第七章 数据库设计

数据库设计概述

- 数据库设计是指对于一个给定的应用领域，构造最优的数据库模式，建立数据库及其应用系统，使之能够有效地存储数据，满足各种用户的应用需求（信息要求和处理要求）数据操作需求，有效地支持各种应用系统的开发和运行。

数据库设计概述

- **数据库设计的关键是构造合理的数据模型。这个数据模型**
  - 要较好地反映现实世界信息、信息之间的联系，
  - 能反映出使用者对数据的需求和操作特点，
  - 能方便地在某个选定的DBMS支持下实现。

数据库设计的特点

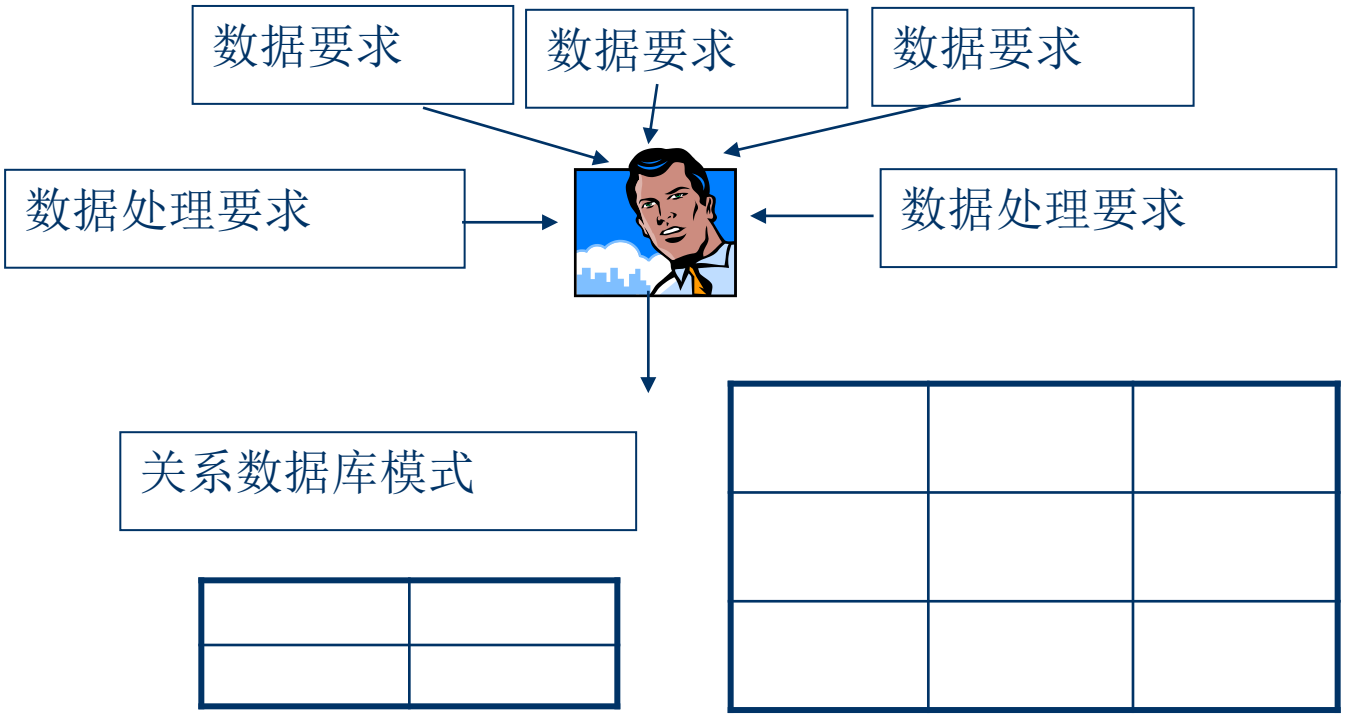
- 三分技术，七分管理，十二分基础数据
- 数据库设计与硬件、软件等紧密相关。
- 数据库设计要把结构（数据）设计和行为（处理）设计密切结合起来。

数据库设计的方法

- 手工试凑方法
- 规范化设计方法
- 自动数据库设计工具

数据库设计的方法

❖手工试凑方法

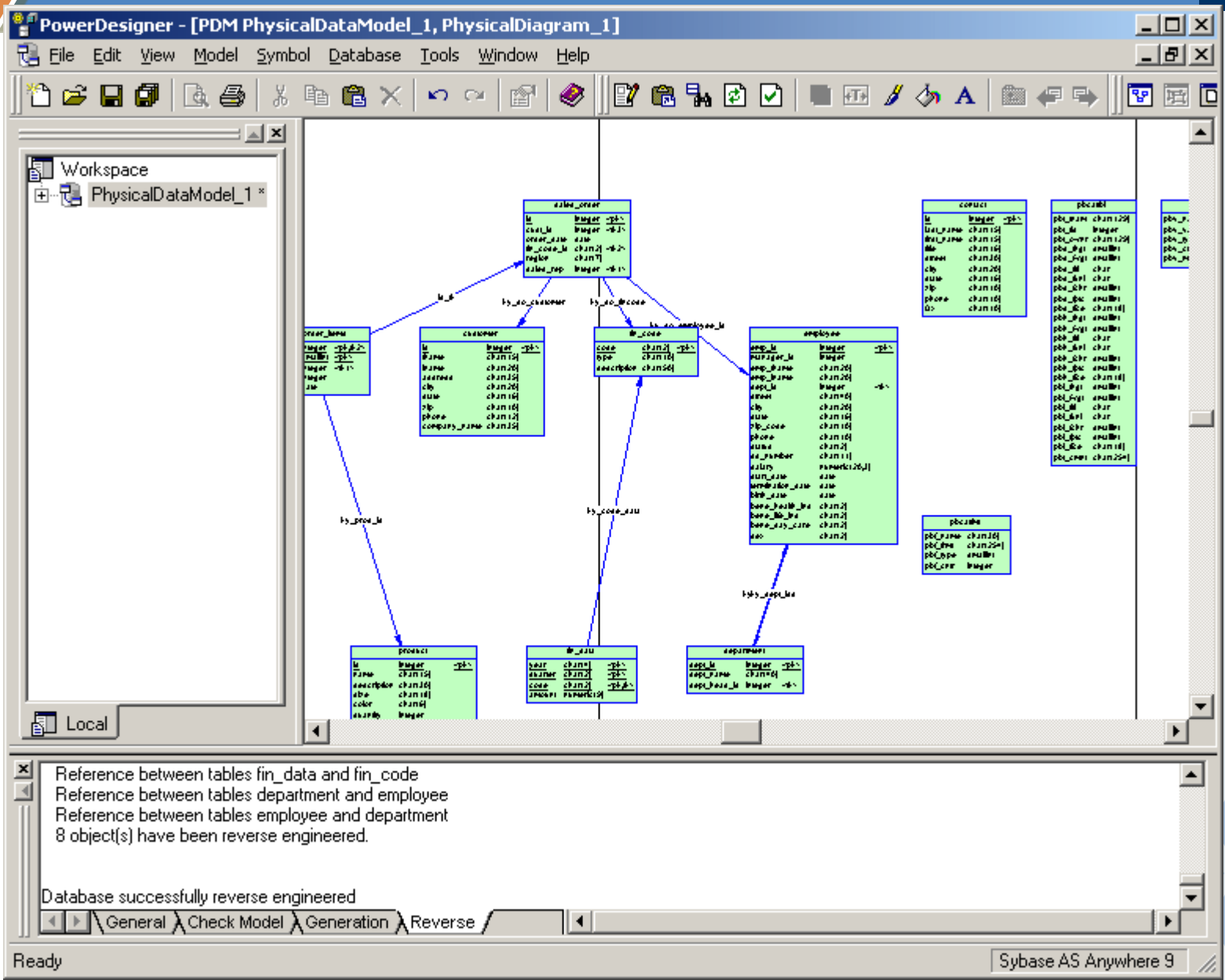


数据库设计的方法

手工试凑方法的缺点

- 现在数据库的数据量大，数据间关系复杂，用户众多，使用要求各式各样，很难仅凭经验进行设计。
- 直接设计把数据的逻辑结构，物理结构、处理要求等一起考虑，很难对模式进行评价和优化。用户需求一旦发生变化，数据结构很难随之发生变化。
- 数据库设计与具体的DBMS紧密结合，移植困难。
- 缺乏文档资料，难于与用户交流，对设计难于评审，往往到运行中才能发现问题。
- 难以由多个人合作进行设计。

数据库设计工具


PowerDesigner - [PDM PhysicalDataModel_1, PhysicalDiagram_1]
File Edit View Model Symbol Database Tools Window Help
Workspace
PhysicalDataModel_1 *
Local
Reference between tables fin_data and fin_code
Reference between tables department and employee
Reference between tables employee and department
8 object(s) have been reverse engineered.
Database successfully reverse engineered
General Check Model Generation Reverse
Ready
Sybase AS Anywhere 9

