第一章　绪论

1.1 引言

1.1.1 数据、数据库、数据库系统和数据库管理系统

- 数据库技术产生于 20 世纪 60 年代中期
- 是数据管理的最新技术
- 计算机科学的重要分支
  - ⌧计算机应用的三个方面：科学计算、数据管理、过程控制
- 数据、数据库、数据库管理系统、数据库系统的关系

1) 数据（Data）
   数据是信息的符号记录。数据是数据库处理和研究的对象。
   ⌧文字、图形、图像、声音、学生档案、货物运输等。
2) 数据库（Database）
   ⌧数据保存方法：人工、文件、数据库
   长期存储在计算机内，有组织的、可共享的相关数据的集合。数据按一定的数据模型组织、描述和存贮、具有最小冗余度、较高的数据独立性、易扩展性、多用户共享等特点。
3) 数据库管理系统（DBMS）
   Database Management System。位于用户和操作系统之间的一层数据管理软件。
   ⌧数据库的建立、运行和维护由 DBMS 统一管理和统一控制。能方便用户数据定义和数据操纵、数据库保护等功能。
4) 数据库系统（DBS）
   计算机硬件为基础的记录保持系统。包括数据库、数据库管理系统、应用系统、管理员和用户，有时还包括计算机硬件。

1.1.2 数据管理技术的发展

- 数据管理是指对数据进行分类、组织、编码、存储、检索和维护，是数据处理的核心问题。分为三个阶段

1) 人工管理阶段
   ⌧特点：1)数据不保存 2)数据由人管理 3)数据不共享 4)数据无独立性
2) 文件系统阶段
   ⌧特点：1)数据长期保存 2)文件系统管理数据 3)共享性差 4）独立性低
3) 数据库系统阶段
   见 1.1.4

	阶段	人工	文件系统	数据库系统
背景	时间	20 世纪 50 年代末	60 年代中期	60 年代末
	应用背景	科学计算	科学计算、管理	大规模管理
	硬件背景	无直接存储设备	磁盘、磁鼓	大容量磁盘
	软件背景	无 OS	有文件系统	有 DBMS
	处理方式	批处理	联机和批处理	联机和批处理
特点	数据管理者	人	文件系统	DBMS
	数据面向对象	某一应用程序	某一应用程序	现实世界
	数据共享程度	无，冗余大	共享性差，冗余较大	共享性高冗余小
	数据独立性	无	独立性差(有逻辑)	有高度独立性
	数据结构化	无	记录有结构，整体无	数据模型描述
	数据控制能力	应用程序控制	应用程序控制	DBMS 保护: 安全性、完整性、并发控制、数据恢复

1.1.3 数据库系统的特点

- 数据的结构化
- 数据的共享性好、冗余度低
- 数据的独立性高：物理、逻辑
- 数据由 DBMS 统一管理
  1) 数据的安全性（Security）
  2) 数据的完整性（Integrity）
  3) 并发控制（Concuurency）
  4) 数据库恢复（Recovery）
- 良好的用户接口

1.1.4*数据库技术的研究领域

1. 数据库管理系统软件的研制（面向对象、多媒体数据库等）
2. 数据库设计（设计方法学和设计工具、数据模型与建模、设计规范与标准）
3. 数据库理论（规范化理论）

1.1.5*数据库在信息科学中的应用

- 三个世界：现实世界、信息世界、计算机世界
  - 现实世界的对象是客观存在的各类实物，每个对象称实体。实体有特征：颜色、重量等。
    实体（entity）

实体集（entity set）
特征 （feature）

- 客观事物在人脑中的反映，称信息。用记录表示实体
  实体记录（record）
  记录集（record set）
  属性（attribute）
- 计算机世界只能管理由字母、符号构成的数字化信息。数字化后的信息称数据。数据是信息的符号化。
  数据（Data）：元组(Tuple)，
  数据集（dataset）：文件/表(File/Table)
  数据项：字段（Field）

1.2 数据模型

1.2.1 概述

- 数据的组织是数据库技术的核心问题
- 数据库的数据组织是通过数据模型来实现的
- 数据模型是创建数据库维护数据库的方式，是数据库系统定义数据内容和数据间联系的方法。
- 数据模型的定义：表示实体类型和实体间联系的模型称数据模型，是对现实世界的抽象。
- 数据模型分两个层次：
  - **概念（数据）模型**：也称信息模型。用来描述信息结构，又称**实体联系模型（ER）;**按照用户观点对信息建模。
  - **（结构）数据模型**：面向数据库的逻辑结构，直接涉及到计算机系统和 DBMS，又称为**逻辑数据模型**，简称数据模型；按照计算机系统的观点对数据建模。

- 数据模型的三个方面要求：
  1) 比较真实模拟真实世界
  2) 容易为人所理解
  3) 便于计算机实现

1.2.2 概念模型

现实世界－(认识抽象)－>信息世界，概念模型－(转换)－>机器世界，数据模型

1. 信息世界涉及的基本概念

- 实体（entity）
- 属性（attribute）
- 码（key）

- 域（domain）
- 实体型（entity type）
- 实体集（entity set）
- 联系（relationship）：一对一(1:1) 一对多(1:n) 多对多(m:n)

2. 概念模型的表示方法－－实体联系模型

- 实体联系模型（Entity Relationship Model，简记 ER 模型）：是 P.P.S Chen 1976 年提出的。直接从现实世界中抽象出实体和实体间联系，然后用实体联系图（ER 图）表示信息模型，这种方法称 ER 方法；ER 模型实际是信息世界的模型。
- ER 图的四个组成部分：
  - 矩形框：实体型
  - 菱形框：联系
  - 椭圆框：实体型和联系的属性
  - 直线：连接实体类型和联系类型，并表示联系的种类
- 举例说明。

1.2.3（结构）数据模型

- **数据模型三要素：**

1. 数据结构：
   a) 用于描述系统的静态特征。
   b) 是对实体类型和实体间联系的表达和实现，数据类型的命名依据。
2. 数据操作：
   a) 是用于描述系统的动态特征。
   b) 是对数据库检索和更新（插入、修改、删除）两类操作
3. 数据的约束条件：
   a) 一组完整性规则的集合。
   b) 给出数据及其联系所具有的制约和依赖原则。

- **结构**数据模型主要有三种：层次模型、网状模型、关系模型，未来的发展有：对象模型、语义模型；
- 非关系模型中的数据结构的单位是基本层次联系。所谓基本层次联系是指的两个记录以及它们之间的一对多（包括一对多）的联系。

1.2.3.1 层次模型

- 典型代表：IBM1968 年的 IMS（Information Management System）
- 层次模型定义：用树型（层次）结构表示实体类型及实体间联系的数据模型

1. 层次模型的数据结构特点：

- 有且仅有一个结点无双亲，该结点是根结点
- 其他结点有且仅有一个双亲。

- 层次模型中每个结点（片断）表示一个记录类型
- 每个记录类型包含若干个字段，字段有字段名、数据类型和长度等
- IMS 中，一个根片断型及其所有从属片断型组成的的一个层次模型称为一个物理数据库记录型 PDBRT（Physical DataBase Record Type）。
- IMS 中，根片断的一个值及其所有从属片断值的整体组成一个物理数据库的记录值，称为物理数据库记录。记 PDBR（Physical DataBase Record）。

2. 层次模型的数据操纵与完整性约束：

- 插入时，没有双亲结点不能插入子女结点；
- 删除时，删除双亲结点要同时删除子女结点；
- 更新时，要更新所有相应的记录；

3. 层次模型的存储结构：

- 邻接法
- 链接法

4. 层次模型的优缺点：

- 优点：
  1) 模型本身简单
  2) 实体间联系是固定的，预先设计好的应用系统，性能优于关系型，不低于网状模型
  3) 层次模型提供了良好的完整性支持。
- 缺点：
  1) 实现多对多关系时，需要采用冗余或虚结点方式，易形成不一致性
  2) 对插入和删除操作限制较多。
  3) 查询子结点需通过双亲结点。
  4) 结构严谨，层次命令趋于程序化。

1.2.3.2 网状模型

- 典型代表：
  - 20 世纪 70 年代 CODASYL（Conference OnData System Language）的 DBTG（DataBase Task Group 数据库任务组）开发系统；
  - HP 公司的 IMAGE/3000；
  - Honeywll 公司的 IDS/II；
  - Burroughs 公司的 DMSII
  - Univac 公司的 DMS1100
  - CwllinetI 公司的 DMS
  - CINCOM 公司的 TOTAL
- 定义：用有向图（网络）结构表示实体类型及实体间联系的数据模型。（network model）

1. 网状数据模型的数据结构特点：

- 允许一个以上的结点无双亲
- 至少一个结点不止一个双亲

2. 网状数据模型的数据操纵与完整性约束：

- 插入时，允许插入未确定双亲结点的子女结点值；

- 删除时，允许只删除双亲结点值；
- 更新时，只需更新指定记录；

3. 网状模型的存储结构：

- 常用链接法
- 其它如引元阵列法、二进制阵列法、索引法等

4. 网状模型的优缺点：

- 优点：
  1) 更加直接描述现实世界
  2) 存取效率高，性能好
- 缺点：
  1) 结构复杂，不利于用户掌握，DDL、DML 语言复杂
  2) 数据独立性差

1.2.3.3 关系模型

- 典型代表：1970 年，美国 IBM 公司 E.F.Colld 提出关系模型，目前有数据库：DB2、Oracle、Sybase、Infomix、SQL Server、Foxpro，Access 等。
- 定义：（Relation model）用表格表示实体集和实体间联系，用外键来实现关系间的联系。

1. 关系数据模型的数据结构特征：

- 在用户看来，一个关系模型的逻辑结构就是一张二维表；
- 必须使用规范化的关系，如分量是原子的；
- 不仅实体用关系表示，实体间的联系也用关系表示。

- 术语：
  - 关系（relation）：一个关系对应通常一张表 记做 R
  - 元组（Tuple）：表中的一行 （V_1，V_2……V_n）
  - 属性（Attribute）/字段（field）：表中的一列（A1，A2……An）
  - 域（Domain）：属性的取值范围，如性别域 DOM（性别）＝（男、女）
  - 分量：元组中的一个属性
  - 关系模式：是记录型。对关系的描述，关系名（属性 1，属性 2……属性 n）记 R（A1，A2，… …An）；关系模式的实例是一个关系。关系实际就是一张表。关系模式的描述包括：关系名、属性名、属性类型、属性长度、主键包含的属性等，记做：R（U，D，DOM，F）
  - 关系数据库模式：一组关系模式的总称即关系数据库模式，简称模式。
  - 候选键（Condidate Key）：在给定的关系中，有这样的属性或最小属性组，它在不同的元组中的值是不同的，利用这个（些）值可以唯一地标示关系中地元组，则称该属性（组）为候选键。
  - 主码（键）（Primary key）：候选键可能不止一个，被指定正在使用地候选键称主键。
  - 超键（Super Key）：能唯一标识元组的属性集。所有候选键都是超键。不是最小属性集，可以有多余属性。
  - 组合键与全键：当主键不是单一的属性时称组合键，组合键包括所有属性时称全键。

- 外部键（Foreign Key）：关系模式 R 中的属性集是其它关系模式的主键，那么该属性集对关系模式 R 而言是外键。
- 主属性和非主属性：包含在任一候选键中的属性叫主属性，否则称非主属性。

2. 关系数据模型的操纵与完整性约束：

- 插入、删除和更新操作必须满足关系的完整性约束；
- 关系的完整性约束包括：实体完整性、参照完整性和自定义完整性

3. 关系模型的存储结构：

- 实体和实体间的联系都用表来表示

4. 关系数据模型的优缺点：

- 优点：
  1) 建立在严格的数学概念基础上
  2) 概念单一，实体、联系均用关系来表示
  3) 存取路径对用户透明，数据独立性更高，保密性更好。简化程序员工作和数据库开发建立工作。
- 缺点：
  1) 存取路径对用户透明，查询效率不高
  2) 因存取路径对用户透明，必须对用户查询进行优化，增加了开发 DBMS 的难度。

1.3 数据库系统的结构及功能

1.3.1 数据库系统的模式结构(从管理系统角度看)

- 型（type）与值（value）：型是数据结构和属性的说明，值是型的具体赋值。
- 1975,美国国家标准学会（ANSI）的标准计划与要求委员会（SPARC:Standard Plan And Request Committee）提出将数据库分三级，通常称 SPARC 分级结构。

1. SPARC 接口分三级结构，两层映象

1) 内部级：内部视图、存储级、物理级、内模式、存储模式
   所有存储在计算机外存上的数据文件，一个数据库只有一个内模式。
2) 概念级：概念视图、全局逻辑级、模式、概念模式
   用以描述数据库的整体逻辑结构，一个数据库只有一个模式。
3) 外部级：用户视图、局部逻辑级、外模式、子模式。
   是数据库与用户的接口，是用户局部逻辑的描述，一个数据库可以有多个。

	schema	**view**	**level**
External	外模式、子模式	用户视图、外视图	外部级、局部逻辑级
conceptual	模式、概念模式	概念视图、DBA 视图	概念级、全局逻辑级
internal	内模式、存储模式	内部视图	内部级、物理级、存储级

2. 数据库的二级映象功能与数据独立性

- 概念模式/内模式的映象
- 概念模式/外模式的映象
- 物理独立性：当存储结构改变时，可以通过修改概念模式/内模式的映象使概念级保持不便，这样外部级和应用程序也不会改变。

- 逻辑独立性：当概念级发生改变时，可以通过修改概念模式/外模式的映象使外部级尽量保持不变，应用也就不需改变。

1.3.2 数据库系统的体系结构（从最终用户角度来看）：

- 数据库应用的三个层次：存储层、业务逻辑层、界面表示层
- 体系结构：
  1. 单用户 DBS
  2. 主从式结构 DBS（终端方式）
  3. 客户/服务器结构 DBS 、BS 机构 DBS
  4. 分布式结构 DBS

1.3.3*数据库系统的组成

- 组成：
  - 计算机硬件　CPU、内存、硬盘
  - 计算机软件　OS、DBMS、应用软件包、应用程序
  - 数据（库）
  - 管理员、用户

1.4 数据库管理系统

1.4.1 数据库管理系统的功能与组成

- DBMS 功能
  1) 数据定义 （DDL 实现）
  2) 数据操纵 （DML 和 DCL 实现）
  3) 数据库运行管理
     - 安全性检查
     - 完整性检查
     - 并发控制
     - 索引、数据字典等内部维护
  4) 数据组织、存储、管理
     - 如数据字典、用户数据、存取路径等的组织、存储管理
  5) 数据库建立与维护功能
     - 数据库数据的输入、数据转换
     - 数据库的转储与恢复
     - 数据库的重组与重构
     - 性能监视与分析
  6) 数据通信接口功能
- DBMS 组成
  1) 数据定义语言(DDL Data Definition/Description Language)

2) 数据操纵语言(DML Data Manipulation Language)
3) 数据库运行控制语言(DCL Data Control Language)
4) 实用程序

1.4.2 数据库管理系统的工作过程

- 用户存取数据库数据的过程；
  ① 用户在程序中嵌入 DML 的一个读记录语句。控制转向 DBMS；
  ② DBMS 检查合法性，查找子模式表，确定对应的存取权限；
  ③ DBMS 依据子模式/模式映象的定义，确定应读入的模式记录；
  ④ DBMS 依据模式/内模式映象的定义，确定应读入的物理记录；
  ⑤ DBMS 向 OS 发送读取所需物理记录的命令；
  ⑥ OS 启动 IO 程序，执行读操作；
  ⑦ OS 将数据从数据库的存储区送到系统缓冲区；
  ⑧ DBMS 依据子模式/模式映象定义，导出用户所要读取的记录格式；
  ⑨ DBMS 将数据记录从系统缓冲区传送到程序的用户工作区；
  ⑩ DBMS 向应用程序返回命令执行情况和状态信息。

