一、关系代数的9种操作：

关系代数中包括了：并、交、差、乘、选择、投影、联接、除、自然联接等操作。

五个基本操作：  
并(∪)、差(-)、笛卡尔积(×)、投影(σ)、选择(π)

四个组合操作：  
交(∩)、联接(等值联接)、自然联接(RcrossS)、除法(÷)  
注2：等值连接表示先做笛卡尔积(×)之后，对相应列进行选择或等值关联后的结果(仅筛选行、不筛选列)  
注2：自然连接表示两个关系中若有相同名称的属性，则自动作为关联条件，且仅列出一列

二、关系代数表达式：

由关系代数运算经有限次复合而成的式子称为关系代数表达式。这种表达式的运算结果仍然是一个关系。可以用关系代数表达式表示对数据库的查询和更新操作。

三、举例说明：

设教学数据库中有3个关系：

学生关系S(SNO,SNAME,AGE,SEX)

学习关系SC(SNO,CNO,GRADE)

课程关系C(CNO,CNAME,TEACHER)

(1) 检索学习课程号为C2的学生学号与成绩

SELECT SNO,GRADE

FROM SC

WHERE CNO='C2'

π SNO,GRADE(σCNO='C2'(SC))

(2) 检索学习课程号为C2的学生学号与姓名

SELECT SC.SNO,S.SNAME

FROM SC,S

WHERE SC.SNO=S.SNO

AND SC.CNO='C2'

π SNO,SNAME(σCNO='C2'(ScrossSC))  
此查询涉及S和SC，先进行自然连接，然后再执行选择投影操作。

π SNO,SNAME（S）cross（πSNO(σCNO='C2'(SC))）  
自然连接的右分量为"学了C2课的学生学号的集合"。  
此表达式比前一个表达式优化，执行起来要省时间、省空间。

(3) 检索选修课程名为MATHS的学生学号与姓名

SELECT SC.SNO,S.SNAME

FROM SC,S,C

WHERE SC.SNO=S.SNO

AND SC.CNO=C.CNO

AND C.CNAME='MATHS'

π SNO,SANME(σCNAME='MATHS'(ScrossSCcrossC))

(4) 检索选修课程号为C2或C4的学生学号

SELECT SNO

FROM SC

WHERE CNO='C2'

OR CNO='C4'

π SNO(σ CNO='C2'∨CNO='C4'(SC))

(5) 检索至少选修课程号为C2或C4的学生学号

SELECT SA.SNO

FROM SC AS SA,SC AS SB

WHERE SA.SNO=SB.SNO

AND SA.CNO='C2'

AND SB.CNO='C4'

π 1(σ1=4∧2='C2'∧5='C4'（SC×SC）)

(6) 检索不学C2课的学生姓名与年龄

SELECT SNAME,AGE

FROM S

MINUS

SELECT S.SNAME,S.AGE

FROM SC,S

WHERE SC.SNO=S.SNO

AND SC.CNO='C2'

(Oracle)

π SNAME,AGE（S）－πSNAME,AGE(σCNO='C2'（ScrossSC）)

(7) 检索学习全部课程的学生姓名  
这个定义用SQL表示比较麻烦，略过  
π SNO,CNO(SC)÷πCNO(C)  
先用除法取出选取所有课程的SNO集(除法可以理解为一个Filter)  
π SNAME(S cross (πSNO,CNO(SC)÷πCNO(C)))  
再关联S表取出SNAME

(8) 检索所学课程包含S3所学课程的学生学号  
这个定义用SQL表示比较麻烦，略过  
π SNO,CNO(SC)÷ πCNO(σSNO='S3'(SC)）  
同样运用了除法的特性

(9) 将新课程元组('C10','PHYSICS','YU')插入到关系C中

INSERT INTO C VALUES('C10','PHYSICS','YU')

(C∪('C10','PHYSICS','YU'))  
记住该符号的用法

(10) 将学号S4选修课程号为C4的成绩改为85分

UPDATE SC SET GRADE=85

WHERE SNO='S4'

AND CNO='C4'

