数据库

Cynthia

2024-11-25

Table of Contents

Chapter 1: 绪论

1.1. 数据库系统概述

1.1.1. 数据库系统的 4 个基本概念

1. 数据: 描述事物的符号记录。
2. 数据库: 是长期储存在计算机内、有组织的、可共享的大量数据的集合。
3. 数据库管理系统(DBMS): 计算机的基础软件, 主要包括以下功能:
   - 数据定义功能。提供数据定义语言(DDL)。
   - 数据组织、存储和管理。
   - 数据操纵功能。提供数据操纵语言(DML)。
4. 数据库系统(DBS): 构成要素:
   - 数据库
   - 数据库管理系统
   - 应用程序
   - 数据库管理员(DBA)

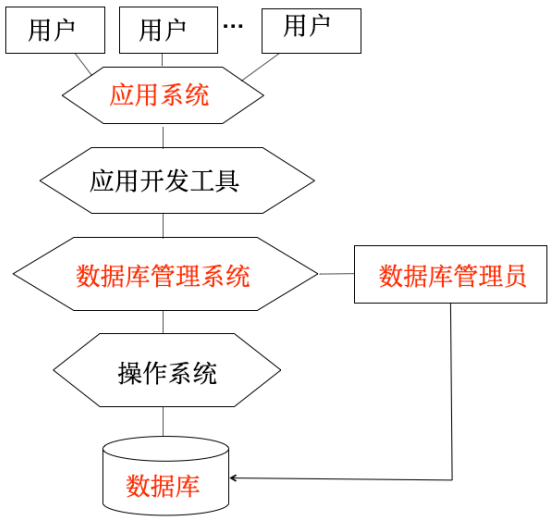

图 1.1 数据库系统

1.1.2. 数据库系统的特点

数据库系统的特点:

1. 数据结构化
   数据库系统实现整体数据的结构化, 这是数据库的主要特征之一, 也是与文件系统的主要区别之一。
2. 数据的共享性高, 冗余度低且易扩充
   数据面向整个系统, 可以被多个用户、多个应用共享使用
3. 数据独立性高

- 物理独立性
  指用户的应用程序与数据库中数据的物理存储是相互独立的。当数据的物理存储改变了，应用程序不用改变。
- 逻辑独立性
  指用户的应用程序与数据库的逻辑结构是相互独立的。数据的逻辑结构改变了，应用程序不用改变。
- 数据独立性由数据库管理系统的二级映像功能来保证。

4. 数据由数据库管理系统统一管理和控制
   DBS 提供的数据控制功能
   - 数据的安全性（Security）保护
   - 数据的完整性（Integrity）保护
   - 数据的并发（Concurrency）控制
   - 数据库恢复（Recovery）

1.2. 数据模型

数据模型就是现实世界的模拟，是数据库系统的核心和基础。

1.2.1. 两类数据模型

1. 概念模型
2. 逻辑模型和物理模型
   - 逻辑模型主要包括网状模型、层次模型、关系模型等，用于 DBMS 实现。
   - 物理模型是对数据最底层的抽象。

1.2.2. 概念模型

信息时间中基本概念:

1. 实体(entity): 客观存在并可以相互区别的事物。
2. 属性(attribute): 实体的特征。
3. 码(key): 唯一标识实体的属性集（可不仅包含一个属性）。
4. 实体型(entity type): 具有相同属性的实体集合。用实体名+属性名集合来抽象同类实体。
5. 实体集(entity set): 同一实体型的集合。如学生中有实体集——{张三、李四、王五}。
6. 联系(relationship): 实体之间的联系。
   - 实体内部的联系通常是指组成实体的各属性之间的联系
   - 实体之间的联系通常是指不同实体集之间的联系
   - 实体之间的联系有一对一、一对多和多对多等多种类型

实体-联系方法（Entity-Relationship Approach, E-R 方法）

1.2.3. 数据模型的三要素

1. 数据结构
2. 数据操纵（增删改查）
3. 数据的完整性约束条件

1.2.4. 层次模型

层次模型是数据库系统中最早出现的数学模型，用树形结构来表示各类实体以及实体间的联系

满足下面两个条件的基本层次联系的集合为层次模型

1. 有且只有一个结点没有双亲结点，这个结点称为根结点
2. 除根以外的其它结点有且只有一个双亲结点

层次模型的特点:

1. 结点的双亲是唯一的
2. 只能直接处理一对多的实体联系
3. 每个记录类型可以定义一个排序字段，也称为码字段
4. 任何记录值只有按其路径查看时，才能显出它的全部意义
5. 没有一个子女记录值能够脱离双亲记录值而独立存在

1.2.5. 网状模型

满足下面两个条件的基本层次联系的集合为网状模型:

1. 允许一个以上的结点无双亲;
2. 一个结点可以有多于一个的双亲。

用网状模型间接表示多对多联系

方法：将多对多联系直接分解成一对多联系

1.2.6. 关系模型

在用户观点下，关系模型中数据的逻辑结构是一张二维表，它由行和列组成。

1. 关系模型的数据结构
   - 关系（Relation）
     一个关系对应通常说的一张表
   - 元组（Tuple）
     表中的一行即为一个元组
   - 属性（Attribute）
     表中的一列即为一个属性
   - 主码（Key）
     也称码键。表中的某个属性组，它可以唯一确定一个元组

- 域（Domain）
  一组具有相同数据类型的值的集合。属性的取值范围来自某个域
- 分量
  元组中的一个属性值
- 关系模式(relation schema)
  关系名（属性 1，属性 2，...，属性 n）
  学生（学号，姓名，年龄，性别，系名，年级）

  最基本的规范条件：关系的每一个分量必须是一个不可分的数据项, 不允许表中还有表，即属性不可再分

2. 数据操纵（集合操作，增删改查）
3. 数据的完整性约束条件：实体完整性、参照完整性、用户定义的完整性

1.3. 数据库系统的结构

1.3.1. 数据库系统模式的概念

数据库系统通常采用三级模式结构，是数据库系统内部的系统结构

1.3.2. 数据库系统的三级模式结构

1. 模式（Schema） 也称“逻辑模式”
   - 数据库中全体数据的逻辑结构和特征的描述
   - 所有用户的公共数据视图
   - 一个数据库只有一个模式
   - 是数据库系统模式结构的中间层
2. 外模式（External Schema） 也称“子模式”(subschema)或“用户模式”
   - 数据库用户(包括应用程序员和最终用户)使用的局部数据的逻辑结构和特征的描述
   - 数据库用户的数据视图，是与某一应用有关的数据的逻辑表示
3. 内模式（Internal Schema） 也称“存储模式”
   - 是数据物理结构和存储方式的描述
   - 是数据在数据库内部的表示方式
   - 一个数据库只有一个内模式

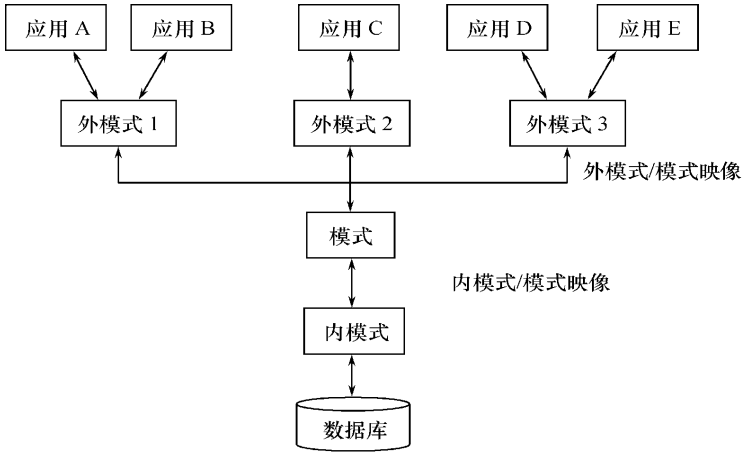

图 1.2　DBS 三级模式结构

1.3.3. 数据库的二级映像功能与数据独立性

二级映像:

1. 外模式/模式
2. 模式/内模式

保证了数据库外模式的稳定性。
从底层保证了应用程序的稳定性，除非应用需求本身发生变化，否则应用程序一般不需要修改

数据独立性:

