Chapter5 规范化设计

规范化设计理论 = 模式设计理论

- 数据依赖（核心）：数据直接的联系。函数依赖
- 范式：关系模式的标准。
- 模式设计方法

5.1 关系模式的设计问题

5.1.1 关系模型的外延和内涵

外延：关系 / 表 / 当前值。

内涵：数据的定义，数据完整性约束的定义。

- 数据的定义：关系，属性，域的定义和说明
- 完整性约束：静态约束（数据依赖，主键，值域），动态约束（定义操作对关系值的影响）

关系模式 = 内涵

5.1.2 泛关系模式与数据库模式

泛关系模式：所有属性组成的关系模式 R(U)

泛关系：当前值 r

数据库模式：$\rho = R_1, R_2, \cdots, R_n$, 其中 $R_1 \cup R_2 \cup \cdots \cup R_n = R(U)$

数据库：R_i的实例

5.1.3 冗余和异常

数据冗余：同一个数据重复存储多次。

操作异常：修改异常，插入异常，删除异常

5.2 函数依赖 FD

5.2.1 函数依赖

1. 函数依赖：X -> Y
    - **X**相等，则**Y**一定相等；**Y**相等，则**X**不一定相等
    - 例：1对多 A:B = 1:N　B -> A; 1对1　A:B = 1:1　A->B　B->A
2. 部分依赖：$X->Y$，且存在X的真子集x',满足$x'->Y$
3. 完全依赖/ 左部不可约依赖：$X->Y$，且不存在X的真子集x',满足$x'->Y$
4. 传递依赖
5. 候选键，超键
6. 主属性：候选键中的属性。
    a. 如何选择候选键？
        - 不在函数依赖右部出现的属性，一定是
        - 只在函数依赖右部出现的属性，一定不是

5.2.2 蕴含逻辑

1. 逻辑蕴含 F|= X->Y
2. 函数依赖集F的闭包 F^+ : { 被F蕴含的函数依赖全体}

5.2.3 FD推理规则

1. "Armstrong" 公理
    a. 自反性：若$Y \subseteq X \subseteq U$,则$X \to Y$在$R$上成立。
    b. 增广性：若$X \to Y$在R上成立，且$Z \subseteq U$,则$XZ \to YZ$在R上成立。
    c. 传递性：若$X \to Y$和$Y \to Z$在R上成立，则$X \to Z$在R上成立。
2. 常用的推理规则
    a. 合并性　若 X→Y，X→Z，则 X→YZ。
    b. 分解性　若 X→Y，Z⊆Y，则 X→Z。
    c. 伪传递性　若 X→Y，YW→Z，则 XW→Z。

d. 复合性　若X→Y，W→Z，则XW→YZ。

e. 通用一致性　若X→Y，W→Z，则X∪（W－Y）→YZ。

3. 充要条件：如果$A_1 \ldots A_n$是关系模式R的属性集，**那么 $X \to A_1 \ldots A_n$ 成立的充分必要条件是$X \to A_i$（i=1，…，n）成立。**

4. 平凡的FD，非平凡的FD

5. 属性集的闭包X^+

定义：设F是属性集U上的FD集,X是U的子集，那么（相对于F）属性集X的闭包用X_F^+表示，它是一个从F集使用FD推理规则推出的所有满足X→A的属性A的集合：

$$X_F^+ = \{ A \mid A \in U，X \to A \in F^+ \}$$

6. F^+ 有 X->Y == $Y \subseteq X^+$

7. 等价的依赖集：在U上依赖集闭包相等，即 $F = G$, 则$F^+ = G^+$。

8. 最小依赖集F_{min}^+

a. 消除右端冗余：右端全分解为单一属性。

b. 消除左端冗余：AD - > B, 用 D -> B 代替，求A^+，若能求出B，则A冗余。

c. 消除FD冗余：取出一个依赖，求F^+，判断是否等价。

5.3 关系模式的分解特性

1. 无损分解，损失分解的判定

a. 分解 = 2 模式，R1 ∩ R2 -> R1 - R2 / R2 - R1

b. Chase规则（表格）

2. 保持函数依赖的判定

a. 求分解模式在F上的投影

b. 投影的并=F，则保持

5.4 范式

5.4.1 范式的概念

1. 局部依赖，完全依赖，传递依赖；主属性，非主属性

2. 1NF：属性不可分。

3. 2NF：1NF，且非主属性完全依赖于候选键。

4. 3NF：1NF，且非主属性都不传递依赖于候选键。

    a. 对于F中每个非平凡FD X->Y, 都有X是R的超键，或者Y的每个属性都是主属性。（左部右部，至少一边包含码）
5. BCNF：1NF，每个属性都不传递依赖于候选键。
    a. 对于F中每个非平凡FD X->Y, 都有X是R的超键。所有的决定因素都包含码（用于判断）

5.4.2 分解成3NF模式集

算法1：保持**ρ**的**3NF**模式集：U + Fmin

1. 所有F中没有出现的属性，放在一个单独的模式。
2. F中有 X -> A , 且{X,A} = U ,则停止
3. 不为a,b，所有左边相同的，放在同一分组X-A == XA
4. 分解结束，输出ρ

算法2：保持**ρ**，保持函数依赖的**3NF**模式集：U + Fmin

1. 算法1得到一个模式集
2. 候选键加入模式集。

5.4.3 分解成BCNF范式

保持无损分解，不一定保持函数依赖。

1. 置初值：ρ = {R}。
2. ρ已经满足BC范式，则结束。
3. 找到左边没有码的依赖X->A，把A从U里面删除，XA单独放在一个模式。继续，直到都满足BC范式。
4. 结束，输出ρ。

5.4.4 模式设计方法的原则

三个特性

1. ρ是BCNF模式集 , 或3NF模式集
2. 无损分解
3. 保持函数依赖

三个原则

1. 表达性：无损联接，保持函数依赖

2. 分离性：范式
3. 最小冗余性：模式个数和模式中属性总数应最少

Chapter7 数据库设计

7.1 数据库设计概述

数据库设计

- 对于给定的软、硬件环境，针对现实问题，设计一个较优的数据模型，建立DB结构和DB应用系统。
- 对于一个给定的应用环境，**提供一个确定最优数据模型与处理模式的逻辑设计**，以及一个确定数据库存储结构与存取方法的物理设计，**建立起既能反映现实世界信息和信息联系**，满足用户数据要求和加工要求，又能**被某个数据库管理系统所接受**，同时能实现系统目标，并有效存取数据的数据库。

7.1.1 软件生存期

1. 软件生存期：指从软件的规划、研制、实现、投入运行 后的维护，直到它被新的软件所取代而停止使用的整个期间。
2. 六个阶段：规划阶段、需求分析阶段、设计阶段、程序编制阶段、调试阶段、运行维护阶段

7.1.2 数据库系统生存期

1. 数据库应用系统：以**数据库为基础的信息系统。**具有对信息的**采集、组织、加工、抽取和传播**等功能。
2. 数据库工程：数据库应用系统的开发。
3. 数据库系统生存期：数据库应用系统从**开始规划、设计、实现、维护到最后被新的系统取代而停止使用**的整个期间。
4. 七个阶段
    1. **规划阶段**
        - 必要性和可行性分析。确定数据库系统的地位，以及数据库间联系。
    2. **需求分析阶段**
        - 分析用户需求
        - 信息要求（数据库中存储哪些数据）
        - 处理要求（需要的处理功能，响应时间要求，批/联机处理）
        - 安全性和完整性要求
    3. **概念设计阶段**
        - 表达用户整体要求，且独立于DBMS和硬件结构。
    4. **逻辑设计阶段**
        - **数据库逻辑结构设计**：ER图 => DBMS的DDL转换为逻辑数据库结构
        - **应用程序设计**：DBMS的DML 进行结构式的程序设计。
    5. **物理设计阶段**
        - **物理数据库结构的选择**
        - **逻辑设计中程序模块说明的精确化**（开发）。
        - 成果：一个完整的，能实现的数据库结构。
    6. **实现**
    7. **运行维护**

