一、 考试题型

- 选择题（共 10 小题，每小题 2 分，共 20 分）
- 填空题（共 10 小题，每小题 2 分，共 20 分）
- 简答题（共 6 小题，每小题 5 分，共 30 分）
- 设计题（共 4 小题，每小题 5 分，共 20 分）
- 综合题（共 1 小题，每小题 10 分，共 10 分）

二、 考试范围（参照教学大纲）

1、关系数据模型

- 数据库系统、数据模型、数据库 3 级抽象相关概念

数据模型是对现实世界数据特征的抽象；

数据库（Database，简称 DB）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；

数据库系统是由数据库、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、处理和维护数据的系统。

- 关系模型：关系数据模型的定义及其相关术语；
  - 域、笛卡尔积、元组（记录）、字段（属性）、关系（表）、超键、侯选码（候选键）、主码（主键）、外码（外键）、关系模式、关系数据库

关系：一个关系对应通常说的一张表

元组：表中的一行即为一个元组

属性：表中的一列即为一个属性，给每一个属性起一个名称即属性名

主码：也称码键。表中的某个属性组，它可以唯一确定一个元组

域：是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。

关系模式：对关系的描述

笛卡儿积：属性之间的排列组合

候选码：若关系中的某一属性组的值能唯一地标识一个元组，则称该属性组为候选码

全码：关系模式的所有属性组是这个关系模式的候选码，称为全码

关系数据库：在一个给定的应用领域中，所有关系的集合构成一个关系数据库

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 Ks 相对应，则称 F 是 R 的外码。

超码：如果 U 部分函数依赖于 K，则 K 为超码。候选码是最小的超码。

- 关系代数：并、差、交、广义笛卡尔积等传统的集合运算；选择、投影、连接、除等专门的关系运算。连接又包括条件连接，自然连接，外连接、左外连接和右外连接
  - 能够使用关系代数式（及 SQL）表示查询

A∪B：全部元素。

A-B：找到只在 A 中的元素。

A∩B：共同元素。

笛卡儿积：排列组合（A（3*3），B（2*2）），则运算后 A×B（6*5）。

选择：σ Sage < 20(Student)。（筛选行）

投影：π Sname,Sdept(Student)。（筛选列）

连接：

R

A	B	C
a1	b1	5
a1	b2	6
a2	b3	8
a2	b4	12

S

B	E
b1	3
b2	7
b3	10
b3	2
b5	2

一般连接 R $\bowtie_{C<E}$ S的结果如下：

A	R.B	C	S.B	E
a1	b1	5	b2	7
a1	b1	5	b3	10
a1	b2	6	b2	7
a1	b2	6	b3	10
a2	b3	8	b3	10

等值连接 $R \bowtie_{R.B=S.B} S$ 的结果如下：

A	R.B	C	S.B	E
a1	b1	5	b1	3
a1	b2	6	b2	7
a2	b3	8	b3	10
a2	b3	8	b3	2

自然连接 $R \bowtie S$ 的结果如下：

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2

下图是例2.8中关系*R*和关系*S*的外连接

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2
a2	b4	12	NULL
NULL	b5	NULL	2

图(b)是例2.8中关系*R*和关系*S*的左外连接,图(c)是右外连接

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2
a2	b4	12	NULL

图(b)

A	B	C	E
a1	b1	5	3
a1	b2	6	7
a2	b3	8	10
a2	b3	8	2
NULL	b5	NULL	2

图(c)

除运算：

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

2、SQL 语言

- SQL 的特点及 SQL 语言的基本概念。

SQL 是结构化查询语言，是关系数据库的标准语言
SQL 是一个通用的、功能极强的关系数据库语言
特点：
①综合统一
②高度非过程化
③面向集合的操作方式
④以同一种语法结构提供多种使用方式
⑤语言简洁，易学易用
基本概念：
基本表：
本身独立存在的表
SQL 中一个关系就对应一个基本表
一个（或多个）基本表对应一个存储文件
一个表可以带若干索引
存储文件：
逻辑结构组成了关系数据库的内模式
物理结构对用户是隐蔽的
视图
从一个或几个基本表导出的表
数据库中只存放视图的定义而不存放视图对应的数据
视图是一个虚表
用户可以在视图上再定义视图

- SQL 语言的功能。
  - 数据定义：定义表、删除表、修改表。

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

ALTER TABLE <表名>
[ADD[COLUMN] <新列名> <数据类型> [完整性约束]]
[ADD <表级完整性约束>]
[DROP [COLUMN] <列名> [CASCADE| RESTRICT]]
[DROP CONSTRAINT<完整性约束名>[RESTRICT | CASCADE]]
[ALTER COLUMN <列名><数据类型>] ;

DROP TABLE <表名>［RESTRICT| CASCADE］;

  - 单表查询：选择表中的若干列、选择表中的若干元组、查询结果排序、分组。

选择表中的若干列:
SELECT Sno,Sname FROM Student;

选择表中的若干元组:

①SELECT Sname,Sage FROM tudent WHERE Sage < 20;

②SELECT Sname, Sdept, Sage FROM Student WHERE Sage NOT BETWEEN 20 AND 23;

③SELECT Sname, Ssex FROM Student WHERE Sdept NOT IN ('IS','MA','CS');

④SELECT Sname，Sno FROM Student WHERE Sname LIKE '__阳%';

查询结果排序:

升序：ASC;降序：DESC;缺省值为升序。

①

```
SELECT Sno, Grade
FROM   SC
WHERE  Cno= ' 3 '
ORDER BY Grade DESC;
```

分组:

```
SELECT  Sno, AVG(Grade)
FROM  SC
GROUP BY Sno
HAVING AVG(Grade)>=90;
```

- NULL 值的处理。

```
SELECT Sno，Cno
FROM   SC
WHERE  Grade IS NULL
```

空值与另一个值（包括另一个空值）的算术运算的结果为空值

空值与另一个值（包括另一个空值）的比较运算的结果为 UNKNOWN。

有 NOT NULL 约束条件的属性不能取空值

加了 UNIQUE 限制的属性不能取空值

码属性不能取空值

```
SELECT Sno
FROM SC
WHERE Cno='1' AND (Grade<60 OR Grade IS NULL);
```

- 连接查询：等值连接、自身连接、外连接、复合条件连接。

等值连接：

①

```
SELECT  Student.*, SC.*
FROM    Student, SC
WHERE   Student.Sno = SC.Sno;
```

②

```
SELECT Student.Sno, Sname
FROM    Student, SC
WHERE  Student.Sno=SC.Sno  AND
                SC.Cno=' 2 ' AND SC.Grade>90;
```

