数据库系统原理与设计

（第3版）

学而时习之，不亦乐乎。

数据库系统原理与设计

（第3版）

第3章　SQL查询语言

目　录

3.4　嵌套子查询

- 在SQL查询中，一个SELECT-FROM-WHERE查询语句称为一个**查询块**
- 将一个查询块嵌入到另一个查询块的WHERE子句或HAVING子句中，称为**嵌套子查询**
- 子查询的结果是**集合**，因此使用子查询是**集合成员的检查**
  - 如判断元组是否属于某个集合，集合的比较运算，以及测试是否为空集等
  - 具体表现在如下几个方面：
    - 元素与集合间的属于关系
    - 集合之间的包含和相等关系
    - 集合的存在关系
    - 元素与集合元素之间的比较关系

3.4　嵌套子查询

- SQL允许多层嵌套子查询，但在子查询中，**不允许使用ORDER BY子句**，该子句仅用于最后结果排序
- 嵌套查询分为**相关子查询**和**非相关子查询**
  - 非相关子查询指子查询的结果不依赖于上层查询
  - 相关子查询指当上层查询的元组发生变化时，其子查询必须重新执行

- 3.4.1　使用IN的子查询
- 3.4.2　使用比较运算符的子查询
- *3.4.3　使用存在量词EXISTS的子查询
- *3.4.4　复杂子查询实例

3.4.1 使用IN的子查询

- [例3.43] 查询选修过课程的学生姓名。
- 本例查询的含义是：
  - 在学生表Student中，将学号出现在成绩表Score中(表明该学生选修过课程)的学生姓名查询出来

```
SELECT studentName
FROM Student
WHERE Student.studentNo IN
        (SELECT Score.studentNo FROM Score)
```

- 在本例中，WHERE子句用于检测元素与集合间的属于关系
  - 其中Student.studentNo为元素，IN为“属于”
  - 嵌套语句“SELECT Score.studentNo FROM Score”的查询结果为选修过课程的所有学生的学号集合
  - 该嵌套SELECT语句称为子查询

3.4.1 使用IN的子查询

■ 该查询属于非相关子查询，其查询过程为：

(1) 从Score表中查询出学生的学号studentNo，构成一个中间结果关系*r*；

(2) 从Student表中取出第一个元组*t*；

(3) 如果元组*t*的studentNo属性的值包含在中间结果关系*r*中（即*t*.studentNo∈*r*），则将元组*t*的studentName属性的值作为最终查询结果关系的一个元组；否则丢弃元组*t*；

(4) 如果Student表中还有元组，则取Student表的下一个元组*t*，并转第(3)步；否则转第(5)步；

(5) 将最终结果关系显示出来。

```
SELECT studentName
FROM Student
WHERE Student.studentNo IN
      (SELECT Score.studentNo FROM Score)
```

3.4.1 使用IN的子查询

■ 该查询的执行过程可以通过图3-28来表示

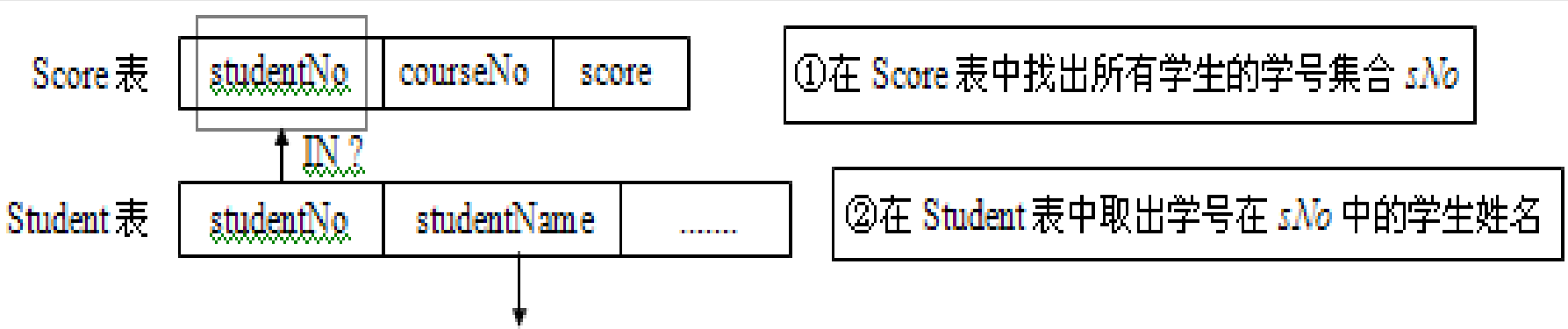


图3-28　二层嵌套图

3.4.1 使用IN的子查询

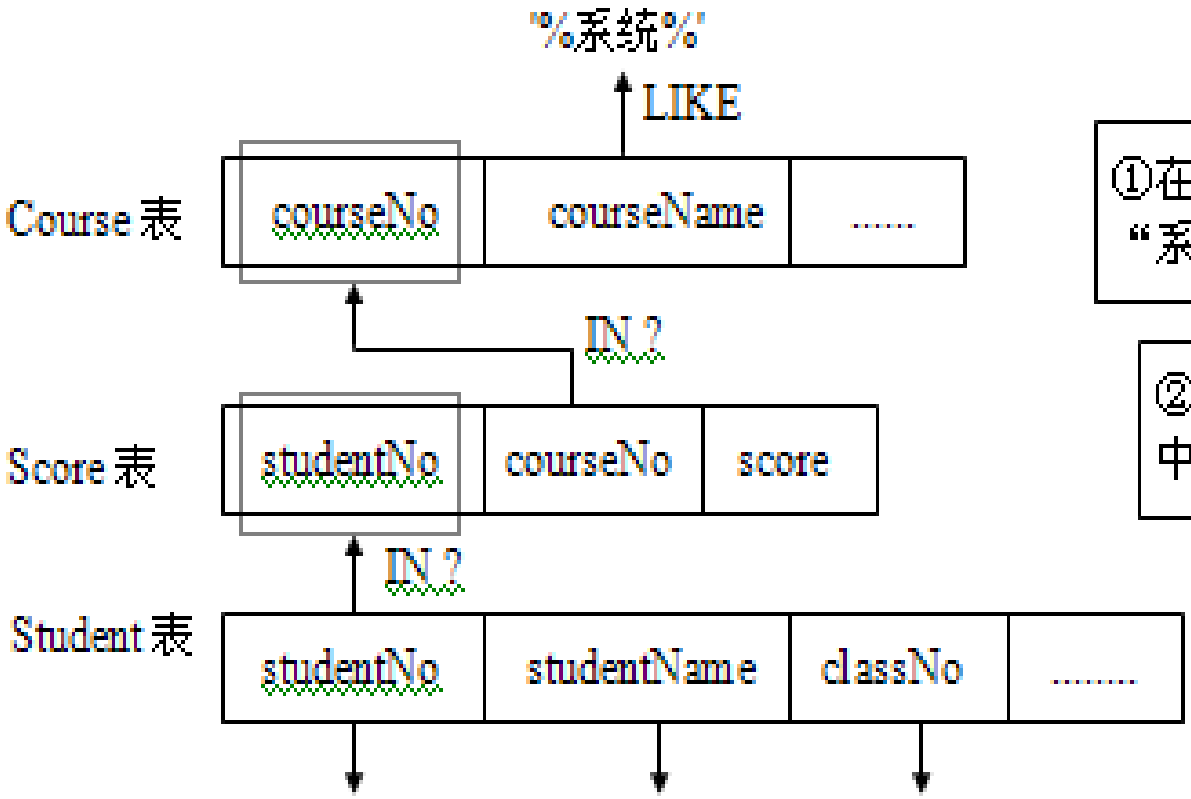
- [例3.44] 查找选修过课程名中包含“系统”的课程的同学学号、姓名和班级编号。

```
SELECT studentNo, studentName, classNo
FROM Student
WHERE studentNo IN
      ( SELECT studentNo FROM Score
        WHERE courseNo IN
              ( SELECT courseNo FROM Course
                WHERE courseName LIKE '%系统%' )
      )
```