- 收集和记录系统实际运行的数据。

7.1.3 数据库设计的具体步骤

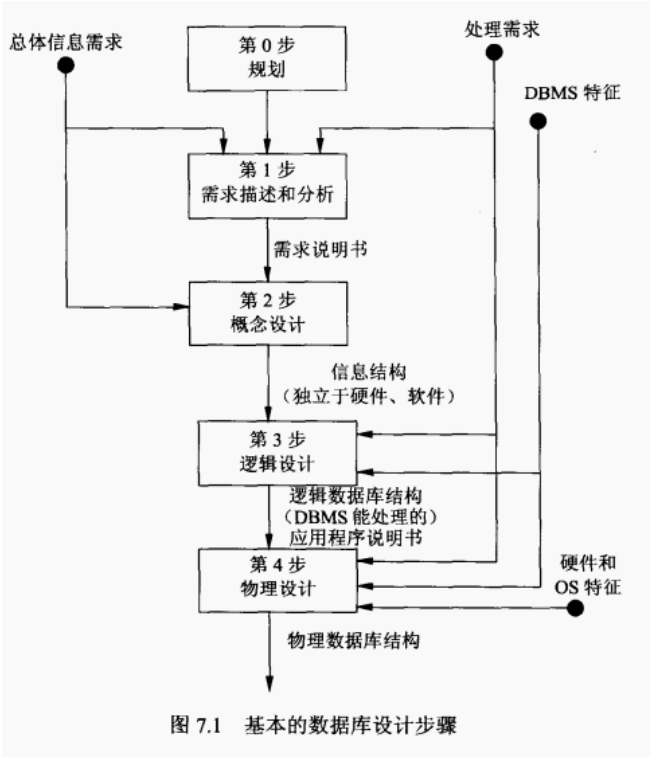


图 7.1　基本的数据库设计步骤

输入：

1. 总体信息需求：数据库的目标说明，数据元素的定义，数据在企业组织中的使用描述。
2. 处理需求：每个应用需要的数据项，数据量，应用执行的频率。
3. DBMS特征：DBMS说明+参数；DBMS模式+子模式，程序语法规则。
4. 硬件和OS特征：DBMS和OS访问方法特有的内容。

输出：说明书

1. **完整的数据库结构**：逻辑结构+物理结构。
2. 基于数据库结构和处理需求的**应用程序的设计原则。**

7.2 规划

任务：建立数据库的必要性及可行性分析，确定数据库系统在组织中和信息系统中的地位，以及各个数据库之间的联系。

输出：**可行性分析报告** + **数据库系统规划纲要**

后者包括：信息范围、信息来源、人力资源、设备资源、软件及支持工具资源、开发成本估算、开发进度计划、现行系统向新系统过渡计划等。

7.3 需求分析

7.3.1 需求描述与分析

1. 对系统的整个应用情况做**全面的详细的调查**，**确定企业组织的目标**
2. 收集**支持系统总的设计目标的基础数据和对这些数据的要求**，**确定用户的需求**
3. 并把这些要求写成用户和数据库设计者都能接受的文档。

7.3.2 需求分析阶段的输入和输出

1. 输入：总体信息需求（数据本质和概念上联系），处理需求（数据处理）
2. 输出：需求说明书（数据流图 + 数据字典）

7.3.3 需求分析的步骤

步骤

1. 分析用户活动，产生用户活动图（即**用户的业务流程图**);
2. 确定系统范围，产生系统范围图（即确定**人机界面**）；
3. 分析用户活动所涉及的数据,产生数据流图(**数据的流向及加工**);
4. 分析系统数据，产生数据字典。

数据流图：→ 数据流；◯ 加工 / 处理；= 文件；☐ 外部实体

数据字典：

- 定义：对系统中数据的详尽描述，提供了对数据库数据描述的集中管理。
- 内容：数据项，数据结构，数据流，数据存储，加工过程
- 功能：存储和检索各种数据描述，为DBA提供有关的报告。
- 在需求分析阶段建立，并在数据库设计过程中不断改进、充实和完善。

7.4 概念设计

目标：**产生**反映企业组织信息需求的数据库概念结构（**概念模式**）。

概念模式独立于数据库逻辑结构，也独立于支持数据库的DBMS。

7.4.1 概念设计的必要性

1. 各阶段的任务相对单一化，设计复杂程度大大降低，便于组织管理。
2. 不受特定的DBMS的限制，也独立于存储安排和效率方面的考虑，比逻辑模式更稳定。
3. 不含具体DBMS附加技术细节，用户理解容易，可以更准确反应用户的信息需求。

7.4.2 概念模型

1. ~是对现实世界的抽象和概括。
2. ~应简洁，明晰，独立于机器，容易理解。
3. ~应易于变动。
4. ~应很容易想关系，层次，网状等各种数据模型转换。

7.4.3 概念设计的主要步骤

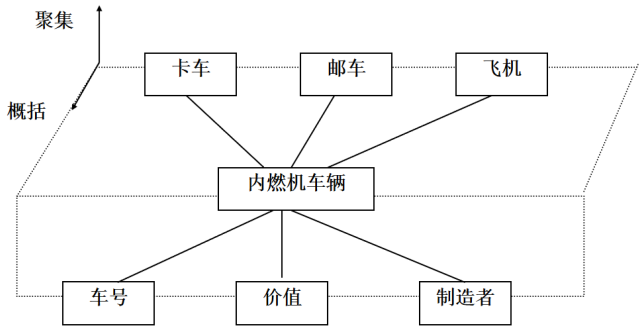
1. 进行数据抽象,设计局部概念模式；（“聚集” + “概括”）
2. 将局部概念模式综合成全局概念模式；（解决冲突：消除冗余，统一命名）
3. 评审。（用户评审 + DBA和应用开发人员评审）

7.4.4 数据抽象

1. 抽象：对实际的人、物、事或概念的人为处理，抽取人们关心的共同特性，忽略非本质的细节，将这些特性用各种概念精确地加以描述，这些概念组成了某种模型。
2. 两种形式：
    - 抽象对象：系统状态的抽象

- 抽象运算：系统转换的抽象

3. 聚集：笛卡尔积，形成对象之间的一个联系对象。
   - 聚集层次表示：“是……的一部分”（is part of）的关系。
4. 概括：从一类对象形成一个对象。
   - 概括层次表示：“是……一种”(is a)的关系。
5. 数据抽象层次
   - 每个对象既可以是聚集对象，也可以是概括对象。
   - 反复进行数据抽象，形成层次关系。



