

《数据库原理》知识点

应用题类型: <https://blog.csdn.net/jonahzheng/article/details/12113359>

重点——根据查询要求, 写关系代数运算;

——根据关系, 计算运算结果

——构建 E-R 图, 并 E-R 图转关系模式, 判断主码外码 (加一个判断范式)

——数据库事务处理技术

——根据 E-R 图, 设计符合 3NF 的关系模式

SQL 语句:

定义表、模式、视图

填表, 考查完整性约束 (其中, 用 **constraint** 写完整性约束名)

对表的增加、删除、修改

查询语言: (In /not in /exist/not exist / having /group by/having/like/is null/is not Null/聚集函数/升降序/嵌套查询 /查询所有/查询只学了/查询至少/查询没有/既又 (自连接、in 子查询、除运算))

图书一系列的问题

两个 not exist + (查询选了全部课程的学生)

第 1 章 绪论

1. 数据库、数据库管理系统、数据库系统的概念

- ✧ 数据库 (DB): 长期存放在计算机内, 有组织的、可共享的大量的数据的集合
- ✧ 数据库管理系统 (DBMS):
- ✧ 数据库系统 (DBS): 由数据库、数据库管理系统、应用程序和数据库管理员组成的存储、管理、处理和维持数据的系统

2. 数据库管理系统的特点、一般功能:

功能:

- ✧ 数据定义
- ✧ 数据的组织、存储、管理
- ✧ 数据操纵
- ✧ 数据库的事务管理和运行管理
- ✧ 数据库的建立和维护

3. 数据管理技术发展三个阶段:

- ✧ 人工管理: 程序和其需要的数据在一起, 数据不保存、不共享、修改必须一起修改
- ✧ 文件管理: 文件管理数据, 以记录为单位存储
- ✧ 数据库管理: 数据结构化、数据共享性高、独立性好

4. 数据库系统的特点

- ✧ 数据结构化
- ✧ 数据共享性高、冗余度低且易扩充
- ✧ 数据的独立性高
- ✧ 数据由 DBMS 统一管理和控制

5. 数据的物理独立性、逻辑独立性的含义

物理独立性：用户应用程序与数据库中的数据物理存储是相互独立的

逻辑独立性：用户的应用程序与数据库的逻辑结构是相互独立的

6. 数据库管理系统提供的数据库控制或保护功能

- ✧ 数据安全性保护
- ✧ 数据完整性检查
- ✧ 并发控制
- ✧ 数据库恢复

7. 数据模型的含义和作用

- ✧ 数据模型是数据库系统的核心和基础
- ✧ 数据模型由数据结构、数据操作、数据的完整性约束条件组成
- ✧ 数据模型：
 - (1) 概念模型——信息模型
 - (2) 逻辑模型（层次、网状、关系、面向对象）和物理模型（数据在系统内部的表示方式和存取方法）
- ✧ 概念模型的表示方法：实体-联系模型
- ✧ 抽象过程：现实世界—信息世界—机器世界

8. 概念模型、逻辑模型、物理模型的含义和作用

9. 概念模型（信息世界）中的基本概念

10. 常用的（逻辑）数据模型

- ✧ 数据模型分类：层次模型、网络模型、关系模型、面向对象数据模型
- 对应的数据结构分别为：树结构、向图结构、二维表结构

11. 关系模型的优缺点：

- ✧ 关系模型与格式化模型不同，建立在严格的数学概念的基础上
- ✧ 关系模型的概念单一（实体和联系都用关系来表示）

12. 数据库系统的三级模式结构

- ✧ 外模式（也是子模式，用户模式，可以多个）、模式（只能有一个）（逻辑模式、概

念模式)、内模式(存储)(只可一个)

——存储在数据字典中

- ◇ 子模式是模式的逻辑子集
- ◇ 模式/外模式的映射保证了逻辑独立性
- ◇ 模式/内模式的映射保证了物理独立性

13. 数据库的二级映像功能和作用

- ◇ 当模式改变时,只需要改变外模式/模式映像,而外模式不变,因此应用程序可以不变,保证了数据与程序的逻辑独立性;
- ◇ 当数据存储结构改变时,数据库管理员只需要对内模式/模式映像做改变,可以使模式不变,进而应用程序不变,保证了数据存储和程序的物理独立性

14. 数据库系统的组成

- ◇ 硬件平台及数据库
- ◇ 软件(操作系统、数据库管理系统、编译系统(接数据库接口,便于开发应用程序))
- ◇ 人员

15. 数据库系统中的人员组成

(数据库管理员(DBA)、系统分析员和数据库设计人员、应用程序员、用户)

第2章 关系数据库

1. 关系数据库理论的奠基者

E.F.Codd

2. 目前常用的关系数据库

- ◇ Oracle
- ◇ MySQL
- ◇ Microsoft SQL Server

3. 笛卡尔积的含义

行相乘、列相加

4. 关系与笛卡尔积的联系

关系是笛卡尔积的有限子集;n目关系必有n个属性

5. 候选码、主码、全码的含义

- ◇ 码：可以确定一个元组的属性或属性集合
- ◇ 候选码：可以唯一标识一个元组的**最少属性集合**（缺一个就标识不了）
- ◇ 主码：从候选码里面随机挑选的

6. 主属性、非主属性的含义

- ◇ 主属性：候选码所有属性的并集
- ◇ 其他的为非主属性

7. 关系的三种类型

- ◇ 基本关系（基本表、基表）——实际存在的表
- ◇ 查询表——查询结果对应的表
- ◇ 视图表——虚表

8. 基本关系的性质

- ◇ 列是同质的
- ◇ 列的顺序无所谓
- ◇ 同一关系中属性名不能相同
- ◇ 同一个关系中任意两个元组不能相同
- ◇ 元组的顺序无所谓
- ◇ 分量必须取原子值

