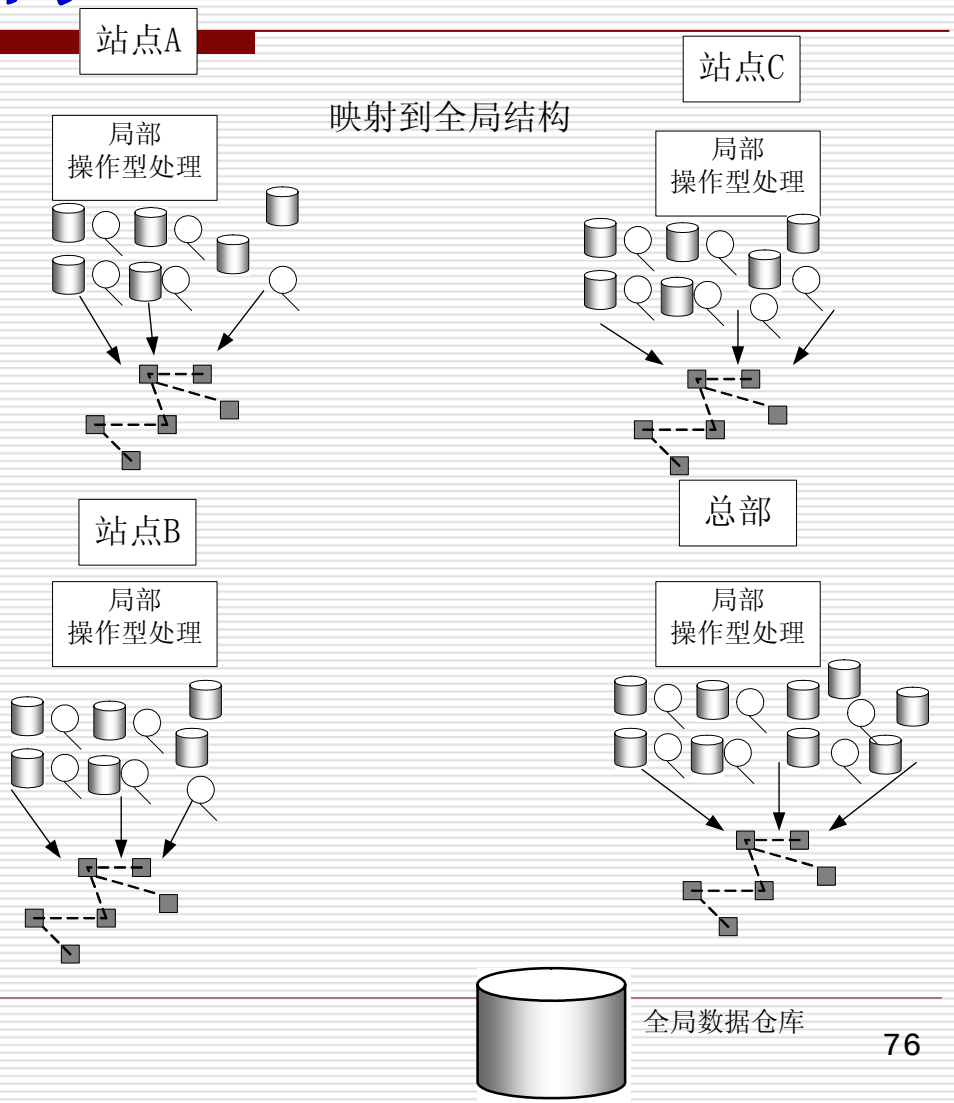


分布式数据仓库环境中的数据映射

- 如何将分支机构操作型系统中的数据映射到全局数据仓库中是分布式数据仓库环境成功的关键
  - 决定哪些数据进入全局数据仓库、数据的结构、必须做的转换
- 映射在设计全局数据时是最重要的部分，对于每一个局部数据仓库来说映射都不同，局部商业行为的差异决定了映射到全局数据仓库的方式
- 在创建全局数据仓库过程中，从局部数据到全局数据的映射很可能是建造全局数据仓库最困难的部分

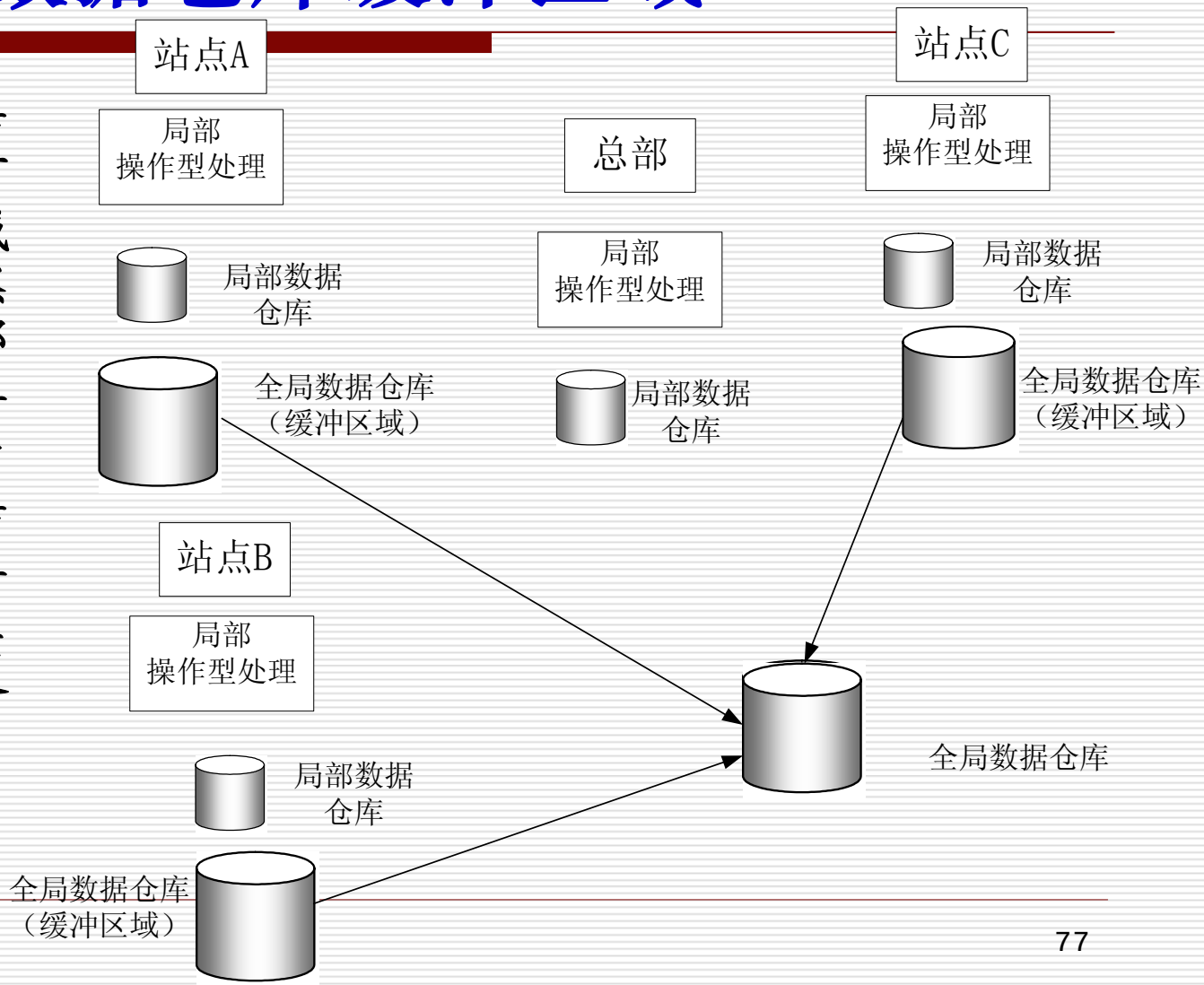
公共数据结构

- 全局数据仓库有一个公共数据结构，公共数据结构包含和定义企业内所有的公共数据，每个局部站点以不同的方式映射到公共结构
- 全局数据仓库根据公共数据结构集中定义和设计，而已存在的局部操作型系统的数据映射是由局部设计者和开发者选择



全局数据仓库缓冲区域

- 局部层在将数据传送到中心位置之前先在每个局部区域缓冲这些数据，然后将其传送到总部层的全局数据仓库
- 当局部数据仓库中保存的缓冲数据传送到全局数据仓库后应该考虑缓冲区数据的删除处理策略与方法



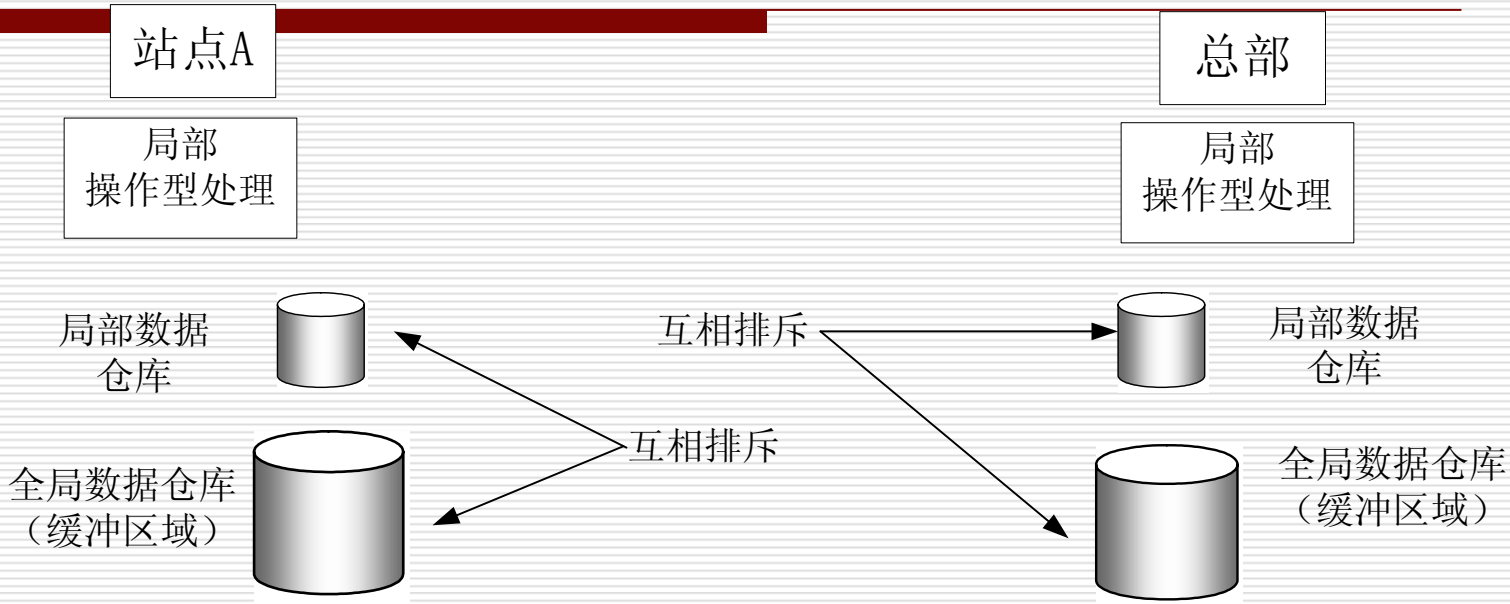
全局数据仓库的最初主题—财务是个好的起点

- 它是相对稳定的
- 具有高的可视性
- 仅是企业业务的一部分（当然，除了金融机构）
- 它是企业的神经中枢
- 仅需处理少量数据

数据冗余问题

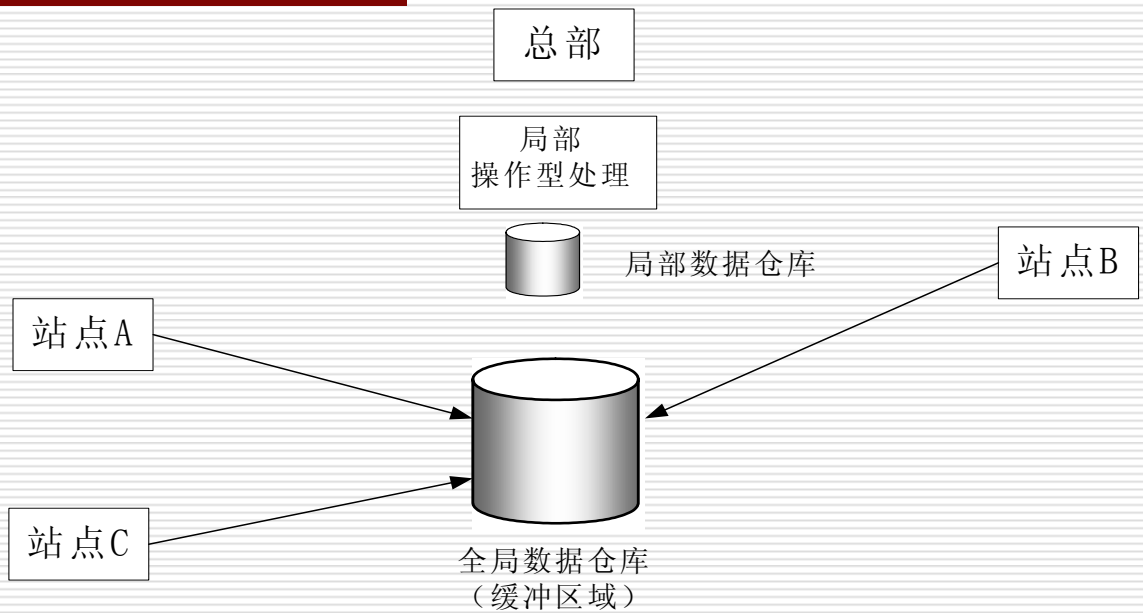
- 如果局部数据仓库和全局数据仓库间存在大量的数据冗余，说明没有正确定义不同级别数据仓库所辖的范围
- 当局部数据仓库和全局数据仓库间出现大量的数据冗余时，将最终导致出现“蜘蛛网”
- 出现大量数据冗余的系统将会带来很多问题——不一致的结果、不能很容易地创建新系统、操作的代价问题等