数据库设计的方法

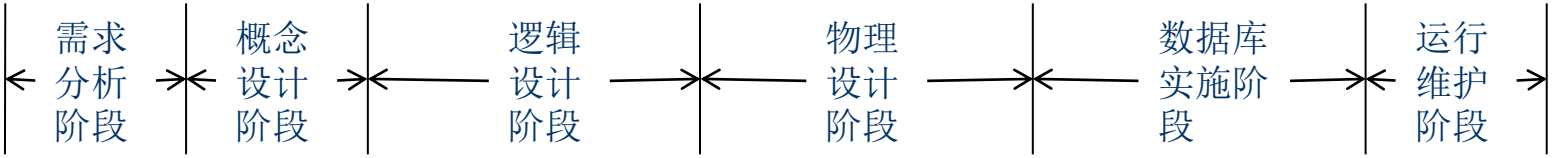
规范化设计方法

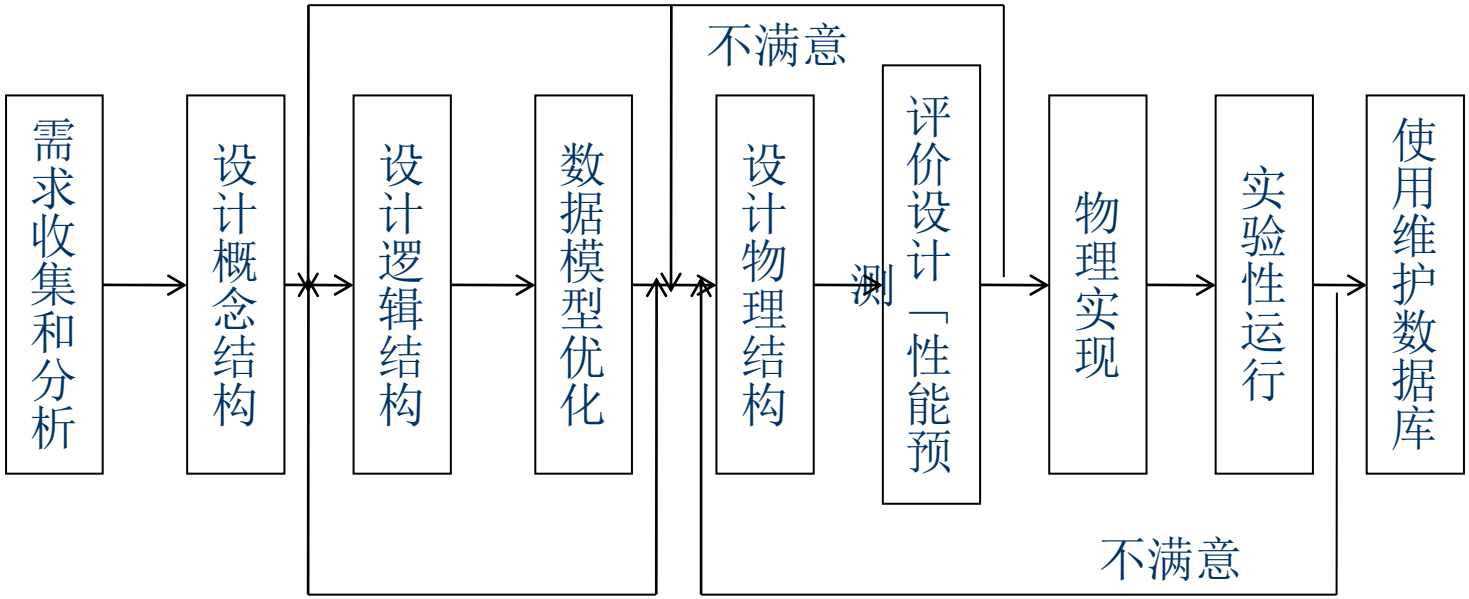
- 规范化设计方法认为数据库设计涉及了很多问题，每类问题有其不同的自然论域。规范化设计方法依据软件工程的思想，把整个设计过程划分为若干阶段，把数据库设计这一复杂的大问题分为若干相对简单的小问题，每个阶段只解决整个设计中的部分问题。整个设计方法是迭代过程，每一过程完成时要进行设计分析，产生各种设计文档，并组织评审和用户交流，如不满足要求则进行修改。

数据库的生命周期

- 需求分析
- 概念结构设计
- 逻辑结构设计
- 物理结构设计
- 数据库的实施
- 数据库运行与维护

数据库的生命周期


需求分析阶段
概念设计阶段
逻辑设计阶段
物理设计阶段
数据库实施阶段
运行维护阶段


需求收集和分析
设计概念结构
设计逻辑结构
数据模型优化
设计物理结构
评价设计「性能预测
物理实现
实验性运行
使用维护数据库
不满意
不满意

规范化设计的特点

分步进行

- 数据库设计常常由不同的人员分阶段进行。这样做，一是由于技术上分工的需要，二是为了分段把关，逐级审查，保证设计的质量和进度。

反复性

- 数据库设计不可能“一气呵成”，需要反复推敲和修改才能完成。前阶段的设计是后阶段设计的基础和起点，但后阶段也可向前阶段反馈其要求，如此反复修改，以臻完善。

试探性

- 数据库设计结果一般不是唯一的。设计的过程是个试探的过程。在设计过程中，有各式各样的要求和制约因素，它们之间往往是矛盾的。数据库的设计很难说是最佳的，常常得之于东，而失之于西，何去何从取决于数据库设计者的权衡和单位的决策。

需求分析

- 需求分析的任务是通过详细调查现实世界要处理的对象（组织、部门、企业等），充分了解原系统（手工系统或计算机系统）的工作情况，明确用户的各种需求，并预测系统今后可能的扩充和改变，然后在此基础上确定新系统的功能。

需求分析

- **调查的重点是“数据”和“处理”，包括：**
  - 信息要求，指用户需要从数据库中获得的信息的内容与性质。从中可以导出数据要求。
  - 处理要求，指用户要完成什么处理功能，对处理的响应时间和处理方式的要求。
  - 安全性与完整性的要求

需求分析的具体步骤

- 调查组织机构情况
- 调查各部门的业务活动情况
- 在熟悉了业务活动的基础上，协助用户明确对新系统的各种要求，包括信息要求、处理要求、完整性与安全性要求。
- 确定新系统的边界
- 预测系统的未来改变
- 强调用户参与

数据字典

- 数据字典是系统中各类数据描述的集合，是进行详细的数据收集和数据分析所获得的主要成果。
- 数据字典包括：
  - 数据项
  - 数据结构
  - 数据流
  - 数据存储
  - 处理过程

数据字典

- **数据项描述**
  - {数据项名，数据项含义说明，别名，数据类型，长度，取值范围，取值含义，与其它数据项的逻辑关系，数据项之间的联系}
- **数据结构描述**
  - {数据结构名，含义说明，组成：{数据项或数据结构}}
- **数据流描述**
  - {数据流名，说明，数据流来源，数据流去向，组成：{数据结构}，平均流量，高峰期流量}

数据字典

- **数据存储描述**
  - {数据存储名，说明，编号，输入的数据流，输出的数据流，组成{数据结构}，数据量，存取频度，存取方式}
- **处理过程描述**
  - {处理过程名，说明，输入：{数据流}，输出：{数据流}，处理：{简要说明}}

预测现行系统的未来改变

- **现行系统的未来改变信息是其他数据库设计阶段的参考信息。通过这些信息，我们可以考虑使最终数据库尽量适应未来改变，减少将来为适应改变而引起的数据库修改或重新设计，使数据库具有较好的适用性。**
  - 应用领域中已有的，但目前尚未被数据库系统支持的应用。
  - 应用领域中各种应用功能可能的扩充、减少和改变。
  - 应用领域的上述改变对数据库支持的信息和应用范围、数据项定义、数据项之间的关系和数据库操作任务的影响。

强调用户的参与

- 设计人员应当与用户对于需求形成共同的理解，并共同确保数据库的设计正确表达和解决了用户的需求，对设计工作的结果承担共同的责任。

概念结构设计

- 将需求分析得到的用户需求抽象成为信息结构即概念模型的过程就是概念结构设计。

概念结构的特点

- 能真实、充分地反映现实世界，包括事物和事物之间地联系，能满足用户对数据处理地要求。
- 易于理解，从而可以用它和不熟悉计算机的用户交换意见。
- 易于更改，当应用环境和应用要求发生改变时，容易对其进行修改和扩充。
- 易于向关系、网状、层次等各种数据模型转换

分析用户需求

- **根据需求分析的结果，对应用领域进行分析，抽象出下列信息：**
  - 应用领域的流动信息的定义。
  - 应用领域的存储信息的定义。
  - 应用领域中各种流动信息的起点和源点。
  - 应用领域的各种应用的定义，包括输入信息，输出信息和应用功能的定义。
  - 上述四者之间的联系的定义。

定义数据库系统支持的信息与应用

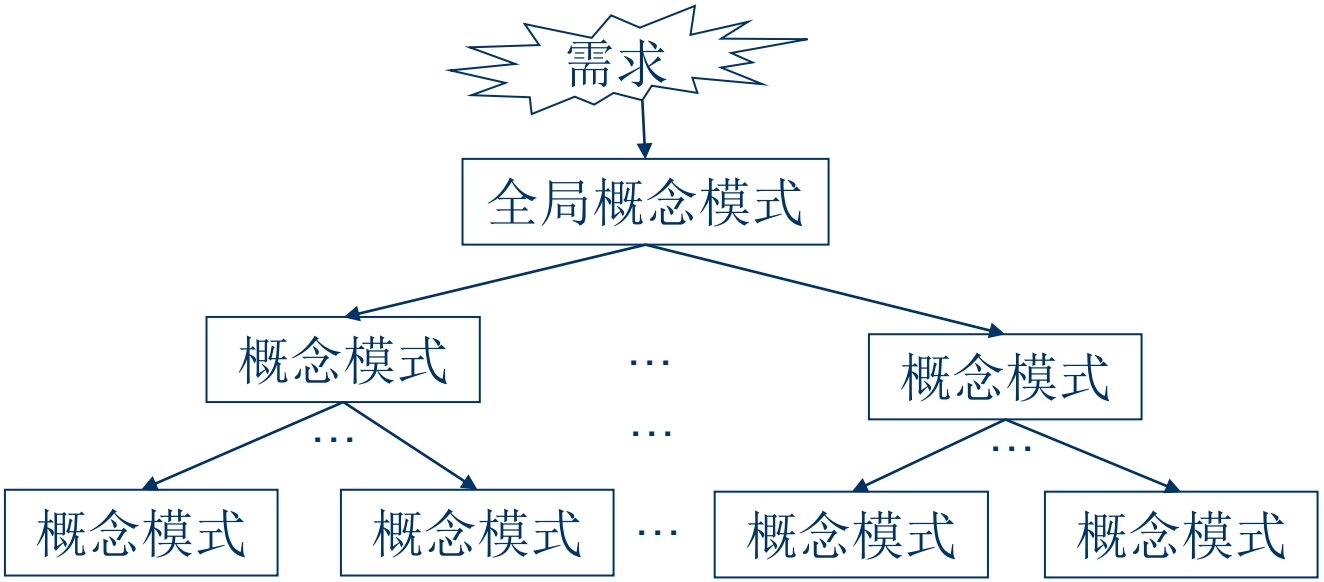
- **应用定义的目的是确定最终数据库支持哪些应用系统。应用领域的逻辑模型是应用定义的基础**
  - 考察数据流图中每个数据处理应用，确定我们正在设计的数据库是否应该而且可能支持这个应用。
  - 对于步骤一确定的每个数据库系统应该支持的应用进行严格定义，内容包括：应用名，处理功能，输入信息和输出信息。