学生表与课程表无法直接关联上，需要借助成绩表！

- WHERE子句中的IN可以实现多重嵌套，本例是一个三重嵌套的例子，该查询的执行过程可以通过图3-29来表示

3.4.1 使用IN的子查询



①在 Course 表中找出课程名中包含“系统”的课程号集合 *cNo*

②在 Score 表中找出选修了 *cNo* 中课程的学生学号集合 *sNo*

③在 Student 表中取出学号在 *sNo* 中的学生的学号、姓名和班级编号

图3-29　三重嵌套图

```
SELECT studentNo, studentName, classNo
FROM Student
WHERE studentNo IN
        ( SELECT studentNo FROM Score
          WHERE courseNo IN
                ( SELECT courseNo FROM Course
                  WHERE courseName LIKE '%系统%' )
        )
```

3.4.1 使用IN的子查询

- 该查询也属于非相关子查询
- 使用IN的非相关子查询的查询过程归纳如下：
  - 首先执行最底层的子查询块，将该子查询块的结果作为中间关系；
  - 执行上一层(即外一层)查询块，对于得到的每个元组，判断该元组是否在它的子查询结果中间关系中：
    - 如果在，取出该元组中的相关属性作为最终输出结果(或该查询块的查询结果中间关系)的一个元组
    - 否则舍弃该元组
  - 如果已经执行完最上层查询块，则将最终结果作为一个新关系输出；否则返回第(2)步重复执行

3.4.1 使用IN的子查询

- [例3.44] 查找选修过课程名中包含“系统”的课程的同学学号、姓名和班级编号。
- 等价于

```
SELECT studentNo, studentName, classNo
FROM Student
WHERE studentNo IN
        ( SELECT studentNo
          FROM Score b, Course c
          WHERE b.courseNo=c.courseNo
              AND courseName LIKE '%系统%'
        )
```

3.4.1 使用IN的子查询

- [例3.44] 查找选修过课程名中包含“系统”的课程的同学学号、姓名和班级编号。
- 等价于

```
SELECT DISTINCT studentNo, studentName, classNo
FROM Student a, Score b, Course c
WHERE a.studentNo=b.studentNo
    AND b.courseNo=c.courseNo
    AND courseName LIKE '%系统%'
```

- 注意：并不是每一个IN子查询都可以转化为连接运算来实现！

3.4.1 使用IN的子查询

- [例3.45] 查找同时选修过“计算机原理”和“高等数学”两门课程的同学学号、姓名以及该同学所选修的所有课程的课程名和相应成绩，按学号(升序)、成绩(降序)排序输出。
- 分析：
  - 在SELECT子句中必须包含studentNo、studentName、courseName和score四个属性
  - 学号、姓名在学生表中，课程成绩在成绩表中，课程名在课程表中，在FROM子句中必须包含学生表、课程表和成绩表，分别为这三张表取元组变量*a*、*b*、*c*
  - 学生表、成绩表和课程表需做连接操作，在WHERE子句中必须包含连接条件：
    - *a*.studentNo=*c*.studentNo AND *b*.courseNo=*c*.courseNo

3.4.1 使用IN的子查询

- 要查询同时选修过“计算机原理”和“高等数学”两门课程的同学，在WHERE子句中必须包含如下的选择条件：
  - 对于学生表，其学号必须是选修过“计算机原理”课程的学号，使用子查询：

```
a.studentNo IN
( SELECT studentNo FROM Score
  WHERE courseNo IN ( SELECT courseNo FROM Course
                      WHERE courseName='计算机原理' ) )
```

  - 对于学生表，其学号还必须是选修过“高等数学”课程的学号，使用子查询：

```
a.studentNo IN
( SELECT studentNo FROM Score
  WHERE courseNo IN ( SELECT courseNo FROM Course
                      WHERE courseName='高等数学' ) )
```

  - 这两个子查询必须同时满足，使用AND逻辑运算符

3.4.1 使用IN的子查询

- 本查询语句为：

```
SELECT  a.studentNo, studentName, courseName, score
FROM Student a, Course b, Score c
WHERE  a.studentNo=c.studentNo AND b.courseNo=c.courseNo
    AND   a.studentNo IN
          ( SELECT studentNo FROM Score
            WHERE courseNo IN
                  ( SELECT courseNo FROM Course
                    WHERE courseName='计算机原理' ) )
    AND   a.studentNo IN
          ( SELECT studentNo FROM Score
            WHERE courseNo IN
                  ( SELECT courseNo FROM Course
                    WHERE courseName='高等数学' ) )
ORDER BY a.studentNo, score DESC
```

3.4.1 使用IN的子查询

- 本查询语句为：

```
SELECT  a.studentNo, studentName
FROM Student a
WHERE
        a.studentNo IN
            ( SELECT studentNo FROM Score
              WHERE courseNo IN
                    ( SELECT courseNo FROM Course
                      WHERE courseName='计算机原理' ) )
    AND  a.studentNo IN
            ( SELECT studentNo FROM Score
              WHERE courseNo IN
                    ( SELECT courseNo FROM Course
                      WHERE courseName='高等数学' ) )
ORDER BY a.studentNo, score DESC
```