9. 关系模式的形式化表示

$R(U, D, \text{dom}, F)$

R-关系名；U-组成该关系的属性名集合；D-属性组 U 中所来自的域；

Dom-属性向域的映像集合（属性的长度），F-属性间数据的依赖关系集合

10. 关系模式的简记表示

R

11. 关系与关系模式的联系

关系是关系模式在某一时刻的状态和内容；

关系模式是静态、稳定的；

关系是动态、随时间变化的；

12. 常用的关系操作

插入、删除、修改、查询（选择、投影、连接、除、并、差、交、笛卡尔积）

13. 5 种基本的查询操作

选择、投影、并、差、笛卡尔积

14. 关系操作的特点

集合操作方式（即操作的对象和结果都是集合）

15. 三类关系数据语言

✧ 关系代数语言

✧ 关系演算语言（ALPHA）

✧ 具有关系代数、关系演算双重特点的语言——结构化查询语言（SQL）

✧ SQL 是一个高度非过程化的语言（只需要告诉他做什么不需要告诉他怎么做）

16. 关系模型中的三类完整性

✧ 实体完整性：主属性不为 null 值

✧ 参照完整性：外码（可以是被参照表的主码或者非主码）

✧ 用户定义完整性（应用中所涉及的数据必须满足的语义要求）

17. 关系的两个不变性

实体完整性、参照完整性

18. 实体完整性含义及规则：

含义：若属性是主属性，则不能取空值

规则：

✧ 针对基本表

✧ 以主码为唯一标识

✧ 主属性不取空值

19. 参照完整性规则：

F 是 S 表的属性，并和 S 中主码相对应，F 在 R 是参照 S 表的外码，则 R 中在 F 上的值

要么取空值，要么取 S 中某个元组的主码值

（取空值（如果被参照关系是主码，则不可取空值）或者目标关系中已经存在的值）

20. 用户定义的完整性含义

针对某一具体关系数据库的约束条件，它反映某一具体应用所涉及的数据必

须满足的语义要求

21. 关系代数的运算按运算符的不同分类

- ✧ 集合运算：并、差、交、笛卡尔积
- ✧ 关系运算：选择、投影、连接、除

22. 交与差运算的联系

$$R \cap S = R - (R - S)$$

23. 关系代数运算：

- ✧ 集合运算（并、交、差、广义笛卡尔积）
- ✧ 除了笛卡尔积其他集合运算必须满足同构（具有相同的列数）条件
 - （1） 并：元组合并并删除重复值
 - （2） 交：相同元组
 - （3） 差： $R - S$ ， R 中去掉 S 中的相同部分
 - （4） 广义笛卡尔积：行相乘、列相加

- ✧ 关系运算（选择、投影）

（1） 选择（选择行）

❖ 1) 选择又称为限制（Restriction）

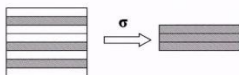
❖ 2) 选择运算符的含义

- 在关系 R 中选择满足给定条件的诸元组

$$\sigma_F(R) = \{t | t \in R \wedge F(t) = \text{'真'}\}$$

F ：选择条件，是一个逻辑表达式

❖ 3) 选择运算是从关系 R 中选取使逻辑表达式 F 为真的元组，是从行的角度进行的运算。



（2）投影（重新组合列，删除重复行）

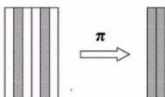
❖ 1) 投影运算符的含义

- 从 R 中选择出若干属性列组成新的关系

$$\pi_A(R) = \{t[A] | t \in R\}$$

A ： R 中的属性列

❖ 2) 投影操作主要是从列的角度进行运算



- 投影之后不仅取消了原关系中的某些列，而且还可能取消某些元组（避免重复行！）

(3) 连接

❖ 1) 连接也称为 θ 连接

❖ 2) 连接运算的含义

- 从两个关系的笛卡尔积中选属性间满足一定条件的元组

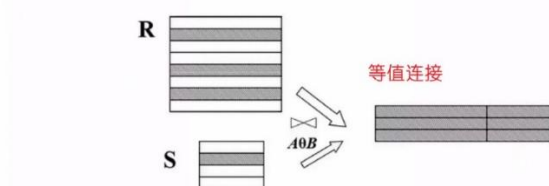
$$R \bowtie_{A\theta B} S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A] \theta t_s[B] \}$$

◆ A 和 B ：分别为 R 和 S 上度数相等且可比的属性组

◆ θ ：比较运算符

- 连接运算就是从 R 和 S 的广义笛卡尔积 $R \times S$ 中选取（ R 关系）在 A 属性组上的值与（ S 关系）在 B 属性组上值满足比较关系 θ 的元组

❖ 4) 一般的连接操作是从行的角度进行运算。



****自然连接还需要取消重复列，所以是同时从行和列的角度进行运算。**

等值连接

■ 等值连接 (equijoin)

◆ 什么是等值连接

θ 为“=”的连接运算称为等值连接

◆ 等值连接的含义

从关系 R 与 S 的广义笛卡尔积中选取 A 、 B 属性值相等的那些元组，即等值连接为：

$$R \bowtie_{A=B} S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[A] = t_s[B] \}$$

自然连接

■ 自然连接 (natural join)

◆ 自然连接是一种特殊的等值连接

✓ 两个关系中进行比较的分量必须是相同的属性组

✓ 在结果中把重复的属性列去掉

◆ 自然连接的含义

✓ R 和 S 的相同属性组的值相等

$$R \bowtie S = \{ \widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge t_r[B] = t_s[B] \}$$

内外连

❖ 内连接 (INNER JOIN)

- 两个关系做自然连接时，连接的结果是满足条件的元组保留下来，不满足条件的元组被舍弃了。

❖ 外连接 (OUTER JOIN)

- 如果把舍弃的元组也保存在结果关系中，而在其他属性上填空值(Null)，这种连接就叫做外连接。

空值表示不存在或不肯定

(4) 除

关系代数的除法示意

1. 取出相同列的投影
2. 不同列上取消重复值
3. 列举不同列上所有的象集，判断哪些象集的相同列包含了被除数中所有的值

SQL 语言的除法结构：

例 1：查询至少选修了 95002 学生选修的全部课程的学生学号

解释：查询学号，不存在这样的课程，学号为 95002 的学生选了，而其他学号的学生没选

```
select distinct sno from student s
```

```
Where not exist (select * from sc sc1
```

```
Where sc1.sno='95002' and not exist ——第一个 not exist 是条件
```

```
(select * from sc sc2
```

```
Where sc2.sno=s.sno and sc2.cno=sc1.cno)) ——第二个 not exist 是做等值连接
```

例 2：查询选修了全部课程的学生姓名

(造表，造一个含学生姓名（学生表）和课程名的表（课程表），介质：成绩表)

```
Select sname from student as s
```

```
Where not exist (select * from course as c
```

```
Where not exist (select * from score as sc
```

```
Where sc.sno=s.sno and sc.cno=c.cno))) ;
```


第3章 关系数据库标准语言 SQL

1. SQL 的含义

结构化查询语言是关系数据库的标准语言

2. SQL 的特点

- ✧ 综合统一（DDL、DML 等一体）
- ✧ 高度非过程化（只需要告诉它做什么，不用告诉它怎么做）
- ✧ 面向集合的操作方式
- ✧ 同一语法结构提供多种使用方式（既是独立语言也是嵌入式语言）
- ✧ 语言简洁、易学易用