定义数据库系统支持的信息与应用

- **信息定义的目的是确定最终数据库需要存储哪些信息。信息的定义也以应用领域的逻辑模型为基础。**
  - 考察数据流图中每个存储信息，确定其是否应该而且可能由数据库存储。
  - 对于步骤一确定的每个需要数据库存储的信息进行严格定义，内容包括：信息名、内容定义、产生该信息的应用和引用该信息的应用。

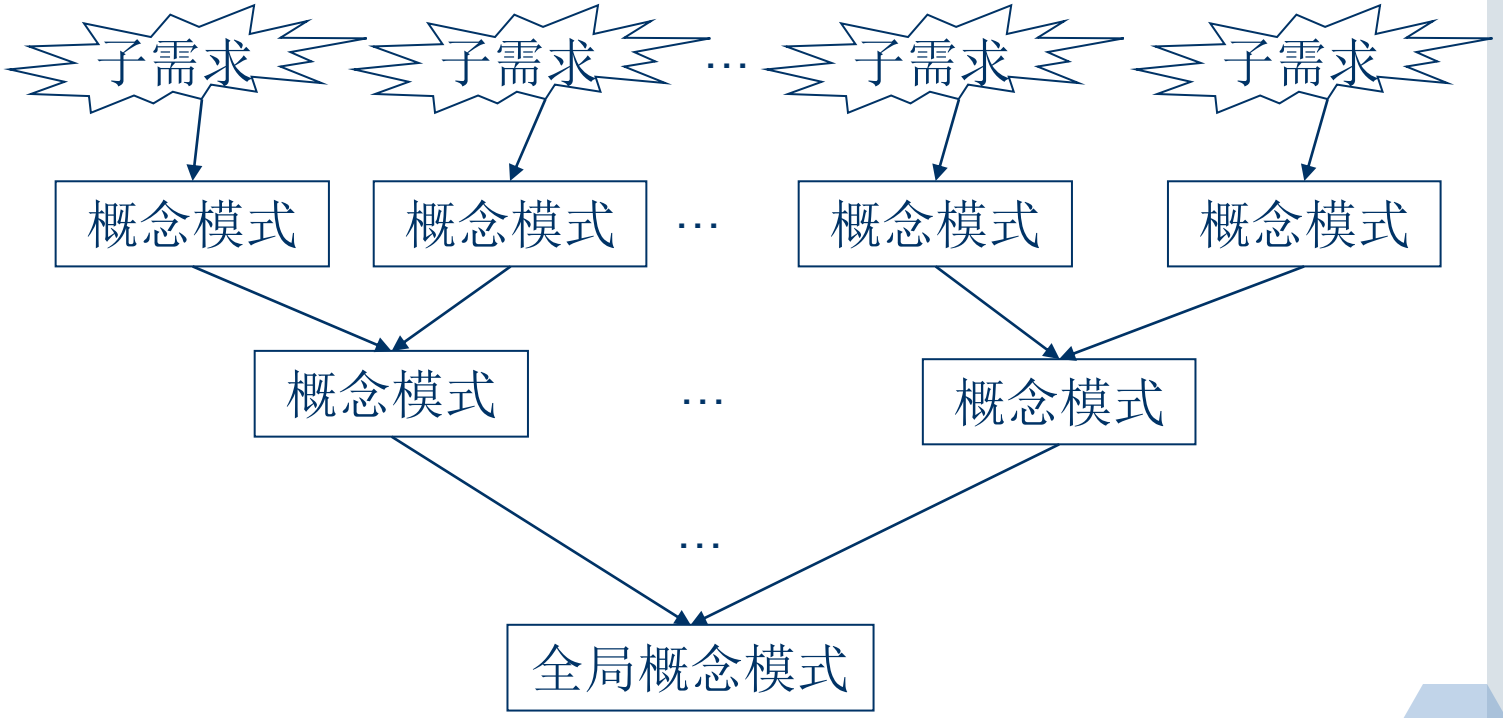
概念结构设计的策略

- **自顶向下**
  - 首先定义全局概念结构的框架，然后逐步细化。



概念设计的策略

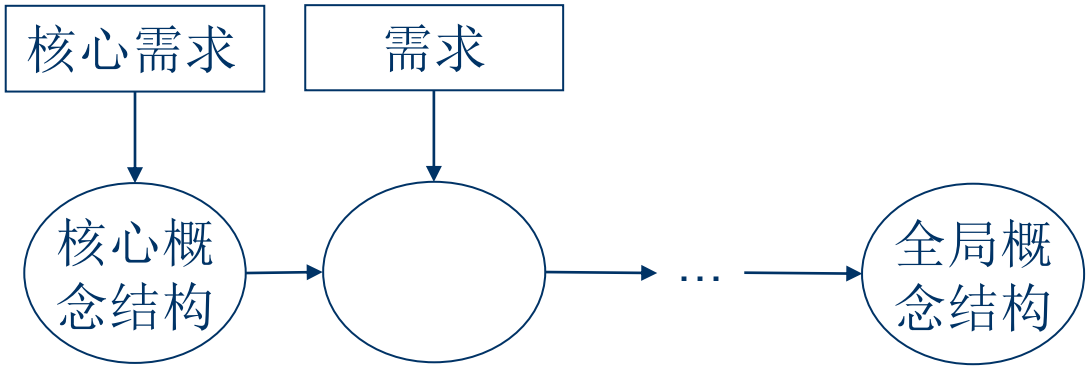
- **自底向上**
  - 首先定义各局部应用的概念结构，然后将它们集成起来，得到全局概念结构。



概念设计的策略

逐步扩张

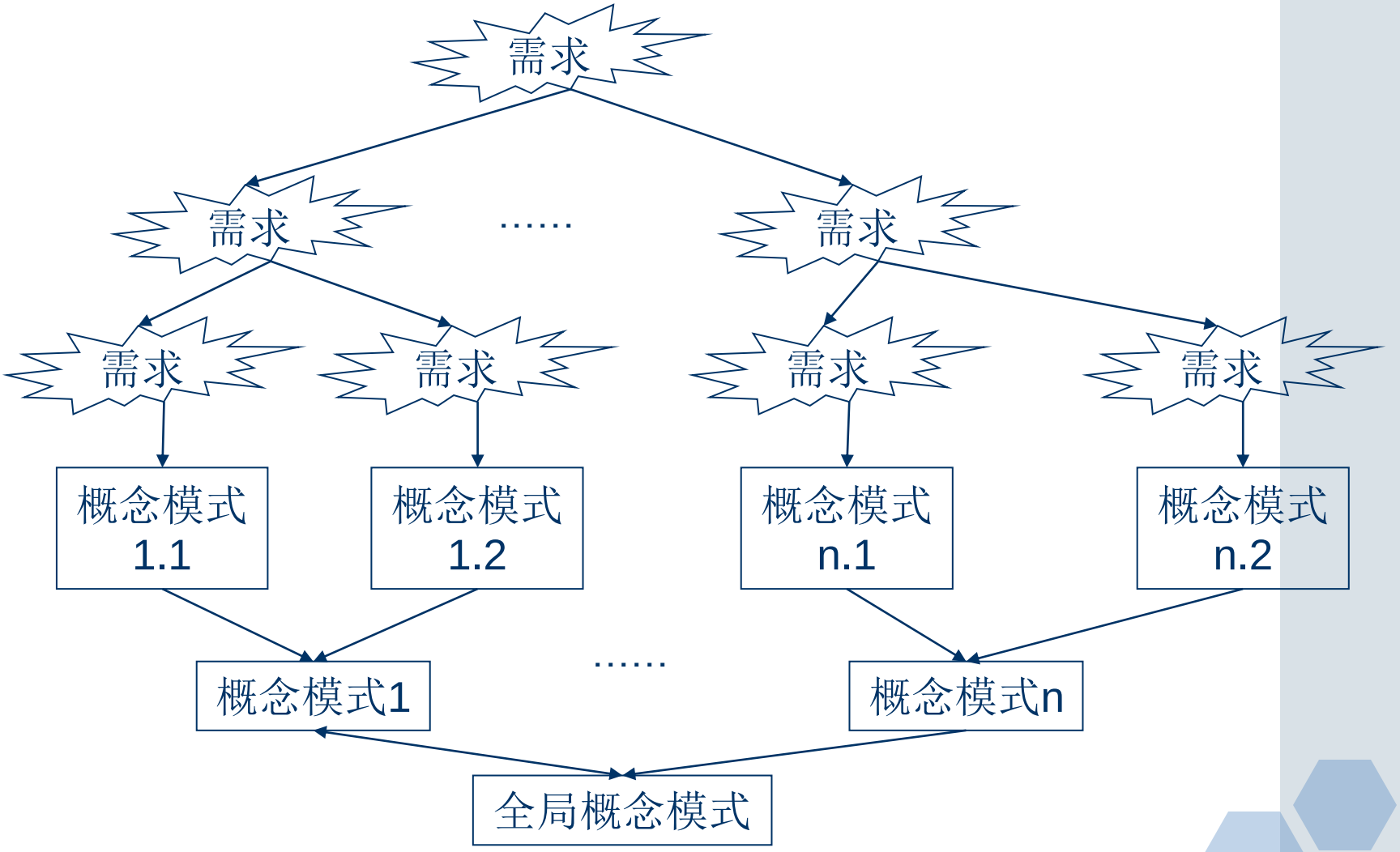
- 首先定义最重要的核心概念结构，然后向外扩充，以滚雪球的方式逐步生成其他概念结构，直至总体概念结构。