数据冗余问题的解决办法



- 对局部数据和全局数据实行互斥（即在两级数据仓库中不存放同一数据），从而可以避免分支机构层和全局层间的数据冗余
- 数据从局部数据仓库导入到全局数据仓库时要经过换算、转换、重新分类或汇总。这种情况下，严格地说，就不存在数据冗余

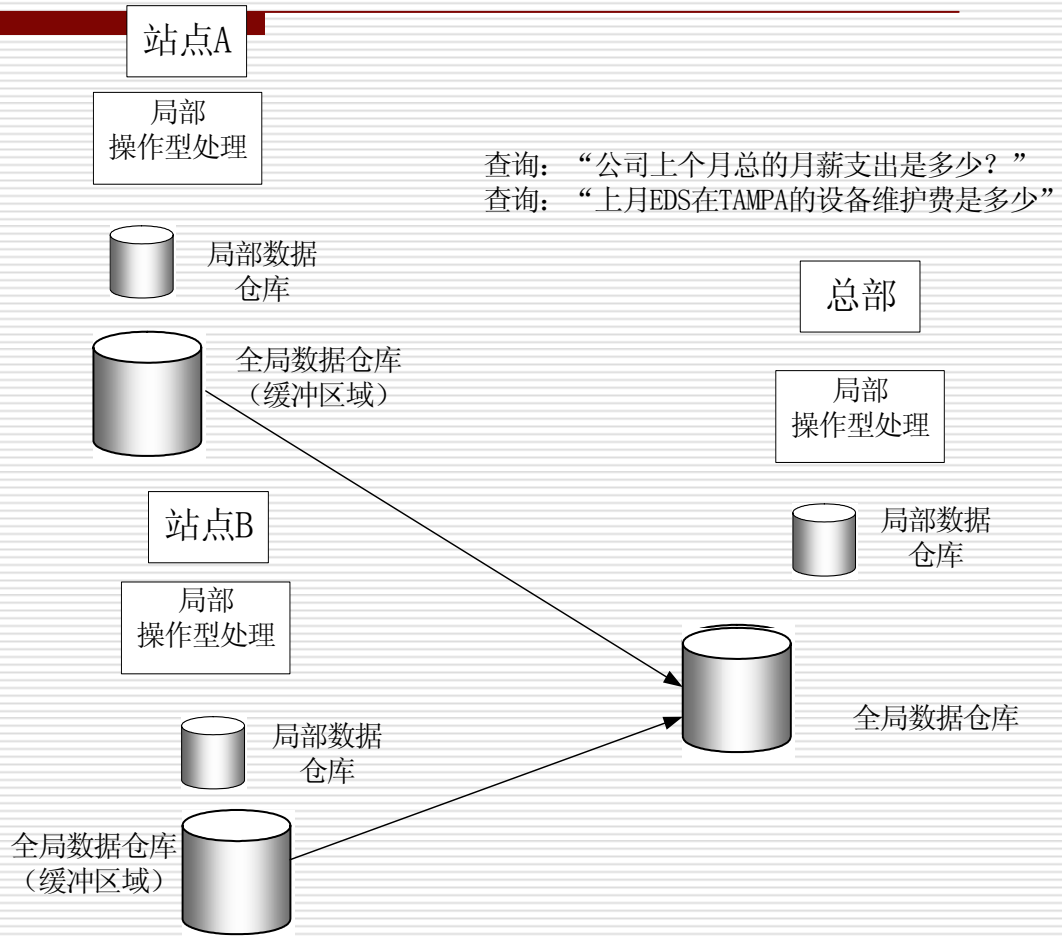
局部和全局数据存取（1）



- 分布式数据仓库环境需要解决的一个重要问题是局部站点是否应访问全局数据仓库
- 原则上，局部数据应局部使用，全局数据应全局使用
- 如果不是为提高局部业务运作能力，一个局部层的分析人员不应该为其他目的查看全局数据

局部和全局数据存取（2）

- 信息请求的路径选择问题：分布式数据仓库中，需要考虑如何确保信息请求来自正确的地方，如：
  - 通过查询局部站点来确定整个公司的薪水是多少是不正确的
  - 在全局数据仓库中查询上月在某一特定站点上某一承包人的费用支付情况也是不正确的



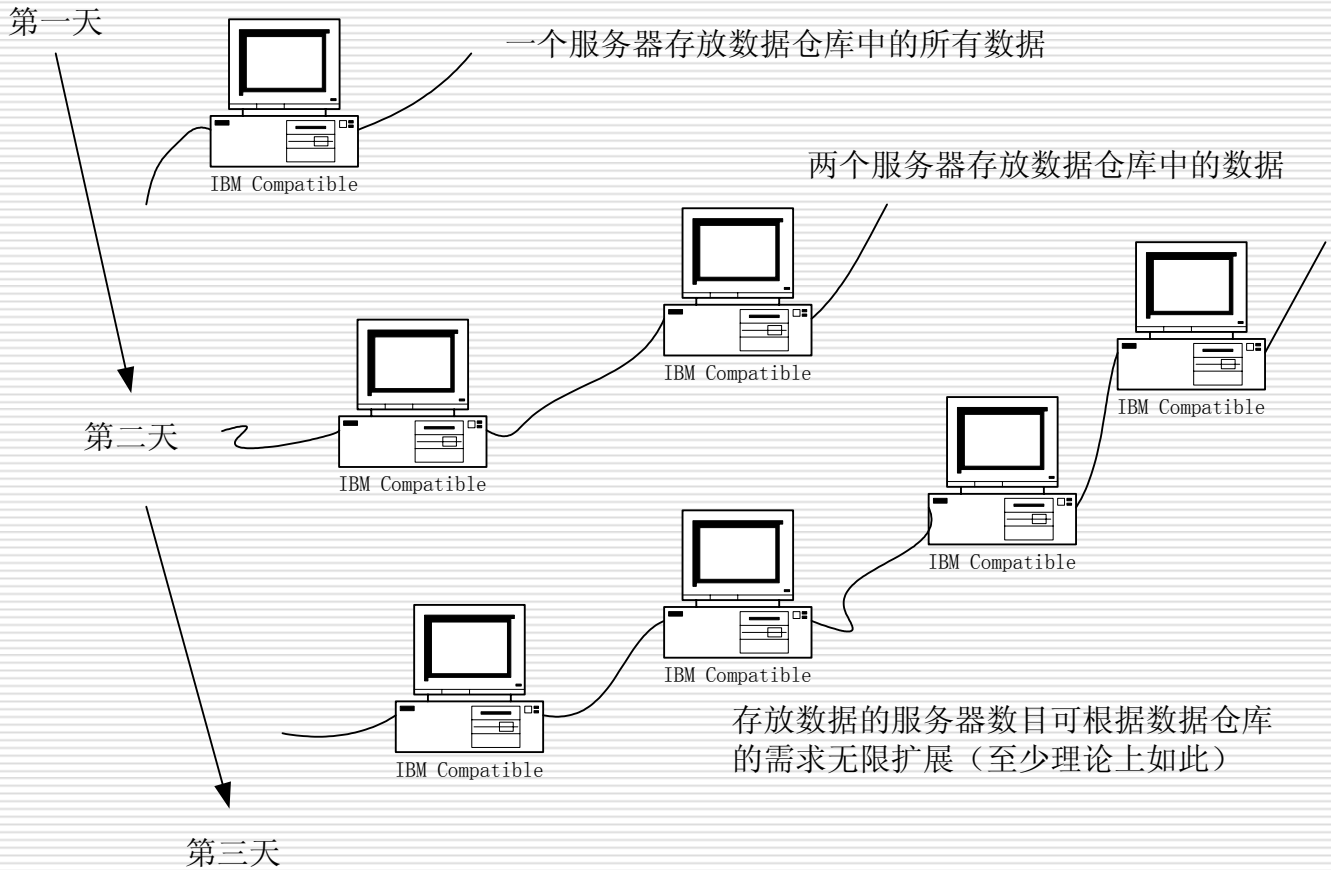
数据传输问题

- 从局部环境到全局环境传输数据的频率如何？一天？一周？或是一个月？全局数据仓库要求数据传输的速度？要传输的数据量是多少？
- 从局部环境到全局数据仓库的传输是否合法？
- 从局部环境到全局环境传输数据要使用什么样的网络？安全可靠性如何？备份的策略是什么？如何确定所有的数据已经传输完毕？
- 传输过程中，应使用什么样的安全保护措施来判断数据是否被非法入侵？
- 为了从局部环境到全局环境传送数据，处理过程的哪一部分是可见的？当数据仓库的负载很重时，是否还传输数据？
- 局部数据应采用什么技术？而全局数据应采用什么技术？将局部技术转换为全局技术必须采取什么措施？在转换过程中会出现数据的丢失吗？

分布式数据仓库环境的其他问题

- 尽管本章假设每个局部站点进行的操作都是自己独有的，但是，局部站点的操作型系统间可能存在某些共性
- 分布式数据仓库环境的整个问题是比较复杂的，范围、协调、元数据、响应能力、数据传输以及局部数据映射等问题使得整个环境复杂化了
- 创建和装载全局数据仓库最大的困难是局部数据和全局数据的映射，这些映射关系不能集中式生成，必须局部生成，必须局部管理和控制映射处理
- 因此，局部层的人员必然参与全局数据的创建
- 局部层的数据应当采用尽可能灵活的形式，即数据应该是关系型的，不应是多维模型进行组织的

增加服务器来保存数据仓库增加的数据



- 尽管数据仓库将包含很多很多数据，但并不是无限量的

增加服务器带来的问题

- 当数据仓库中的处理器（即服务器）扩展到一定数量时，网络上就会出现过量的传输负载，例如:

  假设一台服务器存有1998年的数据，另一台存有1999年的数据，再一台存有2000年的，第四台存有2001年的，当一个查询需要访问从1998至2001年的数据的话，这个查询结果集必须访问存有不同年限数据的服务器

- 问题不仅仅出现在一个查询要存取被多个服务器管理的数据，而且出现在需要从一台服务器上传输大量的数据
- 随着数据仓库越来越庞大，服务器越来越多，问题也就越来越严重