1. 数据的逻辑独立性:　外模式/模式映像
2. 数据的物理独立性:　模式/内模式映像（数据库中模式与内模式映像唯一）

数据库模式，即全局逻辑结构，是数据库的中心与关键。其独立于数据库的其他层次，设计数据库模式结构时应首先确定数据库的逻辑模式。

1.4. 数据库系统的组成

人员:

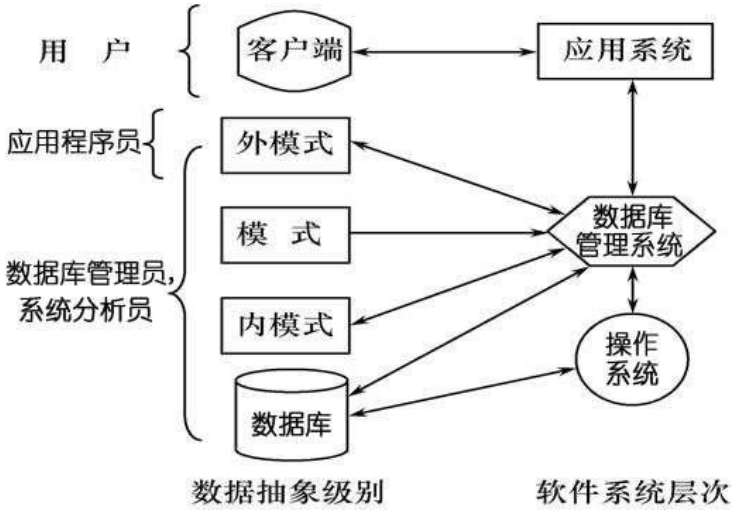
用　户
客户端
应用系统
应用程序员
外模式
模　式
数据库
管理系统
数据库管理员,
系统分析员
内模式
操作
系统
数据库
数据抽象级别
软件系统层次

图 1.3　数据库系统人员

Chapter 2: 关系数据库

2.1. 关系数据结构及形式化定义

- 单一的数据结构——关系 现实世界的实体以及实体间的各种联系均用关系来表示
- 逻辑结构——二维表 从用户角度，关系模型中数据的逻辑结构是一张二维表
- 建立在集合代数的基础上

关系数据结构的形式化定义:

1. 域(domain)
   域是一组具有相同数据类型的值的集合
2. 笛卡尔积(cartesian product)
   笛卡尔积是域上的一种集合运算。
   - 定义: $D_1 \times D_2 \times ... \times D_n = \{(d_1, d_2, ..., d_n) \mid d_i \in D_i, i = 1, 2, ..., n\}$
   - 基数: 一个域允许的不同取值个数
3. 关系(relation)
   关系是笛卡尔积的有限子集，表示为
   $R(D_1, D_2, ..., D_n)$, R为关系名，n为关系的目或度
   - 候选码: 关系中某一属性组的值能够唯一地标识一个元组，而其子集不能
   - 主码: 从候选码中选
   - 主属性: 候选码中的属性
   - 非主属性（非码属性）: 不包含在候选码中的属性
   - 全码: 关系模式中所有属性均为主属性

关系的类型

1. 基本关系（基本表/基表）: 实际存在的表，是实际存储数据的逻辑表示
2. 查询表: 查询结果对应的表，是临时表
3. 视图表: 由基本表或其它视图表导出的表，是虚表，不对应实际存储的数据

基本关系的性质

1. 列是同质的: 每列中的数据必须同一类型，来自同一个域
2. 不同列可以出自同一个域，每一列具有不同的属性名
3. 每一列必须是不可再分的数据项（规范化的基本要求）
4. 行和列次序无关
5. 任意两个元组的候选码不能相同

2.1.1. 关系模式

关系模式是型，关系是值（元组的集合）

- 关系模式包括关系名、诸属性名、属性向域的映像、属性间的依赖
  形式化表示: $R(U, D, \mathrm{DOM}, F)$
  - R 关系名
  - U 组成该关系的属性名集合
  - D U 中属性所来自的域
  - DOM 属性向域的映像集合
  - F 属性间数据的依赖关系的集合

  e.g.

$$\begin{aligned} D_1 &= \text{人名集合} \\ \mathrm{DOM}(\mathrm{Sname}) &= D_1 \end{aligned} \tag{2.1}$$

- 关系模式是静态的、稳定的
- 关系是动态的、随时间不断变化的

2.1.2. 关系数据库

关系数据库的型也称为关系数据库模式，关系数据库的值是这些关系模式在某时刻对应关系的集合，称为“关系数据库”

2.2. 关系操作

2.2.1. 基本的关系操作

常用的关系操作

- 查询: 选择、投影、连接、除、并、差、交、笛卡尔积
- 更新: 插入、删除、修改

查询的表达能力是其中最主要的部分

2.3. 关系的完整性

2.3.1. 实体完整性

> 规则 2.1 实体完整性规则
>
> 若属性 A 是关系 R 的主属性，则属性 A 不能取空值

2.3.2. 参照完整性

关系模型中使用关系描述实体间的联系，因此关系间必然存在引用

外码: 设F是基本关系R的一个或一组属性，但不是关系R的码。如果F与基本关系S的主码K_s相对应，则称 F 是基本关系 R 的外码

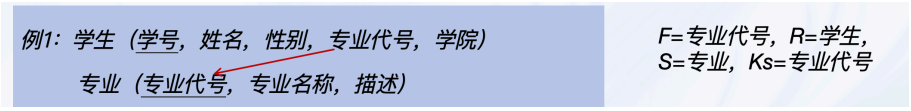

图 2.1　外码示例

规则 2.2 参照完整性规则

若属性组 F 是基本关系 R 的外码，它与基本关系 S 的主码 Ks 相对应（基本关系 R 和 S 不一定是不同的关系），则对于 R 中每个元组在 F 上的值必须为:

1. 或者取空值（F 的每个属性值均为空值）
2. 或者等于 S 中某个元组的主码值

2.3.3. 用户定义完整性

针对某一具体关系数据库的约束条件

2.4. 关系代数

关系代数是一种抽象的查询语言，用对关系的运算来表达查询

2.4.1. 传统的集合运算

设关系 R 和关系 S 具有相同的目 n，且相应的属性取自同一个域，

t是元组变量，$t \in R$表示 t 是 R 的一个元组

1. 并(Union)

$$R \cup S = \{t \mid t \in R \vee t \in S\} \tag{2.2}$$

结果仍为 n 目关系，一个属性、元素只出现一次

2. 差(except)

$$R - S = \{t \mid t \in R \vee t \notin S\} \tag{2.3}$$

结果仍为 n 目关系

3. 交(intersection)
4. 笛卡尔积

$(n+m)$ 目，共 $k_1 \times k_2$个元组

2.4.2. 专门的关系运算

1. 选择

$$\sigma_F(R) = \{t \mid t \in R \wedge tF(t) = \text{true}\} \tag{2.4}$$

F 表示选择条件，是一个逻辑表达式

2. 投影(projection)

从 R 中选择若干属性列组成新的关系

$$\Pi_A(R) = \{t[A] \mid t \in R\} \tag{2.5}$$

A 为 R 的属性列。投影后，删除重复行

3. 连接(join) 从两个笛卡尔积中选取属性间满足一定条件的元组

$$(R \bowtie S)_{A\theta B} = \left\{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A]\theta t_s[B] \right\} \tag{2.6}$$

A和B 列数相等且可比，θ 是比较运算符，选取 $R \times S$ 中满足比较的元组，其中 θ为 $=$ 是为“等值连接”。
自然连接是一种特殊的等值连接，比较分量必须是同名属性组，结果去除重复属性列

悬浮元组 做自然连接时，被舍弃的元组为悬浮元组。
如果把悬浮元组保留在结果关系中，而在其他属性值中填NULL，则为外连接；保留左边关系的悬浮元组为左外连接，保留右边的为右外连接

4. 除
   - 象集
     设关系$R(X,Y)$，X 与 Y 互为补集。
     象集$F(x) = R$中X分量为x的元组集合在Y上的投影
     即，$F(x) = \Pi_Y(\sigma_{X=x}(R))$
   - 除
     $P(X) = R(X,Y) \div S(Y,Z)$，其中 X，Y，Z 为属性组
     两个 Y 必须出自同一属性域
     P 是元组在 X 上分量值 x 的象集Y_x包含 S 在 Y 上投影的集合
     即，$R(X,Y) \div S(Y,Z) = \Pi_X\left(\sigma_{Y \in \Pi_Y(\sigma_{X=x}(S))}(R)\right)$