概念设计的策略

- **混合策略**
  - 即将自顶向下和自底向上相结合，用自顶向下策略设计一个全局概念结构的框架，以它为骨架集成用自底向上策略设计的各局部概念结构。

概念设计的策略


需求
需求
……
需求
需求
需求
需求
需求
概念模式
1.1
概念模式
1.2
概念模式
n.1
概念模式
n.2
……
概念模式1
概念模式n
全局概念模式

视图综合设计方法

- **视图综合设计方法分为两步：**
  - 设计局部概念结构
  - 把局部概念结构合并为一个全局概念结构。

局部视图设计

- **选择局部应用**
  - 根据某个系统的具体情况，在多层的数据流图中选择一个适当层次的数据流图，作为设计分ER图的出发点。
- **逐一设计分ER图**
  - 选择好局部应用之后，就要对每个局部应用逐一设计分ER图。

局部视图设计

- 确定局部实体。
- 确定局部实体之间的联系（包括超类/子类联系）。
- 构造局部**ER**图。

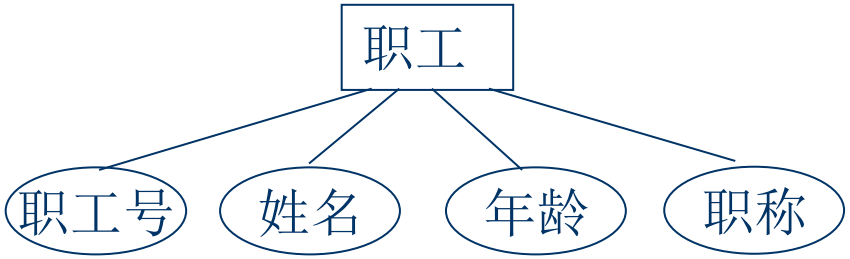
局部视图设计

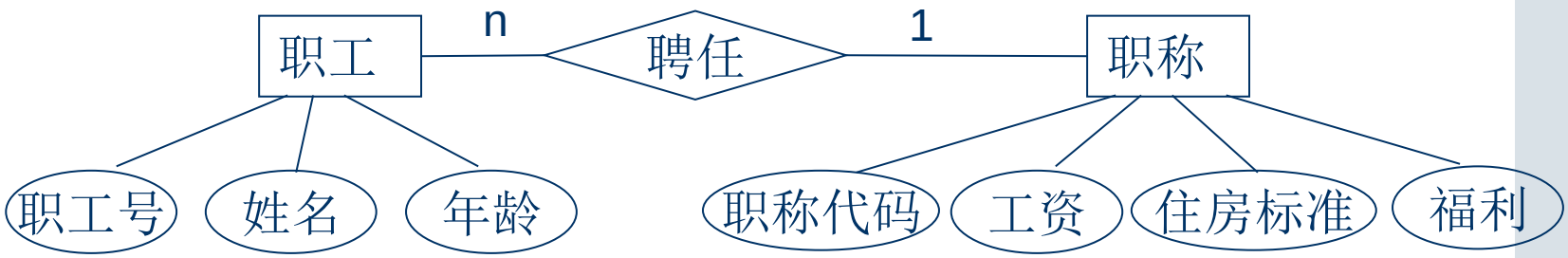
- **局部视图设计的关键在于实体和属性的正确划分。其主要手段为：**
  - 分类
    - 定义某一概念作为现实世界中一组对象的类型。这些对象具有某些共同的特性和行为。它抽象了对象值和型之间的“is member of”的语义。
  - 聚集
    - 定义某一类型的组成成分，它抽象了对象内部类型和成分之间“is part of”的语义。
  - 概括
    - 定义类型之间的一种子集联系，它抽象了类型之间的“is subset of”的语义。

实体与属性

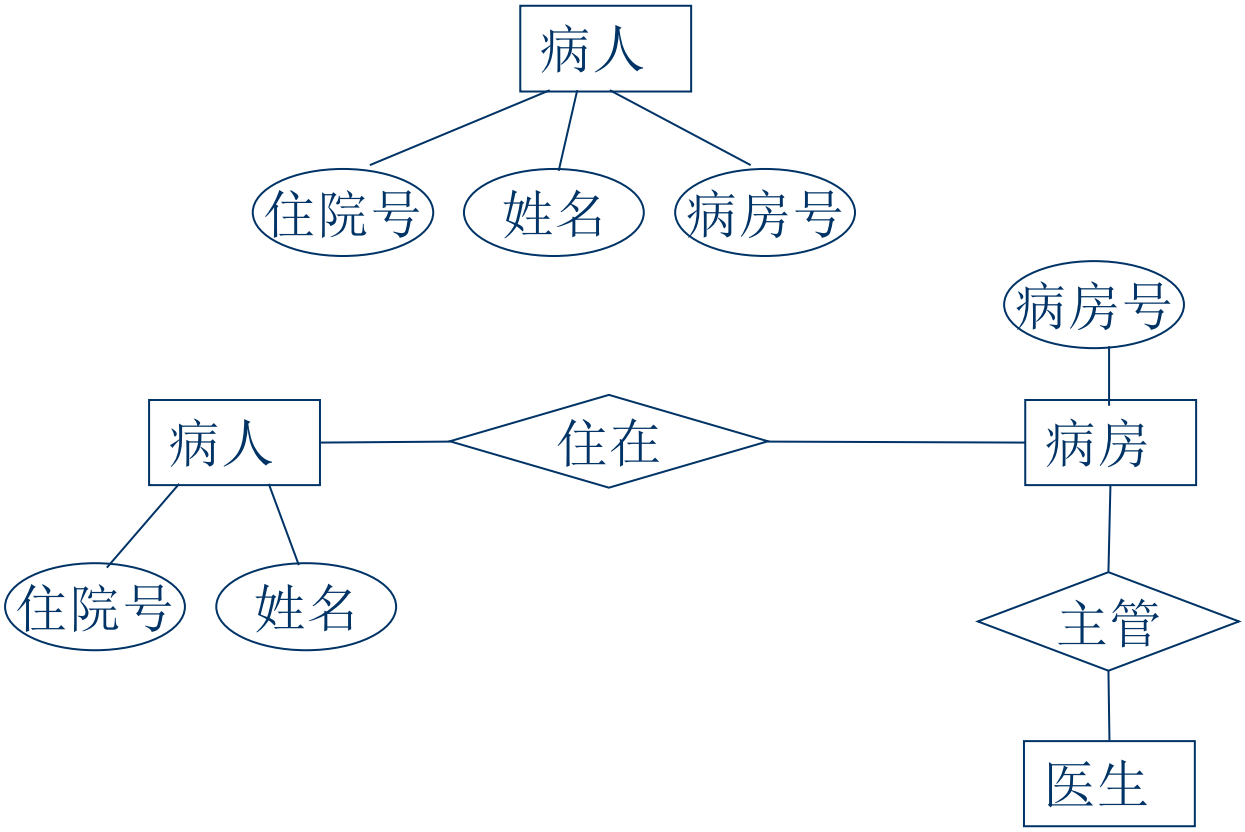
- **实体与属性划分的两条准则：**
  - 作为属性，不能再具有需要描述的性质。属性必须是不可分的数据项，不能包含其他属性。
  - 属性不能与其他实体具有联系，即ER图中所表示的联系是实体之间的联系。