（SC－（'S4','C4',?)∪('S4','C4',85)）  
先用'－'实现DELETE功能，再用'∪'实现INSERT功能  
注意使用?来表示检索时忽略该字段值

**基本运算**

每一种基本运算的结果都是一个新的关系，可以用这个关系继续参与运算，借此便可进行复杂的运算

* **选择运算（select）==>相当于SQL语句中的WHERE子句的职能**
  + **格式：σ选择谓词(关系)**
  + 举个栗子：
    - σSAL>1000(EMP)
    - 上式表示取出查询工资大于1000的所有员工的信息
    - 等价于下面的SQL语句
  + SELECT \*
  + FROM EMP
  + WHERE SAL > 1000
  + 选择谓词的分类
    - 比较：=、≠、<、≤、>和≥
    - 连词：and(∧)、or(∨)和not(¬)==>可以将多个谓词合并成一个大的谓词
    - 可以包括两个属性（字段的比较）：σCOMM>SAL(EMP)表示抽成大于工资的人
* **投影运算（project）==>相当于SQL语句中的SELECT子句的职能**
  + **格式：∏字段序列(关系)**
  + 举个栗子
    - ∏ENAME,SAL(EMP)
    - 上式表示查看所有员工的姓名和工资
    - 等价于下面的SQL语句
  + SELECT ENAME, SAL
  + FROM EMP
* **关系的组合运算==>就像SQL中select、where子句那样的组合效果**
  + 举个栗子
    - ∏ENAME,SAL(σSAL>1000(EMP))
    - 上面的式子求出了所有工资大于1000的员工的名字和工资(实际上就是将**σSAL>1000(EMP)**执行的结果当做一个临时的关系，参与了投影运算得到的)
    - 等价于下面的SQL语句
  + SELECT ENAME, SAL
  + FROM EMP
  + WHERE SAL > 1000
  + 事实关系的组合运算就是那么简单，分析的时候把每个简单运算的结果当做一个新的关系参与后面的运算，这样一层层剥开来，再复杂的语句也变得容易分析
* **并运算（union）==>相当于SQL中UNION关键字的职能**
  + **格式：（关系r）∪（关系s）**
  + 举个栗子
    - ∏ENAME,SAL(σSAL>1000(EMP)) ∪ ∏ENAME,SAL(σCOMM>300(EMP))
    - 上面的式子求出了所有工资大于1000或抽成大于300的员工的姓名和工资，等价于下面的SQL语句
  + SELECT ENAME, SAL
  + FROM EMP
  + WHERE SAL > 1000
  + UNION
  + SELECT ENAME, SAL
  + FROM EMP
  + WHERE COMM > 300
  + **几点需要额外注意的**：
    - 此处的并运算是集合运算，所以结果是去重的，结果集中不存在重复的元组（***而在SQL语句中，指定UNION ALL是可以保留重复的***）
    - 关系r与关系s必须是同元的，即它们的属性的数目要求必须相同（这就和SQL语句中UNION使用的时候要求上下两个语句的字段数相同是一样的意思）
    - 关系r和关系s对应位置的属性域应该是类型兼容的（同样和SQL中UNION使用时，每个对应位置字段类型兼容是一样的意思）
* **集合的差运算（set-defference）==>相当于SQL语句中的EXCEPT**
  + **格式：(关系r)-(关系s)**
  + 举个栗子
    - ∏ENAME,SAL(σSAL>1000(EMP)) - ∏ENAME,SAL(σCOMM>300(EMP))
    - 上面的式子表示工资大于1000但抽成不大于300的员工的姓名和工资，等价于下面的SQL语句
  + SELECT ENAME, SAL
  + FROM EMP
  + WHERE SAL > 1000
  + EXCEPT
  + SELECT ENAME, SAL
  + FROM EMP
  + WHERE COMM > 300
  + **几点需要额外注意的**：
    - 此处的注意同上面的并运算的注意事项
* **笛卡尔积运算（Cartesian-product）==>等价于SQL语句中两个表进行笛卡尔积（全匹配）得到的结果，即SQL中进行多表连接时不指定连接条件的情况**
  + **格式：(关系r)×(关系)**
  + 举个栗子：
    - EMP × DEPT
    - 上面的式子表示两个表进行全匹配，等价于下面的SQL语句
  + SELECT \*
  + FROM EMP, DEPT
  + 下面两个式子是等价的
    - **∏ENAME,DNAME(σEMP.DEPTNO=DEPT.DEPTNO(σJOB="MANAGER"(EMP×DEPT)))**
    - **∏ENAME,DNAME(σEMP.DEPTNO=DEPT.DEPTNO((σJOB="MANAGER"(EMP))×DEPT)**
    - 下面是对这两个式子的SQL转化，转化之后就一目了然了
    - -- 对应第一个式子
    - SELECT ENAME, DNAME
    - FROM EMP JOIN DEPT ON EMP.DEPTNO = DEPT.DEPTNO
    - WHERE JOB = 'MANAGER'
    - -- 对应第二个式子
    - SELECT ENAME, DNAME
    - FROM DEPT JOIN (SELECT \*
    - FROM EMP
    - WHERE JOB = 'MANAGER')
    - ON EMP.DEPTNO = DEPT.DEPTNO
* **更名运算（rename）==>等价于SQL语句中as的职能**
  + **格式：ρX(A1,A2,...,An)(E)==>表示的是将关系E更名为X，Ai表示的是给E的第i个字段指定别名**
  + 举个栗子
    - ∏ENAME,DNAME(σe.DEPTNO=d.DEPTNO(σJOB="MANAGER"(ρe(EMP)×ρd(DEPT))))
    - 上面式子含义就不解释了，是上面举的栗子，只是引入了更名运算符，它等价于下面的SQL语句
  + SELECT ENAME, DNAME
  + FROM EMP e JOIN DEPT d ON e.DEPTNO = d.DEPTNO
  + WHERE JOB = 'MANAGER'
* **来，学习完上面的基本运算，来做个实际的栗子，要求找到员工表中的最高工资（因为目前还没有介绍类似SQL中组函数的操作，后面会介绍。所以通过以下方式来实现）**
  + step1: 找到所有不是最高工资的人
    - **∏e1.SAL(σe1.sal < e2.sal(ρe1(EMP)×ρe2(EMP)))**
  + step2: 用所有的员工减去上面的员工，即得到最高工资
    - **∏SAL(EMP) - ∏e1.SAL(σe1.sal < e2.sal(ρe1(EMP)×ρe2(EMP)))**
* **在书写关系运算表达式的时候可以用序列号代替字段名（但是不直观，不常用，一般不用）**
  + 举个栗子
    - ∏$6(σ$6 < $14(EMP×EMP))
    - 等价于下面的运算
    - ∏e1.SAL(σe1.sal < e2.sal(ρe1(EMP)×ρe2(EMP)))