独立开发的分布式数据仓库

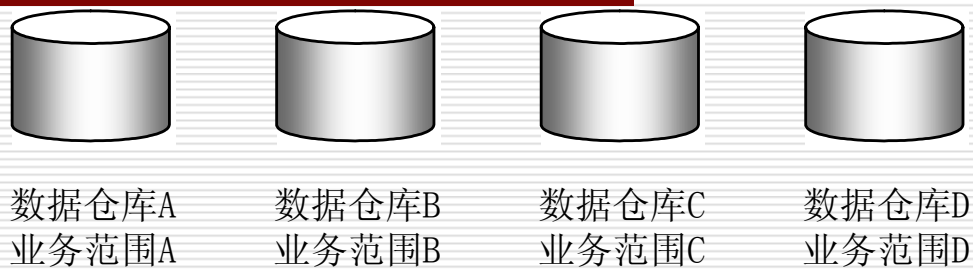
- 多个独立的数据仓库是同时开发的，且数据仓库之间没有进行协调和约束
- 许多企业采用数据仓库技术时，首先是为财务或市场管理部门建立数据仓库，一旦成功，其它部门就希望在此基础上建立相应的数据仓库
- 因此，企业数据仓库的体系结构设计员需要管理和协调企业内的多个数据仓库项目

多个数据仓库开发项目分类

多个开发小组建造数据仓库时可分为四种典型情况：

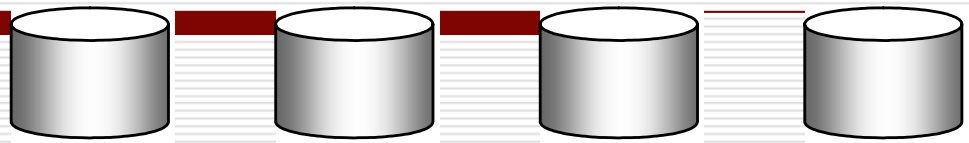
- 完全非集成的业务范围各自拥有自己的数据仓库（各自拥有自己的数据仓库）
- 同一数据仓库具有一些分布式部分（各自拥有不同的分布式部分）
- 同一数据仓库内不同级的数据（各自拥有不同级的数据）
- 数据仓库的细节级的不同的非分布式部分（各自拥有不同的非分布式部分）

完全非集成的业务范围各自拥有自己的数据仓库



- 公司的业务是完全分离的、非集成的，对应的数据仓库是由不同的开发小组独立创建的
- 不同的业务独立向公司汇报情况，但是除了共享公司名称，在公司内没有业务集成或数据共享
- 这种公司结构在现实中是存在的，但不常见
- 在这种情况下，一项数据仓库开发项目与另一个数据仓库开发项目间发生冲突的危险几乎没有，因此，项目间很少或不需要管理和协调

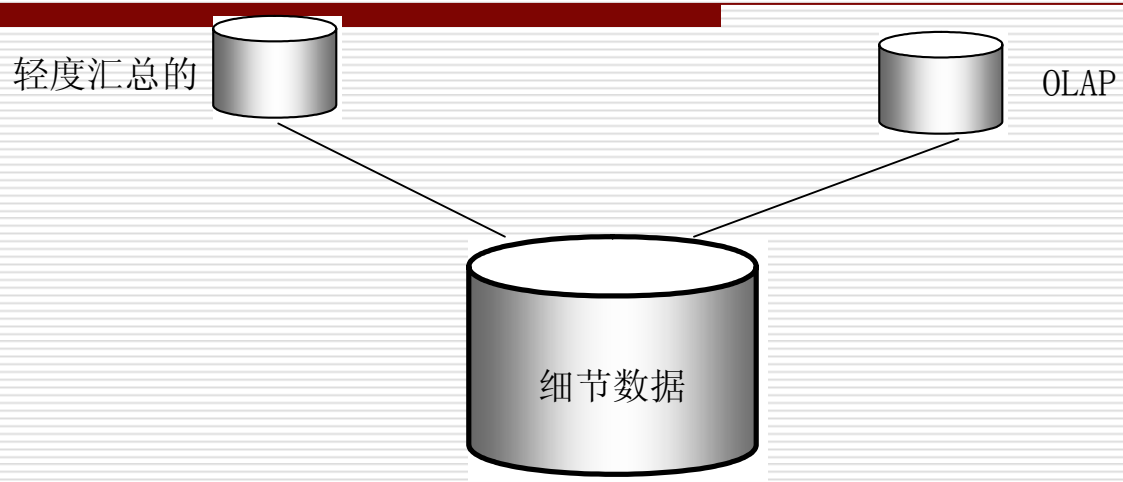
同一数据仓库具有一些分布式部分



东北部数据仓库　中西部数据仓库　　西部数据仓库　　东南部数据仓库

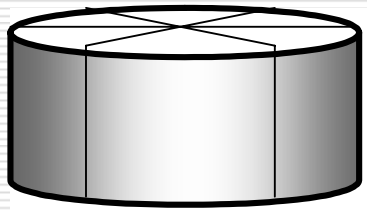
- 各个开发小组负责创建同一个数据仓库的不同部分，从而导致多个数据仓库开发项目同时出现
- 同一级数据是由不同开发小组创建的，它们分散在不同的地理位置
- 不同数据仓库中数据的细节程度是一样的，不采取特殊措施的话，在使用时会产生大量的冲突
- 这种情况十分常见，为了从总体上获得满意的集成效果，要求开发小组间进行密切协作
- 若开发项目不协调，则大量数据的冗余存储和处理将导致数据仓库的效率很低和较大的浪费

同一数据仓库不同级的数据



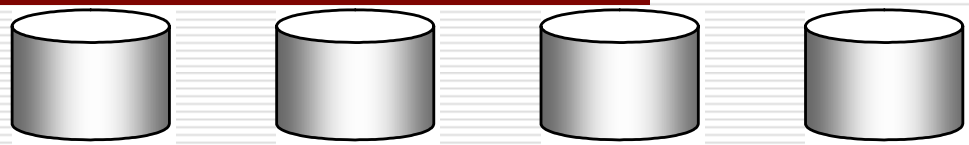
- 不同小组负责建立数据仓库环境中的不同级的数据（例如：汇总数据和细节数据）
- 各层中数据的不同，作用也各不相同，小组之间的协调就有可能简单地运作，小组间的接口也是很简单的

数据仓库细节级的不同的非分布式部分



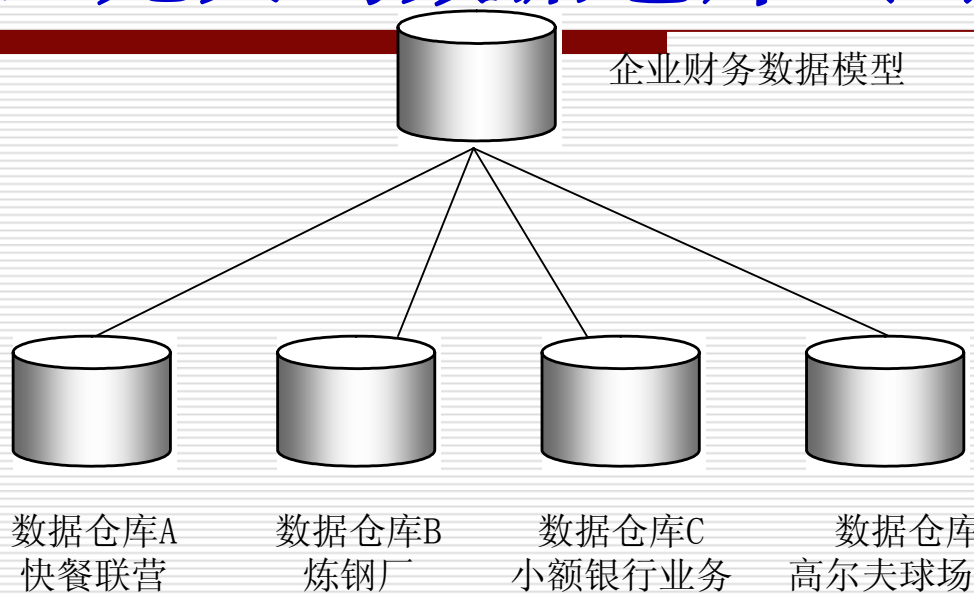
- 多个小组试图以非分布式的方式建立数据仓库中数据当前细节级的不同部分
- 这种情况很少出现，但一旦发生，就必须特别注意，数据体系结构设计者必须明确问题所在以及如何协调以便成功地建立数据仓库

完全无关的数据仓库（1）



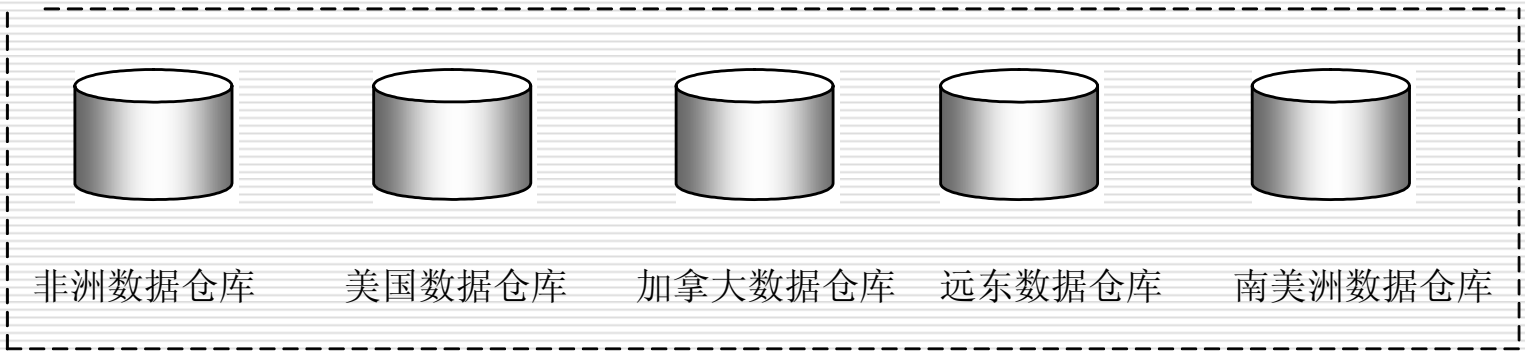
- 完全独立的各种业务部门
- 业务间没有任何集成，尽管一种业务的客户可能是另一种业务的客户，但两者之间没有联系
- 数据仓库的建设不需要协调，从建模到基本技术的选择（即平台、DBMS、存取工具、开发工具等）每种业务的运作均可完全独立地进行

完全无关的数据仓库（2）



- 即使完全自主的业务，在某一层上也是必须集成的，如企业可能需要建立一个企业数据仓库来反映企业财务
- 企业财务数据仓库包含一些简单（和抽象）的实体，业务数据即使有也非常少，即财务数据仓库中没有公用企业描述信息
- 企业财务数据仓库中的数据可能来自于局部数据仓库或企业级的操作型系统
- 元数据在局部层至关重要，在企业财务数据仓库中，也需要元数据，但由于不存在真正的业务集成，因此没必要把任何元数据联系在一起

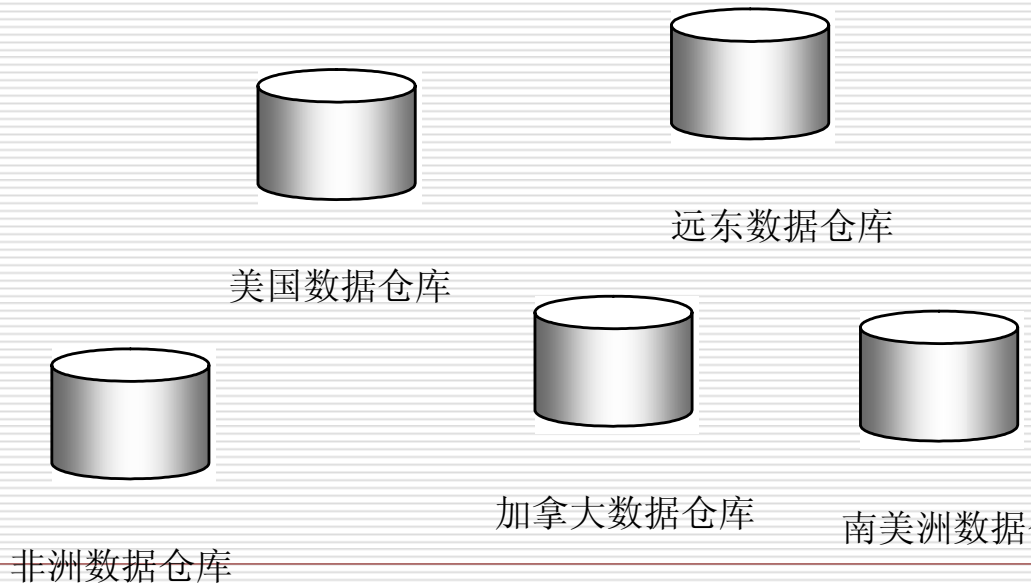
分布式数据仓库（1）



- 逻辑上属于同一个数据仓库
- 每个分支机构具有自己特有的数据，机构间不存在数据重叠，特别是细节事务数据的重叠
- 每个分支机构各创建一个数据仓库作为创建体系结构化环境的第一步
- 不同的分支机构间存在某种程度的业务集成
- 这种企业组织模式很常见