查找出同时选修过“计算机原理”和“高等数学”两门课程的同学——回顾除运算！

3.4.1 使用IN的子查询

- 该查询也可以表示为如下形式：

```
SELECT  a.studentNo, studentName, courseName, score
FROM Student a, Course b, Score c
WHERE  a.studentNo=c.studentNo AND b.courseNo=c.courseNo
    AND   a.studentNo IN
          ( SELECT studentNo FROM Score x, Course y
            WHERE x.courseNo=y.courseNo
                AND courseName='计算机原理' )
    AND   a.studentNo IN
          ( SELECT studentNo FROM Score x, Course y
            WHERE x.courseNo=y.courseNo
                AND courseName='高等数学' )
ORDER BY a.studentNo, score DESC
```

3.4.1 使用IN的子查询

- 该查询也可以表示为如下形式：

```
SELECT  a.studentNo, studentName
FROM Student a
WHERE
        a.studentNo IN
            ( SELECT studentNo FROM Sco
              WHERE x.courseNo=y.courseN
                  AND courseName='计算机
    AND   a.studentNo IN
            ( SELECT studentNo FROM Sco
              WHERE x.courseNo=y.courseNo
                  AND courseName='高等数学
ORDER BY a.studentNo, score DESC
```

查找出同时选修过“计算机原理”和“高等数学”两门课程的同学——回顾除运算！

studentNo	studentName	courseName	score
1500001	李小勇	大学语文	98.0
1500001	李小勇	高等数学	86.0
1500001	李小勇	会计学原理	86.0
1500001	李小勇	操作系统	82.0
1500001	李小勇	体育	82.0
1500001	李小勇	大学英语	82.0
1500001	李小勇	数据库系统原理	77.0
1500001	李小勇	C语言程序设计	77.0
1500001	李小勇	数据结构	77.0
1500001	李小勇	计算机原理	76.0
1500001	李小勇	高等数学	56.0
1500004	张可立	C语言程序设计	88.0
1500004	张可立	操作系统	80.0
1500004	张可立	计算机原理	72.0
1500004	张可立	数据结构	71.0
1500004	张可立	大学英语	70.0
1500004	张可立	大学语文	70.0
1500004	张可立	体育	68.0
1500004	张可立	高等数学	58.0

3.4.1 使用IN的子查询

- [例3.46] 查找同时选修过“计算机原理”和“高等数学”两门课程的同学学号、姓名以及所选修的这两门课程的课程名和相应成绩，按学号(升序)、成绩(降序)排序输出
- 分析：
  - 只查询该同学所选修的这两门课程的课程名和相应成绩，在WHERE子句中还必须包含选择条件：课程名称必须是“计算机原理”或“高等数学”，即
    - courseName='高等数学' OR courseName='计算机原理'

3.4.1 使用IN的子查询

- 本查询语句为：

```
SELECT a.studentNo, studentName, courseName, score
FROM Student a, Course b, Score c
WHERE a.studentNo=c.studentNo AND b.courseNo=c.courseNo
    AND a.studentNo IN
        ( SELECT studentNo FROM Score x, Course y
          WHERE x.courseNo=y.courseNo AND courseName='计算机原理' )
    AND a.studentNo IN
        ( SELECT studentNo FROM Score x, Course y
          WHERE x.courseNo=y.courseNo AND courseName='高等数学' )
    AND ( b.courseName='高等数学' OR b.courseName='计算机原理' )
ORDER BY a.studentNo, score DESC
```

- 请将例3.45、例3.46的查询要求及查询语句的实现形式，与例3.34、例3.37进行比较。

3.4.1 使用IN的子查询

- 本查询语句为：

```
SELECT a.studentNo,  studentName,
FROM  Student a,
WHERE
      a.studentNo IN
            ( SELECT studentNo FROM Score x, Course y
              WHERE x.courseNo=y.courseNo AND courseName='计算机原理' )
   AND  a.studentNo IN
            ( SELECT studentNo FROM Score x, Course y
              WHERE x.courseNo=y.courseNo AND courseName='高等数学' )
ORDER BY a.studentNo, score DESC
```

查找出同时选修过“计算机原理”和“高等数学”两门课程的同学——回顾除运算！

- 请将例3.45、例3.46的查询要求及查询语句的实现形式，与例3.34、例3.37进行比较。

3.4.2 使用比较运算符的子查询

- 元素与集合元素之间还存在更为复杂的关系，如比较关系，常用到谓词ANY(或SOME)和ALL
  - ANY表示子查询结果中的某个值
  - ALL表示子查询结果中的所有值

(100,200,300)

比较运算符	含义	比较运算符	含义
=ANY	等于子查询结果中的某个值	=ALL	等于子查询结果中的所有值
>=ANY	大于等于子查询结果中的某个值	>=ALL	大于等于子查询结果中的所有值
<=ANY	小于等于子查询结果中的某个值	<=ALL	小于等于子查询结果中的所有值
>ANY	大于子查询结果中的某个值	>ALL	大于子查询结果中的所有值
<ANY	小于子查询结果中的某个值	<ALL	小于子查询结果中的所有值
<>ANY	不等于子查询结果中的某个值	<>ALL	不等于子查询结果中的所有值
!=ANY	不等于子查询结果中的某个值	!=ALL	不等于子查询结果中的所有值

3.4.2 使用比较运算符的子查询

- 注意：
  - 如果子查询中的结果关系仅包含一个元组，则可将ALL和ANY去掉，直接使用比较运算符
  - ANY也可以用SOME替代

3.4.2 使用比较运算符的子查询

■ [例3.47] 查询所选修课程的成绩大于所有002号课程成绩的同学学号及相应课程的课程号和成绩。

```
SELECT studentNo, courseNo, score
FROM Score
WHERE score>ALL
        ( SELECT score
          FROM Score
          WHERE courseNo='002' )
```

studentNo	courseNo	score
1500001	001	98.0
1500001	004	86.0
1500001	010	86.0
1500003	010	90.0
1500004	005	88.0
1500005	004	87.0
1500005	007	90.0
1500005	008	87.0
1500005	009	90.0

score
82.0
38.0
58.0
68.0
80.0
46.0
60.0
70.0

3.4.2 使用比较运算符的子查询

- [例3.48] 查询成绩最高分的学生的学号、课程号和相应成绩

```
SELECT studentNo, courseNo, score
FROM Score
WHERE score=( SELECT max(score)
              FROM Score )
```

- 聚合函数可直接用在HAVING子句中(如例3.35)
- 聚合函数也可
- 聚合函数不可

```
SELECT *
FROM Score
WHERE score=
```

SQLQuery1.s... (SA (55))*

```
select * from score where
score >=all(select MAX(score) from score)
```

结果 | 消息

	studentNo	courseNo	termNo	score
1	1500001	001	151	98.0
2	1600002	001	161	98.0
3	1600002	003	162	98.0