自身连接：需要给表起别名以示区别，由于所有属性名都是同名属性，因此必须使用别名前缀。

```
SELECT  TheCourse.Cno, PreviousCourse.Cpno
FROM  Course  TheCourse, Course  PreviousCourse
```

```
        WHERE TheCourse.Cpno = PreviousCourse.Cno;
```

外连接：外连接操作以指定表为连接主体，将主体表中不满足连接条件的元组一并输出。

```
    SELECT Student.Sno,Sname,Ssex,Sage,Sdept,Cno,Grade
    FROM    Student    LEFT OUT JOIN SC ON
                    (Student.Sno=SC.Sno);
```

复合条件连接：

```
  SELECT Student.Sno, Sname, Cname, Grade
   FROM       Student, SC, Course       /*多表连接*/
   WHERE Student.Sno = SC.Sno
                     AND SC.Cno = Course.Cno;
```

- 嵌套查询：带 IN、Exists 谓词的子查询；带比较运算符的子查询，带谓词的子查询，相关子查询的执行方法和不相关子查询的执行方法。

带 IN、Exists 谓词的子查询：

①

```
    SELECT Sno, Sname, Sdept
         FROM Student
    WHERE Sdept    IN
                   (SELECT Sdept
                    FROM Student
                    WHERE Sname= '  刘晨  ');
```

②

```
     SELECT Sname
     FROM        Student
     WHERE NOT EXISTS
                    (SELECT *
                     FROM SC
                     WHERE Sno = Student.Sno AND Cno='1');
```

带比较运算符的子查询：

①

```
SELECT Sno,Sname,Sdept
      FROM       Student
      WHERE Sdept     =
                    (SELECT Sdept
                     FROM       Student
                     WHERE Sname= '刘晨');
```

②

```
   SELECT Sno, Cno
    FROM       SC    x
    WHERE Grade >=(SELECT AVG（Grade）
                                    FROM    SC y
                                       WHERE y.Sno=x.Sno);
```

带谓词的子查询：

①

```
    SELECT Sname,Sage
```

```
FROM       Student
WHERE Sage < ANY (SELECT    Sage
                                        FROM       Student
                                        WHERE Sdept= ' CS ')
 AND Sdept <> 'CS ' ;                  /*父查询块中的条件  */
```

等价于

```
SELECT Sname,Sage
FROM     Student
WHERE Sage <
                          (SELECT MAX（Sage）
                            FROM Student
                            WHERE Sdept= 'CS ')
   AND Sdept <> ' CS ';
```

②

```
SELECT Sname,Sage
FROM Student
WHERE Sage < ALL
                        (SELECT Sage
                         FROM Student
                         WHERE Sdept= ' CS ')
  AND Sdept <> ' CS ';
```

等价于：

```
SELECT Sname,Sage
FROM Student
WHERE Sage <
                          (SELECT MIN(Sage)
                           FROM Student
                           WHERE Sdept= ' CS ')
    AND Sdept <>' CS ';
```

相关子查询的执行方法和不相关子查询的执行方法：

不相关子查询：

子查询的查询条件不依赖于父查询。由里向外 逐层处理。即每个子查询在上一级查询处理之前求解，子查询的结果用于建立其父查询的查找条件。

相关子查询：

子查询的查询条件依赖于父查询。首先取外层查询中表的第一个元组，根据它与内层查 询相关的属性值处理内层查询，若 WHERE 子句返回值为真，则取此元组放入结果表。然后再取外层表的下一个元组。重复这一过程，直至外层表全部检查完为止。

■ 基本聚集查询

查询学生总人数。

```
SELECT COUNT(*)
FROM    Student;
```

查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)
FROM SC;
```

计算 1 号课程的学生平均成绩。

```
SELECT AVG(Grade)
FROM       SC
WHERE Cno= ' 1 ';
```

查询选修 1 号课程的学生最高分数。

```
SELECT MAX(Grade)
FROM SC
WHERE Cno='1';
```

查询学生 201215012 选修课程的总学分数。

```
SELECT SUM(Ccredit)
FROM    SC,Course
WHERE Sno='201215012' AND SC.Cno=Course.Cno;
```

- 数据更新：插入、删除、修改。

①插入元组：

```
INSERT
INTO    Student (Sno,Sname,Ssex,Sdept,Sage)
VALUES ('201215128','陈冬','男','IS',18);
```

②插入查询结果：

第一步：建表

```
CREATE    TABLE    Dept_age
    ( Sdept         CHAR(15)                          /*系名*/
      Avg_age SMALLINT);                  /*学生平均年龄*/
```

第二步：插入数据

```
INSERT
INTO    Dept_age(Sdept,Avg_age)
        SELECT    Sdept，AVG(Sage)
        FROM         Student
        GROUP BY Sdept;
```

③更新：

```
UPDATE    Student
SET Sage=22
WHERE    Sno=' 201215121 ';
```

④带查询的更新：

```
UPDATE SC
SET         Grade=0
WHERE Sno    IN
        (SELETE Sno
         FROM         Student
         WHERE    Sdept= 'CS' );
```

⑤带查询的删除：

```
DELETE
FROM    SC
WHERE    Sno    IN
    (SELETE    Sno
```

```
                    FROM     Student
                    WHERE    Sdept= 'CS') ;
```

- 视图：定义视图、查询视图、更新视图、删除视图。

定义视图：

①建立信息系学生的视图，并要求进行修改和插入操作时只涉及信息系的学生 。

```
CREATE VIEW IS_Student
AS
SELECT Sno,Sname,Sage
FROM    Student
WHERE    Sdept= 'IS'
WITH CHECK OPTION;
```

②建立信息系选修了 1 号课程的学生的视图（包括学号、姓名、成绩）。

```
CREATE VIEW IS_S1(Sno,Sname,Grade)
AS
SELECT Student.Sno,Sname,Grade
FROM    Student,SC
WHERE    Sdept= 'IS' AND
                    Student.Sno=SC.Sno AND
                    SC.Cno= '1';
```

③将学生的学号及平均成绩定义为一个视图

```
CREATE    VIEW S_G(Sno,Gavg)
  AS
  SELECT Sno,AVG(Grade)
  FROM    SC
  GROUP BY Sno;
```

查询视图：

视图消解法（View Resolution）：

- 进行有效性检查
- 转换成等价的对基本表的查询
- 执行修正后的查询

用户角度：和查询基本表相同。

更新视图：

①