分布式数据仓库（2）

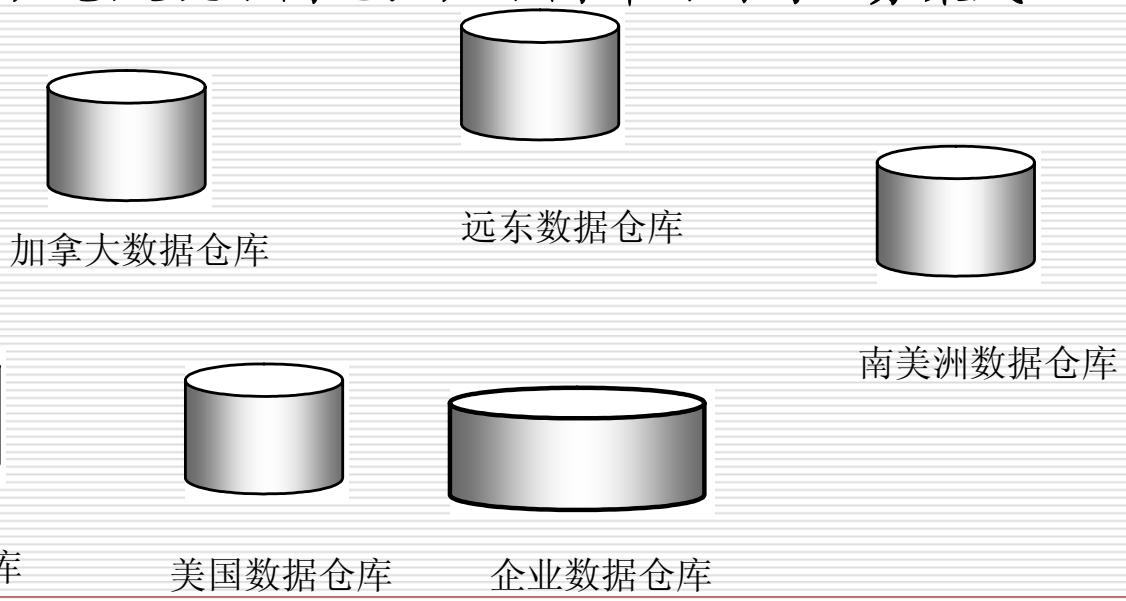
- 首先为每个不同地域的分支机构各创建一个局部数据仓库
- 每个分部根据自己的需要创建特有的自主数据仓库，不同区域间不存在冗余的细节数据
- 优点：建设速度很快
- 缺点：不能识别或合理处理不同部门间数据结构的共同性



在每个子公司建立数据仓库

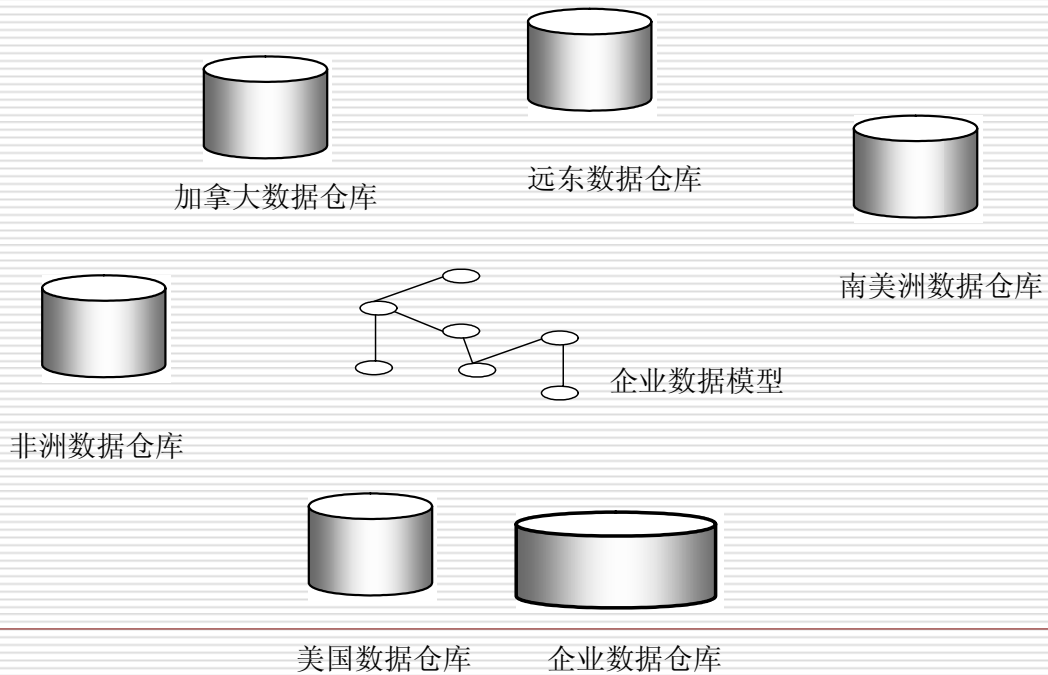
分布式数据仓库（3）

- 在分布的地理位置间协调开发
  - 尽量协调不同的局部组织间的局部数据仓库的开发项目
  - 必须提出一个新的数据模型作为各个局部数据仓库的设计基础
  - 当数据仓库的价值在局部层表现出来后，企业就会决定建造一个企业数据仓库，它反映不同地区，不同部门间的业务集成



分布式数据仓库（4）

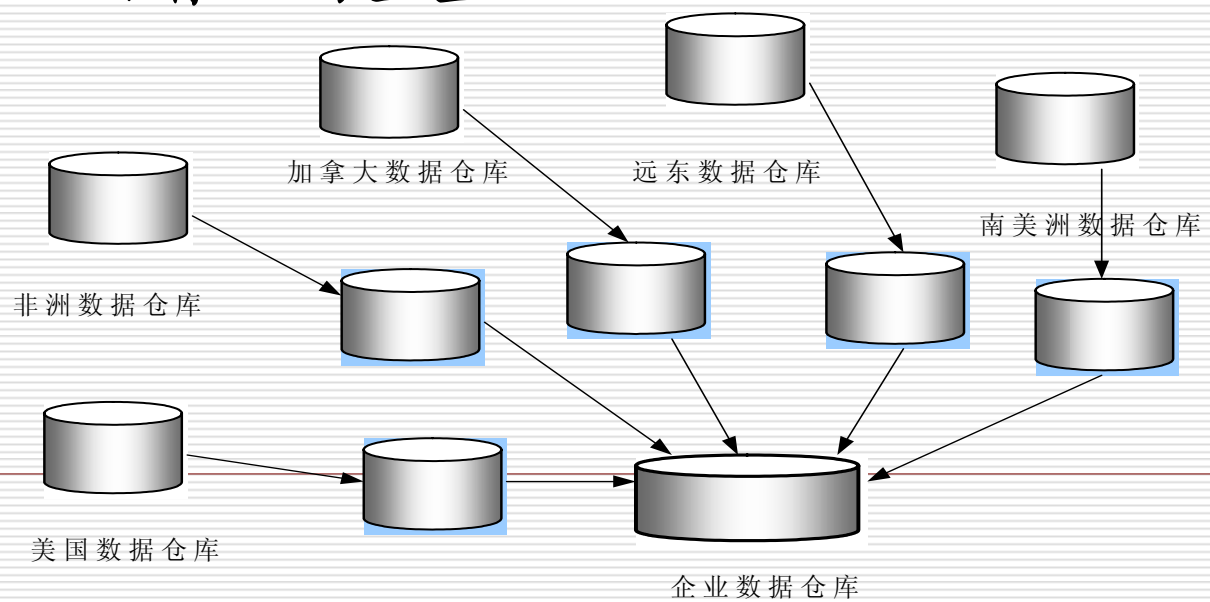
- 建立企业数据仓库的第一步是为相关的业务部门建立企业数据模型
- 一般来说，建立企业数据仓库采用迭代开发方法
- 企业数据模型建立后，将形成企业数据仓库



建立企业数据模型

分布式数据仓库（5）

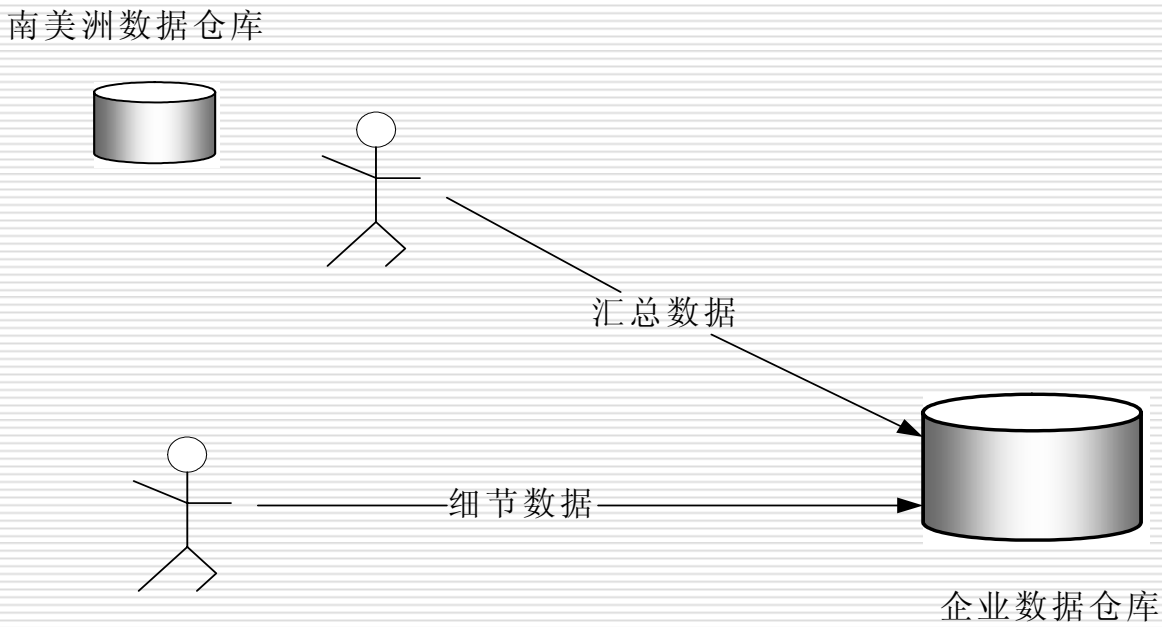
- 企业数据模型反映企业级的业务集成，因此可能与局部数据模型中的某些部分重叠
- 但不管什么情况，都由局部组织来决定如何使企业的数据需求和局部的数据提供能力相适应
- 当局部层间数据结构的重叠部分设计得较好时，数据内容上就不会有大的重叠



分布式数据仓库（6）

- 企业数据仓库的数据源
  - 可能来自于局部数据仓库，也可能来自于局部操作型系统，完全应在局部层决定
  - 记录系统的定义大多需要几次循环往复
  - 从技术角度考虑如何将局部层的记录系统数据创建和传送到企业数据仓库
    - 正式“缓冲”的数据保留在局部层
    - 数据被传送到企业环境，且在局部层不可存取