3.4.2 使用比较运算符的子查询

- [例3.49] 查询年龄小于“计算机科学与技术16-01班”某个同学年龄的所有同学的学号、姓名和年龄。

```
SELECT studentNo, studentName, year(getdate())-year(birthday) AS age
FROM Student
WHERE birthday>ANY
        ( SELECT birthday
          FROM Student a, Class b
          WHERE className='计算机科学与技术16-01班'
              AND a.classNo=b.classNo )
```

- 本查询执行过程是：
  - 首先执行子查询，找出“计算机科学与技术16-01班”同学的出生日期集合
  - 然后在Student表中将出生日期大于该集合中某个同学出生日期的所有同学查找出来（说明：出生日期大于即年龄小于）。
- 在比较运算符中，=ANY 等价于 IN，!=ALL 等价于 NOT IN

小结

- 使用IN的子查询，使用比较运算符的子查询
- 执行顺序？

*3.4.3 使用存在量词EXISTS的子查询

- SQL查询提供量词运算
- 量词有两种：
  - 一是存在量词
  - 二是全称量词
  - 在离散数学中，全称量词可用存在量词替代
  - SQL仅提供存在量词的运算，使用谓词EXISTS表示
  - 全称量词转化通过NOT EXISTS谓词来实现

exists(有元组吗？)

- WHERE子句中的谓词EXISTS用来判断其后的子查询的结果集合中是否存在元素
- 谓词EXISTS大量用于相关子查询中

*3.4.3 使用存在量词EXISTS的子查询

- [例3.50] 查询选修了“计算机原理”课程的同学姓名、所在班级编号。
- 该查询可直接通过连接运算实现，也可以通过IN子查询来实现。还可以通过存在量词实现：

```
SELECT studentName, classNo
FROM Student x
WHERE EXISTS
        ( SELECT * FROM Score a, Course b
          WHERE a.courseNo=b.courseNo
            AND a.studentNo=x.studentNo
            AND courseName='计算机原理' )
```

*3.4.3 使用存在量词EXISTS的子查询

- 本查询涉及Student、Score和Course三个关系，属于相关子查询，查询过程如下：

  (1) 首先取Student表的第一个元组*x*，并取其学号*x*.studentNo；

  (2) 执行子查询，该子查询对表Score和Course进行连接，并选择其学号为*x*.studentNo，其课程名为“计算机原理”的元组；

  (3) 如果子查询中可以得到结果(即存在元组)，则将Student表中元组*x*的学生姓名和所在班级编号组成一个新元组放在结果集合中；否则(即不存在元组)，直接丢弃元组*x*；

  (4) 如果Student表中还有元组，则取Student表的下一个元组*x*，并取其学号*x*.studentNo，转第(2)步；否则转第(5)步；

  (5) 将结果集合中的元组作为一个新关系输出

- 子查询的目标列通常是*
  - 存在量词EXISTS只判断其后的子查
    元素，没有必要给出查询结果的列

```
SELECT studentName, classNo
FROM Student x
WHERE EXISTS
    ( SELECT * FROM Score a, Course b
      WHERE a.courseNo=b.courseNo
        AND a.studentNo=x.studentNo
        AND courseName='计算机原理' )
```

*3.4.3 使用存在量词EXISTS的子查询

- **相关子查询**在SQL中属于复杂的查询，其**子查询的查询条件**依赖于**外层查询的元组值**
  - 当外层查询的元组值发生变化时，其子查询要重新依据新的条件进行查询
  - 使用**EXISTS**的相关子查询处理过程是：
    (1) 首先取外层查询的第一个元组；
    (2) 依据该元组的值，执行子查询；
    (3) 如果子查询的结果非空(EXISTS量词返回真值)，将外层查询的该元组放入到结果集中；否则(EXISTS量词返回假值)，舍弃外层查询的该元组；
    (4) 取外层查询的下一个元组，返回第(2)步重复上述过程，直到外层查询所有的元组处理完毕；
    (5) 将结果集合中的元组作为一个新关系输出
- 本例可直接使用**连接**或**IN**运算来实现

*3.4.3 使用存在量词EXISTS的子查询

- [例3.51] 查询选修了所有课程的学生姓名。
- 分析：
  - 本查询要使用全称量词，含义是：
    - 选择这样的学生，任意一门课程他都选修了
    - 设谓词*P*(*x*, *c*)表示学生*x*选修了课程*c*，本查询可表示为：
      - ✓选择这样的学生*x*，使 $(\forall c)P(x,c)$ 。
  - SQL中没有全称量词，使用存在量词和取非运算来实现，转换公式如下：

$$(\forall c)P(x,c) \Leftrightarrow \neg(\exists c(\neg P(x,c)))$$

    - 谓词 $\neg P(x,c)$表示学生*x*没有选修课程*c*。
    - 根据该转换公式，可将上述查询描述为：
      - ✓查询这样的学生*x*，不存在他没有选修的课程*c*

*3.4.3 使用存在量词EXISTS的子查询

S	SN	S	C	C	CN
1	Li	1	01	01	DB
2	W	1	02	02	CT
		1	03	03	CA
		2	01		
		2	03		

```
SELECT studentName
FROM Student x
WHERE NOT EXISTS
        ( SELECT * FROM Course c
          WHERE NOT EXISTS
                  --判断学生x.studentNo没有选修课程c.courseNo
                ( SELECT * FROM Score
                  WHERE studentNo=x.studentNo
                        AND courseNo=c.courseNo )
        )
```

*3.4.3 使用存在量词EXISTS的子查询

- [例3.52] 查询至少选修了学号为1600002学生所选修的所有课程的学生姓名。
- 分析：
  - 本查询的含义是选择这样的学生，凡是1600002学生选修了的课程，他也选修了。
  - 本例要使用蕴涵量词，SQL不提供蕴涵量词，可通过使用存在量词和取非运算来实现，转换公式如下：
    - 用谓词*R*(*c*)表示1600002学生选修了*c*课程
    - 用谓词*P*(*x*, *c*)表示学生*x*选修了*c*课程
    - 本查询可表示为：选择这样的学生*x*，使 $(\forall c)(R(c) \rightarrow P(x,c))$

*3.4.3 使用存在量词EXISTS的子查询

[例3.52] 查询至少选修了学号为1600002学生所选修的所有课程的学生姓名。

```
SELECT studentName
FROM Student x
WHERE NOT EXISTS
    ( SELECT * FROM Score y      // 不能用Course表
      WHERE studentNo='1600002'
                --查询学生'1600002'所选修课程的情况
        AND NOT EXISTS
                --判断学生x.studentNo没有选修课程y.courseNo
          ( SELECT * FROM Score
            WHERE studentNo=x.studentNo
              AND courseNo=y.courseNo )
    )
```