1.4.3 数据库管理系统的实现方法

1) N 方案
   ⌧DBMS 模块加入到用户进程，DBMS 代码出现多副本，性能大幅下降。
2) 2N 方案
   ⌧每个用户有一个 DBMSshdow 进程和一个后台负责读写和日值的进程。
3) M＋N 方案
   ⌧2N 方案的改进方案，N 用户，M 个 DBMS 进程（M<N）
4) N＋1 方案
   ⌧1 个 DBMS 进程(可设计成多线程的)，N 个用户进程，消息开销大。

1.5 数据库工程与应用

1.5.1 数据库设计的目标和特点

- 数据库设计的任务是：在 DBMS 的支持下，按照应用的要求，为某一部门或组织设计一个结构合理、使用方便、效率较高的数据库及其应用系统。
- 数据库设计包含两方面内容：结构（数据）设计、行为（处理）设计

1.5.2 数据库设计方法

- 核心是：逻辑设计和物理设计
- 著名设计方法：

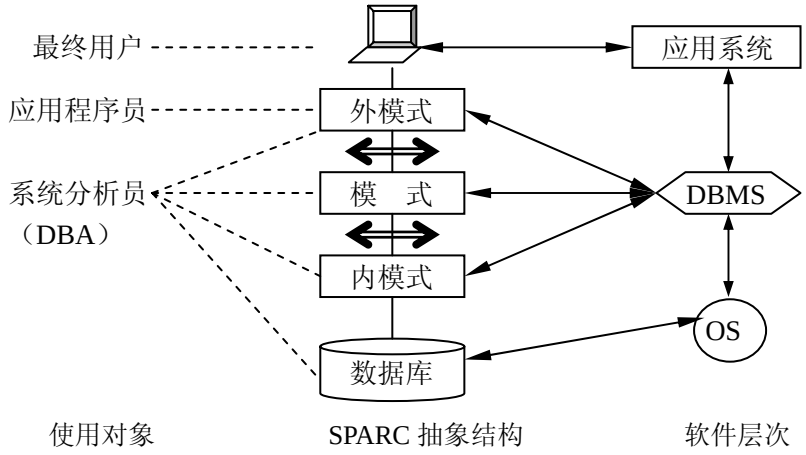
- 新奥尔良设计：需求分析、概念设计、逻辑设计、物理设计
- S.B.Yao 设计：需求分析、模式构成、模式汇总、模式重构、模式分析、物理设计

1.5.3 数据库设计步骤

- 步骤：需求分析、概念结构、逻辑结构、物理结构、实施、运行维护 (P33 图 1-36)
- 设计注意的问题：
  - 让用户参与，调动用户积极性
  - 充分考虑系统的可扩充性（可扩充性有限度）
  - 设计新系统要考虑旧系统的数据平稳迁移到新系统

1.5.4 数据库应用

- 各种用户的数据视图：
  - 最终用户：应用系统
  - 程序员：DBMS 的外模式
  - DBA/系统分析员：DBMS 的模式、内模式、数据库



- DBA 的职责：
  - 设计与定义数据库
  - 帮助最终用户使用数据库
  - 监督和控制数据库运行
  - 改进和重组数据库
  - 转储和恢复数据库
  - 重构数据库

第二章　关系数据库

2.1 关系数据库概述

- 关系数据库系统是支持关系模型的数据库系统
- 关系理论是建立在集合代数理论基础上的，关系的定义和各种操作运算可以用集合代数给出
- 关系模型的三要素：
  - 关系数据结构：二维表
  - 关系操作：选择、投影、连接、除、并，交、差等查询以及增、删、改。
    关系数据语言：
    - 关系代数语言　　ISBL
    - 关系演算语言
      - 元组关系演算语言　　ALPHA，QUEL
      - 域关系演算语言　　QBE
    - 具有关系代数和关系演算双重特点的语言　　SQL
  - 完整性约束：
    1) 域完整性：
    2) 实体完整性：
    3) 参照完整性：
    4) 自定义完整性：

2.2 关系数据结构

2.2.1 关系

1）域：域是一组具有相同数据类型的值的集合。值的个数称为域的基数。如：性别集＝{男、女}。基数＝2
月份集＝{1，2，3，4，5，6，7，8，9，10，11，12}，基数＝12

2）笛卡儿乘积
- 定义：给定一组域：D1，D2，……Dn，域可以相同，定义 D1D2……Dn 的笛卡儿乘积为：D1×D2×……×Dn＝{(d1，d2，……dn)|di∈Di，i＝1，2，……n}
  其中（d1，d2，……dn）称为一个元组。元组中的每个值 di 称为一个分量。
  若 Di（i＝1，2，……n）为有限集，其基数为 mi（i＝1，2，……n），则：笛卡儿乘积 D1×D2×……×Dn 的基数为 m＝∏mi
- 例：
  D1＝姓名集合＝{赵一平，钱峰，孙英}
  D2＝性别集合＝{男，女}
  D3＝年龄集合＝{16，17，18}
  则 m＝3×2×3＝18

3）关系(Relation)：

- 定义：笛卡儿乘积 D1×D2×……×Dn 的任一子集 D’，称作 D1，D2，……Dn 上的关系。用 R（D1，D2……Dn）来表示。
  其中 R 是关系名，n 是关系的度或目，D’中的每个元素（d1，d2，……dn）是关系的一个元组。
  n＝1 是单元关系，一元关系；n＝2 是二元关系
- 实际应用中关系往往是笛卡儿乘积中有意义的子集构成。
- 候选码（candidade key）：关系中的某一属性组的值能唯一地标识一个元组，而其真子集不可以，则称该属性组为候选码。
- 主码（primary key）：多个候选码中，可选定其中一个为主码。
- 主属性（prime attribute）：候选码的各个属性都称为主属性。
- 非主属性（none-key attribute）：不包含在任何候选码中的属性。
- 全码（all-key）：关系模式的所有属性构成这个关系模式的候选码，称为全码。

2.2.2 关系模式

- 定义：关系的描述称为**关系模式（Relation schema）,一般表示为 R(U,D,DOM,F)**
  其中，R 是关系名，U 是组成该关系的属性集合，D 为属性组 U 中属性所来自的域，DOM 是属性向域的映象集合，F 是属性间数据的依赖关系集合。

2.2.3 关系数据库

- 在一个给定的现实世界领域里,所有实体及实体间的联系的关系所构成的集合是一个关系数据库;
- 关系数据库有型和值之分：关系数据库的型也称关系数据库模式，是对关系数据库的描述它包括若干域的定义以及在这些域上定义的若干关系模式;关系数据库的值也称为关系数据库，是这些关系模式在某一时刻对应的关系的集合;
- 关系数据库的值与关系数据库模式通称为关系数据库。

2.3 关系的完整性

1. **域完整性**
   - 关系中属性的值应是对应域中的值，并由语意决定其是否为空值（NULL）。
2. **实体完整性**
   - 若属性 A 是基本关系 R 的主属性，则 A 不能取空值
3. **参照完整性**
   - 外码（foreign key）：设 F 是基本关系 R 的一个或一组属性，但不是 R 的码（主码或候选码），如果 F 与基本关系 S 的主码 Ks 相对应，则称 F 是基本关系 R 的外码。成 R 是参照关系（referencing relation），S 是被参照关系（referenced relation）。
   - 若属性（或属性组）F 是基本关系 R 的外码，它与基本关系 S 的主码 Ks 相对应（关系 R、S 不一定是不同的关系），则对于 R 中的每一个元组在 F 上的取值必须：
     1) 取空值（F 的每个属性值均取空值） 2) 等于 S 中某个元组的主码值

4. 自定义完整性

2.4 关系代数

- 关系代数由一组关系运算组成，是对于关系的操作集。关系运算以一个或多个关系作为操作的对象，运算结果是一个新的关系。用关系运算实现查询。
- 关系代数运算符
  - 传统集合运算符：∪（并）－（差）∩（交）×（笛卡儿积）
  - 专门运算符：σ 选择 Π 投影 ∞ 连接 ÷ 除
  - 比较运算符： > ≥ < ≤ = ≠
  - 逻辑运算符：¬ 非 ∧与 ∨或
- 常用的关系运算有：交、并、差、笛卡儿积、投影、选择、连接、除
- 基本关系运算有：并、差、笛卡儿积、投影、选择，其他的关系运算可以通过基本运算实现
- 同类关系：具有相同的度，且两个关系每个属性取值于同一个域。
- 为了表达的方便，引入以下记号：

1) 设关系模式 R（A1，A2，……An），它的一个关系为 Rt，t∈Rt 表示 t 是 Rt 的一个元组。t[Ai]则表示元组 t 中相应于 Ai 的一个分量
2) 若 A＝{A_{i1}，A_{i2}，……A_{ik}}是 A1，A2，……An 的一部分，k<=n，则 A 称为属性列（组）或域列。t[A]=（t[A_{i1}]，t[A_{i2}]，……t[A_{ik}]）表示元组在属性列 A 上诸分量的集合。
3) R 为 n 元关系，S 为 m 元关系。tr∈R，ts∈S，tr ⌒ ts 称为元组的连接（Concatenation）。它是一个 m＋n 列的元组，前 n 个分量为 R 中的一个 n 元组，后 m 个分量为 S 中的一个 m 元组。
4) 给定一个关系 R（X，Z），X 和 Z 为属性组，定义：当 t[X]=x 时，x 在 R 中的象集（image set）为：
   Zx＝{t[Z]|t∈R，t[X]=x}，表示 R 中属性组 X 上值为 x 的诸元组在 Z 属性组上的分量的集合。

2.4.1 传统的集合运算

- 假设：

R

Name	Sex	Age
Zhang	F	22
Wang	M	25
Lu	M	37
Chen	F	27

S

Name	Sex	Age
Zhang	F	22
Wang	M	25
Lu	F	30

Sun	M	28

- 并（Union）：同类关系 R 和 S 的并记为 R∪S，或 R union S
  定义：R∪S={t|t∈R ∨ t∈S}注意去除重复元组。
  R∪S

Name	Sex	Age
Zhang	F	22
Wang	M	25
Lu	M	37
Chen	F	27
Lu	F	30
Sun	M	28

- 交（Intersection）：同类关系 R 和 S 的交记为 R∩S，或 R intersect S
  定义：R∩S＝{ t|t∈R ∧ t∈S } ≡R－（R－S）≡S－（S－R）
  R∩S

Name	Sex	Age
Zhang	F	22
Wang	M	25

- 差（Minus/Difference）：同类关系 R 和 S 的差记为 R－S 或 R minus S
  定义：R－S＝{ t|t∈R ∧ t∉S }
  R－S

Name	Sex	Age
Lu	M	37
Chen	F	27

- 笛卡儿积（Cartesian Product）：关系 R 和 S 的笛卡儿积记为 R×S，
  定义：R×S＝{ t⌢s|t∈R， s∈S }
  R:

CNo	CN
C-11	OS
C-21	DB

S:

SNo	SN	Age
S-01	Huang	21
S-21	Lin	20
S-30	Shao	22

R×S:

CNo	CN	SNo	SN	Age
C-11	OS	S-01	Huang	21
C-11	OS	S-21	Lin	20
C-11	OS	S-30	Shao	22
C-21	DB	S-01	Huang	21

C-21	DB	S-21	Lin	20
C-21	DB	S-30	Shao	22

2.4.2 专门的关系运算

- 投影（Projection）：

  关系 R 上的投影是从 R 中选择出若干属性，并且去掉重复元组组成一个新关系，属于单目运算。记作：

  $\prod_A(R)=\{t[A]|\ t\in R\}$ A 为 R 中的属性列

  例：

  Student

SNo	SName	Sex	Age
S01	Wang	F	17
S02	Zhang	M	20
S03	Lin	M	18
S04	Sun	F	19

  $Sna=\prod_{Sname,\ Age}(Student)$

SName	Age
Wang	17
Zhang	20
Lin	18
Sun	19

- 选择（Selection）：

  又称限制（Restriction），在给定的关系 R 中，抽出满足条件的元组，组成一个新关系，新关系与原关系同类，是原关系一个子集，记做：

  $\delta_F(R)=\{t|\ t\in R\ \wedge\ F(t)=$’真’$\}$ F 表示条件

  例：

  $Sa_{18}=\delta_{age>=18}$（Student）

SNo	SName	Sex	Age
S02	Zhang	M	20
S03	Lin	M	18
S04	Sun	F	19

- 连接（Jion）：

  从两个关系的笛卡儿乘积中选取属性满足一定条件的元组，组成新的关系，记做：

  $R\infty S=\{tr\frown ts|\ tr\in R\wedge ts\in S\wedge\ tr[A]\theta ts[B]\}\equiv\delta_{i\theta(r+j)}(R\times S)$

  AθB

  AθB 表示 R 上的属性 A 和 S 上的属性 B 满足θ条件，θ是比较运算符，A、B 的度数相等且可比。这里假设 AB 分别在 RS 关系的第 ij 列，R 度为 r。

  例：

  R

A	C	D
30	C1	D3
40	C2	D3
50	C3	D1
10	C4	D1

S

B	E	F
20	E1	F1
50	E2	F3
40	E3	F1

R ∞ S

A>B

A	B	C	D	E	F
30	20	C1	D3	E1	F1
40	20	C2	D3	E1	F1
50	20	C3	D1	E1	F1
50	40	C3	D1	E3	F1

- 等值连接（equi－jion）：θ为“＝”时称为等值连接。

  R ∞ S ＝{tr⌒ts| tr∈R ∧ts∈S ∧ tr[A]＝ts[B]}

  AθB
- 自然连接（Natioal Jion）：两个关系中具有相同的属性，并且在相同的属性上做等值连接。自然连接需要取消重复列，而等值连接不需要。

  R ∞ S ＝{tr⌒ts| tr∈R ∧ts∈S ∧ tr[A]＝ts[A]}

➢ 除法（Division）：

给定关系 R（X，Y）和 S（Y，Z），其中 X，Y，Z 为属性组，R 中的 Y 与 S 中的 Y 可以不同属性名，但必须有相同的域。记 R÷S。令 P（X）＝R÷S，则 P 是 R 中满足以下条件的元组在 X 属性列上的投影：元组在 X 上的分量值 x 的象集 Yx 包含 S 在 Y 上投影的集合，记做：

R÷S＝{tr[X]|tr∈R ∧ Yx⊇$\prod_Y$（S）}

R÷S ≡ $\prod_{1,2...r-s}$（R）－$\prod_{1,2...r-s}$（（$\prod_{1,2...r-s}$（R）×S）－R）

例：

R

A	B	C
a1	b1	c2
a2	b3	c7
a3	b4	c6
a1	b2	c3
a4	b6	c6
a2	b2	c3
a1	b2	c1

S

B	C	D
b1	c2	d1

b2	c1	d1
b2	c3	d2

R÷S：

A
a1

象集：$Y_{x=a1}$={(b1,c2),(b2,c3),(b2,c1)}

$Y_{x=a2}$={(b3,c7),(b2,c3)}

$Y_{x=a3}$={(b4,c6)}

$Y_{x=a4}$={(b6,c6)}

- 外连接（Outer Join）：

  如果 R 和 S 在做自然连接时，把该舍弃的元组也保存在新关系中，在新增加的属性上填空值（null），这种操作称为“外连接”。如果把 R 中该舍弃的元组保留在新关系中称左连接；把 S 中该舍弃的元组保留在新关系中称右连接。
- 外部并（Outer Union）：

  若关系 R 和 S 不同类，则新关系的属性由 R 和 S 的属性组成，公共属性只取一次，新关系的元组由属于 R 或 S 的元组构成，新增的属性上均填空（null）。
- 半连接（Semijoin）：

