数据库复习资料

XOR[xor@pku.edu.cn]

- 本资料根据 09 年 1 月份 db 课程 ppt、out-line.ppt 和课本总结而来。
- 如有错误请自行修改，作者不保证正确性，因为本文档中的错误导致的后果作者不负责。
- 本资料仅仅起到总结作用，复习时候一定要同时参照书本和讲义 ppt 进行复习。
- 最后附上了所有作业及答案，供大家复习时候参考。

目录

第二章　实体-联系模型

一、 基本概念

1.实体、联系、属性

实体： **客观存在并可相互区分的事务叫实体。**
如学生张三、工人李四、计算机系、数据库概论。

联系： **实体之间的相互关联。**
如学生与老师间的授课关系，学生与学生之间有班长关系。
联系也可以有属性，如学生与课程之间有选课联系，每个选课联系都有一个成绩作为其属性。

属性： **实体所具有的某一特性。**
一个实体可以有若干个属性来刻画
例如：学生可由学号、姓名、年龄、系等组成。

2.超码、候选码、主码

超码： **能唯一标识实体的属性或属性组称作超码。**超码的任意超集也是超码。

候选码： **其任意真子集都不能成为超码的最小超码称为候选码。**

主码： **从所有候选码中选定一个用来区别同一实体集中的不同实体，称作主码。**
一个实体集中，任意两个实体在主码上的取值不能相同。如学号是学生实体的码。

3.联系的种类、联系的势

联系的种类： **实体之间的联系的数量**，即一个实体通过一个联系集能与另一实体集相关联的实体的数目。可以有一对一的（1:1），一对多的（1:m），多对多的（m:n）几种情况。注意在 E-R 图中的表示。

一对一： E_1 中的一个实体与 E_2 中至多一个实体相联系，并且 E_2 中的一个实体与 E_1 中至多一个实体相联系。注：一对一不是一一对应。

一对多： E_1 中的一个实体与 E_2 中 $n(n \geq 0)$ 个实体相联系，并且 E_2 中的一个实体与 E_1 中至多一个实体相联系。

多对多： E_1 中的一个实体与 E_2 中 $n(n \geq 0)$ 个实体相联系，并且 E_2 中的一个实体与 E_1 E_1 中 $m(m \geq 0)$ 一个实体相联系。

联系的势： 势表达了一个实体出现在联系中的次数。注意在 E-R 图中的表示。

4.弱实体、特殊化、概括、聚集

弱实体集：如果一个实体集的所有属性都不足以形成主码，则称这样的实体集为弱实体集。弱实体集与其拥有者之间的联系称作标识性联系(identifying relationship)。弱实体集与强实体集之间是一对多的联系。弱实体集必然存在依赖于强实体集（Strong Entity Set)。存在依赖并不总会导致一个弱实体集，从属实体集可以有自己的主码。弱实体集中用于区别依赖于某个特定强实体集的属性集合。也称作部分码（partial key)。如“还款”中的还款号，Logins 中的用户名。弱实体集的主码由该弱实体集所存在依赖的强实体集的主码和该弱实体集的分辨符组成如“还款”主码＝贷款号＋还款号。注意看弱实体集在Ｅ－Ｒ图中的表示。

为什么使用弱实体集?

避免数据冗余（强实体集码重复），以及因此带来的数据的不一致性。

弱实体集反映了一个实体对其它实体依赖的逻辑结构。

弱实体集可以随它们的强实体集的删除而自动删除。

弱实体集可以物理地随它们的强实体集存储。

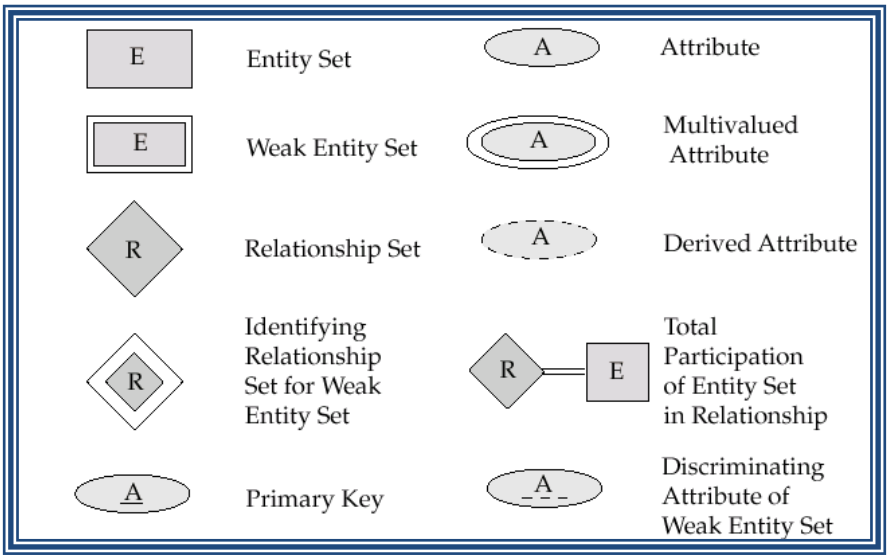
特殊化：实体集中某些子集具有区别于该实体集内其它实体的特性，可以根据这些差异特性对实体集进行分组，这一分组的过程称作特殊化（自顶向下、逐步求精的数据库设计过程）在Ｅ－Ｒ同中为 ISA 的三角形。

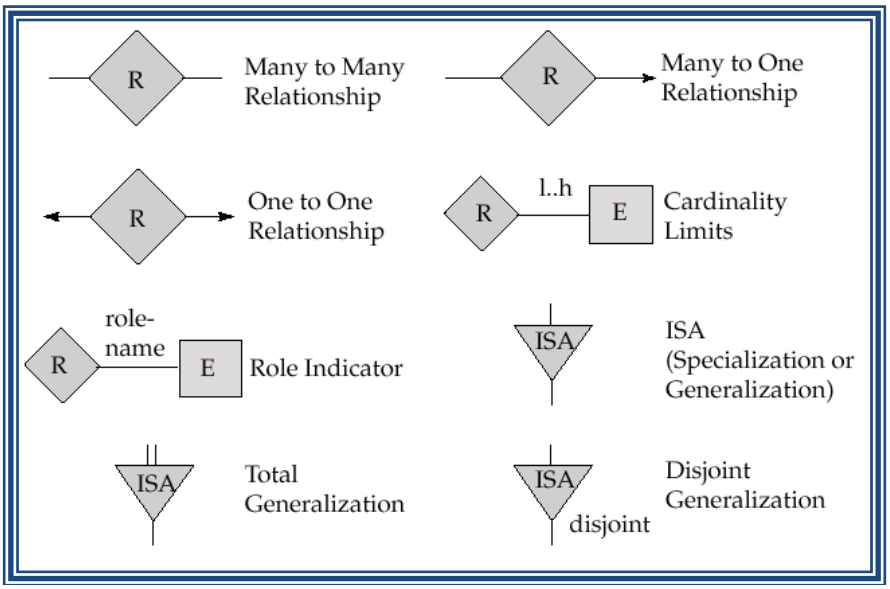
概括：各个实体集根据共有的性质，合成一个较高层的实体集。概括是一个高层实体集与若干个低层实体集之间的包含关系（自底向上、逐步合成的数据库设计过程）

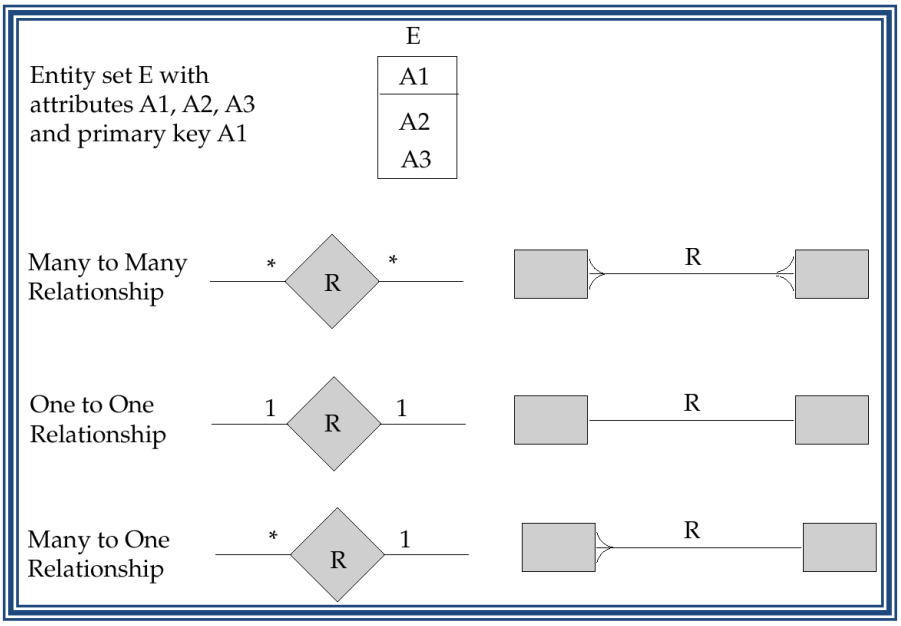
聚集：聚集是一种抽象，通过它联系被作为高层实体集。实体集 A 与 B 以及它们的联系可被看成实体集，并与另一实体集 C 发生联系。

二、 E-R 图表示方法

请注意看书和讲义上相关的章节，注意多练习，尤其是箭头的意义，不要混淆。







三、 E-R 图向关系模式的转换

1. 强实体集

设 E 是具有描述性属性 $a_1, a_2, ..., a_n$ 的强实体集。我们用具有 n 个不同属性的模式 E 来表示这个实体集。该模式上的一个关系的每个元组同实体集 E 的一个实体相对应。强实体集的主码就是模式的主码。

2.弱实体集

设 A 是具有属性 $a_1, a_2, ..., a_m$ 的弱实体集。设 B 是 A 所依赖的强实体集，且 B 的主码包括属性 $b_1, b_2, ..., b_n$。我们用名为 A 的关系模式表示实体集 A，该模式的每个属性对应于以下属性稽核的各个属性：

$$\{a_1, a_2, ..., a_m\} \cup \{b_1, b_2, ..., b_n\}$$

对于从弱实体集导出的模式，该模式的主码尤其所依赖的强实体集的主码与弱实体集的分辨符组合而成。除了创建主码之外，我们还要在关系 A 上建立外码约束，该约束指明属性 $b_1, b_2, ..., b_n$ 参照关系 B 的主码。

3.联系集

设 R 是联系集，而所有参与 R 的实体集的主码的并集形成属性集合 $a_1, a_2, ..., a_m$，如果 R 有描述属性则不妨设为 $b_1, b_2, ..., b_n$。用名为 R 的关系模式表示该联系集，该模式的每个属性对应于以下属性集合的各个属性：

$$\{a_1, a_2, ..., a_m\} \cup \{b_1, b_2, ..., b_n\}$$

R 关系模式主码的选择采用以下方式：

对于多对多的二元联系，主码应该是参与实体集的主码属性的并集。

对于一对一的二元联系集，任何一个参与实体集的主码都可以作为联系的主码。

对于多对多的二元联系集，主码应该是联系集中“多”那一边的实体集的主码。

对于 n 元联系集，如果连接到它的边中没有箭头，则主码是所有参与实体集的主码属性的并。

对于 n 元联系集，如果连接到它的一条边中有一个箭头，则除去“箭头”那一边得实体集的主码属性，其他参与实体集的主码属性的并集作为联系的主码。

R 关系外码约束的建立：对于每个与联系集 R 相关的实体集 E_i，我们建立一个关系 R 上的外码约束，该约束表明 R 中来自 E_i 主码属性的那些属性参照关系 E_i 的主码。

4.连接弱实体集和其相应强实体集的联系

比较特殊，这个联系集的模式是冗余的，可以不必给出。

5. 模式的合并

在多对一的联系中，完全参与的实体集可以合并到关系模式中。在一对一的情况下，联系集的关系模式可以和跟参与联系的任何一个实体集的模式进行合并。即使是部分参与的实体集，也可以用空值来合并到关系模式中。合并模式的主码是其模式中融入了联系集模式的那个实体集的主码。外码约束随之转移即可。

6. 复合属性和多值属性

为每个子属性创建一个单独的属性来处理符合属性。对于多值属性 M ，我们为其创建关系模式 R ，该模式具有对应 M 的属性 A ，以及对应 M 所在的实体集或者联系集的主码的那些属性，主码由模式 R 中的所有属性组成，外码的属性生成自实体集的主码，参照该实体集所生成的关系。

7. 一般化

1. 为高层实体集创建一个模式。为每个低层实体集创建一个模式，模式中的属性包括对应于低层实体集的每个属性，以及对应该高层实体集主码的每个属性。高层实体集的主码属性既是高层实体集的主码属性又是低层实体集的主码属性。