[例3.53] 查询至少选修了学号为“0700002”学生所选修的所有课程的学生学号、姓名以及该学生所选修所有课程的课程名和成绩，按学生姓名、课程名排序输出。

*3.4.3 使用存在量词EXISTS的子查询

- [例3.53] 查询至少选修了学号为1600002学生所选修的所有课程的学生学号、姓名以及该学生所选修所有课程的课程名和成绩，按学生姓名、课程名排序输出。
- 分析：
  - 本查询需输出选课学生的学号、姓名以及所选修所有课程的课程名和成绩，在SELECT子句中必须包含学号、姓名、课程名和成绩
  - 学号和姓名在学生表中，课程名在课程表中，成绩在成绩表中，在FROM子句中必须包含学生表、课程表和成绩表，分别取元组变量*x*、*y*、*z*。
  - 学生表、课程表和成绩表需做连接操作，在WHERE子句中必须包含连接条件：
    - *x*.studentNo=z.studentNo AND *y*.courseNo=z.courseNo

*3.4.3 使用存在量词EXISTS的子查询

- 查询至少选修了学号为1600002的学生所选修的所有课程，必须首先查询学号为1600002的学生所选修的所有课程情况，使用子查询：

```
SELECT * FROM Score b
WHERE studentNo='1600002'
```

- 对学生表中的某个同学x.studentNo的选课记录集合，必须包含学号为1600002的学生的选课记录集合，即学号为1600002学生选修的课程，x.studentNo同学也要选修，在子查询中还必须包含一个条件表示这种包含关系：

```
SELECT * FROM Score b
WHERE studentNo='1600002'
    AND EXISTS    --表示x.studentNo同学也选修了学号为1600002学生选修的课程
        ( SELECT * FROM Score
          WHERE studentNo=x.studentNo
              AND courseNo=b.courseNo )
```

- 对上述查询使用双重否定：不存在1600002学生选修的某门课程，而x.studentNo学生没有选修

*3.4.3 使用存在量词EXISTS的子查询

```
SELECT x.studentNo, studentName, courseName, score
FROM Student x, Course y, Score z
WHERE x.studentNo=z.studentNo AND y.courseNo=z.courseNo
    AND NOT EXISTS
        ( SELECT * FROM Score b
          WHERE studentNo='1600002'   --查询学生'1600002'所选修课程的情况
              AND NOT EXISTS    --判断学生x.studentNo没有选修课程b.courseNo
                  ( SELECT * FROM Score
                    WHERE studentNo=x.studentNo AND courseNo=b.courseNo )
        )
ORDER BY studentName, courseName              // 排序输出
```

*3.4.3 使用存在量词

```
SELECT x.studentNo, studentName, cour
FROM Student x, Course y, Score z
WHERE x.studentNo=z.studentNo AND y.
    AND NOT EXISTS
        ( SELECT * FROM Score b
          WHERE studentNo='1600002'
              AND NOT EXISTS    --判断
```

studentNo	studentName	courseName
1600002	刘晨	大学语文
1600002	刘晨	体育
1600002	刘晨	大学英语
1600002	刘晨	高等数学
1600002	刘晨	C语言程序设计
1600002	刘晨	会计学原理

图 3-30　1600002 学生选修的所有课程

	studentNo	studentName	courseName	score
1	1500001	李小勇	大学语文	98.0
2	1500001	李小勇	体育	82.0
3	1500001	李小勇	大学英语	82.0
4	1500001	李小勇	高等数学	56.0
5	1500001	李小勇	高等数学	86.0
6	1500001	李小勇	C语言程序设计	77.0
7	1500001	李小勇	计算机原理	76.0
8	1500001	李小勇	数据结构	77.0
9	1500001	李小勇	操作系统	82.0
10	1500001	李小勇	数据库系统原理	77.0
11	1500001	李小勇	会计学原理	86.0
12	1500005	王红	大学语文	79.0
13	1500005	王红	体育	80.0
14	1500005	王红	大学英语	69.0
15	1500005	王红	高等数学	87.0
16	1500005	王红	C语言程序设计	77.0
17	1500005	王红	计算机原理	69.0
18	1500005	王红	数据结构	90.0
19	1500005	王红	操作系统	87.0
20	1500005	王红	数据库系统原理	90.0
21	1500005	王红	会计学原理	69.0
22	1500005	王红	中级财务会计	68.0

*3.4.3 使用存在量词EXISTS的子查询

```
SELECT x.studentNo, studentName, courseName, score
FROM Student x, Course y, Score z
WHERE x.studentNo=z.studentNo AND y.courseNo=z.courseNo
    AND NOT EXISTS
        ( SELECT * FROM Score b
          WHERE studentNo='1600002'   --查询学生'1600002'所选修课程的情况
              AND NOT EXISTS    --判断学生x.studentNo没有选修课程b.courseNo
                  ( SELECT * FROM Score
                    WHERE studentNo=x.studentNo AND courseNo=b.courseNo )
        )
ORDER BY studentName, courseName                    // 排序输出
```

*3.4.3 使用存在量词EXISTS的子查询

查找出选修了学号为1600002的学生所选修的所有课程的同学 ——回顾除运算！

```
SELECT  x.studentNo,  studentName,
FROM  Student x,
WHERE
        NOT EXISTS
          ( SELECT * FROM Score b
            WHERE studentNo='1600002'   --查询学生'1600002'所选修课程的情况
                AND NOT EXISTS    --判断学生x.studentNo没有选修课程b.courseNo
                    ( SELECT * FROM Score
                      WHERE studentNo=x.studentNo AND courseNo=b.courseNo )
          )
ORDER BY studentName, courseName                    //  排序输出
```

*3.4.3 使用存在量词EXISTS的子查询

- [例3.53'] 查询至少选修了学号为1600002学生所选修的所有课程的学生学号、姓名以及该学生所选修的1600002学生选修过的所有课程的课程名和成绩。

```
SELECT x.studentNo, studentName, courseName, score
FROM Student x, Course y, Score z
WHERE x.studentNo=z.studentNo AND y.courseNo=z.courseNo
    AND NOT EXISTS
        ( SELECT * FROM Score b
          WHERE studentNo='1600002'   --查询学生'1600002'所选修课程的情况
              AND NOT EXISTS    --判断学生x.studentNo没有选修课程b.courseNo
                  ( SELECT * FROM Score
                    WHERE studentNo=x.studentNo AND courseNo=b.courseNo )
        )
  AND y.courseNo IN
        ( SELECT courseNo FROM Score WHERE studentNo='1600002')
```

*3.4.4　复杂子查询实例

- [例3.54] 查询至少选修了28个学分的同学的学号、姓名以及所选修各门课程的课程名、成绩和学分，按学号排序输出。
- 要求如下：①对于所选修的课程，如果成绩不及格，则不能获得该课程的学分；②如果一个学生选修同一门课程多次，则选取最高成绩输出。
- 分析：

