数据库目录

第一章：引言

1. 事务与概念

事物是所有概念的统称，概念是思维的逻辑单位。概念包含内涵和外延

- 内涵：是概念的性质和**特征**
- 外延：是满足上述性质及特征的所有客观个体的**集合**

2. 数据库与客观世界

数据库是数据化的客观世界，数据库设计是对客观世界的数据化

3. 文件管理数据阶段的弊端：P2

4. 数据库管理数据阶段的基本特点：

- 数据的管理者：DBMS(与应用独立的专门管理软件)
- 数据面向的对象：现实世界(添加和删除方便易行)
- 数据的共享程度：共享性高
- 数据的独立性：高度的物理独立性和一定的逻辑独立性(随着不同发展时期，独立性程度在不断改进)
- 数据的结构化：整体结构化
- 数据控制能力：由 DBMS 统一管理和控制

5. 模型分类（层次）

模型根据**描述用途**的不同，可分成两个层次

(1) 概念模型（信息模型）：（面向客观世界建模）
是按用户的观点来对数据和信息建模，几乎不涉及计算机专业技术知识。

(2) 数据模型：（面向计算机实现建模）
主要包括网状模型、层次模型、关系模型、对象模型等；是按计算机系统的观点对数据建模。

6. 数据模型（详见第四章）

(1) 定义
- 是一种用来**抽象、表示和处理**客观世界数据对象结构的描述方式
- 是对客观世界的模拟(一种主观建模)
- 课本 P5

(2) 满足的要求
- 形式化(书面表示/书面语言)
- 能够尽可能真实的反映客观世界
- 容易人所理解
- 便于在计算机上实现

(3) 分类：参考 4.和 P5

7. 数据库建模步骤

- 现实世界中的客观对象抽象为概念模型;
- 把概念模型转换为某一 DBMS 支持的数据模型。

8. 概念模型作用：

- 是现实世界到机器世界的一个中间层次;
- 可将业务模型与计算机实现工作隔离开;
- 复杂性减少，便于分工，成功性增大。

9. 数据(Data)、数据库(Database)、数据库管理系统(DBMS)、数据库系统(DBS)

- **数据**定义：数据是数据库中存储和管理的基本对象，是描述事物的符号记录
- 原子性：数据库中的数据与其语义是不可分的

(1) 数据库

- 定义：**长期**储存在计算机内的、有组织的、可**共享**的大量数据集合
- 特征：
    - 数据按一定的数据模型组织、描述和储存
    - 可为各种用户（应用程序）共享使用
    - 冗余度较小
    - 数据独立性较高（逻辑独立性和物理独立性）
    - 易于扩展
- 与文件比，数据库：强制约束、可扩展（大数据）、查询便利、结构无关、安全（视图）、并发、崩溃恢复

(2) 数据库管理系统

- 定义：位于用户（应用程序）与操作系统之间的一层数据库管理软件，是独立、开放的数据库管理软件（提供多种外部接口，管理的数据可以被其它外部应用程序调用）
- 用途：科学地组织和存储数据，高效地获取和维护数据
- 功能：
    - 数据定义功能：提供数据定义语言(DDL)，定义数据库中的数据对象
    - 数据操纵功能：提供数据操纵语言(DML)，操纵数据实现对数据库的基本操作（增删查改）
    - 数据库的运行管理：保证数据的安全性、完整性，多用户对数据的并发使用，发生故障后的系统恢复
    - 数据库的建立和维护功能(实用程序)：数据库数据批量装载，数据库转储，介质故障恢复，数据库的重组织，性能监视等

(3) 数据库系统

- 定义：在计算机系统中引入数据库后的系统构成。
- 构成：数据库、数据库管理系统（及其开发工具）、应用系统、数据库管理员（和用户）

10. 数据库设计分为以下四（或六）个阶段（生命周期）：

(1) 需求分析（基础条件/前提/数据源）：分为功能需求和数据需求

方式：编写数据流程图 + 含数据字典；或者采用统一建模语言UML：用例图+时序图+类图等

(2) 系统设计（核心阶段）：分为系统功能设计和数据结构设计

后者按顺序可展开成（概念结构设计（E-R 模型）、逻辑结构设计（关系模型，包含数据结构优化、数据完整性安全性设计）、物理结构设计（物理存储模型）），故也可以认为是六个阶段

其中:

1) 数据库建模阶段（**概念设计**）
根据数据需求→建立概念模型（便于面向用户交互）

2)数据库逻辑结构设计阶段（**逻辑设计**）
基于概念模型→形成商业产品支持的逻辑模型(面向计算机逻辑实现)

3)数据库物理组织设计阶段（**物理设计**）
根据逻辑模型→确定适合应用要求的物理模型(面向计算机物理实现)

⑶ **系统实现**（后期工作）：分为数据库实现（实际建立数据结构）和应用功能实现（程序代码），通过编码实现

⑷ **系统运行和维护**：分为系统运行与维护，数据运行与维护，数据分析

11. DBS 分层：课本扉页和P14

12. 存储管理器和查询管理器：P12

第二章：需求分析

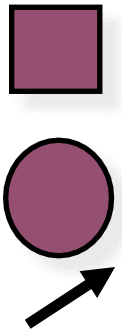
1. 概述

⑴ 基本任务：应用（功能）需求和数据需求
⑵ 其他任务：数据约束，使用频率，性能要求，使用环境，界面友好性，安全要求

(3) 基本方法：数据流图和面向对象分析方法 UML(用例图/类图/时序图/…)

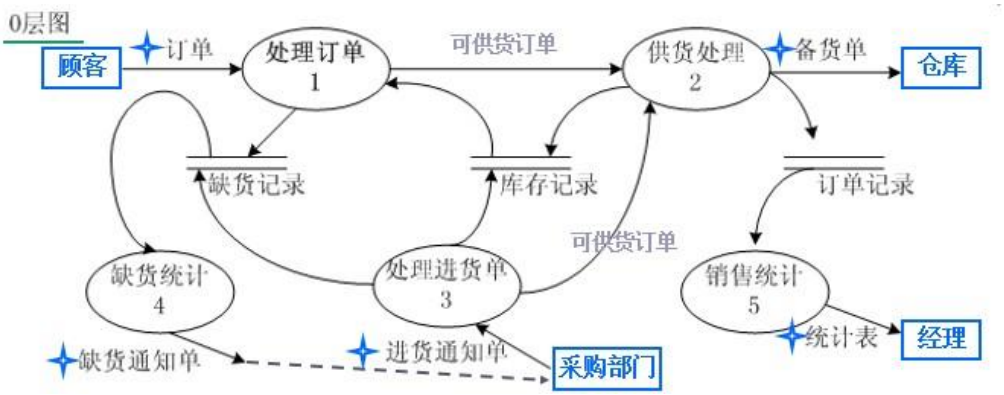
2. 数据流图

- **定义**：是一种便于用户理解和分析系统业务模型的图形化工具，它摆脱了系统实现技术的束缚，**抽象**地描述应用系统的业务模型。**数据流图表达了数据、处理过程及相互作用关系抽象的拓扑结构。**
- **顶层图**(输入输出图，仅一张)：把整个系统视为一个加工，并标出系统从外部对象接收哪些数据流和发送哪些数据流到外部对象。
- **0 层图**(顶层加工细化，仅一张)：对系统的顶层加工进行细化，标出这些加工与外部对象和与这些加工的公共保存文件间的数据流向。
- **画分层数据流图**(循环进行)：把上层加工看作是由下层多个子加工形成的子系统，画出该加工的分层数据流程图。



external entity - 外部对象

Process - 加工/处理



3. 数据字典

对数据流图中外部对象、数据流、保存文件和加工等要素的详细描述，具体包含内容有：

- **数据项的说明（是数据的最小单位）**
- **数据结构的说明（描述某些数据项之间的组成关系）**
- **数据流的说明（由一个或一组固定的数据项或数据结构组成结构，是数据结构在系统内传输的路径）**
- **加工的说明（对加工逻辑进行说明）**
- **保存文件的说明（描述具体的逻辑存储结构，不涉及物理组织，是数据结构停留或保存的地方，也是数据流的来源和去向之一）**
- **外部实体的说明（定义外部对象的编号、名称、简述等，是数据的来源和去向）**

4. 统一建模语言（UML）

采用面向对象软件工程方法来描述一个软件系统的不同部分，主要包括:

- **类图（class diagram）:描述数据对象以及对象关系**
- **用例图（use case diagram）：描述用户与系统之间的交互关系**
- **活动图（activity diagram）：描述系统的不同部分之间的任务流（任务处理过程），又称“时序图/顺序图”**

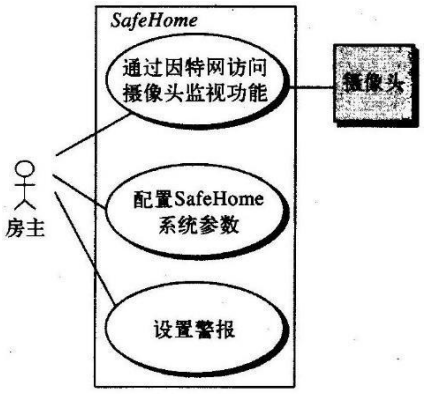


图7-6 SafeHome系统的初步的用例图

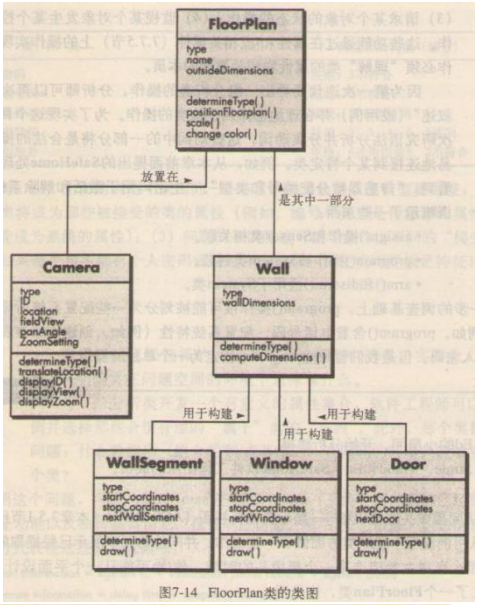


图7-14 FloorPlan类的类图

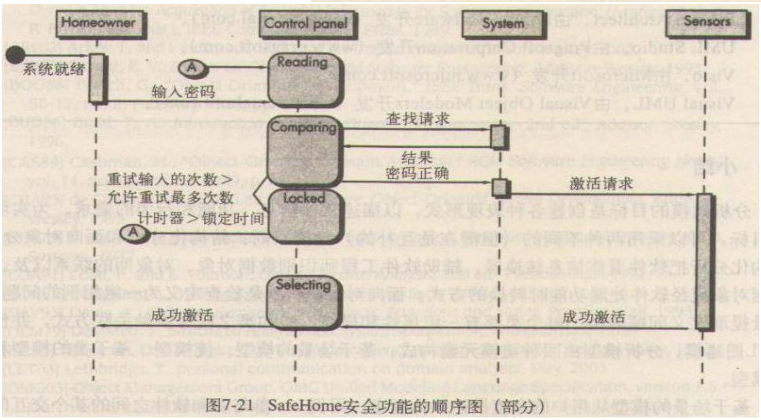


图7-21 SafeHome安全功能的顺序图（部分）

5. 数据库需求分析的核心任务

- 确定所有待保存的“数据对象”，清楚这些数据对象之间存在的关联
- 给出每个数据对象和数据关联的数据字典文件

6. 数据库需求分析的其他任务（保证数据库真正可用的关键）

数据约束，数据访问频率，数据访问安全，数据可靠性，数据响应效率

第三章：E-R 模型

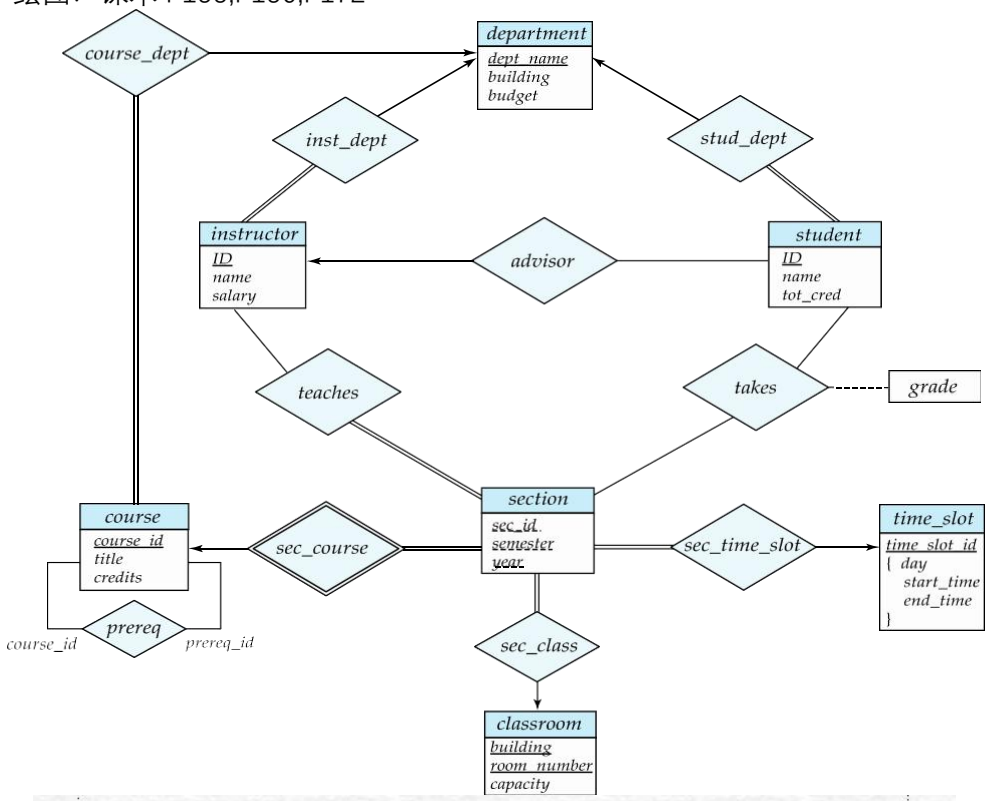
1. E-R 模型与 E-R 图

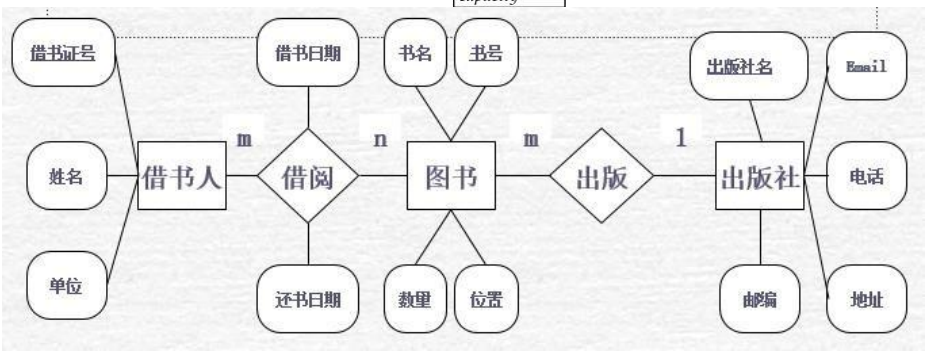
E-R 模型是一种描述方法；

E-R 图采用 E-R 模型方法，对一具体应用的描述(结果)。

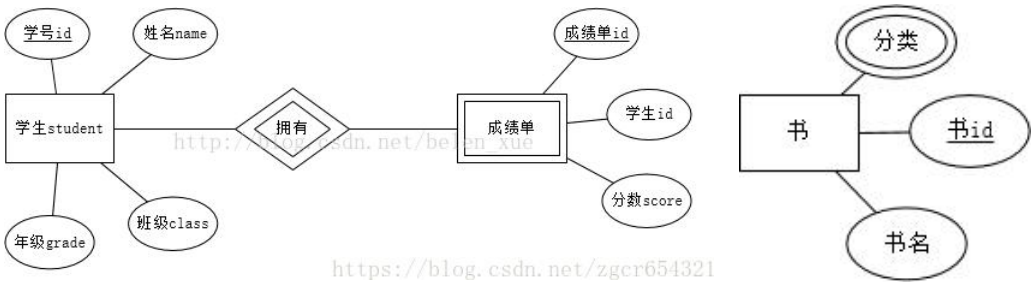
2. E-R 模型的三个基本的要素：

- **实体集**（矩形）：相关类型实体(对象)的集合，描述数据对象及特征(内部结构)；
- **联系集**(菱形、连线)：相关类型联系(连线)的集合，描述数据对像间联系(外部结构)。
- **属性**（第一种用**矩形**，第二种用**椭圆**）
- 绘图：课本 P155,P156,P172