```
UPDATE    IS_Student
SET    Sname= '刘辰'
WHERE    Sno= ' 201215122 ';
```

②

```
INSERT
INTO IS_Student
VALUES(‘201215129’,’赵新’,20);
```

③

更新视图的限制：一些视图是不可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新。

若视图是由两个以上基本表导出的，则此视图不允许更新。

若视图的字段来自字段表达式或常数，则不允许对此视图执行 INSERT 和 UPDATE 操作，但允许执行 DELETE 操作。
若视图的字段来自集函数，则此视图不允许更新。
若视图定义中含有 GROUP BY 子句，则此视图不允许更新。
若视图定义中含有 DISTINCT 短语，则此视图不允许更新。
若视图定义中有嵌套查询，并且内层查询的 FROM 子句中涉及的表也是导出该视图的基本表，则此视图不允许更新。
删除视图：
删除视图 BT_S 和 IS_S1

```
DROP VIEW IS_S;     /*成功执行*/
DROP VIEW IS_S1;    /*拒绝执行*/
```

要删除 IS_S1，需使用级联删除：

```
DROP VIEW IS_S1 CASCADE;
```

视图作用：

- 视图能够简化用户的操作
- 视图使用户能以多种角度看待同一数据
- 视图对重构数据库提供了一定程度的逻辑独立性
- 视图能够对机密数据提供安全保护
- 适当的利用视图可以更清晰的表达查询

3、数据库安全性

- 数据库安全性的基本概念。

数据库的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、更改或破坏 。

- 基于用户和角色授权的 SQL（即 GRANT、REVOKE 语句）

把查询 Student 表权限授给用户 U1

```
GRANT    SELECT
ON    TABLE    Student
TO    U1;
```

把对 Student 表和 Course 表的全部权限授予用户 U2 和 U3

```
GRANT ALL PRIVILIGES
ON TABLE Student,Course
TO U2,U3;
```

把对表 SC 的查询权限授予所有用户

```
GRANT SELECT
ON TABLE SC
TO PUBLIC;
```

把对表 SC 的 INSERT 权限授予 U5 用户，并允许他再将此权限授予其他用户

```
GRANT INSERT
ON TABLE SC
TO U5
WITH GRANT OPTION;
```

收回所有用户对表 SC 的查询权限

```
REVOKE SELECT  ON TABLE SC
FROM PUBLIC;
```

把用户 U4 修改学生学号的权限收回

```
REVOKE UPDATE(Sno)
ON TABLE Student
FROM U4;
```

通过角色来实现将一组权限授予一个用户。

步骤如下：

（1）首先创建一个角色 R1

```
CREATE  ROLE  R1;
```

（2）然后使用 GRANT 语句，使角色 R1 拥有 Student 表的 SELECT 、 UPDATE 、 INSERT 权限

```
GRANT SELECT, UPDATE, INSERT
ON TABLE Student
TO R1;
```

（3）将这个角色授予王平，张明，赵玲。使他们具有角色 R1 所包含的全部权限

```
GRANT  R1  TO  王平,张明,赵玲;
```

（4） 可以一次性通过 R1 来回收王平的这 3 个权限

```
REVOKE  R1
FROM 王平;
```

4、数据库完整性

- 关系的完整性：实体完整性、参照完整性、用户定义的完整性。

实体完整性：

CREATE TABLE 中用 PRIMARY KEY 定义

参照完整性：

在 CREATE TABLE 中用 FOREIGN KEY 短语定义哪些列为外码

用 REFERENCES 短语指明这些外码参照哪些表的主码

用户定义的完整性：

针对某一具体应用的数据必须满足的语义要求。

列值非空（NOT NULL）、列值唯一（UNIQUE）、检查列值是否满足一个条件表达式（CHECK）

- 参照完整性的违约情况与处理方式

情况：

对参照表和被参照表进行增删改操作时有可能破坏参照完整性，必须进行检查

处理方式：

（1） 拒绝（NO ACTION）执行

不允许该操作执行。该策略一般设置为默认策略

（2） 级联（CASCADE）操作

当删除或修改被参照表（Student）的一个元组造成了与参照表（SC）的不一致，则删除或修改参照表中的所有造成不一致的元组

（3）设置为空值（SET-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。

- CHECK 子句

限制每个学期每一门课程最多 60 名学生选修

首先需要修改 SC 表的模式，增加一个“学期（TERM）”属性

```
ALTER TABLE SC ADD TERM DATE;
```

然后，定义断言：

```
CREATE ASSERTION ASSE_SC_CNUM2
CHECK(60 >= ALL (SELECT count(*)
                 FROM SC
                 GROUP by cno,TERM)
                );
```

当学生的性别是男时，其名字不能以 Ms.打头。

```
CREATE TABLE Student
      (  Sno      CHAR(9),
         Sname    CHAR(8) NOT NULL，
         Ssex     CHAR(2),
         Sage     SMALLINT,
```

Sdept CHAR(20),
PRIMARY KEY (Sno),
CHECK (Ssex='女' OR Sname NOT LIKE 'Ms.%')

5、关系数据理论

- 函数依赖：函数依赖的定义以及现实世界的语义表达，关系的码和外码。

定义：

设 R(U)是一个属性集 U 上的关系模式，X 和 Y 是 U 的子集。若对于 R(U)的任意一个可能的关系 r，r 中不可能存在两个元组在 X 上的属性值相等， 而在 Y 上的属性值不等， 则称“X 函数确定 Y”或“Y 函数依赖于 X”，记作 X→Y。

在 R(U)中，如果 X→Y，并且对于 X 的任何一个真子集 X’，都有 X’ ↛ Y，则称 Y 对 X 完全函数依赖，记作 X → Y。

若 X→Y，但 Y 不完全函数依赖于 X，则称 Y 对 X 部分函数依赖，记作 X → Y。

在 R(U)中，如果 X→Y(Y⊈ X)，Y↛X，Y→Z，Z⊈ Y，则称 Z 对 X 传递函数依赖(transitive functional dependency)。记为：X → Z。

超码：

设 K 为 R<U,F>中的属性或属性组合。若 K → U，则 K 称为 R 的一个候选码(Candidate Key)。

如果 U 部分函数依赖于 K，即 K → U,则 K 称为超码（Surpkey）。

候选码是最小的超码，即 K 的任意一个真子集都不是候选码。

外码：

关系模式 R 中属性或属性组 X 并非 R 的码，但 X 是另一个关系模式的码，则称 X 是 R 的外部码（Foreign key）也称外码。

- 函数依赖的 Armstrong 公理系统，推导：

我们可以使用以下三条规则去寻找逻辑蕴涵的函数依赖。通过反复应用这些规则，可以找出给定 F 的全部 F^+。这组规则称为 **Armstrong 公理**(Armstrong's axiom)，以纪念首次提出这一公理的人。

- **自反律**(reflexivity rule)。若 α 为一属性集且 $\beta \subseteq \alpha$，则 $\alpha \rightarrow \beta$ 成立。
- **增补律**(augmentation rule)。若 $\alpha \rightarrow \beta$ 成立且 γ 为一属性集，则 $\gamma\alpha \rightarrow \gamma\beta$ 成立。
- **传递律**(transitivity rule)。若 $\alpha \rightarrow \beta$ 和 $\beta \rightarrow \gamma$ 成立，则 $\alpha \rightarrow \gamma$ 成立。

Armstrong 公理是**正确有效的**(sound)，因为它们不产生任何错误的函数依赖。这些规则是**完备的**(complete)，因为，对于给定函数依赖集 F，它们能产生全部 F^+。文献注解提供了对正确有效性和完备性的证明的参考。

虽然 Armstrong 公理是完备的，但是直接用它们计算 F^+ 会很麻烦。为进一步简化，我们列出另外的一些规则。可以用 Armstrong 公理证明这些规则是正确有效的(参见实践习题 8.4、8.5 及习题 8.26)。

- **合并律**(union rule)。若 $\alpha \rightarrow \beta$ 和 $\alpha \rightarrow \gamma$ 成立，则 $\alpha \rightarrow \beta\gamma$ 成立。
- **分解律**(decomposition)。若 $\alpha \rightarrow \beta\gamma$ 成立，则 $\alpha \rightarrow \beta$ 和 $\alpha \rightarrow \gamma$ 成立。
- **伪传递律**(pseudotransitivity rule)。若 $\alpha \rightarrow \beta$ 和 $\gamma\beta \rightarrow \delta$ 成立，则 $\alpha\gamma \rightarrow \delta$ 成立。