实体与属性

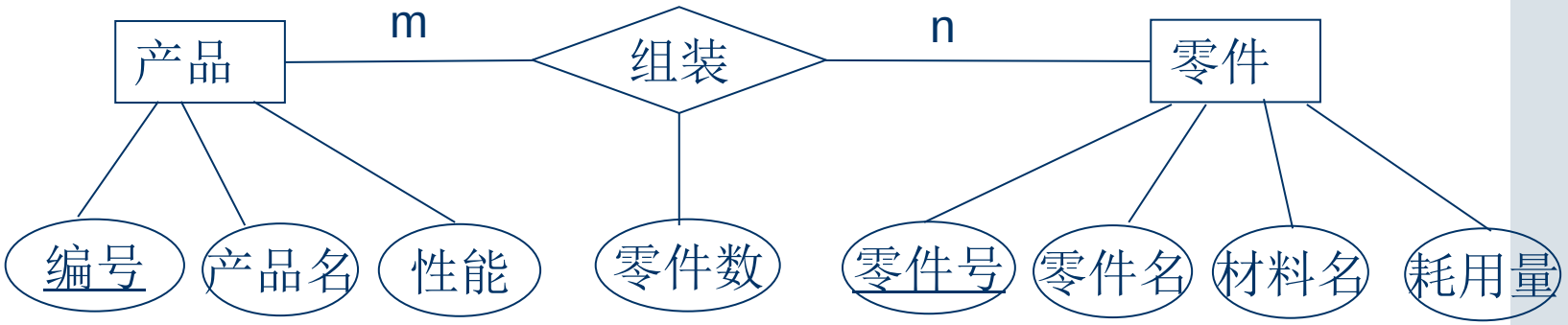




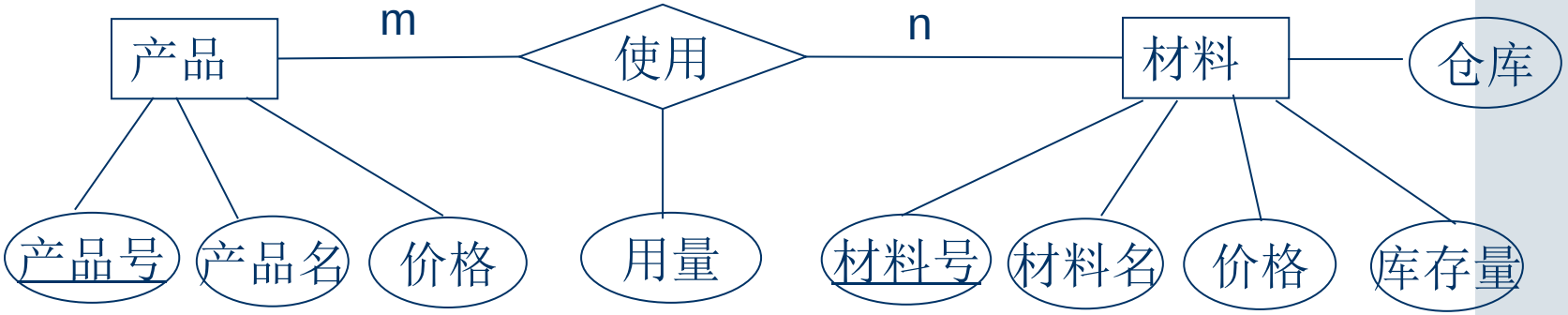
实体与属性



局部ER图设计



生产部门的局部ER图

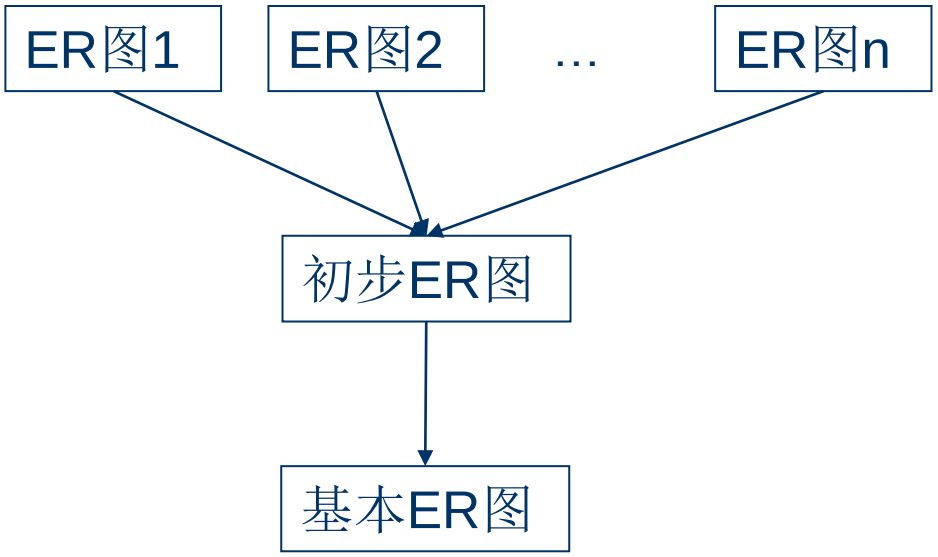


供应部门的局部ER图

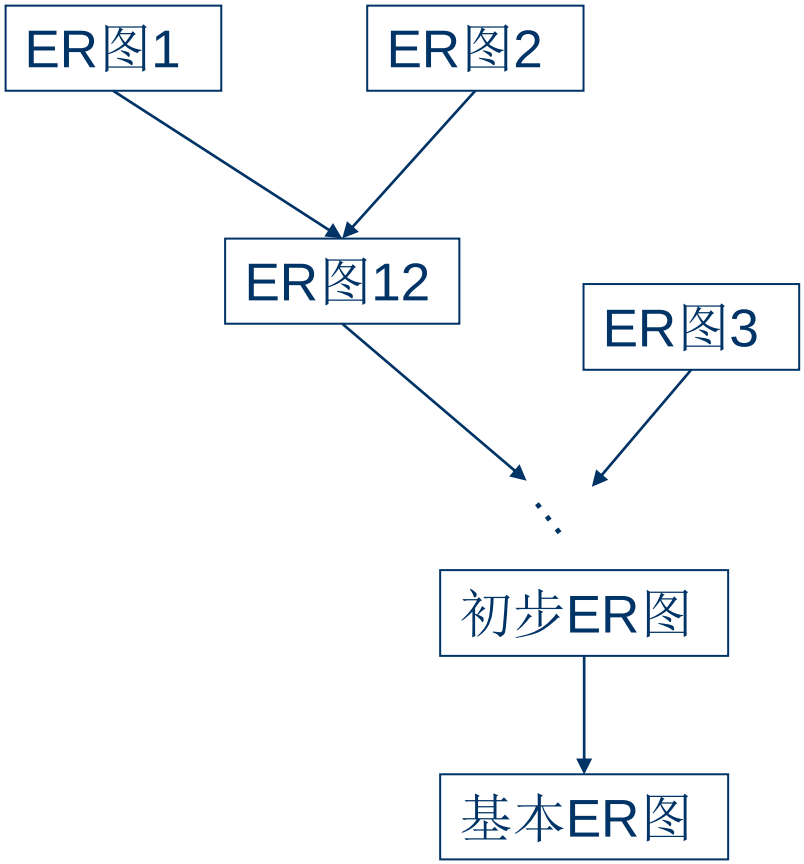
视图的集成

- **视图的集成可以有两种方法：**
  - 多个局部ER图一次集成
  - 逐步集成，用累加的方式一次集成两个分ER图。

一次集成



逐步集成



集成局部ER图的步骤

- **合并**
  - 解决各局部ER图之间的冲突，将各局部ER图合并起来生成初步ER图。
- **修改和重构**
  - 消除不必要的冗余，生成基本ER图。

全局概念模式合成

- 识别局部ER图间的冲突
- 修改局部ER图
- 局部ER图合并

识别局部概念模式间的冲突

- 各个局部应用所面向的问题不同，通常是由不同的设计人员进行局部ER图设计，所以各个局部ER图中必定存在许多不一致的地方，称之为冲突。合理消除各局部ER图的冲突时合并局部ER图的关键。各局部模式之间的冲突主要有：
  - 命名冲突
  - 结构冲突
  - 值域冲突
  - 约束冲突

命名冲突

- 同名异义
  - 即同一名字在不同的局部ER图中表示不同的概念。如“编号”这一名字在学生实体中表示学号，而在课程实体中表示课程号。
- 异名同义
  - 即同一个概念在不同的局部ER图中使用了不同的名字。如在一个局部ER图中学生实体有“何时入学”这一个属性，在另一个局部ER图中学生实体有“入学时间”这一属性，两者是同名异义。

结构冲突

- **结构冲突是模式结构的冲突。它指相同的概念在不同的局部ER图中使用不同的概念结构来表示。**
  - 同一对象在不同的局部ER图中具有不同的抽象。如“系”在一个局部ER图中被表示为一个实体，而在另一个局部ER中则被表示为一个属性或联系。
  - 同一实体在不同的局部ER图中所包含的属性个数和属性的排列次序不完全相同。

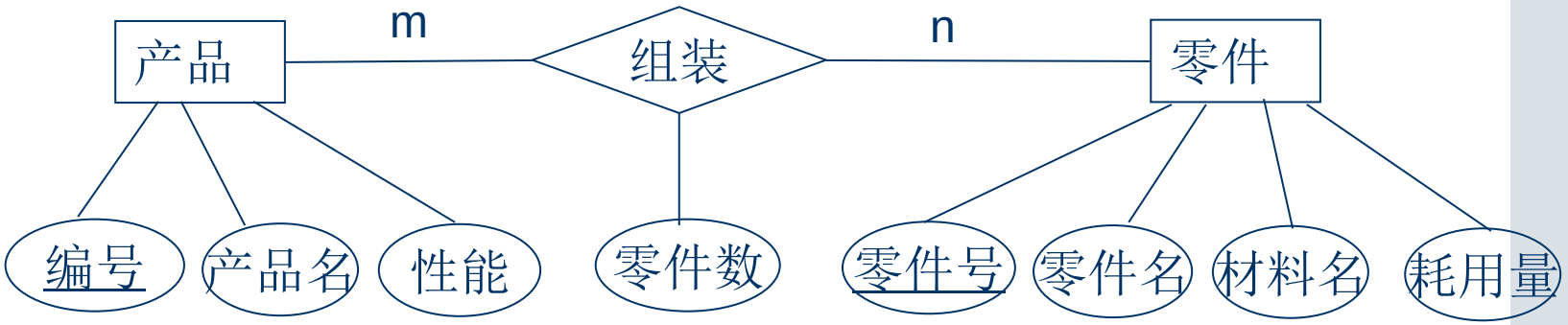
值域冲突

- **指同一个属性在不同的局部ER图中具有不同的值域定义。**
  - 属性域冲突，即属性的类型、取值范围或取值集合不同。如属性“编号”在一个局部模式中定义为整数型，而在另一个局部模式中定义为字符型。
  - 属性取值单位冲突，如数型“身高”在一个局部模式以“厘米”为单位，在另一个局部模式中以“米”为单位。