- 对于第二种画法，弱实体用**双线矩阵**表示；与弱实体相关的联系，用**双线菱形**表示；多值属性（一个实体的某个属性可以有多个不同的取值） 用**双线椭圆**表示。



3.E-R 模型可描述任何复杂客观对象，实体集可以是任何一种复杂数据结构。

因为：E-R 模型重点是**面向客观世界**，建立**易于用户理解**的抽象数据模型（它不关心数据如何才能够被实际存储）

4.强实体集与弱实体集

- 弱实体集：实体集的所有属性都不足以形成主码
- 强实体集：实体集的属性可以形成主码
- 强实体与弱实体的联系只能是 1：1 或 1：N。
- 绘制：P158
- 弱实体集特点
  1) 没有键;
  2)存在依赖于主实体集;
  3)键由主实体集的键和它的分辨符合并构成。

5.特化与概化：P166,P167

- 特化分类及表示:
  - **重叠特化（overlapping specialization）：**一个实体集可能属于多个特化实体集，分开使用两个箭头。
  - **不相交特化（disjoint specialization）：**一个实体集必须属于至多一个特化实体集，使用一个箭头。
- 概化分类及表示:
  - **部分概化：允许父实体不属于任何子实体集 (缺省表示)**
  - **全部概化：每个父实体必属于某一子实体集 (采用标识total)**

6.聚 集 ： P170

第四章：关系模型

1. 数据模型

⑴ 定义：是一个描述数据、数据联系、数据语义以及数据一致性约束的概念工具的集合

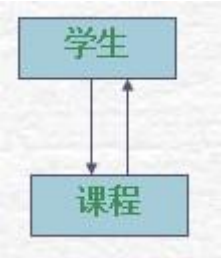
⑵ 包括：

- **数据结构：**
  由一组创建数据库的规则（定义数据库的结构）组成
- **数据操作：**
  定义对数据进行的操作类型（包括更新和查找数据库中的数据以及修改数据库的结构）
- **约束条件：**
  一组数据完整性定义规则，确保数据的正确性。

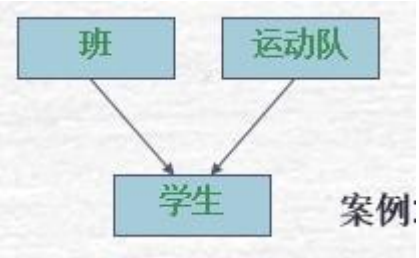
2. 层次（数据）模型

⑴ **层次模型**：利用“记录”(包含多个“属性”)和“双亲子女关系(PCR)”来描述应用的数据结构。

⑵ **层次模式**：利用层次模型描述一个应用的数据结构，称为一个层次模式（型：数据库的结构），**为“树”结构（1:N 且无多双亲联系）。**



M:N 联系



多双亲联系

⑶ 解决方法：

- 采用副本——缺点：数据冗余（增加空间，一致性维护难）
- 虚拟记录（优化方法）——缺点：指针操作增加开销

⑷ 物理存储结构：邻接法

按照层次树前序穿越的顺序，把所有记录值(子段定长，定长记录)依次邻接存放，即通过物理空间的位置相邻来实现层次顺序。可使用**中序遍历**。

⑸ 优点：

- 数据模型比较简单，操作简单。
- 对于实体间联系是固定的，且预先定义好的应用系统，性能较高。
- 提供良好的完整性支持。

⑹ 缺点：

- 不适合于表示非层次性的联系。
- 对插入和删除操作的限制比较多。
- 查询子女结点必须通过双亲结点。

3. 网状（数据）模型

⑴ 表示：

- 记录的表示(内部结构)：
  记录，数据项（允许为**多值和复合**数据）

联系的表示(外部结构):
系（set），包括单属系和多属系:
单属系：首记录和属记录
多属系：属记录值可以是不同记录类型

(2) 特点（与层次结构比）：

- 去掉了层次模型的两个限制：允许结点没有或有多个双亲结点;
- 允许两个结点之间有多种联系（复合联系）

(3) 网状模式：

利用网状模型（记录&系）来描述一个应用，可以得到一个网状模式（数据库的结构），**是一个"图"**。

(4) 物理存储结构：连接法

如：单向链接，双向链接，环状链接，向首链接

(5) 优点：

- 能够更为直接地描述现实世界。
- 具有良好的性能，存取效率较高。

(6) 缺点：

- 其 DDL 语言极其复杂。
- 数据独立性较差。由于实体间的联系本质上通过存取路径指示的，因此应用程序在访问数据时要指定存取路径。

4. 关系模型

(1) 描述：

一个数据对象的(内部)结构、数据对象间的关联关系均通过**关系**(table)描述：关系名加上一组属性。

(2) 模式图：

数据模型：描述数据对象的"内部结构"和"相互关联"
线条：外键-参照关系，描述对象间的关联（"外部结构"）
矩形：关系模式，描述对象的特征（内部结构），上方-关系名称，下划线 (组合)-主键

(3) 完整性约束：

- 定义：对关系的某种约束条件。
- 目的：用于保证关系数据库中数据的正确性和可靠性。
- 类型:
  - **实体完整性规则**
  - **参照完整性规则（引用完整性规则）**
  - **域完整性规则（用户自定义完整性规则）**
- **实体完整性规则**
  - 定义：在任何关系的任何一个元组中，主键的值不能为空值、也不能取重复的值。
  - 目的：用于保证数据库表中的每一个元组都是惟一的。
  - 主码、超码、候选码：P25
- **参照完整性规则**
  - 定义："不引用不存在的实体"。即：不允许在一个关系中引用另一个关系中不存在的元组。
  - 目的：用于确保相关联的表间的数据保持一致。
  - 外码作用：说明元组间联系，保证数据有效性。

- **域完整性规则**
  - 定义：由用户根据实际情况，定义表中属性的取值范围。
  - 目的：用于保证给定字段中数据的有效性,即保证数据的取值在有效的范围内。

(4) 关系代数：过程化的查询语言，P28，均未增强查询操作能力；p124关系代数讲解

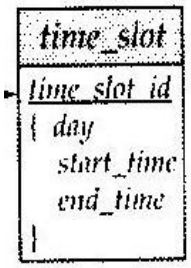
(5) 一个应用的关系模式集合应当如何得到？

1）直接与用户交互，发现数据对象，形成关系模式集合
(适合于原型开发方法，但需与用户不断交互，工作量巨大)
2）根据概念模型设计结果(E-R 图&数据字典)，形成关系模式集合
(方便且高效，避免与用户交互，适合与周期开发方法)
3）根据面向对象方法的分析结果（UML 类图），形成关系模式集合(优缺点2）)

(6) E-R 图转换为关系模式 p159

1）实体集转换

- 简单实体集的转换→新的关系模式:
  属性集与主码:就是实体集的属性集与主码
- 弱实体集的转换→新的关系模式:
  1） 属性集:弱实体集的属性集+主实体集的主码
  2） 主码:主实体集的主码+弱实体集的分辩符
- 复杂实体集的转换→新的关系模式(1NF):
  1） 属性集:包含实体集的所有简单属性，此外其复杂属性（复合属性）需逐层展开变换为多个简单属性（**纵向展开+横向展开**）
    a) 纵向展开(多值属性→多个元组)
    b) 横向展开(结构属性→多个属性)



转换为

time_slot_id	day	start_hr	start_min	end_hr	end_min
A	M	8	0	8	50
A	W	8	0	8	50
A	F	8	0	8	50
B	M	9	0	9	50
B	W	9	0	9	50
B	F	9	0	9	50
C	M	11	0	11	50
C	W	11	0	11	50

2） 主码:需根据唯一确定元组特征来确定

2）联系集转换（一二元关系的转换规则）

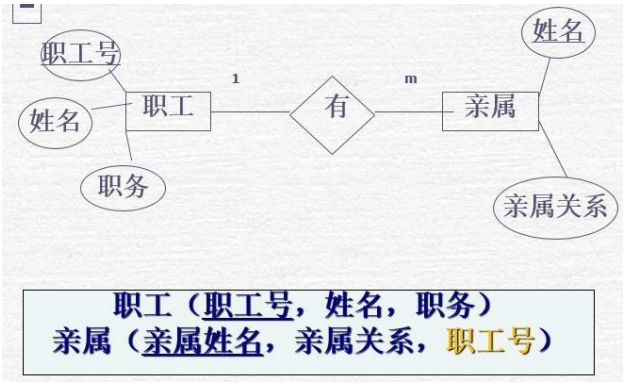
- 总体：**每个实体建一张表，属性转为表属性，关键属性为主键**
- 一对一联系→新的关系模式:
  属性集:两方主码+联系的属性;
  主码:任一方的主码均可。
  任意一方主键出现在另一方中，成外键
- 多对一联系→新的关系模式:
  属性集:两方主码+联系的属性;
  主码:多方主码。
  1 方主键出现在M 方成为外键

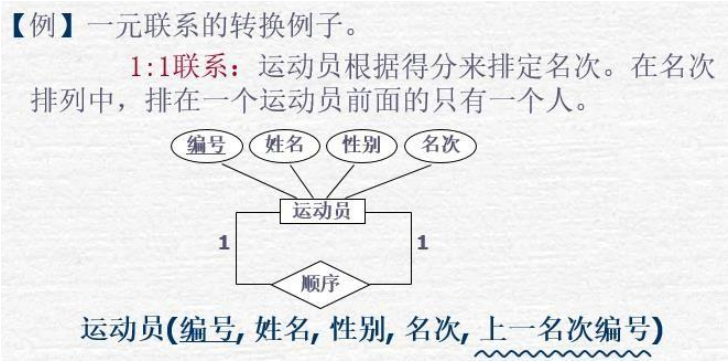
- 多对多联系→新的关系模式:
  属性集:两方主码+联系的属性;
  主码:两方主码。
  联系建为一新表，其主键由两个父实体的主键复合组成
- 弱实体联系→冗余模式:
  直接忽略不转换。

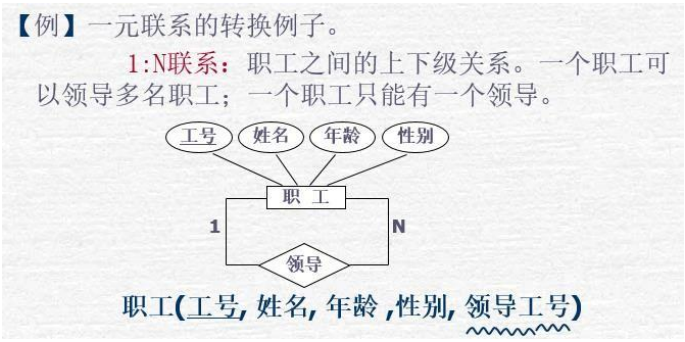
3）三元联系的转换规则

不管三元联系是哪种类型（1:1:1，1:1:n，1:m:n 和 m:n:p），**总是将三元联系也转换成关系模式**，其属性为三端实体类型的键加上联系自己的属性，而联系关系模式的键为三端实体键的组合。

4）实例：

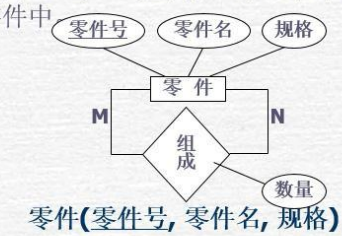






【例】一元联系的转换例子。

M:N联系：零件之间的组成关系。一个零件可以由多个零件组成；一个零件（作为子零件）也可以包括在多个零件中



零件(零件号, 零件名, 规格)

组成(零件号, 子零件号, 数量)

【例】三元联系的转换例子。

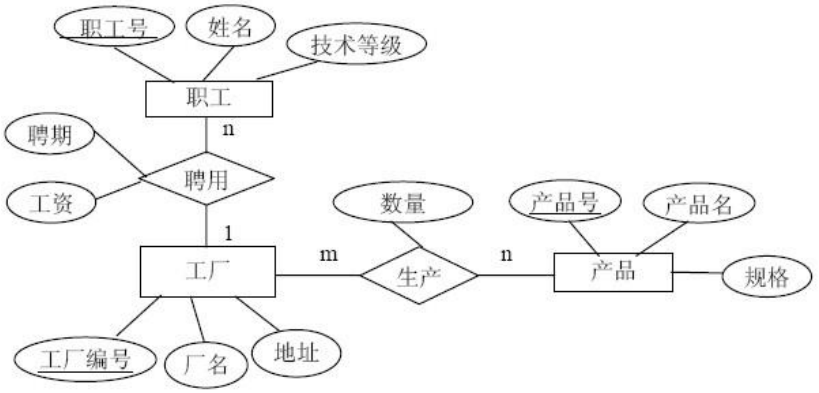
项目—零件—供应商之间的M:N:P联系。

项目(项目编号，项目名称，开工日期)

零件(零件号，零件名，规格)

供应商(供应商编号，供应商名称，供应商地址)

供应(项目编号, 零件号, 供应商编号, 数量)



职工（职工号，姓名，技术等级，工厂编号，聘期，工资）

工厂（工厂编号，厂名，地址）

产品（产品号，产品名，规格）

生产（工厂编号，产品号，数量）

(7) 关系模式的合并：

- **合并主码相同的关系模式，因这些属性均由相同主码唯一确定**
- 一对多及一对一联系都不需要单独增加关系模式，只需要在多方实体集中

添加一方主码和联系的属性即可

- 参照完整性约束涉及所有**因描述联系增加的主码属性**a)因多对一联系而增进的一方主码属性

  b)多对多联系得到模式的两方主码属性
- 如何区分全参与(描述双线)与部分参与特征? 全参与: not NULL-非空约束

第五章：SQL 语法

1. create database 做了什么?