①本例查询结果列是学号、姓名、课程名、成绩和学分，在SELECT子句中必须包含这些属性

②一方面，学号、姓名在学生表中，课程名、学分在课程表中，因此FROM子句中必须包含这2个表，分别取元组变量*a*、*b*；另一方面，成绩在成绩表中，但是如果一个学生选修同一门课程多次时需要选取最高成绩输出，因此关于成绩属性，在FROM子句中需要用到一个如下的查询表*c*：

*3.4.4　复杂子查询实例

- [例3.54] 查询至少选修了28个学分的同学的学号、姓名以及所选修各门课程的课程名、成绩和学分，按学号排序输出。
- 要求如下：①对于所选修的课程，如果成绩不及格，则不能获得该课程的学分；②如果一个学生选修同一门课程多次，则选取最高成
- 分析：

①本例查询结
SELECT子

```
( SELECT studentNo, courseNo, max(score) score
  FROM Score
  WHILE score>=60    -- 只有及格才能获得学分
  GROUP BY studentNo, courseNo ) AS c
```

②一方面，学号、姓名在学生表中，课程名、学分在课程表中，因此FROM子句中必须包含这2个表，分别取元组变量*a*、*b*；另一方面，成绩在成绩表中，但是如果一个学生选修同一门课程多次时需要选取最高成绩输出，因此关于成绩属性，在FROM子句中需要用到一个如下的查询表*c*：

*3.4.4　复杂子查询实例

④结果关系中的学生必须是选修了28个以上(含28)学分的同学，因此需要构建一个子查询Q，用于检索满足该条件的学号。

- 由于学分在课程表中，选课记录在查询表c中，所以子查询Q涉及这2个表的连接操作，连接条件是课程号相同；
- 子查询Q还要求所选修课程的总学分在28分以上，需要使用分组聚合运算，其分组属性为studentNo，分组选择条件是sum(creditHour)>=28

*3.4.4　复杂子查询实例

④结果关系中的学生必须是选修了28个以上(含28)学分的同学，因此需要构建一个子查询Q，用于检索满足该条件的学号。

- 子查询Q语句为：

```
SELECT studentNo   -- 子查询Q
FROM Course x, ( SELECT studentNo, courseNo, max(score) score
                 FROM Score
                 WHILE score>=60     -- 只有及格才能获得学分
                 GROUP BY studentNo, courseNo ) AS y
WHERE y.courseNo=x.courseNo
GROUP BY studentNo
HAVING sum(creditHour)>=28
```

- 本例将分组聚合用于子查询Q中，该子查询Q的含义是：查询选修了28个以上(含28)学分的同学学号

*3.4.4　复杂子查询实例

⑤在结果关系中的学号必须是子查询Q中的学号，在WHERE子句中除了包含学生表*a*、课程表*b*、查询表*c*的连接条件外，还要有一个选择条件：学号必须是子查询Q结果集合中的学号.

WHERE *a*.studentNo=*c*.studentNo AND *c*.courseNo=*b*.courseNo
AND *a*.studentNo IN 子查询Q

⑥本例要求按学号排序输出，需使用排序子句

[例3.55] 查询至少选修了5门课程且课程平均分最高的同学的学号和课程平均分。如果一个学生选修同一门课程多次，则选取最高成绩。

*3.4.4 复杂子查询实例

```
SELECT a.studentNo, studentName, courseName, score, creditHour
FROM Student a, Course b,
        ( SELECT studentNo, courseNo, max(score) score
          FROM Score
          WHERE score>=60    -- 仅列示已经获得学分(即及格了)的课程
          GROUP BY studentNo, courseNo ) AS c      -- 查询表c
WHERE a.studentNo=c.studentNo AND c.courseNo=b.courseNo
    AND a.studentNo IN
        ( SELECT studentNo     -- 子查询Q
          FROM Course x, ( SELECT studentNo, courseNo, max(score) score
                           FROM Score
                           WHERE score>=60    -- 只有及格才能获得学分
                           GROUP BY studentNo, courseNo ) AS y
          WHERE y.courseNo=x.courseNo
          GROUP BY studentNo
          HAVING sum(creditHour)>=28 )
ORDER BY a.studentNo
```

*3.4.4　复杂子查询实例

- [例3.55] 查询至少选修了5门课程且课程平均分最高的同学的学号和课程平均分。如果一个学生选修同一门课程多次，则选取最高成绩。
- 分析：

① 查询同学的学号和课程平均分，使用求平均值的聚合函数，在SELECT子句中包含学号 studentNo和课程平均分avg(score)

② 本例只要使用成绩表，但如果一个学生选修同一门课程多次则需选取最高成绩，因此在FROM子句中需用到如下的查询表*a*

```
( SELECT studentNo, courseNo, max(score) score
  FROM Score
  GROUP BY studentNo, courseNo ) AS a     -- 查询表a
```

*3.4.4 复杂子查询实例

③ 查询至少选修了5门课程且课程平均分最高的同学的学号，使用分组运算，分组属性为学号studentNo

- 必须对分组后的结果进行选择运算，选择至少选修了5门课程且课程平均分最高的同学的学号，使用HAVING子句：
- HAVING count(*)>=5　-- 至少选修了5门课程
- HAVING子句中的第二个条件是课程平均分最高，构造如下：
  - 首先，基于查询表*a*构建一个子查询*Q1*，查找出至少选修了5门课程的同学的学号和课程平均分，显然子查询*Q1*的结果满足第一个HAVING条件。SQL语句如下，其中查询表*a*已改名为*b*。

```
( SELECT studentNo, avg(score)  avgScore      -- 子查询Q1
  FROM  ( SELECT studentNo, courseNo, max(score) score
          FROM Score
          GROUP BY studentNo, courseNo ) AS b    -- 查询表b
  GROUP BY studentNo
  HAVING count(*)>=5 ) AS x      --将子查询Q1作为查询表x
```

*3.4.4　复杂子查询实例

- 然后，基于查询表*x*再构造一个子查询*Q2*，查询最高的平均分：
  - 子查询*Q2*：

```
SELECT max(avgScore)    -- 子查询Q2
FROM ( SELECT studentNo, avg(score)  avgScore     -- 子查询Q1
          FROM  ( SELECT studentNo, courseNo, max(score) score
                     FROM Score
                     GROUP BY studentNo, courseNo ) AS b
          GROUP BY studentNo
          HAVING count(*)>=5 ) AS x      --将子查询Q1作为查询表x
```