7.4.5 ER模型的操作

ER模型的操作：在利用ER模型进行数据库概念设计的过程中，常常对ER图进行的种种变换。 **（实体类型、联系类型和属性的分裂、合并和增删……）**

1. 实体类型的分裂
   1. 垂直分割：把一个实体类型的属性分成若干组，然后按组形成若干实体类型。 **键必须在所有实体中体现。**

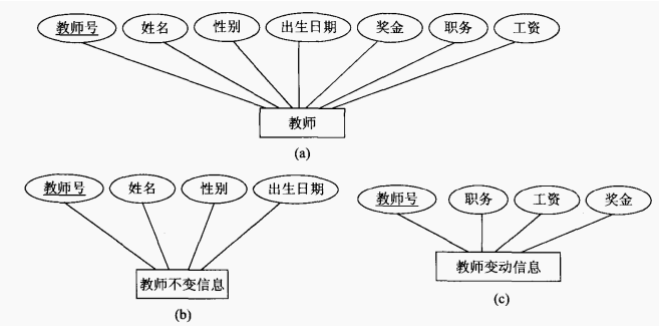


图 7.11　实体类型的垂直分割

   b. 水平分割：分裂成互补相交的子类。 e.g. 教师 => 男教师 + 女教师

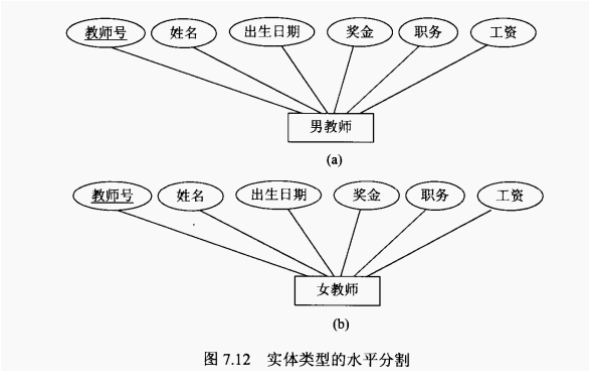


图 7.12　实体类型的水平分割

2. 实体类型的合并
   1. 水平合并，垂直合并。
   2. 是否会产生新联系视情况而定。

3. 联系类型的分裂
    1. 一个联系类型可分裂成几个新联系类型。新联系类型可能和原联系类型不同。

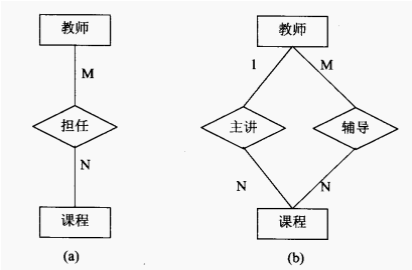


图 7.13　联系类型的分裂

4. 联系类型的合并
    1. 合并的联系类型必须是定义在相同的实体类型组合中，否则，不合法。

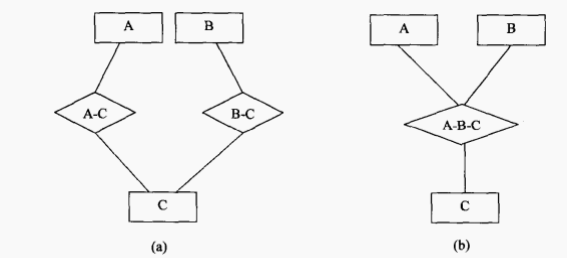


图 7.14　不合法的合并

7.4.6 采用ER方法的数据概念设计

第一步：设计局部ER模式：

1. 确定局部结构范围划分；
2. 实体定义；
3. 联系定义。

第二步：设计全局ER模式：

1. 确定公共实体类型；（同名实体类型；相同主键的实体）
2. 局部ER模式的合并；
3. 消除冲突 （属性冲突，结构冲突，命名冲突）

第三步：全局ER模式的优化原则：

1. 相关实体类型的合并：1:1的实体合并，部分相同键的实体
2. 冗余属性的消除；
3. 冗余联系的消除。

7.5 逻辑设计

目的：把概念设计阶段设计好的全局ER模式转换成与选用的具体机器上的DBMS所支持的数据模型相符合的逻辑结构（包括数据库模式和外模式）。

7.5.1 逻辑设计环境

1. 输入
    1. 独立于DBMS的概念模式：局部，全局概念模式（概念设计阶段）
    2. 处理需求：业务活动分析结果（需求分析阶段）
    3. 约束条件
    4. DBMS特性

2. 输出

    1. DBMS可处理的模式：说明
    2. 子模式
    3. 应用程序设计指南
    4. 物理设计指南：完全文档化的模式和子模式。

7.5.2 逻辑设计的步骤

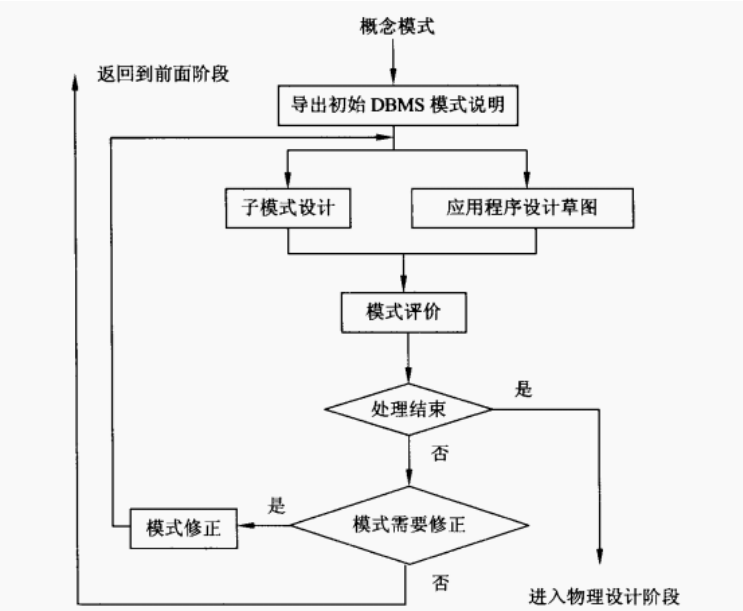


图 7.23　逻辑设计步骤

7.5.3 ER模型象关系模型的转换

ER模型转换为关系模型的一般规则:

1. 实体类型的转换：1个实体1张表。
2. 联系类型的转换
    1. 1:1　　任意一端加入另一端的主键，作为外键。
    2. 1：N　　N端加入1端主键，作为外键。
    3. 弱实体　1：N，N端为弱实体　　N端中加入1端主键，作为外键。且N端主键= 1端主键 + N端外键
    4. M：N　　新增一张表，主键为两端的外键构成。
3. 超类和子类的转换规则：子类 = 父类主键 + 新增属性

7.5.4 关系数据库的逻辑设计

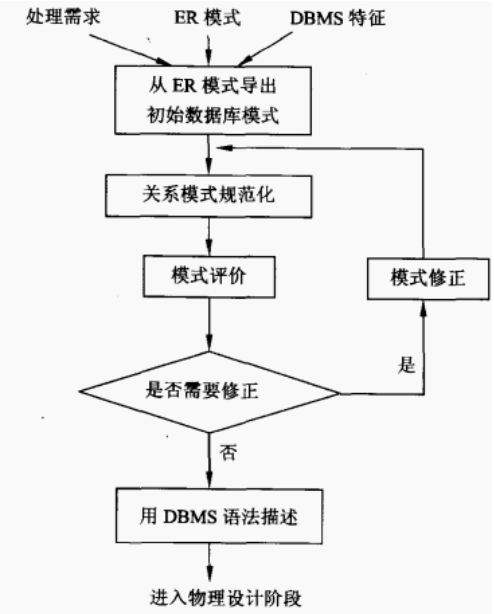


图 7.29　关系数据库的逻辑设计

1. 导出初始关系模式
2. 运用模式设计理论，对初始关系模式进行规范化处理
    1. 确定规范级别，3NF / BCNF
    2. 实施规范化处理
3. 模式评价：功能 + 性能
4. 模式修正

7.6 物理设计

1. 物理设计：对于给定的基本数据模型选取一个最适合应用环境的物理结构。
2. 数据库的物理结构：数据库的存储记录格式，存储记录安排，存取方法。
3. 物理设计
    1. **存储记录结构设计**：记录的组成、数据项的类型、长度，以及逻辑记录到存储记录的映射。
    2. **确定数据存放位置**：把经常同时被访问的数据组合在一起。
    3. **存取方法的设计**：主存取路径，辅存取路径。
    4. **完整性和安全性考虑**
    5. **程序设计**