3. SQL 的 9 个动词

数据定义语言动词（DDL）：create、drop、alter ——针对表的结构

数据操纵语言动词（DML）insert、update、delete ——针对表中的数据

数据查询语言：select

数据控制语言动词（DCL）：grant、revoke ——从用户权限、安全性考虑

4. 模式的定义与删除

- ✧ 创建模式

Create scheme 模式名 authorization 用户名

创建模式，并在模式下面定义表

Create scheme test authorization 小明

Create table Student （sno int ……）

- ✧ 删除模式

Drop scheme 模式名 cascade|restrict

级联 cascade：删除模式同时删除模式中的数据库对象

限制 restrict：如果被删除模式下定义了下属的数据库对象，则拒绝删除

5. 基本表（一个模式包含多个表）的定义和删除

- ✧ 创建表

Create table 表名（列名 数据类型 列级完整性约束

列名 数据类型 列级完整性约束

…

表级完整性约束）；

例：建一个学生表：

```
Create table student (sno char(5) primary key check(sno like '99
[0-9][0-9][0-9]'),
Sname char (20) not null,
Sage smallint
);
```

```
Create table course (cno char (4) primary key,
Cpno char (4),
Cname char (20) not null,
Foreign key (cpno) references course (cno)
);
```

(外码的参照表和被参照表可以是同一个表)

✧ 修改 (add/drop) 表

Alter table 表名+ (以下任意一个)

Add 新列名 数据类型 完整性约束条件 ——在表中增加新列

Eg. Alter table student add ssex char(8) not null;

Add 表级约束条件 ——增加表中的新的表级约束条件

Eg. Alter table student add primary key (sno);

Drop 列名 cascade|restrict ——删除列

级联：同时删除引用该列的其他对象

限制：如果该列被其他对象引用则拒绝删除

Eg. Alter table student drop sname cascade;

Drop constraint 完整性约束名 cascade|restrict ——删除指定的完整性约束条件

Eg. Alter table student drop constraint A_not_null cascade

Alter column 列名 数据类型 ——修改原有列的定义（改名字和数据类型）

Eg. Alter table student alter column sno char(8);

✧ 删除基本表

Drop table 表名 cascade|restrict

限制：此表有被其他表的约束所引用；有视图；有触发器则不能被删除

6. 总结完整性约束的写法（表级、列级只是放的位置不一样）

✧ Constraint 约束名 primary key （列名）

✧ Constraint 约束名 foreign key （该列名）references 被参照表（被参照列名）

7. 模式和表的整合

✧ 定义模式的同时定义表

✧ 系统设置指定的模式

✧ 定义表的同时在表名前面加“模式名”

8. 常用的数据类型

✧ Char

✧ Varchar

✧ Int、smallint、bigint

✧ Date（YYYY-MM-SS）、time（HH：MM：SS）

✧ 函数 getdate（）- year； check （birth<getdate()）

9. 建立索引的目的

✧ 加快查询速度

10. 查询语句

Select 什么 from 表

Where 条件

Group by +列名 having + group by 后的查询条件

Order by 列名 desc|asc（order by 只能用于主查询的结果，不允许用在子查询）

11. 查询经过计算的值

✧ Select 还可以组句:

select sname, 'year of birth' ,year

✧ Select 表达式: select 2014-sage,sname

12. Select 子句中*的作用

查询某表中所有字段

13. Select 子句中指定列别名

Select sno, cno from student as s, score as sc;

14. Select 子句中 distinct 关键词的作用

消除取值重复的行

15. Where 子句常用的查询条件

✧ 比较运算符 (<、>、=、>=、<=、!=、<>、!>、!<)

✧ 范围 (between 下限 and 上限/ not between and) ——上下限被包括

Where sage between 20 and 30;

✧ 确定集合 (in、not in)

Select sname, ssex from student

Where sdept in ('CS','MA','IS');

✧ 字符匹配 (like、not like)

✓ %——任意长度的字符串 (可为 0)

✓ _——表示任意单个字符, 额外的_这个和 C 语言一样, 表示转义_

Eg. Where sname like '六%'

✧ 空值 (is null、is not null)

Eg. Where grade is (not) null

✧ 多重条件 (and、or、not)

16. Order by 注意空值的排序

如果是 desc, 则 null 最先显示; 如果是 asc, 则空值最后显示

17. 常用的聚集函数

Count (*) ——统计元组个数

Count (distinct|all 列名) ——统计一列中值的个数

Sum (distinct|all 列名) ——计算一列值的总和

AVG (distinct|all 列名)

MAX (distinct|all 列名)

MIN (distinct|all 列名)

注意:

✧ 仅 count (*) 会计入空值, 其他的只处理非空值

✧ 聚集函数只能用在 select 后面和 group by 中的 having 子句

✧ Select 语句中使用聚集函数的时候, select 语句后的字段必须在 group by 子句或者聚集函数中出现

Eg. Select sno, count (grade) from SC

Group by sno

18. Group by 和 Having 短语的作用

Select cno, count (sno) From SC

Group by cno ——查询结果按照 cno 分组, 具有相同 cno 的元组为一组

Having count(*)>3; ——count 对每一组计数

19. 两表等值连接、自然连接(等值连接基础上去掉重复的属性列)、左外连接、右外连接查询

✧ 等值连接

Select student.*,SC.* from student, SC

Where student.sno=SC.sno;

✧ 自然连接

Select student.sno,sname,cno,grade

From student,SC

Where student.sno=SC.sno;

✧ 自连接(查询课程的间接先修课)

Select first. Cno , second. Cpno from course first, course second

Where first. Cpno=second. cno

✧ 外连（关键词左边就是左表，会有 null 填充）

（1）左外连

```
Select *from student s
```

```
Left outer join score sc on s.sno=sc.sno
```

（2）右外连

```
Select *from student s
```

```
Right outer join score sc on s.sno=sc.sno
```

（3）全连

```
Select*from student s
```

```
Full join score sc on s.sno=sc.sno
```

20. 嵌套查询（select-from-where 为一个查询块儿，这个查询块儿套在另一个 where 或者 having 语句中，称为嵌套查询）

✧ 带有 in（not in）谓词的子查询

```
Select sno, sname, sdept from student
```

```
Where sdept in (select sdept from student
```

```
Where sname= '刘晨');
```

或者用自连接

```
Select S1.sno, S1.sname, S2.sdept from student S1, student S2
```

```
Where S1.sdept=S2.sdept and S2.sname= '刘晨';
```

✧ 带 exists（not exists）（不返回值，只返回 T 或 F）谓词的子查询

（由 exist 引出的子查询目标列表表达式通常用*，因为只返回真假，给列名没意义）

（1）查询选修了 1 号课程的学生姓名——exist（内查询非空就为真）

```
Select sname from student
```

```
Where exist (select *from sc
```

```
Where cno= '01' and sno=student.sno)
```