  关系 R 和 S 的半连接定义为 R 和 S 的自然连接在关系 R 的属性集上的投影。

  例：

R

A	B	C
a	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b
e	f	g

R Outer Join S

A	B	C	D
a	b	c	d
a	b	c	e
c	a	d	b
b	b	f	null
null	e	f	g

R left Outer Join S

A	B	C	D
a	b	c	d
a	b	c	e
c	a	d	b
b	b	f	null

R right Outer Join S

A	B	C	D
a	b	c	d
a	b	c	e
c	a	d	b
null	e	f	g

R Outer Union S

A	B	C	D
a	b	c	null
b	b	f	null
c	a	d	null
null	b	c	d
null	b	c	e
null	a	d	b
null	e	f	g

R Semijoin S $\equiv \prod_R$（R ∞ S）

A	B	C
a	b	c
a	b	c
c	a	d

S Semijoin R $\equiv \prod_S$（R ∞ S）

B	C	D
b	c	d
b	c	e
a	d	b

2.4.3*关系代数运算应用举例

- S（S#，SN，SSEX，SAGE）
  C（C#，CN，TEACHER）
  SC（S#，C#，GRADE）
- 检索学习课程号为 C2 的学生学号与成绩。

  $\prod_{S\#,\ GRADE}$（$\delta_{C\#='C2'}$（SC））或$\prod_{1,\ 3}$（$\delta_{2='C2'}$（SC））
- 检索学习课程号为 C2 的学生学号与姓名。

  $\prod_{S\#,\ SN}$（$\delta_{C\#='C2'}$（S∞SC））
- 检索选修课程名为 Maths 的学生学号与姓名。

  $\prod_{S\#,\ SN}$（$\delta_{CN='Maths'}$（S∞SC∞C））
- 检索选修课程为 C2 或 C4 的学生学号

$\Pi_{S\#}(\delta_{C\#='C2' \vee C\#='C4'}(SC))$

- 检索至少选修课程为 C2 和 C4 的学生学号

  $\Pi_{S\#}(\delta_{1=4 \wedge 2='C2' \wedge 5='C4'}(SC \times SC)$

  或：$\delta_{S\#,C\#}(SC) \div \delta_{C\#='C2' \vee C\#='C4'}(C)$

- 检索不选修 C2 课程的学生姓名与年龄。

  $\Pi_{SN,\ SAGE}(S) - \Pi_{SN,\ SAGE}(\delta_{C\#='C2'}(SC \infty S))$

- 检索选修全部课程的学生姓名。

  $\Pi_{SN}(S \infty (\Pi_{S\#,\ C\#}(SC) \div \Pi_{C\#}(C)))$

- 检索所学课程包含学生 S3 所学课程的学生学号。

  $\Pi_{S\#,\ C\#}(SC) \div \Pi_{C\#}(\delta_{S\#='S3'}(SC))$

2.5 关系演算**

2.5.1 元组关系演算语言 ALPHA

2.5.2 域关系演算语言 QBE

2.6 关系数据库管理系统

- 关系数据库管理系统简称关系系统
- 一个数据库管理系统可成为关系系统的最小条件
  - 关系数据库（即关系数据结构）
  - 支持选择、投影和连接运算
- E.F.Codd 思想对关系系统的分类：（P63 图 2-5）
  - 表式系统：仅只是关系数据结构，不支持集合级操作
  - 最小关系系统：支持关系结构和选择、投影、连接集合操作
  - 关系完备系统：支持关系结构和所有关系代数操作
  - 全关系系统：支持关系模型的所有特征。

第三章　关系数据库标准查询语言 SQL

3.1 SQL 概述

- SQL(Standard/Structured Query Language)是关系数据库标准。
- 1974 年 Boyce 和 Chamberlin 提出。
- 1986 年 10 月，美国国家标准局（American National Standard Institute ANSI）公布第一个标准 ANSI X3.135-1986，国际标准化组织（International Organization for Standardization ISO）也通过这一标准称 SQL－86。
- 1989 年 ANSI 再次公布标准 ANSI X3.135-1989 ，ISO 相应 SQL－89
- 1992 年 ANSI 再次公布标准 ANSI X3.135-1992 ，ISO 相应 SQL－92 (SQL 2)
- 1999 年，ISO 公布 SQL-1999（SQL99，SQL3）
- 2003 年，ISO 公布 SQL-2003

3.1.1 SQL 语言的组成

- 数据定义（DDL Data Definition/Description Language ）：
  - 定义数据库的逻辑结构，包括基本表、视图、索引等。
- 数据操纵（DML Data Manipulation Language）：
  - 包括查询和更新，更新又包含插入、删除和修改。
- 数据控制（DCL Data Control Language ）：
  - 授权、完整性规则描述、事务控制等。
- 嵌入式 SQL（ESQL）：在宿主语言中使用 SQL 的规则。

3.1.2 SQL 语言的特点

- 综合统一：集 DDL、DML、DCL 于一体，语言风格统一
- 高度非过程化：只需要提出目标，操作过程由 DBMS 完成
- 面向集合的操作方式：操作对象、查询结果都可以是元组的集合
- 以统一的语法结构提供两种使用方式：自含式、嵌入式
- 语言简洁，易学易用，共使用 11 个关键词：
  - DDL：create drop alter
  - DML：select insert delete update
  - DCL：grant revoke commit rollback

3.2 数据定义语言（DDL）

3.2.1 定义、删除与修改基本表

- 定义基本表语法：

  CREATE TABLE <表名> （<列名><数据类型>[列级约束条件]
  [,<列名><数据类型>[列级约束条件]... ...]
  [,<表级完整性约束条件>]);

  - CREATE TABLE S（
    S＃ CHAR（5） NOT NULL UNIQUE，
    SN CHAR（20），
    SA INT，
    SD CHAR（3），
    PRIMARY KEY (S#)
    ）；

- 修改表语法：

  ALTER TABLE <表名>
  [ADD <新列名><数据类型>[列级约束条件]]
  [DROP <完整性约束条件>]
  [MODIFY <列名><数据类型>];

  - 在表 S 中增加入学日期
    ALTER TABLE S ADD SCome DATE；
  - ALTER TABLE S MODIFY SA SMALLINT；
  - ALTER TABLE S DROP UNIQUE(S#)；
- 删除表语法：

  DROP TABLE <表名> ；

  - DROP TABLE S；

3.2.2 建立和删除索引

- 索引的建立：
  - 语法：
    CREATE [UNIQUE][CLUSTER] INDEX <索引名>
    ON <表名>（<列名 1>[<次序>][，<列名 2><次序>... ...]）
    <次序>可以是 ACS 和 DESC
  - CREATE UNIQUE INDEX S_S# ON S(S#)
    CREATE UNIQUE INDEX C_C# ON C(C#)
    CREATE UNIQUE INDEX SC_S#_C# ON SC(S# ASC,C# DESC)
- 索引的删除：
  - 语法：
    DROP INDEX [<表名>.]<索引名>

- DROP INDEX [S.]S_S#

3.3 SQL 的数据查询（DML）

- 关系代数表达式：

  $\Pi_{A1,A2,.....An}(\delta_F(R1\times R2\times ... ...\times Rn))$

- SQL 语句：
  - SELECT A1，A2，.....An
    FROM R1，R2，... ...Rm
    WHERE F
  - 详细语法：
    SELECT [ALL|DISTINCT] {*|<目标表达式 1> [,<目标表达式 2> ... ...]}
    FROM <表名或视图名 1> [，<表名或视图名 2>]... ...
    [WHERE <条件表达式>]
    [GROUP BY <列名表达式 1>[，<列名表达式 2>]] [HAVING <条件表达式>]
    [ORDER BY <列名表达式 1> [ASC|DESC]], <列名表达式 2> [ASC|DESC]]
  - 执行过程：
    1) 先按 WHERE 子句条件从 FROM 子句指定的表/视图中找出满足条件的元组（选择）；
    2) 再按 SELECT 子句中的目标表达式选择出元组中的属性，形成结果表（投影）；
    3) 如有 GROUP 子句，则将结果按<列名表达式>的值分组，该<列名表达式>值相等的元组为一个组，通常会在每组中使用聚合函数。
    4) 如果 GROUP 子句带 HAVING 子句，则对组过虑，将满足条件的组输出
    5) 如果 ORDER 子句，则将结果按<列名表达式 1>的值升序或降序排列。

3.3.1 单表查询：

假设：S（S#，SN，SS，SA，SD）
C（C#，CN，CP，CR）
SC（S#，C#，GR）

- 选取表中的某些列，即投影运算
  - 查指定列
    SELECT S#,SN FROM S
  - 查全部列
    SELECT * FROM STUDENT
  - 查经过计算的列
    SELECT SN，2002-SA FROM S
- 选择表中的若干元组，即选择运算
  - 消除取值重复行
    SELECT DISTINCT SD FROM S

- 查询满足条件的元组
  1) 比较大小：<、<= 、>、>=、=、<>
     SELECT SN，SA FROM S WHERE SD＝’CS’
     SELECT * FROM S WHERE SA<20
  2) 确定范围：BETWEEN... AND
     SELECT * FROM S WHERE SA BETWEEN 20 AND 21
  3) 确定集合：IN
     SELECT * FROM S WHERE SD IN (‘CS’,’IS’,’MA’)
  4) 字符匹配：LIKE，转义字符’\’
     SELECT * FROM S WHERE S# LIKE ‘TB%’
     SELECT * FROM S WHERE SN LIKE ‘刘_’
  5) 涉及空值的查询：IS NULL
     SELECT * FROM SC WHERE GR IS NULL
  6) 多重条件查询：
     SELECT * FROM S WHERE SD=’CS’ AND SA<20

➢ 查询结果排序：ORDER BY <字段表达式> ASC|DESC
  SELECT * FROM SC WHERE C#=’C3’ ORDER BY GR DESC

➢ 使用集(聚合)函数：
  主要有：COUNT（[DISTINCT] *）、COUNT（[DISTINCT] <字段>）
     SUM（<字段>）、AVG（<字段>）、MAX（<字段>）、MIN（<字段>）
  SELECT COUNT(*) FROM S
  SELECT COUNT(DISTINCT S#) FROM SC
  SELECT AVG(GR) FROM SC WHERE S#=’95001’
  SELECT MAX(GR) FROM SC WHERE C#=’1’

➢ 查询分组：GROUP BY
  SELECT C#，COUNT(C#) FROM SC GROUP BY C#
  SELECT S# FROM SC GROUP BY S# HAVING COUNT(*) >3 检索选修>3 门的课学生学号。

3.3.2 连接查询：

➢ 等值与非等值连接查询。自然连接
  SELECT S.*，SC.* FROM S，SC WHERE S.S# = SC.S#

➢ 自身连接
  设 C

C＃	CN	CP	CR
1	DB	5	4
2	MA		2
3	IS	1	4
4	OS	6	3
5	DataStruct	7	4
6	DataProcess		2
7	PASCAL	6	4

- 检索每门课的间接预修课。

SELECT f.C#, s.CP FROM C f,C s WHERE f.CP=s.C#

- 外连接
  - 列出所有学生的修课情况，如果没有选课也列出其基本信息(左外连接)。
    SELECT S＃，SN，SS，SA，SD，C＃，GR
    FROM S，SC WHERE S.S# *=SC.S#　(T-SQL 语法 SYBASE)
    SELECT S＃，SN，SS，SA，SD，C＃，GR
    FROM S，SC WHERE S.S# =SC.S#（＋）　（PL/SQL 语法 ORACLE）
    SELECT S＃，SN，SS，SA，SD，C＃，GR
    FROM S LEFT OUTER JION SC ON S.S#=SC.S#　(MYSQL 、MSSQL)
- 复合条件连接
  - 检索选修课程号‘C2’且成绩在 90 分以上的所有学生。
    SELECT S.S# ,SN FROM S,SC
    WHERE S.S# = SC.S# AND SC.C#=’C2’ AND SC.GR>=90
  - 检索每个学生选修的课程名及其成绩
    SELECT S.S#,SN,C.CN,SC.GR from S,SC,C
    WHERE S.S# = SC.S# AND SC.C# = C.C#

3.3.3 嵌套查询：

- 带 IN 谓词的子查询
  - 检索与“刘晨”同在一系的学生信息。
    SELECT S#，SN，SD FROM S
    WHERE SD IN（SELECT SD FROM S WHERE SN＝‘刘晨’）
  - 本例可以通过自连接来实现：
    SELECT s1.S#, s1.SN, s1.SD FROM S s1, S s2
    WHERE s1.SD = s2.SD AND s2.SN=’刘晨’
  - 检索选修了课程名的为‘信息系统’的学生学号和姓名：
    SELECT S#, SN FROM S WHERE S# IN (SELECT S# FROM SC WHERE C# IN (SELECT C# FROM C WHERE CN=’信息系统’))
  - 本例同样可以用连接来实现：
    SELECT S#, SN FROM S , SC ,C
    WHERE S.S# = SC.S# AND SC.C# = C.C#
    AND C.CN=’信息系统’
- 带比较运算的子查询
  当确定子查询的返回值是唯一时，可以使用比较运算符(注意子查询在比较符后)。
  SELECT S# , SN FROM S
  WHERE SD=(SELECT SD FROM S WHERE CN=’刘晨’)
- 带 ANY 和 ALL 的子查询（子查询返回多值时用）
  - 检索其他系中比 IS 系任一学生年龄小的学生名单
    SELECT S＃，SN FROM S WHERE SA < ANY （
    SELECT SA FROM S WHERE SD＝‘IS’）
    AND SD<>‘IS’
    ORDER BY SA DESC

等价于：
SELECT S＃，SN FROM S WHERE SA < （
SELECT MAX（SA） FROM S WHERE SD＝‘IS’）
AND SD <> ‘IS’
ORDER BY SA DESC

- 检索其他系中比 IS 系所有学生年龄都小的学生名单
  SELECT S＃，SN FROM S WHERE SA < ALL （
  SELECT SA FROM S WHERE SD＝‘IS’）
  AND SD<>‘IS’
  ORDER BY SA DESC
  等价于：
  SELECT S＃，SN FROM S WHERE SA < （
  SELECT MIN（SA） FROM S WHERE SD＝‘IS’）
  AND SD <> ‘IS’
  ORDER BY SA DESC

- 带 EXISTS 的子查询（不返回任何数据，只返回 Ture 和 False）
  - 检索所有选修了课程号为‘C01’的学生姓名
    SELECT SN FROM S WHERE EXISTS（
    SELECT * FROM SC WHERE S# = S.S# AND C# = ‘C01’）
    注意：此例中子查询的查询条件依赖于外层父查询，称此类查询为相关子查询（corelated subquery）。
    等价连接实现：
    SELECT SN FROM S，SC WHERE S.S# = SC.S# AND C# = ‘C01’
  - SQL 中没有(∀x)p 故须转换为¬(∃x(¬p))，如检索选修了全部课程的学生。即没有一门课没有选的学生。
    SELECT SN FROM S WHERE NOT EXISTS（
    SELECT * FROM C WHERE NOT EXISTS （
    SELECT * FROM SC WHERE C＃＝ C.C# AND S# = S.S#））
  - p->q 应被等价为¬p∨q 如检索至少选修了学生 S001 选修的全部课程的学生。令 p=‘学生 S001 选修了 y’ q＝‘学生 x 选修了 y’
    (∀y)（p->q）=¬∃y(¬(p->q))= ¬∃y(¬(¬p∨q))= ¬∃y(p∧¬q))即不存在这么一门课程，学生 S001 选修了而 x 没有选修
    SELECT SN FROM S WHERE NOT EXISTS(SELECT * FROM SC SC2 WHERE S# = ‘S001’ AND NOT EXISTS (SELECT * FROM SC WHERE C# = SC2.C# AND S# = S.S#))