2.5. 关系演算

1. 一些聚集函数：COUNT, SUM, MAX, MIN, AVG 1) COUNT：计数
   计算学号为 001 的学生选修的课程数
   $\mathrm{COUNT}_{\mathrm{Cno}}(\sigma_{\mathrm{Sno}='001'}(SC))$
   2) SUM：求和
   求学号为 001 的学生的总成绩
   $\mathrm{SUM}_{\mathrm{Score}}(\sigma_{\mathrm{Sno}\ =\ '001'}(SC))$
2. 分组
   ${}_{G_i}G_{F_{i_{A_i}}}$ 其中，G_i 是分组的属性，F_i 是聚集函数，A_i 是属性名

   求每位同学的平均成绩:
   ${}_{\mathrm{Sno}}G_{\mathrm{AVG}_{\mathrm{Score}}}(SC)$

Chapter 3: 关系数据库标准语言 SQL

Chapter 3: 目录

3.1. 前言

SQL 语言核心动词:

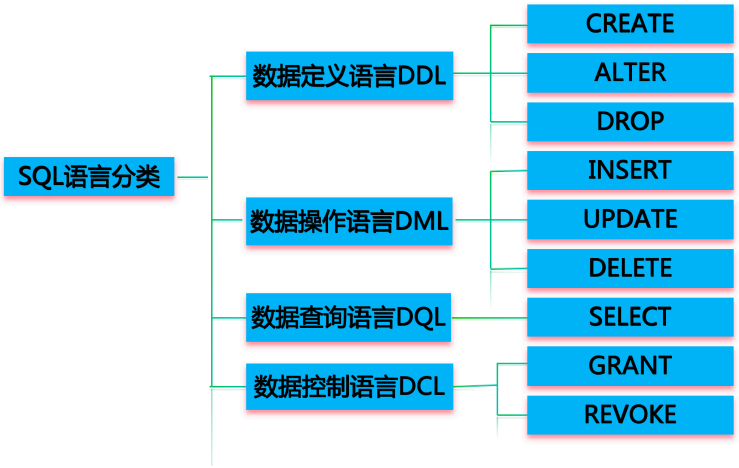

图 3.1 数据库语言分类

常用数据类型:

数据类型	变量类型
字符串类型	char(n), varchar(n), nchar(n), nvarchar(n), clob
数值类型	int, numeric(n), float
日期时间类	datetime, date, time
货币类型	money, smallmoney
布尔类型	bit
二进制类型	varbinary
XML 类型	

3.2. 数据定义语言 DDL

3.2.1. 数据库的定义

SQL 的数据定义语句

操作对象	操作方式		
	创建	删除	修改
数据库	create database	drop database	
模式	create schema	drop schema	
表	create table...	drop table xxx	alter table xxx ...
视图	create view xxx as ...	drop view xxx	
索引	create index xxx	drop index xxx	alter index xxx

1. 创建新数据表

```
create table 表名(
  列名 数据类型 [primary key] [default 缺省值][not null]
  [, unique (列名[,列名])]
  [, primary key(列名)]
  [, foreign key 列名 references 表名(列名)]
  [, check(条件)]
);
```

例如，建立一个学生信息表

```
CREATE TABLE Student (          -- 学生信息表
Sno char(12) PRIMARY KEY,       -- 学号
Sname varchar(30) NOT NULL,       -- 姓名
Ssex char(2) CONSTRAINT ck_Sex CHECK (SSex in ('男', '女')),   -- 性别
Sbirthdate smalldatetime,       -- 出生日期
SchoolID char(3),         -- 所属学院
CONSTRAINT FK_SchoolID FOREIGN KEY (SchoolID) REFERENCES
School(SchoolID));
```

2. 用查询结果创建表

```
create table 表名 as select * from 旧表名 [where 条件];
```

- 若新表已存在，创建失败
- 新表结构根据查询结果产生，但不包括约束条件

3. 修改表的定义

操作	语法
增加列	`alter table 表名 add [column] 列名 数据类型 [约束条件] [,...];`
修改列	`alter table 表名 modify [column] 列名 类型 [约束条件]`
删除列	`alter table 表名 drop [column] 列名 [, 列名];`
删除约束	`alter table 表名 drop constraint 约束名;`

例如，给学生的学院 id 设置缺省值 null:

```
alter table Student add default null for SchoolID;
```

4. 删除数据表

   `drop table [if exists] 表名;` 每次只能删除一张表

   `truncate table 表名;` 清空表中数据，只留结构

   `delete table 表名 where (条件);` 删除表中符合条件的数据

5. 创建索引

   `CREATE [UNIQUE] INDEX [索引名] ON 表名(列名 [次序] [, 列名 [次序]]…);`

   - unique/distinct: 唯一性索引，不允许表中不同的行在 unique 属性列上取相同值。若已有相同值存在，则系统给出相关信息，不建此索引。
   - 次序: ASC 或 DESC，索引表中索引值的排序次序，缺省为 ASC

6. 删除索引

   `DROP INDEX 索引名 ON 表名;`

   删除唯一性索引后，属性可以出现重复值

3.3. 数据操作语言 DML

1. 插入记录

   1) 常规插入

```
insert into 表名(列名 [, 列名])
values(值 [, 值])
[, (值, [值])];
```

可以省略列名的两个前提:

- 插入值的顺序与列的顺序完全一致
- 插入值包含了所有非空且没有默认值的列

2) 通过子查询插入记录

```
insert into 表名 [列名 [, 列名]] 子查询
```

将 employee 表中生日非空的职工 id 和姓名插入表 emp 中:

```
INSERT INTO emp
SELECT id, name
```

```
FROM employee
WHERE hiredate IS NOT NULL;
```

2. 更新记录

```
UPDATE 表名 SET 列名=新值 [,列名=新值...] [WHERE 条件];
```

3. 删除记录

```
DELETE FROM 表名 [WHERE 条件];
```

3.4. 数据查询语言 DQL

查询的一般格式:

```
SELECT [ALL | DISTINCT]
目标列表达式 [AS] [别名] [,目标列表达式 [别名]]...
FROM 表名/视图名 [别名] [,表名/视图名 [别名]]|[(子查询) 别名...]
[WHERE 条件表达式]
[GROUP BY 列名 [,列名…] [HAVING 条件表达式]]
[ORDER BY 列名 [ASC|DESC] [,列名 [ASC|DESC] ...]
[LIMIT 行数 [OFFSET 行数]];
```

1. 不带条件的查询

```
SELECT  目标列表达式 [AS] [别名] [,目标列表达式 [AS] [别名]]...
FROM  表名/视图名;
```

- 未设置别名时，以表达式为属性名
- 使用 distinct 消除重复行
- 聚集函数会忽略 null

使用字符串函数，MySQL 为例

```
SELECT
CONCAT(name,'_',sal) AS Name,
CHAR_LEN(Name) AS Len,
LOCATE('0',Name) AS Index,  -- 0在Name的索引值
SUBSTRING(Name,2,4) AS SubStr
FROM employee;
```

2. 带条件的查询

```
SELECT  目标列表达式 [AS] [别名] [,目标列表达式 [AS] [别名]]...
from 表名/视图名
where 条件表达式;
```

- 使用通配符进行字符串匹配时需要用 LIKE
  - %: 0 至多个字符
  - _: 单个字符
  - 字符串中包含 % 或_时，需要转义

```
select * from Student
where name like 'Jackie\_Chen' escape '\';
```

3. 多表连接查询

```
 SELECT [表名.]目标列表达式 [AS] [别名]
[, [表名.]目标列表达式 [AS] [别名]…]
 FROM 表名/视图名  [AS] [别名]
[,表名/视图名 [AS] [别名]…]
WHERE 选择条件/连接条件
```

- 交叉连接/笛卡尔积，Cross Join

```
WHERE School.SchoolName LIKE '计算机%'
```

- 内连接，Inner Join

```
WHERE Order.RoomID = Room.rID
WHERE e.Sal BETWEEN s.Lo AND s.High
```