分布式数据仓库（7）



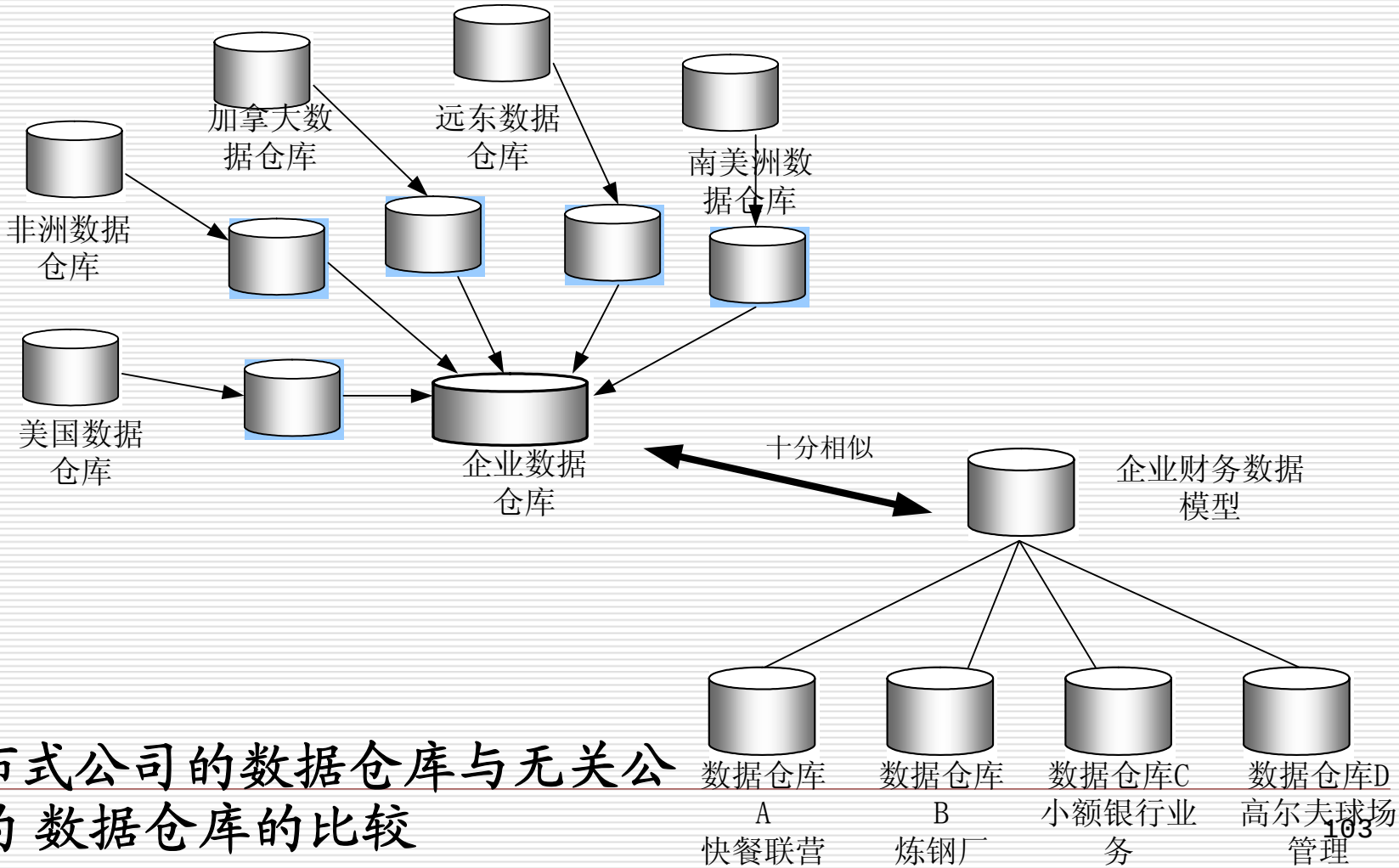
从不同角度观察企业数据仓库

- 企业数据仓库中对企业层的DSS分析员来说是细节数据，同时对局部层的DSS分析员来说却是汇总数据
- 这是由观察者的角度决定的

分布式数据仓库（8）

- 分布式数据仓库中的元数据
  - 在整个分布式的企业数据仓库中元数据起着非常重要的作用
  - 通过元数据可以协调不同地域的数据仓库中的 数据结构
  - 元数据是实现一致性和相容性的工具

两种数据仓库的比较（1）



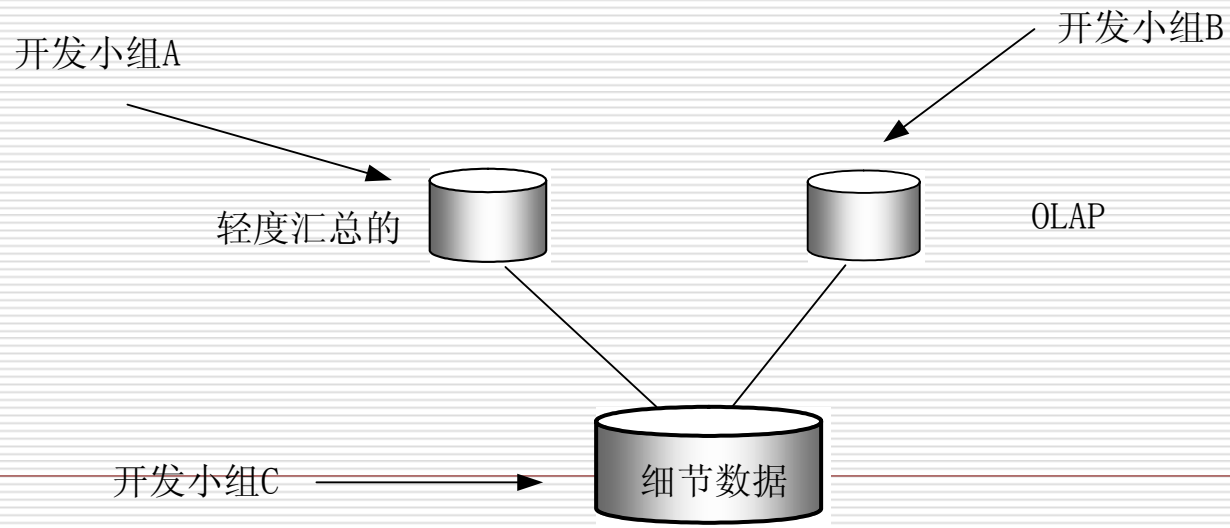
分布式公司的数据仓库与无关公司的 数据仓库的比较

两种数据仓库的比较（2）

- 分布式公司的数据仓库在许多方面非常类似于无关公司的 数据仓库，如设计和运作方面
- 企业分布式数据仓库是对业务本身的扩展，反映客户、销售商、产品等集成信息，因此，企业分布式数据仓库表示了业务本身的体系结构
- 业务无关的公司的企业数据仓库是专门为财务服务的
- 两种数据仓库的不同之处是他们表达数据的深度不同

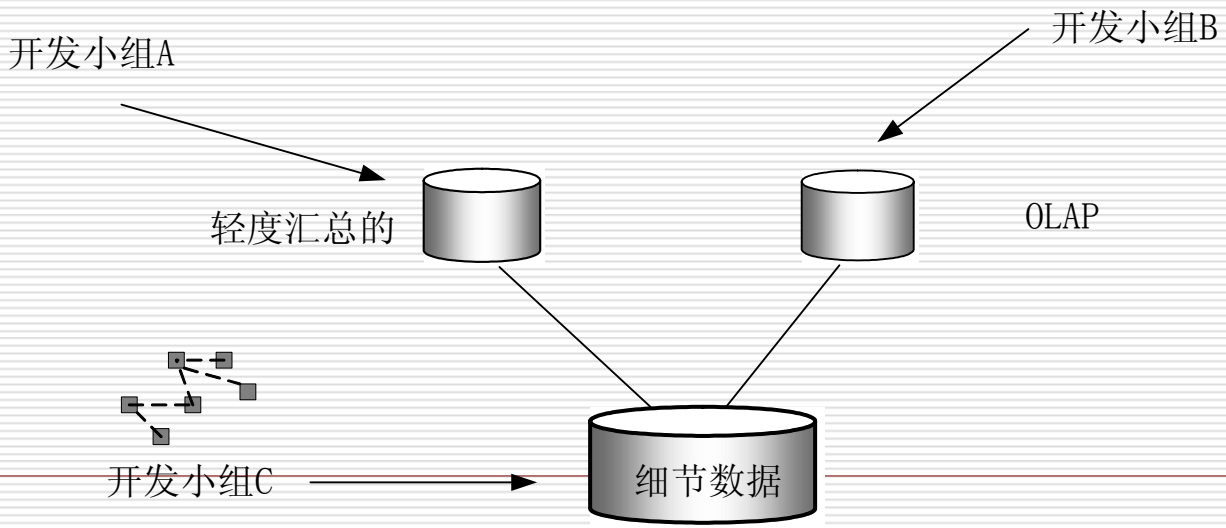
各自拥有不同级的数据（1）

- 在多种层次上构建数据仓库
  - 不同的开发小组负责构建数据仓库的不同层次
  - 这种模式与分布式数据仓库开发模式区别很大
  - 这种数据仓库的多层模式是很常见的，并且这种模式最容易管理，而且风险最小
  - 数据体系结构设计者主要关心的 问题是如何协调不同开发小组的工作，包括内容的规范说明和结构的描述以及开发时间的确定



各自拥有不同级的数据（2）

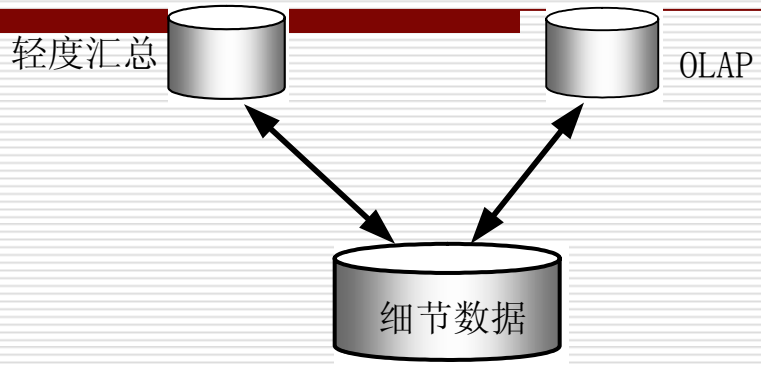
- 数据仓库的数据模型
  - 数据仓库的数据模型直接反映了负责当前细节级分析和设计的开发小组的设计和开发工作
  - 数据仓库模型间接反映了所有开发小组的需求
  - 开发最低细节级的的开发组使用数据仓库的数据模型
  - 在大多数场合下，较高汇总级的开发小组拥有反映他们特定需要的数据模型



各自拥有不同级的数据（3）

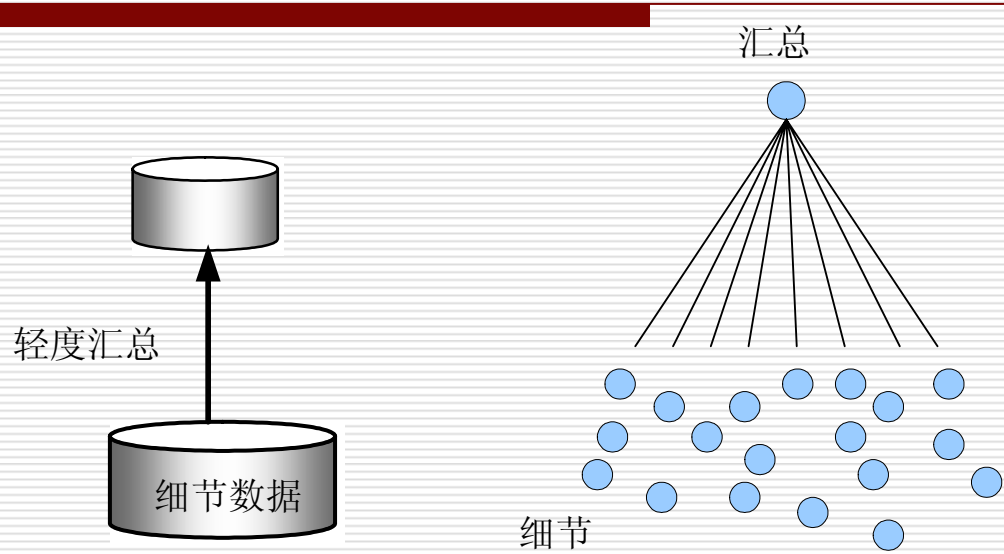
- 过程采用的技术平台问题
  - 一般来说，不同的开发小组选用的技术平台不同，这是因为：
    - 代价问题：数据的细节级，由于处理的数据量大，所以要求一个企业级的平台，不同的汇总级，需要处理的数据量相对较少，所以可以采用不同于细节级的技术平台
    - 各种汇总级可选用的平台提供多种多样的特殊软件，而这些软件许多是细节级单一平台上所不支持的
  - 不管数据的不同层次是采用单一平台或是多种平台，都必须认真存储和管理元数据，以保证从一个细节级到下一层细节级的连续性

