

数据库系统原理

陈岭

浙江大学计算机学院

4

SQL语言（2）

- SQL查询的基本结构
- 集合运算
- 空值
- 聚集函数
- 嵌套子查询
- 数据库的修改

SQL查询的基本结构

□ SQL查询的基本结构由3个子句构成：select, from, where

■ SELECT A_1, A_2, \dots, A_n

FROM r_1, r_2, \dots, r_m

WHERE P

■ 上述查询语句等价于关系代数表达式：

$$-\Pi_{A_1, A_2, \dots, A_n} (\sigma_P (r_1 \times r_2 \times \dots \times r_m))$$

■ 查询的输入是在from子句中列出的关系，在这些关系上进行where和select子句中指定的运算，然后产生一个关系作为结果

select子句

- ❑ 例，找出所有教师的名字

```
select name  
from instructor;
```

表示成关系代数表达式为： $\Pi_{name}(instructor)$

- ❑ 注意：SQL不允许在属性名称中使用字符 ‘-’，例如，使用 dept_name代替dept-name
- ❑ SQL不区分字母的大小写。因此，你可以使用大写字母或小写字母命名表、属性等

❑ SQL允许在关系以及SQL表达式结果中出现重复的元组

■ 若要强化去除重复，可在select后加入关键词distinct

■ 例，查询 *instructor* 关系中的所有系名，并去除重复

```
select distinct dept_name  
from instructor;
```

■ SQL也允许我们使用关键词all来显式指明不去除重复(SQL默认就是all)

■ 例，

```
select all dept_name  
from instructor
```

select子句

□ 星号 “*” 在select子句中，可以用来表示 “所有的属性”

■ 例，

```
select  *  
from  instructor;
```

□ select子句还可带含有+、-、*、/运算符的算术表达式，运算对象可以是常数或元组的属性

■ 例，

```
select  ID, name, salary *1.05  
from  instructor;
```

- where子句允许我们只选出那些在from子句的结果关系中满足特定谓词的元组

- 例，找出所有在Computer Science系并且工资超过70 000美元的教师的姓名

```
select name
```

```
from instructor
```

```
where dept_name = 'Comp. Sci.' and salary > 70000;
```

- 上述SQL查询语句，对应的关系代数表达式为：

$$\Pi_{name} (\sigma_{dept_name = 'Comp. Sci.' \wedge salary > 70000} (instructor))$$

- SQL允许在where子句中使用逻辑连词and, or和not, 也可以使用between指定范围查询。逻辑连词的运算对象可以是包含比较运算符<、<=、>、>=、= 和<>的表达式

■例, 找出工资在90 000美元和100 000美元之间的教师的姓名

```
select name
from instructor
where salary <=100000 and salary >=90000;
```

或者:

```
select name
from instructor
where salary between 90000 and 100000;
```


□ from子句是一个查询求值中需要访问的关系列表，通过from子句定义了一个在该子句中所列出关系上的笛卡尔积

■ 例，找出关系 *instructor* 和 *teaches* 的笛卡尔积

```
select *
```

```
from instructor, teaches;
```

□ 例，找出Computer Science系的教师名和课程标识

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID and instructor.dept_name =
      'Comp. Sci.' ;
```

这个前缀是必要的

instructor (ID, name, dept_name, salary)

teaches (ID, course_id, sec_id, semester, year)

- SQL提供可为关系和属性重新命名的机制，即使用as子句：

old-name as new-name

as子句既可以出现在select子句中，也可以出现在from子句中

- 例，考虑刚刚的查询, 将属性*name*重命名为 *instructor_name*

```
select name as instructor_name, course_id  
from instructor, teaches  
where instructor.ID= teaches.ID and instructor.dept_name =  
    'Comp. Sci. ' ;
```

更名运算

□ 使用更名运算，对关系重命名

- 例，找出所有教师，以及他们所讲授课程的标识

```
select T.name, S.course_id  
from instructor as T, teaches as S 为了引用简洁  
where T.ID= S.ID;
```

- 例，找出所有教师名，他们的工资至少比Biology系某一个教师的工资要高

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

- ❑ 对字符串进行的最通常的操作是使用操作符like的模式匹配，使用两个特殊的字符来描述模式：
 - 百分号（%）：匹配任意子串
 - 下划线（_）：匹配任意一个字符
- ❑ 例，找出所在建筑名称中包含子串 ‘Watson’ 的所有系名