7.7 数据库的实现

1. 数据库的实现阶段： 根据逻辑设计和物理设计的结果，在计算机系统上建立起实际数据库结构、装入数据、测试和试运行的过程。
2. 实现阶段：
    1. 建立实际数据库结构
    2. 装入试验数据对应用程序进行调试
    3. 装入实际数据，进入试运行状态

7.8 数据库的运行和维护

1. 维护数据库的安全性与完整性
2. 监测并改善数据库运行性能
3. 根据用户要求对数据库现有功能进行扩充

4. 及时改正运行中发现的系统错误

Chapter 8 Exercise

例1

一、某支付系统数据库有如下关系模式：

E 支付卡	ENO 卡编号	Ename 持卡人姓名	IDcard 身份证号	PassWord 密码	Balance 余额	CalScore 积分	MaxExpend 每笔最大消费额
EC 消费明细	CNO 消费单号	ENO 卡编号	Date 消费日期	Type 消费类型	Expend 消费金额		

提示：
- 消费明细中的每条消费金额必须满足该支付卡所限定的每笔最大消费额。
- 只有在消费明细表中成功地插入了一条消费明细记录后，才表示此次消费有效。
- 假定消费明细的日期为当前的系统日期。

请用指定的方法定义下列约束：

1. 用断言实现：不允许持卡人同一天消费不同的消费类型超过 5 种。
2. 用 SQL3 触发器完成如下操作：若发现某笔消费金额使支付卡余额透支超过 1000 元，则使此次消费无效。若此次消费有效，积分累加本次消费金额的 10%。

1. 断言实现

```
CREATE ASSERTION ASSE1 CHECK(
    5 >= ALL(SELECT COUNT(DISTINCT (TYPE)
             FROM EC
             GROUP BY ENO, DATE));

CREATE ASSERTION ASSE2 CHECHK(
    (NOT EXISTS (SELECT * FROM EC
                 GROUP BY ENO,DATE
                 HAVING COUNT(DISTINCT (TYPE) > 5)));
```

2. SQL3触发器

```
CREATE TRIGGER TRIG1
AFTER INSERT ON EC
REFERENCING
    NEW AS NEWTUPLE
WHEN (NEWTUPLE.ENO IS NOT NULL)
BEGIN ATOMIC
    UPDATE E     # 未超支1000，且小于单笔最大支出时，执行更新
    SET BALANCE = BALANCE - NEWTUPLE.EXPAND,
    CALSCORE = CALSCORE + NEWTUPLE.EXPEND * 0.1
    WHERE (NEWTUPLE.ENO = E.ENO) AND
          (E.BALANCE - NEWTUPLE.EXPEND > -1000) AND
          (MAXEXPEND >= NEWTUPLE.EXPEND)
    DELETE FROM EC  # 超支1000，或者单笔支出大于单笔时，撤销更新
        WHERE CNO = NEWTUPLE.CNO AND
            NEWTUPLE.EXPEND >= (SELECT MAXEXPEND FROM E
```

```
                                    WHERE E.ENO = NEWTUPLE.ENO) OR
            (-1000 > (SELECT BALANCE - NEWTUPLE.EXPEND FROM E
                        WHERE E.ENO=NEWTUPLE.ENO))
END
FROM EACH ROW;
```

例2

2. 用 SQL3 触发器完成如下操作：若发现某笔消费金额使支付卡余额透支超过 1000 元，则使此次消费无效。若此次消费有效，积分累加本次消费金额的 10%。

二、某图书借阅管理数据库有如下关系模式：

BOOK（BNO, BNAME, AUTHOR, AMOUNT, CATEGORY, PUBLISHER）

LIB_CARD（CNO, NAME, AGE, TEL, ADDR）

BORROW（CNO, BNO, B_DATE, R_DATE, FINE）

分别表示：

书籍表（书号，书名，作者，总数，分类，出版社名）

读者表（借书证号，姓名，年龄，电话，地址）

借阅情况表（借书证号，书号，借书日期，还书日期，罚金）

1. 请用指定的方法定义下列约束：

（1）用表约束实现：书籍表中书名、作者、出版社名三个属性构成的属性组的值不能相同。

（2）用断言实现：借阅情况表中每本图书借出的数目不能大于该图书的总数。

2. 用 SQL3 触发器完成如下操作：若还书时，借阅的时间超过了规定的天数（20 天），那么根据超出的天数按照每天 0.5 元的标准计算罚金，并将罚金存入借阅情况表。

1. 表约束

```
UNIQUE(BNAME,AUTHOR,PUBLISHER) # 利用UNIQUE实现
```

2. 断言

```
CREATE ASSERTION MAX_AMOUNT CHECK(
    NOT EXISTS (SELECT * FROM BOOK
                    WHERE AMOUNT < SELECT COUNT(*) FROM BORROW
                                    WHERE BORROW.BNO = BOOK.BNO
                                        AND R_DATE IS NULL))     # 注意要未还书
```

3. SQL3

```
CREATE TRIGGER TRIG1
AFTER UPDATE OF R_DATE ON BORROW
REFERENCING
    OLD AS OLDTUPLE
    NEW AS NEWTUPLE
WHEN (NEWTUPLE.R_DATE - NEWTUPLE.B_DATE > 20)
    UPDATE BORROW
        SET FINE = (NEWTUPLE.R_DATE-NEWTUPLE.B_DATE-20)*0.5
        WHERE CNO = NEWTUPLE.CNO AND     # 要加全
            BNO = NEWTUPLE.BNO AND
            B_DATE = NEWTUPLE.B_DATE
FOR EACH ROW;
```

第8章 数据库的管理

数据库的管理/保护：DBMS（数据库管理系统）对DB（数据库）的监控。

实现：数据库的恢复、并发控制、完整性控制、安全性控制。

DBS的最小逻辑单位：事务。

8.1 事务的概念

8.1.1 事务的定义

定义8.1 事务是构成**单一逻辑工作单元的操作集合。**DBS必须保证事务**正确，完整的执行。**

```
# 银行转账事务
T:BEGIN TRANSACTION;              /* 事务开始语句 */
  read(A);
  A := A - 50;
  write(A);
  if (A < 0): ROLLBACK;           /* 事务回退语句 */
  else {
        read(B);
        B := B + 50;
        write(B);
        COMMIT;}                  /* 事务提交语句 */
```

COMMIT语句：事务执行成功，“提交”，所有更新都写入磁盘。

ROLLBACK语句：事务执行不成功，“回退”，撤销所有更新，数据库恢复初始状态。

read(X)：把数据X从磁盘的数据库，读到内存的缓冲区。

write(X)：把数据X从内存的缓冲区，写入磁盘的数据库。

注意：write操作不一定导致数据立即写回磁盘。很可能先暂存在内存缓冲区，稍后再写回磁盘。

8.1.2 事务的ACID性质

1. **原子性** Atomicity：不可分割。要么全部执行，要么一个不做。 [事务管理子系统]
2. **一致性** Consistency：一个事务独立执行的结果，应保持数据库的一致性。 [完整性子系统 测试]
3. **隔离性** Isolation：并发时，事务串行，不受其他数据干扰。 [并发控制子系统]
4. **持久性** Durability：正确的事务对数据库的更新应该永久保留。 [恢复管理子系统]