**附加运算**

附加运算是由基本运算组成的，不能增强基本运算的运算能力，但是能简化运算

* **集合交运算（intersection）==>相当于SQL语句中INTERSECT关键字的职能**
  + **格式：(关系r)∩(关系s)**
  + 因为集合交运算是可以由前面的基本运算组合产生的，所以把它归到附加运算
    - A ∩ B <=> A - (A - B)
  + 举个栗子
    - ∏ENAME,SAL(σSAL>1000(EMP)) ∩ ∏ENAME,SAL(σCOMM>300(EMP))
    - 上面的式子表示工资大于1000并且抽成大于300的员工的姓名和工资，等价于下面的SQL语句
  + SELECT ENAME, SAL
  + FROM EMP
  + WHERE SAL > 1000
  + INTERSECT
  + SELECT ENAME, SAL
  + FROM EMP
  + WHERE COMM > 300
* **自然连接（natural join）==> 相当于SQL语句中的NATURAL JOIN**
  + **格式：(关系)⋈(关系)**
  + **自然连接的形式化定义**
    - r,s是两个关系
    - R，S是上面两个关系对应的关系模式（其实就是上述两个关系各自的属性列表）
    - R ∩ S 表示r和s的同名属性列表
    - R ∪ S 表示出现在r或s上的属性名列表(是一个集合，不包同名属性，存在同名属性会去重)
    - R - S 表示出现在R上，但不出现在S上的属性名列表
    - 则可做如下定义
    - r⋈s = ∏R∪S(σr.A1=s.A1 ∧ r.A2=s.A2 ∧ ... ∧ r.An=s.An(r×s)) ，其中 R∩S={A1, A2, ..., An}
  + 举个栗子
    - ∏name, course\_id(instructor ⋈ teaches)
    - 上面的式子列出了所有老师的名字以及其所授课程的id，等价于下面的SQL语句
  + SELECT name, course\_id
  + FROM intructor natural join teaches