- 产生了一个数据库（空仓库，仅包括系统数据字典）
- 初始库小，数据增长需要时才增大库空间
- 产生了一个日志存放的空仓库（备份回复用）
- 还涉及到物理设计工作：库放在何位置、库大小、库增量而且日志仓库位置可用与数据仓库位置不同(保证安全)

  实例:

  - CREATE DATABASE 数据库名
  - ON
  - (NAME = 逻辑文件名,
  - FILENAME= 'mdf 数据文件路径',
  - SIZE = 10 MB, /*数据文件初始大小*/
  - MAXSIZE = 20 MB, /*数据文件最大值*/
  - FILEGROWTH =2 MB), /*数据文件增长值*/
  - LOG ON /*创建日志文件，可省略*/
  - (NAME = 日志文件名,
  - FILENAME= '日志文件名',
  - SIZE = 10 MB,
  - MAXSIZE = 20MB,
  - FILEGROWTH =10%)
  - GO

2. SQL 查询能力很强：

1) 实现了基本代数运算
2) 灵活的表间连接方式
3) 实现了代数运算复合 (下面的嵌套子查询)
4) 灵活的 where 条件
5) 聚集函数等常用函数
6) 嵌入式和动态 SQL

3. 自然连接 Natural join 与笛卡尔积X 两点最大不同:

1) 仅包含符号连接条件的元组
2) 连接属性仅出现一次

4. 视 图 ： P68-70

1) 可以在任何 QL 语句中像表一样的被使用
2) 增强查询能力且方便(用户/程序员)使用
3) 可以提供数据访问安全控制(隐藏数据)
4) 作为外模式(1 级映射)有利于应用独立性

5. 完整性约束与断言:P72-76

- 完整性约束包括：主键约束（primary）、参照约束（foreign）、唯一约束（unique）、检查约束（check）、默认约束（default）、非空约束（not null），其中

前 4 种按作用范围分为列级约束和表级约束，区别在作用在一列还是多列；后 2 种只有列级约束。

- 参照约束：能维护插入或更新外键值以及删除或更新主键值时的数据完整性， 被参照的表必须存在，在被参照的列上已定义了唯一索引，且或是主键约束或唯一约束。外键可以为空、可以不唯一。
- 唯一约束：列上的值（包括空值）唯一，即最多只有一个空值。
- 检查约束：列上的值非空。

6. 完整性规则五元组表示: (D，O，A，C，P)

- D (Data) 约束作用的数据对象;
- O (Operation) 触发完整性检查的数据库操作
  当用户发出什么操作请求时需要检查该完整性规则，是立即检查还是延迟检查;
- A (Assertion) 数据对象必须满足的断言或语义约束这是规则的主体;
- C (Condition) 选择 A 作用的数据对象值的谓词;
- P (Procedure) 违反完整性规则时触发的过程。

7. 被参照关系删除元组的违约处理策略

(1) 级联删除（CASCADE）

- 将参照关系中外码值与被参照关系中要删除元组主码值相对应的元组**一起删除**。

(2) 受限删除（RESTRICT）

- 当参照关系中**没有**任何元组的外码值与要删除的被参照关系的元组的主码值相对应时，系统才执行删除操作，否则**拒绝此删除操作**。

(3) 置空删除（SET NULL）

- 删除被参照关系的元组，并将参照关系中与被参照关系中被删除元组主码值相等的外码值**置为空值**。

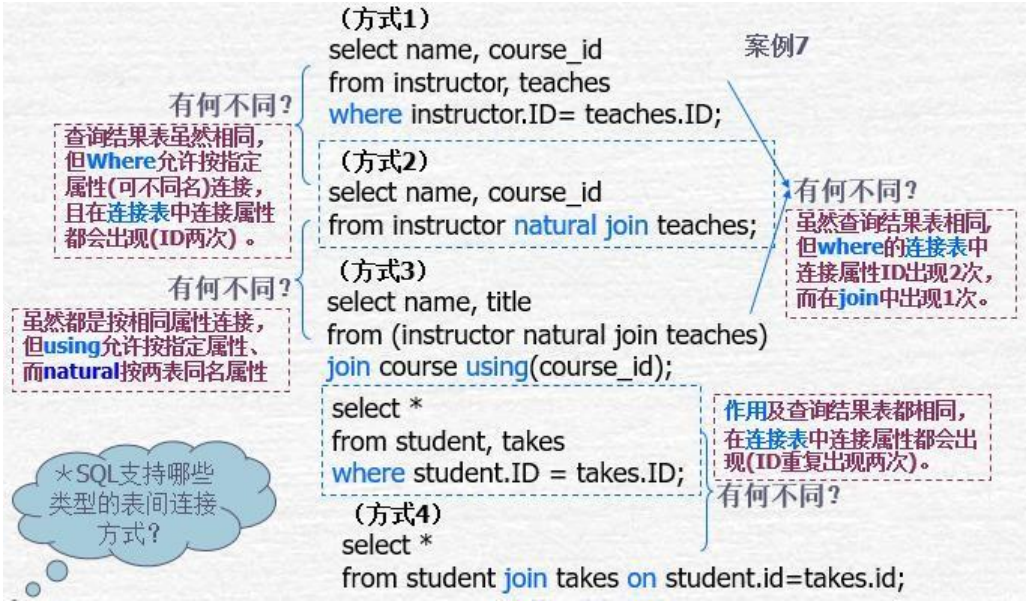
8. 连接相关：P39-40,P63-67

- 内连接 ((natural) join) : 不保留未匹配元组的连接
- 外连接分三种:
  - 左外连接 (left (outer) join) : 只保留运算符左边关系中的元组
  - 右外连接 (right (outer) join) : 只保留运算符右边关系中的元组
  - 全外连接 (full (outer) join) : 保留运算符两侧关系中的元组
- **Natural join, join on 和 where 的区别与联系（P63）：**
  - **join on 和 where 等价，生成结果相同，连接表重复的属性（可用 select *查看）会出现两次；**
  - **Natural join 连接表重复的属性只出现一次；**
  - **当 select 后选取某些特定属性（非全部属性）时，三者返回的结构相同。**
  - **三者示例：**

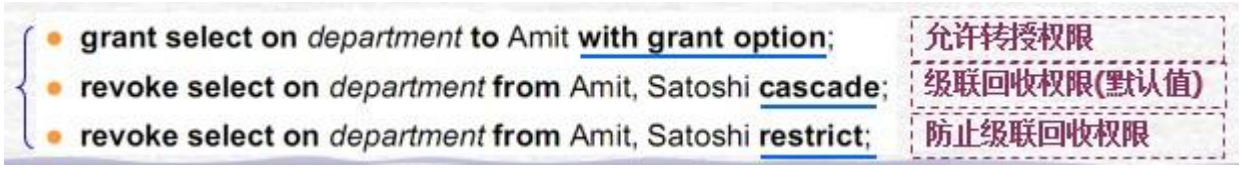
    SELECT * FROM student NATURAL JOIN course;

    SELECT * FROM student JOIN course ON student.DEPT=course.DEPT;

    SELECT * FROM student,course WHERE student.DEPT=course.DEPT;



9. 授权与角色：P81-83



- 表的创建者，拥有表上的一切权限
- 视图的创建者，拥有该视图上的所有权限，**但执行操作时需要拥有对应原表的对应权限（P83）**
- 函数与过程的创建者，拥有其上所有权限

10. 权限授予图：P84

- 作用：
  (4) 描述在一张表上某种授权的当前状态，便于系统动态管理授权；
  (5) 当 DBA 或具有权限的用户(树上节点)进行授权时，树扩展(生长)；
  (6) 当 DBA 或具有权限的用户(树上节点)回收权限时，树收缩(枯萎)；

11. SQL 函数与过程：P98

- **过程：一段SQL 语句程序；函数：有处理还有返回值**
- 存储过程可在 SQL 过程中或嵌入式 SQL 中通过 call 命令调用（**过程用 call 关键字调用**），还可在动态 SQL 中用。
- **外部语言过程：**SQL 允许用程序语言(Java, C#, C, C++)来定义函数或过程，运行效率要**高于 SQL 中定义的函数**，用于完成无法在 SQL 中执行的计算。
- 过程示例：

```
create procedure dept_count_proc (
  in dept_name varchar(20), out d_count integer)
  begin
        select count(*) into d_count
        from instructor
        where instructor.dept_name =
                              dept_count_proc.dept_name
    end
```

- **declare** d_count **integer**;
- **call** dept_count_proc('Physics', d_count);

12. 触发器：P102 示例：

```
DROP TRIGGER IF EXISTS cid_update;
CREATE TRIGGER cid_update AFTER UPDATE ON sc
FOR EACH ROW
BEGIN
IF new.CID NOT IN(
        SELECT CID
        FROM course
    ) THEN
    DELETE FROM sc
    WHERE sc.CID = new.CID;
END IF;
END;
```

13. 域与类型的区别：

- 1）域上允许声明(定义)结束;
- 2）域不是强制类型，一个域类型可以被赋予另一个域类型，只有它们的基本类型是相容的（可强制转换）。

14. 常用 SQL 命令：

(1) **查看表名：select table_name from information_schema.tables where table_schema='当前数据库';**

(2) **查看数据库：show databases;**

(3) **进入某数据库：use 数据库名;**

(4) **查看所有用户：SELECT DISTINCT CONCAT('User: ''',user,'''@''',host,''';') AS query FROM mysql.user;**

(5) **查看当前登录用户：SELECT user();**

(6) **创建用户：create user '新用户名'@' ' identified by '新用户密码';**

(7) **赋予其他用户权限：**

① 赋予单个数据库的几条功能权限：grant 权限 1，权限 2 on 表名.数据库名 to '用户名'@'localhost' with grant option;（最后是给他赋予其他用户已拥有的权限）

如：grant select on stu.* to 'test2'@'localhost' with grant option;

② 赋予所有数据库所有功能权限：grant all privileges on *.* to 'root'@'%' with grant option;

(8) **删除赋予其他用户的权限：将（7）中的'grant'改为'revoke'，'to'改为'from'**

(9) **查询当前用户权限：show grants for '用户名'@'localhost';**

(10) **刷新用户权限数值（一般在对其他用户权限修改后执行）：flush privileges;**

(11) **删除用户：drop user '用户名'@'localhost';**

第六章：逻辑模型（优化，范式与函数依赖）

1. 基础概念

- **关系：**描述实体、属性、实体间的联系。从形式上看，它是一张二维表，是所涉及属性的笛卡尔积的一个子集。
- **关系模式：**用来定义关系，相当于关系的表头。关系模式由五部分组成，即它是一个五元组：**R(U, D, DOM, F)**。也可简化为一个三元组：**R（U, F）**，当且仅当 U 上的一个关系 r 满足 F 时，r 称为关系模式 R（U, F）的一个关系。
    - R: 关系名
    - U: 组成该关系的属性名集合
    - D: 属性组 U 中属性所来自的域
    - DOM: 属性向域的映象集合
    - F: 属性间数据的依赖关系集合
- **关系数据库：**基于关系模型的数据库，利用关系来描述现实世界。从形式上看，它由一组关系组成。
- **关系数据库的模式：**定义这组关系的关系模式的全体。

2. 数据依赖

- **定义：**是通过一个关系中属性间值的相等与否体现出来的数据间的相互关系， 是现实世界属性间相互联系的抽象，是数据内在的性质，是**语义**的体现。
- **类型：函数依赖（Functional Dependency，简记为 FD），多值依赖**
- **（Multivalued Dependency，简记为 MVD）**

3. 冗余

- 定义：当数据的某些部分能由其他部分推导出来，就意味着存在冗余。
- 原因：冗余的存在是因为存在完整性约束（特别是函数依赖）。**如果没有函数依赖，则没有冗余。（非主属性）**
- 解决：通过分解关系模式来消除其中不合适的数据依赖。

4. 规范化理论：

用来改造关系模式，通过分解关系模式来消除其中不合适的数据依赖，以解决插入异常、删除异常、更新异常和数据冗余问题。

5. 函数依赖：P185

- 函数依赖不是指关系模式 R 的某个或某些关系实例满足的约束条件，而是指 R 的**所有关系实例**均要满足的约束条件。
- 函数依赖是**语义**范畴的概念。只能根据数据的语义来确定函数依赖。例如“姓名→年龄”这个函数依赖只有在不允许有同名人的条件下成立。
- 数据库设计者可以对现实世界作强制的规定。例如规定不允许同名人出现，函数依赖“姓名→年龄”成立。所插入的元组必须满足规定的函数依赖，若发现有同名人存在， 则拒绝装入该元组。
- **传递函数依赖：**在关系模式 R(U)中，如果 X→Y，Y→Z，且不满足（Y ⊆X，Z ⊆X，Z ⊆Y，Y→X）中任一条件，则称 Z 传递函数依赖于 X。

6. 范式

- 定义：范式是符合某一种级别的关系模式的集合。关系数据库中的关系必须满足一定的要求。满足不同程度要求的为不同范式。

- 种类：$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$
- 表示：某一关系模式 R 为第 n 范式，可简记为R∈nNF。

7. 1NF

- 如果一个关系模式 R 的所有属性都是不可分的基本数据项，则 R∈1NF。
- 第一范式是对关系模式的最起码的要求。不满足第一范式的数据库模式不能称为关系数据库。

8. 2NF

- 若关系模式 R∈1NF，并且**每一个非主属性都完全函数依赖于 R 的主码而不依赖于主码的部分属性**，则 R∈2NF。
- 将一个 1NF 关系通过投影分解法分解为多个 2NF 的关系，**并不能完全消除关系模式中的各种异常情况和数据冗余**。

9. 3NF

- 关系模式R<U，F> 中不存在非主属性对于码的**传递函数依赖**（如 X→Y，Y→Z，X 是主键），则称 R<U，F> ∈ 3NF。
- 关系 R，函数依赖集 F 中每一个函数依赖X→A (X、A ⊆ R), 符合下列描述中的一个:
  - – A ⊆ X (平凡的 FD), or
  - – X 是超键, or
  - – A 是 R 的候选键的一部分
- 将一个 2NF 关系通过投影分解法分解为多个 3NF 的关系，**并不能完全消除关系模式中的各种异常情况和数据冗余。**
- **无损分解，保持函数依赖。**

10. BCNF

- 对于 R 有函数依赖集 FDs F，如果 R 符合 BCNF 当且仅当每一个非平凡的FD X → A in F , X 是 R 的超键。也就是说, **R 符合 BCNF 当且仅当非平凡的函数依赖箭头左侧是超键。如果 R 只有两个属性, 那么它符合BCNF。**
- 关系 R，函数依赖集 F 中每一个函数依赖X→A (X、A⊆ R), 符合下列描述中的一个:
  - – A ⊆ X (平凡的 FD), or
  - – X 是 超 键
- **BCNF 一般不会出现冗余, 无损分解，可能不保持函数依赖。**
- 

- **例题 1**：R(A,B,F), F = {B → F}：不是 BCNF 也不是 3NF
  原因：候选键是AB，B 不是超键所以不是 BCNF；F 不是候选键的一部分所以不是 3NF。
- **例题 2**：R(B,C), F = {A→D,D→A}：是 BCNF 也是 3NF 原因：候选键是 BC，A 和 D 间的函数依赖是平凡的。

11. 规范化

- 定义：一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级范式的关系模式集合，这种过程就叫关系模式的规范化。

- 步骤: 1NF
  ↓ 消除非主属性对码的部分函数依赖
  2NF
  ↓ 消除非主属性对码的传递函数依赖
  3NF
  ↓ 消除主属性对码的部分和传递函数依赖
  BCNF
  ↓ 消除非平凡且非函数依赖的多值依赖
  4NF
- 基本思想:
  - 消除不合适的数据依赖
  - 各关系模式达到某种程度的"分离"
  - 采用"一事一地"的模式设计原则: 让一个关系描述一个概念、一个实体或者实体间的一种联系。若多于一个概念就把它"分离"出去
  - 所谓规范化实质上是概念的单一化
  - 不能说规范化程度越高的关系模式就越好

12. Armstrong 公理系统

(1) 逻辑蕴含：P189

对于满足一组函数依赖 F 的关系模式 R <U，F>，其任何一个关系 r，若函数依赖 X→Y 都成立, 则称F 逻辑蕴含X →Y。

(2) 用途：求给定关系模式的码；从一组函数依赖求得蕴含的函数依赖。

(3) 六大定理：P190

- **自反律（Reflexivity）：若 Y ⊆ X ⊆ U，则 X →Y 为 F 所蕴含。**
- **增广律（Augmentation）：若 X→Y 为 F 所蕴含，且 Z ⊆ U，则XZ→YZ 为 F 所蕴含。**
- **传递律（Transitivity）：若 X→Y 及 Y→Z 为 F 所蕴含，则X→Z 为 F 所蕴含。**
- **合并律（union）：由 X→Y，X→Z，有 X→YZ。（A2， A3）**
- **伪传递律（pseudotransitivity）：由 X→Y，WY→Z，有 XW→Z。（A2，A3）**
- **分解律（decomposition）：由 X→Y 及 Z⊆Y，有 X→Z。（A1， A3）**
- **导出定理：$X \rightarrow A_1 A_2 \ldots A_k$ 成立的充分必要条件是 $X \rightarrow A_i$ 成立（i=1，2，…，k）。**

(4) 定理证明：

- **自反律**

  证：设 Y ⊆ X ⊆ U
  对 R <U，F> 的任一关系 r 中的任意两个元组 t，s： 若
  t[X]=s[X]，由于 Y ⊆ X，有 t[y]=s[y]，
  所以 X→Y 成立.