2. 如果一般化是不相交并且是全部的，则可以不单独为高层实体集创建关系模式。只需在低层实体集的每一个关系中包含高层实体集的所有属性。

8. 聚集

实体集 A 与 B 以及它们的联系 R 被看成实体集 C ， C 与另一实体集 D 构成联系 S ，则 S 所对应的关系的码由 R 和 D 的码构成。

第三章　关系模型

一、 概念

1.关系模型的完整性

i. 实体完整性

关系的主码中的属性值不能为空值

空值：不知道或无意义

意义：关系对应到现实世界中的实体集，元组对应到实体，实体是相互可区分的，通过主码来唯一标识，若主码为空，则出现不可标识的实体，这是不容许的

ii. 参照完整性

如果关系 R_2 的外部码 F_k 与关系 R_1 的主码 P_k 相对应，则 R_2 中的每一个元组的 F_k 值或者等于 R_1 中某个元组的 P_k 值，或者为空值

意义：如果关系 R_2 的某个元组 t_2 参照了关系 R_1 的某个元组 t_1，则 t_1 必须存在

例如关系 S 在 $D^{\#}$ 上的取值有两种可能

空值，表示该学生尚未分到任何系中

若非空值，则必须是 $DEPT$ 关系中某个元组的 $D^{\#}$ 值，表示该学生不可能分到一个不存在的系中

iii. 用户定义的完整性

用户针对具体应用环境定义的完整性约束条件

如 $S^{\#}$ 要求是 8 位整数，SEX 要求取值为“男”或“女”

iv. 系统支持

实体完整性和参照完整性由系统自动支持

系统提供定义和检验用户定义的完整性的机制

2.视图及视图的作用

视图是命名的、从基本表中导出的虚表，它在物理上并不存在，存在的只是它的定义。视图中的数据是从基本表中导出的，每次对视图查询都要重新计算。

```
create view view_name as  <查询表达式>
```

视图之上可以再定义视图。

视图的优点

个性化服务：简化了用户观点，使不同用户可以从不同角度观察同一数据。

安全性：“知必所需”，限制用户数据的访问范围。

逻辑独立性：视图作为基本表与外模式之间的映象

视图之上可以再定义视图

二、 关系运算

选择、投影、更名、笛卡儿积、集合并、差、交、自然连接、除、外连接

选择(select)运算选出满足给定谓词的元组。举例：选择 loan 关系中支行名称为"Perryridge"的那些元组，应该写作

$$\sigma_{branch\_name="Perryridge"}(loan)$$

投影(project)运算返回作为参数的关系的某些属性。列出 loan 中所有贷款号码和金额的查询如下：

$$\Pi_{loan\_number,amount}(loan)$$

更名(rename)运算，将一个表达式赋予名字，并且返回表达式的值。举例：

$$\rho_{x(A_1,A_2,...,A_n)}(E)$$

笛卡儿积(Cartesian-product)运算。举例：

$$r = borrower \times loan$$

集合并(union)运算。举例：查询找出所有的有贷款或者有账户或者二者兼有的银行客户

$$\Pi_{customer\_name}(borrower) \cup \Pi_{customer\_name}(depositor)$$

集合差(set-difference)运算。举例：查询找出所有的有帐户而且无贷款的用户：

$$\Pi_{customer\_name}(depositor) - \Pi_{customer\_name}(depositor)$$

集合交(intersection)运算。举例：查询找出所有的有帐户并且有贷款的用户：

$$\Pi_{customer\_name}(depositor) \cap \Pi_{customer\_name}(depositor)$$

集合交的性质：$r \cap s = r - (r - s)$

自然连接(natural join)设$r(R)$和$s(S)$是两个关系，

$$R \cup S = \{A_1, A_2, ..., A_n\}$$

$$r \bowtie s = \prod_{R \cup S} (\sigma_{r.A_1=s.A_1 \wedge r.A_2=s.A_2 \wedge ... \wedge r.A_n=s.A_n} (r \times s))$$

除运算(division)用于表达“对所有的”的查询。设$r(R)$和$s(S)$是两个关系，则

$$r \div s = \prod_{R-S}(r) - \prod_{R-S}((\prod_{R-S}(r) \times s) - \prod_{R-S,S}(r))$$

外连接(outer-join)运算是连接运算的扩展，可以处理缺失的信息。

左外连接(left outer join)符号不好画，略。取出左侧关系中所由于右侧关系任意元组都不匹配的元组，用空值填充所有来自右侧关系的属性，再把产生的元组加到自然连接的结果上。

右外连接(right outer join)符号也不好画，略。取出右侧关系中所由于左侧关系任意元组都不匹配的元组，用空值填充所有来自左侧关系的属性，再把产生的元组加到自然连接的结果上。

全外连接(full outer join)符号更不好画，略。完成左外连接和右外连接运算。

三、 能用关系代数表达关系数据操作

举例：作业题

第四章　SQL

一、 能用 SQL 表达各种数据库查询操作

以下语句仅收集了常用的语句，不常用的语句请关注 PPT 和相关书籍。

1.基本表的定义（CREATE）

格式

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

例子

```
create table   S (
    S#     char（8）,
    SNAME  char（8） not null default ‘Unknown’,
    AGE    tinyint,
    SEX    char(1),primary key (S#),
    check (SEX=‘M’  or  SEX=‘F’)
)
create table   C(
    C#     char（4） primary key ,
    CNAME  char（8）  not null  unique,
    PC#    char（4）  foreign key references C(C#)
)
create table   SC(
       S#     char（8）,
       C#        char（4）,
       GRADE  tinyint,
       primary key (S#, C#),
       foreign key (S#)  references S(S#),
       foreign key (C#)  references C(C#),
       check((GRADE is null)  or  GRADE between 0 and 100))
)
```

2.修改基本表定义（ALTER）

更改、添加、除去列和约束

格式：

```
alter table 表名
    [add 子句]     增加新列和约束
    [drop 子句]    删除列和约束
    [modify 子句] 修改列定义
```

示例

```
alter table S add LOCATION char[30]
alter table S add resume char[100] not null
alter table S alter column resume char[80]
```

3.撤消基本表定义（drop）

格式

```
drop table 表名
```

删除表定义及该表的所有数据、索引、触发器、约束和权限规范。任何引用已除去表的视图或存储过程必须通过 DROP VIEW 或 DROP PROCEDURE 语句显式除去

DROP TABLE 不能用于除去由 FOREIGN KEY 约束引用的表。必须先除去引用的 FOREIGN KEY 约束或引用的表

4.私有临时表

```
create table #my_table
```

5.全局临时表

```
create table ##my_table
```

6.IDENTITY（属性）

在表中创建一个标识列。此属性与 CREATE TABLE 及 ALTER TABLE Transact-SQL 语句一起使用。为一些没有有效主码的表提供计数器，有一个起始数（种子），增量值（步长），IDENTITY 属性不能为空，也不能带有 default。

示例

```
create table customer1
   (cust_id     smallint  IDENTITY  not null,
   cust_name varchar(50)  not null)
create table customer2
   (cust_id     smallint  IDENTITY(100,20)  not null
cust_name varchar(50)  not null)
```

7.定义索引

格式

```
create  [unique]  [cluster]  index  索引名
on  表名 (列名 [asc/desc]  [ , 列名 asc/desc]…)
```

unique:唯一性索引,不允许表中不同的行在索引列上取相同值。若已有相同值存在,则系统给出相关信息,不建此索引。系统拒绝违背唯一性的插入、更新

cluster:聚簇索引,表中元组按索引项的值排序并物理地聚簇在一起。一个基本表上只能建一个聚簇索引

asc/desc:索引表中索引值的排序次序,缺省为 asc

8.删除索引

```
drop  index  索引名
```

DROP INDEX 语句不适用于通过定义 PRIMARY KEY 或 UNIQUE 约束创建的索引,它们必须通过除去约束来撤销

9.约束

PRIMARY KEY

作为主码的关系称为基本关系,作为外码的关系称为依赖关系

删除基本关系元组时

RESTRICT 方式只有当依赖关系中没有一个外码值与要删除的基本关系的主码值相对应时,才可以删除该元组,否则系统拒绝此删除操作

CASCADE 方式将依赖关系中所有外码值与基本关系中要删除的主码值所对应的元组一起删除

SET NULL 方式删除基本关系中元组时,将依赖关系中与基本关系中被删主码值相对应的外码值置为空值

修改基本关系主码时

RESTRICT 方式只有当依赖关系中没有一个外码值与要修改的基本关系的主码值相对应时,才可以修改该元组主码,否则系统拒绝此次修改

CASCADE 方式将依赖关系中所有与基本关系中要修改的主码值所对应的外码值一起修改

为新值

SET NULL 方式修改基本关系中元组主码时，将依赖关系中与基本关系中被修改主码值相对应的外码值置为空值

UNIQUE

unique (A_1, A_2,…, A_n)

unique 约束指出(A_1, A_2,…, A_n)形成了一个候选码，即没有任何两个元组能在所有主码属性上相等。然而候选码属性可以为空，除非他们已经被显式的声明为 not null。

FOREIGN KEY

```
foreign key (attr_name) references table_name
```

表达了参照完整性约束。

举例

```
foreign key(branch_name) references branch
                                on delete cascade
                                on update cascade
```

CHECK

```
check ( 谓词 )
```

用于保证属性值满足制定的谓词条件。

举例

```
check (degree_level in ('Bachelors',"Masters','Doctorate'))
check (branch_name in (select branch_name from branch))
```

DEFAULT

设置默认值

对约束的命名、撤消和添加

命名

```
CONSTRAINT  约束名 <约束条件>
```

示例

```
S#  CHAR(4) CONSTRAINT S_PK PRIAMRY KEY
AGE SAMLLINT CONSTRAINT AGE_VAL
CHECK(AGE >= 15 AND AGE <= 25)
```

关系上约束的撤消与添加

撤消用 alter table…drop constraint…

添加用 alter table…add constraint…

示例

```
alter table S drop constraint S_PK
alter table SC add constraint  SC_CHECK
          check(S# in select S# from S)
```

查询语句 select

语法

```
select … from … where …
```

由于大家都做了数据库实习，因此 select 就不多介绍了。同样略去的有 order by, distinct, as.

10. 字符串操作

命令格式

列名 [not] like ‘字符串’

找出满足给定匹配条件的字符串

匹配规则

‘%’ ：匹配零个或多个字符

‘_’：匹配任意单个字符

[ ]：任何在指定范围内的字符

[a-f]，[abcdef]

[^]：任何不在指定范围内的字符

[^ a-f]，[^ abcdef]

Escape

定义转义字符，以去掉特殊字符的特定含义，使其被作为普通字符看待

如 escape “\”，定义 \ 作为转义字符，则可用\%去匹配%，用_去匹配_

举例

```
‘ %a_bx ’
select * from tt
where c1 like ‘x%%xx' escape ‘x'
```

示例

列出姓名以“张”打头的教师的所有信息

```
select   *
from    PROF
where  PNAME  like  “张%”
```

列出名称中含有 3 个以上字符，且倒数第三个是 d，倒数第二个是_的课程

```
select   *
from  C
where CNAME LIKE ’%_d\ _ _’ escape ‘\’
```

11. 全文检索

创建

```
create fulltext catalog  catalog_name
create fulltext index on table_name [(column_name ]
      key index  index_name on  catalog_name
```

key index 指定 table_name 上唯一键索引的名称，最好是聚簇索引

查询

Contains(属性列|*，查找条件)

FREETEXT(属性列|* ，查找文本)

使用 FREETEXT 时，全文查询引擎内部将查找文本拆分为若干个搜索词，并赋予每个词以不同的加权，然后查找匹配

示例：全文索引的创建及使用演示

doc(doc_id, title, author, abstract, content)

```
create    unique clustered index  doc_idx on  doc(doc_id)
create    fulltext catalog  doc_fulltext_ catalog
create    fulltext index on doc (title, author, abstract, content)
          key index  doc_idx on  doc_fulltext_catalog

select *
from   documents
where  contains (*, 'database and dataspace')

select *
from   documents
where  contains (author, 'Jim Gray and not Jeff Ullman')

select *
from   documents
where  contains ((title, abstract), 'graph mining')

select *
from   documents
where  freetext (content,
                 ' Adaptive Query Processing')
```

12. 关系的连接

连接类型：inner join，left outer join，right outer join，full outer join
连接类型定义参加上面关系代数。
连接条件：nature，on<谓词>，using(属性集)
nature：出现在结果关系中的两个连接关系的元组在公共属性上取值相等，且公共属性只出现一次。
on<谓词 P>:出现在结果关系中的两个连接关系的元组在公共属性上取值满足谓词条件 P，且公共属性出现两次。
using (A1，A2 ,…，An):(A1，A2 ,…，An)是两个连接关系的公共属性的子集，元组在(A1，A2 ,…，An)上取值相等，且(A1，A2 ,…，An)只出现一次。
cross join:两个关系的笛卡儿积
union join:左边关系中失配的元组+ 右边关系中失配的元组