8.2 数据库的恢复

8.2.1 恢复的定义原则和方法

1. 恢复的定义

   恢复管理子系统采取一些列的措施保证在任何情况下**保持事务的原子性和持久性**，确保**数据不丢失、不破坏。**

   定义 8.2 **数据库的可恢复性**：系统能把数据库从破坏、不正确的状态，恢复到最近一个正确状态。

2. 恢复的基本原则和实现方法

    1. 转储和建立日志

        - 周期地对整个数据库复制。
        - 建立日志数据库。

    2. 发送故障时，分情况处理

        - 数据库破坏：复制最近一次数据到数据库，利用日志“重做”。
        - 数据库完好，数据损坏：利用日志“撤销”处理。

8.2.2 故障类型和恢复方法

软故障：系统故障；硬故障：介质故障

1. 事务故障：

    1. 可预期的：程序编写ROLLBACK 语句。
    2. 不可预期的：系统进行UNDO处理。

2. 系统故障：

    1. 定义：引起系统停止运转随之要求重新启动的事件。
    2. 特点：**正在运行的事务有影响，内存只能够数据丢失，不破坏数据库。**
    3. 重启时：1）对未完成事务进程UNDO处理；2）对已提交事务但更新还在缓冲区的事务进行REDO处理。

3. 介质故障：

    1. 特点：物理损坏，磁盘数据丢失。

    2. 恢复：

        - 重装转储的副本，恢复数据库到正确状态。
        - 在日志中找出转储以后所有已提交的事务。
        - 对已提交事务REDO处理。

8.2.3 检查点机制

1. 检查点方法

DBMS定时设置检查点。在检查点时刻才真正把数据写回磁盘，并在日志文件中写入检查点信息。

当DB需要恢复时，仅需恢复检查点之后还在执行的事务。

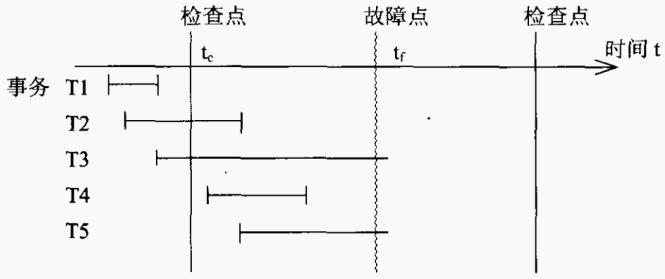


图 8.1　与检查点和系统故障有关的事务的可能状态

设 DBS 运行时，在 t_c 时刻产生了一个检查点，而在下一个检查点来临之前的 t_f 时刻系统发生故障。我们把这一阶段运行的事务分成五类（T1～T5）：

T1 : 已经完成，并且写入磁盘，无需操作。

T2 和 T4：已经完成，没有写回，需要REDO。

T3 和 T5：还未完成，需要UNDO。

2. 恢复算法
    1. 根据日志文件建立事务重做队列和事务撤销队列。
        - 重做队列：正向扫描日志，添加故障前**COMMIT的事务**。
        - 撤销队列：正向扫描日志，添加故障前**未COMMIT的事务**。
    2. 对重做队列进行REDO，对撤销队列进程UNDO。
        - REDO：正向扫描日志，根据重做队列进行REDO。
        - UNDO：**反向**扫描日志，根据撤销队列进行UNDO（逆操作）。

8.2.4 运行记录优先原则

1. 至少要等相应运行记录（日志记录）已经写入运行日志文件后，才能允许事务往数据库中写记录；**“先日志，后数据库”**
2. 直至事务的所有运行记录（日志记录）都已经写入运行日志文件后，才能允许事务完成COMMIT处理。**“写全日志，才COMMIT”**

8.2.5 SQL对事务的支持

8.3 数据库的并发控制

8.3.1 三个问题

1. 丢失更新问题
2. 读“脏”数据：”脏数据“，未提交并随后撤销的数据。
3. 不一致分析问题 / 幻行

8.3.2 封锁机制

1. 排它锁 = X锁 = 写锁
    1. **定义8.3　加X锁后，不允许再对该数据加其他任何锁。**
    2. PX协议：
        - 更新前，执行 "XFIND R"，获取X锁；
        - 未获取，一直等待，直到获取。
    3. PXC协议：
        - PX协议 + 在 COMMIT / ROLLBACK 时，解除X锁。
2. 共享锁 = S 锁 = 读锁
    1. **定义 8.4 加S锁后，仍允许再加S锁。但在S锁全解除之前，不允许添加X锁。**
    2. PS协议：
        - 更新前，执行 "SFIND R"，获取S锁；
        - 若更新，执行 "UPDX R", 升级X锁；
        - 无X锁，一直等待，直到获取。
    3. PSC协议：
        - PS协议 + 在 COMMIT / ROLLBACK 时，解除X锁。
3. 封锁的相容矩阵

8.3.3 活锁、饿死和死锁

1. 活锁问题
    1. **定义 8.5 活锁：某个事务一直处于等待状态，得不到封锁的机会。**
    2. 解决：队列，FIFO
2. 饿死问题
    1. **定义 8.6 饿死：对某数据，S锁永远存在，没有机会上X锁。**
    2. 解决：授权加锁：T2对Q加锁
        - 不存在在数据项Q上持有X锁的其他事务。
        - 不存在等待对数据项Q加锁且先于T2申请加锁的事务。
3. 死锁问题
    1. **定义 8.7 死锁：相互等待。**
    2. 解决：事务依赖图（有向图）消除环
        - 死锁测试程序：每隔一段时间检查并发的事务之间是否发生死锁。

8.3.4 并发调度的可串行化

1. 概念 （定义8.8）
    1. **调度：事务的执行次序。**
    2. **串行调度：多个事务依次执行。**
    3. **并发调度：利用分时方法，同时处理多个事务。**
2. 可串行化概念
    1. **定义 8.9 每个事务中，语句的先后顺序在各种调度中始终保持一致。**
    2. 可串行化的调度：并发调度的结果 = 某一串行调度的结果
3. 两段锁
    1. 作用：确保可串行，并发度降低。
    2. 扩展阶段（第一阶段）：获得封锁
    3. 收缩阶段（第二阶段）：释放封锁，ROLLBACK / COMMIT 阶段。
    4. **满足“两段锁” ==> 可串行**

8.3.5 SQL中事务的存取模式和隔离级别

8.4 数据库的完整性

8.4.1 完整性子系统和完整性规则

1. 概念
    1. **数据库的完整性**：数据的 **正确性、有效性、相容性**
        - 正确性：数据的合法性，e.g. 数值类型中只含数字，不含字母。
        - 有效性：数据是否属于定义的有效范围。
        - 相容性：同一事实的两个数据应相同。
    2. **完整性检查**：检查数据库中数据是否满足规定条件。
    3. **完整性约束条件**：数据库中数据应满足的条件。
    4. **完整性子系统**：DBMS中执行完整性检查的子系统。
2. 功能
    1. 监督事务执行，测试是否违法完整性规则。

2. 有违反现象，进行处理。

3. 完整性规则的组成

    1. 触发条件：什么时候检查。
    2. 约束条件/谓词：要检查什么。
    3. ELSE子句：查出错误，怎么办。

8.4.2 SQL中的完整性约束

1. 域约束

```
CREATE DOMAIN COLOR CHAR(6) DEFAULT '???'   # 若插入COLOR不再范围内，则置为???
    CONSTRAINT VALID_COLORS # 域约束名
        CHECK(VALUE IN ('RED', 'YELLOW', 'BLUE', 'GREENN', '???'))
```

2. 基本表约束

    * 都可以加 `CONSTRAINT 约束名     # 指定域约束名`

    1. 候选键的定义

    ```
    UNIQUE(<列名序列>)  # 候选键，仅表示唯一，非空需要加 NOT NULL。
    PRIMARY KEY(<列名序列>) # 主键，仅有一个，默认非空。
    ```