- **增广律**

  证：设 X→Y 为 F 所蕴含，且 Z ⊆ U。
  设 R<U，F> 的任一关系 r 中任意的两个元组 t，s； 若
  t[XZ]=s[XZ]，则有 t[X]=s[X]和 t[Z]=s[Z]；
  由 X→Y，于是有 t[Y]=s[Y]，所以 t[YZ]=s[YZ]，所以 XZ→YZ 为 F 所蕴含.

- **传递律**

证：设 X→Y 及 Y→Z 为 F 所蕴含。
对 R<U，F> 的任一关系 r 中的任意两个元组 t，s。若
t[X]=s[X]，由于 X→Y，有 t[Y]=s[Y]；
再由 Y→Z，有 t[Z]=s[Z]，所以 X→Z 为 F 所蕴含.

13. 闭包（函数依赖集）

(1) 定义：

- 1. 在关系模式 R<U，F>中为 F 所逻辑蕴含的函数依赖的全体叫作 F 的闭包，记为 F+。
- 2. 设 F 为属性集U 上的一组函数依赖，X ⊆U，XF+={ A|X→A 能由 F 根据 Armstrong 公理导出}，XF+称为属性集 X 关于函数依赖集 F 的闭包（也可写作 X+）。

(2) 引理：

- 设 F 为属性集 U 上的一组函数依赖，X，Y ⊆ U，X→Y 能由 F 根据 Armstrong 公理导出的充分必要条件是 $Y \subseteq X_F^+$。

(3) 用途：P191

- ***1.*** 测试超键
  判断 X 是否是一个超键：只需要计算 X+, 检查 X+是否包括 R 的所有属性.
- ***2.*** 检测函数依赖
  判断 X → Y 是否成立 (或者说, 是否在 F+中), 只需要判断 Y ⊆ X+.
- ***3.*** 计算 ***F*** 的函数依赖集闭包

(4) 求闭包算法：P190-191

- 求属性集X（X ⊆ U）关于 U 上的函数依赖集 F 的闭包 X_F^+
- 输入：X，F；输出：X_F^+
- 步骤（最多执行|U|-|X|次循环）：

(1) 令 $X^{(0)}=X$，i=0
(2) 求 B，这里 $B = \{ A \mid (\exists V)(\exists W)(V \to W \in F \wedge V \subseteq X^{(i)} \wedge A \in W)\}$;
(3) $X^{(i+1)}=B \cup X^{(i)}$
(4) 判断 $X^{(i+1)} = X^{(i)}$ 吗?
(5) 若相等或 $X^{(i)}=U$, 则 $X^{(i)}$ 就是 X_{F+},算法终止。
(6) 若否，则 i=i+l，返回第（2）步。

(5) 函数依赖集等价：

- 如果 $G^+=F^+$，就说函数依赖集 F 覆盖G（F 是 G 的覆盖，或G 是 F 的覆盖），或 F 与 G 等价。
- $F^+ = G^+$ 的充分必要条件是 $F \subseteq G^+$，和 $G \subseteq F^+$。
  证: 必要性显然，只证充分性。
  (1) 若 $F \subseteq G^+$， 则 $X_F^+ \subseteq X_{G+}^+$。
  (2) 任取 $X \to Y \in F^+$ 则有 $Y \subseteq X_F^+ \subseteq X_{G+}^+$。
  所以 $X \to Y \in (G^+)^+ = G^+$。即 $F^+ \subseteq G^+$。
  (3) 同理可证 $G^+ \subseteq F^+$，所以 $F^+ = G^+$。

(6) 最小依赖集（最小覆盖）

- 定义 1：如果函数依赖集F 满足下列条件，则称 F 为一个极小函数依赖集，亦称为最小依赖集或最小覆盖。

(1) F 中任一函数依赖的右部仅含有一个属性。
(2) F 中不存在这样的函数依赖 X→A，使得 F 与 F-{X→A}等价。
(3) F 中不存在这样的函数依赖 X→A，X 有真子集Z 使得(F-{X→A}) ∪ {Z→A} 与 F 等价。

- **定义 2：每一个函数依赖集 F 均等价于一个极小函数依赖集 Fm，此 Fm 称为F 的最小依赖集。**
- 定义 2 证明:

  (1) 逐一检查 F 中各函数依赖 FDi：X→Y，若 Y=A1A2…Ak，k > 2，则用{ X→Aj | j=1, 2，…，k} 来取代 X→Y。

  (2) 逐一检查 F 中各函数依赖 FDi：X→A，令 G=F-{X→A}，若 A∈XG+， 则从 F 中去掉此函数依赖。

  由于 F 与 G =F-{X→A}等价的充要条件是A∈XG+，因此 F 变换前后是等价的。

  (3) 逐一取出 F 中各函数依赖 FDi：X→A。

  设 X=B1B2…Bm，逐一考查 Bi（i=l，2，…，m），若 A∈（X-Bi）F+，则以 X-Bi 取代 X。

  由于 F 与 F-{X→A}∪{Z→A}等价的充要条件是 A∈ZF+ ，其中 Z=X-Bi 因此F 变换前后是等价的。

  (4) 由定义，最后剩下的 F 就一定是极小依赖集。

  因为对 F 的每一次“改造”都保证了改造前后的两个函数依赖集等价，因此剩下的 F 与原来的 F 等价。
- **F 的最小依赖集 Fm 不一定是唯一的它与对各函数依赖 FDi 及 X→A 中 X 各属性的处置顺序有关。**
- **求最小依赖集方法：**

  1.将 F 中的所有依赖右边化为单一元素**（简右）**

  2.去掉 F 中的所有依赖左边的冗余属性**（简左）**

  3.去掉 F 中所有冗余依赖关系**（简关系）（二三步骤可互换）**

(7) 正则覆盖：P192

在最小覆盖的基础上，添加要求：Fc 左侧属性唯一。如 X→Y, X→Z 要合并成 X→YZ。

(8) 模式分解：

- **三种模式分解的等价定义**

  1. 分解具有无损连接性
  2. 分解要保持函数依赖
  3. 分解既要保持函数依赖，又要具有无损连接性
- **模式分解表示**
  - 关系模式R<U,F>的一个分解:

    ρ={ R_1<U_1,F_1>，R_2<U_2,F_2>，…，R_n<U_n,F_n>}，U=U_1∪U_2∪…∪U_n，且不存在 U_i⊆U_j，F_i为 F在 U_i上 的 投 影
  - 函数依赖集合**{X→Y | X→Y ∈ F^+∧XY⊆U}** 的一个覆盖 F 叫作 F 在属性 U_i 上的投影
- **可能存在的问题**

  1. 一些查询可能会代价变高.
  2. 分解后，根据分解的实例, 可能不能重新构建分解前的实例
  3. 检查某些依赖需要考虑分解后的多个关系.
- **无损分解**
  - **定义：**

    关系模式R<U,F>的一个分解 ρ={ R1<U1,F1>，R2<U2,F2>，…， Rn<Un,Fn>}若 R 与 R1、R2、…、Rn **自然连接**的结果相等，则称关系模式 R 的这个分解 ρ 具有无损连接性（Lossless join），即满足:

$$\pi_{R_1}(r) \bowtie \pi_{R_2}(r) = r$$

- **特点：**具有无损连接性的分解保证不丢失信息
- 无损连接性不一定能解决插入异常、删除异常、修改复杂、数据冗余等问题

- **分解无损的特征：**
  - 将 R 分解为 R1 何 R2 是无损分解，如果下面至少一个成立的话， 那么分解问无损分解: (仅适用于分解为两模式情形)
    - $R1 \cap R2 \rightarrow R1$ (函数依赖)
    - $R1 \cap R2 \rightarrow R2$ (函数依赖)
  - 实际上将 R 分解为(UV)和(R-V)，如果 U→V 在 R 上成立，那么分解是无损连接分解

- **函数依赖**
  - **定义：**
    - 设关系模式 $R<U,F>$ 被分解为若干个关系模式 $R_1<U_1,F_1>$，$R_2<U_2,F_2>$，…，$R_n<U_n,F_n>$（其中 $U=U_1 \cup U_2 \cup \cdots \cup U_n$，且不存在 $U_i \subseteq U_j$，F_i 为 F 在 U_i 上的投影），**若 F 所逻辑蕴含的函数依赖一定也由分解得到的某个关系模式中的函数依赖 F_i 所逻辑蕴含**，则称关系模式R 的这个分解是保持函数依赖的（Preserve dependency）。
  - **特点：**
    - **如果一个分解具有无损连接性，则它能够保证不丢失信息。**
    - **如果一个分解保持了函数依赖，则它可以减轻或解决各种异常情况。**
    - **分解具有无损连接性和分解保持函数依赖是两个互相独立的标准。具有无损连接性的分解不一定能够保持函数依赖，保持函数依赖的分解也不一定具有无损连接性。**
  - **分解保持函数依赖的特征：**

    将R 分解为X 和Y 是保持函数依赖的，当且仅当 $(F_X \cup F_Y)^+ = F^+$

- **转换为 3NF 的分解算法：**

  1：（合成法）转换为 3NF 的保持函数依赖的分解。

  1. 对 R<U,F>中的函数依赖集 F 进行极小化处理
  2. 找出不在 F 中出现的属性，把这样的属性构成一个关系模式
  3. 若有 $X \rightarrow A \in \mathbf{F}$，且 **XA=U**，则 ρ={R}，算法终止
  4. 否则，对 F 按具有**相同左部**的原则分组，得到ρ

  2：转换为 3NF 既有无损连接性又保持函数依赖的分解

  1. 检查第一大步的ρ是否是有损分解（通过画表法或者**直接观察ρ中是否有一个分解包含任意一组候选码**），若为无损分解直接输出
  2. 若为有损分解，则ρ加上由一组候选码组成的关系（函数依赖为空），也可直接在无关属性的分解中添加。

14. 例题：

设计关系 r(R)模式为：R=(职工名,项目名,工资,部门号,部门经理) F:{项目名　部门号，部门号　部门经理，职工名,项目名　工资}

1. 求 R 的候选键

2. 指出使 R 违反 3NF 的所有函数依赖

3.将 R 保持函数依赖且无损分解为一组 3NF

解 1：候选键是(职工名,项目名)。因(职工名,项目名)$^{+}$=R

解 2：

因：部门号→部门经理：存在非主属性对键的部分函数依赖(必是对键的传递依赖), 项目名→部门号，部门号→部门经理：存在非主属性对键的传递函数依赖,

故：R 不是 3NF，甚至不是 2NF。

解 3：因 F 不存在无关属性，已是正则覆盖。

故直接将R 保持依赖且无损分解为如下一组 3NF:

R1={项目名 , 部门号} , R2={部门号,部门经理} , R3={职工名,项目名,工资} , F1={项目名→部门号} ， F2={部门号→部门经理}，F3={职工名,项目名→工资} **注：因键已包含在 R3 中，故不需单独做成一个新关系模式。**

第七章：物理设计——数据库存储技术

1. 文件组织：P255-256

2. 文件中记录的组织方式：P259

(1) **堆文件**：记录在文件空间中任意放置

(2) **顺序文件**：按一定的顺序在文件中组织记录

- 顺序文件的逻辑组织方式：
  1) 将关系中记录按"某属性/组-搜索码"排列
  2) 用**指针**将记录依序连接
  特点：按搜索码搜索的效率高
- 顺序文件的物理组织方式：
  1) 将关系中记录按搜索码次序进行物理存储
  2) 采用定长记录或变长记录方式
  3) 一个记录的信息不能分存在两个物理块中

(3) **散列文件**：按照散列函数计算值存放相应记录

(4) **多表簇集文件**：不同关系表里的记录存放在同一个文件中，指将多个关系的数据组织在一个文件中（它们的记录相互交织在一起），需底层操作系统配合，实现对文件的管理(只有大型数据库系统才支持)。**优缺点：P261**

3. 数据库物理设计（应用开发中）的主要工作：

(1) 在定义关系模式时，需要确定采用定长还是变长记录;
(通过确定采用的属性类型，因有变长属性)

(2) 对每个关系模式，需要确定影响记录存放次序的搜索码;
(根据常用/重要的查询要求，确定主码或建聚集索引 Cluster Index)

(3) 对每个关系模式，需要确定影是否还需要建立辅助索引文件;
(根据常用/重要的查询要求，确定建哪些索引Index)

(4) 对具有连接条件的一组关系模式，需要确定是否采用多表聚集文件存储;
(根据多表连接上重要应用查询快速访问需要)

(5) 对应用的所有关系模式，需要确定应当划分为多少个数据库来存储;
(根据关系模式间相关性、应用相关性、数据保密需要、数据库备份需要等)

(6) 对每个数据库，需要确定数据库文件存放的物理路径（不同服务器，不同硬盘，不同介质）
(根据访问效率需要、应用相关性、数据重要性等)

4. 数据字典：P261-262

作用：是 RDBMS 和 DBA 管理数据库的基础手段

特点：关系数据库管理系统-RDBMS，采用自描述方式；数据和元数据存储采用关系模

5.数据库更新操作：先删除再添加

6.数据库缓冲区

(1) 定义及作用：P262

1)CPU 处理信息快捷，但从磁盘读取记录缓慢.