13. 空值测试

```
is   [not]   null
```

测试指定列的值是否为空值
注意事项

除 is [not] null 之外，空值不满足任何查找条件

如果 null 参与算术运算，则该算术表达式的值为 null

如果 null 参与比较运算，则结果可视为 false。在 SQL-92 中可看成 unknown

举例

找出成绩值为空的学生号

```
select   S#
from     SC
where   GRADE  is null
```

不可写为 where GRADE = null

isnull

isnull(check_expression, replacement_value)

如果 check_expression 值为空，则返回 replacement_value，否则返回 check_expression

举例

select S#, C#, isnull (GRADE, '缺考') from SC

14. 聚集函数

avg,min,max,sum,count

注意使用的时机(什么字句里面可以用,什么子句里面不能用)

15. 分组命令

语法

group by 列名 [having 条件表达式]

group by 将表中的元组按指定列上值相等的原则分组，然后在每一分组上使用聚集函数，得到单一值

having 则对分组进行选择，只将聚集函数作用到满足条件的分组上

举例

```
select     S#，avg(GRADE)
    from      SC
    group by  S#
    having    min(GRADE) >= 60
```

16. 嵌套子查询

in, not in, some(any), all, exists, not exists, unique

综合应用 除运算 思路 not exists…not exists…
举例 选修了全部课程的学生姓名

```
select    SNAME
from      S as S1
where     not exists
          (select      C#
          from      C as C1
          where     not exists
              (select   *
              from      SC
              where     C# = C1.C#
                  and S# = S1.S# ))
```

列出至少选修了 s1 号学生选修的所有课程的学生名

```
select    SNAME
from      S as S2
where     not exists
          (select   C#
          from      C as C1
          where     exists
              (select   *
              from      SC
              where  C# = C1.C#
                        and S# = s1)
              and    not  exists
              (select   *
              from      SC
          where     C# = C1.C#
                and   S# = S2.S#)
```

17. with 子句

语法

```
with 临时视图名 as 子查询
```

with 定义的临时视图仅对之后的那个查询有效。
举例：

```
 with max_balance(value) as
       select max(balance) from account
select account_number
from account, max_balance
where account.balance=max_balance.value
```

集合操作

union，union all, intersect, intersect all, except, exceptall

举例

```
R except S
     select  A
     from    (select   1 as flag, A
          from   R
          union all
          select 0, A
          from   S)  RS
     group by A
     having  count(*) = 1
       and   max(flag) = 1

R except all S
select A
from      (select   A, max(Rcnt) - max(Scnt) as Cnt
          from   (select   1 as flag, A, count(*) Rcnt, 0
          from      R
          group by  A
          union all
          select     0, A, 0, count(*) Scnt
          from       S
          group by   A)  RS_1
       group by  A)  RS_2
       join   SeqNums on s_number <= Cnt
```

18. 常用 SQL 函数

getdate( )：返回系统的当前日期

datepart(datepart, date)：返回 date 中的 datepart 部分

```
datepart( dd, '11/23/2008') = 23
datepart( yy, getdate( )) = 2008
```

day(date)，month(date)，year(date)：分别返回 date 中的“日”“月”“年”日期部分

```
day('11/23/2008') = 23
month ('11/23/2008') = 11
year (getdate( )) = 2008
```

datediff(datepart, startdate, endate)：按照 datepart 返回两个日期 startdate 和 endate 之差

datediff (mm, ‘8/1/2005’ , getdate ())返回 2005 年 8 月 1 日和当前日期之间相隔的月数

dateadd(datepart, number, date)：将 number 加到 date 的 datepart 部分上

```
dateadd(dd, 20, '11/23/2008') = '12/13/2008'
```

len(string_expression)：返回 string_expression 中的字符个数。如

len('China') = 5。
lower(string_expression)：返回小写字符的 string_expression。如 lower('China') = 'china'。
upper(string_expression)：返回大写字符的 string_expression。如 upper('China') = 'CHINA'。
replicate (string _expression ,integer_expression)：按指定的次数重复字符串表达式。如 replicate ('ha', 3) = 'hahaha'。
reverse(string_expression)：返回字符表达式的逆向表达式。如 reverse('databases') = 'sesabatad'。
substring (string_expression, start, length)：从 string_expression 中由 start 位置开始返回长度为 length 的部分字符串。如 substring ('abcdef ', 2, 3) = 'bcd'。
replace (expression1, expression2, expression3)：用第三个表达式替换第一个表达式中所出现的所有第二个字符串表达式。如 replace('abcdefghicde', 'cde', 'xxx') = 'abxxxfghixxx'，replace('database', 'a', '') = 'dtbse'。

举例 计算'a'在字符串'databases'中出现的次数
select len('databases') - len(replace('databases', 'a', ''))

19. 插入操作

命令

insert into 表名 [（列名[，列名]…] values （值 [，值]…）
插入一条指定好值的元组

insert into 表名 [（列名[，列名]…]（子查询）
插入子查询结果中的若干条元组

20. 删除操作

delete from 表名 [where 条件表达式]
从表中删除符合条件的元组，如果没有 where 语句，则删除所有元组
truncate table
删除表中的所有行，而不记录单个行删除操作
truncate table 在功能上与不带 where 子句的 delete 语句相同。但 truncate table 比 delete 速度快，且使用的系统和事务日志资源少
identity 计数器重置为种子值

21. 更新操作

```
update   表名
set    列名 = 表达式 | 子查询
        列名 = [，表达式 | 子查询]…
[where  条件表达式]
```

指定对哪些列进行更新，以及更新后的值是什么

定义视图

```
create   view   view_name[(列名[，列名] …)]
          as   （查询表达式）
          [with check option]
```

视图的属性名缺省为子查询结果中的属性名，也可以显式指明

with check option 指明当对视图进行 insert，update 时，要检查是否满足视图定义中的条件

撤销视图

```
drop   view   view_name
```

视图更新约束

select 子句中的目标列不能包含聚集函数

select 子句中不能使用 unique 或 distinct 关键字

不能包括 group by 子句

不能包括经算术表达式计算出来的列

对于行列子集视图可以更新（视图是从单个基本表使用选择、投影操作导出的，并且包含了基本表的主码）(列举一种不可更新的行列子集视图)

22. 授权命令

```
grant 表级权限 on {表名 | 视图名} to
      {用户 [，用户]… | public}
     [with grant option]
```

表级权限包括：select，update，insert，delete，index，alter，drop，resource 等以及它们的总和 all，其中对 select ，update 可指定列名

with grant option 表示获得权限的用户可以把权限再授予其它用户

23. 回收权限

```
revoke 表级权限 on {表名 | 视图名} from
{用户 [，用户]… | public}
```

收回权限时，若该用户已将权限授予其它用户，则也一并收回。授权路径的起点一定是 DBA

示例

```
grant select , insert on S to Liming
      with grant option
revoke insert on S from Liming
```

二、 掌握游标的使用

declare

定义一个游标，使之对应一个 select 语句

declare 游标名[insensitive] [scroll] cursor for
select 语句[for update [of 列表名]]

insensitive：创建游标使用的数据临时复本。对游标的所有请求都从 tempdb 中的临时表中得到应答；对游标进行提取操作时返回的数据不反映对基表所做的修改，并且该游标不允许修改。如果省略 insensitive，任何用户对基表提交的删除和更新都反映在后面的提取中

for update：表示该游标可用于对当前行的修改与删除

open

打开一个游标，执行游标对应的查询，结果集合为该游标的活动集

open 游标名

fetch

在活动集中将游标移到特定的行，并取出该行数据放到相应的宿主变量中

```
fetch [next |prior|first | last | current | relative n | absolute m]
      游标名 into [宿主变量表]
```

close

关闭游标，释放活动集及其所占资源。需要再使用该游标时，执行 open 语句

close 游标名

free

删除游标，以后不能再对该游标执行 open 语句

free 游标名

举例

```
DECLARE my_curs CURSOR SCROLL FOR
   SELECT au_id, au_lname
   FROM authors
OPEN my_curs
FETCH ABSOLUTE 6 FROM my_curs

declare my_curs cursor for
   select *
   from SC
   where C# = ‘c1’
```

```
for update

open my_curs

update SC
set  GRADE = GRADE *1.05
where current of my_curs
```

三、 编写触发器

1.创建触发器

```
create trigger trigger_name
       { before | after }
       { insert | delete | update [of column-name] }
       on table_name
        [referencing { old table as id | new table as id | old row
       as id | new row as id}]
       [for each statement | for each row ]
       { when(search_condition)
          begin atomic
             triggered_SQL_statement
          end
       }
```

举例

```
EMP(ENO, ENAME, SAL, JOB)
```

职工工资增幅不得超过 10%

```
create trigger RAISE_LIMIT
after update of SAL on EMP
   referencing new row as nrow old row as orow
   for each row//行级触发器
   when (nrow.SAL > 1.1 * orow.SAL)
      begin
         signal SQLSTATE ‘7500’ (“Salary increase 10%)
      end
```

当帐户透支时，将帐户余额设为 0，并建一笔贷款，其金额为透支额

```
create trigger overdraft-trigger after update on account
   referencing new row as nrow  for each row
   when nrow.balance < 0
   begin atomic
```

```
        insert into borrower
            (select customer-name, account-number
            from depositor
            where nrow.account-number = depositor.account-number);
                insert into loan values(nrow.account-number,
                    nrow.branch-name, - nrow.balance);
            update account set balance = 0
        where account.account-number = nrow.account-number
    end
```

```
EMP(ENO, ENAME, SAL, JOB)
职工平均工资不得低于 800
create trigger RAISE_LIMIT
after update of SAL on EMP
referencing new table as n_tb old table as o_tb
for each statement //语句级触发器
when (800 > (select avg(SAL) from EMP)
    begin
        delete from EMP
        where ENO in ( select ENO from n_tb )
        insert into EMP ( select * from o_tb )
    end
```

2.递归触发器举例

设计触发器，保证部门预算始终等于该部门预算与其所有子部门预算之和

```
create table dept

(dept_name     varchar(30)    not null,
parent_name    varchar(30)    null,
budget    money         not null)
insert into  dept values ('d1', 'd2', $10)
insert into  dept values ('d2', 'd3', $100)
insert into  dept values ('d3', null, $500)

create trigger  budget on  dept after update
as
if ( @@rowcount = 0) return
if (@@rowcount > 1)
begin
    print 'Only one row can be updated at a time'
    rollback tran
    return
```

```
end
if (select parent_name from inserted) is null return
update    dept
set
   budget = budget + (select budget from inserted) - (select budget
from deleted)
where  dept_name = (select parent_name from inserted)
exec  sp_dboption university, 'recursive triggers', true
```

3.替代触发器举例

```
create  view  COMPUTER_PROF   as
   (select  P# , PNAME , SAL
   from    PROF，DEPT
   where   PROF.P# = DEPT.P#
    and     DEPT.DNAME = “计算机系”)

insert into  COMPUTER_PROF (‘p01’, ‘tom’, 800)
insert into  PROF (‘p01’, ‘tom’, 800, null, null)

create trigger INSERT_VIEW on COMPUTER_PROF
instead of insert
as
   declare @d_no  char(10)
   set @d_no = (select  d#  from  DEPT where DNAME = “计算机系”)
   insert into PROF(inserted.p#, inserted.pname,
             inserted.sal, null , @d_no)

create view join_view as
   select  Table1.a as a1, Table2.a as a2
   from     Table1 join Table2 on Table1.a = Table2.a

create trigger DELETE_JOIN
on join_view
instead of delete
as
   delete  Table1
   where  a in (select  a1 from deleted)
   delete  Table2
   where  a in (select  a2 from deleted)
```

第六章　关系数据库设计

一、 理解概念

1. 函数依赖、部分函数依赖、完全函数依赖、传递函数依赖、多值依赖

函数依赖

设 $R(U)$ 是属性集 U 上的关系模式， $X,Y \subseteq U$ ，r 是 $R(U)$ 上的任意一个关系，如果成立

$$\forall t,s \in r, 若 t[X]=s[X], 则 t[Y]=s[Y]$$

那么称“ X 函数决定 Y ”，或“ Y 函数依赖于 X ”，记作 $\mathrm{X} \to \mathrm{Y}$ 。称 X 为决定因素

部分函数依赖，完全函数依赖

在 $R(U)$ 中，如果 $\mathrm{X} \to \mathrm{Y}$ ，且对于 X 的任意真子集 X' ，都有 $X' \to Y$ 不成立，则称 Y 对 X 完全函数依赖，记作 $X \xrightarrow{f} Y$ 。否则称 Y 对 X 部分函数依赖，记作 $X \xrightarrow{p} Y$

传递函数依赖

在 $R(U)$ 中，如果 $\mathrm{X} \to \mathrm{Y}$ ， $\mathrm{Y} \to \mathrm{Z}$ ， $\mathrm{Y} \to \mathrm{X}$不成立，且 $Z \not\subset Y$ ，则称 Z 对 X 传递函数依赖。部分函数依赖是传递函数依赖的特例。