3.3.4 集合查询：

- 使用交、并、差的集合运算概念，INTERSECT ，UNION，MINUS
  - 检索计算机科学系及年龄不大于 19 岁的学生
    SELECT * FROM S WHER SD=’CS’
    UNION
    SELECT * FROM S WHERE SA<＝19
    等价于：
    SELECT * FROM S WHERE SD＝‘CS’ OR SA <=19

- 检索选修了课程号为 C01 或 C02 的学生学号
  SELECT S# FROM SC WHERE C＃＝‘C01’
  UNION
  SELECT S# FROM SC WHERE C＃＝‘C02’
  等价于：
  SELECT S# FROM SC WHERE C＃ IN（‘C01’，‘C02’）
- 检索同时选修了课程号为 C01 和 C02 的学生学号
  SELECT S# FROM SC WHERE C＃＝‘C01’
  INTERSECT
  SELECT S# FROM SC WHERE C＃＝‘C02’(仅 ORACLE)
  等价于：
  SELECT S# FROM SC WHERE C# = ‘C01’ AND S# IN (
  SELECT S# FROM SC WHERE C# = ‘C02’)
- 检索选修了课程号为 C01 而未选修 C02 的学生学号。
  SELECT S# FROM SC WHERE C＃＝‘C01’
  MINUS
  SELECT S# FROM SC WHERE C＃＝‘C02’(仅 ORACLE)
  等价于：
  SELECT S# FROM SC WHERE C＃＝‘C01’ AND S＃ NOT IN（
  SELECT S# FROM SC WHERE C＃＝‘C02’）

3.4 SQL 的数据更新（DML）

3.4.1 数据插入：

- 插入单个元组
  - 语法：
    INSERT INTO <表名>[(<列名 1> [, <列名 2>].....)]
    VALUES(<常量 1>[,<常量 2>].....)
  - INSERT INTO S VALUES（’S001’，‘张三’，‘男’，18，‘IS’）
- 插入子查询结果
  - 语法：
    INSERT INTO <表名> [(<列名 1> [, <列名 2>].....)]
    子查询
  - 为所有学生插入一条选修 C01 课程的记录
    INSERT INTO SC
    SELECT S# , ‘C01’,null
    FROM S

3.4.2 数据修改：

- 语法：
  UPDATE <表名> SET <列名>＝<表达式>[, <列名>＝<表达式>].....
  [WHERE <条件>];
- 修改某一个元组的值
  - 将学生 S001 的年龄该为 22 岁
    UPDATE S SET SA=22 WHERE S# =’S001’
- 修改多个元组的值
  - 将所有的学生年龄增加 1 岁
    UPDATE S SET SA＝SA＋1
- 带子查询的修改语句
  - 将计算机科学系所有的学生成绩置零
    UPDATE SC set GR=0 where S# in (SELECT S# from S where SD=’CS’)
    不同的 DBMS 可以使用 join 或相关子查询实现同样功能，
    UPDATE SC SET GR=0 (相关子查询 oracle 支持)
    WHERE ‘CS’ = (SELECT SD FROM S WHERE S# = SC.S#)
    UPDATE a set GR=0 from S，SC a where S.S#=a.S# and S.SD=’CS’(SYBASE 中)
    UPDATE S,SC set SC.GR=0 where S.S#=SC.S# and S.SD=’CS’ (mysql 中)
- 修改操作与数据库的一致性
  - 某学生因退学要修改学号
    UPDATE S SET S＃＝’S002’ WHERE S＃＝’S001’
    UPDATE SC SET S＃＝’S002’ WHERE S＃＝’S001’
    必须保证数据库的一致性，引入事务概念。

3.4.3 数据删除：

- 语法：
  DELETE FROM <表名>
  [WHERE <条件>];
- 删除某一个元组的值
  - 删除学号为 S001 的学生
    DELETE FROM S WHERE S＃＝‘S001’
- 删除多个元组的值
  - 删除所有学生的选课记录
    DELETE FROM SC
- 带子查询的删除语句
  - 删除计算机科学系所有学生的选课记录
    DELETE FROM SC WHERE ‘CS’＝（
    SELECT SD FROM S WHERE S＃＝SC.S＃ ） (相关子查询)
    DELETE from SC where S# in (SELECT S# from S where SD=’CS’)

3.5 视图

- 视图只是一个窗口，其数据依赖于基本表

3.5.1 定义视图

1. 建立视图

- 语法：

  CREATE VIEW <视图名> [(<列名 1>[，<列名 2>.....])]
  AS <子查询>
  [WITH CHECK OPTION]

  - 列名必须出现的情况：
    1) 子查询的目标列是集函数等，不是单纯的列
    2) 多表连接时出现同名的列作为视图字段
    3) 需要在视图中启用新的名字
  - WITH CHECK OPTION 表示对视图更新时自动验证子查询条件。
  - 建立学生视图

    CREATE VIEW IS_S AS
    SELECT S#, SN , SA FROM S
    WHERE SD='IS'
  - 行列子集视图：若一个视图是从单个基本表导出的，并且只是去掉了基本表的某些行和某些列，但保留了码，称**行列子集视图**。
  - 建立信息系选修了 C1 号课程的学生视图。

    CREATE VIEW IS_S1 (S#, SN ,GR)
    AS
    SELECT S.S# ,SN ,GR FROM S, SC
    WHERE S.S# = SC.S# AND S.SD='IS' AND SC.C# = 'C1'
- 视图之上可以建立视图。
  - 建立信息系选修了 C1 课程且成绩在 90 以上的学生视图

    CREATE VIEW IS_S2
    AS
    SELECT S#, SN, GR FROM IS_S1 WHERE GR>=90
- 视图建立的例子
  - 建立一个反映学生出生年月的视图

    CREATE VIEW BT_S（S＃，SN，SB）
    AS
    SELECT S#, SN,2003－SA FROM S
  - 建立一个学生学号和平均成绩的视图

    CREATE VIEW S_G（S＃，AVG_GR）
    AS
    SELECT S#, AVG(GR) FROM SC
    GROUP BY S#
  - 建立一个女学生的视图

CREATE VIEW S_F（S＃，SN，SS，SA，SD）
AS
SELECT * FROM S　WHERE SS=’女’
本视图在 S 表结构改变时会出错，解决办法是去掉列说明或改*为列表

2. 删除视图

- 语法：
  DROP VIEW <视图名>
  - 删除视图 DROP VIEW IS_S;

3.5.2 查询视图

- 把对视图的查询转化为对基本表的查询称为视图的消解(View Resolution)。
  - SELECT S＃，SA FROM IS_S WHERE SA <20
    消解为：
    SELECT S# ,SA FROM S WHERE SD=’IS’ AND SA <20

  - SELECT * FROM S_G WHERE AVG_GR>90
    消解为：
    错：SELECT S#, AVG(GR) FROM SC WHERE AVG(GR)>90 GROUP BY S#
    对：SELECT S#, AVG(GR) FROM SC GROUP BY S# HAVING AVG(GR)>90

3.5.3 更新视图

- 视图的修改：
  - 将信息系学生视图中学号为 S001 的学生姓名改为‘刘辰’
    UPDATE IS_S SET SN=‘刘辰’WHERE S＃＝‘S001’
    视图消解：
    UPDATE S SET SN=‘刘辰’WHERE S＃＝‘S001’AND SD＝‘IS’

- 视图的插入：
  - 在信息系学生视图中插入记录
    INSERT INTO IS_S VALUES（‘S001’，‘刘辰’，20）
    视图消解：
    INSERT INTO S VALUES（‘S001’，‘刘辰’，NULL，20，‘IS’）
- 视图的删除：
  - 在信息系学生视图中插入记录
    DELETE FROM IS_S WHERE S＃＝‘S001’
    视图消解：
    DELETE FROM S WHERE S＃＝‘S001’AND SD＝‘IS’
- 某些带聚合/集函数的视图是不可修改的，如
  错：UPDATE S_G SET AVG_GR＝80 WHERE S＃＝‘S001’
- 不运行更新的视图规则（db2 中）：

1) 由两个以上基本表导出的视图
2) 视图的字段来自常数或表达式，只运行 DELETE
3) 视图的字段来自集函数
4) 视图中含有 GROUP BY 子句
5) 视图中含有 DISTINCT 语句
6) 视图定义有嵌套查询，且内层查询涉及到导出本视图的基本表
7) 不允许更新的视图上定义的视图

3.5.4 视图的用途

- 视图能简化用户的操作
- 视图可以使用户多角度看待同一数据
- 视图对重构数据库提供了一定的逻辑独立性
  - S（S＃，SN，SS，SA，SD）需要拆分为
    SX（S＃，SN，SS，SA），SY（S＃， SD）
    则可以通过视图来保证应用不需改变
    CREATE VIEW S AS SELECT SX.S＃，SN，SS，SA，SD
    FROM SX，SY WHERE SX.S＃＝SY.S＃
- 视图能对数据提供安全保护

3.6 数据控制语言（DCL）

3.6.1 授权

- 语法：
  GRANT {ALL PRIVILEGES|<权限>[,<权限>... ...]}
  [ON <对象类型> <对象名>]
  TO {PUBLIC|<用户>[,<用户>]... ...}
  [WITH GRANT OPTION];
  - 示例：
    GRANT SELECT ON TABLE S TO USER1;
    GRANT ALL Privileges ON TABLE S，C TO U2，U3;
    GRANT SELECT ON TABLE SC TO PUBLIC;
    GRANT UPDATE(SD),SELECT ON TABLE S TO U4;
    GRANT INSERT ON TABLE SC TO U5 WITH GRANT OPTION;
    GRANT CREATETAB ON DATABASE S_C TO U8;

3.6.2 收回权限

- 语法：
  REVOKE {ALL PRIVILEGES|<权限>[,<权限>... ...]}
  [ON <对象类型> <对象名>]

FROM {PUBLIC|<用户>[,<用户>]... ...}；

- 示例：

  REVOKE SELECT ON TABLE SC FROM PUBLIC;

  REVOKE UPDATE(SD),SELECT ON TABLE S FROM U4;

  REVOKE INSERT ON TABLE SC FROM U5;

3.7 嵌入式 SQL 语言

- SQL 语言是非过程的，而应用大多是过程化的，故通过高级语言来弥补 SQL 过程控制的不足，将 SQL 嵌入(Embedded SQL)高级语言（宿主语言）来执行。

3.7.1 嵌入式 SQL 的一般形式

- 对于嵌入式 SQL 的处理，DBMS 一般有两种处理方式：
  - 预编译
  - 修改和扩充宿主语言以处理 SQL
- 一般形式：EXEC SQL <SQL 语句>；
- 嵌入式 SQL 根据其作用不同分为两类：可执行语句和说明性语句。可执行语句包括 DDL、DML、DCL。两类 SQL 语句应和宿主语言两类语句出现在同一地方。

3.7.2 嵌入式 SQL 语句与主语言之间的通信

- 数据库工作单元和主语言工作单元之间的通信包括：
  - 向主语言传递 SQL 语句的执行状态
  - 主语言向 SQL 语句提供参数
  - 将 SQL 语句查询数据库结果交主语言进一步处理

  主要通过 SQLCA、主变量、游标来实现。

3.7.2.1 SQL 通信区

- SQLCA（SQL Communication Area）是一个数据结构，定义语句：
  - EXEC SQL INCLUDE SQLCA；
- SQLCODE 反映每次执行 SQL 语句的结果。

3.7.2.2 主变量

- 主要功能：嵌入式 SQL 可以使用主语言的变量来输入和输出数据
- 分类：输入、输出主变量、指示变量
- 使用方法：
  - 所有主变量必须在定义区定义（BEGIN DECLARE SECTION、END DECLARE SECTION）

- 可以在 SQL 中任意表达式的对方出现
- 在 SQL 语句中，主变量前要加‘:’,而在主语言中不必加。
- 指示变量用于为输入变量赋空值或指示输出变量是否空值。

3.7.2.3 游标

- 使用原因：SQL 语句是面向集合的，而主语言是面向记录的
- 主语言和 SQL 语言的分工：
  - SQL 语言负责直接与数据库打交道
  - 主语言用来控制程序流程以及对 SQL 的执行结果进一步处理
  - SQL 语言用主变量从主语言接受执行参数操作数据库－>SQL 语言的执行状态由 DBMS 送至 SQLCA->主语言从 SQLCA 取出状态信息，据此决定下一步操作。
  - SQL 的执行结果通过主变量或游标传给主语言处理。

3.7.3 不使用游标的 SQL 语句

- 不使用游标的语句有：
  - 说明性语句
  - 数据定义语句
  - 数据控制语句
  - 查询结果为单记录的 SELECT 语句
  - 非 CURRENT 形式的 UPDATE 语句
  - 非 CURRENT 形式的 DELETE 语句
  - INSERT 语句

3.7.3.1 说明性语句

- 包括：
  - EXEC SQL INCLUDE SQLCA；
  - EXEC SQL BEGIN DECALRE SECTION；
  - EXEC SQL END DECALRE SECTION；

3.7.3.2 数据定义语句

- 例：
  - EXEC SQL CREATE TABLE S
    （S＃　char（10），
    SN　char（10），
    SS　char（2），
    SA　int，

SD　　char（5））；

- EXEC SQL DROP TABLE；

- 数据定义语句中不允许使用主变量。
  - 例：错：EXEC SQL DROP TABLE :tablename；

3.7.3.3 数据控制语句

- 例：
  - EXEC SQL GRANT SELECT ON TABLE S TO U1

3.7.3.4 查询结果为单条记录 SELECT 语句

- 语法：

  EXEC SQL SELECT [ALL|DISTINCT] {*|<目标表达式 1> [,<目标表达式 2> ... ...]} INTO <主变量 1> [<指示变量 1>][,<主变量 1> [<指示变量 1>].....]
  FROM <表名或视图名 1> [，<表名或视图名 2>]... ...
  [WHERE <条件表达式>]
  [GROUP BY <列名表达式 1>[，<列名表达式 2>]] [HAVING <条件表达式>]
  [ORDER BY <列名表达式 1> [ASC|DESC][, <列名表达式 2> [ASC|DESC]]

- 例：

  EXEC SQL SELECT S#, SN INTO :sno, :sn
  　　FROM S WHERE S# =:GivenSno

- 注意：
  - into、where 和 having 子句中均可以使用主变量，需要事先申明。
  - 返回值某列为 NULL 时，系统会将指示变量赋值为－1，主变量不变。
  - 如查询结果没有满足条件的记录，则 DBMS 置 sqlcode 值为 100，正常有结果为 0。
  - 如结果不止单条，程序出错，SQLCA 中包含返回信息。

3.7.3.5 非 CURRENT 形式的 UPDATE 语句

- 例：
  - 将课程 C01 全部提分

    EXEC SQL UPDATE SC SET GR=GR+:Raise
    WHERE C# =’C01’
  - 重新设置某个学生成绩

    EXEC SQL UPDATE SC SET GR=：newgr
    WHERE S# =’S001’
  - 将计算机系所有同学成绩置空

    grid＝－1

EXEC SQL UPDATE SC SET GR=：newgr ：grid
WHERE S# IN （SELECT S＃ FROM S WHERE SD＝‘CS’）
等价：
EXEC SQL UPDATE SC SET GR=NULL
WHERE S# IN （SELECT S＃ FROM S WHERE SD＝‘CS’）

3.7.3.6 非 CURRENT 形式的 DELETE 语句

- 例：
  - 删除某学生的选课情况
    EXEC SQL DELETE FROM SC WHERE S＃IN
    （SELECT S＃ FROM S WHERE SN＝：sname）
    或者：
    EXEC SQL DELETE FROM SC WHERE ：sname＝
    （SELECT SN FROM S WHERE S.S# = SC.S#）

3.7.3.7 INSERT 语句

- 例：某个学生选修了一门课程
  grid＝－1；
  EXEC SQL INSERT INTO SC VALUES （：sno，：cno，：gr ：grid）；
  或者：
  EXEC SQL INSERT INTO SC （S＃，C＃）VALUES （：sno，：cno）；

3.7.4 使用游标的 SQL 语句

- 使用游标的语句有：
  - 查询结果为多条记录的 SELECT 语句
  - CURRENT 形式的 UPDATE 语句
  - CURRENT 形式的 DELETE 语句

3.7.4.1 查询结果为多条记录的 SELECT 语句

- 游标在 SELECT 语句的集合和主语言的一次只能处理一条记录之间架起桥梁。
- 游标步骤有：
  - 说明游标：仅仅是定义，并不执行查询
    EXEC SQL DECLARE <游标名> CURSOR FOR <SELECT 语句>；
  - 打开游标：执行相应的查询，把结果放进缓冲区，并把指针指向第一条记录
    EXEC SQL OPEN <游标名>
  - 读取当前记录并推进游标指针
    EXEC SQL FETCH <游标名>
    INTO <主变量>[<指示变量>][,<主变量>[<指示变量>].....]