    2. 外键的定义

    ```
    FOREIGN KEY(<列名序列>)
        REFERENCES<参照表>[<列名序列>]
        [ON DELETE<参照动作>]
        [ON UPDATE<参照动作>]
    ```

    参照动作：NO ACTION（无影响），CASCADE（级联方式），RESTRICT（受限方式），SET NULL（置空值），SET DEFAULT（置缺省值）

    3. "检查约束"的定义

    ```
    CHECK (weight >= 80 and (color = 'red' and weight <= 200) or (color != 'red' and weight <= 400))
    ```

    Check子句只对定义它的关系起作用。在哪张表里面定义，只对该表起作用。

    例：在SPJ表中用CHECK子句定义S.SNO外键，删除S表中的SNO，并不会检查SPJ表中的CHECK语句。

    检查子句中的条件，尽可能不要涉及其他关系。

3. 断言

    1. 适用：约束与多个关系有关 / 与聚合操作有关。
    2. 触发时刻：所有修改都会触发，所有使断言为假的操作都被拒绝。

    ```
    CREATE ASSERTION <断言名> CHECK (<条件>) # 定义
    DROP ASSERTION <断言名> # 撤销，不提供RESTRICT和CASCADE
    ```

8.4.3 SQL-3 的触发器

1. 触发器结构
    1. **定义 8.10 触发器 / 主动规则 / 时间—条件—动作规则 是一个能有系统自动执行对数据库修改的语句。**
    2. 组成：ECA规则
        1. 事件：插入，删除，修改等操作。
        2. 条件：判断测试条件是否成立。
        3. 动作：满足条件，DBMS执行具体的动作。
2. SQL-3触发器实例

```
CREATE TRIGGER TRIG1    # 触发器名：TRIG1
AFTER UPDATE OF PRICE ON SPJ    # 触发事件：修改SPJ中的PRICE后激活
REFERENCING
    OLD AS OLDTUPLE # 修改前元组
    NEW AS NEWTUPLE # 修改后元组
WHEN (OLDTUPLE.PRICE > NEWTUPLE.PRICE)  # 条件部分
    UPDATE SPJ
    SET PRICE = OLDTUPLE.PRICE
    WHERE SNO = NEWTUPLE.SNO AND PNO = NEWTUPLE.PNO AND JNO = NEWTUPLE.JNO
FROM EACH ROW; # 对每一个修改的元组，都检查
```

    撤销语句：`DROP TRIGGER TRIG1`

3. 触发器的结构组成
    1. 时间关键字：AFTER, BEFORE, INSTEAD OF
    2. 触发事件： UPDATE (OF <属性表>), DELETE, INSERT
    3. 动作条件： WHEN 定义
    4. 动作体： BEGIN ATOMIC … END
    5. UPDATE： OLD AS ,NEW AS ; DELETE: OLD AS; INSERT: NEW AS
    6. 触发器种类：元祖级触发器 FROM EACH ROW，语句级触发器 FOR EACH STATEMENT(默认)

8.5 数据库的安全性

安全性 vs 完整性：前者，**非法，蓄意**；后者，**合法，无意。**

8.5.1 安全性级别

1. 环境级、职员级、OS级、网络级、DBS级

8.5.2 权限

1. 权限：用户使用数据库的方式。
2. 访问DB的权限：读 ，插入，修改，删除 权限
3. 修改DB的权限：索引，资源（创建新关系），修改，撤销 权限
4. 存储控制的类别
    1. 自主存储权限 DAC：创建者有控制权，可以通过授权传递权限。
    2. 强制存取控制 MAC：【安全性更高】
        1. 主体：活动实体（用户，进程等）； 客体（文件，表，索引等）。

2. 敏感度标记：绝密，机密，可信，公开。
    1. 主体：许可证级别；客体：密级。
    2. 读：许可证级别 >= 密级
    3. 修改：许可证级别 = 密级

8.5.3 SQL中的安全性机制

1. 视图
    1. 用户只能使用视图中定义的数据，不能使用定义之外的数据。
    2. **优点：数据安全性，逻辑独立性，操作简便性。**
2. 用户权限及其操作
    1. 用户权限：SELECT INSERT DELETE UPDATE REFERENCES USAGE（允许使用已定义的域）
    2. 授权语句：

```
GRANT <权限表> ON <数据库元素> TO <用户名表> [WITH GRANT OPTION]

# 所有权限：ALL PRIVILEGES
# 数据库元素：关系，视图，域（域名前加 DOMAIN）
# WITH GRANT OPTION: 可以传递权限
```

    3. 回收语句

```
REVOKE <权限表> ON <数据库元素> FROM <用户名表> [RESTRICT|CASCADE] # 回收权限
REVOKE GRANT OPTION FOR ... # 回收转授出去的转让权限
```

8.5.4 数据加密

8.5.5 自然环境的安全性

Chapter10 Exercise

10.6 题目

10.6 图 10.11 是有关教师（Faculty）、系（Department）和系主任（Director）信息的对象联系图。
（1）试用 ORDB 的定义语言定义这个数据库。
（2）试用 ORDB 的查询语言分别写出下列查询的 SELECT 语句：
① 检索精通俄语（Russian）的教师工号和姓名。
② 检索复旦大学出访过瑞士（Switzerland）并且精通日语（Japanese）的系主任。

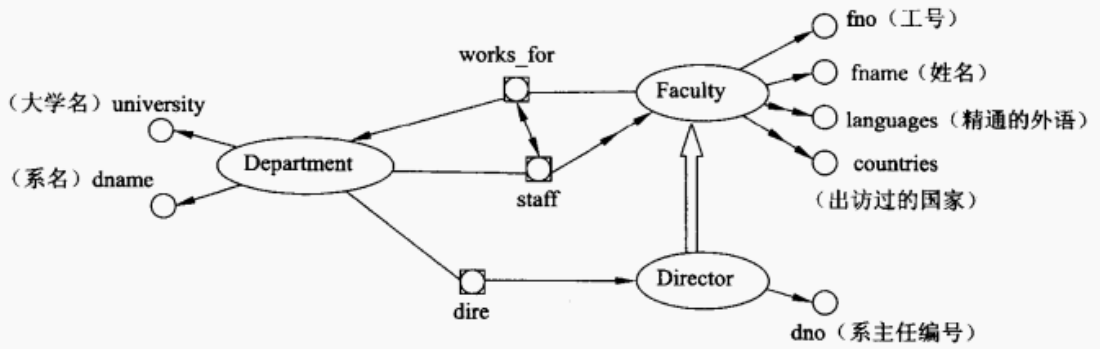


图 10.11 对象联系图

1. 试用ORDB的定义语言定义数据库。

```
CREATE TYPE MyString char varying;
/*定义 faculty类型*/
CREATE TYPE faculty(fno integer,                          /*工号*/
                    fname MyString,                       /*姓名*/
                    languages setof(language MyString),   /*精通的外语*/
                    countries setof(country MyString),    /*出访过的国家*/
                    works_for ref(department));           /*工作部门（单值，引用）
*/
/*定义 director类型*/
CREATE TYPE director (dno integer) UNDER faculty;
/*定义 department类型*/
CREATE TYPE department(university MyString,               /*大学名*/
                       dname MyString,                    /*系名*/
                       staff setof(ref(faculty)),         /*员工（多值，引用）*/
                       dire ref(director));               /*系主任（单值，引用）*/
/*定义 表*/
CREATE TABLE Faculty of TYPE faculty;
CREATE TABLE Director of TYPE director;
CREATE TABLE Department of TYPE department;
```