多值依赖

在 $R(U)$ 中， $X,Y,Z \subset U, Z=U-X-Y$ ，对于 $R(U)$ 的任一关系 r ，若存在元组 t_1,t_2 ，使得 $t_1[X]=t_2[X]$ ，那么就必然存在元组 t_3,t_4 ，使得

$$\begin{cases} t_3=(t_1[X],t_1[Y],t_2[Z]) \\ t_4=(t_2[X],t_2[Y],t_1[Z]) \end{cases}$$

则称 Y 多值依赖于 X ，记作 $X \to\!\!\to Y$

由多值依赖的定义，我们可以得出以下规则

$$若 \alpha \to \beta ，则 \alpha \to\!\!\to \beta$$

换句话说，每一个函数依赖都是多值依赖。

2.主码、主属性、全码

主码

代表被数据库设计者选中的、用来在同一关系中区分不同元组的候选码。

主属性

主属性：包含在每一个候选码中的属性，称作主属性。

全码

全码：关系模式的码由整个属性组构成，如 SPJ

3.1NF、2NF、3NF、BCNF

$1NF$

如果某个域的元素被认为是不可分的单元，那么这个域就是原子的。

如果一个关系模式 R 的所有属性与都是原子的，那么这个关系模式 R 属于第一范式。

$2NF$

若 $R \in 1NF$，且每个非主属性完全依赖于码，则称 $R \in 2NF$ 。

$2NF$ 消除了非主属性对码的部分依赖。

课本上的定义见课后习题

$3NF$

关系模式 $R<U,F>$ 中，若不存在这样的码 X ，属性组 Y 及非主属性 $Z(Z \not\subset Y)$ 使得下式成立： $X \to Y, Y \to Z, Y \not\to X$ 。则称 $R \in 3NF$ 。

$3NF$ 消除非主属性对码的传递依赖。

$BCNF(Boyce-Codd\ \ NormalForm)$

PPT 定义：

关系模式 $R<U,F>$ 中，对于属性组 X,Y ，当 $X \to Y$ 且 $Y \to X$ 不成立时，X 必含有码。

则 $R<U,F> \in BCNF$ 。

课本定义：

具有函数依赖集 F 的关系模式 R 属于 $BCNF$ 的条件是，对于 F^+ 中型如 $\alpha \to \beta$ 的函数依赖（ $\alpha \subseteq R, \beta \subseteq R$ ），下面至少有一个成立：

- $\alpha \to \beta$ 是平凡的函数依赖（即 $\beta \subseteq \alpha$ ）
- α 是模式 R 的一个超码

4.无损连接分解、保持函数依赖分解

无损连接分解

设R为一关系模式，F为R上函数依赖集。令R_1和R_2为R的分解。令$r(R)$是模式R上的一个关系。我们称该分解是无损分解（无损连接分解），如果对于所有的合法数据库实例（即满足指定的函数依赖和其他约束的数据库实例，都有）

$$\prod_{R_1}(r) \triangleright\triangleleft \prod_{R_2}(r) = r$$

换句话说，如果我们把r投影至R_1, R_2上，然后计算投影结果的自然连接，我们仍然得到一模一样的r。

保持函数依赖分解

Z是U的子集，函数依赖集合F在Z上的投影定义为：

$$\prod_Z(F) = \{X \to Y \mid X \to Y \in F^+ \wedge XY \subseteq Z\}$$

设$\rho = \{R_1, R_2, ..., R_n\}$是关系模式$R<U,F>$的一个分解，如果$F^+ = (\bigcup_{i=1}^{n} \prod_{R_i}(F))^+$，则称$\rho$是保持函数依赖的分解。

二、 判断将一个关系模式分解为两个模式时是无损的

定理：关系模式$R(U)$的分解$\rho = \{R_1, R_2\}$，则ρ是一个无损连接分解的充要条件是

$$R_1 \cap R_2 \to R_1 - R_2 \text{ 或 } R_1 \cap R_2 \to R_2 - R_1 \text{ 成立}$$

三、 计算关系模式的候选码及关系模式的范式级别的判定

1. 候选码的计算

定义

左部属性，只出现在 F 左边的属性
右部属性，只出现在 F 右边的属性
双部属性，出现在 F 两边的属性
外部属性，不出现在 F 中的属性

定理

左部属性一定出现在任何候选码中

右部属性一定不出现在任何候选码中
外部属性一定出现在任何候选码中

对于关系模式$R<U,F>$，设其左部属性集A，右部属性集B，双部属性集C，外部属性D，设集合M遍历C的所有子集，计算$(A\cup D\cup M)_F^+$，如果它等于U，则证明$A\cup D\cup M$是一个候选码，接下来不用计算$A\cup D\cup M$的任何超集（候选码的极小性）。

2. 范式级别的判定

$1NF$ 判断所有属性是否是原子属性
$2NF$ 判断是否有非主属性对码的部分依赖

$3NF$ 判断是否有传递依赖$(NP$难$)$

$BCNF$ 判断是否有基于函数依赖的冗余（主要按定义）方法如下

1. 检查平凡的函数依赖$\alpha\to\beta$是否违反$BCNF$，计算α^+（α的属性闭包），并验证它是否包含了R的所有属性，即验证它是否是R的超码。
2. （模式分解之前）检查关系模式R是否属于$BCNF$，仅需检查给定集合F中的函数是否违反$BCNF$就足够了，不用检查F^+中所有的函数依赖。
3. （模式分解之后）检验R_i是否属于$BCNF$，对于R_i上的属性的所有子集α，确保α^+（F下α的属性闭包）要么不包含$R_i-\alpha$的任意属性，要么包含R_i的所有属性。

四、 掌握关系模式的三个分解算法

0. 预备知识

1) 函数依赖集的等价判断

设函数依赖集F,G，若$F^+=G^+$，则称F与G等价。

性质：$F^+=G^+\Leftrightarrow F\subseteq G^+\wedge G\subseteq F^+$

2) 最小覆盖的计算

第一步：单属性化

对于 F 中任一函数依赖 $X \to A$， A 必是单属性。

逐个检查 F 中各个函数依赖 $X \to Y$，若 $Y = A_1A_2...A_k, k \geq 2$，则用 $X \to A_i$ 代替 Y。

第二步：无冗余化

F 中不存在这样的函数依赖 $X \to A$，使得 F 与 $F-\{X \to A\}$ 等价。

逐个检查 F 中的各个函数依赖 $X \to A$，令 $G = F-\{X \to A\}$，若 $A \in X_G^+$，则从 F 中去掉该函数依赖。

第三步：既约化

F 中不存在这样的函数依赖 $X \to A$，在 X 中有真子集 Z，使得 F 与 $(F-\{X \to A\}) \cup \{Z \to A\}$ 等价。

逐个检查 F 中的各个函数依赖 $X \to A$，设 $X \to B_1B_2...B_m$，逐个考察 B_i，若 $A \in (X-B_i)_F^+$，则以 $X-B_i$ 取代 X

3) 函数在给定属性集合上的投影的计算

目标：用给定属性集合上函数的最小覆盖来描述投影

做法：设 $R<U,F>$，求 R 在 $U' \subseteq U$ 上的投影 F'

第一步：首先令 F' 为空，并求出 U' 的所有非空子集 $S_1, S_2, ..., S_n$

第二步：设 $S_i = A_1A_2...A_t$，计算 $(S_i)_F^+$，$(1 \leq i \leq n)$，如果有 $S_i \subset (S_i)_F^+$，那么令 $T_i = (S_i)_F^+ - S_i$，$T_i = B_1B_2...B_m$，将函数依赖 $A_1A_2...A_t \to B_1B_2...B_m$ 加入 F'。

第三步：求 U' 上 F' 的最小覆盖，即是所求的投影。

1. 保持函数依赖的 3NF 分解

设给定关系模式 $R<U,F>$。

step 1. 求 F 的最小覆盖 $F_{\min}$

step 2. 找出不在 $F_{\min}$ 中出现的属性，将它们构成一个关系模式，并从 U 中去掉

他们（剩余属性依然记为U）。

step 3. 若有 $X \to A \in F_{\min}$ 且 $XA = U$，则 $\rho = \{R\}$，算法终止。

step 4. 否则，对 $F_{\min}$ 按具有相同左部的原则进行分组（设为 k 组），每一组函数依赖所涉及的属性全体为 U_i，令 F_i 为 $F_{\min}$ 在 U_i 上的投影，则

$$\rho = \{R_1 < U_1, F_1 >, R_2 < U_2, F_2 >, ... R_k < U_k, F_k >\}$$

是 $R < U, F >$ 的一个保持函数依赖的分解，并且每一个 $R_i < U_i, F_i > \in 3NF$。

2. 保持无损连接的 BCNF 分解

设给定关系模式 $R < U, F >$。

step 1. 令 $\rho = \{R < U, F >\}$。

step 2. 检查 ρ 中各关系模是是否属于 $BCNF$，若是，则算法终止。

step 3. 设 $R_i < U_i, F_i > \in \rho, R_i < U_i, F_i > \notin BCNF$，则存在函数依赖 $X \to A \in F_i^+$，且 X 不是 R_i 的码，则 XA 是 R_i 的真子集，将 R_i 分解为 $\sigma = \{S_1, S_2\}$，其中 $U_{S_1} = XA, U_{S_2} = U_i - A$，以 σ 代替 R_i，回到 step 2.

3. 同时保持函数依赖和无损连接的 3NF 分解算法

设 $\rho = \{R_1 < U_1, F_1 >, R_2 < U_2, F_2 >, ... R_k < U_k, F_k >\}$ 是 $R < U, F >$ 的一个保持函数依赖的 $3NF$ 分解，设 X 为 $R < U, F >$ 的码，若有某个 $U_i, X \subseteq U_i$ 则 ρ 为所求；否则，令 $T = \rho \cup \{R^* < X, F_X >\}$，$T$ 即为所求。

第七章　事务

一、 理解概念

1.事务及事务的 ACID 特性

事务定义

事务是由一系列操作序列构成的程序执行单元，这些操作要么都做，要么都不做，是一个不可分割的工作单位。

SQL 中事务的定义

事务以 Begin transaction 开始，以 Commit transaction 或 Rollback transaction 结束。

Commit transaction 表示提交，事务正常结束。

Rollback transaction 表示事务非正常结束，撤消事务已做的操作，回滚到事务开始时状态。

事务特性(ACID)

原子性(Atomicity)

事务中包含的所有操作要么全做，要么全不做

原子性由恢复机制实现

一致性(Consistency)

事务的隔离执行必须保证数据库的一致性

事务开始前，数据库处于一致性的状态；事务结束后，数据库必须仍处于一致性状态

数据库的一致性状态由用户来负责

如银行转帐，转帐前后两个帐户金额之和应保持不变

隔离性(Isolation)

系统必须保证事务不受其它并发执行事务的影响

对任何一对事务T_1,T_2，在T_1看来，T_2要么在T_1开始之前已经结束，要么在T_1完成之后再开始执行

隔离性通过并发控制机制实现

持久性(Durability)

一个事务一旦提交之后，它对数据库的影响必须是永久的

系统发生故障不能改变事务的持久性

持久性通过恢复机制实现

2.可恢复调度、无级联调度

可恢复调度

事务的恢复：一个事务失败了，应该能够撤消该事务对数据库的影响。如果有其它事务读取了失败事务写入的数据，则该事务也应该撤消

可恢复调度应满足：对于每对事务T_i,T_j，如果T_j读取了T_i所写入的数据项，则T_i先于T_j提交。

级联回滚

因一个事务故障导致一系列事务回滚的现象称为级联回滚。

无级联调度

对于每对事务T_i,T_j，如果T_j读取了T_i所写入的数据项，则T_i必须在T_j这一读取前提交。

无级联调度总是可恢复的。

3.四种数据不一致性

丢失修改

两个事务T_1和T_2读入同一数据并修改，T_1提交的结果破坏了T_2提交的结果，导致T_2的修改丢失。

读脏数据

事务T_1修改某一数据，并将其写回磁盘，事务T_2读取同一数据后，T_1由于某种原因被撤消，这时T_1已修改过的数据恢复原值，T_2读到的数据与数据库中数据不一致，则T_2读到的数据就是脏数据。