- 关闭游标：释放缓冲区等资源
  EXEC SQL CLOSE <游标名>；

➢ 例：查询某个指定系的所有学生情况。

```
... ...
EXEC SQL INCLEDE SQLCA；     //说明性语句
EXEC SQL BEGIN DECLARE SECTION；
    VARCHAR depname[5]；
    VARCHAR HSno[10]；
    VARCHAR HSname[10]；
    VARCHAR HSex[2]；
    int          HSage；
EXEC SQL END DECLARE SECTION；
.... ...
gets(depname)；
... ...
EXEC SQL DECLARE SX CURSOR FOR       //说明游标
        SELECT S# , SN ,SS , SA FROM S
        WHERE SD = ： depname；
EXEC SQL OPEN SX；                          //打开游标
WHILE （1）
{
    EXEC SQL FETCH SX INTO ：HSno，：HSname，：HSex，：HSage；
                                    //读取当前记录并推进游标指针
    if（sqlca.sqlcode!=SUCCESS）break；
    printf("%s,%s,%s,%d\n", HSno， HSname， HSex， HSage);
    ... ...
}
EXEC SQL CLOSE SX；      //关闭游标
```

➢ :depname 值改变后可以重新打开游标，获得不同的集合。

3.7.4.2 CURRENT 形式的 UPDATE 和 DELETE 语句

➢ 操作步骤：

1) 说明游标
   EXEC SQL DECLARE <游标名> CURSOR FOR <SELECT 查询> **FOR UPDATE [OF <列名>]；**
2) OPEN 游标
3) FETCH 游标
4) 检查是否要修改或删除，若是执行 DELETE 或 UPDATE，并且使用 WHERE CURRENT OF <游标名>。
5) 处理完毕 CLOSE 游标

➢ 例：检索某系的学生，根据要求处理数据

```
    ... ...
EXEC SQL INCLEDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR depname[5];
    VARCHAR HSno[10];
    VARCHAR HSname[10];
    VARCHAR HSex[2];
    int          HSage;
EXEC SQL END DECLARE SECTION;
.... ...
gets(depname);
... ...
EXEC SQL DECLARE SX CURSOR FOR
        SELECT S# , SN ,SS , SA FROM S
        WHERE SD = ： depname
        [FOR UPDATE OF SA];
EXEC SQL OPEN SX;
WHILE （1）
{
    EXEC SQL FETCH SX INTO ：HSno,：HSname,：HSex,：HSage;
    if（sqlca.sqlcode!=SUCCESS）break;
    printf("%s,%s,%s,%d\n", HSno， HSname， HSex， HSage);
    printf("UPDATE(U) or DELETE(D) or NO(N)?\n");
    scanf("%c",&op);
    if(op=='U')
    {
        printf("Input new age:");
        scanf("%d",&newage);
        EXEC SQL UPDATE S SET SA=:newage WHERE CURRENT OF SX;
    }
    else if(OP=='D')
        EXEC SQL DELETE FROM S WHERE CURRENT OF SX;
    else continue;
    ... ...
}
EXEC SQL CLOSE SX;
```

3.7.5 动态 SQL 语句（以 SYBASE 的 ESQL 为例）

- 在预编译时无法获得如下信息的必须使用动态 SQL 技术：
  - SQL 语句正文
  - 主变量个数
  - 主变量数据类型
  - SQL 语句引用的数据对象

3.7.5.1 动态语句的四种实现方式

- 方法一：使用 execute immediate
  - 特点
    执行语句不能返回任何结果。
  - 语法：
    EXEC SQL [at connection_name] EXECUTE IMMEDIATE
    {: host_variable | string};
  - 例：
    EXEC SQL BEGIN DECLARE SECTION;
    CS_CHAR sqlstring[200];
    EXEC SQL END DECLARE SECTION;
    char cond[150];
    strcpy(sqlstring,"update titles set price=price*1.10 where ");
    printf("Enter search condition:");
    scanf("%s", cond);
    strcat(sqlstring, cond);
    EXEC SQL EXECUTE IMMEDIATE :sqlstring;
- 方法二：使用 prepare and execute
  - 特点
    1) 没有返回值并要求多次执行
    2) 仅有单条返回记录
  - 语法：
    EXEC SQL [at connection] PREPARE statement_name
    FROM {: host_variable | string};
    EXEC SQL [at connection] EXECUTE statement_name
    [INTO host_var_list] [USING host_var_list];
  - 例：
    EXEC SQL BEGIN DECLARE SECTION;
    CS_CHAR sqlstring[200];
    CS_FLOAT multiplier;
    EXEC SQL END DECLARE SECTION;
    char cond[150];
    printf(“Enter search condition:”);
    scanf(“%s”, cond);
    printf(“Enter price multiplier: “);
    scanf(“%f”, &multiplier);
    strcpy(sqlstring,“update titles set price = price * ? where “);
    strcat(sqlstring, cond);
    EXEC SQL PREPARE update_statement FROM :sqlstring;
    EXEC SQL EXECUTE update_statement USING :multiplier;
- 方法三：使用 prepare 和 Cursor
  - 特点
    1) SELECT 列表固定

2) 可以返回多条记录

- 语法：

```
EXEC SQL [at connection_name] DECLARE cursor_name
CURSOR FOR statement_name;
EXEC SQL [at connection_name] OPEN cursor_name
[USING host_var_list];
EXEC SQL [at connection_name] FETCH cursor_name INTO :host_variable
EXEC SQL [at connection_name] CLOSE cursor_name;
```

- 例：

```
EXEC SQL BEGIN DECLARE SECTION;
CS_CHAR sqlstring[200];
CS_FLOAT bookprice,condprice;
CS_CHAR booktitle[200];
EXEC SQL END DECLARE SECTION;
char orderby[150];
strcpy(sqlstring,"select title,price from titles where price>? order by ");
printf("Enter the order by clause:");
scanf("%s", orderby);
strcat(sqlstring, orderby);
EXEC SQL PREPARE select_state FROM :sqlstring;
EXEC SQL DECLARE select_cur CURSOR FOR select_state;
condprice = 10;
EXEC SQL OPEN select_cur using :condprice;
EXEC SQL WHENEVER NOT FOUND GOTO END;
for (;;)
{
EXEC SQL FETCH select_cur INTO :booktitle,:bookprice;
printf("%20s %bookprice=%6.2f\n",booktitle, bookprice);
}
end:
EXEC SQL CLOSE select_cur;
```

- 方法四：使用 prepare 和 动态描述符 Dynamic Descriptors
  - 特点
    1) SELECT 列表可以不固定
    2) 可以返回多条记录，条件也不固定
  - 相应内容见相关参考书

3.8 存储过程*（T-SQL）

- 语法：

```
create procedure [owner.]procedure_name
  [(@parameter_name datatype [= default][output]
  [, @parameter_name datatype [= default][output]]...)]
```

[with recompile]
as SQL_statements

- 语言要素：
  - 语句块

    begin
        statement block
    end
  - 变量

    以@开始的为用户变量，以@@开始的为全局变量
    定义变量：DECLARE
    @@rowcount 操作影响的行数
    @@sqlstatus 游标 Fetch 的状态
  - 条件控制

    if logical_expression
        statements
    [else
        [if logical_expression]
        statement]
  - 循环控制

    while boolean_expression
        statement
    break
        statement
    continue
  - 顺序控制

    label:
    goto label
  - 返回值

    return [integer_expression]
  - 打印信息

    print {format_string | @local_variable | @@global_variable} [, arg_list]
    select @local_variable | @@global_variable
  - 执行

    [execute] [@return_status =][[[server.]database.]owner.]procedure_name
        [[@parameter_name =] value | [@parameter_name =] @variable [output]
        [,[@parameter_name =] value|[@parameter_name =] @variable [output]...]]
    [with recompile]
  - 例：给定学号，获得该学生成绩，若是 C01 课程，成绩加 1，否则加 2

    CREATE PROCEDURE get_gr @sno varchar(10), @GR OUTPUT int
    AS
    DECLARE @cno varchar(5)
    BEGIN
        SELECT @cno=C#,@GR=GR FROM S WHERE S# = @sno
        IF (@cno =’C01’)THEN
            select @GR=@GR+1

```
    ELSE
      select @GR=@GR+2
END;
```

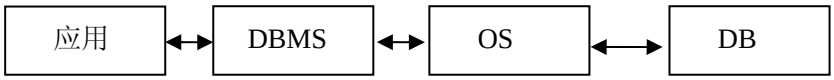
执行：

```
declare @gr int
execute get_gr ‘s001’,@gr output
select @gr  或 print @gr
```

第四章　数据库的保护

4.1 数据库的安全性

- 数据库的安全性是指保护数据库以防止不合法的使用造成泄漏、更改或破坏等。采用的方法有：用户标识和鉴别、存取权限控制、使用视图、审计、数据加密等。
- 数据库系统的安全模型：



用户标识　　存取控制　　操作系统保护　　密码存储

4.1.1 用户标识和鉴别

- 一般由系统管理员（DBA）为每个用户建立一个用户名（用户标识/帐号）和用户口令。用户必须使用此标识方可进入系统。
- 语法：
  CREATE USER <username> IDENTIFIED BY <password>
- 二次登录问题

4.1.2 存取控制

- 存取权限两要素：数据对象、操作类型
- 授权：定义某用户对哪些数据对象具有哪些类型的操作。
- 数据对象的粒度：表、列、行(一般由视图实现)。粒度越细，授权子系统越灵活，安全性越完善。但系统开销越大，数据字典庞大。

4.1.2.1 权限限制的种类

- 对用户进行控制。用户只可以访问自己的子模式
- 对操作类型进行控制。操作一般包括：查询、修改、插入、删除等。用户可以被授予不同的操作权限。
- 对数据对象的控制。用户可以被限制访问某些表或表的列

4.1.2.2 权限组(角色)概念

- 角色是数据库预先设置的一系列具有某种常用权限的组合。某用户属于一个角色就拥有该角色的所有权限了。

- SYBASE 的一些角色：
  - sa_role
  - sso_role
  - sybase_ts_role
  - navigator_role
- ORACLE 中的三类角色:
  - Connect（用于数据库录入人员）
    1) 可以访问 ORACLE
    2) 对于允许的表可以做查询和更新操作。
    3) 建立视图和同义词
  - Resource（用于开发人员）
    1) 具有 Connect 的所有权限
    2) 建表、索引、聚簇等权限
    3) 对于自己创建的对象可以操作并可以给他人赋权
    4) 使用审计命令
  - DBA（用于管理人员）
    1) 具有 Resource 的所有权限
    2) 可以访问任何用户的数据
    3) 授予或收回用户权限
    4) 建立公共同义词
    5) 建立和修改分区
    6) 执行数据库的卸出
    7) 审计

4.1.2.3 Oracle 中存取控制的语法

- 授权语句/新建用户：
  GRANT [CONNECT | RESOURCE | DBA] TO <username>
  [IDENTIFIED BY <password>]
- 收回权限/删除用户：
  REVOKE [CONNECT | RESOURCE | DBA] FROM <username>
- 操作权限控制：
  GRANT <操作权> ON <表名> TO <username|public>
  [WITH GRANT OPTION]
  - 其中“操作权”包括：SELECT、INSERT、DELETE、UPDATE、ALTER、INDEX、ALL
  - 对象拥有者自然用于对对象的操作权，只有拥有者和 DBA 可以将对象的操作权赋予别人。

4.1.3 定义视图

- 可以通过建立视图屏蔽用户不该看到的数据内容，但视图的主要功能是实现数据独立性，其安全保护功能不够精细。

4.1.4 审计

- 非强制性安全保护措施，自动记录对数据库的访问存取痕迹。
- 分为用户级和系统级。用户级主要用户设置，针对用户自己创建的对象的审计，包括对这些对象的各种访问；系统级审计是 DBA 进行的，针对用户登录的成功与否以及对数据库级权限的操作。
- 审计常常消耗大量时间和空间资源，所以一般作为可选项可灵活打开和关闭，视系统的安全性要求而定。
- 审计结果存放在 SYS.AUDIT_TRAIL 中（ORACLE 中）
- 例：对 SC 表的 ALTER 和 UPDATE 进行审计
  AUDIT ALTER，UPDATE ON SC
- 例：取消对 SC 表的任何审计
  NOAUDIT ALL ON SC

4.1.5 数据加密

- ORACLE 和 SYBASE 数据库都提供对存储过程的加密，在 SYBASE 中使用 SP_HIDETEXT

4.2 数据库的完整性

- 数据库安全性是防止非法用户的非法操作，而完整性是是防止不合语义的数据。
- 数据库完整性的实现机制：
  - DBMS 检查数据库中的数据是否满足语义规定的条件，这些加在数据库数据之上的语义约束条件称为**数据库完整性约束条件**。
  - 而 DBMS 中检查数据是否满足完整性条件的机制称**完整性检查**。

4.2.1 完整性约束条件

- 完整性约束条件作用的对象可以分为：列级、元组级和关系级三个粒度。
  - 列级约束主要约束列的取值类型、范围、精度排序等；
  - 元组级约束主要约束记录中各个字段之间的联系；
  - 关系级约束主要是约束多个记录或关系之间的联系。
- 完整性约束就其状态可以分为静态和动态的。
  - 静态主要是反映数据库的状态是合理的。
  - 动态主要是反映数据库的状态变迁是否合理。

4.2.1.1 静态列级约束

- 对数据类型的约束，包括数据的类型、长度、单位、精度等，如 char(10)。
- 对数据格式的约束，如日期 YY/MM/DD

- 对取值范围或集合的约束。
- 对空值的约束
- 其他约束。

4.2.1.2 静态元组级约束

- 静态元组约束只作用于单个元组上，如高考成绩，某个字段为其他个字段的和。

4.2.1.3 静态关系级约束

- 静态关系约束常见有四种：
  - 实体完整性约束
  - 参照完整性约束
  - 函数依赖约束
  - 统计约束

4.2.1.4 动态列级约束

- 修改定义时的约束：如由空改为非空
- 修改列值时的约束：如新旧值之间有某种要求

4.2.1.5 动态元组级约束

- 修改后的值与原来多个字段相关

4.2.1.6 动态关系级约束

- 关系变化前后的状态限制，主要是事务的一致性和原子性。

4.2.2 完整性控制

- 完整性控制机制应该具有的功能：
  - 定义功能：定义约束条件
  - 检查功能：检查用户的操作请求是否违背了完整性约束条件
  - 执行动作：在发现用户的操作违背了完整性约束条件后，能采取一定的动作来保证数据的完整性。
- 检查功能分两类：
  - 立即执行约束（immediate constraint）
  - 延迟执行约束（deferred constraint）:如帐号之间的转移与“借贷平衡约束”。
- 完整性规则的形式化表示（D，O，A，C，P）：

