嵌套子查询

- SQL提供嵌套子查询机制。子查询是嵌套在另一个查询中的select-from-where表达式。子查询嵌套在where子句中，通常用于对集合的成员资格、集合的比较以及集合的基数进行检查。主要用于：
  - 集合成员资格
  - 集合的比较
  - 空关系测试
  - 重复元组存在性测试
  - from子句中的子查询
  - with子句

集合成员资格

- SQL允许测试元组在关系中的成员资格。连接词in测试元组是否是集合中的成员，集合是由select子句产生的一组值构成的，对应的还有not in
  - 例1，找出在2009年秋季和2010年春季学期同时开课的所有课程

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
course_id in (select course_id
                from section
                where semester = 'Spring' and year= 2010);
```

集合成员资格

- 例2，找出（不同的）学生总数，他们选修了ID为10101的教师所讲授的课程

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year)
            in (select course_id, sec_id, semester, year
                  from teaches
                  where teaches.ID = 10101);
```

集合的比较

- 考虑查询“找出满足下面条件的所有教师的姓名，他们的工资至少比Biology系某一个教师的工资要高”，在前面，我们将此查询写作：

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';
```

但是SQL提供另外一种方式书写上面的查询。短语“至少比某一个要大”在SQL中用>some表示，则此查询还可写作：

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = 'Biology');
```

集合的比较

- some子句的定义： $C\ \langle comp \rangle\ \text{some}\ r \Leftrightarrow \exists\, t \in r\,(C\ \langle comp \rangle\ t)$，其中$\langle comp \rangle$可以为：$<$, $\leq$, $>$, $=$, $\neq$

$(5 <\ \text{some}\ \{0, 5, 6\})$ = true

$(5 <\ \text{some}\ \{0, 5\})$ = false

$(5 =\ \text{some}\ \{0, 5\})$ = true

$(5 \neq\ \text{some}\ \{0, 5\})$ = true （因为 $0 \neq 5$）

$(= \text{some}) \equiv \text{in}$

但是，$(\neq \text{some}) \not\equiv \text{not in}$

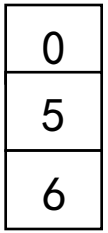
集合的比较

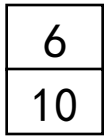
- 考虑查询“找出满足下面条件的所有教师的姓名，他们的工资比Biology系每个教师的工资都高”，在SQL中，结构>all对应于词组“比所有的都大”，则

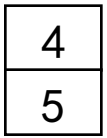
```
select name
     from instructor
     where salary > all(select salary
                        from instructor
                        where dept_name = 'Biology');
```

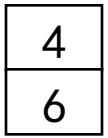
集合的比较

- all子句的定义： $C\ \langle comp \rangle\ \text{all}\ r \Leftrightarrow \forall\ t \in r\ (C\ \langle comp \rangle\ t)$

(5< all ) = false

(5< all ) = true

(5 = all ) = false

(5 ≠ all ) = true (因为 4 ≠ 5, 6 ≠ 5)

(≠ all) ≡ not in

但是，(= all) ≢ in

集合的比较

- 例，找出平均工资最高的系

```
select dept_name
from instructor
group by dept_name
having avg (salary) >= all (select avg (salary)
                            from instructor
                            group by dept_name);
```

空关系测试

- SQL还有一个特性可测试一个子查询的结果中是否存在元组。exists结构在作为参数的子查询为空时返回true值
  - exists $r \Leftrightarrow r \neq \emptyset$
  - not exists $r \Leftrightarrow r = \emptyset$
- 我们还可以使用not exists结构模拟集合包含（即超集）操作：可将“关系A包含关系B”写成“not exists(B except A)”

空关系测试

- 例，找出在2009年秋季学期和2010年春季学期通识开课的所有课程

使用exists结构，重写该查询：

```
select course_id
from section as S
where semester = 'Fall' and year = 2009 and
      exists (select *
              from section as T
              where semester = 'Spring' and year= 2010 and
                    S.course_id = T.course_id );
```

空关系测试

- 例，找出选修了Biology系开设的所有课程的学生

  使用except结构，写该查询：

```
select distinct S.ID, S.name
from student as S
where not exists ((select course_id
                   from course
                   where dept_name = 'Biology')
                  except
                  (select T.course_id
                   from takes as T
                   where S.ID = T.ID));
```

在Biology系开设的所有课程集合

找出*S.ID*选修的所有课程

- 注意：$X - Y = \emptyset \Leftrightarrow X \subseteq Y$

重复元组存在性测试

- SQL提供一个布尔函数，用于测试在一个子查询的结果中是否存在重复元组。如果作为参数的子查询结果中没有重复的元组unique结构将返回true值
  - 例1，找出所有在2009年最多开设一次的课程

```
select T.course_id
from course as T
where unique (select R.course_id
                    from section as R
                    where T.course_id = R.course_id and
                        R.year = 2009);
```

  - 也可以将上述查询语句中的unique换成1>=

重复元组存在性测试

- 例2，找出所有在2009年最少开设两次的课程

```
select T.course_id
from course as T
where not unique (select R.course_id
                  from section as R
                  where T.course_id = R.course_id and
                        R.year = 2009);
```

- unique, not unique 在oracle8, sql server7中不支持

from子句中的子查询

- SQL允许在from子句中使用子查询表达式。任何select-from-where表达式返回的结果都是关系，因而可以被插入到另一个select-from-where中任何关系可以出现的位置
  - 例，找出系平均工资超过42 000美元的那些系中教师的平均工资

    在前面的聚集函数中，我们使用了having写此查询。现在，我们用在from子句中使用子查询重写这个查询：

```
select dept_name, avg_salary
from (select dept_name, avg (salary)
          from instructor
          group by dept_name)
          as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

相当于局部视图

from子句中的子查询

- 例，找出在所有系中工资总额最大的系

  在此，having子句是无能为力的。但我们可以用from子句的子查询轻易地写出如下查询：

```
select max(tot_salary)
from (select dept_name, sum(salary)
from instructor
group by dept_name) as dept_total(dept_name, tot_salary);
```

with子句

- with子句提供定义临时关系的方法，这个定义只对包含with子句的查询有效
  - 例，找出具有最大预算值的系

```
with max_budget (value) as
        (select max(budget)
         from department)
select budget
from department, max_budget
where department.budget = max_budget.value;
```

定义临时关系

使用临时关系

with子句

- 例，找出工资总额大于平均值的系

```
with dept_total (dept_name, value) as
     (select dept_name, sum(salary)
      from instructor
      group by dept_name),
     dept_total_avg (value) as
     (select avg(value)
      from dept_total)
select dept_name
from dept_total A, dept_total_avg B
where A.value >= B.value ;
```

每个系的工资总和

所有系的平均工资