各自拥有不同级的数据（4）



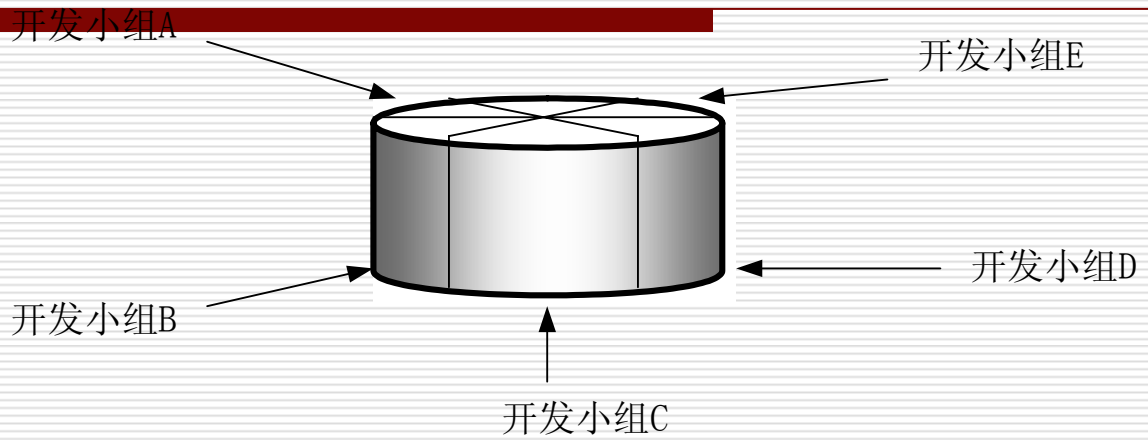
- 不同层次间的互连
  - 由于数据仓库的不同开发小组在开发不同级数据时通常采用不同平台，这就出现了互连性问题
  - 互连性要考虑的几个方面
    - 调用级存取的兼容性：在数据仓库的任何两级之间构成的细节数据和汇总数据时所采用的技术之间在调用语法上是否兼容？如果调用语法不兼容，那么接口将不会有用
    - 有效带宽：如果两级数据仓库中某一级有很大的传输处理负载，那么两个系统间的接口将会成为瓶颈

各自拥有不同级的数据（5）



- 汇总级数据的基础
  - 细节级数据是建立汇总级数据的基础
  - 管理低级细节数据的开发小组必须为建立汇总级数据的开发小组提供一个正确的数据基础
  - 协调各开发小组，使所有开发小组在需要数据时都能获得所需要的较低级上的数据

各自拥有不同的非分布式部分（1）

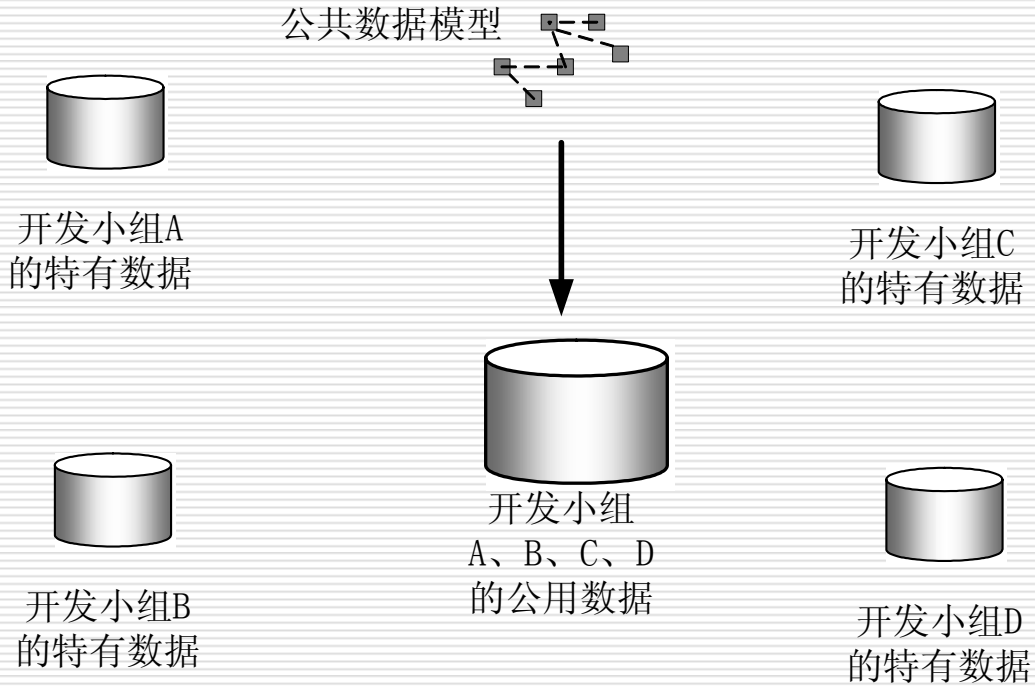


- 多个小组建立当前细节级
  - 如果开发当前细节级的开发小组使用相同的数据模型，开发的数据集是互斥的，且不同开发小组的技术平台间是兼容的，那就没有什么风险，但是，更常见的是，多个开发小组设计和装载的是一些或全部相同的数据
  - 当开发小组的工作出现重叠时，将会引发一系列问题：
    - 费用，尤其是存储和处理的费用
    - “蜘蛛网”问题，由于存在大量冗余的细节数据，这违背了建立数据仓库的初衷

各自拥有不同的非分布式部分（2）

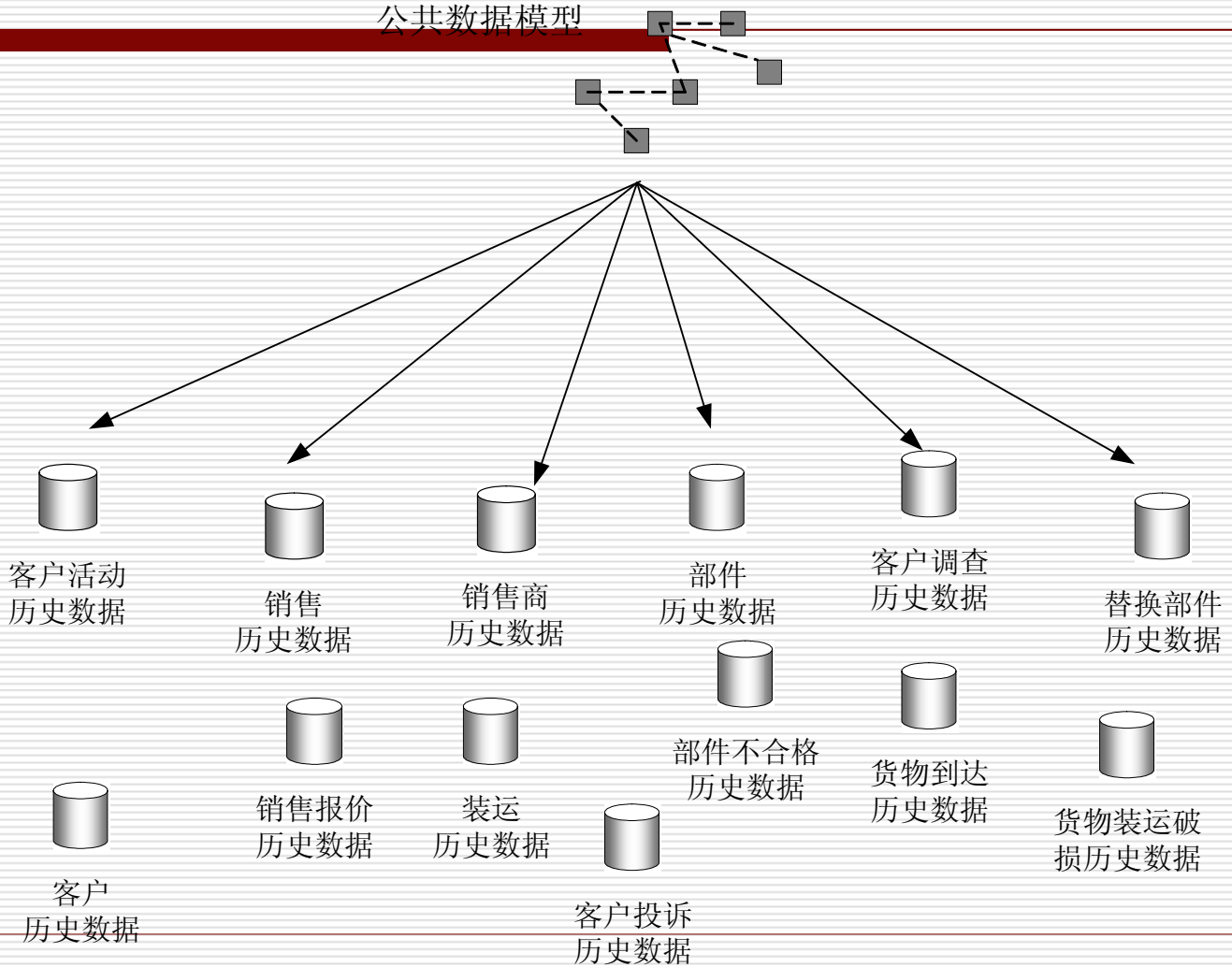
- 公共细节数据的细节数据模型
  - 如果多个开发小组并行地设计和装载当前的细节数据，一定要确保没有创建冗余数据，这就需要创建一个公用数据的数据模型
  - 公用细节数据模型反映了数据仓库中不同开发小组对细节数据的共同需求
  - 开发小组可以是正在进行开发的小组，也可以是将来可能介入的其他开发小组，都可以提出他们的要求
  - 如果开发小组知道将来的需求，但是又不能清楚描述他们，那么这些需求也不会作为公用细节数据模型中的考虑因素