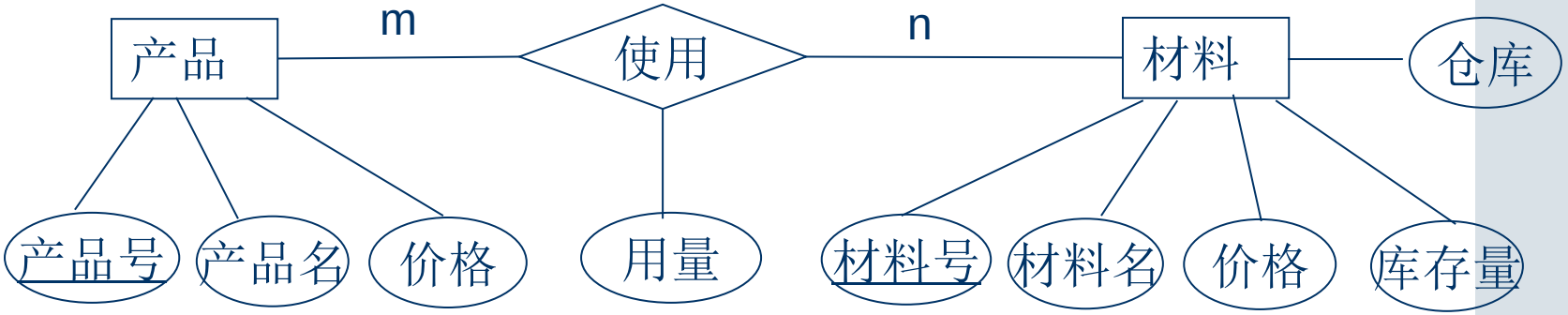
约束冲突

- 这种冲突指两个局部模式在同一个概念上定义了不同的约束。如在两个模式中，同一个实体的键不同；又如对于选课这个联系，大学生和研究生对选课的最少门数和最多门数可能不一样。

局部ER图设计

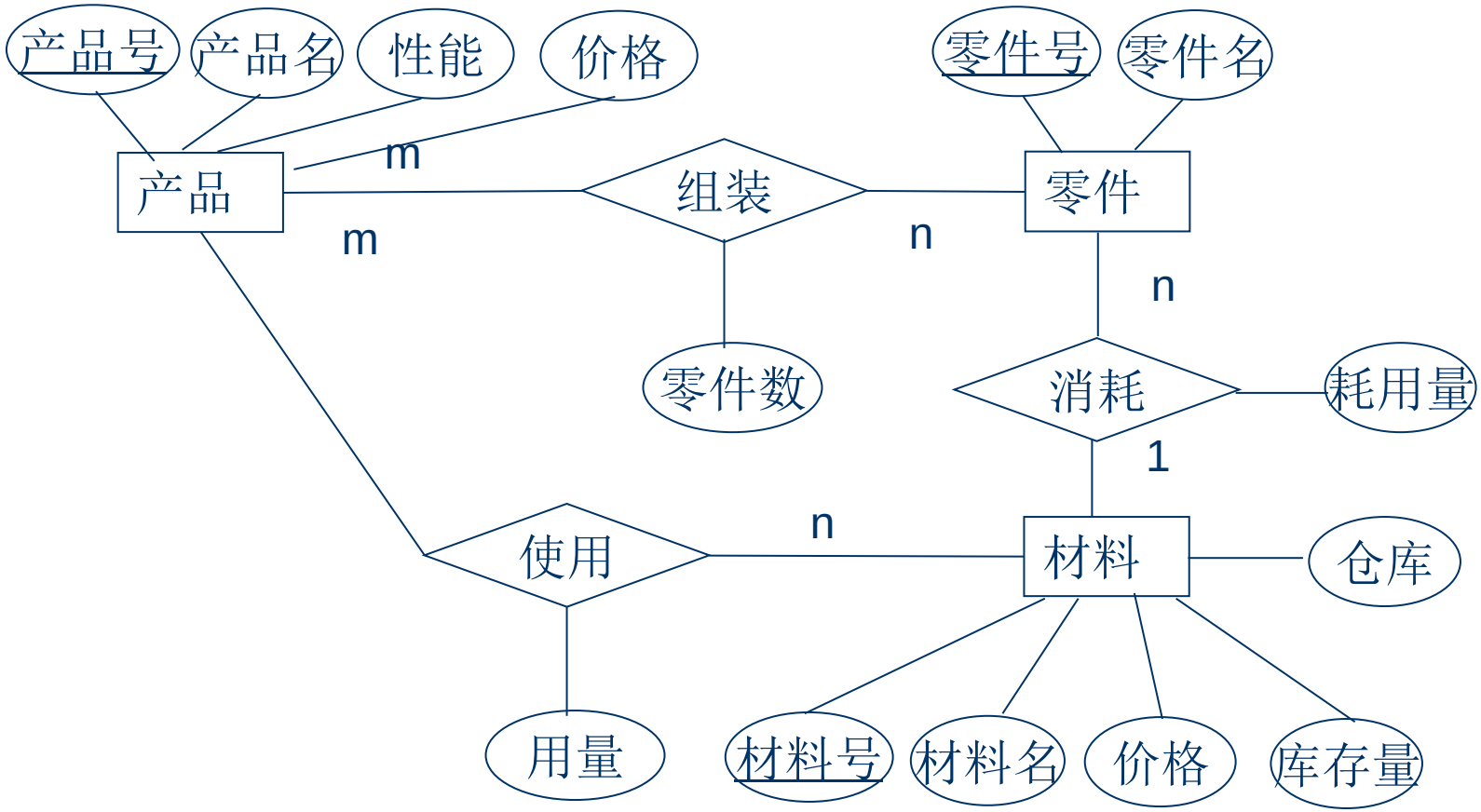


生产部门的局部ER图



供应部门的局部ER图

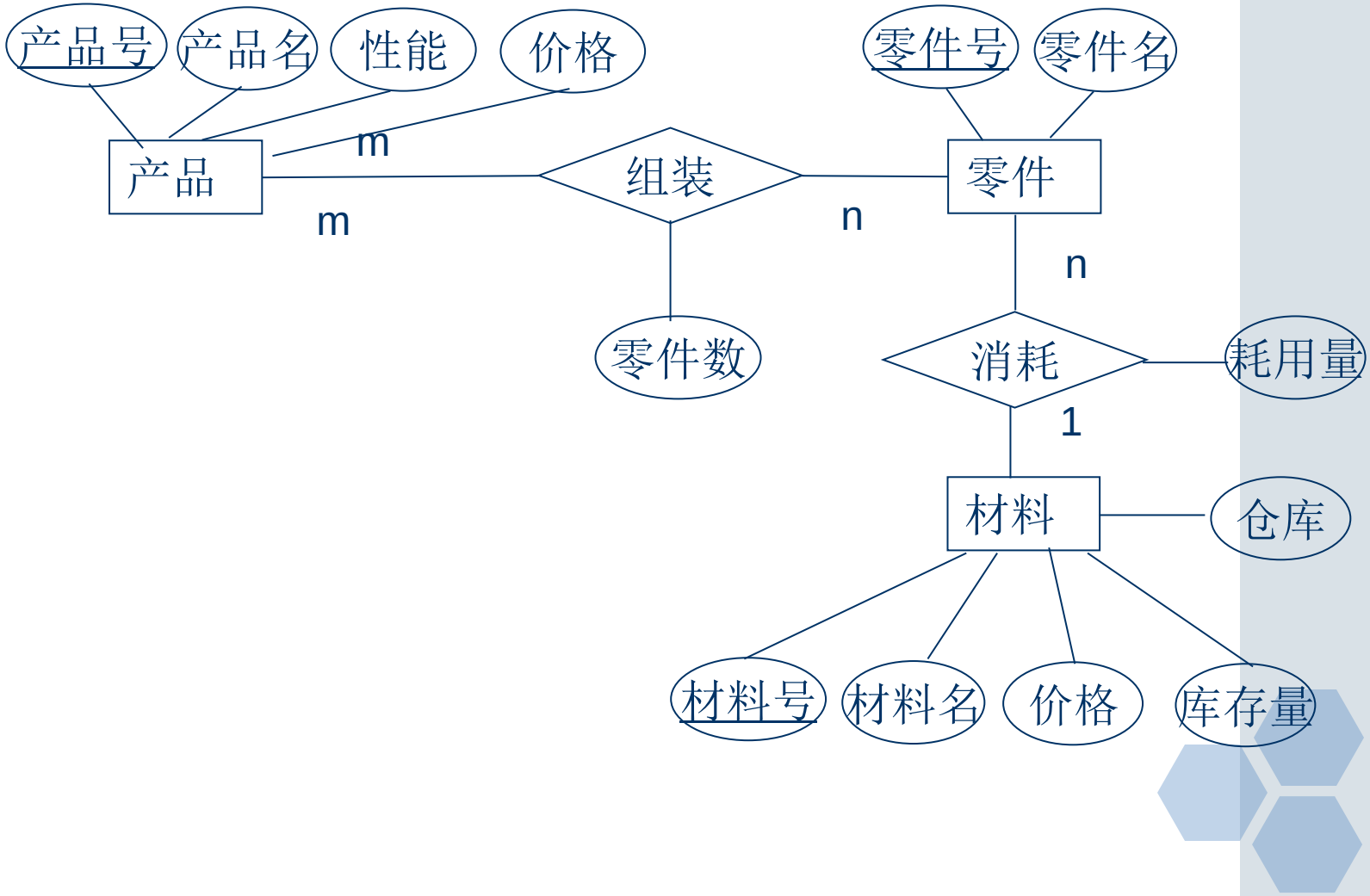
总体ER图设计


产品号
产品名
性能
价格
零件号
零件名
产品
m
m
组装
n
零件
n
零件数
消耗
耗用量
1
n
使用
材料
仓库
用量
材料号
材料名
价格
库存量

修改和重构

- 在初步ER图中，可能存在一些冗余的数据和实体间冗余的联系。所谓冗余的数据是指可由基本数据导出的数据，冗余的联系是指可由其他联系导出的联系。冗余数据和冗余的联系容易破坏数据库的完整性，应当予以消除。消除了冗余后的初步ER图称为基本ER图。消除冗余主要采用分析方法，即以数据字典和数据流图为依据，根据数据字典中关于数据项之间的逻辑关系的说明来消除冗余。