- D（data）：数据对象
- O（opration）：触发完整性检查的数据库操作
- A（assertion）：数据对象必须满足的语义约束/断言，规则的主体
- C（condition）：选择数据对象的条件，可包含在 D 内
- P（procedure）：违反完整性规则所触发的操作过程。

例：教授工资不低于 1000 元

D：约束作用对象是属性“工资”

O：当用户插入或修改“工资”属性时触发完整性检查

A：“工资”不能小于 1000

C：仅作用于“职称”属性值为教授的记录

P：拒绝执行用户请求

- 参照完整性应考虑的问题：
  - 外键是否可以接受空值，如 S.SD 可以，SC.S＃不可以
  - 删除被参照关系的元组时的考虑
    1) 级联删除（Cascade）：删除参照关系中相应外键值的元组
    2) 受限删除（Restricted）：只有参照关系中没有相应外键值的元组时才允许删除
    3) 置空值删除（nullified）：将参照关系中相应外键置空。

  - 修改被参照关系的元组时的考虑
    1) 级联修改（Cascade）：修改参照关系中相应元组的外键值
    2) 受限修改（Restricted）：只有参照关系中没有相应外键值的元组时才允许修改
    3) 置空值修改（nullified）：将参照关系中相应外键置空。

4.2.3 Oracle 的完整性控制

- Oracle 中的实体完整性

```
CREATE TABLE S（
    S＃  char（10），
    SN   char（20），
    SS    char（2），
    SA   int，
    SD   char（5），
    CONSTRAINT pk_1 PRIMARY KEY （S#））；
```

- Oracle 中的参照完整性

```
CREATE TABLE SC（
    S＃  char（10），
    C＃  char（4），
    GR  int，
    CONSTRAINT fk_1 FOREIGN KEY (C#) REFRENCES C（C＃），
    CONSTRAINT fk_2 FOREIGN KEY (S#) REFRENCES S（S＃）
    ON {DELETE CASCADE|SET NULL}）；缺省的 P 操作是 RESTRICT。
```

- Oracle 中的用户自定义完整性

- 列级

```
CREATE TABLE S（
    S＃ char（10）
    CONSTRAINT CHK1 CHECK （substr (S#,3,8) BETWEEN ‘00000001’ AND ‘50999999’），
    SN char（20），
    SS char（2） CONSTRAINT CHK2 CHECK （ SS IN(‘男’,’女’)），
    SA int CONSTRAINT CHK3 CHECK （SA<100），
    SD char（5），
        CONSTRAINT pk_1 PRIMARY KEY （S#））；
```

- 元组级

```
CREATE TABLE EMP（
eno       NUMBER（4）
ename     VARCHAR（20），
sal       number(7,2)，
deduct    number（7,2），
CONSTRAINT CHK1 CHECK（sal+deduct<3000））；
```

- 触发器

```
CREATE TRIGGER UPDATE_SAL:
BEFORE INSERT OR UPDATE OF sal, pos ON Teacher
FOR EACH ROW
WHEN(:new.pos=’教授’)
BEGIN
    IF :new.sal<800 THEN
        :new.sal:=800;
    END IF
END
```

4.3 数据库的并发控制

4.3.1 事务

- 事务（Transaction）：是并发控制的单位，是数据库的逻辑工作单位，是用户定义的一组操作序列，表现为一组 SQL 语句或一个程序。事务可以由用户显式指定，或隐式为数据库缺省事务。
- 显式事务的语句：
  - BEGIN TRANSACTION [transaction name]
  - COMMIT [TRANSACTION transaction name]
  - ROLLBACK [TRANSACTION transaction name]

- 事务的特性（ACID）：
  - 原子性（Atomicity）：一个事务对数据库所做的操作是不可分割的，要么全部

执行，要么什么也不做，即要么提交，要么回滚。保证事务的原子性是数据库的职责，由 DBMS 事务管理子系统实现。

- 一致性（Consistency）：事务的执行将保持数据库的一致性，即数据不会因事务的执行而被破坏。该性质由程序员和系统完整性约束检查完成。
- 隔离性（Isolation）：多个事务并发时，应和这些事务先后单独执行的结果一样，即事务内操作的数据对并发的其他事务而言是隔离的，这种隔离的程度被定义为事务的隔离级别。隔离性由 DBMS 并发控制子系统实现。
- 持久性（Durability）：一个事务一旦完成提交，其对数据库的更新将永久反映在数据库中，不会因为系统的故障而丢失。这主要有数据库的恢复管理子系统完成。

4.3.2 数据库的并发操作带来的四大问题

- 修改丢失问题：事务 2 对 A 的修改 A＝20 丢失

时间	t_0	t_1	t_2	t_3	t_4	t_5
事务 1		读 A	A=A+5		写 A	
数据库 A 值	5			20	10	
事务 2		读 A	A=A+15	写 A		

- 污（脏）读问题：正确值应该是读出 A＝5

时间	t_0	t_1	t_2	t_3	t_4	t_5
事务 1		读 A	A=A+5	写 A		回滚
数据库 A 值	5			10		A=5
事务 2					读 A(10)	

- 不可重复读问题：两次读出 A 不一致

时间	t_0	t_1	t_2	t_3	t_4	t_5
事务 1		读 A	A=A+5	写 A		
数据库 A 值	5			10		
事务 2		读 A(5)			读 A(10)	

- 幻影读问题：同样的语句返回结果不一样

时间	t_0	t_1	t_2	t_3
事务 1			insert into t1 values(4,'Fl")	
数据库列 A、B 值	Col1 Col2 1 TX 2 NY 9 CO		Col1 Col2 1 TX 2 NY 4 FL 9 CO	
事务 2		select * from t1 where		select * from t1 where

		col1<5 (结果 2 行)		col1<5 (结果 3 行)

4.3.3 数据库的并发操作调度

- 可串行性是衡量并发事务正确性的唯一标准：多个事务并发时是正确的，当且仅当其结果和某一种次序串行地执行它们时地结果相同。称这种调度策略为可串行化（Serializable）的调度。
- 串行化调度示例：

  事务 1：读 B；A＝B＋1；写回 A

  事务 2：读 A；B＝A＋1；写回 B

  - 串行调度 1（A＝10，B＝2）－－>（A＝3，B＝4）

时间	t_0	t_1	t_2	t_3
事务 1	读 B＝2	A=B+1,写 A		
事务 2			读 A＝3	B＝A+1，写 B

  - 串行调度 2（A＝10，B＝2）－－>（A＝12，B＝11）

时间	t_0	t_1	t_2	t_3
事务 1			读 B＝11	A=B+1,写 A
事务 2	读 A＝10	B＝A+1，写 B		

  - 不可串行化调度 3（A＝10，B＝2）－－>（A＝3，B＝11）

时间	t_0	t_1	t_2	t_3
事务 1	读 B＝2		A=B+1,写 A	
事务 2		读 A＝10		B＝A+1，写 B

  - 可串行化调度 4（A＝10，B＝2）－－>（A＝3，B＝4）

时间	t_0	t_1	t_2	t_3
事务 1	读 B＝2	A=B+1,写 A		
事务 2	开始	等待	读 A＝3	B＝A+1，写 B

4.3.3 数据库的封锁机制

- 锁类型：
  - 排它锁（独占锁:Exclusive Lock 简记 X）：修改数据时加 X 锁
  - 共享锁（Share Lock 简记 S）：查询时一般加 S 锁
  - 修改锁（Update Lock）：将要修改的数据加 Update 锁（并非所有 DBMS 都有）
  - 锁的相容性：

	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

- 锁的粒度(策略)：
  - DDL Lock

- Internal Lock
- Distribute Lock
- DML Lock：
  - Table Lock（All pages）
  - Page Lock（data Page）
  - Row Lock（row Lock）

4.3.4 数据库的封锁协议－－事务隔离级别

- 事务隔离级别 0（READ UNCOMMITTED）：即一级封锁协议。
  - 协议内容有：
    1) 事务 T 在修改数据前必须先对其加 X 锁，直到事务结束才释放。事务结束可以是正常结束（Commit）或非正常结束（Rollback）。
    2) 事务 T 读数据不加锁。
  - 事务功能
    1) 本事务级别防止了修改丢失问题。
    2) 不能防止脏读和保证重复读。

- 事务隔离级别 1（READ COMMITTED）：即二级封锁协议。
  - 协议内容有：
    1) 事务 T 在修改数据前必须先对其加 X 锁，直到事务结束才释放。
    2) 事务 T 读数据前必须先对其加 S 锁，读完后即释放 S 锁，而不是到事务结束才释放。
  - 事务功能
    1) 本事务级别防止了脏读问题。
    2) 不能保证重复读。

- 事务隔离级别 2（REPEATABLE READ）：即三级封锁协议。
  - 协议内容有：
    1) 事务 T 在修改数据前必须先对其加 X 锁，直到事务结束才释放。
    2) 事务 T 读数据前必须先对其加 S 锁，直到事务结束才释放。
  - 事务功能
    1) 保证重复读。

- 事务隔离级别 3（SEARIALIZABLE）：即两阶段封锁协议，可串行性封锁协议。
  - 协议内容有：在事务中，
    1) 在对任何数据读写前首先要获得对数据的锁，称扩展阶段（锁的粒度与事务隔离级别 2 有区别）。
    2) 在释放一个锁以后，不再申请任何锁，称收缩阶段。
  - 事务功能
    1) 保证事务可串化执行。

- 事务隔离级别与锁类型对照表

事务隔离级别		0 (Read Uncommited)	1 (Read Committed)	2 (Repeatable Read)	3 Serializable
X锁	操作结束释放				
	事务结束释放	√	√	√	√
S锁	操作结束释放		√		
	事务结束释放			√	√
一致性保证	防止丢失修改	√	√	√	√
	防止脏读		√	√	√
	可重复读			√	√
	防止幻影读				√

4.3.5 死锁与活锁

- 活锁：由于系统并不是按照事务申请锁的先后来决定哪个事务获得资源锁，从而导致某个事务可能永远等下去的情况称活锁。
  - 避免活锁的办法：对锁资源的分配采用先申请先服务的策略
- 死锁：事务 T1 封锁了 A，事务 T2 封锁了 B，然后事务 T1 申请封锁 B，而事务 2 申请封锁 A，这种 T1 等待 T2 而 T2 又等待 T1 释放资源的情况称死锁。
  - 死锁避免的方法：
    1) 一次封锁法：即一次将所有用到的数据全部加锁，否则不可执行。
    2) 顺序封锁法：即尽量按照同一次序去访问数据资源。当然应用有时必须按照某种次序执行，甚至是不同条件会有不同的访问次序。

4.4 数据库的恢复

4.4.1 恢复的原理

- 数据库的恢复子系统就是使发生故障的数据库恢复到一致性状态。
- 数据库系统发生故障的种类有：事务故障、系统故障和介质故障。

4.4.1.1 事务故障

- 由于某种原因，如运算溢出、违反完整性限制、并行事务发生死锁等导致事务没有正常结束而异常中止，称事务故障。
- 恢复程序要强行回滚（Rollback）未完成的事务中的部分修改，回到事务运行前的状态，这种回滚操作称为事务撤消（UNDO）。

4.4.1.2 系统故障

- 由于某种原因，如 OS 和 DBMS 错误、硬件错误（CPU 故障）、突然掉电等导致事务务没有正常结束而异常中止，这时外设上的数据正常而内存中的数据全部丢失，称系统故障。
- 恢复程序要强行回滚（Rollback）未完成的事务回到事务运行前的状态（UNDO）；同时需要将已提交的事务但可能存在内存而尚未写回外设的事务重新写回外设，这个过程称为重做（REDO）。

4.4.1.3 介质故障

- 由于某种原因，如硬盘损坏、强磁场干扰等导致存储在外设上的数据部分或全部丢失，称介质故障。
- 恢复程序需要装入数据库发生介质故障前的某个时刻的数据库副本，并重做自此时刻开始的所有成功事务，直到发生故障前一时刻。

4.4.2 恢复的实现技术

- 恢复就是利用存储在系统其他地方的冗余数据来修复数据库中被破坏或不正确的数据，所以恢复的核心问题有两个：
  - 如何建立冗余数据
  - 如何利用冗余数据实施恢复。
- 常常是数据转储技术和日志技术的联合使用。

4.4.2.1 数据转储

- 数据转储就是数据库 DBA 将数据库复制到磁带或磁盘的其他地方的过程。这些复制的数据称为数据库的后备副本或后援副本或备份。
- 数据的后备副本在恢复后还要重做转储以后运行的所有更新事务。
- 数据转储从转储时数据库的状态来分，分为：
  - 静态转储：转储时数据库必须无运行的事务，会影响数据库的生产
  - 动态转储：转储和事务可以并发运行，减少对生产的影响，但转储数据库副本的同时必须记录转储期间的事务，即事务日志文件。这样才可以用该后数据库副本和日志文件进行数据库恢复。
- 数据转储有几种方式：
  - 海量转储（完全转储）：每次都完全备份的数据库数据
  - 增量转储：每次备份在前一次基础上增加的数据
  - 差值转储：每次备份自上次完全转储以来的数据库增加的数据

4.4.2.2 日志文件

- 日志文件是用来记录事务对数据库的更新操作的文件。
- 日志有两种：
  - 以记录为单位
  - 以数据块为单位

4.4.3 恢复的策略

4.4.3.1 事务故障的恢复

- 反向扫描日志文件，查找该事务更新操作
- 对该事务的更新操作执行逆操作（UNDO）

4.4.3.2 系统故障的恢复

- 反向扫描日志文件，查找该事务更新操作
- 对该事务的更新操作执行逆操作（UNDO）
- 正向扫描日志文件，对重做队列中的事务进行重做（REDO）

4.4.3.3 介质故障的恢复

- 装入最新的数据库副本
- 装入相关的日志文件，重做相关事务

4.5 数据库的复制与镜像

4.5.1 数据库的复制

- 分三种：
  - 对等复制：各个结点平等，可以互相复制数据
  - 主从复制：数据只会从主数据库复制到从数据库
  - 级联复制：数据从主数据库复制到从数据库后，再从从数据库复制到其他数据库

4.5.2 数据库的镜像

- DBMS 可以提供日志文件和数据库的镜像，以防止介质故障后无法快速恢复应用。
- 所谓数据库镜像是指根据 DBA 的要求，DBMS 自动把整个数据库或其中的关键数据复制到另一个**磁盘**上，保证镜像数据与主数据一致性。一旦介质故障出现，DBMS 会自动利用镜像磁盘进行数据库恢复，不需要关闭系统和重装数据库副本，同时镜像数据库还可以用于并非操作。如主数据库有 X 锁，用户可以查询镜像数据库。
- 镜像数据库平时始终和源数据库保持同步，还可以在源数据库进行修改操作时提供查询操作。

第五章　关系数据库设计理论

5.1 数据依赖

- 一个关系数据库包含一组关系，定义这组关系的关系模式的全体就构成了数据库模式。

5.1.1 关系模式中的数据依赖

- 完整的关系模式的描述：R（U，D，DOM，F）。
  - R 关系名
  - U 属性组
  - D 是 U 的取值范围，是域的集合。
  - DOM 是属性向域映象的集合。
  - F 是属性间数据的依赖关系集合。
- 关系模式是静态的、稳定的；关系是动态的，不同时刻关系模式中的关系可能不同，但关系都必须满足关系模式中数据依赖关系集合 F 指定的完整性约束。
- 影响数据库模式设计的主要是 U 和 F，所以一般关系模式简化为：R（U， F）