- 自身连接，Self Join

```
WHERE e.Mgr = m.EmpNo
```

- 求并集 UNION
- 求交集 INTERSECT
- 求差集 EXCEPT

查询未选修课程的学号

a.
```
select Student.Sno
from Student
left join SC
on Student.Sno = SC.Sno
where Cno is null;
```

b.
```
select Sno
from Student
EXCEPT
select Sno
from SC
where Cno is null;
```

4. 子查询

- 子查询返回且仅返回一个值，可以使用单行比较符
- 多行子查询返回多个值，只能使用多行比较符

运算符	描述
in	等于结果中的任意一个值
any	与结果中的任意一个值做比较
all	与结果中的所有一个值做比较

可搭配 not 使用

>any ⟺ >MIN

>all ⟺ >MAX

- 使用 exists 检查行是否存在

查询与出版商在同一个城市的作者姓名

```
select Author.name
from Author a
where exists (
  select *
  from Publisher p
  where a.city = p.city;
)
```

- 子查询中使用聚集函数时，派生表要写属性名
- 子查询在 from 中，需要使用别名

5. 视图查询

```
CREATE VIEW 视图名 AS 子查询 [WITH CHECK OPTION];
```

必须使用别名的情况:

- 子查询中使用了聚集函数或者列表达式
- 子查询中存在同名列
- with check option 影响:
  - 更新后，数据要能够被视图查询出来，否则失败
  - 插入后，新数据要能够被视图查询出来，否则失败
  - 删除不受影响
  - 如果子查询未包含 WHERE 语句，则无影响
- 删除视图 `DROP VIEW 视图名;`

3.5. 数据控制语言 DCL

1. 授权（授权应根据最小化原则进行）

```
GRANT 权限[, 权限…] ON 对象类型 对象名[, 对象类型 对象名]
TO 用户名/用户组[, 用户名/用户组…]
  [WITH GRANT OPTION];
```

2. 撤权

```
REVOKE 权限[, 权限…] ON 对象类型 对象名[, 对象类型 对象名]
  FROM 用户名/用户组[, 用户名/用户组…]
[CASCADE | RESTRICT];
```

- cascade:级联撤销，与之相关的都撤销
- restrict:只撤销无依赖的权限

权限包含: select, update, insert, delete, all...

对象类型包含: table, column, view...

3.6. 注意事项

1. SQL 书写顺序
   SELECT⟶ FROM⟶ WHERE⟶ GROUP BY⟶ HAVING⟶ ORDER BY⟶ LIMIT
   WHERE 用于筛选原始数据，分组过后若想进行条件判断，只能用 HAVING
2. HAVING 中出现的对象除了聚集函数，只能是作为分组依据的属性

Chapter 4: 数据库安全性

Chapter 4: 目录

4.1. 数据库安全概述

4.1.1. 数据库不安全因素

1. 问题的提出
   - 数据库安全性是由数据共享所引起的
   - 数据库安全性是指保护数据库以防止不合法使用所造成的数据泄露、篡改或破坏
   - 系统安全保护措施是否有效是数据库系统主要的指标之一
2. 不安全因素
   - 非授权用户对数据库的恶意存取和破坏
     - 数据库管理系统提供的安全措施主要包括用户身份鉴别、存取控制和视图等技术
   - 数据库中重要或敏感的数据被泄露
     - 数据库管理系统提供的主要技术有强制存取控制、数据加密存储和加密传输等。
     - 审计日志分析
   - 安全环境的脆弱性
     - 计算机系统的安全性
     - 计算机硬件、操作系统、网络系统等的安全性
     - 建立一套可信（Trusted）计算机系统的概念和安全标准

4.1.2. 安全标准简介

TCSEC/TDI 将系统分为 4 组 7 个等级

A1	验证设计，给出形式化设计说明与验证
B3	安全域，数据隐藏与分层、屏蔽
B2	结构化保护，支持硬件保护
B1	标记安全保护
C2	有自主的访问安全性，区分用户（安全产品最低档次）
C1	不区分用户，基本的访问控制
D	没有安全性可言

4.2. 数据库安全控制

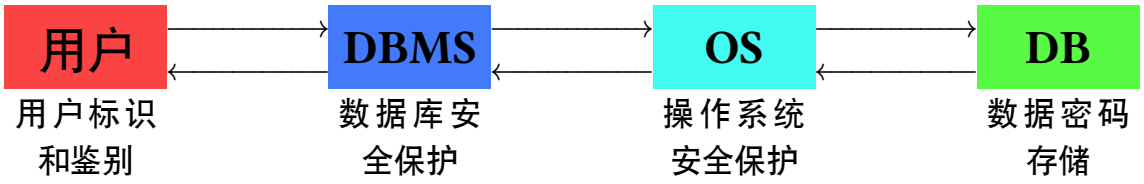

图 4.1 计算机系统的安全模型

数据库安全性控制的常用方法

- 用户身份鉴别,在用户登陆时，由 DBMS 对用户身份进行核对
- 存取控制: DAC、MAC

4.2.1. 自主存取控制方法（DAC）

- C2 级
- 用户对不同的数据对象有不同的存取权限
- 不同的用户对同一对象也有不同的存取权限
- 用户还可将其拥有的存取权限转授给其他用户-灵活

通过 SQL 中的 GRANT 和 REVOKE 实现 DAC

对象类型	对象	操作类型
数据库模式	模式	CREATE SCHEMA
	基本表	CREATE TABLE, ALTER TABLE
	视图	CREATE VIEW
	索引	CREATE INDEX
数据	基本表和视图	SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALL PRIVILEGES
	属性列	同上

1. 授权 GRANT

a. 把 Student 表和 Course 表的所有权限分配给 User2 和 User3

```
GRANT all ON Table Student, Course TO User2, User3;
```

b. 把查询 SC 表和修改学号的权限分配给 User4，并允许授权

```
GRANT update(Sno), select
  ON Table SC TO User4
  WITH GRANT OPTION;
```

2. 撤权 REVOKE

a. – 把 User1 修改学生学号的权限收回

```
REVOKE update(Sno) ON Table Student FROM User1;
```

b. – 把 User5 插入 SC 表的权限收回

```
REVOKE insert
  ON Table SC
  FROM User5 CASCADE;
```

3. 创建数据库用户
   - 只有超级用户可以创建新的数据库用户
   -
   ```
   CREATE USER 用户名
   [WITH]
   [CONNECT | RESOURCE | DBA];
   ```
     - CONNECT：只能登陆数据库
     - RESOURCE: 可以创建基本表和视图，成为所创建对象的属主
     - DBA: 系统的超级用户。
4. 创建数据库角色
   - 角色是被命名的一组和数据库操作相关的权限
   -
   ```
   CREATE ROLE 角色名; -- 创建空角色

   GRANT 权限          -- 把权限授予角色
     ON 对象类型 对象名
     TO 角色名;

   GRANT 角色名        -- 把角色授予用户（相当于把权限集合授予USER）
     TO 用户名
     [WITH ADMIN OPTION];
   ```
5. 撤销角色权限、回收用户角色
   -
   ```
   DROP USER 用户名;    -- 删除用户
   DROP ROLE 角色名;    -- 删除角色

   REVOKE 权限          -- 回收角色权限
     ON 对象类型 对象名
     FROM 角色名;

   REVOKE 角色名        -- 回收用户角色
   ```

```
FROM 用户名;
```

DAC 无法解决数据泄露问题

4.2.2. 强制存取控制方法（MAC）

- B1 级
- 每一个数据对象被标以一定的密级
- 每一个用户也被授予某一个级别的许可证
- 对于任意一个对象，只有具有合法许可证的用户才可以存取

对于主体和客体，DBMS 为它们每个实例（值）指派一个敏感度标记

绝密（Top Secret，TS）
机密（Secret，S）
可信（Confidential，C）
公开（Public，P）

↓ 敏感度下降，$TS > S > C > P$

- 主体的敏感度标记称为许可证级别
- 客体的敏感度标记称为密级
- 强制存取规则
  - 仅当主体许可证级别大于或等于客体密级时，该主体才能读取相应的客体
  - 仅当主体许可证级别等于（或小于等于）客体密级时，该主体才能写相应的客体
  - 上写下读
  - 特点：禁止了拥有高许可证级别的主体更新低密级的数据对象，从而防止了敏感数据的泄露

