连接查询

- WHERE是对每一行

> ## 连接查询
>
> 当查询的结果列来自多个表时，完成查询要求多个表进行连接操作
>
> - 在FROM子句中出现多个表，表名之间用逗号分隔，形成指定表的笛卡尔积
> - 如果存在WHERE子句，对笛卡尔积运用查找条件进行行过滤
> - 如果指定DISTINCT，则消除结果中的冗余行

- 排序用ORDER，分组用GROUP

例 找出每一位职员管理的房产的数量

```
SELECT s.branchNo, s.staffNo,
       COUNT(*) AS myCount
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```

- 等值连接，也是非常常见的一种连接方式

等值连接

例 查询每个学生及其选修课程的情况

```
SELECT Student.*, SelCourse.*
FROM Student, SelCourse
WHERE Student.sno = SelCourse.sno;

SELECT Student.*,
       SelCourse.cno, SelCourse.grade
FROM Student, SelCourse
WHERE Student.sno = SelCourse.sno;
```

- 自身连接，这个容易被忽略，但有时候必须用自身连接才能解题

自身连接

例 查询每一门课的间接先修课

```
SELECT First.cno, Second.pcno
FROM Course First, Course Second
WHERE First.pcno = Second.cno;
```

- 一个例子，这种得多写写题感受才会深一些

例 按平均成绩的降序给出所有课程都及格的学生学号、姓名及其平均成绩，成绩统计时不包括008号考查课

```
SELECT Student.sno, sname,
       AVG(grade) AS avgGrade
FROM Student, SelCourse
WHERE Student.sno = SelCourse.sno AND
      cno <> '008'
GROUP BY sno HAVING MIN(grade) >= 60
ORDER BY avgGrage DESC;
```

子查询

一个SELECT语句之内嵌有另一个SELECT语句，子查询一般会在第一个SELECT的WHERE表达式中

- 子查询出现的位置
- 简单来说就是一般只会在WHERE语句或者GROUP BY HAVING中出现，而且一般会用IN,SOME/ANY,ALL,EXISTS语句

子查询出现在什么位置

- 不能任意地将一个SELECT语句嵌入另一个SELECT语句，FROM子句中的表不能是SELECT语句的结果
- 不能用在ORDER BY子句中
- 子查询可以使用在SELECT语句的WHERE和GROUP BY HAVING子句中
- 子查询一般跟在IN、SOME(ANY)、ALL或(NOT) EXISTS等谓词后面
- 子查询也可以出现在INSERT、UPDATE和DELETE语句中

- 不相关子查询

- 内层子查询完全独立于外层SELECT语句, 子查询不接收外层查询的任何输入数据, 向外层SELECT语句传递一个行集
- 概念性执行顺序: 先执行内层子查询, 然后执行外层

不相关子查询

例 查询与'刘晨'在同一个系学习的学生

嵌套子查询法

```
SELECT sno, sname, depart
FROM Student
WHERE depart IN (
 SELECT depart
 FROM Student
 WHERE sname = '刘晨'
);
```

- 当子查询是比较表达式中的一个操作数时, 子查询须出现在表达式的右边

- 相关子查询

相关子查询

- 内层子查询使用外层SELECT语句的列X，
  对于X的每一个取值x，都执行一次内层子查询

例 使用另一种方法实现查询选修了课程C1的学生学号和姓名

```
SELECT DISTINCT sno, sname
FROM Student
WHERE 'C1' IN (
 SELECT cno
 FROM SelCourse
 WHERE Student.sno = SelCourse.sno
);
```

- 默认情况下，子查询中列名取自子查询FROM子句中给定的表，
  也可通过限定列名的办法指定取自外查询FROM子句中的表

- 相关子查询的一些说明，其实有很多子查询都可以直接通过连接、选择、映射操作替代，**同时注意子查询SELECT必须是单个列名或表达式，除非使用了EXISTS**

例 查询所有选修了课程C1的学生姓名

```
SELECT sname
FROM Student
WHERE EXISTS (
 SELECT *
 FROM SelCourse
 WHERE sno = Student.sno AND cno = 'C1'
);