基本ER图


产品号
产品名
性能
价格
产品
m
m
组装
n
零件
零件号
零件名
n
零件数
消耗
耗用量
1
材料
仓库
材料号
材料名
价格
库存量

事务设计

- 数据库设计的目的是支持各种事务的运行。在数据库的设计中，需要考虑所有事务的特点和要求，这样才能保证所设计的数据库包含的各种事务所需要的信息。在概念数据库设计阶段，事务设计的任务是定义事务的功能，其方法是说明事务的输入、输出信息和功能。事务可以分为三类：数据查询型事务，数据更新型事务，混合型事务。

逻辑结构设计

- **逻辑结构设计的任务是把概念结构设计阶段设计好的基本ER图转换为与选用的DBMS所支持的数据模型相符合的逻辑结构。逻辑结构设计的目标包括：**
  - 满足用户的完整性和安全性要求。
  - 动态关系至少满足第三范式，静态关系至少满足第一范式。
  - 能够在逻辑级上高效率地支持各种数据库事务的运行。
  - 存储空间利用率高。

逻辑结构设计的步骤

- 形成初始关系数据库模式
- 关系模式规范化
- 关系模式优化
- 定义关系上的完整性和安全性约束
- 子模式定义
- 性能估计

形成初始关系数据库模式

- 初始关系数据库模式是指直接由概念结构设计的结果生成的关系数据库模式。初始数据库模式生成的目的是把ER图中的实体、实体间的联系等变换为关系模式。

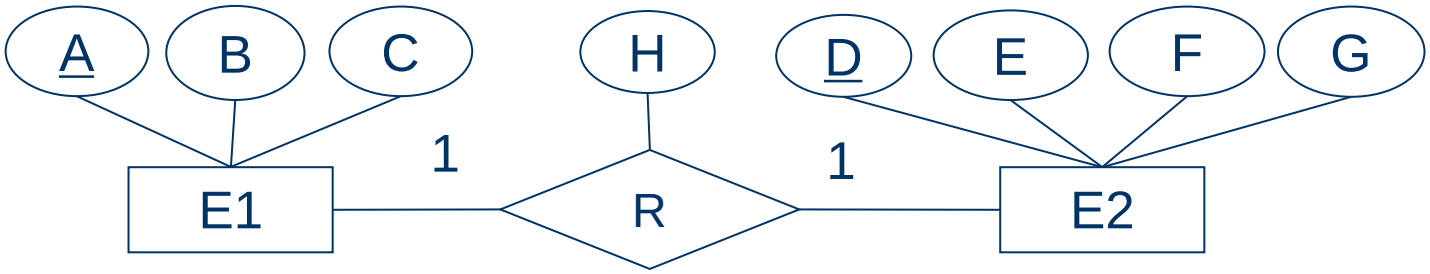
实体型的转换

- 直接将一个实体型转化为一个关系模式，实体的属性就是关系的属性，实体的码就是关系的码。

实体间联系的转换

- **1：1**联系可以直接转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，每个实体的码均是该关系的候选码。如果与某一端的实体所对应的关系模式合并，则需要在该关系模式的属性中加入另一个关系模式的码和联系本身的属性。

实体间联系的变换



- 可转换为以下关系模式
  - E1（A，B，C）
  - E2（D，E，F，G）
  - R（A，D，H） 或者R（A，D，H）
- 或：
  - E1 (A，B，C，D，H）
  - E2（D，E，F，G）

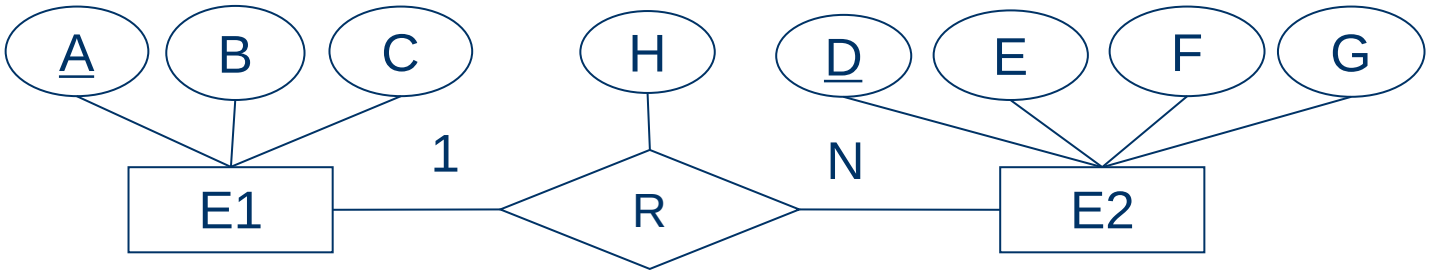
实体间联系的变换

- 或：
  - E1（A，B，C）
  - E2（D，E，F，G，A，H）

实体间联系的转换

- 1：N联系可以直接转换为一个独立的关系模式，也可以与n端对应的关系模式合并。如果转换为一个独立的关系模式，则与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为n端实体的码。

实体间联系的变换

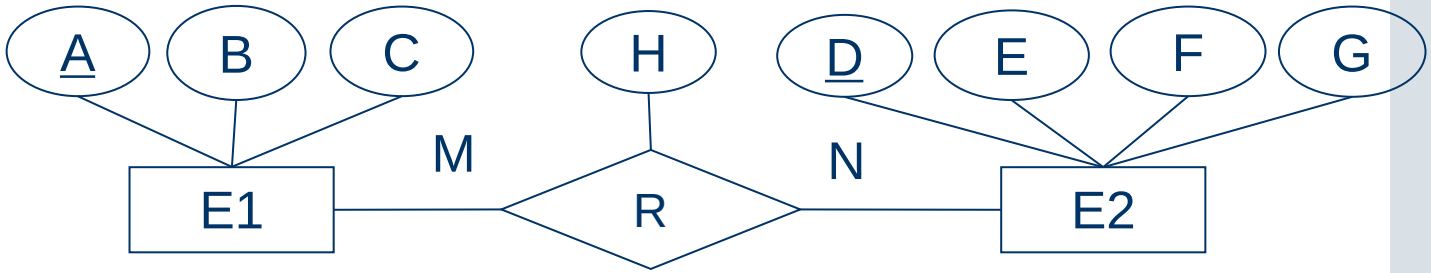


- 可转换为以下关系模式
  - E1（A，B，C）
  - E2（D，E，F，G）
  - R（A，D，H）
- 或：
  - E1（A，B，C）
  - E2（D，E，F，G，A，H）

实体间联系的转换

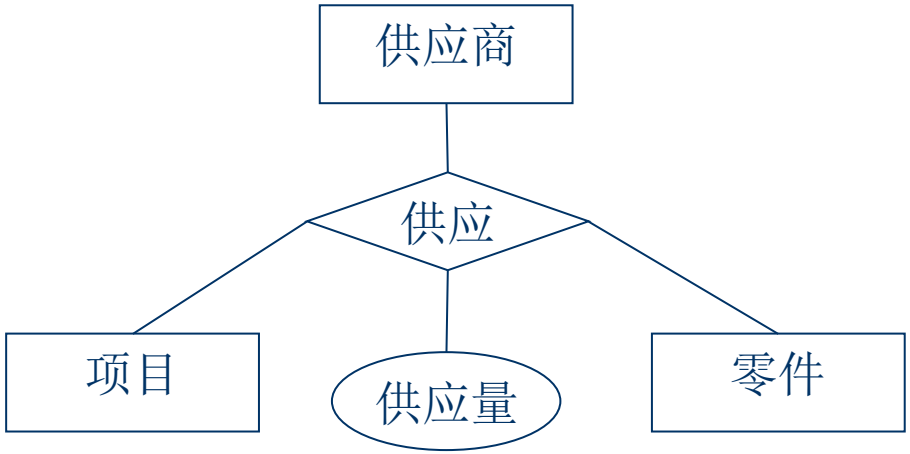
- M：N联系转换为一个独立的关系模式，与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为各实体码的组合。
- 三个或三个以上实体间的一个多元联系可以转换为一个关系模式。与该多元联系相连的各实体的码以及联系本身的属性均转换为关系的属性，而关系的码为各实体码的组合。

实体间联系的变换



- 只能转换为以下关系模式
  - R（A，D，H）

实体间联系的转换



- **可将三元联系供应转化为以下关系模式：**
  - 供应（项目编号，供应商号，零件号，供应量）

关系模式的规范化

- 确定函数依赖
- 对于各个关系模式之间的数据依赖进行极小化处理，消除冗余联系。
- 按照数据依赖的理论对关系模式逐一进行分析，对其进行规范化处理。
- 按照需求分析阶段得到的处理要求，分析这些关系模式对于这样的应用环境是否合适，是否要对某些模式进行合并或分解。

关系模式的优化

- 在数据库设计中，并非关系的规范化程度越高越好，因为在数据库的设计中不仅应当考虑数据本身的特点，还需要考虑应用的要求。规范化的主要方法是通过模式分解进行的，规范化程度越高通常也意味着分解产生的关系越多，当查询这些关系时需要进行的连接操作也就越多，从而会影响到系统的效率。因此，必须根据应用的特点对数据库设计进行优化，对关系模式进行必要的分解，提高数据操作的效率和存储空间的利用率。常用的两种分解方法是水平分解和垂直分解。

关系模式的优化

- **优化关系模式的两种方法：**
  - 水平分解

    是把关系的元组分为若干子集合，定义每个子集合为一个子关系，以提高系统效率。根据80/20原则， 在一个大关系中，经常被使用的数据只是关系的一个部分，则可以把经常使用的那一部分数据分解出来作为一个关系，其他数据作为另一个关系。如果关系R上具有n个事务，而且多数事务存取的数据不相交，则R可以分解为少于或等于n个子关系。