将 M 表中数据氛围 H 和 L 两部分，设置密级（客体）$S > P$
管理层、老板、员工许可证设置为 $S > C > P$

MAC	H(S)		L(P)	
	读	写	读	写
老板（C）	×	✓	✓	×
管理层（S）	✓	✓	✓	×
员工（P）	×	✓	✓	✓

安全检查：DAC 检查 $\longrightarrow$ MAC 检查

4.3. 视图机制

- 把要保密的数据对无权存取这些数据的用户隐藏起来
- 间接地实现支持存取谓词的用户权限定义

创建视图，把对视图的权限授予用户

4.4. 审计加密等安全性保护

4.4.1. 审计

1. 特征描述:
   a. C2 及以上安全级别的 DBMS 必须具备的功能
   b. 审计功能把用户对数据库的所有操作自动记录下来，放到审计日志
   c. DBA 灵活开关审计功能
2. 审计事件:
   a. 服务器事件
      审计数据库服务器发生的事件，数据库启动、停止、配置文件重新加载等
   b. 系统权限
      - 对系统拥有的结构或模式对象进行操作的审计
      - 要求该操作的权限是通过系统权限获得的
   c. 语句事件
      对 SQL 语句，如 DDL、DML、DQL 及 DCL 语句的审计
   d. 模式对象事件
      对特定模式对象（表、视图、存储过程、函数等）上进行的 SELECT 或 DML 操作的审计
3. 审计实施

```
-- 对修改SC表结构或修改SC表数据的操作进行审计
AUDIT ALTER, UPDATE ON SC

-- 取消对SC表的审计
NOAUDIT ALTER, UPDATE ON SC
```

4.4.2. 数据加密

4.4.3. 其他方法

Chapter 5: 数据库完整性

5.1. 实体完整性

要点:

1. 一个表只能有一个码
2. 主码的值非空且唯一
3. 对单属性构成的码，可定义在列/表级；多属性的必须在表级

5.2. 参照完整性

要点:

1. 外码值须在原表存在
2. 或者设置为空（但要考虑本表中外码是否设置为 NOT NULL）

参照完整性违约处理

```
-- 显示定义参照完整性的违约处理
CREATE TABLE SC (
  Sno   CHAR(9)  NOT NULL,
  Cno   CHAR(4)  NOT NULL,
  Grade SMALLINT,
  PRIMARY KEY (Sno, Cno),
  FOREIGN KEY (Sno) REFERENCES Student(Sno)
    ON DELETE CASCADE /*级联删除SC表中相应的元组*/
    ON UPDATE CASCADE,  /*级联更新SC表中相应的元组*/
  FOREIGN KEY (Cno) REFERENCES Course(Cno)
    ON DELETE NO ACTION /*拒绝删除SC表中相应的元组*/
    ON UPDATE CASCADE /*级联更新SC表中相应的元组*/
```

5.3. 用户定义的完整性

用户自定义的约束，如 `Score INT CHECK (Score IN BETWEEN 0 AND 100)`（BETWEEN...AND...取闭区间）

- 插入元组或修改属性的值时，检查约束条件是否被满足
- 如果不满足则拒绝执行操作（NO ACTION）

5.4. 完整性约束命名子句

断言:

```
CREATE ASSERTION 断言名 CHECK子句
DROP ASSERTION 断言名
```

```
-- 限制选修数据库这门课的学生数不超过60
CREATE ASSERTION ASSE_SC_DB_Num
```

```
    CHECK (60 >= (SELECT COUNT(*)
            FROM SC
            JOIN Course
            ON SC.Cno = Course.Cno
            WHERE Course.Cname='数据库')
);
```

断言增加了数据与业务的耦合

5.5. 触发器

1. 用户定义在关系表（不可为 VIEW）上的由事件驱动的特殊过程
2. 用编程的方法实现复杂的商业规则，它可以实现一般的数据完整性约束实现不了的复杂的完整性约束
3. 不需要由用户调用执行，而是当用户对表中的数据进行 UPDATE、INSERT 或 DELETE 操作时自动触发执行的
4. 表的拥有者才可以在表上创建触发器

5.5.1. 定义触发器

```
CREATE TRIGGER 触发器名
  {BEFORE | AFTER} 触发器事件 ON 表名
  REFERENCING NEW | OLD ROW | TABLE AS 变量名
  FOR EACH {ROW | STATEMENT}
  [WHEN 触发条件] 触发动作体
```

注意:

1. 同一模式下，触发器名必须唯一
2. 触发器名和表名必须在同一模式下
3. 触发器事件
   - INSERT、UPDATE、DELETE，或是这几个事件的组合，如 INSERT OR UPDATE
   - 也可以指定具体的列，如 UPDATE OF <触发列, ...>
4. BEFORE、AFTER 指定是在触发器事件执行前或后激活触发器
5. 触发器类型
   - 行级触发器（For Each Row）：以行为单位激活触发器
   - 语句级触发器（For Each Statement）：以 SQL 语句为单位激活触发器
6. 变量
   - 行级触发器可以将新行/旧行定义为变量，在触发动作体中使用
   - 语句级触发器可以将新表/旧表定义为变量，在触发动作体中使用
7. 触发条件
   触发器激活后，可以进一步检查触发条件，以确定是否执行触发动作体
8. 触发动作体

- 可以是匿名的过程块，也可以调用存储过程
- 触发动作体执行失败会导致激活触发器的事件被终止执行

```
-- 修改学生分数时，自动保存修改前后的分数
CREATE TRIGGER SC_T
  AFTER UPDATE OF Grade ON SC
  REFERENCING NEW ROW AS New, OLD ROW AS Old
  FOR EACH ROW
  WHEN (New.Grade >= 1.1 * Old.Grade)
    (INSERT INTO SC_U(Sno, Cno, OldGrade, NewGrade)
      VALUES (Old.Sno, Old.Cno, Old.Grade, New.Grade)
  );

-- 自动保存每次插入后的学生总数
CREATE TRIGGER Student_Count
  AFTER INSERT ON Student
  REFERENCING NEW TABLE AS NewStu
  FOR EACH STATEMENT
    (INSERT INTO Log(Numbers)
      SELECT COUNT(*) FROM NewStu
    );
```

5.5.2. 激活触发器

由触发事件激活，并由数据库服务器自动执行

执行顺序

1. 执行该表上的 BEFORE 触发器
2. 执行激活触发器的 SQL 语句
3. 执行该表上的 AFTER 触发器

5.5.3. 删除触发器

SQL 语句:

```
DROP TRIGGER <触发器名> ON <表名>
```

- 必须是已经创建的触发器
- 只能由具有相应权限的用户删除

Chapter 6: 关系数据理论

Chapter 6: 目录

6.1. 问题的提出

如何评价一个关系模式的优劣？ 好的关系模式:

1. 不会发生插入异常、删除异常、更新异常
2. 数据冗余应尽可能少

关系模式组成: R(U, D, DOM, F)

简化为: R(U, F)

6.1.1. 数据与函数依赖

1. 数据依赖
   - 数据内部属性与属性之间的约束
   - 属性间值的相等与否体现出来的数据间的相互关系
   - 现实世界属性间相互联系的抽象
   - 数据的内在性质
   - 语义的体现
2. 函数依赖
   - X 属性集决定 Y 属性集时， 称 Y 属性集依赖于 X 属性集； 记为 $X \rightarrow Y$
   - 表述唯一决定关系， 即只要 X 的具体取值确定了， 就能确定唯一的 Y 值
   - 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

函数依赖就是唯一确定的关系

学生信息（学号， 所在学院， 院长姓名， 课程名， 成绩）

Student<U, F>

U = { Sno, Sdept, Mname, Cname, Grade }

F = ?