——红色标注为判断真假的条件

（2）查询没有选修了 1 号课程的学生姓名——not exist（内查询空就为

真)

Select sname from student

Where not exists (select *from sc

Where cno= '01' and sno=student.sno)

(3) 查询选修了全部课程的学生姓名

等价转换为存在量词:

对于一个学生, 不存在这样的课程, 选修课表里有, 但是它没选

Select sname from student s

——对于一个学生

Where not exist ——不存在这样的课程

(select *from course c ——存在于课程表

Where not exist ——但是不存在于他的选课表

(select *from score sc

Where sc.sno=s.sno and sc.cno=c.cno)) ;

(4) 查询至少选修了学生 201215122 选修的全部课程的学生学号

转化为: 对于一个学生, 不存在这样的课程, 学生 201215122 选修了, 但是他没有选修

Select distinct sno from student s ——对于一个学生

Where not exists(select *from score sc1

Where sc1.sno='201215122' and not exists(select *from score sc2

Where sc2.sno=s.sno and sc2.cno=sc1.cno));

✧ 带有比较运算符的子查询

找出每个学生超过他自己选修课程平均成绩的课程号

Select sno,cno from score X

Where grade>= (select avg (grade) from score Y

Where Y.sno=X.sno);

——子查询结果受父查询传递的 sno 影响(相关子查询)

——从外层查询取出一个元组 x, 将元组 x 的值传递给内查询

✧ 带有 any 或 all 谓词的子查询

子查询返回单值的时候用比较运算符，返回多值的时候使用 any/all

- (1) Any 表示某一个就行；all 表示所有值

查询非计算机系比计算机系任意一个学生年龄都要小的学生姓名和年龄

```
Select  sname, sage from student
```

```
Where sdept != '计算机系' and sage < any( select sage from student
```

```
Where sdept= '计算机' );
```

或者用集合函数

```
Select sname, sage from student
```

```
Where sdept != '计算机系' and sage < (select max (sage) from student
```

```
Where sdept= '计算机系' )
```

21. 集合查询的关键词

Union（并）、intersect（交）、except（差）

- (1) 查询选修了 1 号课程或者 2 号课程的学生（并集）

```
Select sno from SC
```

```
Where cno= '01'
```

```
Union
```

```
Select sno from SC
```

```
Where cno= '02';
```

- (2) 查询既选修了 01 课程又选修了 02 课程的学生

```
Select sno from SC
```

```
Where cno= '01'
```

```
Intersect
```

```
Select sno from SC
```

```
Where cno='02';
```

或者使用 in

```
Select sno from SC
```

```
Where cno= '01' and sno in (select sno from SC
```

```
Where cno= '02');
```

22. 数据更新语句 insert、update、delete

✧ Insert

(1) 插入元组

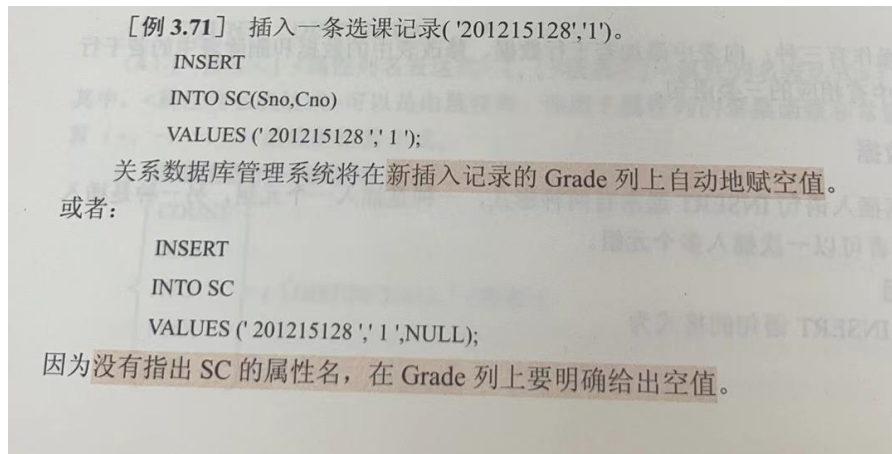
Insert into 表名 (列名 1, 列名 2, ……)

——也可只写表名，则后面需要插入表中对应的所有字段内容

无值就插入 null

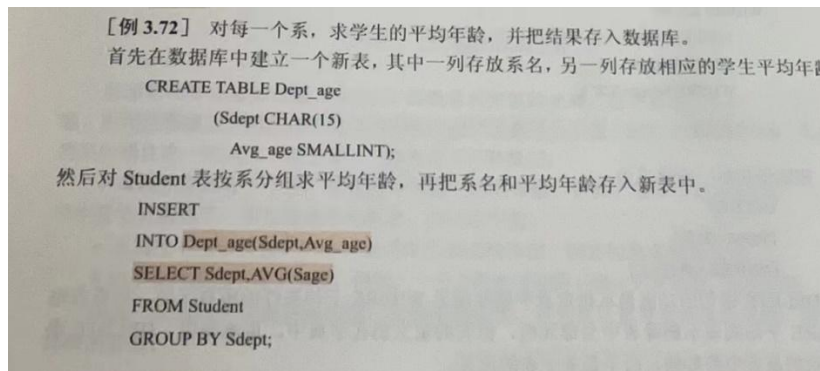
Values ('插入的内容','插入的内容','插入的内容',……)

——注意插入的内容和上述的列名必须一一对应



(2) 插入子查询结果

Insert into 表名+子查询



✧ Update

Update 表名 set 列名/表达式

Where 条件

Eg.将学号为 '01' 的学生年龄改为 22 岁——修改一个元组

Update student set sage=22

Where sno= '01';

Eg.将所有学生的年龄增加 1 岁——修改多个元组值

Update student set sage=sage+1;

Eg.将计算机系的同学的成绩记为 0——带子查询的修改语句

Update SC

Set grade=0

Where sno (select sno from student

Where sdept= '计算机系');

- ✧ Delete (从指定表中删除满足 where 条件的元组)(不同于 drop, delete 是删除的数据, drop 是删除的表的定义)

Delete from 表名

Where 条件

Eg. delete from student ——删除整张表的数据

Where sno= '01'; ——删除学号为 01 的学生的数据

23. 定义视图语句 create view as

Create view 视图名

As 子查询+[With check option]