不能重复读

事务T_2读取某一数据后，事务T_1对其做了修改，当T_2再次读取该数据时，得到与前次不同的值。

发生幻象

事务T_2按一定条件读取了某些数据后，事务T_1插入了一些满足这些条件的数据，当T_2再次按相同条件读取数据时，发现多了一些记录。

4.快照隔离

SI：快照隔离，任何读取操作得到事务开始那一刻最近已经提交过的数据版本，属于事务级快照隔离

RCSI：已提交读快照隔离，任何读取操作得到语句开始那一刻最近已经提交过的数据版本，属于语句级快照隔离

二、 理解 SQL 中的四个事务隔离性级别定义

1.serializable

一个调度的执行必须等价于一个串行调度的结果。
不会发生任何不一致。

2.repeatable read

只允许读取已提交的记录，并要求一个事务对同一记录的两次读取之间，其它事务不能对该记录进行更新。
会发生的不一致现象：幻象。

3.read committed

只允许读取已提交的记录，但不要求可重复读
会发生的不一致现象：不能重复读，幻象。

4.read uncommitted

允许读取未提交的记录
会发生的不一致现象：读脏数据，不能重复读，幻象。

三、 冲突可串行化及其判定

1.冲突

当I_i, I_j是不同事务对相同的数据项的操作，并且其中至少有一个是 *write* 指令时，我们说I_i, I_j是冲突的。

2.冲突等价

如果调度S可以经过一系列非冲突指令交换转换成S'，我们称S与S'是冲突等价的。

3.冲突可串行化

如果一个调度S与一个串行调度冲突等价，称调度S是冲突可串行化的。

4.冲突可串行化的判定。

优先图$G=(V,E)$，顶点集由所有参与调度的事务组成，边集由满足下列三个条件之一的边$T_i \to T_j$组成：

1．在T_j执行$read(Q)$之前，T_i执行$write(Q)$。

2．在T_j执行$write(Q)$之前，T_i执行$read(Q)$。

3．在T_j执行$write(Q)$之前，T_i执行$write(Q)$。

如果优先图中存在边$T_i \to T_j$，则在任何等价于S的串行调度S'中，T_i必出现在T_j之前。

如果调度S的优先图中有环，则调度S是非冲突可串行化的。如果图中无环，则调度S是冲突可串行化的，并且优先图的每一个拓扑排序给出了一个等价的串行调度顺序。

四、 视图可串行化及其判定

把握**从读一致性**来判定，由于这是NP难问题，只有指数级时间的充分判定。

1.视图等价

考虑关于某个事务集的两个调度S，S'，若调度S，S'满足以下条件，则称它们是视图等价的：

① 于每个数据项Q，若事务T_i在调度S中读取了Q的初始值，那么T_i在调度S'中也必须读取Q的初始值

②对于每个数据项Q，若事务T_i在调度S中执行了$read(Q)$，并且读取的值是由T_j产生的，那么T_i在调度S'中读取的Q值也必须是由T_j产生的

③对于每个数据项Q，若在调度S中有事务执行了最后的$write(Q)$，则在调度S'中该事务也必须执行最后的$write(Q)$

注：条件①、②保证两个调度中的每个事务都读取相同的值，从而进行相同的计算
条件③保证两个调度得到最终相同的系统状

2.视图可串行化

如果某个调度视图等价于一个串行调度，则称该调度是视图可串行化的
冲突可串行化调度一定是视图可串行化的，存在视图可串行化但非冲突可串行化的调度，盲目写操作存在于任何不适冲突可串行化的视图可串行化调度中。

3.带标记的优先图的构造

设调度S包含了事务$\{T_1,T_2,...,T_n\}$，设T_b,T_f是两个虚事务，其中T_b为S中所有$write(Q)$操作，T_f为S中所有$read(Q)$操作。在调度S的开头插入T_b，在调度S的末尾插入T_f，得到一个新的调度S'

1. 如果T_j读取T_i写入的数据项的值，则加入边$T_i \xrightarrow{0} T_j$
2. 删除所有关联无用事务的边。如果在优先图中不存在从T_i到T_f的通路，则T_i是无用事务。
3. 对于每个数据项Q，如果T_j读取T_i写入的Q值，T_k执行$write(Q)$操作且$T_k \neq T_b$，则
   i. 如果$T_i = T_b$且$T_j = T_f$，则在带标记的优先图中插入边$T_j \xrightarrow{0} T_k$
   ii. 如果$T_i \neq T_b$且$T_j = T_f$，则在带标记的优先图中插入边$T_k \xrightarrow{0} T_i$
   iii. 如果$T_i \neq T_b$且$T_j \neq T_f$，则在带标记的优先图中插入边$T_k \xrightarrow{p} T_i$与$T_j \xrightarrow{p} T_k$。其中p是一个唯一的，在前面标记中未曾用过的大于0的整数。

判定准则：只要有一个优先图无环，则调度是视图可串行化的

第十二章　事务处理

一、 各种封锁模式

X 锁、S 锁、U 锁、IS 锁、IX 锁、SIX 锁

1.排它锁（*X* 锁， *eXclusive lock*）

事务T对数据对象R加上X锁，则其它事务对R的任何封锁请求都不能成功，直至T释放R上的X锁；又称写锁。

申请对R的排它锁：$lock-X(R)$

2.共享锁（*S* 锁， *Share lock*）

事务T对数据对象R加上S锁，则其它事务对R的X锁请求不能成功，而对R的S锁请求可以成功；又称读锁。

申请对R的共享锁：　$lock-S(R)$

3.更新锁（*U* 锁， *Update lock*）

在锁转换中,一个更新事务读取数据项Q，获取Q上的S锁，然后修改行，此操作要求锁转换为X锁。如果两个事务都获得了Q上的S锁，然后试图同时更新数据，则一个事务尝试将S锁转换为X锁，由于另一个事务持有Q上的S锁，所以发生锁等待。同样第二个事务也试图将其S锁升级为X锁以便进行更新，需要等待第一个事务释放Q上的锁，从而导致两个事务都无法进行，因此发生死锁。

当一个事务查询数据以便要进行修改时,可以对数据项施加更新锁,如果事务修改资源,则更新锁会转换为排它锁。一次只有一个事务可以获得资源上的更新锁，它允许其它事务对资源的共享式访问但阻止排他式的访问。

4.意向（预约）封锁

在分层封锁中，封锁了上层节点就意味着封锁了所有内层节点。如果有事务 T1 对某元组加了 S 锁，而事务 T2 对该元组所在的关系加了 X 锁，因而隐含地 X 封锁了该元组，从而

造成矛盾。

引入**意向锁** *I*（intend）：当为某节点加上 *I* 锁，表明其某些内层节点已发生事实上的封锁，防止其它事务再去显式封锁该节点

I 锁的实施是从封锁层次的根开始，依次占据路径上的所有节点，直至要真正进行显式封锁的节点的父节点为止。

5. *IS* 锁

如果对一个数据对象加 *IS* 锁，表示它的后裔节点拟（意向）加 *S* 锁。
例如，要对元组加 *S* 锁，则首先要对关系和数据库加 *IS* 锁。

6. *IX* 锁

如果对一个数据对象加 *IX* 锁，表示它的后裔节点拟（意向）加 *X* 锁。
例如，要对元组加 *X* 锁，则首先要对关系和数据库加 *IX* 锁。

7. *SIX* 锁

如果对一个数据对象加 *SIX* 锁，表示对它加 *S* 锁，再加 *IX* 锁
例如对某个表加 *SIX* 锁，则表示该事务要读整个表（对该表加 *S* 锁），同时会更新个别元组（对该表加 *IX* 锁）

8. 相容矩阵

相容矩阵 comp(A,B)						
请求锁模式 A	现有锁模式 B					
	IS	S	U	IX	SIX	X
IS	是	是	是	是	是	否
S	是	是	是	否	否	否
U	是	是	否	否	否	否
IX	是	否	否	是	否	否
SIX	是	否	否	否	否	否
X	否	否	否	否	否	否

二、 两段锁协议及其作用

1. 两阶段封锁协议(*two – phase locking protocol*)

该协议要求每个事务分两个阶段提出加锁和解锁申请。
增长阶段：事务可以获得锁，但不能释放锁。
缩减阶段：事务可以释放锁，但不能获得锁。
两阶段封锁协议可以保证冲突可串行化。对于任何事务，在调度中该事务获得其最后加锁的位置（增长阶段结束点）成为事务的**封锁点**。这样，多个事务可以根据他们的封锁点进行排序，实际上，这个顺序就是事务的一个可串行化顺序。
两阶段封锁协议并不保证不发生死锁。。
两阶段封锁协议并不保证无级联。

2. 严格两阶段封锁协议(*strict two – phase locking protocol*)

严格两阶段封锁协议不仅要求封锁是两阶段，还要求事务所持有的所有排它锁必须在事务提交之后方可释放。这个要求保证未提交事务所写的任何数据在该事务提交之前均以排他方式加锁，防止了其他事务读取这些数据。

3. 强两阶段封锁协议(*rigorous two – phase locking protocol*)

要求事务提交之前不释放任何锁。在此条件下，事务可以按照其提交的顺序串行化。

大部分数据库系统要么采用严格两阶段封锁，要么采用强两阶段封锁。

4. 锁转换(*lock conversion*)

一种将共享锁提升为排它锁及将排它锁降级为共享锁的机制。

upgrade：从共享锁升级为排它锁，只能在增长阶段使用。

downgrade：从排它锁降级为共享锁，只能在缩减阶段使用。

对于一个事务集，可能存在某些不能通过两阶段封锁协议得到的冲突可串行化调度。然而，如果要通过非两阶段封锁协议得到冲突可串行化调度，我们或者需要事务的附加信息，或者要求数据库的数据项集合有一定的结构或者顺序。没有这样的信息，两阶段封锁对于冲突可串行化就是必不可少的——如果T_i是一个非两阶段事务，则总是可以找

到另一个两阶段事务T_j，使得T_i与T_j的某个调度不是冲突可串行化的。

严格两阶段封锁与强两阶段封锁（含锁转换）在商用数据库系统中广泛使用。
这里介绍一个简单却广泛使用的机制，它给予来自事务的读、写请求，自动地为事务产生加锁、解锁指令。

当事务T_i发出$read(Q)$操作时，系统就发出一个$lock-S(Q)$指令，该$read(Q)$指令紧跟其后。

当事务T_i发出$write(Q)$操作时，系统检查T_i是否已对Q持有共享锁。若是，则系统发出$upgrade(Q)$指令，后接$write(Q)$指令；否则系统发出$lock-X(Q)$指令，后接$write(Q)$指令。

当一个事务提交或者中止后，该事务所持有的所有锁都被释放。

三、 理解基于时间戳的并发控制协议

1.时间戳

对于系统中每个事务T_i，把一个唯一的固定时间戳和它联系起来，次时间戳记为$TS(T_i)$。该时间戳是在事务T_i开始执行前由数据库系统赋予的。若事务T_i已被赋予时间戳$TS(T_i)$，并且有一新事务T_j进入系统，则$TS(T_i)<TS(T_j)$。实现这种机制可以用以下两种简单的办法：

1. 使用系统时钟指作为时间戳；也就是说，事务的时间戳等于该事务进入系统时的时钟值。
2. 使用逻辑计数器(logical counter)，每赋予一个时间戳，计数器增加一次；即事务的时间戳等于事务进入系统时的计数器值。

事务的时间戳决定了串行化顺序。因此，若$TS(T_i)<TS(T_j)$，则系统必须保证所产生的调度等价于事务T_i出现在事务T_j之前的某个调度。

要实现这个机制，每个数据项Q需要与两个时间戳值相关联：

$W-timestamp(Q)$表示成功执行$write(Q)$的任意事务的最大时间戳。

$R-timestamp(Q)$表示成功执行$read(Q)$的任意事务的最大时间戳。

每当新的 $read(Q)$ 或者 $write(Q)$ 指令执行时，这些时间戳就更新。

2.时间戳排序协议

时间戳排序协议保证任何有冲突的 *read* 或者 *write* 操作按时间戳顺序执行。

1．假设事务 T_i 发出 $read(Q)$。

a) 如果 $TS(T_i) < W-timestamp(Q)$，则 T_i 需读入的 Q 值已经被覆盖。因此，*read* 操作被拒绝，T_i 回滚。

b) 如果 $TS(T_i) \geq W-timestamp(Q)$，则执行 *read* 操作，$R-timestamp(Q)$ 设置为 $R-timestamp(Q)$ 与 $TS(T_i)$ 的最大值。

2．假设事务 T_i 发出 $write(Q)$。

a) 如果 $TS(T_i) < R-timestamp(Q)$，则 T_i 产生的 Q 值是先前所需要的值，且系统已假定该值不会产生。因此 *write* 操作被拒绝，T_i 回滚。

b) $TS(T_i) < W-timestamp(Q)$，则 T_i 试图写入的 Q 值已经过时，因此系统拒绝 *write* 操作，T_i 回滚。

c) 否则，系统执行 *write* 操作，将 $W-timestamp(Q)$ 设置为 $TS(T_i)$。

如果事务 T_i 由于发出 *read* 和 *write* 操作而被并发控制机制滚回，则系统赋予它新的时间戳并重新启动。

四、 死锁及解决措施

1.死锁

如果存在一个事务集，盖集合中的每个事务在等待该集合中的另一个事务，那么我们说系统处于死锁状态。
处理死锁问题的办法：死锁预防、死锁检测与死锁恢复。

2.死锁发生的条件

①互斥条件：事务请求对资源的独占控制
②有等待条件：事务已持有一定资源，又去申请并等待其它资源
③抢占条件：直到资源被持有它的事务释放之前，不可能将该资源强制从持有它的事务夺去
④循环等待条件：存在事务相互等待的等待圈
定理：在条件① ② ③成立的前提下，条件④是死锁存在的充分必要条件