ps: 两个关系模式执行自然连接以后属性的排布顺序：

* + 排在最前面的是两个关系模式相同的属性
  + 其次是只属于第一个关系模式的属性
  + 最后是只属于第二个关系模式的属性

**！！！所以所两个关系模式进行自然连接以后，总的属性的个数是减少了，具体减少的个数等于同名属性的个数**

* **theta连接==>是带限定条件的笛卡尔积**
  + **格式：(关系)⋈Θ(关系)**
  + 形式化定义：
    - r ⋈Θ s = σΘ(r × s)
  + 举个栗子
    - ∏name, course\_id(instructor ⋈instructor.ID = teaches.ID ∧ instructor.salary > 5000 teaches)
    - 上面的式子表示列出所有工资大于5000的老师的名字以及其所授课程的id， 等价于下面的SQL语句
  + -- 使用 join...on 的时候 on 后面写连接条件，然后将其它条件放在where里
  + SELECT name, course\_id
  + FROM instructor join teaches on instructor.ID = teaches.ID
  + WHERE instructor.salary > 5000
* **除运算（division）**

这个在书上没讲，是老师上课的时候补充的

* + **格式：(关系)÷(关系)**
  + 形式化定义：  
    R÷S = ∏R∪S( ( ∏R-S(r) × S ) - ∏R-S, S(r) )
  + 解释起来挺麻烦的，这边给出一个博客链接：[点我](http://blog.csdn.net/qq_22627687/article/details/53789362" \t "_blank)
* **赋值运算**

就是将一个关系表达式的结果赋值取一个临时的名字，就相当于定义了一个临时关系。这个操作就相当于SQL中with语句的职能

* + **格式： temp\_name ← 关系表达式**
  + 举个栗子：  
    temp1 ← R × S  
    temp2 ← σr.A1=s.A1 ∧ r.A2=s.A2 ∧ ... ∧ r.An=s.An(temp1)  
    result = ∏R∪S(temp2)
  + 上面的式子等价于： result = r⋈s
* **外连接运算**
  + 左外连：⟕
  + 右外连：⟖
  + 全外连：⟗

**扩展运算**

扩展运算是不能用基本的关系代数运算来实现的一类查询，可以满足复杂的查询需求

* **广义投影（Generalized-projection）**

与基本运算中的投影运算相比，就是多了允许在选择列表中出现表达式（在基本运算中的投影的选择列表中只能出现字段）

* + **格式：∏F1, F2, ... , Fn(E)**
    - 其中F1, F2, ... , Fn可以是字段或者是表达式
    - E代表一个关系
  + 举个栗子：
    - ∏name, sal \* 1.2(instructor)
    - 上面的式子表示查出所有老师的名字，以及涨了20%以后的工资，等价于下面的SQL语句
  + SELECT name, sal \* 1.2
  + FROM instructor
* **聚集函数（Aggregation function）**

聚集函数的符号表示是用书写体G，这边就直接用G指代了

**聚集函数是输入值的一个汇聚，以多个值作为输入，将一个单一的值作为返回结果**

多重集：使用聚集函数对其进行操作的汇集中，一个值可以出现多次，值出现的顺序是无关紧要的。这样的汇集称为**多重集**（就比方说统计一个员工表中员工的数量，然后我们通过统计员工的名字来统计，即便是同名的员工我们也是计算的）