24. 视图

- ✧ 建立视图的注意事项

(1) 组成视图的属性列名全部省略或者全部指定

若省略, 则默认该视图的字段名为子查询中的字段名组成

某个目标列是聚集函数或者列的表达式必须指定视图中的列名

多表连接的时候采用的是同名列作为字段名必须指定视图的列名

不是所有的视图都可以被更新

- ✧ 视图的作用

(1) 简化用户操作

(2) 使用户从多个角度看待同一个数据

(3) 对机密数据提供安全性保护

(4) 对重构数据库提供了一定程度的逻辑独立性

(5) 适当利用视图可以更清晰地表达查询

第4章 数据库安全性

1. 数据库安全性的含义

安全性：保护数据库，防止恶意破坏和非法存取（防范非法用户和非法操作）

2. 对数据库安全性产生威胁的主要因素

非法用户+非法操作

3. 数据库安全性控制主要技术

✧ 用户身份鉴别

✧ 存取控制（包括定义用户权限和合法权限检查）

4. 自主存取控制的含义

✧ 用户权限的组成：数据对象+操作类型

✧ 存取控制的对象：数据+数据库模式（模式、基表、视图、索引等）

✧ 同一用户对于不同的数据库对象有不同的存取权限，不同用户对于同一数据库有不同的权限，且用户可以将拥有的存取权限转授给其他用户

5. 自主存取控制的实现：

✧ GRANT 语句

Grant 权限 on 对象类型 对象名 to 用户 （with grant option）

例：grant select/all privileges/update（sno） on table student to U1/public（所有用户）

✧ REVOKE 语句

Revoke 权限 on 对象类型 对象名 from 用户 （cascade/restrict）

例：revoke select on table sc from public

Revoke update（sno） on table student from U1 cascade

（如果授权 U1 有授权权限，那么收回 U1 的权限时需要级联一同收回 U1 授权别人的权限）

6. 强制存取控制的含义

✧ 用户被授予许可证，每一个数据库对象被标以不同的密级

✧ 合法许可证的用户才可以存取相应的数据库对象

7. 自主存取控制与强制存取控制的联系

强制存取控制安全性更高

8. 数据库角色的含义

角色是被命名的一组与数据库操作相关的权限，角色是权限的集合。

第 5 章 数据库完整性

1. 数据库的完整性含义

是指数据的正确性和相容性

2. 数据的完整性

✧ 完整性：是为了防止数据库中存在不符合语义的数据，也就是防止数据库中不存在不正确的数据。

3. 为维护数据库的完整性，数据库管理系统必须实现的功能

- ✧ 提供定义完整性约束条件的机制
- ✧ 提供完整性检查的方法
- ✧ 进行违约处理

4. 所有完整性都是在 `creat table` 中定义

5. 关系模型中实体完整性定义方法

Primary key：单属性构成的码定义为列级约束条件

多属性定义为表级约束

[例 5.1] 将 Student 表中的 Sno 属性定义为码。

```
CREATE TABLE Student
(
  Sno CHAR(9) PRIMARY KEY, /*在列级定义主码*/
  Sname CHAR(20) NOT NULL,
  Ssex CHAR(2),
  Sage SMALLINT,
  Sdept CHAR(20)
);
```

[例 5.2] 将 SC 表中的 Sno、Cno 属性组定义为码。

```
CREATE TABLE SC
(
  Sno CHAR(9) NOT NULL,
  Cno CHAR(4) NOT NULL,
  Grade SMALLINT,
  PRIMARY KEY (Sno,Cno) /*只能在表级定义主码*
);
```

6. 实体完整性的检查方法和违约处理方法

- ✧ 检查内容：主码值是否唯一；主码的各个属性是否为空
- ✧ 检查方法：全表扫描
- ✧ 违约处理方法：

7. 关系模型中参照完整性定义方法

Foreign key 定义外码，references 指明外码参照哪些表的主码

```
CREATE TABLE SC
(Sno CHAR(9) NOT NULL,
Cno CHAR(4) NOT NULL,
Grade SMALLINT,
PRIMARY KEY (Sno, Cno),    /*在表级定义实体完整性*/
FOREIGN KEY (Sno) REFERENCES Student(Sno), /*在表级定义参照完整性*/
FOREIGN KEY (Cno) REFERENCES Course(Cno) /*在表级定义参照完整性*/
);
```

8. 可能破坏参照完整性的 4 种情况

- ✧ 在被参照表中删除元组、修改主码值
- ✧ 在参照表中，插入元组，修改外码值

9. 参照完整性违约处理的策略

- ✧ 拒绝执行
- ✧ 级联操作：当删除或修改被参照表（Student）的一个元组导致与参照表（SC）的不一致时，删除或修改参照表中的所有导致不一致的元组
- ✧ 设置为 null：当删除或修改被参照表的一个元组时造成了不一致，则将参照表中的所有造成不一致的元组的对应属性设置为空值。

10. 用户定义完整性

（1）属性上的约束条件

- ✧ 列值非空的定义方法

```
CREATE TABLE SC
(Sno CHAR(9) NOT NULL,
Cno CHAR(4) NOT NULL,
Grade SMALLINT NOT NULL,
PRIMARY KEY (Sno, Cno), /*在表级定义实体完整性，隐含了Sno, Cno不允许取空值，
在列级不允许取空值的定义可不写 */
);
```

- ✧ 列值唯一(unique)的定义方法

```
CREATE TABLE DEPT
  (Deptno NUMERIC(2),
   Dname CHAR(9) UNIQUE NOT NULL, /*要求Dname列值唯一, 并且不能取空值*/
   Location CHAR(10),
   PRIMARY KEY (Deptno)
  );
```

✧ 列值默认值的定义方法

```
1 CREATE TABLE <表名> (
2   <列名1> 数据类型 (长度) DEFAULT 默认值
3   <列名1> 数据类型 (长度) DEFAULT 默认值
4   ... )
```

```
1 ALTER TABLE goods
2 ADD CONSTRAINT def_g_time DEFAULT (getdate()) FOR g_time
```

✧ Check 指定列值应该满足的条件

```
CREATE TABLE Student
  (Sno CHAR(9) PRIMARY KEY,
   Sname CHAR(8) NOT NULL,
   Ssex CHAR(2) CHECK (Ssex IN ('男', '女')), /*性别属性Ssex只允许取'男'或'女' */
   Sage SMALLINT,
   Sdept CHAR(20)
  );
```

```
CREATE TABLE SC
  (Sno CHAR(9),
   Cno CHAR(4),
   Grade SMALLINT CHECK (Grade>=0 AND Grade<=100), /*Grade取值范围是0到100*/
   PRIMARY KEY (Sno,Cno),
   FOREIGN KEY (Sno) REFERENCES Student(Sno),
   FOREIGN KEY (Cno) REFERENCES Course(Cno)
  );
```

(2) 元组上的约束条件

用 check 定义元组上的约束条件

11. 用户定义的完整性违约处理方法

- ✧ 属性上约束条件的检查——拒绝执行
- ✧ 元组上约束条件的检查——拒绝执行

12. Constraint 子句的功能

- ✧ Constraint 完整性约束条件名 完整性约束条件 (check、notnull、unique、

主码、外码)