SELECT sname
FROM Student, SelCourse
WHERE Student.sno = SelCourse.sno AND cno = 'C1';
```

- 子查询SELECT列表须是单个列名或表达式，除非使用了谓词EXISTS

- IN谓词子查询

IN谓词子查询

父查询与子查询之间用IN进行连接，判断某个属性列值是否在子查询的结果中

例 查询选修了2学分课程的学生号、成绩

```
SELECT sno, grade
FROM SelCourse
WHERE cno IN (
 SELECT cno
 FROM Course
 WHERE credit = 2
);
```

- 带有比较运算符的子查询，子查询返回单值时可以用比较运算符

带有比较运算符的子查询

子查询返回单值时可以用比较运算符

通用形式

```
expr <|<=|=|<>|>|>= (Subquery)
```

例 查询与'刘晨'在同一个系学习的学生

```
SELECT sno, sname, depart
FROM Student
WHERE depart = (
 SELECT depart
 FROM Student
 WHERE sname = '刘晨'
);
```

- 带有量化比较谓词的子查询，此时会用到SOME/ANY,ALL，这个时候子查询返回值可以不是单值，可以是集合

通用形式

```
expr <|<=|=|<>|>|>= {SOME|ANY|ALL} (Subquery)
```

- expr θ SOME|ANY (Subquery)为真，
  当且仅当至少存在一个由子查询返回的元素s，expr θ s为真
  SOME与ANY含义相同
- expr θ ALL (Subquery)为真，
  当且仅当对每一个由子查询返回的元素s，expr θ s为真

- 带EXISTS谓词的子查询，**因为WHERE是对每一行进行判断，EXISTS返回真/假，相当于判断每一行，所以会经常看到用了EXISTS谓词的子查询一般都是SELECT *。同时注意其他谓词都能用EXISTS替换**

带EXISTS谓词的子查询

通用形式

```
[NOT] EXISTS (Subquery)
```

- EXISTS (Subquery)为真，
  当且仅当子查询返回一个非空的集合
- NOT EXISTS (Subquery)为真，
  当且仅当子查询返回的集合为空
- 一些带EXISTS的子查询不能被其他形式的子查询等价替换
- 所有带IN、比较运算符、SOME(ANY)和ALL谓词的子查询都能用带[NOT] EXISTS的子查询等价替换

- 用NOT EXISTS实现全称量词子查询,主要是sql不支持全称量词

- SQL语言不支持全称量词
- 全称量词可以转换为等价的带有存在量词的谓词
  $(\forall x)P \equiv \neg(\exists x(\neg P))$
- 用NOT EXISTS实现全称量词查询，实现关系代数除运算

- 因此遇到全部这种明显要用全称量词的，要先进行逻辑上的转化，再用NOT EXISTS语句

用NOT EXISTS实现全称量词子查询

例 查询选修了全部课程的学生姓名

形式化

- P: 学生s选修了课程c
- $\forall(c)P$: 对所有课程而言, 学生s都选修了
- 谓词演算转换: $\forall(c)P \equiv \neg\left(\exists c(\neg P)\right)$

语义转换: 查询这样的学生s, 不存在一门课程c是s没有选修的

- 一个NOT EXISTS可以看成一个非，这样好理解一些。一开始想变成逻辑表达式再转化，结果发现还不如直接用语言描述。**其实就是不存在一门课程该学生没有选**

例 查询选修了全部课程的学生姓名

```
-- query student s
SELECT sname
FROM Student
-- no course c exists
WHERE NOT EXISTS (
 SELECT *
 FROM Course
 -- student s does not select course c
 WHERE NOT EXISTS (
  SELECT *
  FROM SelCourse
  WHERE sno = Student.sno AND cno = Course.cno
 )
);
```

- 一个例子，同样用语言描述。只要95002选了此课程，该学生一定选了此课程=>不存在一门课程，95001选了，但是该学生没选

例 查询至少选修了学生95002选修的全部课程的学生的学号