关系模式的优化

- **垂直分解**
  - 是把关系模式R的属性分解为若干子集合，形成若干子关系模式。垂直分解的原则是，经常在一起使用的属性从R中分解出来形成一个子关系模式。垂直分解必须确保无损连接性和保持函数依赖。

逆规范化

- 连接操作代价很高，是造成关系数据库低效的主要原因之一。如果经常需要对多个关系进行连接操作，且大多数操作为查询操作，更新很少，则可以考虑将这些关系合并为一个关系，从而提高查询效率。

设计子模式

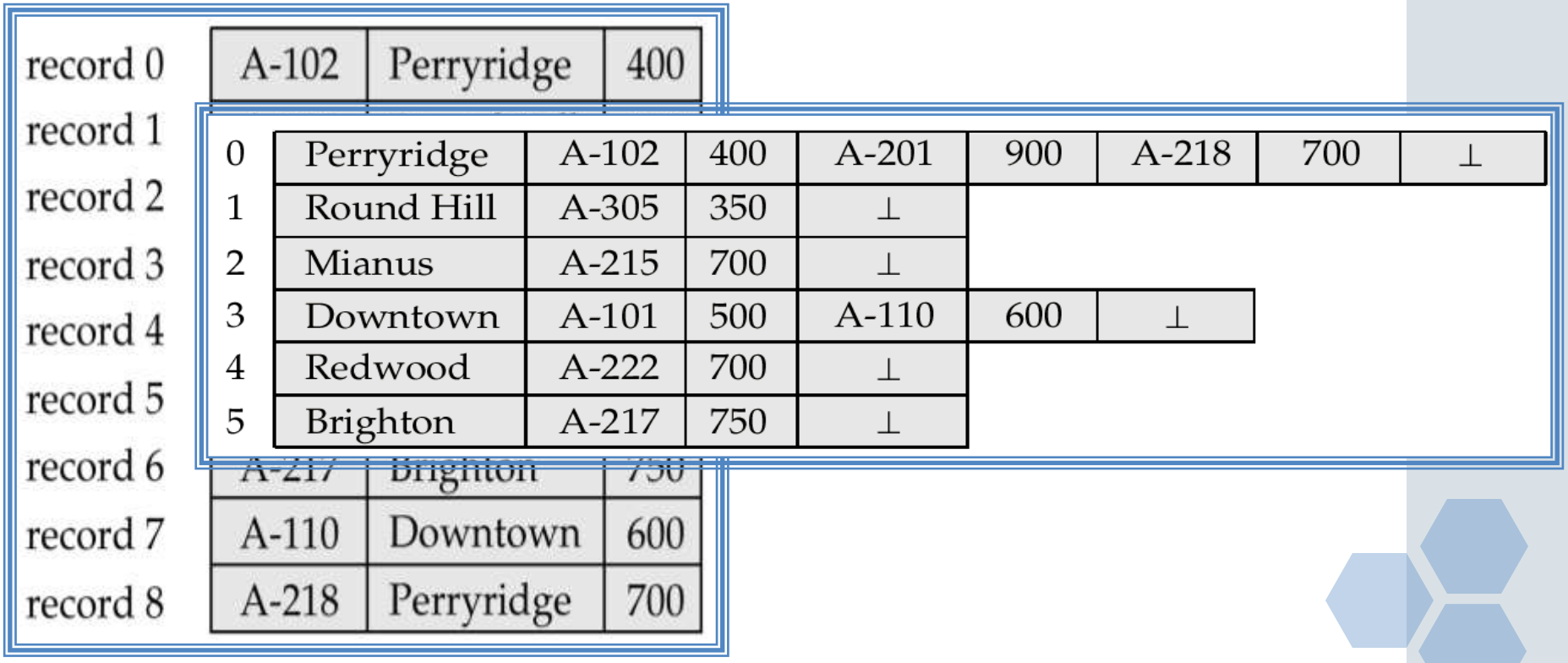
- **将概念模型转换为全局模型后，还应根据局部应用的需求，结合具体DBMS的特点，设计用户外模式。由于用户外模式与全局模式是相对独立的，因此定义用户外模式时可以注重考虑用户的习惯和方便。包括：**
  - 使用更符合用户习惯的别名
  - 可以对不同级别的用户定义不同的View，以保证系统的安全性。
  - 简化用户对系统的使用。

数据库物理设计

- 数据库在物理设备上的存储结构和存取方法称为数据库的物理结构。为一个给定的逻辑模型选取一个最合适应用要求的物理结构的过程，就是数据库物理设计。

数据库物理设计的内容和方法

- 了解应用的类型和特点
- 了解所用的**RDBMS**提供了哪些存取方法和存储结构



关系存取方法的选择

❖索引存取方法

- 经常在查询条件中出现的属性，应为其建立索

关系存取方法的选择

聚簇存取方法

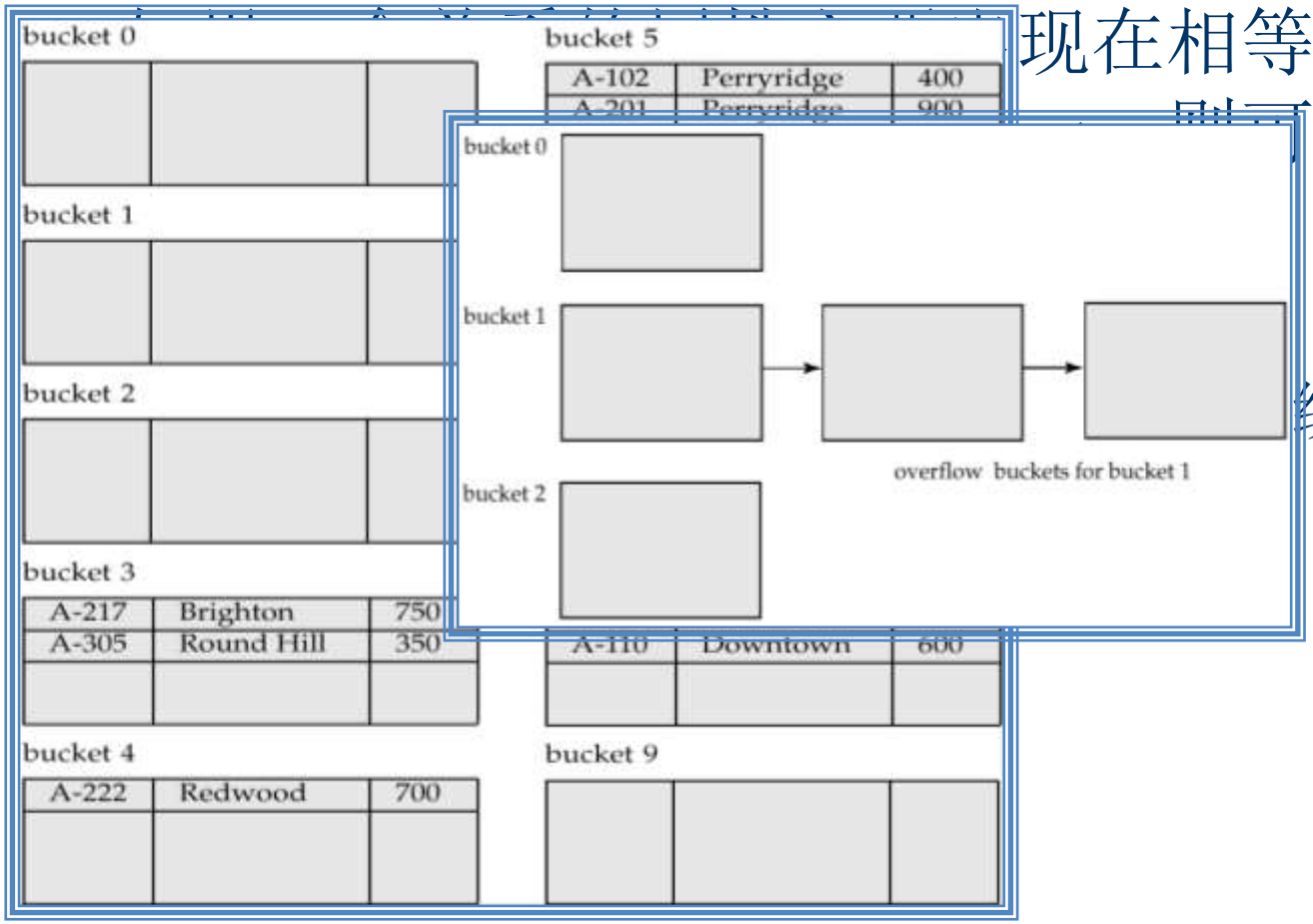
- 聚簇存取方法将相关的数据存放在连续的物理块中，从而提供查询的效率。
- 设计聚[illegible]

Hayes	Main	Brooklyn
Hayes	A-102	
Hayes	A-220	
Hayes	A-503	
Turner	Putnam	Stamford
Turner	A-305	

  - 对经[illegible]可以建立聚簇。
  - 如果[illegible]性经常出现在相等比[illegible]立聚簇。
  - 如果[illegible]性上的值重复率很高，

关系存取方法的选择

- **HASH存取方法**



现在相等比较条件

以使用

统提供了动

确定数据库的存储结构

- **确定数据的存放位置**
  - 为了提高系统性能，根据应用情况将数据的易变部分与稳定部分、存取频率高的部分与存取频率低的部分分开存放。
- **确定系统配置**

数据库的实施和维护

- **数据的载入和应用程序的调试**
- **数据库的试运行（功能和性能）**
- **数据库的运行和维护**
  - 数据库的转储和恢复
  - 数据库的安全性、完整性控制
  - 数据库的性能监督、分析和改造
  - 数据库的重组织和重构造