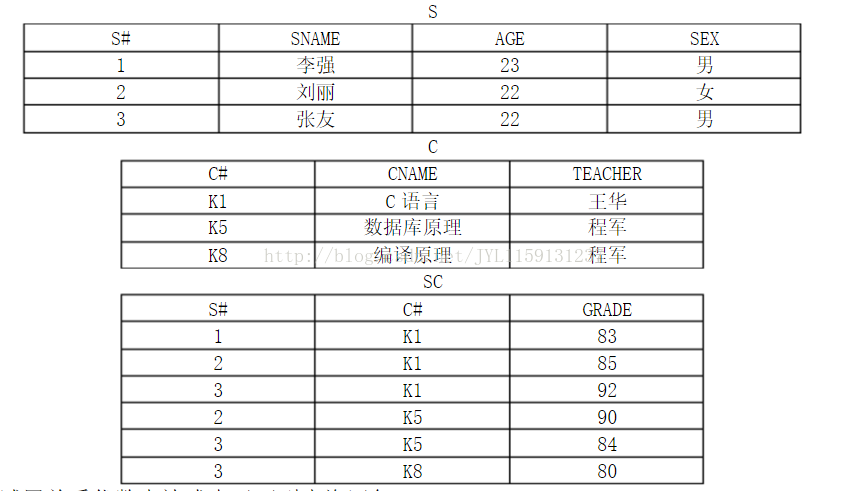
* + **格式： G1, G2, ... , GnGF1(A1), F2(A2), ..., Fn(An)(E)**
    - 其中前面的G1, G2, ... , Gn表示的是分组条件
    - 后面的F1(A1), F2(A2), ..., Fn(An)是聚集函数表达式列表
    - Fi(i = 1, 2, ..., n)表示聚集函数：sum、count、average、max、min
    - A1, A2, ... , An代表字段
  + 举个栗子：
    - A1, A2Gsum(A3)(∏A1, A2, ..., An(σP(r1×r2×...×rm)) )
    - 等价于下面的SQL语句
  + SELECT A1, A2, sum(A3)
  + FROM r1, r2, ..., rm
  + WHERE P
  + GROUP BY A1, A2
  + 上面的聚集函数在进行计算的时候采用的都是多重集，也就是相同的值可以多次重复计算（也就是在执行聚集函数的时候是不去重计算），如果要去重计算的话就要采用下面的几个函数写法
    - sum\_distinct
    - count\_distinct
    - averag\_distinct
    - max\_distinct
    - min\_distinct
  + 举个栗子：
    - A1, A2Gsum\_distinct(A3)(∏A1, A2, ..., An(σP(r1×r2×...×rm)) )
    - 上面的式子等价于下面的SQL语句
  + SELECT A1, A2, sum(distinct A3)
  + FROM r1, r2, ..., rm
  + WHERE P

GROUP BY A1, A2

题目来自百度文库，这里摘录出来方便复习。 ωϖ为连接符号。

------------------------------------------------------------------------------------------------------------------------------------------------------

1、设有如下所示的关系：学生表S(S#,SNAME,AGE,SEX)、课程表C(C#,CNAME,TEACHER)和学生选课表SC(S#,C#,GRADE)，其中：S#为学号，SNAME为姓名，AGE为年龄，SEX为性别，C#为课程号，CNAME为课程名，GRADE为成绩, TEACHER为教师。



试用关系代数表达式表示下列查询语句：

（1）检索“程军”老师所授课程的课程号(C#)和课程名(CNAME)。

         πC#,CNAME(σTEACHER=‘程军’(C))  

（2）检索年龄大于21的男学生学号(S#)和姓名(SNAME)。

         πS#,SNAME(σAGE>21∧SEX=‘男’(S)) 

（3）检索至少选修“程军”老师所授全部课程的学生姓名(SNAME)。

        πSNAME(Sωϖ (πS#,C#(SC)÷πC#(σTEACHER=‘程军’(C))))

（4）检索“李强”同学不学课程的课程号(C#)。

        πC#(C)-πC#(σSNAME=‘李强’(S) ωϖ SC)

（5）检索至少选修两门课程的学生学号(S#)。

        πS#(σ1=4∧2≠5(SC × SC))