```
-- query student s
SELECT sno FROM Student
-- no course c exists
WHERE NOT EXISTS (
 SELECT * FROM Course
 -- student 95002 select course c
 WHERE EXISTS (
  SELECT  *FROM SelCourse SelCourseX
  WHERE SelCourseX.cno = Course.cno AND SelCourseX.sno = '95002'
 )
 -- student s does not select course c
 AND NOT EXISTS (
   SELECT * FROM SelCourse SelCourseY
   WHERE SelCourseY.cno = Course.cno AND SelCourseX.sno = Student.sno
 )
);
```

扩展SQL语句

- 并运算 UNION

  并运算

  例 查询计算机系的学生及年龄大于19岁的学生

  ```
  SELECT * FROM Student
  WHERE depart = 'computer'
  UNION ALL
  SELECT * FROM Student
  WHERE age > 19;
  ```

  - UNION的结果中保留两个子查询中重复的行

- 交运算 INTERSECT

交运算

例 查询既选修了课程'C1'又选修了'C2'的学生学号与姓名

```
SELECT Student.sno, sname FROM Student, SelCourse
WHERE Student.sno = SelCourse.sno AND cno = 'C1'
INTERSECT
SELECT Student.sno, sname FROM Student, SelCourse
WHERE Student.sno = SelCourse.sno AND cno = 'C2';
```

交运算 INTERSECT ALL

```
Q:= Q1 INTERSECT [ALL] Q2
```

假定x在Q1结果中出现m次，在Q2结果中出现n次

- INTERSECT
  - 若m或n为0, 则Q中x出现的次数为0; 否则为1
- INTERSECT ALL
  - Q中x出现的次数为 $\min(m, n)$

例 $Q1 := \{a, a, a, b, b, c, d\}, Q2 := \{a, a, b, b, b, c, e\}$

$Q1$ INTERSECT $Q2 = \{a, b, c\}$

$Q1$ INTERSECT ALL $Q2 = \{a, a, b, b, c\}$

- 差运算 EXCEPT，WHERE本质上就是选择操作

差运算

例 查询没选修'95001'所选修的任何课程的学号

```
SELECT sno FROM Student
EXCEPT
SELECT SelCourse1.sno
FROM SelCourse SelCourse1, SelCourse SelCourse2
WHERE SelCourse1.cno = SelCourse2.cno AND
      SelCourse2.sno = '95001';
```

差运算 EXCEPT ALL

```
Q:= Q1 EXCEPTT [ALL] Q2
```

假定x在Q1结果中出现m次，在Q2结果中出现n次

- EXCEPTT
  - 若m不为0且n为0, 则Q中x出现的次数为1; 否则为0
- EXCEPT ALL
  - Q中x出现的次数为$m-n$; 若$m-n<0$, 则次数为0

例 $Q1:=\{a,a,a,b,b,c,d\}, Q2:=\{a,a,b,b,b,c,e\}$

$Q1$ EXCEPT $Q2=\{d\}$

$Q1$ EXCEPT ALL $Q2=\{a,d\}$

- 扩展FROM子句,这个就看PPT里面的内容吧
- CTE与递归查询，也看PPT内容

SQL过程与函数

过程没有返回值，函数有返回值

语法

```
CREATE PROCEDURE ProcName ([[IN|OUT|INOUT] paramName type[, ...]])
 routine_body
```

```
CREATE FUNCTION FuncName ([[IN|OUT|INOUT] paramName type[, ...]])
RETURNS type
 routine_body
```

```
DROP {PROCEDURE|FUNCTION} PFName;
```

- 过程

SQL过程

举例

```
delimiter //
CREATE PROCDURE ProcEmp(IN id INT, OUT num INT, INOUT salCount INT)
BEGIN
 DELETE FROM emp WHERE empno = id;
 SELECT MAX(sal) FROM emp INTO num;
 SELECT COUNT(*) INTO salCount FROM emp WHERE sal > salCount;
END //
delimiter ;
SET @salCount = 1250;
CALL ProcEmp(7369, @num, @salCount);
SELECT @num, @salCount;
```

- 函数

SQL函数

举例

```
delimiter //
CREATE FUNCTION GetSal(id INT)
RETUNS INT
BEGIN
 RETURN (SELECT sal FROM emp WHERE empno = id);
END //
delimiter ;
SELECT GetSal(7698);
```