5.1.2 数据依赖对关系模式的影响

- 数据依赖目前有以下几种：函数依赖、多值依赖和连接依赖
- 一个关系模式示例：

  U＝{Sno,Sdept,Mname,Cname,Grade}

  F＝{Sno->Sdept, Sdept->Sname, (Sno,Cname)->Grade}

  该关系模式存在如下问题：
  - 数据冗余太大：系主任名字重复出现，和所有学生的所有课程成绩次数一样
  - 更新异常：更换系主任必须修改每一个学生信息
  - 插入异常：刚成立的系如果还没有招生就无法存储系主任信息
  - 删除异常：某个系的学生全部毕业删除时会丢失系主任信息

5.1.3 相关概念

- **函数依赖**：R（U）是一个关系模式，U 是 R 的属性集合，X 和 Y 是 U 的子集，对于 R(U)的任意一个可能的关系 r，如果 r 中不存在两个元组 w，v，使得 w[X]=v[X]而 w[Y]<>v[Y]，称 X 函数决定 Y，或 Y 函数依赖于 X，记 X->Y。
- **平凡的和非平凡的函数依赖**：关系模式 R（U）， X 和 Y 是 U 的子集，如果 X->Y，且 Y⊄X，则称 X->Y 是非平凡的函数依赖，否则称平凡的函数依赖，我们讨论的都是非平凡的函数依赖。

- **完全函数依赖和部分函数依赖**：关系模式 R（U），如果 X－>Y，且对于任意的 X 的真子集 X’都有 X’$\not\rightarrow$>Y，则称 Y 完全函数依赖于 X，记 X$\xrightarrow{f}$>Y。若 Y 不完全依赖于 X 则称 Y 部分依赖于 X，记 X$\xrightarrow{P}$>Y。
- **传递函数依赖**：关系模式 R（U），如果 X－>Y，Y－>Z，且 Y$\not\rightarrow$>X，则称 Z 传递函数依赖于 X，记 X$\xrightarrow{t}$>Z。
- **码的重新定义**：关系模式 R（U,F），K 为属性组合，若 K$\xrightarrow{f}$U，则 K 是一个候选码。

5.2 范式

- 范式是数据依赖满足某种条件级别的关系模式的集合。
- 目前共 6 种范式，1NF⊃2NF⊃3NF⊃BCNF⊃4NF⊃5NF

5.2.1 第一范式（1NF）

- 定义：如果一个关系模式 R 的所有属性都是原子的，即不可再分的基本数据项，则 R∈1NF。
- 示例：SCL（S＃，SN，SA，CLS，MON，C＃，CN，CRD，GR）属于 1NF
  它有以下问题：
  - 数据冗余大，如 MON，CRD 等
  - 插入异常，当无课程时学生信息无法插入
  - 删除异常，当某个学生的选课信息全部删除时无法保留学生基本信息。
- SCL 存在的函数依赖关系：
  - （S＃，C＃）$\xrightarrow{f}$>GR
  - （S＃，C＃）$\xrightarrow{P}$>SN　S#$\xrightarrow{f}$>SN
  - （S＃，C＃）$\xrightarrow{P}$>SA　S#$\xrightarrow{f}$>SA
  - （S＃，C＃）$\xrightarrow{P}$>CLS　S#$\xrightarrow{f}$>CLS
  - （S＃，C＃）$\xrightarrow{P}$>CN　C#$\xrightarrow{f}$>CN
  - （S＃，C＃）$\xrightarrow{P}$>CRD　C#$\xrightarrow{f}$>CRD
  - CLS$\not\rightarrow$>MON　S#$\xrightarrow{t}$>MON

5.2.2 第二范式（2NF）

- 定义：如果一个关系模式 R∈1NF，并且每一非主属性都完全依赖于 R 的码，则 R∈2NF。显然码只包含一个属性的 R 如果是 1NF，则必是 2NF
- 示例：S_L（S＃，SN，SA，CLS，MON）
  C（C＃，CN，CRD）
  S_C（S＃，C＃，GR）都属于 2NF
  它有以下问题：
  - 数据冗余大，如 MON

- 插入异常，无学生信息无法插入班长信息
- 删除异常，当学生的信息删除无法保存班长。

➢ 函数依赖关系：

- S＃ $\xrightarrow{f}$ >SA
- S＃ $\xrightarrow{f}$ >CLS
- S＃ $\xrightarrow{f}$ >SN
- CLS——>MON　S# $\xrightarrow{t}$ >MON

- C＃ $\xrightarrow{f}$ >CN
- C＃ $\xrightarrow{f}$ >CRD
- （S＃，C＃） $\xrightarrow{f}$ >GR

5.2.3 第三范式（3NF）

➢ 定义：如果一个关系模式 R 中不存在非主属性对码的传递依赖，则 R∈3NF。

➢ 示例：S（S＃，SN，SA，CLS）

L（CLS，MON）

C（C＃，CN，CRD）

S_C（S＃，C＃，GR）都属于 3NF

5.2.4 BC 范式（BCNF）

➢ 定义：如果一个关系模式 R（U,F）∈1NF，对 R 中的任意一个非平凡的函数依赖 X－>Y，X 都含有候选码，则 R∈BCNF。

➢ 示例：STC（S，T，C），S 学生，T 教师，C 课程

（S，C）－>T

（S，T）－>C

T－>C

分解为：ST（S，T），TC（T，C）则都属于 BCNF。

5.2.5 第四范式（4NF）

➢ 关系模式 R（U）属性集 U，X、Y 和 Z 是 U 不相交的子集，且 Z＝U－X－Y，若关系模式 R 的任一关系 r 对于 X 的一个给定值，存在 Y 的一组值与之对应，且 Y 的这一组值与 Z 无关，称 Y 多值依赖于 X，记 X－>－>Y。当 Z 非空时称非平凡的多值依赖。

➢ 定义：如果一个关系模式 R（U,F）∈1NF，对 R 中的任意一个非平凡的多值依赖 X－>－>Y，X 都含有候选码，则 R∈4NF。

➢ 示例：CTX（C，T，X） C：课程，T：教师，X：参考书

候选码：（C，T，X）

C－>－>T，C－>－>X，C 不是候选码，故 CTX 不属于 4NF

分解为 CT（C，T），CX（C，X），则都满足 4NF

5.2.6 第五范式（5NF）

- 定义：关系模式 R，其属性集 U，X1，X2.....Xn 分别为 U 的子集，∪Xi＝U，如果对于 R 的每一个关系 r 都有 r＝∞Xi，则称连接依赖（JD）在关系模式 R 上成立，记为*（X1，X2，.....Xn）,若某个 Xi 就是 R，称平凡的连接依赖。
- 示例：关系 AFP

A	F	P
a1	f1	p1
a1	f1	p2
a1	f2	p1
a2	f1	p2
a2	f3	p2

AF

A	F
a1	f1
a1	f2
a2	f1
a2	f3

FP

F	P
f1	p1
f1	p2
f2	p1
f3	p2

AP

A	P
a1	p1
a1	p2
a2	p2

AF∞FP

A	F	P
a1	f1	p1
a1	f1	p2
a1	f2	p1
a2	f1	p1
a2	f1	p2
a2	f3	p2

AF∞FP∞AP=AFP

A	F	P
a1	f1	p1

a1	f1	p2
a1	f2	p1
a2	f3	p2
a2	f1	p1

若在 AFP 中插入(a2,f2,p2)则

AFP

A	F	P
a1	f1	p1
a1	f1	p2
a1	f2	p1
a2	f1	p2
a2	f3	p2
a2	f2	p2

AF

A	F
a1	f1
a1	f2
a2	f1
a2	f3
a2	f2

FP

F	P
f1	p1
f1	p2
f2	p1
f3	p2
f2	p2

AP

A	P
a1	p1
a1	p2
a2	p2

AF∞FP

A	F	P
a1	f1	p1
a1	f1	p2
a1	f2	p1
a2	f1	p1
a2	f1	p2
a2	f3	p2
a1	f2	p2
a2	f2	p1

a2	f2	p2

AF∞FP∞AP=AFP

A	F	P
a1	f1	p1
a1	f1	p2
a1	f2	p1
a2	f3	p2
a2	f1	p1
a1	f2	p2
a2	f2	p2

- 定义：如果一个关系模式 R（U,F）∈1NF，对 R 中的任意一个连接依赖都都由候选码蕴涵，则 R∈5NF。

5.3 关系模式的规范化

5.3.1 关系模式的规范化步骤

消除决定属性集非码的非平凡函数依赖	范式级别	处理方法
	1NF	
	↓	消除非主属性对码的部分函数依赖关系
	2NF	
	↓	消除非主属性对码的传递函数依赖关系
	3NF	
	↓	消除主属性对码的部分函数依赖关系
	BCNF	
	↓	消除非主属性的非平凡的多值依赖
	4NF	
	↓	消除非候选码蕴涵的连接依赖
	5NF	

5.3.2 关系模式的分解

- 关系模式的规范化是通过对关系模式的分解来实现的。但把低级别的关系模式分解为高级别的关系模式；分解不唯一，只有保证分解后的关系模式与原关系模式等价，才有意义。
- 示例：
  关系模式 SL（Sno，Sdept，Sloc），Sno－>Sdept，Sdept－>Sloc，Sno$\xrightarrow{t}$>Sloc
  设 SL 有如下关系：

Sno	Sdept	Sloc
95001	CS	A
95002	IS	B
95003	MA	C
95004	IS	B
95005	PH	B

- 分解方法一：

  SN(Sno)、SD(Sdept)、SO(Sloc)

  SN、SD、SO 都是很高的范式，属于 5NF，但分解后丢失很多信息。

SN	SD	SO
Sno	Sdept	Sloc
95001	CS	A
95002	IS	B
95003	MA	C
95004		
95005	PH	

- 分解方法二：

  NL(Sno，Sloc)、DL(Sdept，Sloc)分解后关系为：

NL		DL	
Sno	Sloc	Sdept	Sloc
95001	A	CS	A
95002	B	IS	B
95003	C	MA	C
95004	B	PH	B
95005	B		

NL 和 DL 的自然连接结果是：

Sno	Sdept	Sloc
95001	CS	A
95002	IS	B
95002	PH	B
95003	MA	C
95004	IS	B
95004	PH	B
95005	IS	B
95005	PH	B

多出三个元组，而实际上无法确定哪个是多余的，因此丢失了信息。

- ◆ 定义：如果关系模式 R（U，F）在分解为若干个关系模式 Ri（Ui，Fi）后（其中 U＝∪Ui），若 Ri 的自然连接和原关系相等，则称该分解具有无损连接性。

■ 分解方法三：

ND(Sno，Sdept)、NL(Sno，Sloc)分解后关系为：

ND		NL	
Sno	Sdept	Sno	Sloc
95001	CS	95001	A
95002	IS	95002	B
95003	MA	95003	C
95004	IS	95004	B
95005	PH	95005	B

NL 和 DL 的自然连接结果是：和原关系相同，但当某个学生转系后需要修改两个表，如 95005 转为 CS 系后需 PH－>CS，B－>A。原因是分解时丢失了 Sdept－>Sloc 的函数依赖关系。

- ◆ 定义：如果在分解过程中原函数依赖 F 被每个分解后的某个关系函数依赖 Fi 所逻辑蕴涵，则称该分解是保持函数依赖的。

■ 分解方法四：

ND(Sno，Sdept)、DL(Sdept，Sloc)分解后关系为：

ND		DL	
Sno	Sdept	Sdept	Sloc
95001	CS	CS	A
95002	IS	IS	B
95003	MA	MA	C
95004	IS	PH	B
95005	PH		

NL 和 DL 的自然连接结果是：和原关系相同，该分解是保持函数依赖的。

➢ 衡量关系分解有两个标准：

■ 是否具有无损连接性

■ 是否保持了函数依赖

➢ 关系分解理论定理：

■ 若要求关系模式分解具有无损连接性，则分解一定可达到 4NF

■ 若要求关系模式分解保持函数依赖，则分解一定可达到 3NF，但不一定达到 BCNF

■ 若要求关系模式分解既具有无损连接性，又保持函数依赖，则分解一定可达到 3NF，但不一定达到 BCNF

第六章　数据库设计

6.1 数据库数据的步骤

- 需求分析
- 概念结构设计
  - 设计局部视图
  - 集成视图
- 逻辑结构设计
  - 设计逻辑结构
  - 优化逻辑结构
- 数据库物理设计
  - 设计物理结构
  - 评价物理结构
- 数据库实施
  - 数据库系统的物理实现
  - 试验性运行
- 数据库运行维护

6.2 需求分析

6.2.1 需求分析的任务

- **需求分析的任务是**通过详细调查现实世界和要处理的对象（组织、部门、企业等），充分了解原系统的工作概况，明确用户的各种需求，然后在此基础上确定新的系统功能。新系统应该考虑可扩展性。
- **需求分析的重点是：**调查、收集与分析用户在数据库管理中的信息要求、处理要求、安全要求和完整性要求。
- **需求分析的结果是：**DD（数据字典）、DFD（数据流图）。

6.2.2 需求分析的方法

- **调查与初步分析的步骤：**
  - 调查组织机构情况
  - 调查各部门业务活动情况
  - 在熟悉业务基础上，协调用户明确对新系统得要求
  - 对上述结果初步分析，确定新系统得边界，及人与计算机得工作边界。
- **常用的调查方法：**

- 跟班作业
- 开调查会
- 请专业认识介绍
- 询问
- 设计调查表请用户填写
- 查阅记录

- **分析用户需求的方法：**
  - 自顶而下，结构化分析方法(Structured Analysis，简称 SA)
  - 自底向上

6.2.3 数据字典

- 数据字典是详细数据收集和数据分析的结果。包涵以下内容：
  - 数据项：不可再分的数据单位。
    对数据项的描述包括：数据项名、含义说明、别名、数据类型、
    长度、取值范围、取值含义、与其他数据项的逻辑关系
  - 数据结构
    数据结构反映了数据之间的组合关系。
    数据结构的描述包括：数据结构名,含义说明,组成(数据项、数据结构)
  - 数据流
    数据流是数据结构在系统内传输的路径。
    数据流的描述包括：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、高峰期流量等
  - 数据存储
    数据存储是数据结构停留或保存的地方，也就是数据流的来源和去向之一。
    数据存储的描述：数据存储名、说明、编号、流入数据流、流出数据流、组成（数据结构）、数据量、存取方式
  - 处理过程
    处理过程的处理逻辑一般用判定树和判定表来描述。数据字典一般只是描述说明性信息，描述包括：处理过程名、说明、输入（数据流）、输出（输出流）、简要说明。

6.3 概念结构设计

6.3.1 概念结构的设计方法与步骤

- 自顶向下：先定义全局概念结构，再细化
- 自底向上：先定义局部应用的概念结构，再集成起来，得到全局概念结构
- 逐步扩张：先定义核心概念结构，再逐步向外扩充，直至全局概念结构。
- 混合策略：即使用自顶向下、自底向上相集合

6.3.2 数据抽象与局部视图设计

1. 选择局部应用
2. 逐一设计分 E－R 图
   - 属性与实体很难有截然划分的界线：
   1) 属性不能再具有需要描述的性质
   2) 属性不能与其他实体具有联系

6.3.3 视图的集成

1. 合并分 E－R 图，生产初步 E－R 图，合并分 E－R 图过程中存在的冲突有：
   - 属性冲突：属性域冲突、属性单位冲突
   - 命名冲突：同名异义，异名同义
   - 结构冲突：同一对象抽象不同，同一实体属性不同，联系类型不同。
2. 修改与重构，生成基本 E－R 图
   初步 E－R 图消除不必要冗余后得到基本 E－R 图。
   视图集成后形成整体概念结构，必须满足：
   - 结构内部必须具有一致性，不能有互相矛盾的表达
   - 整体结构必须能反映原来的每一个视图结构，包括实体，属性和联系
   - 结构能满足需求分析阶段的所有需求。

6.4 逻辑结构设计

- 逻辑结构设计的**任务**：将概念结构转化为某一数据模型。
- 逻辑结构设计的**步骤**：
  - 将概念模型转化为一般的关系、层次、网状模型。
  - 将转化来的关系、层次和网状模型向特定的 DBMS 支持下的数据模型转换。
  - 对数据模型进行优化。