一般语义:

- 一个学生只属于一个学院 $\longrightarrow$ Sno→Sdept
- 一个学院有若干学生 $\longrightarrow$ Sno→Sdept
- 一个学院只有一名院长 $\longrightarrow$ Sdept→Mname

- 一个学生可以选修多门课程 $\longrightarrow$ None
- 每门课程有若干学生选修 $\longrightarrow$ None
- 每个学生所学的每门课程都有一个成绩 $\longrightarrow$ (Sno, Cname)$\rightarrow$Grade

  得到 $F = \{\text{Sno} \rightarrow \text{Sdept}, \text{Sdept} \rightarrow \text{Mname}, (\text{Sno}, \text{Cname}) \rightarrow \text{Grade}\}$

但这个模式存在很多问题:

1. 数据冗余
2. 更新异常
   (某学院更换院长后，需要修改该学院学生相关的所有元组)
3. 插入异常
   (例: 新成立学院，还没有学生，则无法保存学院和院长信息)
4. 删除异常
   (例: 某学院学生全部毕业，删除这些学生时，学院及院长信息也丢失了)

6.2. 函数依赖及候选码

6.2.1. 函数依赖的基本概念

函数依赖指R的所有关系实例均要满足的约束条件

6.2.2. 函数依赖的分类

1. 平凡函数依赖与非平凡函数依赖
   在关系模式 R(U)中，对于 U 的子集 X 和 Y
   - 如果 X 决定 Y，且$Y \subseteq X$，则称$X \rightarrow Y$是平凡函数依赖
   - 如果 X 决定 Y，但$Y \not\subseteq X$，则称$X \rightarrow Y$是非平凡函数依赖

   e.g. 学生(学号, 课程号, 成绩)
   - 平凡函数依赖: (学号, 课程号) $\rightarrow$ 学号， (学号, 课程号) $\rightarrow$ 课程号
   - 非平凡函数依赖: (学号, 课程号) $\rightarrow$ 成绩

   ps:任一关系模式，平凡函数依赖都是必然成立的，它不反映新的语义。无特殊说明，都是讨论非平凡函数依赖。
2. 完全函数依赖与部分函数依赖
   - 如果X决定Y，且 X 的任一真子集X'不决定Y, 则称$X \rightarrow Y$是完全函数依赖 F
   - 如果X决定Y，同时存在某一真子集X'决定Y ，则称$X \rightarrow Y$是部分函数依赖 P

   学生(学号, 年龄, 课程号, 成绩)
   - 完全函数依赖: $(\text{学号}, \text{课程号}) \overset{F}{\rightarrow}$ 成绩，因为学号 $\not\rightarrow$ 成绩且课程号 $\not\rightarrow$ 成绩

- 部分函数依赖：$(\text{学号}, \text{课程号}) \xrightarrow{P}$ 年龄，因为学号 $\rightarrow$ 年龄

3. 传递函数依赖与直接函数依赖
   在关系模式$R(U)$中，Y非平凡函数依赖X（即$\boldsymbol{X} \rightarrow \boldsymbol{Y}, \boldsymbol{Y} \not\subseteq \boldsymbol{X}$），$Z$非平凡函数依赖$Y$（即$\boldsymbol{Y} \rightarrow \boldsymbol{Z}, \boldsymbol{Z} \not\subseteq \boldsymbol{Y}$）
   - 如果同时X不函数依赖Y（$\boldsymbol{Y} \nrightarrow \boldsymbol{X}$），称$Z$传递函数依赖于$X$
   - 如果同时 X 函数依赖Y（$\boldsymbol{Y} \rightarrow \boldsymbol{X}$），即$\boldsymbol{X} \leftrightarrow \boldsymbol{Y}$，称$Z$直接函数依赖于$X$

   车辆(车架号, 车牌号, 型号, 座位数)
   - 传递函数依赖：车架号 $\rightarrow$ 型号，型号 $\rightarrow$ 座位数，所以车架号 $\rightarrow$ 座位数
   - 直接函数依赖：车架号 $\leftrightarrow$ 车牌号，车架号 $\rightarrow$ 型号，所以车牌号 $\rightarrow$ 型号

6.2.3. 码的概念及快速求候选码

1. 码的概念
   定义：设K为关系模式$R<U, F>$中的属性或属性组合。若$K \xrightarrow{F} U$，则K称为R的一个侯选码（Candidate Key）。若关系模式R有多个候选码，则选定其中的一个做为主码（Primary key）
   - 主属性：候选码中包含的属性称为主属性
   - 非主属性：不包含在任何一个候选码中的属性
   - 全码：整个属性组都是码

   因此，找候选码等价于找关系模式中的完全函数依赖

   考虑这样一个关系：SLC (Sno, Sdept, Sloc, Cno, Grade)
   假设每个系的学生住同一个地方，确定主码
   a. 分析关系F
      - $S \rightarrow$ Sdept
      - Sdept $\rightarrow$ Sloc
      - (Sno, Cno) $\rightarrow$ Score
   b. 从主属性最多的候选码开始，直到完成闭包
      - 选定(Sno, Cno)
      - 发现(Sno Cno) $+ = U$，则其为主码

2. 快速求候选码
   对于给定的 R(U)，分析函数依赖集 F，进行属性分类
   - **L 类**：仅出现在 F 的函数依赖左部的属性，必为主属性
   - **R 类**：仅出现在 F 的函数依赖右部的属性，不为主属性
   - **N 类**：在 F 的函数依赖左部和右部均未出现的属性，必为主属性
   - **LR 类**：在 F 的函数依赖左部和右部同时均出现的属性，可为主属性

属性分类 → 确定主属性 → 计算闭包

ps: 如果L和N属性集X，$X+=U$，则其为唯一候选码

6.3. 规范化理论

各范式关系

1NF ⊃ 2NF ⊃ 3NF ⊃ BCNF ⊃ 4NF

某一关系模式R为第n范式，可简记为$R \in$ nNF

6.3.1. 第一范式

定义：R 的所有属性都是不可分的基本数据项，则$R \in$ 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

会有插入、更新、删除异常和数据冗余（源于非主属性对码的部分函数依赖）

6.3.2. 第二范式

定义：每一个非主属性都完全函数依赖于 R 的码，则$R \in$ 2NF

会有更新异常和数据冗余（源于非主属性对码的传递函数依赖）

6.3.3. 第三范式

定义：每一个非主属性都直接函数依赖于 R 的码，则$R \in$ 3NF

可能也会导致数据冗余等，原因在于：主属性的直接完全函数依赖中，存在非码的决定因素

对于仓库表（仓库，管理员，物品，数量），若选择码为(仓库, 物品)

1. 存在(仓库, 物品) → 管理员，即管理员作为主属性部分依赖于不包含它的候选码 (仓库, 物品)
2. 存在管理员 → 仓库，即仓库作为主属性完全函数依赖于非码的属性管理员

6.3.4. BC 范式

定义：每一个决定属性都直接完全函数依赖于码，则$R \in$ BCNF

- BCNF 要求所有主属性都完全函数依赖于每个不包含它的候选码
- BCNF 要求没有任何属性完全函数依赖于非码的任何一组属性
- 在函数依赖范畴内，BCNF 为最高范式

6.3.5. 范式判断

1. 是否满足 1NF 要求：检查基本数据项
   - 可分⇒不满足，不可分⇒满足
2. 是否满足 2NF 要求：检查非主属性对码的依赖关系
   - 存在非主属性部分函数依赖于码⇒不满足，不存在⇒满足

3. 是否满足 3NF 要求: 检查非主属性对码的依赖关系
   - 存在非主属性传递函数依赖于码⇒不满足，不存在⇒满足
4. 是否满足 BCNF 要求: 检查所有依赖关系的决定因素
   - 唯一候选码=满足
   - 存在非码的决定因素⇒不满足，不存在⇒满足

6.4. 规范化程度的选择

Chapter 7: 关系数据库设计

7.1. 数据库设计概述

1. 数据库和应用系统
   数据库是应用系统的核心和基础
2. 数据库设计概念
   定义：数据库设计是指对于一个给定的应用环境，构造（设计）优化的数据库模式、外模式和内模式，并据此建立数据库及其应用系统，使之能够有效地存储和管理数据，满足各种用户的应用需求

7.2. 需求分析

1. 数据字典是对数据库中数据的描述，不是数据本身。其通过对数据项和数据结构的定义来描述数据流、数据存储的逻辑内容。
2. 数据项是不可再分的数据单位
3. 数据结构反映了数据之间的组合关系（类）
4. 数据流是数据结构在系统内传输的路径