2)缓冲区一次 I/O 读多硬盘上多个记录(按块)，可明显减少磁盘 I/O 开销

3)缓冲区中的记录，可能为多个应用所需要，可明显减少磁盘 I/O 开销注：连续读-节省时间；重复读-浪费时间。

(2) 缓冲区管理器：P262

缓冲区要被移除的块清除前先将块中最新修改数据回写到硬盘（数据库）

(3) 缓冲区替换策略：P263

第八章：物理设计——索引与散列

1. 索引、搜索码、索引项：P268-269

(数据库)索引是一种与(数据库)文件相关联的附加结构，额外增加的一个辅助文件在满足条件的记录数较少(这种应用占多数情况)时，索引的效果才明显。

2. 顺序索引

(1) 概念：P269

- 顺序索引：基于搜索码值的顺序排序(指索引项在索引文件中)
- 主索引：索引文件排序与数据文件排序相同（只能有一个）
- 辅助索引：索引文件排序与数据文件排序不相同（可多个）

(2) 为何能加快查找效率?

1)索引小，可在内存中处理

2)索引排了序，查找效率高

3)避免逐个读全部记录文件

(3) 稠密索引和稀疏索引：P269-270

(4) 多级索引：P271

(5) 辅助索引：P272-273

3. B+树索引文件

(1) 结构：P274-275

(2) 特点：

- 包含 n 个(应用需要设定)指针和n-1 个搜索码，搜索码依序排列。
- 指针 Pi (内部节点)指向一棵子树，或者(叶节点)直接指向记录。

(3) B+树查询：P275

- B+树索引特点:查询效率极高
- N 记录/搜索码总数,树高 $\log_{\lceil n/2 \rceil} N$ 故查找效率(读块次数):对数时间

(4) B+树插入更新：P278

- B+树插入特点：开销小(仅极小部分有变化)
- 节点未满时，只需直接将搜索码填写到相应节点;
- 原节点已满，需'一分为二'(各放一半搜索码);
- 上层节点未存两新节点指针，需增加一个索引码：按照搜索码与子树上搜索码的关系,为右子树上搜索码中的最小者。

(5) B+树删除更新：P280

- B+树删除特点:开销小(仅极小部分有变化)
- 节点半满时，只需直接将搜索码从相应节点删除;
- 节点太空时，将搜索码删除后还需调整相应节点。

4. （静态）散列索引

（1） 桶和散列函数：P288-289

（2） 溢出桶：P290

- 可能多个,尤其不均匀时
- 当某桶装满时，存储溢出索引项

（3）插入或删除一个索引项：

计算搜索码的散列值确定桶，然后在相应桶中写入或删除索引项

（4）查找记录：

计算搜索码的散列值确定桶，然后在相应桶中得到索引项，根据索引项中指针得到记录

（5）散列索引：

采用散列函数将搜索码映射到散列桶，通过散列索引，支持基于搜索码的记录快速查找。特别适合按搜索码的等值查找。

5. 动态散列：P292-293

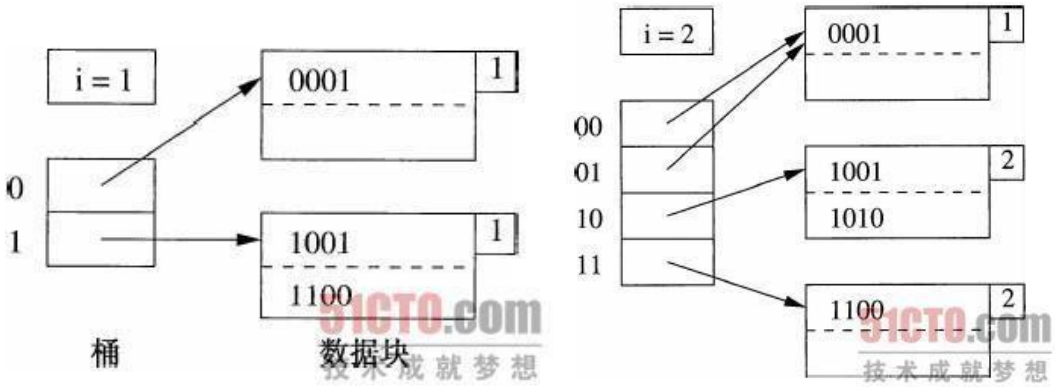
(1) 特点：

散列桶数目不固定,通过散列前缀 i 控制,开始非常少,随索引项增加而逐渐增大。

(2) 插入数据：P293，

例题如下:

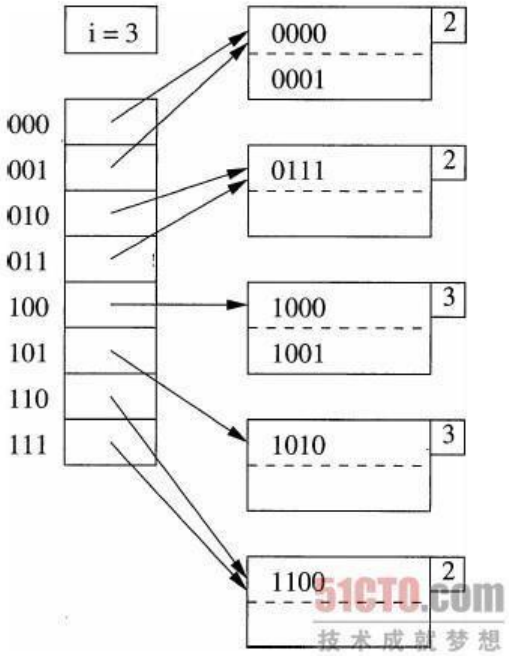
散列函数 h 为每个键计算出一个 4 位二进制序列。当前使用的只有其中一位，正如桶数组上方的框中 i=1 所标明的那样。因此，桶数组只有两个项，一个对应 0，一个对应 1。



我们在前面图表中插入一个键值散列为 1010 序列的记录。因为第一位是 1，所以该记录属于第二个块。然而，该块已满，因此需要分裂。这时我们发现j=i=1，因此我们首先需要将桶数组加倍，如图所示。上右图中我们已将 i 设为 2。

现在，假定我们来插入键值分别列为 0000、0111 和 1000 的记录。这两个记录都属于图中第一个存储块，于是该块溢出。因为该块中只用一位来确定其成员资格。 而 i=2，所以我们就不用调整桶数组。我们只需分裂该块，让 0000 和 0001 留在该

块，而将 0111 存放到新块中，桶数组中 01 项改为指向新块。插入 1000 略。



第九章：查询处理

1. 查询处理过程：P305 图

- **语法分析器与翻译器：检查语法和关系有效性 + 转换为关系代数操作**
- **优化器：通过等价变换得到优化执行方案（操作执行次序），含注释是否需要采用索引、具体采用的操作执行算法**
- **执行计划：执行一个查询的计算原语(标注了如何执行的一个或多个关系代数操作)的操作序列**
- **执行引擎：执行优化计划，形成返回结果**

2. 查询代价度量：P306-307

3. 选择操作

(1) 实现方法

1. 简单的全表扫描方法

- 对查询的基本表顺序扫描，逐一检查每个元组是否满足选择条件，把满足条件的元组作为结果输出
- 适合小表，不适合大表

2. 索引(或散列)扫描方法

- 适合选择条件中的属性上有索引(例如 B+树索引或 Hash 索引)
- 通过索引先找到满足条件的元组主码或元组指针，再通过元组指针直接在查询的基本表中找到元组

(2) 例题

例1． Sno＝‘200215121’，并且 Sno 上有索引(或 Sno 是散列码)

- 使用索引(或散列)得到Sno 为‘200215121’ 元组的指针
- 通过元组指针在 student 表中检索到该学生例

2． Sage>20，并且 Sage 上有 B+树索引

- 使用 B+树索引找到 Sage＝20 的索引项，以此为入口点在 B+树的顺序集上得到 Sage>20 的所有元组指针
- 通过这些元组指针到 student 表中检索到所有年龄大于 20 的学生。例3． Sdept ＝‘CS’ AND Sage>20，Sdept 和 Sage 上都有索引

算法一：分别用上面两种方法分别找到 Sdept＝‘CS’的一组元组指针和Sage>20 的另一组元组指针

- 求这 2 组指针的交集
- 到 student 表中检索
- 得到计算机系年龄大于 20 的学生

算法二：找到 Sdept＝‘CS’的一组元组指针，

- 通过这些元组指针到 student 表中检索
- 对得到的元组检查另一些选择条件(如 Sage>20)是否满足
- 把满足条件的元组作为结果输出。

4. 连接操作

(1) 特点：查询处理中最耗时的操作之一。

(2) 嵌套循环方法(nested loop)

- **步骤：**
  - 对外层循环(Student)的每一个元组(s)，检索内层循环(SC)中的每一个元组

(sc)

- 检查这两个元组在连接属性(sno)上是否相等，如果满足连接条件，则串接后作为结果输出，直到外层循环表中的元组处理完为止

(3) 排序-合并方法(sort-merge join 或 merge join)

- **适合情况：连接的诸表已经排好序**
- **步骤：**
  - 如果连接的表没有排好序，先对 Student 表和 SC 表按连接属性 Sno 排序
  - 取 Student 表中第一个Sno,依次扫描 SC 表中具有相同 Sno 的元组
  - 当扫描到 Sno 不相同的第一个 SC 元组时，返回 Student 表扫描它的下一个元组，再扫描 SC 表中具有相同 Sno 的元组，把它们连接起来
  - 重复上述步骤直到 Student 表扫描完
- **特点：**
  - Student 表和 SC 表都只要扫描一遍
  - 如果 2 个表原来无序，执行时间要加上对两个表的排序时间
  - 对于 2 个大表，先排序后使用 sort-merge join 方法执行连接， 总的时间一般仍会大大减少

(4) 索引连接(index join)方法

- **步骤：**
  - ① 在 SC 表上建立属性Sno 的索引，如果原来没有该索引
  - ② 对 Student 中每一个元组，由 Sno 值通过 SC 的索引查找相应的 SC 元组
  - ③ 把这些 SC 元组和 Student 元组连接起来
  - 循环执行②③，直到 Student 表中的元组处理完为止

(5) Hash Join 方法

- **描述：把连接属性作为 hash 码，用同一个 hash 函数把R 和 S 中的元组散列到同一个 hash 文件中**
- **使用情况：两个表中较小的表在第一阶段后可以完全放入内存的 hash 桶中**
- **步骤：**
  - 划分阶段(partitioning phase):
    - 对包含较少元组的表(比如 R)进行一遍处理
    - 把它的元组按 hash 函数分散到 hash 表的桶中
  - 试探/连接阶段(probing/join phase):
    - 对另一个表(S)进行一遍处理
    - 把 S 的元组散列到适当的 hash 桶中
    - 把元组与桶中所有来自R 并与之相匹配的元组连接起来

5. 表达式计算

(1) 物化计算：P322

(2) 流水线计算：P323

第十章：查询优化

1. 查询优化概述

(1) 概述

- 关系查询优化是影响 RDBMS 性能的关键因素。
- 由于关系表达式的语义级别很高，使关系系统可以从关系表达式中分析查询语义，提供了执行查询优化的可能性。

(2) 好处

查询优化的优点不仅在于用户不必考虑如何最好地表达查询以获得较好的效率，而且在于系统可以比用户程序的“优化”做得更好：

- 优化器可以从数据字典中获取许多统计信息，而用户程序则难以获得这些信息。
- 如果数据库的物理统计信息改变了，系统可以自动对查询重新优化以选择相适应的执行计划。在非关系系统中必须重写程序，而重写程序在实际应用中往往是不太可能的。
- 优化器可以考虑数百种不同的执行计划，程序员一般只能考虑有限的几种可能性。
- 优化器中包括了很多复杂的优化技术，这些优化技术往往只有最好的程序员才能掌握。系统的自动优化相当于使得所有人都拥有这些优化技术。

(3) 代价概述

RDBMS 通过某种代价模型计算出各种查询执行策略的执行代价，然后选取代价最小的执行方案

- **集中式数据库**

  执行开销=磁盘存取块数(I/O 代价)+处理机时间(CPU 代价)+查询的内存开销 （**I/O 代价是最主要的**）

- **分布式数据库**

  总代价=I/O 代价+CPU 代价+内存代价 + 通信代价

(4) 目标

- 选择有效的策略
- 求得给定关系表达式的值
- 使得查询代价最小（实际上是较小）

(5) 步骤（P329）

1. SQL 转换为关系代数

2. 将关系代数转换成某种内部表示，通常是语法树

3. 根据一定的等价变换规则把语法树转换成标准（优化）形式。选择低层的操作算法，找到一颗最优树(执行效率最佳)

对于语法树中的每一个操作，计算各种执行算法的执行代价选择代价小的执行算法。

4. 生成查询计划(查询执行方案)

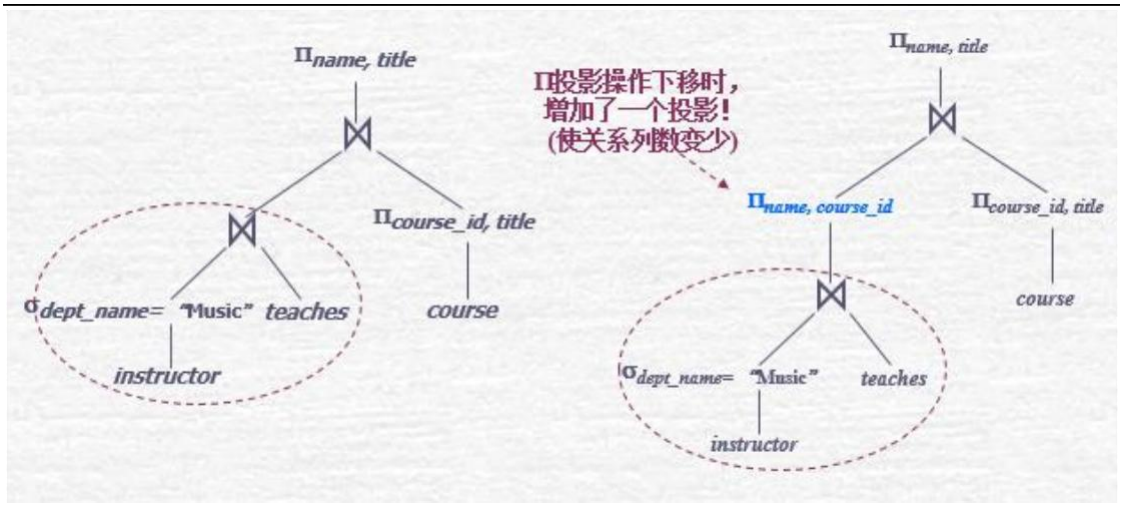
查询计划是由一系列内部操作组成的

2. 关系表达式转换（P329）

(1) 等价规则：P331-332

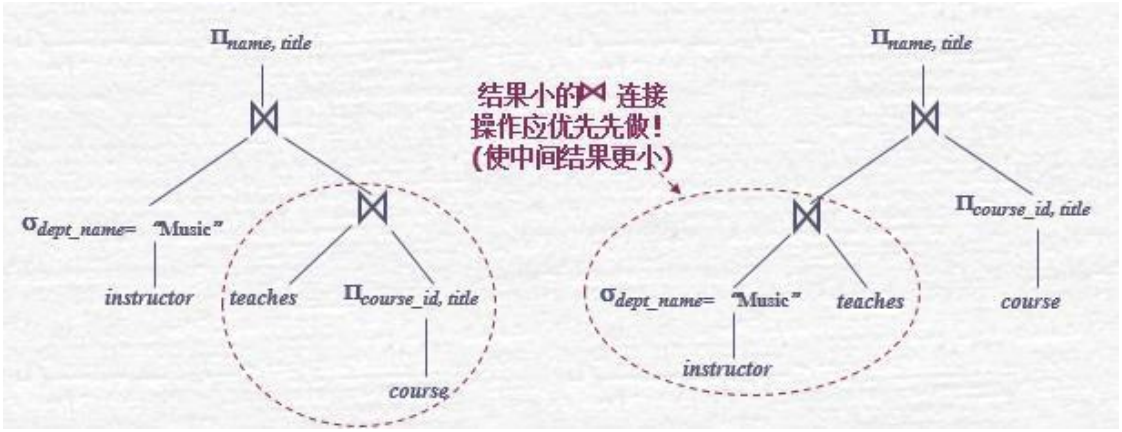
(2) “选择下移”优化策略：选择操作尽可能早做（在树的叶子）：P334 图

(3) “投影下移”优化策略：通过去除不必要的属性，可以减少中间结果的列数，从而减少中间结果的大小。



(4) "选择连接顺序"优化策略：

- **小关系的连接应优先(中间结果的元组数更少)**
- **尽可能不做笛卡尔积（X)**



3. 执行计划：P330

4. 启发式优化：P342-343

5.表达式结果集大小的估计

(1) 目录（统计）信息：P335

在关系被修改时系统自动维护。不必精确，可以是统计得到（比如直方图），采用随机抽样法

(2) 选择运算结果大小的估计：P336

(3) 连接运算结果大小的估计：P338

第十一章：数据库事务

1. **事务的概念（transaction）**
   - **（1） 定义：P356**
   - **（2） 作用：**

     事务是恢复和并发控制的基本单位，是数据库处理的最小逻辑单位，是访问并可能更新各种数据项的一个程序执行单元。DBMS 通过保证要么执行整个事务，要么一个操作也不执行，达到使数据始终处于一个一致状态。
   - **（3） 事务和程序：**