3.预防死锁

1．预先占据所需的全部资源，要么一次全部封锁要么全不封锁
缺点：难于预知需要封锁哪些数据并且数据使用率低
2．所有资源预先排序，事务按规定顺序封锁数据
3．使用抢占与事务回滚
a) wait-die：如果 T1 等待 T2，仅当 T1 的时间戳小于 T2 时，允许 T1 等待，否则回滚 T1。
b) wound-wait：如果 T1 等待 T2，仅当 T1 的时间戳大于 T2 时，允许 T1 等待，否则回滚 T2

死锁检测和恢复
超时法：如果等待封锁的时间超过限时，则撤消该事务
等待图法

4.活锁

可能存在某个事务永远处于等待状态，得不到执行，称之为活锁（饿死）
T2 持有对 R 的 S 锁，T1 申请对 R 的 X 锁，则 T1 必须等待 T2 释放 S 锁；若在 T2 完成之前有 T3 申请对 R 的 S 锁，则可以获得授权封锁，于是 T1 必须等待 T2、T3 释放 S 锁
避免活锁的策略是遵从“先来先服务”的原则，按请求封锁的顺序对各事务排队；当事务 Ti 对数据项 R 加 M 型锁时，获得封锁的条件是
不存在在 R 上持有与 M 型锁冲突的锁的其他事务
不存在等待对 R 加锁且先于 Ti 申请加锁的事务

五、 备份概念及其类型

1.转储

将数据库复制到磁带或另一个磁盘上保存起来的过程。
这些备用的数据称为后备（后援）副本
静态转储：转储期间不允许对数据库进行任何存取、修改活动

动态转储：转储期间允许对数据库进行存取或修改
海量转储：每次转储全部数据库
增量转储：每次只转储上次转储后更新过的数据

2.数据库备份

数据库备份创建备份完成时数据库内存在的数据的副本，通常按常规时间间隔调度
还原数据库备份将重新创建数据库和备份完成时数据库中存在的所有相关文件。但是，自创建备份后所做的任何数据库修改都将丢失

3.差异数据库备份(DCM)

差异数据库备份只记录自上次数据库备份后发生更改的数据，比数据库备份小而且备份速度快。使用差异数据库备份将数据库还原到差异数据库备份完成时的那一点。

4.事务日志备份

事务日志是自上次备份事务日志后对数据库执行的所有事务的一系列记录，它可以将数据库恢复到特定的即时点或恢复到故障点

5.文件备份

可以备份和还原数据库中的个别文件。这样可以只还原已损坏的文件，而不用还原数据库的其余部分，从而加快恢复速度

六、 日志内容、WAL、检查点

1.日志文件

日志文件是以事务为单位用来记录数据库的每一次更新活动的文件，由系统自动记录
日志内容包括：记录名、旧记录值、新记录值、事务标识符、操作标识符等
事务 Ti 开始时，写入日志：<Ti start>
事务 Ti 执行 write(X)前，写入日志：<Ti , X, V1, V2>，V1 是 X 更新前的值，V2 是 X 更新后的值
事务 Ti 结束后，写入日志：<Ti commit>
input(X)：将包含数据库元素 X 的磁盘块拷贝到内存缓冲区
read(X, t)：将数据库元素 X 拷贝到事务的局部变量 t
write(X, t)：将局部变量 t 的值拷贝到内存缓冲区中的数据库元素 X
output(X)：将包含 X 的缓冲区拷贝回磁盘
read 和 write 由事务发出，input 和 output 由缓冲区管理器（或日志管理器）发出

圆满事务：日志文件中记录了事务的 commit 标识
夭折事务：日志文件中只有事务的 Begin transaction 标识，无 commit

2.先写日志的原则（WAL）

对于尚未提交的事务，在将 DB 缓冲区写到外存之前，必须先将日志缓冲区内容写到外存去

日志记录将要发生何种修改

写入 DB 表示实际发生何种修改

如果先写 DB，则可能在写的中途发生系统崩溃，导致内存缓冲区内容丢失，而外存 DB 处于不一致状态，由于日志缓冲区内容已破坏，导致无法对 DB 恢复

3.检查点

当系统故障发生时，我们必须搜索整个日志，以决定哪些事务需要 redo，哪些需要 undo。大多数需要被重做的事务其更新已经写入了数据库中（redo2）。尽管对它们重做不会造成不良后果，但会使恢复过程变得更长

检查点原理

保证在检查点时刻日志与数据库内容是一致的

带有检查点记录的日志生成

将当前日志缓冲区的所有日志记录写入稳存中，在日志文件中写入一个检查点记录
将当前数据缓冲区的所有数据记录写入稳存中，输出检查点时活跃事务的列表 L

七、 各种故障恢复措施

1.事务故障恢复

撤消事务已对数据库所做的修改

措施

反向扫描日志文件，查找该事务的更新操作
对该事务的更新操作执行逆操作，即将事务更新前的旧值写入数据库
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
如此处理下去，直至读到此事务的开始标识，事务的故障恢复就完成了

2.系统故障恢复

不一致状态原因

未完成事务对数据库的更新已写入数据库
已提交事务对数据库的更新未写入数据库

措施

正向扫描日志文件，找出圆满事务，记入重做队列；找出夭折事务，记入撤消队列

反向扫描日志，对撤消队列中事务 Ti 的每一个日志记录执行 undo 操作
正向扫描日志文件，对重做队列中事务 Ti 的每一个日志记录执行 redo 操作

3.介质故障恢复

磁盘上数据文件和日志文件遭到破坏

措施

装入最新的数据库后备副本，使数据库恢复到最近一次转储时的一致性状态
装入相应的日志文件副本，重做已完成的事务

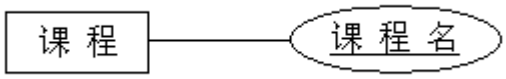
第二章 ER 模型作业参考答案

一 用 ER 图可以表达下列哪些数据完整性约束，不能表达哪些约束？能表达的给出 ER 图。

1. 每门课选课人数不能低于 10 个，不能高于 100 个

问题点评：很多学生将（10, 100）写在“选课”与“课程”之间。
xor 注：答案错，应写在右边。这个图表示的是一个学生参与 10 到 100 个选课关系。

2. 课程名是唯一的



3. 不能供应不存在的零件

问题点评：“不存在零件”描述本身不是很清晰。一般理解，数据库中的每个元素都代表存在的零件，则 ER 图本身就具有“不能供应不存在”的涵义。这个问题上，学生的理解也各不相同，很大一部分人选择了不能表示。

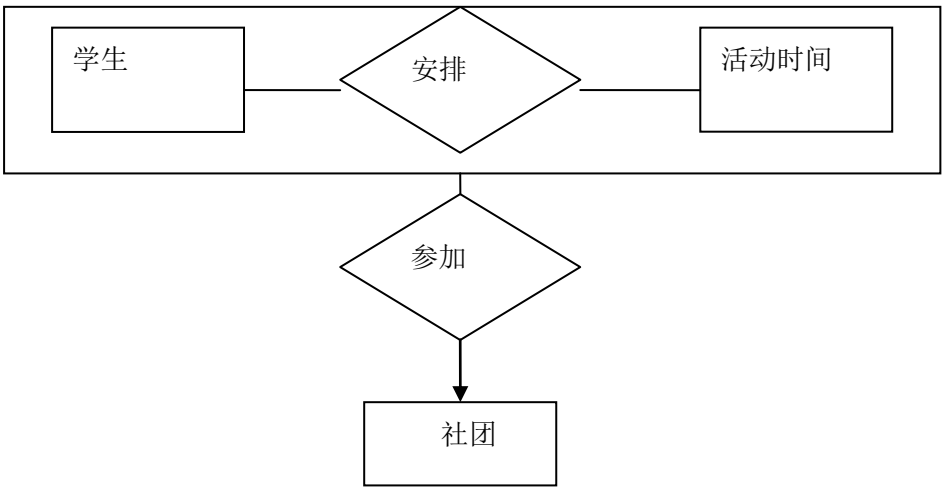
4. 性别只能为男或女

不能表示
问题点评：ER 图本身不具有枚举功能。有些学生将泛化概念加上完整性约束，也可以达到模拟的效果。

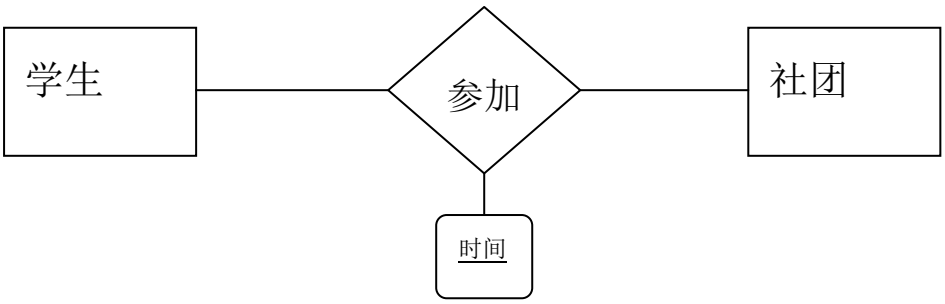
5. 每个学生都必须得选课

6. 学生可以参加多个社团，但所参加的社团的活动时间必须不同、

若将活动时间看成是社团的属性，则不可以表示。若是看成是实体则可以表示。



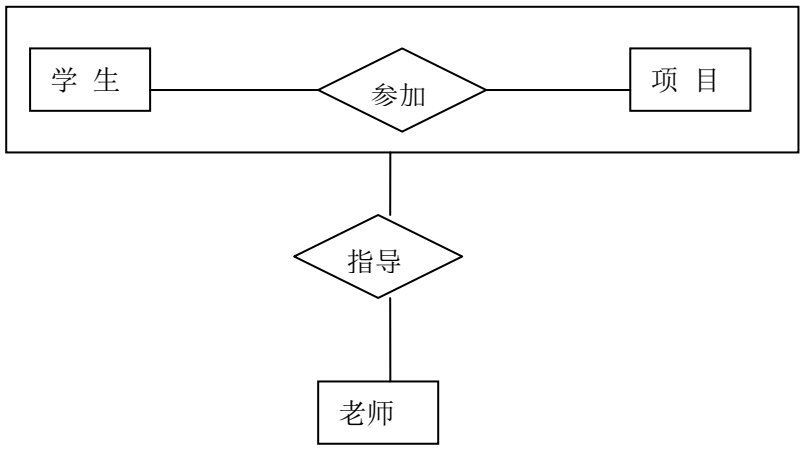
问 题点评：如果把“时间”看成是“学生”和“社团”之间的“活动”联系的属性。属性是没有办法表达其唯一性的。有些同学认为可以表示，例如



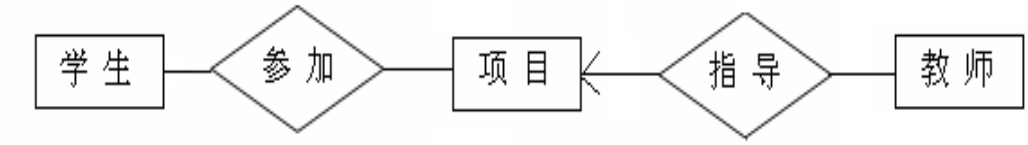
该例子混淆了时间的属性和实体的概念。

7. 学生可以参加多个项目，参加不同的项目其指导老师也不同

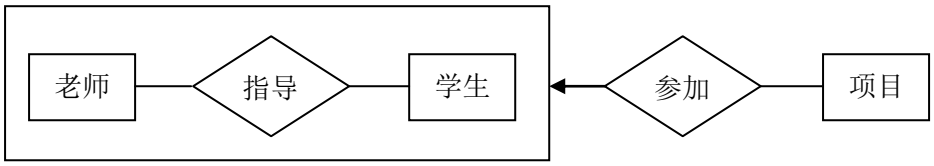
这个问题有两种理解，第一种就是对每个“参加”联系，有一个指导老师。则可以用聚集：



第二种理解就是对每个项目都有一个老师，老师是与项目相关的，可以用二元关系：

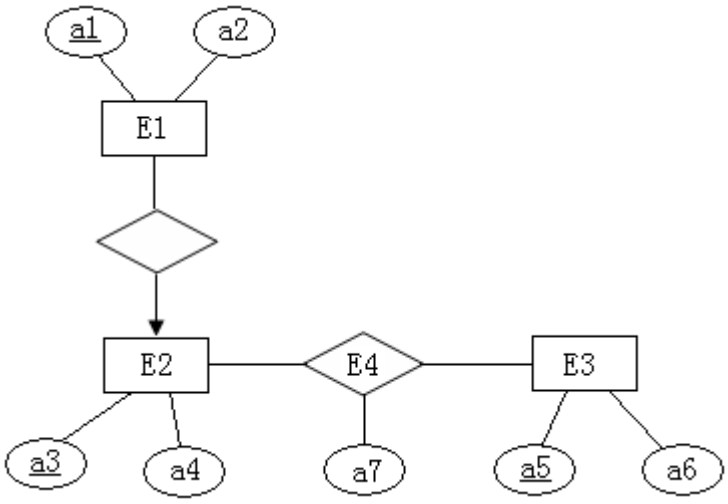


问题点评：上述两种理解都有不少同学。但是这道题另外还有很多其他的答案，感觉上理解比较混乱，例如：
在第一种情况中，出现了答案为：



将老师与学生直接联系起来，这是没有道理的。
第二种情况中，出现的主要问题是项目和教师之间的“不同项目老师也不同”的意思表达方面。有的用完全参与来解决这个问题。

三 已知有如下关系模式：E1(a1，a2，a3)，E2(a3，a4)，E3(a5，a6)， E4(a3，a5，a7)，其中带下划线的属性标识为所在关系模式的主码。试画出相应的 E-R 图，使得可以从该 E-R 图推导出上述关系模式。