✧ 建立完整性约束条件

```
CREATE TABLE Student
(
    Sno NUMERIC(6)
    CONSTRAINT C1 CHECK (Sno BETWEEN 900000 AND 999999),
    Sname CHAR(20)
    CONSTRAINT C2 NOT NULL,
    Sage NUMERIC(3)
    CONSTRAINT C3 CHECK (Sage < 30),
    Ssex CHAR(2)
    CONSTRAINT C4 CHECK (Ssex IN ('男', '女')),
    CONSTRAINT StudentKey PRIMARY KEY(Sno)
);
```

✧ 修改完整性约束条件

【例 5.13】修改表 Student 中的约束条件，要求学号改为在 900000~999999 之间，年龄由小于 30 改为小于 40。

```
/* 可以先删除原来的约束条件，再增加新的约束条件。 */
ALTER TABLE Student
    DROP CONSTRAINT C1;
ALTER TABLE Student
    ADD CONSTRAINT C1 CHECK (Sno BETWEEN 900000 AND 999999);
ALTER TABLE Student
    DROP CONSTRAINT C3;
ALTER TABLE Student
    ADD CONSTRAINT C3 CHECK(Sage < 40);
```

13. 完整性约束命名的好处

灵活的增加、删除一个完整性约束条件

第 6 章 关系数据理论

1. 规范化的含义

规范化是指一个低一级范式的关系模式通过模式分解，可以转换为若干个高一级范式的关系模式的集合

2. 关系模式不好（不规范）可能存在的问题

✧ 数据冗余

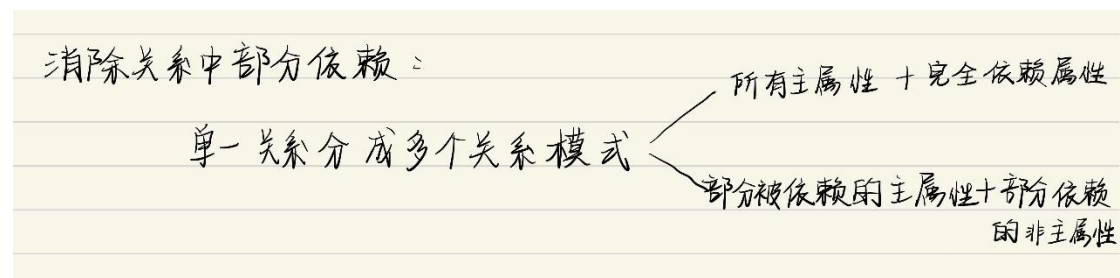
- ✧ 更新异常
- ✧ 插入异常
- ✧ 删除异常

3. 函数依赖、完全函数依赖、部分函数依赖、传递函数依赖的含义

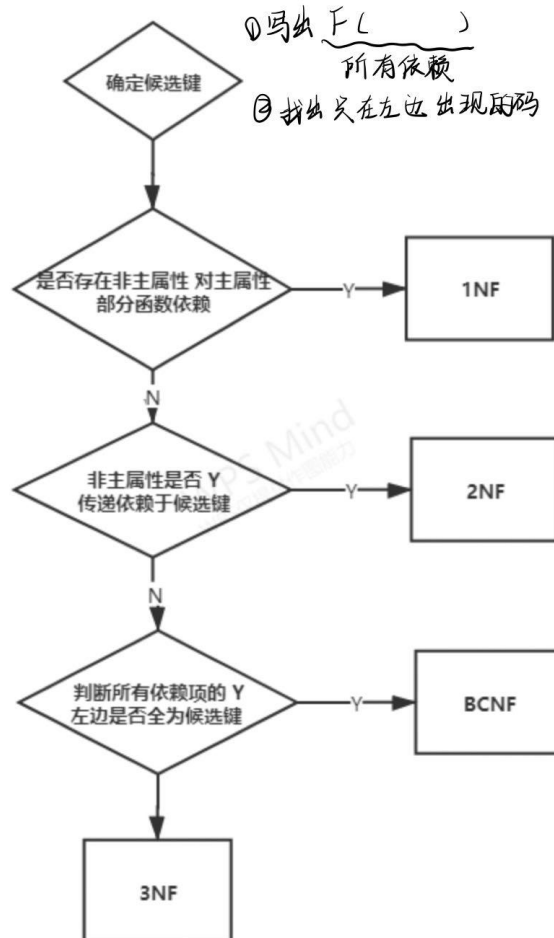
- ✧ 函数依赖：已知 X 可以确定 Y
- ✧ 完全函数依赖：非主属性完全依赖于主属性，主属性 X 可以确定非主属性 Y ，但对于 X 的任意一个真子集不能确定 Y
- ✧ 部分函数依赖：当 1 个关系模式中主键由 2 个及以上的属性组成时，非主属性只依赖于其中 1 个主属性，就是部分函数依赖
- ✧ 传递函数依赖：当关系模式中，出现非主属性决定非主属性时，就是传递函数依赖。

4. 1NF、2NF、3NF 的含义和联系

- ✧ 1NF：每一个分量必然是不可再分的数据项=列名（属性）不可再分
- ✧ 2NF：满足 1NF，非主属性完全函数依赖于主属性（消除部分依赖）
- ✧ 3NF：满足 2NF，非主属性部分全部直接依赖于主属性部分（消除传递依赖）
- ✧ 如何消除部分依赖：