6.4.1E－R 图向数据模型转换

- 转化的**原则**：
  - 一个实体型转化为一个关系模式。
  - 一个 m：n 的联系转化为一个关系模式，码为各实体码组合。
  - 一个 1：n 的联系　　转化为一个独立的关系模式，码为 n 端实体码；也可以与 n 端关系模式合并。
  - 一个 1：1 的联系转化为一个独立的关系模式，每个实体的码均是候选码；也可以与任一端关系模式合并。
  - 三个及三个以上实体间的一个多元联系转化为一个关系模式。
  - 同一实体集的实体间的联系即自联系，也可按上面的联系方式处理。
  - 具有相同码的个模式可以合并。

6.4.2 数据模型的优化

- 确定数据依赖
- 对各个关系模式间的数据依赖进行极小化处理，消除冗余的联系。
- 按照数据依赖理论对关系模式逐一进行分析考察是否存在部分函数依赖、传递函数依赖、多值依赖等，确定各关系模式分别属于第几范式。
- 按照需求分析阶段得到的各种应用对数据处理的要求，分析这样的应用环境这些关系模式是否合适，确定是否要对它们进行合并和分解。
- 对关系模式进行必要的分解和合并。

6.4.3 设计用户子模式

- 使用更符合用户习惯的别名。
- 针对不同级别的用户，定义不同的外模式，以满足系统对安全性的要求。
- 简化用户对系统的使用。

6.5 数据库物理设计

6.5.1 确定数据库的物理结构

- 确定数据的存储结构。综合考虑存取时间、存储空间利用率和维护代价。
  聚簇的使用条件：
  - 通过聚簇码进行访问是改关系的主要应用。
  - 对应与每个聚簇码的平均元组数既不太少，也不太多。
  - 聚簇码值相对稳定，以减少修改码值引起的维护开销。
- 设计数据存取路径。主要是如何建立索引。
- 确定数据存放位置。主要是日志/数据、索引/数据的存放尽量分开。
- 确定系统配置。打开对象数、缓冲区大小、时间片大小、锁数目等。

6.5.2 评价物理结构

- 需要权衡的因素：时间效率、空间效率、维护代价、用户需求
- 评价数据库的方法完全依赖于选用的 DMBS

6.6 数据库实施

数据库实施的主要工作包括：

- 用 DDL 定义数据库结构
- 组织数据入库
- 编制调试应用程序

- 数据库试运行

6.6.1 定义数据库结构

6.6.2 数据装载

- 对于小型系统数据装载步骤：
  - 筛选数据
  - 转换数据格式
  - 输入数据
  - 校验数据
- 对于大中型系统的数据装载步骤：
  - 筛选数据
  - 输入数据
  - 校验数据
  - 转换数据
  - 综合数据

6.6.3 编制和调试应用程序

6.6.4 数据库试运行

- 包括功能测试和性能测试。

6.7 数据库运行维护

本阶段主要是 DBA 的工作。包括：

6.7.1 数据库的转储和恢复

6.7.2 数据库的安全性完整性控制

6.7.3 数据库性能的监督、分析和改进

6.7.4 数据库的重组织和重构造

- 数据库重组：不会改变数据逻辑和物理结构，只是重新安排存储，回收垃圾等。

- 数据库重构：应用需求改变，要求改变逻辑设计，如表结构等，就是重构。
- 数据库系统重新设计：数据库重构的程度是十分有限的，当重构的代价太大时，就表示现有的数据库系统的生命周期已经结束，应该重新设计新的数据库系统了。

第七章　关系数据库管理系统实例

7.1 关系数据库管理系统产品概述

RDBMS 的发展情况：

- 对关系数据库支持的三阶段
  - 70 年代，支持数据结构和基本数据操作（选择、投影、连接等），DBASE 等。
  - 80 年代，支持国际标准 SQL，甚至超出（TSQL，PL/SQL）,Oracle 等。
  - 90 年代，加强安全性和完整性。

- 运行环境发展的三阶段：
  - 一般多为多用户系统的大中小型机器上运行的单机 RDMBS，微机上的均为单用户的，因为微机 DOS 是单用户操作系统。
  - 两个方向发展：一是提高移植性，使之在多种硬件和操作系统上；二是数据库联网，向分布式系统发展，支持多网络协议。
  - 在网络环境下，分布式数据库和客户/ 服务器结构数据库系统的推出。追求数据库的开放性（可移植性 portability、可连接性 connectivity、可伸缩性 scalability）。

- RDBMS 的系统构成变化：
  - 早期的 RDBMS 主要实现 DDL、DML、DCL 等基本操作以及数据存储组织、并发控制、安全性完整性检查、系统恢复、数据库的重组和重构。
  - 后期的 RDBMS 以数据管理的基本功能为核心，开发外围软件系统，包括 FORMS、REPORT、GRAPHICS 等。

- RDBMS 对应用的支持
  - 第一阶段主要是用于信息管理，对联机速度要求不高。
  - 第二阶段主要针对联机事务处理，一提高事务吞吐量；二缩短联机响应时间
  - RDBMS 的改善技术主要有：
    - 性能
    - 可靠性

7.2 ORACLE 数据库

7.2.1 Oracle 公司简介

- 成立于 1977 年
- 1979 年，Oracle 第一版是世界上首批商用 RDBMS 之一。
- 1992 Oracle7、1997 年 Oracle8 、Oracle 9i 、Oracle 10g

7.2.2 Oracle 产品特性

- 兼容性（compatibility）：兼容其他厂商的数据库兼容。
- 可移植性（portability）：可以安装 70 多种机器，多种操作系统。
- 可联结性（connectability）：支持多种网络协议，如 TCP/IP、DECnet 等。
- 高生产率（high productivity）：提供 PRO*C 、Forms 等接口与工具。
- 开放性：前述特性保障了其开放性。

7.2.3 Oracle 数据库服务器产品

- 标准服务器
  - 多进程、多线程体系结构
  - 为提高性能改进核心技术：无限制行级锁、无竞争查询、多线程顺序号产生。
  - 高可用性
  - SQL 的实现：符合 ISO 的 SQL 标准
- 并行服务器选件（Oracle Parallel Server）和并行查询（Parallel query Option）
- 分布式选件（distributed Option）
- 过程化选件（Procedural option）：提供用户自定义数据库对象。

7.2.4 Oracle 工具

- Developer/2000
  - ORACLE Forms：屏幕工具
  - ORACLE Reports：报表工具
  - ORACLE Graphics：图形工具，如直方图等。
  - ORACLE Book ：用于生产联机文档。
- Designer/2000
  - BPR ：用于过程建模
  - Modellers：用于系统设计与建模。
  - Generators ：应用生产器。
- Discoverer/2000：OLAP 工具，应用于数据仓库等。
- Oracle Office：办公自动化。
- SQL DBA：用户动态性能监控。
- ORACLE 预编译器 Proc*C
- ORACLE 调用接口 OCI

7.2.5 Oracle 连接产品

- SQL *Net：负责 Client 和 Server 的通讯。
- Oracle 多协议转换器
- Oracle 开放式网关：利用透明网关和过程化网关，可以实现对其他数据库的直接访问。

7.2.6 Oracle 数据仓库解决方案

- OracleOLAP
  - Oracle Express Server
  - Oracle Express Objects
  - Oracle Express Analyzer

7.2.7 Oracle 的 Internet 解决方案

- Oracle WebServer 2.0

7.3 Sybase 数据库

7.3.1 Sybase 公司简介

- 1984 年成立，INGRES 大学版本的主要设计人员之一 Dr.Robert Epstein 是 Sybase 的创始人之一。
- 致力于 C/S 数据库体系结构以满足 OLTP 应用要求，1987 年推出 SYBASE SQL Server。
- Sybase 11.0、11.5、11.9、12.0、12.5 都是很优秀的版本。

7.3.2 Sybase 关系数据库产品

- 数据库服务器：
  - Sybase SQL Server
  - Sybase MPP
  - Sybase IQ
  - Sybase Anywhere
- 中间件：
  - Replication Server
  - Open Client
  - Open Server
  - OmniCONNECT
  - ObjectCONNECT for C++ Directconnect
- 工具：
  - PowerBuilder
  - S－Designor
  - Optima＋＋
  - NetImpact Studio
  - Internet Developer Toolkit for PowerBuilder

7.3.3 Sybase 数据库服务器(Adaptive Server)

- SQL Server：RDBMS，负责高速计算、数据管理、事务管理。
  - 单进程多线程体系结构
  - 高性能
  - 数据完整性检查和控制
  - 加强安全保密功能，基于角色管理，提供审计
  - 支持分布式查询和更新
- 备份服务器（backup server）：附属于 SQL Server，完成对数据的备份工作。
  - 支持联机备份：备份时不影响 SQL Server 处理。
  - 支持转储分解：允许使用多台外设进行转储。
  - 支持异地转储：DBA 可以管理多个远程服务器的备份和装载。
  - 支持限值转储：日志转储可以在限值事件触发下自动完成。
- SYBASE MPP：多处理器并行服务器产品
- SYBASE IQ：高性能决策支持和交互式数据集成产品。
- SYBASE SQL　Anywhere：基于 PC 的具有 SQL 功能的分布式数据库管理系统。

7.3.4 Sybase 开发工具

- PowerBuilder：基于图形界面的 C/S 前端应用开发工具。
- PowerDesigner：一组紧密集成的计算机辅助软件工程（CASE）工具，用于数据库的分析、设计维护、建立文档和创建数据库等功能。
- PowerJ：开发基于 java 应用的快速开发工具。
- Power++（Optima++）：RAD C++ C/S 和 Internet 面向对象的开发工具。
- SQL Manager：可视化的系统和数据库管理工具。

7.3.5 Sybase 中间件

- Open Client/Open Server：构成 Sybase 开放式 C/S 互连基础。Open Client 用于建立有效的前端应用；Open Server 是一个服务构造工具，用于集成企业的资源与服务。
- Jaguar CTS：是 jaguar 组件事务服务器（component transaction server）简称，专门为 NetOLTP 应用设计的事务服务器。
- Replication Server：复制服务器。
- OmniCONNECT：不同数据库管理系统（都是 sybase）之间完全透明的数据集成。
- DirectConnect：用于同非 Sybase 数据源建立联系的访问服务器。

7.3.6 Sybase 数据仓库解决方案

- Sybase Web. Works

7.4 INFORMIX 数据库

- 1988 年 Infomix 推出第一代数据库服务器 INFORMIX－TURBO 后来被 IBM 收购。

7.5 DB2 数据库

- DB2 是 IBM 的数据库管理系统，起源于 System R 和 System R*。

7.6 INGRES 数据库

- INGRES 公司成立于 1980。其技术源于加州伯克利大学。PostgreSQL 的前身。
- PostgreSQL 可以说是最富特色的自由数据库管理系统，过于学院味，因为首先它的目的是数据库研究，因此不论在稳定性，性能还是使用方便方面都比商用数据库欠缺。

7.7* 两层及多层应用系统体系架构

第八章　数据库技术新进展

8.1 数据库发展概述

- 数据管理发展三阶段
  - 人工阶段
  - 文件管理阶段
  - 数据库阶段
- 数据库数据模型发展的三阶段：
  - 格式化数据模型（层次、网状）
  - 关系模型
  - 丰富的数据模型（对象、语义）
- 关系数据库发展的三阶段
  - 支持数据结构和基本数据操作（选择、投影、连接等），DBASE 等
  - 支持国际标准 SQL，甚至超出（TSQL，PL/SQL）Oracle 等
  - 加强安全性和完整性

8.2 数据库技术与其他技术的结合

8.2.1 分布式数据库

- 分布式数据库特点：
  - 数据的物理分布性
  - 数据的逻辑整体性
  - 数据的分布独立性（透明性）：除了物理独立性、逻辑独立性，用户不必关心数据的分布细节。
  - 场地自治与协调：各结点能执行局部应用请求，也能通过网络处理全局请求。
  - 数据的冗余及冗余透明性：适当冗余提高系统效率和可靠性，冗余对用户透明。
- 分布式数据库优点：
  - 分布式控制：减少通信开销
  - 数据共享：全局和局部
  - 可靠性和可用性加强：可以根据冗余数据恢复某一场地数据
  - 性能改善：数据分片，减少资源竞争
  - 可扩充性好：因分布独立性，不会影响用户程序。
- 分布式数据库缺点：
  - 复杂：主要是自治与协调。
  - 开销大：硬件、通信、冗余开销、安全性、完整性和并发控制开销。
- 分布式数据库体系结构：（若干局部数据模式＋一个全局数据模式）
  - 全局数据模式
    - 全局外模式

        - 全局概念模式
        - 分片模式：水平、垂直、混合分片；完全性、可重构性、不相交性。
        - 分布模式
    - 局部数据模式
        - 局部外模式
        - 局部概念模式
        - 局部内模式
    - 四级映象
        - 全局外模式与全局概念模式之间对应关系。
        - 全局概念模式（关系）与全局分片模式（片断）之间对应关系。
        - 全局分片模式（片断）与全局分布模式（网络结点）之间的对应关系。
        - 全局分布模式与局部概念模式之间的映射。

8.2.2 并行数据库

- 并行计算机体系类型：
    - 紧耦合全对称多处理器（SMP）所有 CPU 共享内存与磁盘（share memory）。
    - 松耦合群集机系统，所有 CPU 共享磁盘（share disk）
    - 大规模并行处理（MPP），所有 CPU 有自己内存和磁盘无共享资源（share nothing）

- 相应并行数据库的体系结构：
    - 共享内存（share memory）（SMP）
    - 共享磁盘（share disk）
    - 无共享资源（share nothing）
- 并行数据库的粒度：
    - 不同事务间并行
    - 同一事务不同查询间并行
    - 同一查询不同操作间并行
    - 同一操作内并行

8.2.3 多媒体数据库

- 主要特征：
    - 能够表示多种媒体数据。
    - 能够协调处理各种媒体数据。
    - 应提供比传统数据管理系统更强的适合非格式化数据查询的搜索功能。
    - 应提供特种事务处理与版本管理能力。

8.2.4 主动数据库

- 目标是提供紧急情况及时反应能力，同时提高数据库模块化程度。

- 通常采用方法是在传统数据库中嵌入 ECA（事件－条件－动作）规则。
- 要解决的问题：
  - 数据模型和知识模型
  - 执行模型
  - 条件检测
  - 事务调度
  - 体系结构
  - 系统效率

8.2.5 对象－关系数据库

- 对象－关系数据库兼有关系数据库和面向对象数据库的两方面特征。即在关系数据库基础上提供一下功能：
  - 允许扩充基本数据模型
  - SQL 中支持负责对象
  - 支持子类对超类的特性继承
  - 提供通用规则系统

8.3 面向应用的数据库新技术

8.3.1 数据仓库

- 数据库技术发展，从事务处理到决策分析。
- 数据库处理分两类：操作型（事务处理）和分析型（决策分析）。
- 数据仓库是体系化环境的核心，是建立决策支持系统（DSS）的基础。
- 从数据库到数据仓库，数据库不适合分析型处理：
  - 事务处理与分析处理性能特性不同。
  - 数据集成问题
  - 数据动态集成
  - 历史数据
  - 数据综合问题

8.3.2 工程数据库

- 过程数据库是一种能存储和管理各种工程图形，并能为过程设计提供各种服务的数据库。适用于 CAD、CIM 等。

8.3.3 统计数据库

- 统计数据库是一种用来对统计数据进行存储、统计、分析的数据库系统。

8.3.4 空间数据库

- 空间数据库是以描述空间位置和点、线、面、体特征的拓扑结构的位置以及描述这些特征性能的属性数据为对象的数据库。