各自拥有不同的非分布式部分（3）



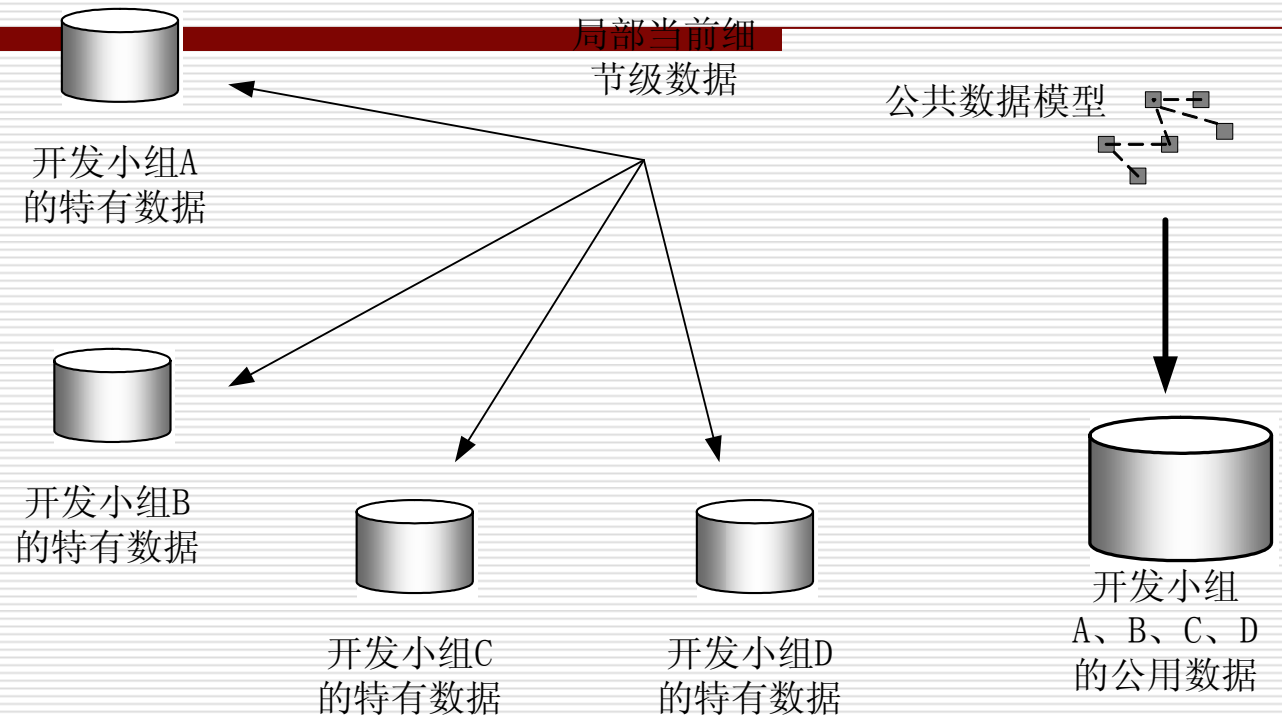
- 对所有开发小组，公共数据模型标识公用数据

各自拥有不同的非分布式部分（4）



数据仓库在物理上分布在多个物理表和数据库中

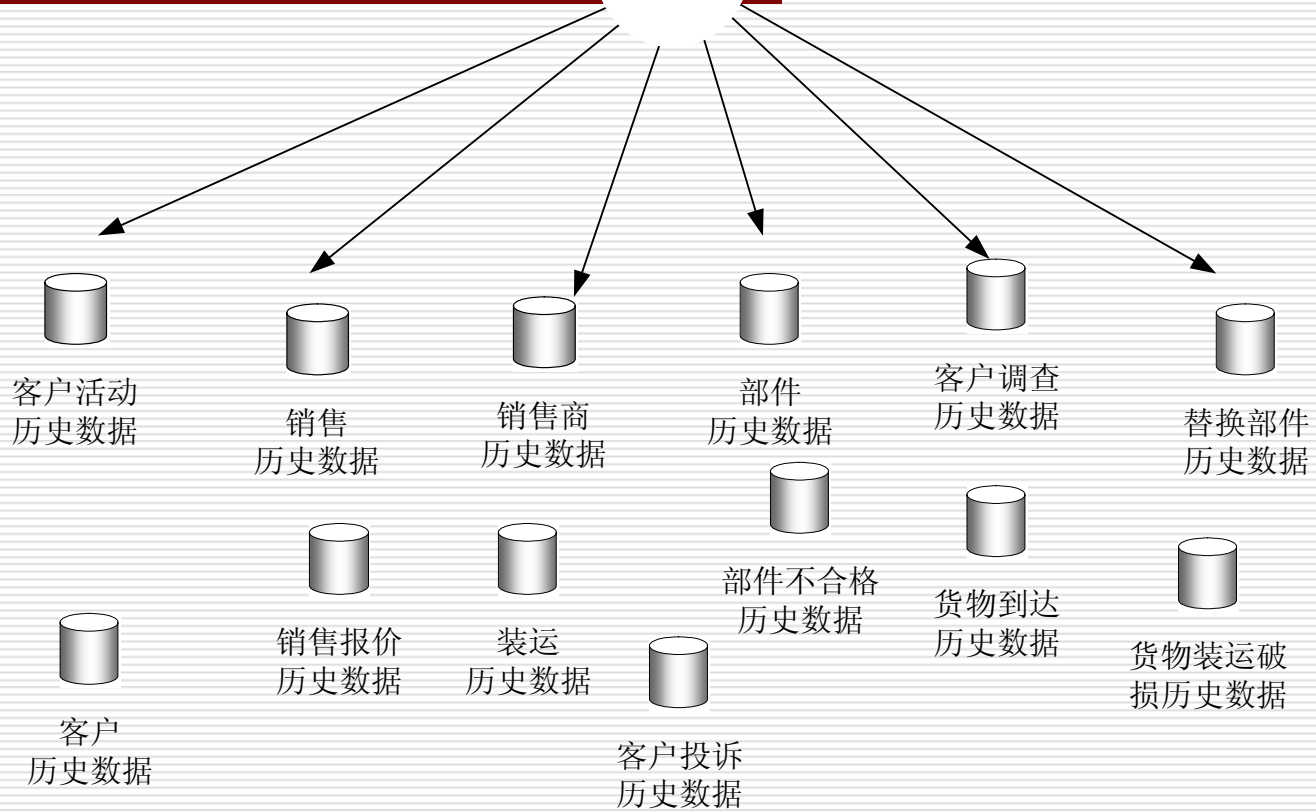
各自拥有不同的非分布式部分（5）



数据仓库中的当前细节级包含各开发小组的特有数据

- 不同开发小组的需求不同，这些特殊的需求导致了所谓“局部的”当前细节级
- 局部数据有自己的数据模型，通常比公用细节数据模型小得多也简单得多

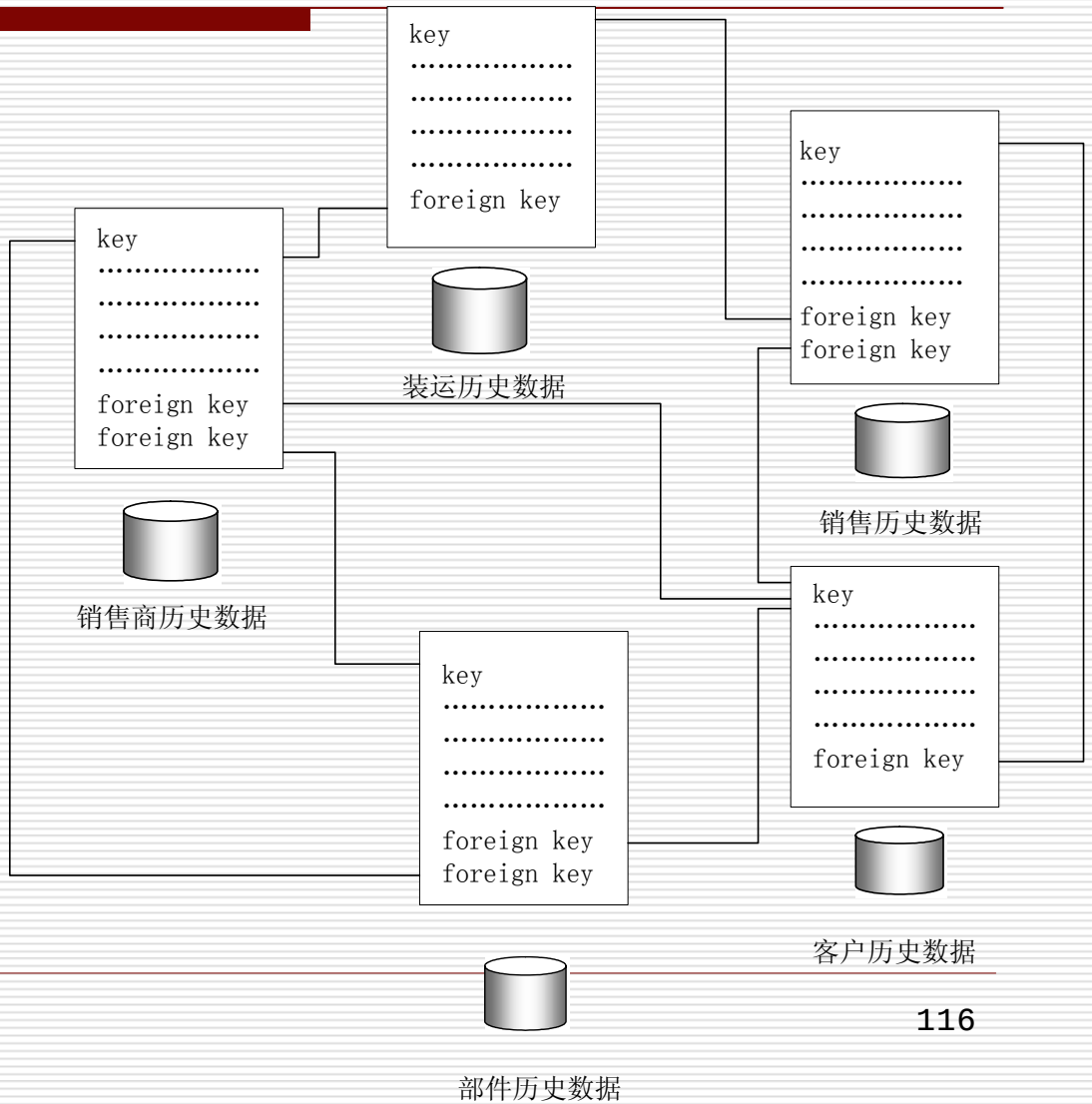
各自拥有不同的非分布式部分（6）



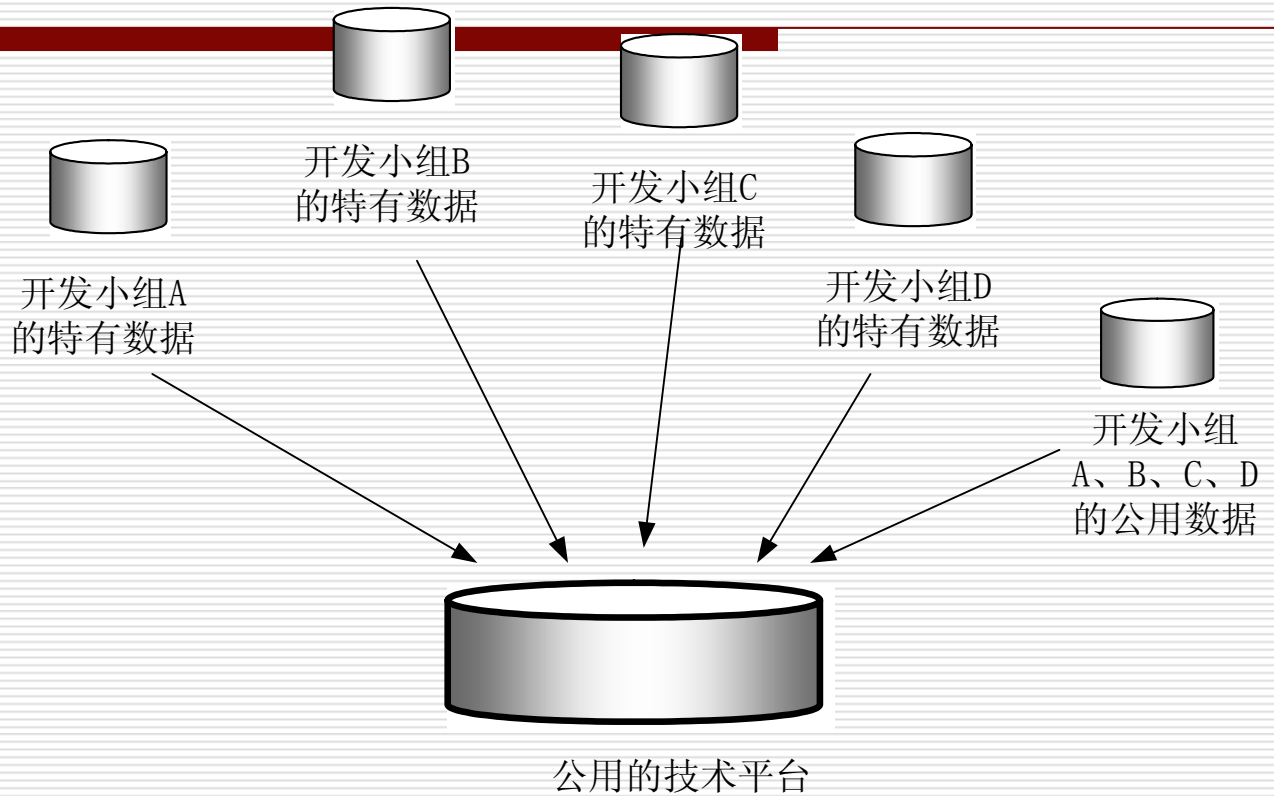
- 细节数据不存在冗余
  - 构成数据仓库细节级的多张表中非键码数据的非冗余性
  - 主键数据肯定是冗余的，因为外键用于将不同类型的数据相关联

各自拥有不同的非分布式部分（7）

- 数据仓库环境中的外键
  - 数据仓库表中的外键与受参照完整性所支配的典型的外键关系不同
  - 数据仓库中收集和存取的是快照数据，出现外键的关系是以人工关系组织的