     在关系数据库中，一个事务可以是一条 SQL 语句，一组 SQL 语句或整个程序；一个应用程序通常包含多个事务。
   - **（4） 事务的定义方式：**
   - 显式方式：直接在 SQL 程序中写出 COMMIT 或 ROLLBACK
   - 隐式方式：当用户没有显式地定义事务时，DBMS 按缺省规定自动划分事务
   - **（5） 事务结束状态**
   - **COMMIT**
     - **事务正常结束**
     - **提交事务的所有操作（读+更新）**
     - **事务中所有对数据库的更新永久生效**
   - **ROLLBACK**
     - **事务异常终止**
     - **事务运行的过程中发生了故障，不能继续执行**
     - **回滚事务的所有更新操作**
     - **事务滚回到开始时的状态**
2. **事务的 ACID 特性（property）：P357-358**
   - **（1） 原子性（A）：事务是数据库的逻辑工作单位。事务中包括的诸操作要么都做，要么都不做**
   - **（2） 一致性（C）：事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态**
     - **一致性状态：数据库中只包含成功事务提交的结果**
     - **不一致状态：数据库中包含失败事务的结果**
     - **如事务A 由B 和 C 两个操作构成，则这两个操作：**
       - **全做或者全不做，数据库都处于一致性状态。**
       - **只做一个操作，数据库就处于不一致性状态。**
   - **（3） 隔离性（I）：对并发执行而言，一个事务的执行不能被其他事务干扰**
     - **一个事务内部的操作及使用的数据对其他并发事务是隔离的**
     - **并发执行的各个事务之间不能互相干扰**
   - **（4） 持久性（D）：一个事务一旦提交，它对数据库中数据的改变就应该是永久性的，接下来的其他操作或故障不应该对其执行结果有任何影响。**
   - **（5） 破坏事务 ACID 特性的因素**
     - **多个事务并行运行时，不同事务的操作交叉执行**
     - **事务在运行过程中被强行停止**
3. **事务的状态与事务状态图**
   - **（1） 事务的状态（如中止、提交、回滚）：P359-360**
   - **（2） 事务状态图作用：**

- **便于系统跟踪各事务执行情况，保证事务原子性和持久性特点**

4. 调度

(1) 调度：

- **一组指令包括指令在系统中执行的特定时间顺序。(指令可能来自多个事务，包括 commit,abort 指令)**

(2) 串行调度：

- **调度中凡属于同一个事务的指令都紧挨着一起。即一个事务的指令都执行完成后在执行下一事务**
- **虽保证执行结果正确，但应用并发执行度差。不能充分利用系统资源，发挥数据库共享资源的特点**

(3) 并行调度：

- **调度中多个事务的指令在时间上相互交叉地在执行**
- **虽应用并发执行度高，但执行结果可能错误。是单处理机系统中的并发方式，能够减少处理机的空闲时间，提高系统的效率**

(4) 可串行化调度：

- **虽然可能是一个并行调度但在执行的效果上等同于某一个串行调度执行结果**
- **应用并发执行度较高且执行结果依然正确**
- **如 P363 的图 14-5 调度不是一个可串行调度，其结果与串行调度T1→T2 和 T2→T1 的结果都不相同**

5. 串行化

(1) 冲突可串行化：P364

(2) 优先图：P364-365，用于判断一个并行调度是否为冲突可串行化调度

6. 可恢复调度和无级联调度：P366

7. 事务隔离性级别

(1) 分类：P366-367

(2) 应用情景：

- 通常情况下，要求数据库 DBMS 始终保持数据的一致性。此时采用可串行化级别。
- 但特殊情况，可以允许放松要求，如数据字典中统计值；又如，当一些应用仅仅需要粗略（非精确）统计信息时。此时采用后三种级别。

(3) 实现方式：P368，锁/时间戳/快照隔离

8. SQL 的事务处理分类（按事务的启动与执行方式）

(1) 显示事务

- 定义：也称之为用户定义或用户指定的事务，即可以显式地定义启动和结束的事务。分布式事务属于显示事务。
- 关键字：通过 begin transacton、commit transaction、commit work、rollback transaction 或 rollback work 等语句完成。
- 启动事务格式：begin tran 事务名或变量 with mark 描述
- 结束事务格式：commit tran 事务名或变量或 commit work 但此没有参数
- 回滚事务格式：rollback tran 事务名或变量 | savepoint_name | savepoint_variable 或 rollback work

说明:清除自事务的起点或到某个保存点所做的所有数据修改

- 在事务内设置保存点格式: save tran savepoint_name | savepoint_variable

(2) 自动提交事务
- 默认事务管理模式。如果一个语句成功地完成，则提交该语句；如果遇到错误，则回滚该语句。
- sql 连接在 begin tran 语句启动显式事务，或隐性事务模式设置为打开之前，将以自动提交模式进行操作。当提交或回滚显式事务，或者关闭隐性事务模式时，将返回到自动提交模式。

(3) 隐性事务
- 当连接以此模式进行操作时，sql 将在提交或回滚当前事务后自动启动新事务。
- 无须描述事务的开始，只需提交或回滚每个事务。它生成连续的事务链。
- 通过 API 函数或 Transact-SQL SET IMPLICIT_TRANSACTIONS ON 语句将隐性事务模式设置为打开。下一个语句自动启动一个新事务。当该事务完成时，再下一个 Transact-SQL 语句又将启动一个新事务。

第十二章：并发控制

1. 多事务执行方式

(1) 串行执行方式
- 每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行
- 不能充分利用系统资源，发挥数据库共享资源的特点

(2) 交叉并发方式（interleaved concurrency）
- 事务的并行执行是这些并行事务的并行操作轮流交叉运行
- 是单处理机系统中的并发方式，能够减少处理机的空闲时间，提高系统的效率

(3) 同时并发方式（simultaneous concurrency）
- 多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务真正的并行运行
- 最理想的并发方式，但受制于硬件环境

2. 并发控制机制的任务

(1) 对并发操作进行正确调度
(2) 保证事务的隔离性
(3) 保证数据库的一致性

3. 并发操作带来的数据不一致性

(1) 丢失修改（lost update）

指事务 1 与事务 2 从数据库中读入同一数据并修改事务 2 的提交结果破坏了事务 1 提交的结果，导致事务 1 的修改被丢失。

(2) 不可重复读（non-repeatable read）

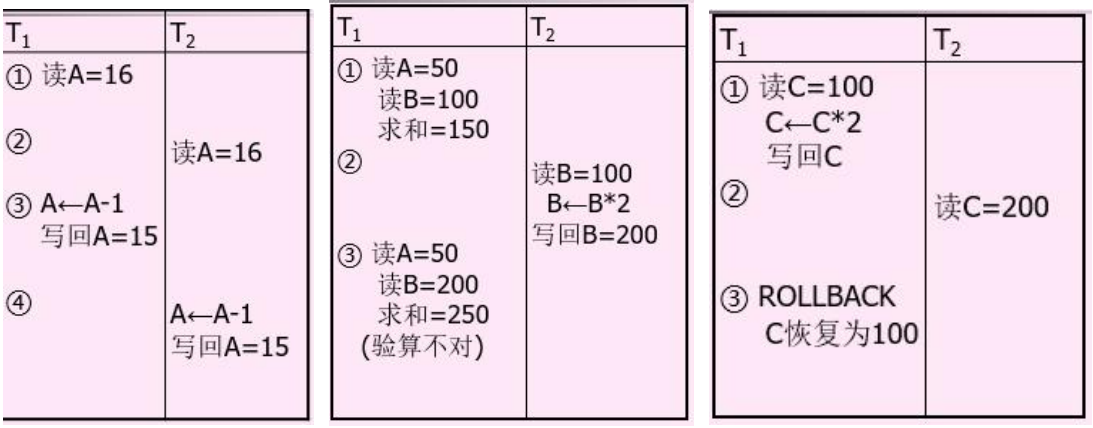
指事务 1 读取数据后，事务 2 执行更新操作，使事务 1 无法再现前一次读取结果。
- 事务 1 读取某一数据后:
  1. 事务 2 对其做了修改，当事务 1 再次读该数据时，得到与前一次不同的值。
  2. 事务 2 删除了其中部分记录，当事务 1 再次读取数据时，发现某些记录神密消失了。
  3. 事务 2 插入了一些记录，当事务 1 再次按相同条件读取数据时，发现多了一些记录。

后两种不可重复读有时也称为幻影现象（phantom row）。

(3) 读“脏”数据（dirty read）

事务 1 修改某一数据，并将其写回磁盘。事务 2 读取同一数据后，事务 1 由于某种原因被撤消，这时事务 1 已修改过的数据恢复原值。事务 2 读到的数据就与数据库中的数据不一致，是不正确的数据，又称为“脏”数据。

(4) 示例



丢失修改　　不可重复读　　读脏数据

4. 封锁

(1) 定义

- 封锁就是事务T 在对某个数据对象（例如表、记录等）操作之前，先向系统发出请求，对其加锁
- 加锁后事务T 就对该数据对象有了一定的控制，在事务T 释放它的锁之前，其它的事务不能更新此数据对象。

(2) 类型

- 排它锁/写锁（eXclusive lock，简记为 X 锁）
  若事务T 对数据对象 A 加上 X 锁，则只允许T 读取和修改A，其它任何事务都不能再对A 加任何类型的锁，直到T 释放A 上的锁。T 对 A 可读可写。
- 共享锁/读锁（Share lock，简记为 S 锁）
  若事务T 对数据对象 A 加上 S 锁，则其它事务只能再对A 加 S 锁，而不能加X 锁，直到 T 释放A 上的S 锁。T 对A 可读不可写。

(3) 锁的相容性矩阵

T_2 / T_1	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

(4) 注意：加锁和对数据的读写操作没有直接联系，需要根据封锁协议确定执行何种操作的时候是否加锁、加何种锁。

5. 封锁协议

(1) 在运用X 锁和 S 锁对数据对象加锁时，需要约定一些规则：封锁协议（Locking Protocol）

- 何时申请X 锁或S 锁

- 持锁时间、何时释放

(2) 作 用：

不同的封锁协议，在不同的程度上为并发操作的正确调度提供一定的保证

(3) 封锁协议分类：一级/二级/三级封锁协议

(4) 1 级封锁协议

- 定义：事务 T 在修改数据 R 之前必须先对其加X 锁，直到事务结束才释放（事务结束：正常结束（COMMIT）和非正常结束（ROLLBACK））
- 特点：1 级封锁协议可防止丢失修改；但如果是读数据，是不需要加锁的，所以它不能保证可重复读和不读“脏”数据。

(5) 2 级封锁协议

- 定义：1 级封锁协议+事务 T 在读取数据R 前必须先加S 锁，读完后即可释放S 锁
- 特点：2 级封锁协议可以防止丢失修改和读“脏”数据。在 2 级封锁协议中，由于读完数据后即可释放S 锁，所以它不能保证可重复读。

(6) 3 级封锁协议

- 定义：1 级封锁协议+事务 T 在读取数据R 之前必须先对其加S 锁， 直到事务结束才释放
- 特点：3 级封锁协议可防止丢失修改、读脏数据和不可重复读。

(7) 三级协议区别

- 什么操作需要申请封锁
- 何时释放锁（即持锁时间）

	X锁		S锁		一致性保证		
	操作结束释放	事务结束释放	操作结束释放	事务结束释放	不丢失修改	不读"脏"数据	可重复读
1级封锁协议		√			√		
2级封锁协议		√	√		√	√	
3级封锁协议		√		√	√	√	√

6. 活锁与死锁

(1) 活锁

- 定义：（类似饥饿，某个事务一直等待）
  活锁指的是任务或者执行者没有被阻塞，由于某些条件没有满足，导致一直重复尝试—失败—尝试—失败的过程。处于活锁的实体是在不断的改变状态，活锁有可能自行解开。
- 解决：采用先来先服务的策略
  当多个事务请求封锁同一数据对象时，按请求封锁的先后次序对这些事务排队。该数据对象上的锁一旦释放，首先批准申请队列中第一个事务获得锁。

(2) 死锁

- 定义：（类似数据库死锁）
- 原因：两个或多个事务都已封锁了一些数据对象，然后又都请求对已为其他事务封锁的数据对象加锁，从而出现死等待。
- 解决：

法 1. 死锁预防：破坏产生死锁的条件

a) 死锁预防方法 1：一次封锁法

- **定义：要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行**
- 问题 1：降低并发度
  - 扩大封锁范围
  - 将以后要用到的全部数据加锁，势必扩大了封锁的范围，从而降低了系统的并发度
- 问题 2：难于事先精确确定封锁对象
  - 数据库中数据是不断变化的，原来不要求封锁的数据，在执行过程中可能会变成封锁对象，所以很难事先精确地确定每个事务所要封锁的数据对象
  - 解决方法：将事务在执行过程中可能要封锁的数据对象全部加锁，这就进一步降低了并发度。

b) 死锁预防方法 2：顺序封锁法

- **定义：预先对数据对象规定一个封锁顺序，所有事务都按这个顺序实行封锁。**
- 问题 1：维护成本高
- 问题 2：数据库系统中可封锁的数据对象极其众多，并且随数据的插入、删除等操作而不断地变化，要维护这样极多而且变化的资源的封锁顺序非常困难，成本很高
- 问题 3：难于实现
- 问题 4：事务的封锁请求可以随着事务的执行而动态地决定，很难事先确定每一个事务要封锁哪些对象，因此也就很难按规定的顺序去施加封锁。