```
F⁺ = F
    repeat
        for each F⁺中的函数依赖 f
            在 f 上应用自反律和增补律
            将结果加入到 F⁺中
        for each F⁺中的一对函数依赖 f₁和 f₂
            if f₁和 f₂可以使用传递律结合起来
                将结果加入到 F⁺中
    until F⁺不再发生变化
```

- 范式及分解：1NF、2NF、3NF、BCNF

1NF：

作为二维表，关系要符合一个最基本的条件：每个分量必须是不可分开的数据项。满足了这个条件的关系模式就属于第一范式（1NF）。

2NF：

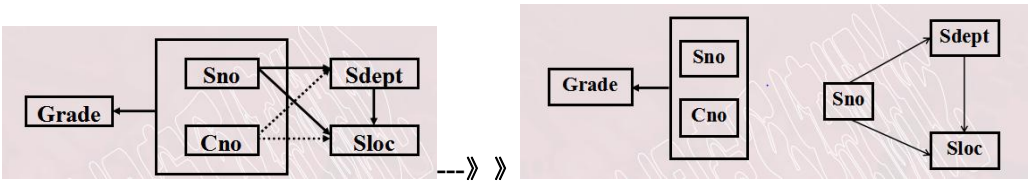
若关系模式 R∈1NF，并且每一个非主属性都完全函数依赖于任何一个候选码，则 R∈2NF。

分解：

用投影分解把关系模式 S-L-C 分解成两个关系模式

SC(Sno,Cno,Grade)

S-L(Sno,Sdept,Sloc)



3NF：

设关系模式 R<U,F>∈1NF,若 R 中不存在这样的码 X、属性组 Y 及非主属性 Z（Z ⊇ Y），使得 X→Y，Y→Z 成立，Y ↛ X 不成立，则称 R<U,F> ∈ 3NF。

即表中不存在可以确定其他非关键字的非键字段。

接 2NF 的例子：

SC 没有传递依赖，因此 SC ∈ 3NF

S-L 中 Sno →Sdept(Sdept ↛ Sno), Sdept→Sloc，可得 Sno → Sloc。

解决的办法是将 S-L 分解成

S-D(Sno,Sdept)∈ 3NF

D-L(Sdept,Sloc)∈ 3NF

BCNF：

设关系模式 R<U,F>∈1NF，若 X →Y 且 Y ⊆ X 时 X 必含有码，则 R<U,F>∈BCNF。

换言之，在关系模式 R<U,F>中，如果每一个决定属性集都包含候选码，则 R∈BCNF。

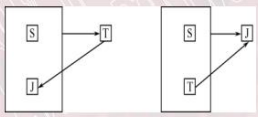
也即只要属性或属性组 A 能够决定任何一个属性 B，则 A 的子集中必须有候选键。

❖**[例6.7]** 关系模式**SJP(S,J,P)**中，**S**是学生，**J**表示课程，**P**表示名次。每一个学生选修每门课程的成绩有一定的名次，每门课程中每一名次只有一个学生（即没有并列名次）。

- 由语义可得到函数依赖：(S,J)→P；(J,P)→S
- (S,J)与(J,P)都可以作为候选码。
- 关系模式中没有属性对码传递依赖或部分依赖，所以SJP∈3NF。
- 除(S,J)与(J,P)以外没有其他决定因素，所以SJP∈BCNF。

❖**[例6.8]** 关系模式**STJ(S,T,J)**中，**S**表示学生，**T**表示教师，**J**表示课程。每一教师只教一门课。每门课有若干教师，某一学生选定某门课，就对应一个固定的教师。

- 由语义可得到函数依赖：(S,J)→T；(S,T)→J；T→J
- 因为没有任何非主属性对码传递依赖或部分依赖，STJ ∈ 3NF。
- 因为T是决定因素，而T不包含码，所以STJ ∉ BCNF关系。



非 BCNF 的关系模式也可以通过分解成为 BCNF。

例如 STJ 可分解为 ST(S,T)与 TJ(T,J)，它们都是 BCNF。

- 属性集闭包、关系码、极小函数依赖集(即正则覆盖、最小覆盖)的求解算法。

属性集闭包：

如果 $\alpha \rightarrow B$，我们称属性 B 被 α **函数确定**(functionally determine)。要判断集合 α 是否为超码，我们必须设计一个算法，用于计算被 α 函数确定的属性集。一种方法是计算 F^+，找出所有左半部为 α 的函数依赖，并合并这些函数依赖的右半部。但是这么做开销很大，因为 F^+ 可能很大。

在本节后面我们将看到，一个用于计算被 α 函数确定的属性集的高效算法不仅可以用来判断 α 是否为超码，还可以用于其他的一些任务。

令 α 为一个属性集。我们将函数依赖集 F 下被 α 函数确定的所有属性的集合称为 F 下 α 的**闭包**，记为 α^+。图 8-8 是以伪码写的计算 α^+ 的算法。输入是函数依赖集 F 和属性集 α。输出存储在变量 *result* 中。