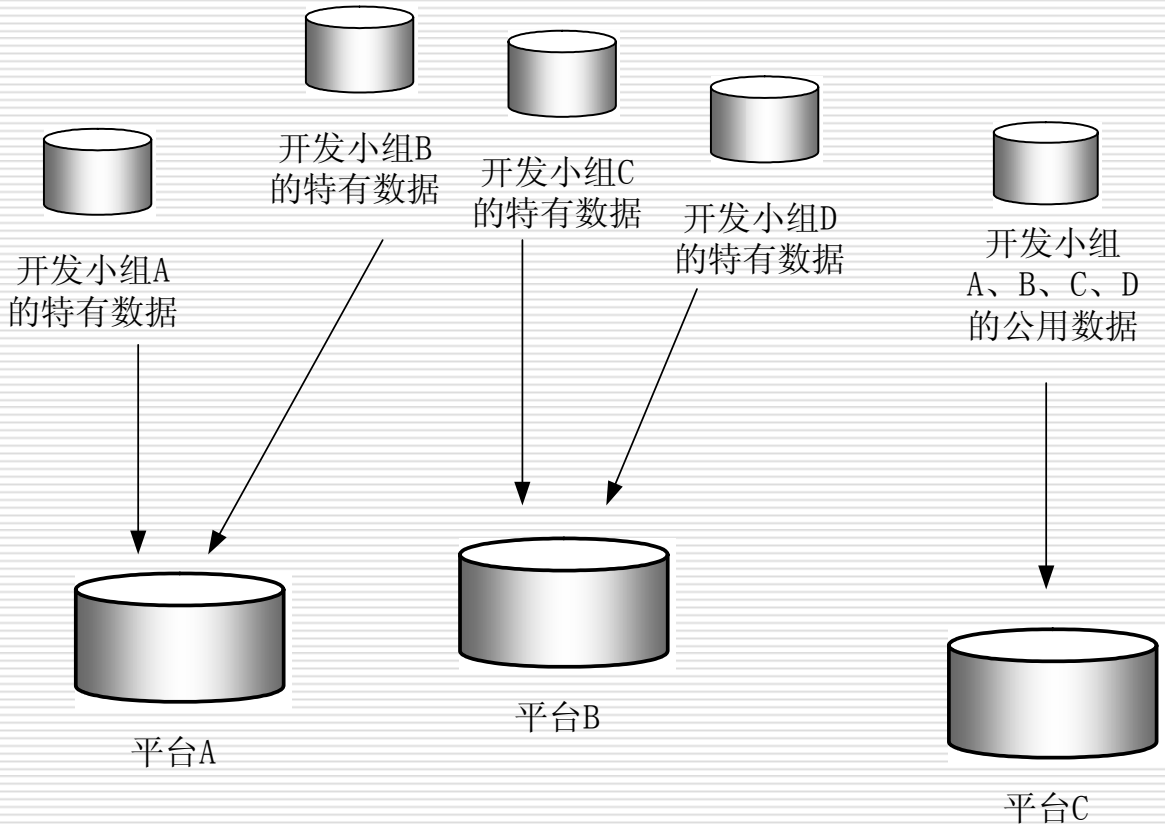
各自拥有不同的非分布式部分（8）



- 数据仓库细节级不同类型的数据都在同一个技术平台上
  - 所有表以同一个技术平台存放，代价低，维护和培训费也低，运行效果好

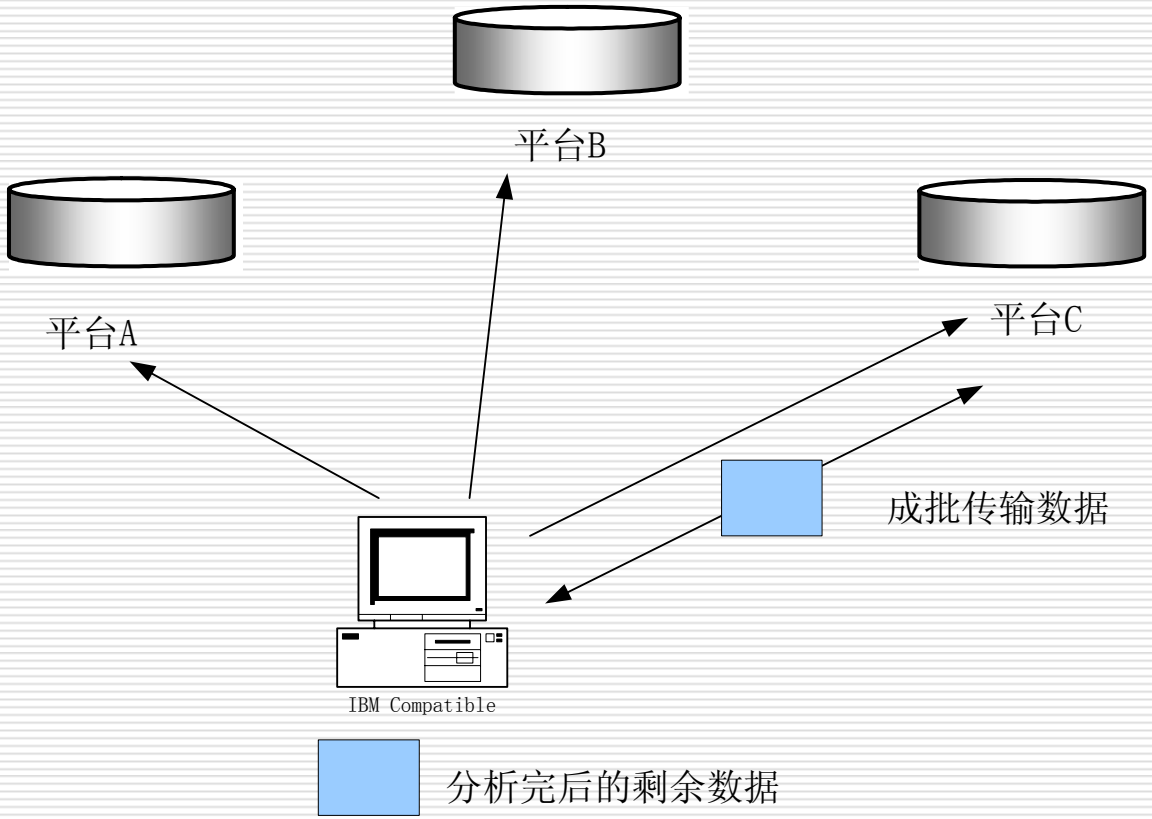
各自拥有不同的非分布式部分（9）

- 数据仓库细节级数据的不同部分分散在不同的技术平台上
  - 合理性：
    - 能很好地满足企业内不同的策略需求
    - 不同的开发小组能对自己特有的需求具有一定程度的控制能力
  - 缺陷：
    - 必须购买和支持多个技术平台
    - 最终用户必须接受多种技术培训
    - 各种技术可能很难融合在一起

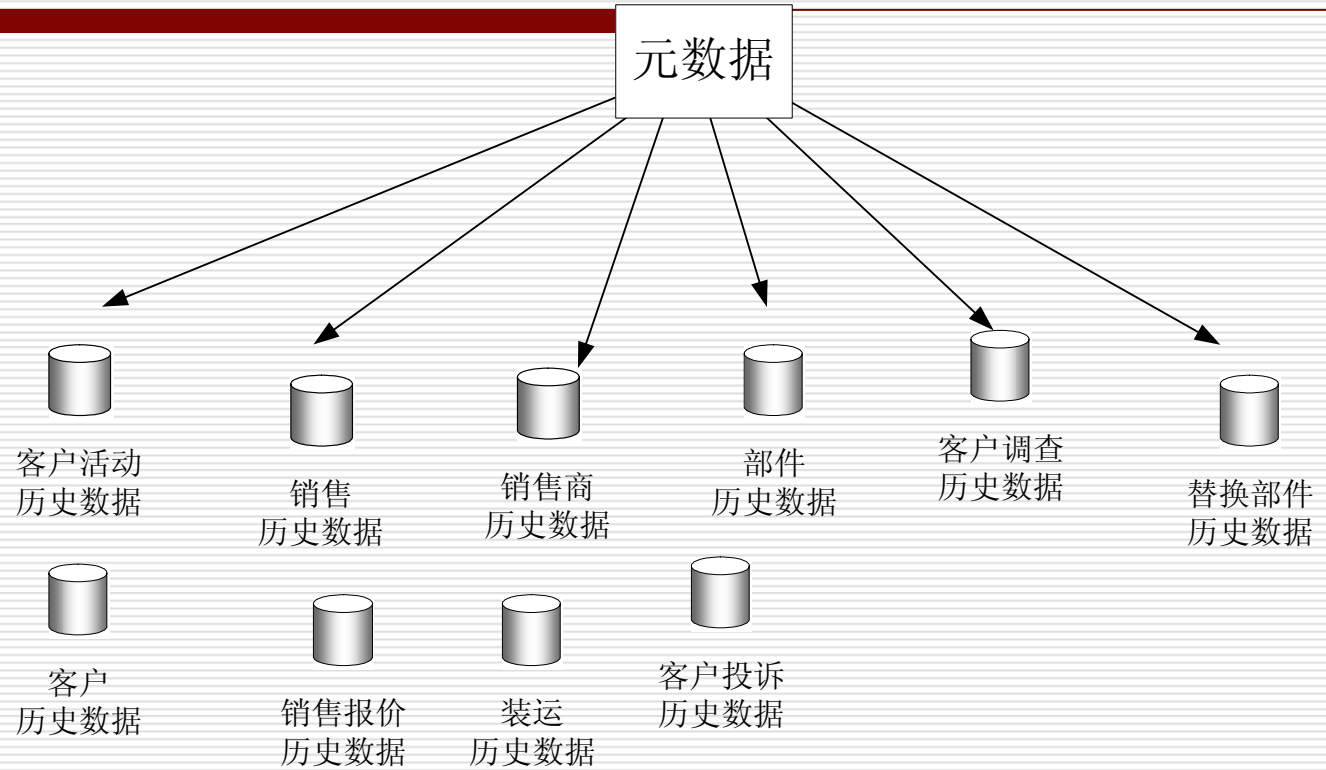


各自拥有不同的非分布式部分（10）

- 不同平台间的接口问题
  - 数据传输
    - 多接口技术用于大量的数据传输，那么接口软件将成为性能的瓶颈
    - 细节数据位于多种平台上时，将会出现资源的利用和管理的问题
  - “剩余”细节数据
    - 当细节数据从数据仓库的一个地方传送到另一个地方后驻留在那里
    - 随意的细节数据的搬迁将会导致细节级上出现冗余数据

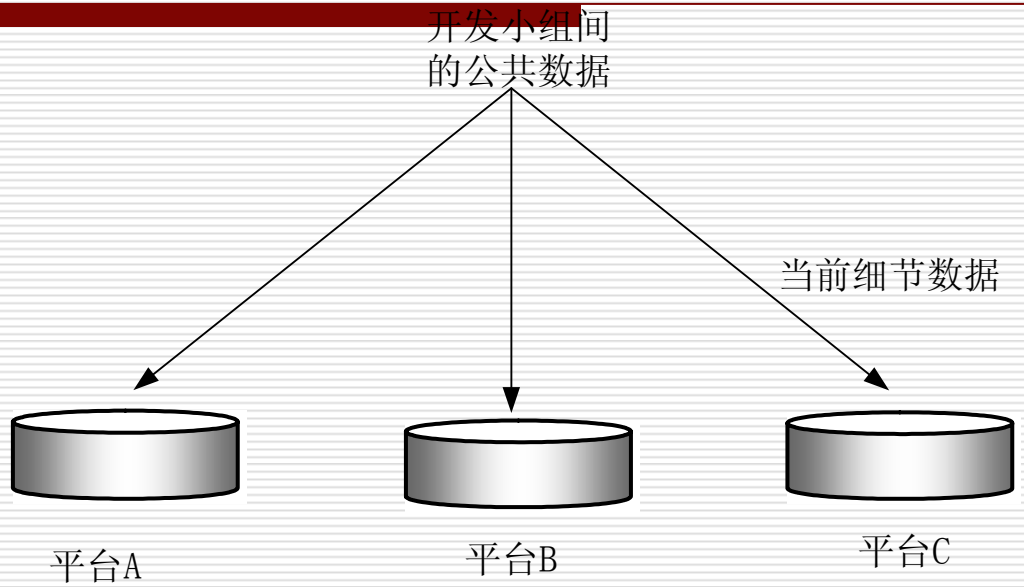


各自拥有不同的非分布式部分（11）



- 元数据位于数据仓库中实际细节数据内容的顶层
- 无论采用多种技术或是单一技术管理细节数据，元数据都起着很重要的作用

各自拥有不同的非分布式部分（12）



- 公共的细节数据采用多种开发平台
  - 公共细节数据采用多种技术平台是一种可能
  - 这不是一种很好的解决方案，因为跨多种技术平台的复杂性只能增加管理的难度

下一次……

联系信息:

E-mail: yhtong@db.pku.edu.cn

Telephone: 62757756 (O)

13701205200 (M)