- 最后，HAVING子句中的第二个条件是：平均分等于子查询*Q2*中查询出来的最高的平均分：

```
AND avg(score)=子查询Q2
```

*3.4.4　复杂子查询实例

④ 该查询语句为：

```
SELECT studentNo, avg(score)  avgScore
FROM  ( SELECT studentNo, courseNo, max(score) score
          FROM Score
          GROUP BY studentNo, courseNo ) AS a
GROUP BY studentNo
HAVING  count(*)>=5
 AND  avg(score)=
       ( SELECT max(avgScore)    -- 子查询Q2
         FROM ( SELECT studentNo, avg(score)  avgScore    -- 子查询Q1
                  FROM  ( SELECT studentNo, courseNo, max(score) score
                            FROM Score
                            GROUP BY studentNo, courseNo ) AS b
                  GROUP BY studentNo
                  HAVING count(*)>=5 ) AS x
       )
```

注意：在查询表中，如果查询的列是表达式，可以给该表达式取一个别名，这样在SELECT语句中就可直接使用该别名。

在本例的查询表*a*、*b*中，将表达式max(score)取别名score；

在本例的查询表*x*中，将表达式avg(score)取别名avgScore。

```
            GROUP BY studentNo, courseNo ) AS a
GROUP BY studentNo
HAVING  count(*)>=5
 AND  avg(score)=
      ( SELECT max(avgScore)    -- 子查询Q2
        FROM ( SELECT studentNo, avg(score)  avgScore    -- 子查询Q1
               FROM  ( SELECT studentNo, courseNo, max(score) score
                       FROM Score
                       GROUP BY studentNo, courseNo ) AS b
               GROUP BY studentNo
               HAVING count(*)>=5 ) AS x
      )
```

*3.4.4　复杂子查询实例

- [例3.56] 查询选修了所有4学分课程(即学分为4的课程)的同学的学号、姓名以及所选修4学分课程的课程名和成绩。
- 分析：

①与例3.52、例3.53类似，本例也要使用双重否定。在第一重否定中查询4学分的课程，在第二重否定中查询某学生选修某门4学分的课程

②该查询表达的含义是：查询这样的学生，不存在某门4学分的课程他没有选修

*3.4.4　复杂子查询实例

③该查询语句为：

```
SELECT a.studentNo, studentName,   courseName, score
FROM Student a, Course b, Score c
WHERE a.studentNo=c.studentNo AND b.courseNo=c.courseNo
    AND NOT EXISTS
        ( SELECT * FROM Course x
          WHERE creditHour=4    --查询4学分课程的情况
              AND NOT EXISTS    --判断学生a.studentNo没有选修课程x.courseNo
                  ( SELECT * FROM Score
                    WHERE studentNo=a.studentNo AND courseNo=x.courseNo
                  )
        )
    AND creditHour=4
        --只显示满足上述要求的学生所选修4学分课程的课程名和成绩
```

目　录

3.5 集合运算

- SQL支持集合运算
- SELECT语句查询的结果是集合
- 传统的集合操作主要包括并UNION、交INTERSECT、差EXCEPT运算
- 在执行集合运算时要求参与运算的查询结果的列数一样，其对应列的数据类型必须一致

3.5 集合运算

- [例3.57] 查询“信息管理学院”1999年出生的同学的学号、出生日期、班级名称和所属学院以及“会计学院”1998年出生的同学的学号、出生日期、班级名称和所属学院。

```
SELECT studentNo, birthday, className, institute
FROM Student a, Class b
WHERE a.classNo=b.classNo AND year(birthday)=1999
      AND institute='信息管理学院'
UNION
SELECT studentNo, birthday, className, institute
FROM Student a, Class b
WHERE a.classNo=b.classNo AND year(birthday)=1998
      AND institute='会计学院'
```

3.5 集合运算

- 该查询实际上是查询“信息管理学院”1999年出生的或“会计学院”1998年出生的同学的学号、出生日期、班级名称和所属学院，上述SQL语句可以改写为：

```
SELECT studentNo, birthday, className, institute
FROM Student a, Class b
WHERE a.classNo=b.classNo
  AND ( year(birthday)=1999 AND institute='信息管理学院'
        OR year(birthday)=1998 AND institute='会计学院' )
ORDER BY institute
```

3.5 集合运算

- [例3.58] 查询同时选修了“001”号和“005”号课程的同学的学号和姓名

```
SELECT a.studentNo, studentName
FROM Student a, Score b
WHERE a.studentNo=b.studentNo AND courseNo='001'
INTERSECT
SELECT a.studentNo, studentName
FROM Student a, Score b
WHERE a.studentNo=b.studentNo AND courseNo='005'
```

3.5 集合运算

■ [例3.58] 查询同时选修了“001”号和“005”号课程的同学的学号和姓名

● 本例也可用下面的SQL语句实现

```
SELECT a.studentNo, studentName
FROM Student a, Score b
WHERE a.studentNo=b.studentNo AND courseNo='001'
    AND a.studentNo IN (
        SELECT studentNo FROM Score WHERE courseNo='005' )
```

3.5 集合运算

■ [例3.59] 查询没有选修“计算机原理”课程的同学的学号和姓名。

```
SELECT studentNo, studentName
FROM Student
EXCEPT
SELECT DISTINCT a.studentNo, studentName
FROM Student a, Score b, Course c
WHERE a.studentNo=b.studentNo
    AND  b.courseNo=c.courseNo
    AND  courseName='计算机原理'
```

3.5 集合运算

- [例3.59] 查询没有选修“计算机原理”课程的同学的学号和姓名。

- 本例也可用下面的SQL语句实现

```
SELECT studentNo, studentName
FROM Student
WHERE studentNo NOT IN
      ( SELECT studentNo
        FROM Score x, Course y
        WHERE x.courseNo=y.courseNo
            AND  courseName='计算机原理' )
```

目　录

3.6　SQL查询一般格式

■ SELECT共有6个子句，其中SELECT和FROM是必须的，其它是可选项，必须严格按照如下顺序排列：

SELECT [ALL | DISTINCT] <*目标列表达式*> [AS] [<*别名*>]
　　[, <*目标列表达式*> [AS] [<*别名*>] ...]
FROM {<*表名*> | <*视图名*> | <*查询表*>} [AS] [<*别名*>]
　　[, {<*表名*> | <*视图名*> | <*查询表*>} [AS] [<*别名*>] ...]
[WHERE <*条件表达式*>]
[GROUP BY <*列名1*> [, <*列名2*> ...]
　[HAVING <*条件表达式*>]]
[ORDER BY <*列名表达式*> [ASC | DESC]
　　[, <*列名表达式*> [ASC | DESC] ...]]