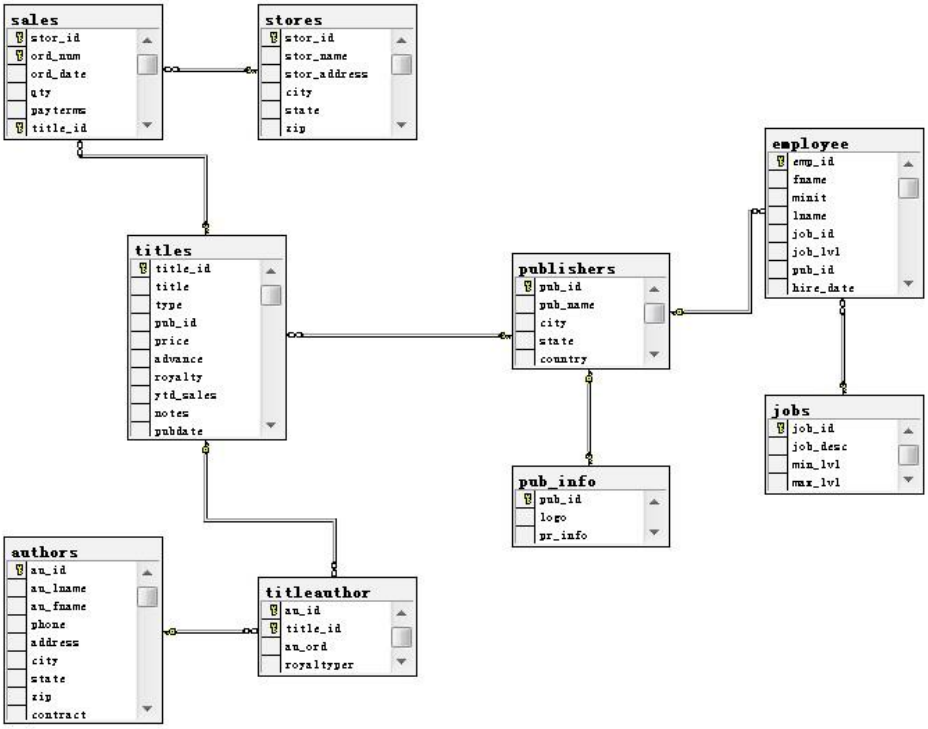


问题点评：很多同学忽略了 E1 和 E2 之间的一对多关系。如果没有这个关系 E1 和 E2 之间联系是需要关系来描述的。

六． 在 SQL Server 中有两个示例数据库，pubs 和 Northwind，要求：

1. 选择其中的一个，画出表之间的关系图。

SQL Server 本身有一个工具，是可以查看表之间的关系图的。例如 pubs 数据库的表关系图如下：



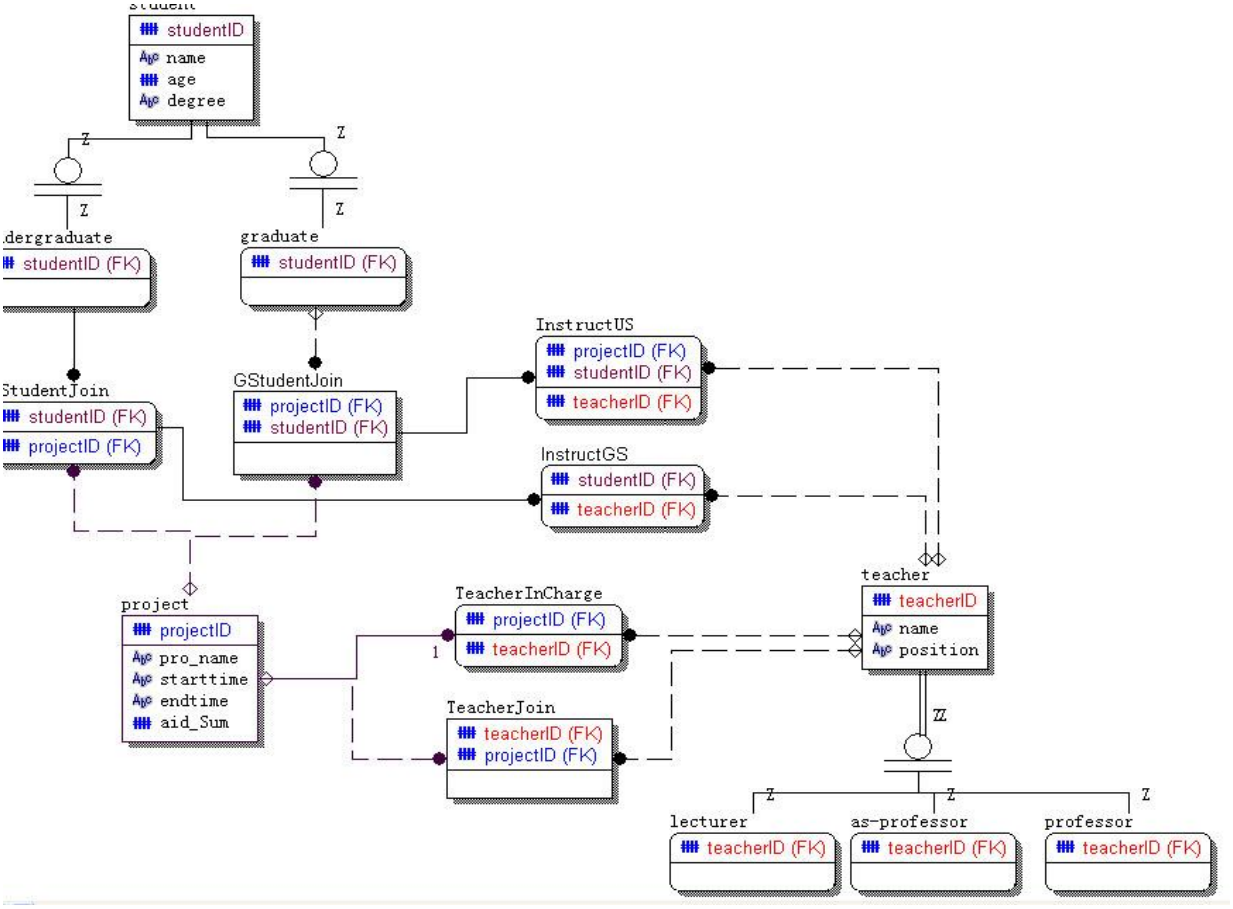
2. 画出相应的 ER 图。

答案略

3. 建立维度模型，以满足特定（自己拟定）的分析需求。

七． 考虑设计一个数据库，它要存储以下信息：教师有教工号、教工名、职称；项目有项目号、项目名称、项目类型、起始年份、截至时间、资助额；学生有学号、学生名、年龄、学位。学生分为本科生和研究生，老师按职称可以分为讲师、副教授、教授，副教授以上职称的可以作为研究生的导师。一个教工可以负责多个项目；每个项目只能有一个负责人；一个老师可以参与多个项目；一个本科生只能参与一个项目，一个研究生学生可以参与多个项目；一个项目可以有多个学生和老师参与；学生参与项目时必须（如果改为可以呢？）有一个老师作为他的指导老师。

1. 用 **ER** 图表示出信息需求（要求使用 **erwin** 工具）



作业点评：

“学生参与项目时必须有一个老师作为他的指导老师”使得必须要聚合才能完成上述关系。很多同学在 ER 图中没有表示出这种关系出来。如果对每个学生参与项目这个联系没有完整性约束的话，可以不用聚合，而用三元关系就可以了。具体见 6.er1

这道题做的好的同学不多。可能是软件环境也不是很熟悉，很少有同学能表达清楚所有的要求。例如本科生只能选一个项目，每个项目只能有一个人负责，还有每个同学在参与一个项目时都必须有一个老师指导等要求，都没有很好的表达出来。

2. 转换为关系模型，并图示出关系间的主、外码关联

见上图。

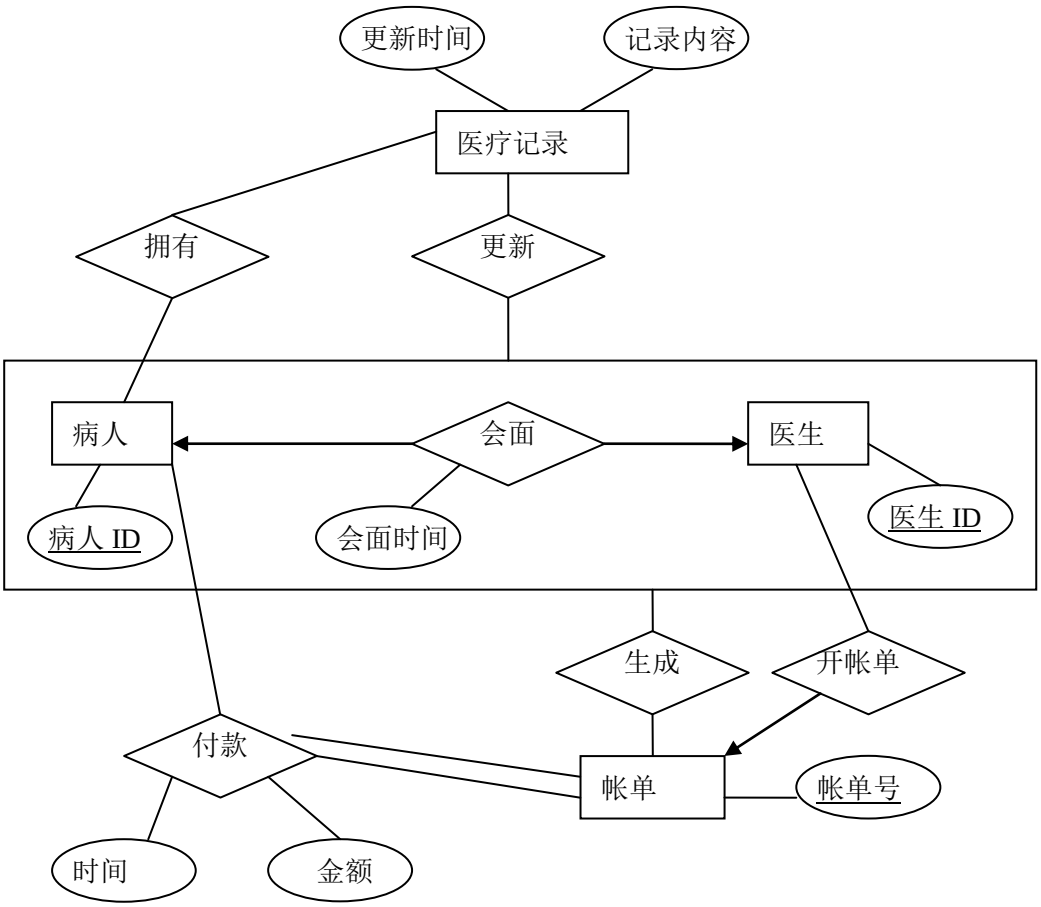
八． 为下面的医疗诊所创建一个 ER 图，要求：

1.一个病人可能同该诊所中的一个或多个医生约定时间会面，一个医生可以接收许多病人的会面。但是，每次会面只能有一个医生，所牵涉的病人只能有一个。

2. 每次会面都会更新病人的医疗记录，从而生成一次医疗记录。

3. 每次会面都生成一个账单，每次会面只能由一个医生开账单，一个医生可以给很多病人开账单。

4. 必须支付每个账单，但一个账单可以分为多次支付，一次付款可以支付很多账单。



作业点评：

1. 很多同学没有注意到帐单的生成和医疗记录的更新实际上是和会面关系发生的关系，因此没有用聚合。
2. 有些同学没有表达出来每个帐单只能由一个医生开的限定条件。
3. 大部分同学生成了付款这一实体，但是实际上把付款堪称病人和帐单之间的关系更恰当。如果不这样的话，很难表达所有帐单必须要付的意思。
4. 医疗记录应该由病人拥有，很多同学没有发现这一点。该关系可以将医疗记录看成是依赖病人的实体，但是实际上也可以看成独立的实体，上图看成独立的实体。