7.3. 概念结构设计

E-R 图是概念模型

1. E-R 模型与 E-R 图
   - E-R 模型是用 E-R 图来描述现实世界的概念模型
     - 实体型（矩形）
     - 属性（椭圆）
     - 实体型之间的联系（菱形）联系只发生在实体之间
2. 联系类型
   - 两个实体之间
     - 1:1
     - 1:n
     - m:n
   - 单个实体内
     - 1:1
     - 1:n
     - m:n
   - 多个实体之间
     - 1:1:1
     - 1:m:n
     - m:n:p
3. E-R 图的集成
   - 合并，消除三类冲突

  - 属性冲突（行政手段解决）
  - 命名冲突（行政手段解决）
  - 结构冲突
- 修改于重构，消除不必要冗余

4. 要点:
   - 实体要标明自己的码
   - 标明联系类型
   - 联系的属性
   - 区分属性和枚举（枚举可归为一个属性，若没给出属性）

7.4. 逻辑结构设计

E-R 图向关系模型的转换，逻辑模型是表结构 转换:

1. 实体型
   - 关系的属性对应实体型的属性
   - 关系的码对应实体型的码
2. 相同码的关系可以合并
   将一个关系模式的全部属性加入到另一个关系模式中，去掉其中的同义属性(可能同名也可能不同名)，并调整属性的次序

联系/转换	联系		实体
1:1	属性	联系两端码+联系属性	原属性+另一实体码+联系属性
	码	某端实体型码	本身码
1:n	属性	联系两端码+联系属性	N 端原属性+1 端码+联系属性
	码	N 端实体型的码	本身码（N 端）
m:n	属性	联系两端码+联系属性	原属性
	码	两端实体性码组合	本身码

表 7.1　不同联系类型转换原则

7.5. 物理结构设计

7.6. 数据库实施和维护

数据库的重组和重构

- 数据库重组分为:
  - 索引的重组：调整不合理的索引，增加新的索引
  - 数据库/表空间的重组：调整存储方式等
- 数据库重构是指根据新环境调整数据库的模式和内模式
  - 增加新的数据项
  - 改变数据项的类型
  - 改变数据库的容量

- 增加或删除索引
- 修改完整性约束条件
- 表的拆分与合并

ps: 数据库重构意味着应用程序、数据库都需要改变

Chapter 8: 关系查询处理和查询优化

8.1. 查询处理

8.1.1. 查询处理的四个阶段

1. 查询分析
   - 词法分析
   - 语法分析
2. 查询检查
   - 语义分析
   - 符号名转换
   - 安全性检查
   - 完整性检查
3. 查询优化
   - 代数优化（对关系代数表达式进行等价变换）
   - 物理优化（存取路径和底层操作算法的选择）
4. 查询执行
   - 代码生成

8.1.2. 选择操作算法的实现

1. 全表扫描
   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
2. 索引扫描
   - 适合选择条件中的属性上有索引的情况（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找到元组

8.1.3. 连接操作算法的实现

了解

连接操作是查询处理中最常用最耗时的操作之一 这里只讨论等值连接（或自然连接）最常用的实现算法

1. 嵌套循环算法（Nested Loop Join）
   - 对外层循环的每一个元组，检索内层循环中的每一个元组
   - 检查这两个元组在连接属性上是否相等
   - 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
2. 排序-合并算法（Sort-Merge Join）
   - 先对两个表按照连接属性排序，然后进行合并输出

3. 索引连接算法（Index Join）
   - 在 b 表上建立连接属性的索引
   a. 对 a 表中每一个元组，由连接属性值通过索引查找 b 表上的元组
   b. 把这些 a 表的元组和 b 表的元组连接起来
   c. 循环执行②③，直到 a 表中的元组处理完为止
4. 哈希连接算法
   - 对较少元组的表把它的元组按 hash 函数分散到 hash 表的桶中
   - 试探阶段：也称连接阶段，对另一个表进行一遍处理，把元组散列到适当的 hash 桶中，把元组与桶中与之相匹配的元组连接起来

8.2. 查询优化

8.2.1. 执行代价估算

1. 集中式数据库:
   总代价 = I/O 代价+ CPU 代价+ 内存访问
2. 分布式数据库:
   总代价 = I/O 代价 + CPU 代价 + 内存访问 + 通信代价

代数优化:

假设 1: Stu 有 1000 条记录，SC 有 10000 条记录，选修 C2 号课程有 50 条记录
假设 2: 一个存储块可以装 10 个 Stu 元组或 100 个 SC 元组或 10 个结果元组
假设 3: 内存一次可以存放 5 块 Stu 元组，1 块 SC 元组和 n 块结果元组
假设 4: 读写速度: 20 块/秒
假设 5: 连接方法: 基于数据块的嵌套循环法

1. $Q_1 = \Pi_{\text{Sname}}(\sigma_{\text{Stu.Sno=SC.Sno^SC.Sno='2'}}(\text{Stu} \times \text{SC}))$
   a. 笛卡尔积
   Stu 块数 $= \frac{1000}{10} = 100$块
   SC 块数 $= \frac{10000}{100} = 100$块
   读取总块数 $= 100(\text{Stu 表}) + \frac{100}{5}(\text{Stu 加载到内存次数}) \times 100(\text{SC 表块数}) = 2100$
   嵌套循环读取时间 $= \frac{2100}{20} = 105s$
   中间结果大小 $= 1,000 * 10,000 = 10,000,000$条
   写中间结果时间 $= 10,000, \frac{\frac{000}{10}}{20} = 50,000s$
   b. 选择操作
   读中间结果时间 = 写中间结果时间 $= 50,000s$
   c. 投影
   时间忽略不计
   总时间 $= 100,105s = 27.8h$

2. $Q_2 = \Pi_{\text{Sname}}(\sigma_{\text{SC.Sno}='2'}(\text{Stu} \bowtie \text{SC}))$
3. $Q_3 = \Pi_{\text{Sname}}(\text{Stu} \bowtie \sigma_{\text{SC.Sno}='2'}(\text{SC}))$

在属性列上建立索引是物理优化

8.3. 代数优化

8.3.1. 等价变换规则

8.3.2. 代数优化的一般准则

准则:

1. 优化中间结果（最重要最基本）
   a. 分解选择运算
   利用规则 4 把形如 σF1 ^ F2 (E) 变换为 σF1 (σF2 (E))
   b. 通过交换选择运算，尽可能将选择移到叶端
   对每一个选择运算，利用规则 4 ~ 9（选择串接律、选择投影交换律、选择与笛卡尔积交换、选择与并分配、选择与差分配、选择与自然连接分配）尽可能把它移到树的叶端
   c. 通过交换投影运算，尽可能将投影移到叶端
   对每一个投影运算，利用规则 3，5，10，11 尽可能把它移向树的叶端
2. 减少重复扫描
   - 投影 & 选择同时
   - 投影与前后双目运算结合

   a. 选择等+笛卡尔积 → 自然连接
3. 加快连接速度
   - 被连接属性排序、建立索引

8.3.3. 关系代数表达式的优化算法

1. 笛卡尔积换成等值连接等
2. 连接前先选择
3. 选择后先投影
4. 连接前先投影

8.4. 物理优化

8.5. SQL 优化

Chapter 9: 数据库恢复技术

9.1. 数据库恢复中的基本概念

9.1.1. 事务的概念及 ACID 特性

1. 事务的概念
   - 用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位
     - 一个事务可以是一条 SQL 语句、一组 SQL 语句或整个程序
     - 一般来说，一个程序通常包含多个事务
   - 事务是恢复和并发控制的基本单位
2. 定义事务
   - 隐式：default
   - 显式：

```
BEGIN TRANSACTION;  -- 事务正常结束，提交所有操作
SQL语句...
COMMMIT; -- 提交

BEGIN TRANSACTION;  -- 事务异常终止，撤销已完成，数据回到初始状态
SQL语句...
ROLLBACK; -- 回滚
```

3. 事务的 ACID 特性
   - 原子性（一个事务中的操作要么都做，要么都不做，不可拆分）
   - 一致性（使数据库从一个一致性状态变到另一个一致性状态，只包含成功事务提交的结果）
   - 隔离性（一个事务的执行不被其它事务干扰）
   - 持续性（一个事务一旦提交，它对数据库的改变应是永久性的，接下来的其它操作或故障不应该对其执行结果有任何影响）