5. 判断属于第几范式



第 7 章 数据库设计

1. 数据库设计的广义和狭义

广义：数据库及其应用系统的设计，即设计整个数据库应用系统

狭义：设计数据库本身，即设计数据库的各级模式并建立数据库，这是数据库应用系统设计的一部分

2. 数据库设计的特点

三分技术、七分管理、十二分基础数据

结构设计和行为设计相结合

3. 数据库设计的基本步骤

✧ 需求分析——得到数据字典、数据流图

✧ 概念结构设计——得到 E-R 图

✧ 逻辑结构设计——得到与数据模型相适应的逻辑结构（关系模式）

✧ 物理结构设计——

- ✧ 数据库实施
- ✧ 数据库运行和维护

4. 需求分析阶段的数据字典的内容

关系模式、视图、索引、完整性约束的定义、用户对数据库的操作权限、数据项、数据结构、数据流、数据存储和处理过程

5. E-R 图设计

- ✧ 实体间的联系（ N 个实体型之间的联系称为 N 元联系；单个实体型内也可能存在联系）

（1）一对一

（2）一对多

（3）多对多

6. 实体与属性的划分原则

- ✧ 属性必须为不可分的数据项
- ✧ 属性不能与其他实体具有联系

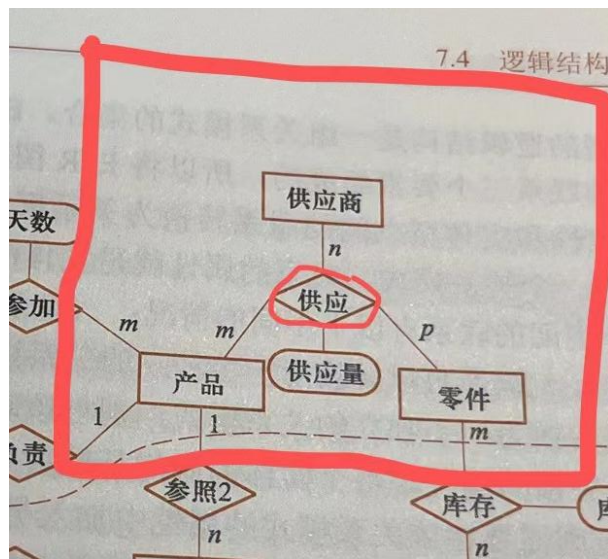
7. 各子系统的 E-R 图之间的冲突类型

- ✧ 属性冲突（属性域冲突，有些把零件号定义为整数，有些定义为小数）
- ✧ 命名冲突（同名异义、异名同义）
- ✧ 结构冲突（同一对象在不同应用中具有不同的抽象；同一实体在不同子系统中属性不同；实体间的联系在不同的 E-R 图中为不同的类型）

8. E-R 图转关系模式

- ✧ 一个实体一个关系
- ✧ 联系的转化：
 - （1）1:1 的联系：可与任意一端对应的关系模式合并；也可转化为一个独立的关系
 - （2）1: n 的联系：可与 n 端对应的关系模式合并；也可单独转化为一个关系模式
 - （3） $m:n$ 的联系：转化为一个关系模式

- (4) 三个或者三个以上的实体间的一个多元联系可以转化为一个关系模式



供应联系转关系：供应（产品号、供应商号、零件号、供应量）

- (5) 具有相同码的关系模式可以合并

9. 根据 E-R 图，设计符合 3NF 的关系模式

10. 数据库的维护工作

- ✧ 数据库的转储和恢复
- ✧ 数据库的安全性和完整性控制
- ✧ 数据库性能的监督、分析和改造
- ✧ 数据库的重组与重构造

第 10 章 数据库恢复技术

1. 数据库事务的含义

事务是用户定义的一个数据库操作序列，这些操作要么不执行要么全部执行
一个程序包含多个事务

2. 数据库事务处理技术

3. 在 SQL 中定义事务的 3 条语句和作用

Begin transaction 开始

Commit/Rollback 结束

Rollback 表示回滚：事务运行过程中发生某种故障，事务不能继续执行，系统撤销已完成的操作，回滚到事务开始的状态

4. 事务的 ACID 特性的含义

事务的特性：

- ✧ 原子性：事务里的操作要么不做要么全做
- ✧ 一致性：事务完成，所有数据处于一致的状态
- ✧ 隔离性：并行事务的修改必须与其他并行事务的修改相互独立
- ✧ 持续性：事务一旦提交，数据的改变是永久的

5. 事务 ACID 特性可能遭到破坏的因素

- ✧ 多个事务并行运行，不同事务的操作交叉执行
——保证多个事务的交叉运行不影响事务的原子性
- ✧ 事务在运行过程中被强行停止
——保证被强行终止的事务对数据库和其他事务没有任何影响

6. 数据库恢复的含义

DBMS 把数据库从错误状态恢复到某一已知的正确状态的功能

7. 数据库故障的种类及对数据库的影响

- ✧ 事务内部的故障（事务未正常完成）——强行回滚、事务撤销
- ✧ 系统故障（造成系统停止运转，需要重启系统）——撤销所有未完成的事务，**REDO**（重做）所有已提交的事务
- ✧ 介质故障——硬故障（外存故障）、破坏性大
- ✧ 计算机病毒

恢复的基本原理——冗余

错误数据利用存储在别处的冗余数据重建

8. 建立冗余数据最常用的技术

- ✧ 数据转储
- ✧ 登记日志文件

9. 数据转储的含义及方法

- ✧ **DB** 定期将整个数据库复制到磁带、磁盘或者其他存储介质上保存的过程
备用数据称为后备副本

✧ 方法：

- (1) 静态转储（静态海量、静态增量）——系统中无运行事务时进行转储
- (2) 动态转储（动态海量、动态增量）——转储期间允许对数据库进行存储或修改

10. 日志文件的含义、作用及登记原则

- ✧ 日志文件是记录事务对数据库的更新操作的文件（包括以记录为单位的日志文件、以数据块为单位的日志文件）
- ✧ 作用：事务故障恢复和系统故障恢复必须用到日志文件
- ✧ 登记原则：
 - (1) 登记的次序严格按并发事务执行的时间次序
 - (2) 必须先写日志文件后写数据库

11. 事务故障的恢复步骤

- ✧ 事务故障是指事物在运行至正常终点前被终止，这时恢复子系统利用日志文件 **undo** 此事务对数据库已完成的操作，将数据库恢复到事务开始前
 - (1) 反向扫描日志文件
 - (2) 对该事务的更新操作执行逆操作，将更新操作前的值写入数据库
 - (3) 如此反复，直至读到事务开始的标志

12. 系统故障的恢复步骤

撤销（**undo**）所有未完成的事务，**REDO**（重做）所有已提交的事务

- (1) 正向扫描日志文件
找出故障发生前已经提交的事务，将其事务标识计入 **redo**（重做）队列；
找出故障发生时尚未完成的事务，将其事务标识计入 **undo**（撤销）队列；
- (2) 对撤销队列的事务做 **undo** 处理
（反向扫描日志文件，执行逆操作，将日志记录中更新前值写入数据库）
- (3) 对重做队列中的各个事务进行 **redo** 处理
（正向扫描日志文件，将日志记录中更新后的值写入数据库）