```
result := α;
    repeat
        for each 函数依赖 β→γ in F do
            begin
                if β ⊆ result then result := result ∪ γ;
            end
    until (result 不变)
```

正则覆盖：

F 的**正则覆盖**(canonical cover) F_c 是一个依赖集，使得 F 逻辑蕴含 F_c 中的所有依赖，并且 F_c 逻辑蕴涵 F 中的所有依赖。此外，F_c 必须具有如下性质：

- F_c 中任何函数依赖都不含无关属性。
- F_c 中函数依赖的左半部都是唯一的。即，F_c 中不存在两个依赖 $\alpha_1 \rightarrow \beta_1$ 和 $\alpha_2 \rightarrow \beta_2$，满足 $\alpha_1 = \alpha_2$。

函数依赖集 F 的正则覆盖可以按图 8-9 中描述的那样计算。需要重视的一点是，当检验一个属性是否无关时，检验时用的是 F_c 当前值中的函数依赖，而不是 F 中的依赖。如果一个函数依赖的右半部只包含一个属性，例如 $A \rightarrow C$，并且这个属性是无关的，那么我们将得到一个右半部为空的函数依赖。这样的函数依赖应该删除。

```
F_c = F
    repeat
        使用合并律将 F_c 中所有形如 α1→β1 和 α1→β2 的依赖
            替换为 α1→β1β2
        在 F_c 中寻找一个函数依赖 α→β，它在 α 或在 β 中
            具有一个无关属性
            /* 注意，使用 F_c 而非 F 检验无关属性 */
        如果找到一个无关属性，则将它从 F_c 中的 α→β 中删除
    until (F_c 不变)
```

考虑模式 (A, B, C) 上的如下函数依赖集 F：

$$A \rightarrow BC$$
$$B \rightarrow C$$
$$A \rightarrow B$$
$$AB \rightarrow C$$

让我们来计算 F 的正则覆盖。

- 存在两个函数依赖在箭头左边具有相同的属性集：

$$A \rightarrow BC$$
$$A \rightarrow B$$

我们将这些函数依赖合并成 $A \rightarrow BC$。

- A 在 $AB \rightarrow C$ 中是无关的，因为 F 逻辑蕴涵 $(F - \{AB \rightarrow C\}) \cup \{B \rightarrow C\}$。这个断言为真，因为 $B \rightarrow C$ 已经在函数依赖集中。
- C 在 $A \rightarrow BC$ 中是无关的，因为 $A \rightarrow BC$ 被 $A \rightarrow B$ 和 $B \rightarrow C$ 逻辑蕴涵。

于是，正则覆盖为：

$$A \rightarrow B$$
$$B \rightarrow C$$

- 多值依赖：多值依赖的定义；4NF 范式；

多值依赖：

设 R(U)是属性集 U 上的一个关系模式。X,Y,Z 是 U 的子集，并且 Z=U-X-Y。关系模式 R(U)中多值依赖 X→→Y 成立，当且仅当对 R(U)的任一关系 r，给定的一对(x,z)值，有一组 Y 的值，这组值仅仅决定于 x 值而与 z 值无关。

例　Teaching（C, T, B）

对于 C 的每一个值，T 有一组值与之对应，而不论 B 取何值。因此 T 多值依赖于 C，即 C→→T。

课程 C	教员 T	参考书 B
物理	李 勇 王 军	普通物理学 光学原理 物理习题集
数学	李 勇 张 平	数学分析 微分方程 高等代数
计算数学	张 平 周 峰	数学分析 …
…	…	…

4NF：

关系模式 R<U,F>∈1NF，如果对于 R 的每个非平凡多值依赖 X→→Y（Y ⊈ X），X 都含有码，则 R<U,F>∈4NF。限制关系模式的属性之间不允许有非平凡且非函数依赖的多值依赖。

6、数据库设计

- 数据库设计的六个阶段及其主要任务

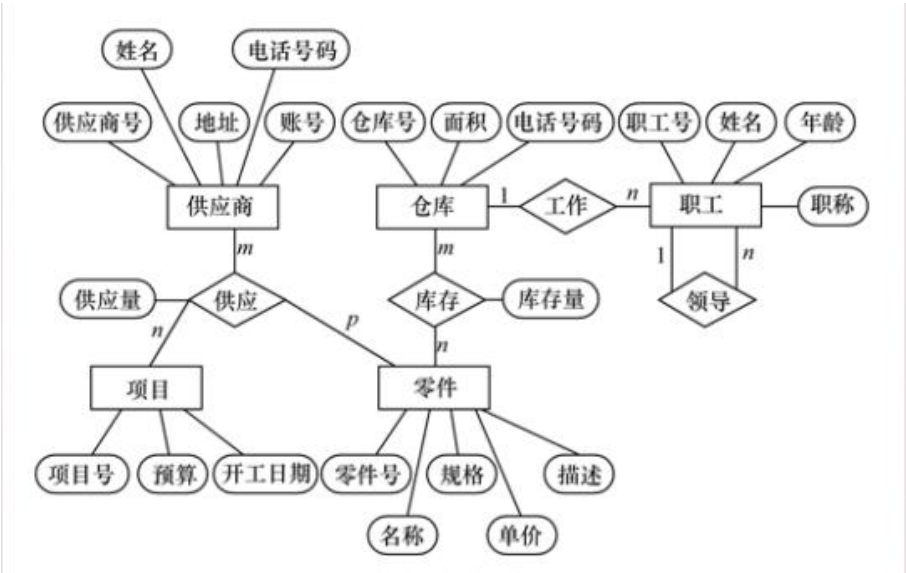
①需求分析：是否做得充分与准确，决定了构建数据库的速度和质量
②概念结构设计：通过对用户需求进行综合、归纳与抽象，形成一个独立于具体数据库管理系统的概念模型
③逻辑结构设计：将概念结构转换为某个数据库管理系统所支持的数据模型，并对其进行优化
④物理结构设计：为逻辑数据结构选取一个最适合应用环境的物理结构，包括存储结构和存取方法
⑤数据库实施：根据逻辑设计和物理设计的结果构建数据库，编写与调试应用程序，组织数据入库并进行试运行
⑥数据库运行和维护：经过试运行后即可投入正式运行，在运行过程中必须不断对其进行评估、调整与修改

- ER 模型的基本概念

E-R 图提供了表示实体型、属性和联系的方法，是用来描述现实世界的概念模型。

- 设计 ER 模型，将 E-R 图向关系模型的转换。（1 题综合题）

实体与属性的划分原则：
为了简化 E-R 图的处置，现实世界的事物能作为属性对待的，尽量作为属性对待。
两条准则：
（1）作为属性，不能再具有需要描述的性质。属性必须是不可分的数据项，不能包含其他属性。
（2）属性不能与其他实体具有联系，即 E-R 图中所表示的联系是实体之间的联系。
E-R 图例子：
实体型：用矩形表示，矩形框内写明实体名。
属性：用椭圆形表示，并用无向边将其与相应的实体型连接起来。
联系：用菱形表示，菱形框内写明联系名，并用无向边分别与有关实体型连接起来，同时在无向边旁标上联系的类型（1∶1，1∶n 或 m∶n）。



转换例子：

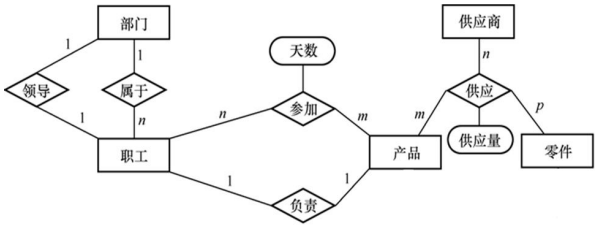
（1） 一个 1:1 联系可以转换为一个独立的关系模式，也可以与任意一端对应的关系模式合并。