（6）检索全部学生都选修的课程的课程号(C#)和课程名(CNAME)。

       πC#,CNAME(Cωϖ (πS#,C#(SC)÷πS#(S)))

（7）检索选修课程包含“程军”老师所授课程之一的学生学号(S#)。

        πS#(SCωϖ πC#(σTEACHER=‘程军’(C)))

（8）检索选修课程号为K1和K5的学生学号(S#)。

        πS#,C#（SC）÷πC#(σC#=’k1’∨ C#=’k5’(C))

（9）检索选修全部课程的学生姓名(SNAME)。

       πSNAME（Sωϖ (πS#,C#（SC）÷πC#(C))）

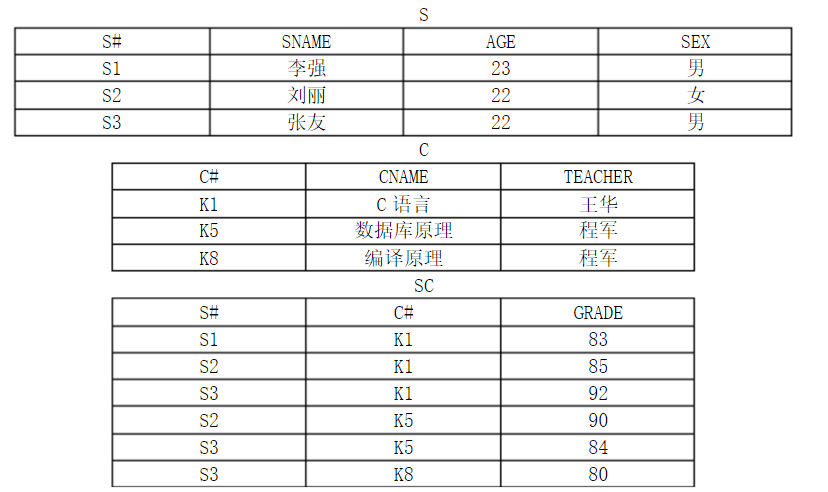
（10）检索选修课程包含学号为2的学生所修课程的学生学号(S#)。

         πS#,C#（SC）÷πC#(σS#=’2’(SC))

（11）检索选修课程名为“C语言”的学生学号(S#)和姓名(SNAME)。

        πS#,SNAME(Sωϖ (πS#(SCωϖ σCNAME=‘C语言’(C))))

 2、设有如下所示的关系：学生表S(S#,SNAME,AGE,SEX)、课程表C(C#,CNAME,TEACHER)和学生选课表SC(S#,C#,GRADE)，其中：S#为学号，SNAME为姓名，AGE为年龄，SEX为性别，C#为课程号，CNAME为课程名，GRADE为成绩, TEACHER为教师。



试用关系代数表达式表示下列查询语句：   
（1）检索LIU老师所授课程的课程号和课程名；

       πC#,CNAME(σTEACHER=‘LIU’(C)) 

（2）检索年龄大于23岁的男学生的学号和姓名；

      πS#,SNAME(σAGE>23∧SEX=‘男’(S)) 

（3）检索学号为S3学生所学课程的课程名与任课教师名；

      πCNAME,TEACHER(σS#=‘S3’(SCωϖ C)) 

（4）检索至少选修LIU老师所授课程中一门课的女学生姓名；

      πSNAME(σSEX=’女’∧TEACHER=‘LIU’(Sωϖ SCωϖ C)) 

（5）检索WANG同学不学的课程的课程号；

      πC#(C)-πC#(σSNAME=‘WANG’(Sωϖ SC))  
（6）检索既选修了LIU老师的课程、又选修了LI老师课程的学生学号；

     π1(σ1=4∧2≠5(SC×SC)) 

（7）全部学生都选修的课程的课程号与课程名；

    πC#,CNAME(Cωϖ (πS#,C#(SC)÷πS#(S)))     
（8）检索选修课程包含LIU老师所授全部课程的学生学号。

      πS#,C#（SC）÷πC#(σTEACHER=‘LIU’(C))