- **结论：**

A. 在操作系统中广为采用的预防死锁的策略并不适合数据库的特点

B. DBMS 在解决死锁的问题上普遍采用的是诊断并解除死锁的方法

法 2. 死锁的诊断与解除（死锁检测与恢复）：允许死锁发生，发生后解除死锁

a) 特点：由 DBMS 的并发控制子系统定期检测系统中是否存在死锁，一旦检测到死锁，就要设法解除

b) 死锁检测方法 1：超时法

- **定义：如果一个事务的等待时间超过了规定的时限，就认为发生了死锁**
- 优点：实现简单
- 缺点：
  - 有可能误判死锁

  时限若设置得太长，死锁发生后不能及时发现

c) 死锁检测方法 2：等待图法（P381）

- **定义：用事务等待图动态反映所有事务的等待情况**
- 事务等待图：
  一个有向图 G=(T，U)，T 为结点的集合，每个结点表示正运行的事务；U 为边的集合，每条边表示事务等待的情况。若 T1 等待 T2，则 T1，T2 之间划一条有向边，从 T1 指向 T2。
- 并发控制子系统周期性地（比如每隔 1 min）检测事务等待图，**如果发现图中存在回路，则表示系统中出现了死锁。**

d) 解除死锁：

- **方法：选择一个处理死锁代价最小的事务，将其撤消，释放此事务持有的所有的锁，使其它事务能继续运行下去。**
- 步骤：选择牺牲者→回滚→饿死（P381）

7. 并发调度的可串行性

(1) 正确的并发调度操作

- **所有事务的串行执行。**
- **几个事务的并行执行是正确的，当且仅当其结果与按某一次序串行地执行它们时的结果相同。**
- **总之，可串行性是并行事务正确性的唯一准则。**

(2) 保证并发操作调度正确性的方法

- **封锁方法：两段锁（Two-Phase Locking，简称 2PL）协议**
- **时标方法**
- **乐观方法**

8. 两段锁协议

(1) 内容

- **在对任何数据进行读、写操作之前，事务首先要获得对该数据的封锁**
- **在释放一个封锁之后，事务不再获得任何其他封锁**

(2) 含义：P376

- 遵循两段锁的封锁序列：
  Slock A … Slock B … Xlock C … Unlock B … Unlock A … Unlock C;
- 不遵循两段锁的封锁序列：
  Slock A … Unlock A … Slock B … Xlock C … Unlock C … Unlock B;

(3) 特点：

- **事务遵守两段锁协议是可串行化调度（并行执行结果正确）的充分不必要条件**
- **符合两段锁协议→可串行化调度（正确）**
- **可串行化调度→符合两段锁协议（错误）**

(4) 两段锁协议与防止死锁的一次封锁法

- 一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行，因此**一次封锁法遵守两段锁协议**
- 但是两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，因此**遵守两段锁协议的事务可能发生死锁**

(5) 两段锁协议与三级封锁协议

- **目的不同：**
  - **两段锁协议：保证并发调度的正确性**
  - **三级封锁协议：在不同程度上保证数据一致性**
- **遵守第三级封锁协议必然遵守两段协议**

9. 封锁粒度

(1) 定义：封锁对象的大小称为封锁的粒度(Granularity)

- **X 锁和 S 锁都是加在某一个数据对象上的**

- **封锁的对象:逻辑单元，物理单元**
- **例：在关系数据库中，封锁对象：**
  - **逻辑单元：属性值、属性值集合、元组、关系、索引项、整个索引、整个数据库等**
  - **物理单元：页（数据页或索引页）、物理记录等**

(2) 选择封锁粒度：

- **特点：封锁的粒度越大，系统被封锁的对象少，并发度小，系统开销小**
- **选择封锁粒度：考虑封锁结构和并发度两个因素，对系统开销与并发度进行权衡**
- **举例：**
  - 需要处理多个关系的大量元组的用户事务：以数据库为封锁单位;
  - 需要处理大量元组的用户事务：以关系为封锁单元;
  - 只处理少量元组的用户事务：以元组为封锁单位

10. 多粒度封锁

(1) 定义：在一个系统中同时支持多种封锁粒度供不同的事务选择

(2) 多粒度树：以树形结构来表示多级封锁粒度（P382）

(3) 显式封锁和隐式封锁

- **显式封锁：直接加到数据对象上的封锁**
- **隐式封锁：由于其上级结点加锁而使该数据对象加上了锁**
- **显式封锁和隐式封锁的效果是一样的**

(4) 多粒度封锁协议

1. **特点：P382**
2. **对某个数据对象加锁时系统检查的内容：**
   - **该数据对象：有无显式封锁与之冲突**
   - **所有上级结点：检查本事务的显式封锁是否与该数据对象上的隐式封锁冲突：(由上级结点封锁造成的)**
   - **所有下级结点：看上面的显式封锁是否与本事务的隐式封锁（将加到下级结点的封锁）冲突。**

11. 意向锁

(1) 目的：提高对某个数据对象加锁时系统的检查效率

(2) 含义：

- **对任一结点加基本锁，必须先对它的上层结点加意向锁**
- **如果对一个结点加意向锁，则说明该结点的下层结点正在被加锁**

(3) 常用意向锁：

1. 意向共享锁(Intent Share Lock，简称 IS 锁)
   - 如果对一个数据对象加 IS 锁，表示它的后裔结点拟（意向）加 S 锁。
   - 例：要对某个元组加 S 锁，则要首先对关系和数据库加 IS 锁
2. 意向排它锁(Intent Exclusive Lock，简称 IX 锁)
   - 如果对一个数据对象加 IX 锁，表示它的后裔结点拟（意向）加X 锁。
   - 例：要对某个元组加 X 锁，则要首先对关系和数据库加 IX 锁
3. 共享意向排它锁(Share Intent Exclusive Lock，简称 SIX 锁)

- 如果对一个数据对象加SIX 锁，表示对它加 S 锁，再加 IX 锁， 即 SIX = S + IX。
- 例：对某个表加 SIX 锁，则表示该事务要读整个表（所以要对该表加 S 锁），同时会更新个别元组（所以要对该表加 IX 锁）

(4) 相容性矩阵：P382

(5) 锁的强度：

- **定义：指它对其他锁的排斥程度**
- **特点：一个事务在申请封锁时以强锁代替弱锁是安全的，反之则不然**
- **排序：X>SIX>S=IX>IS>无锁**

(5) 具有意向锁的多粒度封锁方法：

- **特点：申请封锁时应该按自上而下的次序进行，释放封锁时则应该按自下而上的次序进行**
- **例：事务T 要对一个数据对象加锁，必须先对它的上层结点加意向锁**

12. 总结

(1) 数据的共享与一致

- **数据共享与数据一致性是一对矛盾。**
- **数据库的价值在很大程度上取决于它所能提供的数据共享度。**
- **数据共享在很大程度上取决于系统允许对数据并发操作的程度。**
- **数据并发程度又取决于数据库中的并发控制机制。**
- **另一方面，数据的一致性也取决于并发控制的程度。施加的并发控制愈多，数据的一致性往往愈好。**

(2) 数据库的并发控制以事务为单位

(3) 数据库的并发控制通常使用封锁机制——两类最常用的封锁（S/X 锁）

(4) 三级封锁协议

不同级别的封锁协议提供不同的数据一致性保证和不同的数据共享度。

(5) 并发控制机制调度并发事务操作是否正确的判别准则是**可串行性**

- **并发操作的正确性则通常由两段锁协议来保证。**
- **两段锁协议是可串行化调度的充分条件，但不是必要条件**

(6) 对数据对象施加封锁带来的活锁与死锁问题

- **活锁： 先来先服务**
- **死锁：**
  - **死锁预防：一次封锁法、顺序封锁法**
  - **死锁的诊断与解除：超时法、等待图法**

(7) 不同的数据库管理系统提供的封锁类型、封锁协议、达到的系统一致性级别不尽相同。但是其依据的基本原理和技术是共同的。

第十三章：备份与恢复

1. 数据库故障

(1) 影响：运行事务非正常中断，破坏数据库

(2) DBMS 对策：

- DBMS 提供恢复子系统
- 保证故障发生后，能把数据库中的数据从错误状态恢复到某种逻辑一致的状态
- 保证事务ACID

(3) 分类：P403

1. 事务故障

1) 概念：某个事务在运行过程中由于种种原因未运行至正常终止点就夭折了

2) 分类及原因：逻辑错误和系统错误，P403

3) 恢复：撤消事务（UNDO）

- 强行回滚（ROLLBACK）该事务，清除该事务对数据库的所有修改，使得这个事务象根本没有启动过一样

2. 系统故障

1) 概念：整个系统的正常运行突然被破坏，所有正在运行的事务都非正常终止。内存中数据库缓冲区（易失存储器）的信息全部丢失，外部存储设备（非易失存储器）上的数据未受影响。

2) 原因：

a) 操作系统或 DBMS 代码错误

b) 操作员操作失误

c) 特定类型的硬件错误（如 CPU 故障）

d) 突然停电

3) 恢复：

a) 清除尚未完成的事务对数据库的所有修改

- 系统重新启动时，恢复程序要强行撤消（UNDO）所有未完成事务

b) 将缓冲区中已完成事务提交的结果写入数据库

- 系统重新启动时，恢复程序需要重做（REDO）所有已提交的事务

3. 介质故障（磁盘故障）

1) 概念：硬件故障使存储在外存中的数据部分丢失或全部丢失

2) 特点：介质故障比前两类故障的可能性小得多，但破坏性大得多

3) 原因：

a) 磁盘损坏

b) 磁头碰撞

c) 操作系统的某种潜在错误

d) 瞬时强磁场干扰

4) 恢复：

e) 装入数据库发生介质故障前某个时刻的数据**副本**

f) **重做**自此时始的所有**成功事务**，将这些事务已提交的结果重新记入数据库

2. 数据库恢复技术

(1) 基本原理：冗余

- **利用存储在系统其它地方的冗余数据来重建数据库中已被破坏或不正确的那部分数据**

(2) 关键问题

1. **如何建立冗余数据**
   - **数据转储（backup）**
   - **登录日志文件（logging）**
2. **如何利用冗余数据实施数据库恢复**

(3) 转储（备份）

1. **概念：转储是指 DBA 将整个数据库复制到磁带或另一个磁盘上保存起来的过程。这些备用的数据文本称为后备副本或后援副本。**
2. **转储方法分类：**
   a) **按转储状态：静态转储、动态转储**
   b) **按转储方式：海量转储、增量转储**
   二者可以交叉组合，如：静态海量转储
3. **静态转储**
   a) 概念：在系统中无运行事务时进行转储。转储开始时数据库处于一致性状态，转储期间不允许对数据库的任何存取、修改活动。
   b) 优点：实现简单
   c) 缺点：
   - 降低了数据库的可用性：转储必须等用户事务结束，新的事务必须等转储结束
   - 见课本 P403
4. **动态转储**
   a) 概念：转储操作与用户事务并发进行，转储期间允许对数据库进行存取或修改
   b) 优点：
   - 不用等待正在运行的用户事务结束
   - 不会影响新事务的运行

   c) 缺点：不能保证副本中的数据正确有效
   d) 利用动态转储得到的副本进行故障恢复
      A. 需要把动态转储期间各事务对数据库的修改活动登记下来， 建立日志文件
      B. 后备副本加上日志文件才能把数据库恢复到某一时刻的正确状态
5. **海量转储与增量转储**
   a) 海量转储: 每次转储全部数据库
   b) 增量转储: 只转储上次转储后更新过的数据
   c) 海量转储与增量转储比较
   - 从恢复角度看，使用海量转储得到的后备副本进行恢复往往更方便
   - 但如果数据库很大，事务处理又十分频繁，则增量转储方式更实用更有效

6. 转储策略
   - 应定期进行数据转储，制作后备副本。但**转储又是十分耗费时间和资源的，不能频繁进行。**DBA 应该根据数据库使用情况确定适当的转储周期和转储方法。

(4) 日志文件

1. **概念：日志文件(log)是用来记录事务对数据库的更新操作的文件**
2. **格式：**
   - **以记录为单位的日志文件**
   - **以数据块为单位的日志文件（P406）**
3. **日志文件内容：**
   - **各个事务的开始标记(BEGIN TRANSACTION)**
   - **各个事务的结束标记(COMMIT 或 ROLLBACK)**
   - **各个事务的所有更新操作**
   - **与事务有关的内部更新操作**
4. **日志记录内容：**
   a) **基于记录的日志文件**
      - **事务标识**
      - **操作类型（插入、删除或修改）**
      - **操作对象（记 录ID、Block NO.）**
      - **更新前数据的旧值（对插入操作而言，此项为空值）**
      - **更新后数据的新值（对删除操作而言,此项为空值）**

   b) **基于数据块的日志文件：P406**
5. **用途：**
   a) **进行事务故障恢复**
   b) **进行系统故障恢复**
   c) 协助**后备副本**进行**介质故障**恢复
      A. 与静态转储后备副本配合
      - **静态转储的数据已是一致性的数据。**
      - 如果静态转储完成后，仍能定期转储日志文件，则在出现介质故障重装数据副本后，可以利用这些日志文件副本对已完成的事务进行重做处理。这样不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态。

      B. LOG FILE + 动态转储后备副本
      - **动态转储数据库：同时转储同一时点的日志文件。**
      - 后备副本与该日志文件结合起来才能将数据库恢复到一致性状态。利用这些日志文件副本进一步恢复事务，避免重新运行事务程序。
6. **原则：**
   a) **登记的次序严格按并行事务执行的时间次序**
   b) **必须先（把日志记录）写日志文件，后（把对数据的修改）写数据库**

   问：为什么要先写日志文件?
   - 写数据库和写日志文件是两个不同的操作
   - 在这两个操作之间可能发生故障

- 如果先写了数据库修改，而在日志文件中没有登记下这个修改，则以后就无法恢复这个修改了
- 如果先写日志，但没有修改数据库，按日志文件恢复时只不过是多执行一次不必要的 UNDO 操作，并不会影响数据库的正确性

3. 数据库恢复策略

(1) 事务故障恢复

1. **方法：由恢复子系统应利用日志文件撤消（UNDO）此事务已对数据库进行的修改**
2. **特点：事务故障的恢复由系统自动完成，不需要用户干预**
3. **步骤：**
   1. **反向扫描**文件日志（即从最后向前扫描日志文件），查找该事务的更新操作。
   2. 对该事务的更新操作执行逆操作。即将日志记录中“更新前的值”（Before Image, BI）写入数据库。
   3. 继续**反向扫描**日志文件，查找该事务的其他更新操作，并做同样处理。
   4. 如此处理下去，直至读到此事务的开始标记，事务故障恢复完成。

(2) 系统故障恢复

1. **系统故障造成数据库不一致状态的原因：**
   - **一些未完成事务对数据库的更新已写入数据库**
   - **一些已提交事务对数据库的更新还留在缓冲区没来及写入数据库**
2. **方法：Undo 故障发生时未完成的事务，Redo 已完成的事务**
3. **特点：系统故障的恢复由系统在重新启动时自动完成，不需要用户干预**
4. **步骤：**
   1. **正向扫描**日志文件（即从头扫描日志文件）
      - **Redo 队列：在故障发生前已经提交的事务**
      - **Undo 队列：故障发生时尚未完成的事务**
   2. 对 Undo 队列事务进行 UNDO 处理
      反向扫描日志文件，对每个 UNDO 事务的更新操作执行逆操作
   3. 对 Redo 队列事务进行 REDO 处理
      正向扫描日志文件，对每个REDO 事务重新执行登记的操作