（2） 一个 1:n 联系可以转换为一个独立的关系模式，也可以与 n 端对应的关系模式合并。

（3） 一个 m:n 联系转换为一个关系模式；关系的属性：与该联系相连的各实体的码以及联系本身的属性；关系的码：各实体码的组合。

（4） 三个或三个以上实体间的一个多元联系转换为一个关系模式。

（5） 具有相同码的关系模式可合并。



- 部门（部门号，部门名，经理的职工号，…）
- 职工（职工号、部门号，职工名，职务，…）
- 产品（产品号，产品名，产品组长的职工号，…）
- 供应商（供应商号，姓名，…）
- 零件（零件号，零件名，…）
- 职工工作（职工号，产品号，工作天数，…）
- 供应（产品号，供应商号，零件号，供应量）

7、数据库查询与优化

- 数据库查询处理的基本过程及查询优化的基本策略与方法。

查询处理：查询分析--》查询检查--》查询优化--》查询执行。

查询优化：

Q1= π Sname(σ Student.Sno=SC.Sno∧Sc.Cno='2' (Student×SC))

Q2= π Sname(σ Sc.Cno='2' (Student∞SC))

Q3= π Sname(Student∞ σ SC.Cno='2'(SC))

- 有选择和连接操作时，先做选择操作，这样参加连接的元组就可以大大减少，这是代数优化

- 对于 Student 和 SC 表的连接，利用 Student 表上的索引，采用索引连接代价也较小，这就是物理优化。

查询优化的选择依据：

基于规则(rule based)

基于代价(cost based)

基于语义(semantic based)

- 代数优化
  - 常用关系代数等价式

代数优化策略：通过对关系代数表达式的等价变换来提高查询效率。

❖常用的等价变换规则：

1.连接、笛卡尔积交换律

设E_1和E_2是关系代数表达式，F是连接运算的条件，则有

$E_1 \times E_2 \equiv E_2 \times E_1$

$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$

$E_1 \underset{F}{\bowtie} E_2 \equiv E_2 \underset{F}{\bowtie} E_1$

2.连接、笛卡尔积的结合律

设E_1，E_2，E_3是关系代数表达式，F_1和F_2是连接运算的条件

$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$

$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$

$(E_1 \underset{F_1}{\bowtie} E_2) \underset{F_2}{\bowtie} E_3 \equiv E_1 \underset{F_1}{\bowtie} (E_2 \underset{F_2}{\bowtie} E_3)$

3.投影的串接定律

$\pi_{A_1,A_2,\cdots A_n}(\pi_{B_1,B_2,\cdots B_m}(E)) \equiv \pi_{A_1,A_2,\cdots A_n}(E)$

- E是关系代数表达式
- $A_i(i=1，2，\ldots，n)$，$B_j(j=1，2，\ldots，m)$是属性名
- $\{A_1，A_2，\ldots，A_n\}$构成$\{B_1，B_2，\ldots，B_m\}$的子集

4.选择的串接定律

$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$

- E是关系代数表达式，F_1、F_2是选择条件
- 选择的串接律说明选择条件可以合并,这样一次就可检查全部条件

5.选择与投影操作的交换律

$\sigma_F(\pi_{A_1,A_2,\cdots A_n}(E)) \equiv \pi_{A_1,A_2,\cdots A_n}(\sigma_F(E))$

- 选择条件F只涉及属性A_1，…，A_n。
- 若F中有不属于A_1，…，A_n的属性B_1，…，B_m有更一般规则：

$\pi_{A_1,A_2,\cdots A_n}(\sigma_F(E)) \equiv \pi_{A_1,A_2,\cdots A_n}(\sigma_F(\pi_{A_1,A_2,\cdots A_n,B_1,B_2,\cdots B_m}(E)))$

6. 选择与笛卡尔积的交换律

- 如果F中涉及的属性都是E_1中的属性，则
  $\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$
- 如果$F=F_1 \wedge F_2$，并且F_1只涉及E_1中的属性，F_2只涉及E_2中的属性，则由上面的等价变换规则1，4，6可推出：
  $\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2)$
- 若F_1只涉及E_1中的属性，F_2涉及E_1和E_2两者的属性，则仍有
  $\sigma_F(E_1 \times E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \times E_2)$
  它使部分选择在笛卡尔积前先做。

7. 选择与并的分配律

设$E=E_1 \cup E_2$，E_1，E_2有相同的属性名，则

$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$

8. 选择与差运算的分配律

若E_1与E_2有相同的属性名，则

$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$

9. 选择对自然连接的分配律

$\sigma_F(E_1 \bowtie E_2) \equiv \sigma_F(E_1) \bowtie \sigma_F(E_2)$

F只涉及E_1与E_2的公共属性

10. 投影与笛卡尔积的分配律