数据库概论
第三次作业参考答案

November 14, 2007

1 题目

$$
\begin{array}{ll}
S & (SNO, SNAME, STATUS, CITY) \\
P & (PNO, PNAME, COLOR, WEIGHT, CITY) \\
J & (JNO, JNAME, CITY) \\
SPJ & (SNO, PNO, JNO, QTY)
\end{array}
$$

S表示供应商，各属性依次为供应商号，供应商名，供应商状态值，供应商所在城市；

P表示零件，各属性依次为零件号，零件名，零件颜色，零件重量，零件存放的城市；

J表示工程，各属性依次为工程号，工程名，工程所在城市；

SPJ表示供货关系，各属性依次为供应商号，零件号，工程号，供货数量。

基于以上SPJ关系模式用关系代数表达查询：

1. 求向北京的工程供应了红色零件的供应商姓名。
2. 求只向北京的工程供应零件的供应商姓名。
3. 求至少供应了两种不同零件的供应商姓名。
4. 求只供应了两种不同零件的供应商姓名。
5. 求没有供应任何零件的供应商姓名。
6. 求供应了所有零件的供应商姓名。
7. 求供应了所有红色零件的供应商姓名。
8. 求供应了s1号供应商所供应的所有零件的供应商的供应商号。
9. 求和s1号供应商所供应的零件完全相同的供应商的供应商号。
10. 求只供应了p1号零件的供应商号码。
11. 求只供应了p1号和p2号零件的供应商号码。

2 参考答案

说明：答案可能不只一种。

1. 求向北京的工程供应了红色零件的供应商姓名。
   $\prod_{SNAME}(\sigma_{S.SNO=SPJ.SNO \wedge P.PNO=SPJ.PNO \wedge J.JNO=SPJ.JNO \wedge J.CITY=\text{“北京”} \wedge P.COLOR=\text{“红”}}(S \times P \times J \times SPJ))$

2. 求只向北京的工程供应零件的供应商姓名。
   $\prod_{SNAME}(\sigma_{S.SNO=SPJ.SNO \wedge J.JNO=SPJ.JNO \wedge J.CITY=\text{“北京”}}(S \times J \times SPJ)) - \prod_{SNAME}\left(\sigma_{S.SNO=SPJ.SNO \wedge J.JNO=SPJ.JNO \wedge J.CITY \neq \text{“北京”}}(S \times J \times SPJ)\right)$

3. 求至少供应了两种不同零件的供应商姓名。
   $\prod_{SNAME}(\sigma_{X.SNO=S.SNO}(\sigma_{X.SNO=Y.SNO \wedge X.PNO \neq Y.PNO}(\rho_X(SPJ) \times \rho_Y(SPJ)) \times S))$

4. 求只供应了两种不同零件的供应商姓名。
   $\prod_{SNAME}(\sigma_{X.SNO=S.SNO}(\sigma_{X.SNO=Y.SNO \wedge X.PNO \neq Y.PNO}(\rho_X(SPJ) \times \rho_Y(SPJ)) \times S)) - \prod_{SNAME}(\sigma_{X.SNO=S.SNO}(\sigma_{X.SNO=Y.SNO \wedge Y.SNO=Z.SNO \wedge X.PNO \neq Y.PNO \wedge Y.PNO \neq Z.PNO \wedge X.PNO \neq Z.PNO}(\rho_X(SPJ) \times \rho_Y(SPJ) \times \rho_Z(SPJ)) \times S))$

5. 求没有供应任何零件的供应商姓名。
   $\prod_{SNAME}(S) - \prod_{SNAME}(S \bowtie SPJ)$

6. 求供应了所有零件的供应商姓名。
   $\prod_{SNAME,PNO}(S \bowtie SPJ) \div \prod_{PNO}(P)$

7. 求供应了所有红色零件的供应商姓名。
   $\prod_{SNAME,PNO}(S \bowtie SPJ) \div \prod_{PNO}(\sigma_{COLOR=\text{“红”}}(P))$

8. 求供应了s1号供应商所供应的所有零件的供应商的供应商号。
   $\prod_{SNO,PNO}(SPJ) \div \prod_{PNO}(\sigma_{SNO=s1}(SPJ))$

9. 求和s1号供应商所供应的零件完全相同的供应商的供应商号。
   $(\prod_{SNO,PNO}(SPJ) \div \prod_{PNO}(\sigma_{SNO=s1}(SPJ))) - \prod_{SNO}((\prod_{PNO}(SPJ) - \prod_{PNO}(\sigma_{SNO=s1}(SPJ))) \bowtie SPJ)$

10. 求只供应了p1号零件的供应商号码。
    $\prod_{SNO}(SPJ) - \prod_{SNO}(\sigma_{PNO \neq p1}(SPJ))$

11. 求只供应了p1号和p2号零件的供应商号码。
    $\prod_{SNO}(\sigma_{PNO=p1}(SPJ)) \cap \prod_{SNO}(\sigma_{PNO=p2}(SPJ)) - \prod_{SNO}(\sigma_{PNO \neq p1 \wedge PNO \neq p2}(SPJ))$

关系规范化参考答案

1. 分别写出关系代数和 SQL 语句，验证关系 R(ABC)上 A→B 和 A→→B 是否成立

答：(1) 验证 A→B 是否成立

(a)关系代数语句：

若以下语句选出的集合为空，则 A→B 成立，否则不成立

$$\sigma_{R1.A=R2.A \ \wedge \ R1.B\neq R2.B}(\rho_{R1}(R)\times\rho_{R2}(R))$$

(b)SQL 语句：

```
create assertion    A1     check
   (not exists( select * from R R1,R R2
                where R1.A=R2.A
                and     R1.B<>R2.B))
```

(2)验证 A→→B 是否成立

(a)关系代数语句：

若 $\Pi_{AB}(R) \bowtie \Pi_{AC}(R)$ 等于 R，则 A→→B 成立，否则不成立。

(b)SQL 语句：

```
create assertion    A2     check
(not exists(    select * from R R1,R R2
                where     R1.A=R2.A
                and       R1.B<>R2.B
                and       not exists(     select * from R R3,R R4
                                          where     R3.A=R1.A
                                          and       R4.A=R2.A
                                          and       R3.B=R1.B
                                          and       R4.B=R2.B
                                          and       R3.C=R2.C
                                          and       R4.C=R1.C    )
                ))
```

点评：验证 A→→B 的关系代数同学们错的较多，想用定义来写的很难写对。下面给出一种错误写法：验证下面集合是否为空

$$\sigma_{T.A = S.A \ \wedge \ (T.A=P.A \wedge T.B=P.B \wedge S.C=P.C) \wedge (S.A=Q.A \wedge S.B=Q.B \wedge T.C=Q.C)}$$

$$(\rho_T(R)\times\rho_S(R)\times\rho_P(R)\times\rho_Q(R))$$

这种写法错在不能保证任意一对元组 t1,t2,当 t1[A]=t2[A]时，都会存在 t3,t4。

另外在写 SQL 的时候，不要用赋值和声明。

2. R(ABCDE), F = {AB→C, B→D, C→E, CE→B, AC→B}。

给出其候选码

判断范式级别

分别给出保持无损连接和函数依赖的分解

1) 左部属性：{A}

右部属性：{D}

双部属性：{B，C，E}

$$(A)_F^+ = A$$

$$(AB)_F^+ = ABCDE$$

$$(AC)_F^+ = ACBDE$$

$$(AE)_F^+ = AE$$

因此 AB，AC 是候选码。

2） 取 AB 为码，则 D 部分依赖码

取 AC 为码，则 E 部分依赖码

故不满足 2NF

所以范式级别为 1NF。

点评：**这两道题错的较少。但仍有同学在第二题误认为是 2NF。**

3） 用达到 BCNF 无损连接分解算法做：

将 R 分为：R1={B,D}. F1={B→D}

R2={A,B,C,E}. F2={ AB→C, C→E, CE→B, AC→B }

再将 R2 分为：R21={C,E} F3={ C→E }

R22={A,B,C} F4={ AB→C , AC→B }

得到保持无损连接的分解为：

{<BD,{ B→D }>,<CE,{ C→E }>,<ABC,{ AB→C , AC→B }>}

用达到 3NF 且保持函数依赖的分解算法做：

Fmin={ AB→C, B→D, C→E, C→B }

得到保持函数依赖的分解：

{R1<ABC,{ AB→C}>, R2<BD,{ B→D}>, R3<BCE,{ C→E, C→B }>}

说明：此小题还可用达到 3NF 且同时保持无损连接与函数依赖的分解算法。

点评：**此题需要用课堂上所讲的算法来做，而不是自己尝试着给出答案，再验证。**

3. 已知 R(ABCD)的封闭属性集是Φ，{AB}和{ABCD},给出 R 的函数依赖集

R 的函数依赖集为 F={A->B,B->A,C->D,D->C,D->A}

或者 F={A->B,B->A,C->D,D->C,C->A}

或者 F={A->B,B->A,C->D,D->C,D->B}

或者 F={A->B,B->A,C->D,D->C,C->B}

说明：答案不限于上述几种。

点评：此题有的同学给出 F={A→B, C→D,D→A,B→A}，这个答案是错误的，可以验证{ABD}。

4. R(ABCDE), 给出下面函数依赖集在 S(ABCD)上的投影

F={AB → D,AC → E,BC → D,D → A, E → B}

A^+={A},B^+={B},C^+={C},D^+={DA}

AB^+={ABD},AC^+={ACEBD},AD^+={AD}

BC^+={BCDAE},BD^+={BDA},CD^+={CDAEB}

ABC^+={ABCDE},ABD^+={ABD},ACD^+={ACDEB},BCD^+={BCDAE}

F_i={D→A,AB→D,AC→B,AC→D,BC→A,BC→D,~~BD→A~~,CD→B,~~ABC→D~~,~~ACD→B~~,~~BCD→A~~}

={D→A,AB→D,AC→B,~~AC→D~~,~~BC→A~~,BC→D, ~~CD→B~~}

={D→A,AB→D,AC→B, BC→D } （最小覆盖）

点评：此题同学们大都没有写过程。投影用最小覆盖表示即可。

5. 证明多值传递律和联合律

（1）多值传递律

描述：若 X →→ Y，Y →→ Z，则 X →→ Z－Y

证明：

对于R上任意元组t1, t2若满足**t1[X] = t2[X]**

因为X →→ Y

则存在t3, t4： t3[X] = t1[X] = t2[X], t3[Y] = t1[Y], t3[R-X-Y] = t2[R-X-Y] ①

t4[X] = t2[X] = t1[X], t4[Y] = t2[Y], t4[R-X-Y] = t1[R-X-Y] ②

那么由Y→→Z

那么我们有t1[Y] = t3[Y] ——①

存在t5满足t5[Y] = t1[Y], t5[Z] = t3[Z], t5[R-Y-Z] = t1[R-Y-Z]

又由于t3[X] = t1[X] ——①

可知**t5[X] = t1[X]**

因为t3 [R-Y] = t2[R-Y] ——① (除了Y部分，其他的关系都相等)

于是有**t5[Z-Y]** = t3[Z-Y] = **t2[Z-Y]**

由于t5[R-Y-Z] = t1[R-Y-Z], t5[Y] = t1[Y]

于是有t5[R-(Z-Y)] = t1[R-(Z-Y)]

于是有 **t5[R-(Z-Y)-X] = t1[R-(Z-Y)-X]**

同理可证，利用t2[Y] = t4[Y]，Y→→Z，

存在t6满足**t6[X] = t2[X], t6[Z-Y] = t1[Z-Y], t6[R-(Z-Y)-X] = t2[R-(Z-Y)-X]**

于是可以得到X →→ Z－Y

（2）多值联合律

描述：若 X →→ Y，Z⊆Y，且存在 W，使得 W⊆R，W∩Y=Φ，W→Z，则 X →Z

证明：

设关系模式 R<U,F>, X⊆U, Y⊆U, r 是 R<U,F>上的任一关系，设 t1, t2 ⊆ r,

为了证明 X 〉 Z，即证若 t1[X]=t2[X],那么 t1[Z]=t2[Z].

设 t1[X]=t2[X]，由于 X →→ Y，则存在元组 t3,使得

t3=(t2[X],t1[Y],t2[U-X-Y])，其中：

t1=(t1[X],t1[Y],t1[U-X-Y])

t2=(t2[X],t2[Y],t2[U-X-Y])

可以看出，t3[U-Y]=t2[U-Y]

即 t3,t2 元组在 Y 属性组以外的属性组上值相等。

又因为 W∩Y=Φ，即 W 是在 Y 属性组以外的属性组，所以 t3[W]=t2[W]

又由条件 W→Z，得到 t3[Z]=t2[Z]。

又由于 t3[Y]=t1[Y]，Z⊆Y，故 t3[Z]=t1[Z]，由上式得 t1[Z]=t2[Z]。

即当 t1[X]=t2[X]时，可推出 t1[Z]=t2[Z]，即证明了 X →Z。

点评：联合律证明较为简单。同学们证明传递律的大致思路都是正确的，但写的欠缺条理。