(3) 介质故障恢复

1. **特点：**
   - **需要 DBA 介入，但具体的恢复操作仍由 DBMS 完成**
   - **DBA 的工作：**
     - **重装最近转储的数据库副本和有关的各日志文件副本，执行系统提供的恢复命令**
2. **步骤：**
   1. 装入最新的**后备数据库副本**，使数据库恢复到最近一次转储时的**一致性状态**。
      - 对于**静态转储**的数据库副本，装入后数据库即处于一致性状态
      - 对于**动态转储**的数据库副本，还须同时装入转储时刻的日志文件副本，利用与恢复系统故障相同的方法（即 REDO + UNDO），才能将数据库恢复到一致性状态。
   2. 装入有关的**日志文件副本**，**重做已完成**的事务。
      - 首先扫描日志文件，找出故障发生时已提交的事务的标识， 将其记入重

做队列。

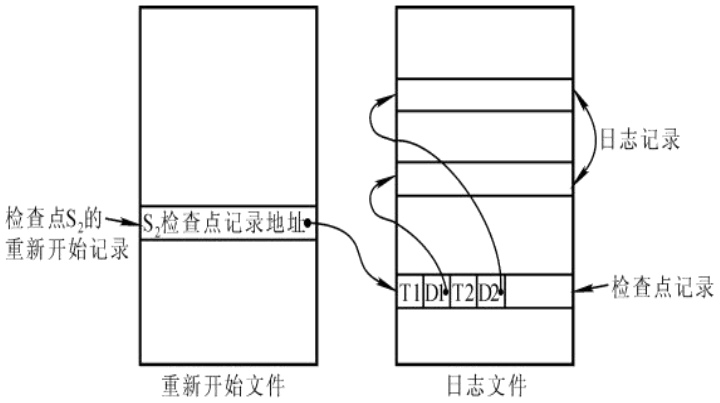
- 然后正向扫描日志文件，对重做队列中的所有事务进行重做处理。即将日志记录中“更新后的值”写入数据库。

4. 检查点技术（提高恢复效率）

(1) 提出原因：P410

(2) 具有检查点（checkpoint）的恢复技术特点

- **在日志文件中增加检查点记录（checkpoint）**
- **增加重新开始文件**
- **恢复子系统在登录日志文件期间动态地维护日志**



(3) 文件内容

A. 检查点记录的内容：

1. 建立检查点时刻所有正在执行的事务清单

2. 这些事务最近一个日志记录的地址

B. 重新开始文件的内容：记录各个检查点记录在日志文件中的地址

(4) 在检查点维护日志文件过程：（参照 P410）

1.将当前日志缓冲区中的所有日志记录写入磁盘的日志文件上。**（对应课本第 1 步）**

2.在日志文件中写入一个检查点记录。

3.将当前数据缓冲区的所有数据记录写入磁盘的数据库中。**（对应课本第 2 步）**

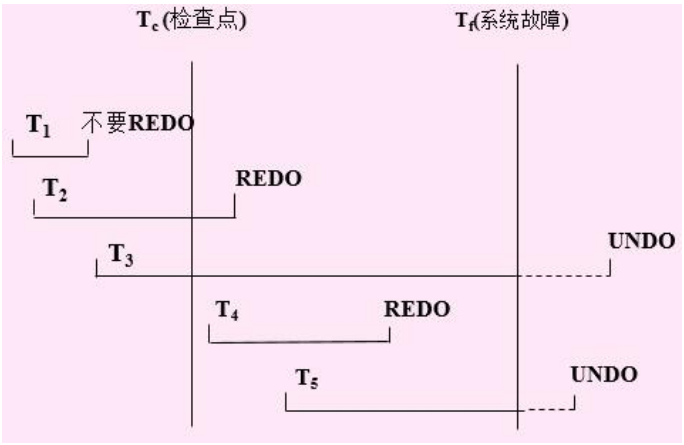
4.把检查点记录在日志文件中的地址写入一个重新开始文件。**（对应课本第 3 步）**

(5) 建立检查点策略

- 定期：按照预定的一个时间间隔
- 不定期：按照某种规则，如日志文件已写满一半建立一个检查点

(6) 利用检查点的恢复策略

- **当事务T在一个检查点之前提交，T对数据库所做的修改已写入数据库。在进行恢复处理时，没有必要对事务T执行REDO操作。**



(7) 利用检查点的恢复步骤：（参照 P410）

1. 从重新开始文件中找到最后一个检查点记录在日志文件中的地址，由该地址在日志文件中找到**最后一个检查点记录**。(对应课本第 1 步)
2. 由该检查点记录得到检查点建立时刻所有正在执行的事务清单 ACTIVE- LIST。建立两个事务队列：**UNDO-LIST 和 REDO-LIST**。把 ACTIVE-LIST 暂时放入 UNDO-LIST 队列，REDO 队列暂为空。
3. 从检查点开始**正向扫描**日志文件，直到日志文件结束。
   - **如有新开始的事务 Ti，把 Ti 放入 UNDO-LIST 队列**
   - **如有提交的事务Tj，把 Tj 从 UNDO-LIST 队列移到 REDO-LIST 队列**
4. 对 UNDO-LIST 中的每个事务执行 UNDO 操作，对 REDO-LIST 中的每个事务执行 REDO 操作。（对应课本第 2 步）

5. 总结

(1) DBMS 必须对事务故障、系统故障和介质故障进行恢复

(2) 恢复中最经常使用的技术：数据库转储和登记日志文件

(3) 恢复的基本原理

利用存储在后备副本、日志文件和数据库镜像中的冗余数据来重建数据库

(4) 常用恢复技术

- **事务故障的恢复：UNDO**
- **系统故障的恢复：UNDO + REDO**
- **介质故障的恢复：重装备份并恢复到一致性状态 + REDO**

(5) 提高恢复效率的技术

a) 检查点技术

- **可以提高系统故障的恢复效率**
- **可以在一定程度上提高利用动态转储备份进行介质故障恢复的效率**

b) 镜像技术：可以改善介质故障的恢复效率

ppt目录

题

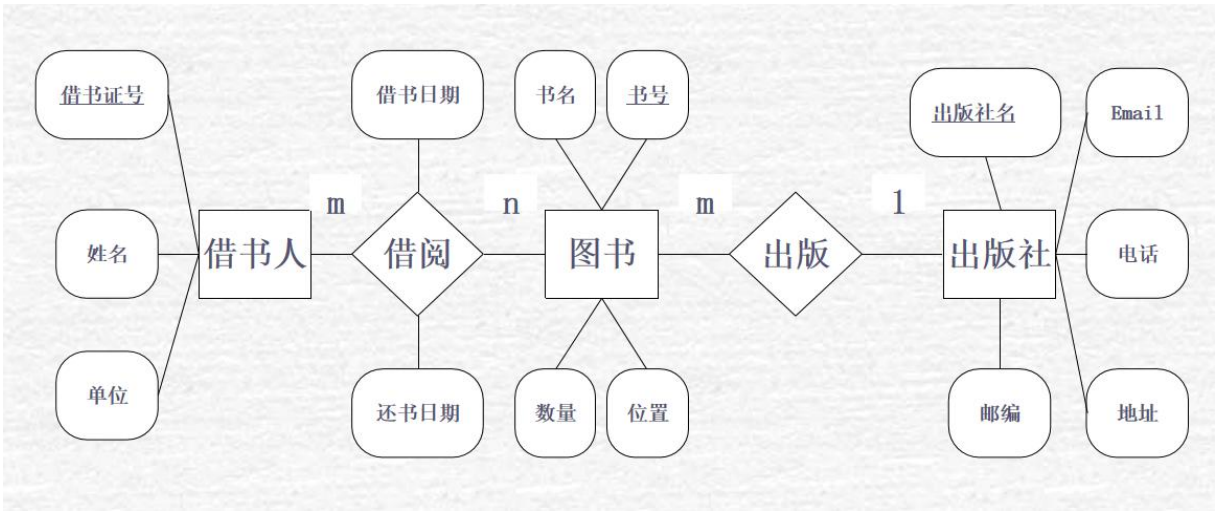
1. 图书借阅管理系统的ER图（ppt23），转换关系模式（ppt48）

练习

图书借阅管理系统具有以下功能:

1. 可随时查询书库中现有书籍的数量与存放位置。
   - 所有各类书籍均可由书号唯一标识。
2. 可随时查询书籍借还情况，包括借书人单位、姓名、借书证号、借书日期和还书日期。
   - 任何人可借多种书，任何一种书可为多个人所借；
   - 借书证号具有唯一性。
3. 可通过数据库中保存的出版社的Email、电话、邮编及地址等信息向相应出版社增购有关书籍。
   - 一个出版社可出版多种书籍，同一本书仅为一个出版社出版；
   - 出版社名具有唯一性。

请为该系统作概念模型设计，画出ER图.

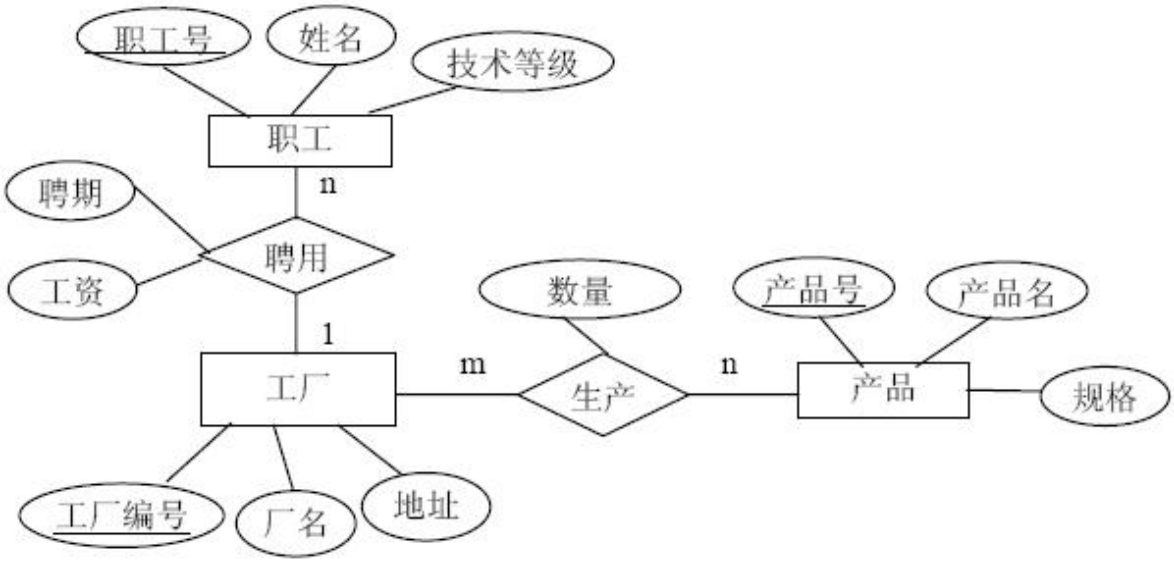


借书人（借书证号，姓名，单位）
图书（书号，书名，数量，位置，出版社名）
出版社（出版社名，Email，电话，邮编，地址）
借阅（借书证号，书号，借书日期，还书日期）

2. 企业生产产品绘制ER图（ppt30），转换关系模式（ppt48）

随堂小测试

某企业集团有若干工厂，每个工厂生产多种产品，且每一种产品可以在多个工厂生产，每个工厂按照固定的计划数量生产产品，计划数量不低于300；每个工厂聘用多名职工，且每名职工只能在一个工厂工作，工厂聘用职工有聘期和工资。工厂的属性有工厂编号、厂名、地址，产品的属性有产品编号、产品名、规格，职工的属性有职工号、姓名、技术等级。请为该集团进行概念设计，画出E-R图。

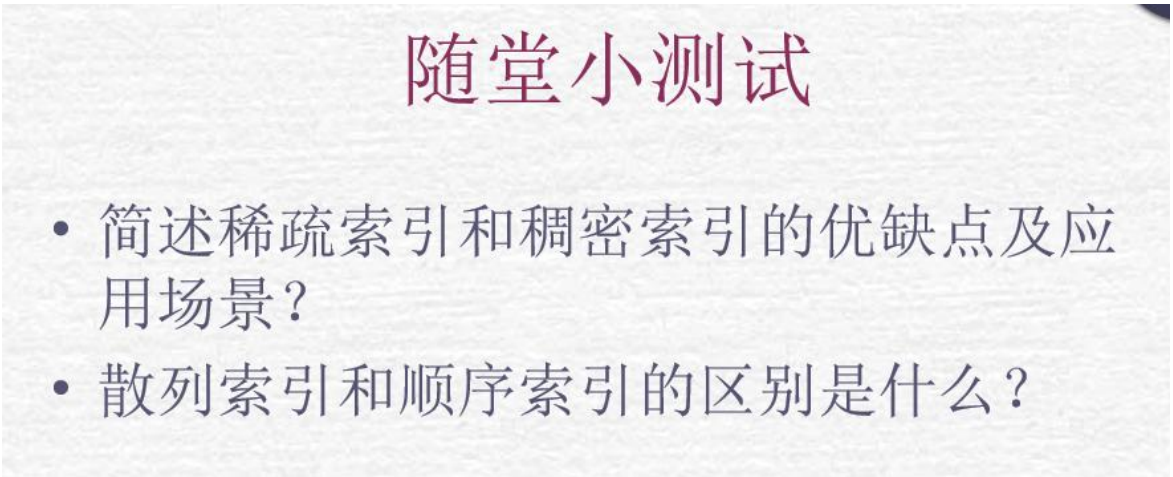


职工（职工号，姓名，技术等级，工厂编号，聘期，工资）
工厂（工厂编号，厂名，地址）
产品（产品号，产品名，规格）
生产（工厂编号，产品号，数量）

3. 某大学的ER图转换关系模式（ER图ppt44）

classroom(*building*, *room_number*, *capacity*)
department(*dept_name*, *building*, *budget*)
course(*course_id*, *title*, *dept_name*, *credits*)
instructor(*ID*, *name*, *dept_name*, *salary*)
section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)
student(*ID*, *name*, *dept_name*, *tot_cred*)
takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
advisor(*s_ID*, *i_ID*)
time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
prereq(*course_id*, *prereq_id*)

4. 索引对比



稀疏索引与稠密索引：（书上的）
优缺点：
1.稠密索引比稀疏索引更快地定位一条记录。
2.稀疏索引所占空间小，并且插入和删除时所需的维护开销也小。
应用场景：
稠密索引：可以为任何搜索码建立稠密索引，但是对于大型数据库，相应的空间开销和维护开销也会增大。
稀疏索引：只有索引是聚集索引时才能使用稀疏索引。一般为每个块建立一个索引项的稀疏索引是一个较好的折中。

网上的答案：
1、简述稀疏索引和稠密索引的优缺点及应用场景？
稀疏矩阵占用的空间小，速度慢，精确率相对于稠密索引低，插入、删除记录代价较大。常应用于搜索引擎搜索记录的数据库。
稠密索引占用的空间大，速度快，方便插入删除。常应用于订单数据库，精准，快速。
2、散列索引和顺序索引的区别是什么？
顺序索引：基于搜索码值的顺序排序。顺序索引更适合与范围查询
散列索引：采用散列函数将搜索码映射到散列桶，通过散列索引，支持基于搜索码的记录快速查找。散列索引更适用于等值查询