3.6　SQL查询一般格式

■其中：

(1) <*目标列表达式*>可以是下面的可选格式：

- [{<*表名*> | <*别名*>}.]*
- [{<*表名*> | <*别名*>}.]<*列名*>
- <*函数*>
- <*聚合函数*>

SELECT共有6个子句，其中SELECT和FROM是必须的，其它是可选项，必须严格按照如下顺序排列：

```
SELECT [ALL | DISTINCT] <目标列表达式> [AS] [<别名>]
        [, <目标列表达式> [AS] [<别名>] ... ]
FROM {<表名> | <视图名> | <查询表>} [AS] [<别名>]
      [, {<表名> | <视图名> | <查询表>} [AS] [<别名>] ... ]
[ WHERE <条件表达式> ]
[ GROUP BY <列名1> [, <列名2> ... ]
  [ HAVING <条件表达式> ] ]
[ ORDER BY <列名表达式> [ASC | DESC]
          [, <列名表达式> [ASC | DESC] ... ] ]
```

(2) FROM子句指定查询所涉及的表、视图或查询表.

- 为操作方便，常给表取一个别名，称为元组变量

3.6　SQL查询一般格式

(3) WHERE子句给出查询条件，随后的<*条件表达式*>中可以使用下面的谓词运算符：

- 比较运算符：>，>=，<，<=，=，<>，!=；
- 逻辑运算符：AND，OR，NOT；
- 范围运算符：[NOT] BETWEEN...AND；
- 集合运算符：[NOT] IN；
- 空值运算符：IS [NOT] null；
- 字符匹配运算符：[NOT] LIKE；
- 存在量词运算符：[NOT] EXISTS。

SELECT共有6个子句，其中SELECT和FROM是必须的，其它是可选项，必须严格按照如下顺序排列：

```
SELECT [ALL | DISTINCT] <目标列表达式> [AS] [<别名>]
        [, <目标列表达式> [AS] [<别名>] ... ]
FROM {<表名> | <视图名> | <查询表>} [AS] [<别名>]
        [, {<表名> | <视图名> | <查询表>} [AS] [<别名>] ... ]
[ WHERE <条件表达式> ]
[ GROUP BY <列名1> [, <列名2> ... ]
  [ HAVING <条件表达式> ] ]
[ ORDER BY <列名表达式> [ASC | DESC]
        [, <列名表达式> [ASC | DESC] ... ] ]
```

- 在WHERE <*条件表达式*>中可以包含子查询，但不可以直接使用聚合函数，若要使用聚合函数，必须引出一个子查询，如例3.52所示。

3.6　SQL查询一般格式

- [例3.60] 查询每一个同学的学号以及该同学所修课程中成绩最高的课程的课程号和相应成绩。

```
SELECT studentNo, courseNo, score
FROM Score a
WHERE score=( SELECT max(score)
              FROM Score
              WHERE studentNo=a.studentNo )
```

3.6　SQL查询一般格式

SELECT共有6个子句，其中SELECT和FROM是必须的，其它是可选项，必须严格按照如下顺序排列：

```
SELECT [ALL | DISTINCT] <目标列表达式> [AS] [<别名>]
        [, <目标列表达式> [AS] [<别名>] ... ]
FROM {<表名> | <视图名> | <查询表>} [AS] [<别名>]
      [, {<表名> | <视图名> | <查询表>} [AS] [<别名>] ... ]
[ WHERE <条件表达式> ]
[ GROUP BY <列名1> [, <列名2> ... ]
  [ HAVING <条件表达式> ] ]
[ ORDER BY <列名表达式> [ASC | DESC]
          [, <列名表达式> [ASC | DESC] ... ] ]
```

(4) GROUP BY子句表示的含义是：

- 首先按<*列名*1>进行分组，<*列名*1>值相同的元组分为一组；
- 在同组情况下，再按<*列名*2>进行分组，<*列名*2>值相同的元组分为一组；依次类推
- 包含GROUP BY时，SELECT通常选择GROUP BY的分组属性以及聚合属性（通常将聚合函数作用于聚合属性，如avg(score)、sum(creditHour)、count(*)等）输出。

3.6　SQL查询一般格式

(5) **HAVING子句**给出分组后的选择条件，用来选择满足条件的分组。

- 随后的<*条件表达式*>中可直接使用聚合函数，也可以使用子查询。

- **[例3.61]** 查询学生人数不低于500的学院的学院名称及学生人数。

studentNo	studentName	sex	birthday	native	nation	classNo
1500001	李小勇	男	1998-12-21 00:00:00.000	南昌	汉族	CS1501
1500002	刘方晨	女	1998-11-11 00:00:00.000	九江	汉族	IS1501
1500003	王红敏	女	1997-10-01 00:00:00.000	上海	汉族	IS1501
1500004	张可立	男	1999-05-20 00:00:00.000	南昌	蒙古族	CS1501
1500005	王红	男	2000-04-26 00:00:00.000	南昌	蒙古族	CS1502
1600001	李勇	男	1998-12-21 00:00:00.000	南昌	汉族	CS1601
1600002	刘晨	女	1998-11-11 00:00:00.000	九江	汉族	IS1601
1600003	王敏	女	1998-10-01 00:00:00.000	上海	汉族	IS1601
1600004	张立	男	1999-05-20 00:00:00.000	南昌	蒙古族	CS1601
1600005	王红	男	1999-04-26 00:00:00.000	南昌	蒙古族	CP1602
1600006	李志强	男	1999-12-21 00:00:00.000	北京	汉族	CP1602

classNo	className	institute	grade	classNum
CP1601	注册会计16_01班	会计学院	2016	NULL
CP1602	注册会计16_02班	会计学院	2016	NULL
CP1603	注册会计16_03班	会计学院	2016	NULL
CS1501	计算机科学与技术15-01班	信息管理学院	2015	NULL
CS1502	计算机科学与技术15-02班	信息管理学院	2015	NULL
CS1601	计算机科学与技术16-01班	信息管理学院	2016	NULL
ER1501	金融管理15-01班	金融学院	2015	NULL
IS1501	信息管理与信息系统15-01班	信息管理学院	2015	NULL
IS1601	信息管理与信息系统16-01班	信息管理学院	2016	NULL

3.6　SQL查询一般格式

- [例3.62] 查询平均分最高的课程的课程号、课程名和平均分。

```
SELECT a.courseNo, courseName, avg(score) 最高平均分
FROM Course a, Score b
WHERE a.courseNo=b.courseNo
GROUP BY a.courseNo, courseName
HAVING avg(score)=
            ( SELECT max(avgScore)
              FROM ( SELECT avg(score) avgScore
                     FROM Score
                     GROUP BY courseNo ) x
            )
```

3.6　SQL查询一般格式

(6) ORDER BY子句实现对查询结果的排序

- 它是SQL查询的最后一个操作，必须放在最后；
- 其中的<*列名表达式*>可以是列名，也可以是表达式；
- 如果是表达式，则先计算表达式的值，然后排序输出；
- 排序有升序ASC和降序DESC，默认为升序。

(7) 集合运算

- SELECT语句之间可以进行集合运算，包括并UNION、交INTERSECT、差EXCEPT运算。