设E_1和E_2是两个关系表达式，A_1，…，A_n是E_1的属性，B_1，…，B_m是E_2的属性，则

$\pi_{A_1,A_2,\cdots A_n,B_1,B_2,\cdots B_m}(E_1 \times E_2) \equiv \pi_{A_1,A_2,\cdots A_n}(E_1) \times \pi_{B_1,B_2,\cdots B_m}(E_2)$

11. 投影与并的分配律

设E_1和E_2有相同的属性名，则

$\pi_{A_1,A_2,\cdots A_n}(E_1 \cup E_2) \equiv \pi_{A_1,A_2,\cdots A_n}(E_1) \cup \pi_{A_1,A_2,\cdots A_n}(E_2)$

- 物理优化
  - 对于选择运算,能够对全表扫描算法、索引选择算法的开销进行估算

物理优化就是要选择高效合理的操作算法或存取路径，求得优化的查询计划。

全表扫描：

- 如果基本表大小为 B 块，全表扫描算法的代价 cost＝B。

- 如果选择条件是“码＝值”，那么平均搜索代价 costB/2。

索引扫描算法的代价估算公式：

①如果选择条件是“码＝值”则采用该表的主索引

- 若为 B+树，层数为 L，需要存取 B+树中从根结点到叶结点 L 块，再加上基本表中该元组所在的那一块，所以 cost=L+1

②如果选择条件涉及非码属性

- 若为 B+树索引，选择条件是相等比较，S 是索引的选择基数(有 S 个元组满足条件)

- 满足条件的元组可能会保存在不同的块上，所以(最坏的情况)cost=L+S。

③如果比较条件是＞，＞＝，＜，＜＝操作，假设有一半的元组满足条件,就要存取一半的叶结点。

- 通过索引访问一半的表存储块 cost=L+Y/2+B/2

- 如果可以获得更准确的选择基数，可以进一步修正 Y/2 与 B/2。

8、数据库恢复技术

- 事务的基本概念和基本特征（ACID 特性）。

基本概念：

事务(Transaction)是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。也是恢复和并发控制的基本单位。

基本特征（ACID）：

- 原子性（Atomicity）：事务是数据库的逻辑工作单位，事务中包括的诸操作要么都做，要么都不做。

- 一致性（Consistency）：事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。

- 隔离性（Isolation）：一个事务的执行不能被其他事务干扰。

- 持续性（Durability ）：一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其执行结果有任何影响。

- 数据库故障的种类。

1. 事务内部的故障
2. 系统故障：称为软故障，是指造成系统停止运转的任何事件，使得系统要重新启动。
3. 介质故障：称为硬故障，指外存故障。介质故障破坏数据库或部分数据库，并影响正在存取这部分数据的所有事务 。介质故障比前两类故障的可能性小得多，但破坏性大得多
4. 计算机病毒

- 数据库恢复的实现技术：数据转储、登记日志文件。

数据库管理系统必须具有把数据库从错误状态恢复到某一已知的正确状态(亦称为一致状态或完整状态)的功能，这就是数据库的恢复管理系统对故障的对策。

数据转储：

转储是指数据库管理员定期地将整个数据库复制到磁带、磁盘或其他存储介质上保存起来的过程。

数据库遭到破坏后可以将后备副本重新装入，要想恢复到故障发生时的状态，必须重新运行自转储以后的所有更新事务。

转储方法：

转储方式	转储状态	
	动态转储	静态转储
海量转储	动态海量转储	静态海量转储
增量转储	动态增量转储	静态增量转储

- 日志文件、日志记录的含义

定义：

日志文件(log file)是用来记录事务对数据库的更新操作的文件。

以记录为单位的日志文件内容：

- 各个事务的开始标记(BEGIN TRANSACTION)
- 各个事务的结束标记(COMMIT 或 ROLLBACK)
- 各个事务的所有更新操作

TIP：以上均作为日志文件中的一个日志记录。

每条日志记录的内容：

- 事务标识（标明是哪个事务）

- 操作类型（插入、删除或修改）

- 操作对象（记录 ID、Block NO.）

- 更新前数据的旧值（对插入操作而言，此项为空值）

- 更新后数据的新值（对删除操作而言，此项为空值）

登记日志文件原则：

- 登记的次序严格按并发事务执行的时间次序

- 必须先写日志文件，后写数据库

日志文件作用：

- 把已完成的事务进行重做处理

- 对故障发生时尚未完成的事务进行撤销处理

- 不必重新运行那些已完成的事务程序，就可把数据库恢复到故障前某一时刻的正确状态

- 恢复策略：事务故障的恢复、系统故障的恢复、介质故障的恢复。

事务故障：

由恢复子系统利用日志文件撤消（UNDO）此事务已对数据库进行的修改。

步骤：

（1）反向扫描文件日志（即从最后向前扫描日志文件），查找该事务的更新操作。

（2）对该事务的更新操作执行逆操作。即将日志记录中“更新前的值” 写入数据库。

（3） 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。

（4） 如此处理下去，直 至读到此事务的开始标记，事务故障恢复就完成了。

系统故障：

1. Undo 故障发生时未完成的事务

2. Redo 已完成的事务

步骤：

（1）正向扫描日志文件（即从头扫描日志文件）

重做(REDO) 队列: 在故障发生前已经提交的事务（有 COMMIT 记录）

撤销 (UNDO)队列:故障发生时尚未完成的事务（无相应的 COMMIT 记录）

（2）对撤销(UNDO)队列事务进行撤销(UNDO)处理

（3）对重做(REDO)队列事务进行重做(REDO)处理

介质故障（需要数据库管理员介入）：

1.重装数据库

2.重做已完成的事务

步骤：

（1） 装入最新的后备数据库副本(离故障发生时刻最近的转储副本) ，使数据库恢复到最近一次转储时的一致性状态。