2. ORDB SELECT语句

    1. 检索精通俄语的教师工号和姓名。

```
SELECT F.fno, F.name
FROM Faculty as F
WHERE 'Russian' in F.languages;
```

    b. 检索复旦大学出访过瑞士并且精通日语的系主任。

```
SELECT D.dire.fname
FROM Department as D
WHERE D.university = 'Fudan University' AND
      'Switzerland' in D.dire.countries AND
      'Japanese' in D.dire.languages;
```

10.7 题目

10.7 图 10.12 是有关学生（student）和学习（study）信息的对象联系图。

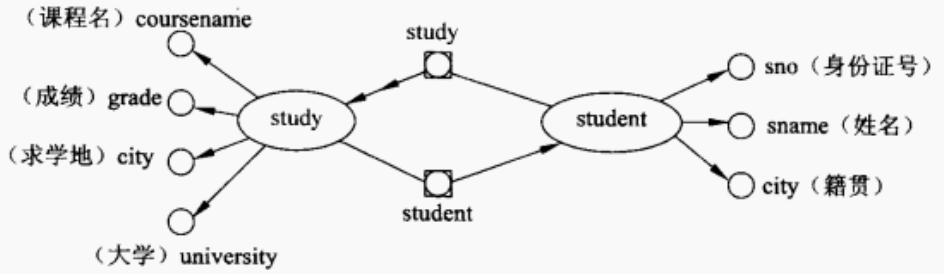


图 10.12 对象联系图

（1）试解释这个对象联系图。
（2）试用 ORDB 的定义语言定义这个数据库。
（3）试用 ORDB 的查询语言分别写出下列查询的 SELECT 语句：
① 检索每个学生的学习课程和成绩。
② 检索至少有一门课程的求学地与籍贯在同一城市的学生学号和姓名。

1. 解释对象联系图

   学生（身份证号，姓名，籍贯）

   学习（课程名，成绩，求学地，大学）

   1个学习信息对应一个学生，而1个学生可以有n个学习信息。因此，学生和学习信息是1：N的关系。

2. ORDB定义数据库

```
CREATE TYPE MyString CHAR VARYING;
CREATE TYPE t_study(coursename MyString, grade float, city MyString, university
MyString, student ref(t_student))
CREATE TYPE t_student (sno integer, sname MyString, city MyString, study
setof(t_study));
CREATE TABLE Student AS TYPE t_student;
CREATE TABLE Study AS TYPE t_study;
```

3. SELECT 语句

   1. 检索每个学生的学习课程和成绩

```
/*1NF*/
SELECT A.sname, B.coursename, B.grade
FROM student as A, A.study as B;

/*非1NF*/
SELECT A.sname, set(B.coursename, B.grade)
FROM student as A, A.study as B
GROUP BY A.sname;
```

b. 检索至少有一门课程的求学地和籍贯在同一城市的学生学号和姓名。

```
SELECT A.sno, A.sname
FROM studnet as A
WHERE EXISTS(SELECT * FROM A.study as B WHERE B.city = A.city)
```

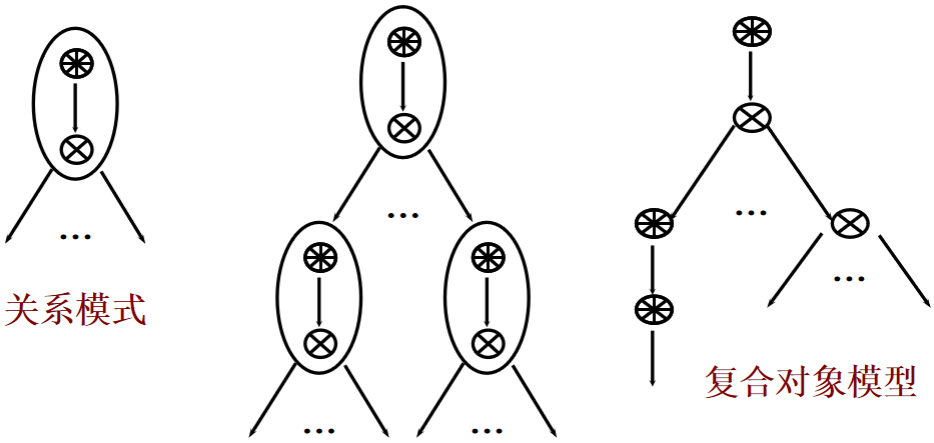
第10 章 对象关系数据库

10.1 对象联系图

10.1.1 DB技术发展过程

1. 平面关系模型：属性不可分
2. 嵌套关系模型：属性：关系/表（可多次嵌套），基本类型
3. 复合对象模型：属性：元组 / 结构，关系 / 集合，基本类型
4. 面向对象模型：继承，属性：元组，关系，基本类型

   集合用 ⊛ 表示，元组用 ⊗ 表示。



5. 定义方法

   // 定义 Dept(dno,dname,staff(empno,ename,age)) 的嵌套关系

   方法1：类型

```
type DeptRel = relation(dno:integer, dname:string,
staff:EmpRel);    # 定义车间类型
type EmpRel = relation(empno:integer, ename:string,
age:integer);    #定义职员类型
persistent var Dept:DeptRel #持久变量形式说明
```

   方法2：先元组，再类型

```
type DeptTup = tuple(dno:integer, dname:string,
staff:EmpRel);  # 定义车间元组
type DeptRel = set(DeptTup);     # 定义车间类型（元组的集合）
type EmpTup = tuple(empno:integer, ename:string,
age:integer);  # 定义职员元组
type EmpRel = set(EmpTup);
persistent var Dept:DeptRel
```

方法3：元组 + set

```
type DeptTup = tuple(dno:integer, dname:string,
staff:set(EmpTup));
type EmpTup = tuple(empno:integer, ename:string,
age:integer);
persistent var Dept:set(DeptTup);
```

10.1.2 引用类型

嵌套关系和符合对象无法表达递归结构；利用指针，指向关系。

10.1.3 对象联系图的成分

名称	符号	含义
椭圆		对象 / 实体
小圆圈	◯	基本数据类型（整型、实型、字符串型）属性
椭圆间的边		“ 引用 ”
单箭头	→	单值属性
双箭头	→→	多值属性
双线箭头	⇒	子类 ⇒ 超类
双向箭头	⟷	两属性间的逆联系

10.1.4 泛化 / 细化

1. 超类是子类的泛化，子类是超类的细化。
2. 超类型：较高层的对象；子类型：较低层的对象。具有继承性。
3. 泛化/细化联系：泛化边(双线箭头)；泛化边：子类 ⇒超类。

10.2 面向对象的类型系统

1. 基本类型：整型，浮点型，字符，字符串，布尔型，枚举型。
2. 复合类型：

类型	类型	顺序	重复性	元素个数	例子
行 / 元组 / 结构 / 对象类型	不同	有序			日期 (1,October,2007)
数组类型	相同	有序	可重复	预置	[1,2,1] ≠ [2,1,1]
列表类型	相同	有序	可重复	未预置	{1,2,1} ≠ {2,1,1}
包 / 多集类型	相同	无序	可重复	未预置	{1,2,1} = {2,1,1}
集合 / 关系类型	相同	无序	不可重复	未预置	{1,2} = {2,1}

后四行统称“聚集类型”；数据类型可以嵌套。

3. 引用类型：数据类型的定义只能嵌套，若要允许递归，就要前面提到的引用类型。

10.3 ORDB的定义语言

10.3.1 ORDB

1．对象关系数据模型：在传统的关系数据模型基础上，提供元组、数组、集合一类丰富的数据类型以及处理新的数据类型操作的能力，并且有继承性和对象标识等面向对象特点。
1．对象关系数据库系统：基于对象关系数据模型的DBS。