13. 介质故障的恢复步骤

重装数据库，重做已完成的事务

(1) 装入最新的数据库后备副本，是数据库恢复到最近一次转储时的一致性状态

(2) 装入相应的日志文件副本，重做已完成的事务

14. 使用检查点方法进行恢复的好处

✧ 改善恢复效率

15. 使用检查点方法进行恢复的步骤

✧

16. 数据库镜像技术的用途

选取关键数据和日志文件复制数据(镜像)到另外的磁盘上

用于数据库的恢复

用于数据的并发操作（不用等待，直接使用镜像磁盘的数据）

第 11 章 并发控制

1. 在单处理机系统中，事务并发的含义、意义

当多个用户并发存取数据库时会产生多个事务同时存取同一数据的情况

2. 并发控制的含义、目的

✧ 事务是并发控制的基本单元

3. 并发操作带来的数据不一致性

✧ 读脏数据

✧ 不可重复读

✧ 丢失修改

4. 并发控制的主要技术

✧ 封锁

✧ 时间戳

✧ 乐观控制法

✧ 多版本并发控制

5. 封锁：

(1) 排他锁（X 锁——写锁）

仅允许一个事务对数据的读取、修改

(2) 共享锁（S 锁——读锁）

只读锁，可允许多个事务只读，在上 S 锁期间，任何事务不允许修改

✧ 一级封锁协议：

修改数据前先加 X 锁，事务结束后释放
可防止丢失修改

✧ 二级封锁协议：

修改数据前先加 X 锁，事务结束后释放
读取数据前加 S 锁，读完即可释放
防止丢失修改、读“脏”数据

✧ 三级封锁协议：

修改数据前先加 X 锁，事务结束后释放
读取数据前加 S 锁，事务结束后释放
避免丢失修改、读“脏”数据、不可重复读

表 11.1 不同级别的封锁协议和一致性保证							
	X 锁		S 锁		一致性保证		
	操作结束 释放	事务结束 释放	操作结束 释放	事务结束 释放	不丢失 修改	不读“脏” 数据	可重 复读
一级封锁协议		✓			✓		
二级封锁协议		✓	✓		✓	✓	
三级封锁协议		✓		✓	✓	✓	✓

6. 不同级别封锁协议的一致性程度

✧ 一级封锁协议：不丢失修改

✧ 二级封锁协议：不丢失修改、不读脏数据

✧ 三级封锁协议：不丢失修改、不读脏数据、可重复读

7. 活锁的含义

8. 避免活锁的先来先服务策略

封锁子系统按照请求封锁的先后次序对事务排队

9. 死锁的含义

陷入互相等待的困局

10. DBMS 在解决死锁的问题上普遍采用的方法

✧ 一次封锁法

事务需要一次将所有要使用的数据全部加锁

◇ 顺序封锁法

对数据对象规定一个封锁顺序，所有事务按顺序实施封锁

11. 普遍采用诊断与解除死锁来避免死锁

◇ 超时法的含义

如果一个事务的等待时间超过了规定时限就认为发生了死锁；

规定时限设置过短会误判、过长会发现不了

◇ 事务等待图法

◇ 画有向图 ($G = (T, U)$ T 表示节点的集合，节点表示正在运行的事务；

U 表示边的集合，边表示事务等待的情况)

◇ 如果形成回路就发生等待

检测出死锁后，通常会采用选择一个处理死锁代价最小的事务，将其撤销，释放此事务持有的所有的锁，使其他事务运行下去

12. 可串行化调度的含义

- 当且仅当事务的并发结果与按某一次序串行地执行这些事务时的结果相同时，我们认为多个事务的并发执行正确。
- 可串行性是并发事务正确调度的准则
- 一个给定的调度，当且仅当它是可串行化的，才认为是正确的调度

可串行化

调度经过两两交换，使相同事务在一起，
最终形成事务的集合排列

例：有调度 $W_3(y) R_1(x) R_2(y) W_3(x) W_2(x) W_3(z) R_4(z) W_4(x)$

事务调整为：

$R_1(x) \quad W_3(x) W_3(y) W_3(z) R_2(y) W_2(x) W_4(x) R_4(z)$
 $T_1 \quad T_3 \quad T_2 \quad T_4$

~~$W_3(y) R_2(y)$~~
 ~~$R_1(x) W_3(x) W_2(x) W_4(x)$~~
 ~~$W_3(z) R_4(z)$~~

13. 目前 DBMS 普遍采用的实现并发调度可串行性的方法

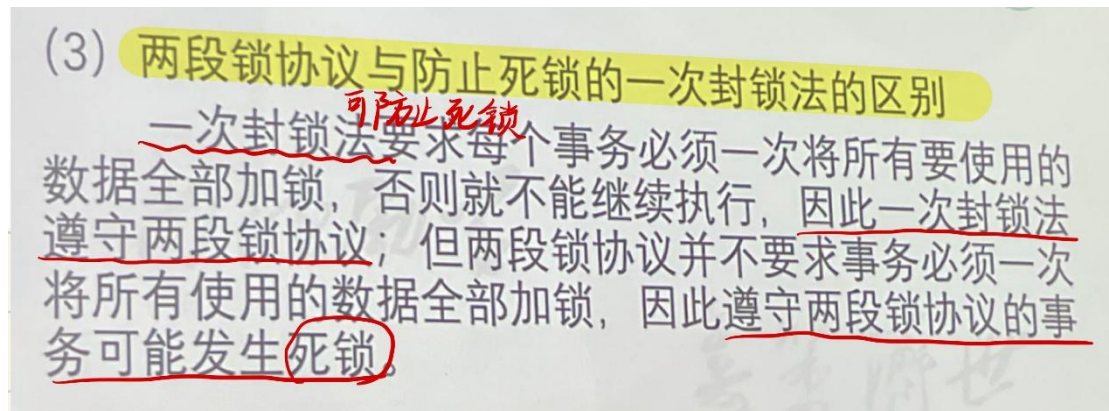
两段锁协议

采用两段锁协议可实现并发调度的可串行性，但实现可串行性调度并非都是遵循两段锁协议（两段锁协议是充分条件不是必要条件）

14. 两段锁协议的规定

（两段锁：第一段是获得封锁；第二段是释放封锁）——lock 一堆，unlock 一堆

两段锁协议与一次性封锁法：



15. 封锁粒度的含义

封锁对象（属性值、元组、关系等）的大小称为封锁粒度

16. 封锁粒度与系统的并发度和并发控制的开销之间的联系

- ✧ 封锁的粒度越大，则数据库能够封锁的数据单元就越少，并发度就越小，系统开销就小
- ✧ 封锁的粒度越小，则数据库能够封锁的数据单元就越多，并发度就越高，系统开销就越大

补充:

- (1) 对所有的视图都可以进行下面哪一个操作? (查询)
- (2) SQL 语言具有数据定义、数据操纵和数据控制的功能, 它的一次查询的结果是一个 (表)
- (3) SQL 语言中, SELECT 语句的执行结果是元组
- (4) SQL 语言中, 条件“年龄 BETWEEN 20 AND 30”表示年龄在 20 至 30 之间, 且包括 20 岁和 30 岁
- (5) 视图与基本表的对比:
 - 定义能力强于表, 因为可以在多张表上定义视图
 - 操作能力弱于表, 控制能力相当