9.1.2. 数据库的三类非预期故障

三大故障	描述	产生原因	恢复策略
事务故障	某一个事务在运行过程中由于各种原因未运行至正常终止点就夭折了	1. 输入数据有误 2. 运算溢出 3. 违反了某些完整性限制 4. 某些应用程序出错 5. 并行事务发生死锁	强行回滚（ROLLBACK），也即是撤消（UNDO）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
系统故障	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的信息全部丢失；外部存储设备上的数据未受影响	1. 操作系统或 DBMS 代码错误 2. 特定类型的硬件错误 3. 突然停电	• 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，恢复程序要撤消（UNDO）所有未完成事务 • 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
介质故障	数据库硬件故障使存储在外存中（永久保存）的数据部分丢失或全部丢失	1. 磁盘损坏 2. 磁头碰撞 3. 瞬时强磁场干扰	• 装入数据库发生介质故障前某个时刻的数据副本 • 重做自该时刻起所有提交的事务，将这些事务已提交的结果重新记入数据库

9.2. 恢复的实现技术

9.2.1. 数据转储

1. 静态转储（冷备份）
2. 动态转储（热备份）
3. 海量转储（完全备份）
4. 增量转储（差异备份）

9.2.2. 日志文件

事务处理日志

- 日志文件: 以流水方式记录每个事务对数据库的每一次更新操作和操作的顺序
- 日志用途: 进行事务故障、系统故障恢复; 协助进行介质故障恢复

必须先写日志文件, 后写数据库

9.3. 恢复策略

9.3.1. 事务故障的恢复

- UNDO 此事务已对数据库进行的修改。

  （不进行 REDO, 因为只是当前这个事务被影响了, 其他仍然 ok）
- 系统在重新启动时自动完成

9.3.2. 系统故障的恢复

- 原因: 数据库进入了不一致状态
  1. 未完成事务对数据库的更新已写入数据库
  2. 已提交事务对数据库的更新还留在缓冲区
- 恢复方法
  1. UNDO 故障发生时未完成的事务
  2. REDO 故障发生时已提交的事务（RLLBACK 不用管）

9.3.3. 介质故障的恢复

介质故障的恢复

- 原因: 硬件故障导致数据丢失
- 恢复方法
  1. 通过数据备份, 使数据库恢复到某个一致性状态
  2. 根据日志备份对相关事务进行 REDO 和 UNDO
- DBA 手动完成

9.4. 具有检查点的恢复技术

需要搜索的日志

无检查点	有检查点
从日志开始点至故障点	从检查点至故障点

有了检查点, 就可以不用大量增加记录数据, 而使持久化状态无限接近故障点状态

Chapter 10: 并发控制

Chapter 10: 目录

10.1. 并发控制概述

10.1.1. 事务的不同执行方式

1. 串行
2. 交叉并发（同一时刻只有一个事务在执行，事务之间交替执行）
3. 同时并发（多处理机系统中同时运行多个事务，实现多个事务真正并行运行）

10.1.2. 并发操作带来的数据不一致性

数据不一致产生原因：并发操作破坏了事务的隔离性

1. 丢失修改
   - 事务 A 与事务 B 从数据库中读入同一数据并修改
   - 事务 B 的提交结果破坏了事务 A 提交的结果，导致事务 A 的修改被丢失
2. 脏读
   - 事务 B 修改某一数据后，事务 A 读取同一数据；
   - 事务 B 由于某种原因进行撤消，数据恢复原值
   - 事务 A 读到的数据与数据库中的数据不一致，是不正确的数据，又称为“脏”数据
3. 不可重复读
   - 事务 A 读取数据后，事务 B 执行更新操作，当事务 A 再次读该数据时，得到与前一次不同的值
4. 幻读
   - 事务 B 删除部分记录，事务 A 再次读取，发现某些记录神密地消失
   - 事务 B 插入记录，事务 A 再次读取，发现多了一些记录

10.1.3. 事务的四种隔离级别

隔离级别	数据一致性			
	丢失修改	脏读	不可重复读	幻读
读未提交	×	✓	✓	×
读已提交	×	×	✓	✓
可重复读	×	×	×	✓
可串行化	×	×	×	×

越往下数据一致性越强，系统代价越高。事务隔离级别并非越高越好 。

10.2. 封锁协议

封锁: 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，其它的事务不能读取/更新此数据对象。

1. 排它锁，又称为写锁（X 锁）
   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
2. 共享锁，又称为读锁（S 锁）
   若事务 T 对数据对象 A 加上 S 锁，则事务 T 可以读 A 但不能修改 A，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

10.2.1. 一级封锁协议

- 写数据之前申请
- 申请 X 锁
- 事务结束释放（包括 Commit 和 Rollback）

脏读、不可重复读

由于未加 S 锁，其他事务可以直接读取被加了 X 锁读对象，如果 ROLLBACK，则产生 dirty data。

10.2.2. 二级封锁协议

1. - 写数据之前申请
   - 申请 X 锁
   - 事务结束释放（包括 Commit 和 Rollback）
2. - 读数据之前申请
   - 申请 S 锁
   - 读完数据释放

不可重复读

10.2.3. 三级封锁协议

1. • 写数据之前申请
   • 申请 X 锁
   • 事务结束释放（包括 Commit 和 Rollback）
2. • 读数据之前申请
   • 申请 S 锁
   • 事务结束释放（包括 Commit 和 Rollback）

不论哪个锁，都影响并发程度

	X 锁	S 锁		一致性保证			隔离级别
	事务结束释放	操作结束释放	事务结束释放	不丢失修改	不脏读	可重复读	
一级	✓			✓			读未提交
二级	✓	✓		✓	✓		读已提交
三级	✓		✓	✓	✓	✓	可串行化

10.3. 活锁与死锁

数据库系统允许死锁发生，DBMS 定期检测系统中是否存在死锁，一旦检测到死锁，就设法解除

死锁概念：两个或多个事务都已封锁了一些数据对象，然后又都请求对已被其他事务封锁的数据对象加锁，从而出现互相等待

预防死锁：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
存在的问题：

1. 降低了并发度
2. 将以后可能要用到的全部数据加锁，势必扩大了封锁的范围
3. 数据库中数据是不断变化的，原来不要求封锁的数据，在运行中可能会变成封锁对象，很难事先精确地确定每个事务所要封锁的数据对象

解除死锁的常用方法

- 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锁，使其它事务得以继续进行下去
- 选择标准：选择最迟交付的事务；选择获得锁最少的事务；选择回退代价最小的事务

10.4. 并发调度的可串行性

10.4.1. 调度的正确性

可串行性是并行事务正确性的唯一准则

可串行化的调度

- 计算机系统对并行事务中并行操作的调度是随机的，而不同的调度可能会产生不同的结果
- 将所有事务串行起来的调度策略一定是正确的调度策略
- 以不同的顺序串行执行事务也有可能会产生不同的结果，但由于不会将数据库置于不一致状态，所以都可以认为是正确的
- 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

10.4.2. 冲突可串行化的调度

冲突可串行化调度是可串行化调度的充分非必要条件

10.4.3. 两段锁协议

两段锁协议是保证并发操作调度正确性的方法之一。

1. 协议内容
   - 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. 两段锁将事务分为两个阶段
   - 第一阶段是获得封锁，也称为扩展阶段
   - 第二阶段是释放封锁，也称为收缩阶段

   事务遵守两段锁协议是可串行化调度的充分非必要条件

10.4.4. 不同协议对比

1. 一次封锁法是为了预防死锁
   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
2. 三级封锁协议是为了保证数据一致性
   遵守第三级封锁协议必然遵守两段锁协议
3. 两段锁协议是为了保证并行调度的正确性
   - 要求释放锁之后不能再申请新锁
   - 遵守两段锁协议的事务可能发生死锁

10.5. 封锁的粒度

封锁对象的大小称为封锁的粒度

X 锁和 S 锁都是加在某一个数据对象上的

- 逻辑单元: 属性值、属性值集合、元组、关系、索引项、整个索引、整个表、整个数据库等
- 物理单元: 页（数据页或索引页）、物理记录等