10.3.2 数据类型

```
CREATE TYPE DATE(day integer, month char(10), year integer);    /*
定义结构类型*/
CREATE TYPE MyString char varying;              /*定义变长字符串类型*/
CREATE TYPE NameArray MyString[10];             /*定义数组类型*/
CREATE TYPE StudentGrade multiset(integer);     /*定义包类型*/
CREATE TYPE CourseList setof(MyString);         /*定义集合类型*/

/*使用中间变量，建立表类型*/
CREATE TYPE CourseGrade (course MyString, grade integer)
CREATE TYPE StudentCourse (name MyString, cg setof(CourseGrade));
/*定义嵌套类型*/
CREATE TABLE sc of Type StudentCourse;

/*不使用中间变量，建立表类型*/
CREATE TABLE sc(name MyString, cg setof(course MyString, grade
integer));
```

10.3.3 继承性

1. 类型级

```
/*超类型*/
CREATE TYPE Person(name MyString, social_number integer);
/*子类型*/
CREATE TYPE Student(degree MyString, department MyString)
                    under Person;
CREATE TYPE Teacher(salary integer,department MyString)
                    under Person;
```

2. 表级

```
/*超表*/
CREATE TABLE People(name MyString, social_number integer);
/*子表*/
CREATE TABLE Students(degree MyString, department MyString)
                    under People;
CREATE TABLE Teachers(salary integer,department MyString)
                    under People;
```

3. 一致性要求

a. 超表 中每个元组最多可以与每个子表中的一个元组对应。
b. 子表 中每个元组在超表中恰有一个元组对应，并在继承的属性上有相同的值。

10.3.4 引用类型

利用ref(表名)表示引用类型，即地址。实现递归。若集合类型，需加setof()

```
CREATE TYPE MyString char varying;
CREATE TABLE dpept(dno integer, dname MyString, staff
setof(ref(emp)));
CREATE TABLE emp(empno integer, ename MyString, age integer,
works_for setof(ref(dept));
```

10.4 ORDB的查询语言

10.4.1 SELECT语句

1. 扩充SQL对SELECT语句的使用规定
    a. 允许用于计算关系的表达式出现在任何关系名可以出现的地方。
    b. 每个基本表设置一个元组变量，然后才可引用，在**FROM**子句中要为每个关系定义一个元组变量。

    `e.g. FROM university as U`

    c. 当属性值为单值或结构值时，属性的引用方式仍和传统的关系　模型一样，在层次之间加点“.”。

    `U.president.fname √`　**university** 的**president**属性，只有一个**faculty**元组与之对应。

    d. 当路径中某个属性值为集合时，就不能连着写下去, 必须定义元组变量。

    `U.staff.frame ×`　　**university** 的**staff**属性，有多个**faculty**元组与之对应，不能直接获取属性值。

10.4.2 嵌套与解除嵌套

“接触嵌套”：将一个嵌套关系转换成1NF的过程。

> 例： 检索使用本校教材开课的教师工号、姓名及所在学校：

```
SELECT  A.uname, B.fno, B.fname
FROM  university as A, A.staff as  B, B.teach  as  C
WHERE  C.editor.uname = A.uname;
```

uname	fno	fname
Fudan University	1357	ZHAO
Fudan University	2468	LIU
Jiaotong University	4567	WEN
Jiaotong University	5246	BAO
Jiaotong University	3719	WU

图 10.9　1NF 关系

“嵌套”：将一个1NF关系转化为嵌套关系。

```
/* 利用set 和 group by 实现嵌套*/
SELECT  A.uname, set（B.fno, B.fname） as  teachers
FROM  university  as  A, A.staff  as  B, B.teach  as  C
WHERE  C.editor.uname = A.uname
GROUP  BY  A.uname;
```

uname	teachers
	（fno，fname）
Fudan University	{（1357，ZHAO），（2468，LIU）}
Jiaotong University	{（4567，WEN），（5246，BAO），（3719，WU）}

图 10.10　非 1NF 关系（嵌套关系）

*10.4.3 函数的定义和使用

```
/*定义类型 和 表*/
CREATE TYPE StudentCourseGrade(name MyString, cg setof(course
MyString, grade integer));
CREATE TABLE sc of TYPE StudentCourseGrade

/*定义函数*/
CREATE FUNCTION course_count(one_student StudentCourseGrade) /*参数
类型:Student CourseGrade*/
RETURNS integer AS  /*返回值*/
SELECT count(B.cg)
FROM one_student as B
```

10.4.4 复合值的创建和查询

1. INSERT 复合值

```
INSERT INTO sc VALUES ('ZHANG', set(('DB',80),('OS',85)));
```

2. SELECT 复合值

```
/*检索 WANG LIU ZHANG三位学生选修的课程门数 */
SELECT A.name, count(A.cg)
FROM sc as A
WHERE A.name IN set('WANG','LIU','ZHANG')
GROUP BY A.name
```

第5章

1. 求候选键
2. 属性集闭包
3. 最小函数依赖集，函数依赖集等价
4. 分解特性，概念
5. 范式：
    1. 给出函数依赖，判断范式。
    2. 特点：全码BC范式
    3. 左边只有单属性，则必为2范式
    4. 主属性，非主属性....
6. 分解成3NF，BC，保持ρ，保持函数依赖
7. **表达性，分离性，最小冗余性**
8. 写答案注意顺序

填空总结

第7章

7.2 数据库系统的生存期阶段：**规划、需求分析、概念设计、逻辑设计、物理设计、实现和运行维护。**（7）

7.4 基于数据库系统生存期的**数据库设计**阶段：规划；需求描述和分析；概念设计；逻辑设计；物理设计。（5）

7.6 数据库设计的**需求分析阶段**：(4)

1. 分析用户活动，产生用户活动图；
2. 确定系统范围，产生系统范围图；
3. 分析用户活动所涉及的数据，产生数据流图；
4. 分析系统数据，产生数据字典。

7.10 **概念设计**的具体步骤：(3)

（1） 进行数据抽象，设计局部概念模式；

（2） 将局部概念模式综合成全局概念模式；

（3） 评审。

7.14 **逻辑设计**阶段： **形成初始模式，设计子模式，设计应用程序梗概，评价模式和修改模式。**（4）

7.15 **物理设计**步骤：**确定 DB 的存储记录结构；确定数据存储按排；存取方法的设计；完整性和安全性的设计；应用程序设计。**（5）

7.17 数据库实现阶段：建立实际 DB 结构；装入试验数据调试应用程序；装入实际数据进入试运行状态。（3）

7.18 运行维护阶段：（4）

1. 维护 DB 的安全性与完整性及系统的转储和恢复；

2. DB 性能的监督、分析与改进；
3. 增加 DB 新功能；
4. 改正运行中发现的系统错误。

第8章

1. **DB 完整性**是指数据的**正确性、有效性和相容性**，防止错误的数据进入 DB。
2. **完整性规则**由三部分组成：**触发条件，约束条件和 ELSE 子句。**
3. SQL 的完整性约束：**域约束。基本表约束和断言。**
4. 数据库有哪些级别的安全措施：**环境级、职员级、OS 级、网络级、DBS 级。**
5. SQL 的视图机制使系统具有三个优点：**数据安全性，逻辑独立性和操作简便性。**

第10章

面向对象的类型系统：**基本**数据类型、**复合**类型和**引用**类型三部分组成。

10.5 在 ORDB 中有哪些基本数据类型？有哪些复合数据类型？

答：基本数据类型有整型、浮点型、字符串型和日期型等。

复合类型有结构类型、数组类型、多集类型和集合类型等四种。