- 对于静态转储的数据库副本，装入后数据库即处于一致性状态

- 对于动态转储的数据库副本，还须同时装入转储时刻的日志文件副本，利用恢复系统故障的方法（即 REDO+UNDO），才能将数据库恢复到一致性状态。

（2） 装入有关的日志文件副本(转储结束时刻的日志文件副本) ，重做已完成的事务。

- 首先扫描日志文件，找出故障发生时已提交的事务的标识，将其记入重做队列。

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

9、并发控制

- 并发操作可能带来的数据不一致现象。

1. 丢失修改（Lost Update）：两个事务 T1 和 T2 读入同一数据并修改，T2 的提交结果破坏了 T1 提交的结果，导致 T1 的修改被丢失。

① 甲售票点(事务T_1)读出某航班的机票余额A，设A=16；
② 乙售票点(事务T_2)读出同一航班的机票余额A，也为16；
③ 甲售票点卖出一张机票，修改余额A←A-1，所以A为15，把A写回数据库；
④ 乙售票点也卖出一张机票，修改余额A←A-1，所以A为15，把A写回数据库
■ 结果明明卖出两张机票，数据库中机票余额只减少1

2. 不可重复读（Non-repeatable Read）：不可重复读是指事务 T1 读取数据后，事务 T2 执行更新操作，使 T1 无法再现前一次读取结果。

①事务 T1 读取某一数据后，事务 T2 对其做了修改，当事务 T1 再次读该数据时，得到与前一次不同的值。

例如：

T_1	T_2
① R(A)=50	
R(B)=100	
求和=150	
②	R(B)=100
	B←B*2
	W(B)=200
③ R(A)=50	
R(B)=200	
求和=250	
(验算不对)	

不可重复读

■ T_1读取B=100进行运算
■ T_2读取同一数据B，对其进行修改后将B=200写回数据库。
■ T_1为了对读取值校对重读B，B已为200，与第一次读取值不一致

②事务 T1 按一定条件从数据库中读取了某些数据记录后，事务 T2 删除了其中部分记录，当 T1 再次按相同条件读取数据时，发现某些记录神秘地消失了。

③事务 T1 按一定条件从数据库中读取某些数据记录后，事务 T2 插入了一些记录，当 T1 再次按相同条件读取数据时，发现多了一些记录。

3. 读“脏”数据（Dirty Read）：事务 T1 修改某一数据，并将其写回磁盘；事务 T2 读取同一数据后，T1 由于某种原因被撤销；这时 T1 已修改过的数据恢复原值，T2 读到的数据就与数据库中的数据不一致；T2 读到的数据就为“脏”数据，即不正确的数据。

例如

T_1	T_2
① R(C)=100	
C←C*2	
W(C)=200	
②	R(C)=200
③ ROLLBACK	
C恢复为100	

- 封锁、封锁协议、死锁的概念。
  - 等待图判断是否存在死锁

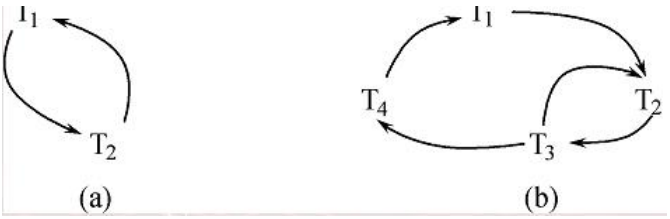
- 封锁就是事务 T 在对某个数据对象（例如表、记录等）操作之前，先向系统发出请求，对其加锁。（排他锁、共享锁）

- 在运用 X 锁和 S 锁对数据对象加锁时，需要约定一些规则，这些规则为封锁协议。

- 产生死锁的原因是两个或多个事务都已封锁了一些数据对象，然后又都请求对已为其他事务封锁的数据对象加锁，从而出现死等待。（解决：一次封锁、顺序封锁）

等待图：

用事务等待图动态反映所有事务的等待情况，事务等待图是一个有向图 G=(T，U)，T 为结点的集合，每个结点表示正运行的事务，U 为边的集合，每条边表示事务等待的情况，若 T1 等待 T2，则 T1，T2 之间划一条有向边，从 T1 指向 T2。



如果发现图中存在回路，则表示系统中出现了死锁。

- 并发调度的可串行性。
  - 可串行化调度

概念：执行结果等价于串行调度的调度也是正确的，称为可串行化调度。

  - 冲突可串行化调度

冲突操作：是指不同的事务对同一数据的读写操作和写写操作：

①Ri(x)与 Wj(x)　　/*事务 Ti 读 x，Tj 写 x，其中 i≠j*/

②Wi(x)与 Wj(x)　　/*事务 Ti 写 x，Tj 写 x，其中 i≠j*/

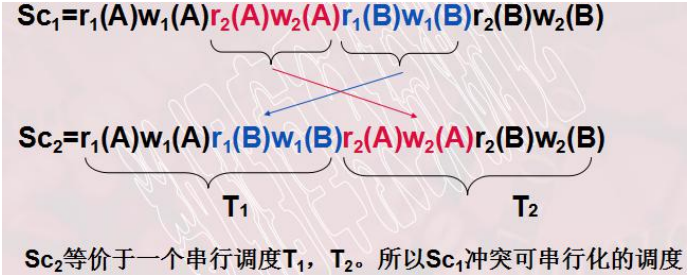
TIP：若一个调度是冲突可串行化，则一定是可串行化的调度。

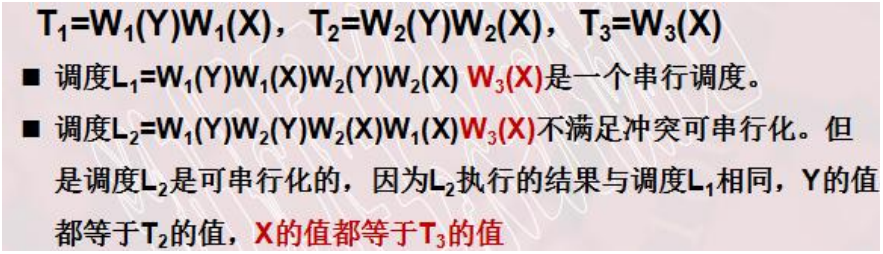
不能交换（Swap）的动作：

①同一事务的两个操作

②不同事务的冲突操作

概念：一个调度 Sc 在保证冲突操作的次序不变的情况下，通过交换两个事务不冲突操作的次序得到另一个调度 Sc'，如果 Sc'是串行的，称调度 Sc 是冲突可串行化的调度。





- 两段锁协议。
  - 应用方法

概念：指所有事务必须分两个阶段对数据项加锁和解锁。

- 第一阶段是获得封锁，也称为扩展阶段：事务可以申请获得任何数据项上的任何类型的锁，但是不能释放任何锁
- 第二阶段是释放封锁，也称为收缩阶段：事务可以释放任何数据项上的任何类型的锁，但是不能再申请任何锁

事务遵守两段锁协议是可串行化调度的充分条件，而不是必要条件。

事务T_1	事务T_2
Slock A	
R(A)=260	
	Slock C
	R(C)=300
Xlock A	
W(A)=160	
	Xlock C
	W(C)=250
	Slock A
Slock B	等待
R(B)=1000	等待
Xlock B	等待
W(B)=1100	等待
Unlock A	等待
	R(A)=160
	Xlock A
Unlock B	
	W(A)=210
	Unlock C

和一次封锁法的比较：

- 一次封锁法要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行，因此一次封锁法遵守两段锁协议
- 两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，因此遵守两段锁协议的事务可能发生死锁