```
select dept_name  
from department  
where building like '%Watson%' ;
```

- ❑ 为使模式中能够包含特殊字符（即%和_），SQL允许定义转义字符。我们在like比较运算中使用escape关键词来定义转义字符
 - 例，使用反斜线（\）作为转义字符
 - like 'ab\%cd%' escape '\' 匹配所有以“ab%cd”开头的字符串
 - like 'ab\\cd%' escape '\' 匹配所有以“ab\cd”开头的字符串
- ❑ SQL还允许在字符串上有多种函数，例如串联（“||”）、提取子串、计算字符串长度、大小写转换（用upper（s）将字符串s 转换为大写或用lower（s）将字符串s 转换为小写）、去掉字符串后面的空格（使用trim（s））等等

排列元组的显示次序

□ SQL为用户提供了一些对关系中元组显示次序的控制。order by子句就可以让查询结果中元组按排列顺序显示

■ 例，按字母顺序列出在Physics系的所有教师

```
select name
from instructor
where dept_name = 'Physics'
order by name;
```

排列元组的显示次序

- `order by`子句默认使用升序。要说明排序顺序，我们可以用`desc`表示降序，或者用`asc`表示升序
 - 例，按`salary`的降序列出整个`instructor`关系，如果有几位教师的工资相同，就将他们按姓名升序排列

```
select *  
from instructor  
order by salary desc, name asc;
```


- ❑ 在关系模型的形式化数学定义中，关系是一个集合。因此，重复的元组不会出现在关系中。但在实践中，包含重复元组的关系是有用的
- ❑ 可以用关系运算符多重集版本（ Multiset versions ）来定义SQL查询的复本定义，在此定义几个关系代数运算符的多重集版本，已知多重集关系 r_1 和 r_2
 - $\sigma_{\theta}(r_1)$: 如果在 r_1 中有元组 t_1 的 c_1 个复本，而且 t_1 满足选择 σ_{θ} ，那么有 c_1 个 t_1 的复本在 $\sigma_{\theta}(r_1)$ 中
 - $\Pi_A(r)$: 对于 r_1 中 t_1 的每个复本，在 $\Pi_A(r_1)$ 中都有一个 $\Pi_A(t_1)$ 的复本与其对应，其中 $\Pi_A(t_1)$ 表示单个元组 t_1 的投影
 - $r_1 \times r_2$: 如果有 c_1 个 t_1 的复本在 r_1 中且有 c_2 个 t_2 的复本在 r_2 中，那么有 $c_1 * c_2$ 个 $t_1 \cdot t_2$ 元组的复本在 $r_1 \times r_2$ 中

□ 例，假设多重集关系 $r_1(A, B)$ 和 $r_2(C)$ 如下所示：

$$r_1 = \{(1, a), (2, a)\}$$

$$r_2 = \{(2), (3), (3)\}$$

那么， $\Pi_B(r_1) = \{(a), (a)\}$ ，则 $\Pi_B(r_1) \times r_2$ 为：
 $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$

□ SQL中的select子句也支持关系代数运算符的多重集版本： σ_θ 、 Π_A 、 \times ，
 形如 $\text{select } A_1, A_2, \dots, A_n$

from r_1, r_2, \dots, r_m

where P ;

的SQL查询等价于关系代数表达式（多重集版本）：

$$\Pi_{A_1, A_2, \dots, A_n}(\sigma_P(r_1 \times r_2 \times \dots \times r_m))$$

集合运算

- ❑ SQL作用在关系上的union、intersect和except运算对应于数学集合论中的 \cup , \cap 和 $-$ 运算
- ❑ union、intersect和except运算与select子句不同，它们会自动去除重复
- ❑ 如果想保留所有重复，必须用union all、intersect all和except all
- ❑ 假设一个元组在关系 r 中重复出现了 m 次，在关系 s 中重复出现了 n 次，那么这个元组将会重复出现：
 - 在 r union all s 中，重复出现 $m+n$ 次
 - 在 r intersect all s 中，重复出现 $\min(m, n)$ 次
 - 在 r except all s 中，重复出现 $\max(0, m-n)$ 次

当 $m < n$

当 $m > n$

集合运算

- 例1，找出在2009年秋季开课，**或者**在2010年春季开课或两个学习**都开**课的所有课程

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

section(course_id, sec_id, semester, year, building, room_number, time_slot_id)

□ 例2, 找出在2009年秋季和2010年春季同时开课所有课程

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

□ 例3, 找出在2009年秋季开课, 但不在2010年春季开课的所有课程

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

集合运算

- ❑ 在Oracle中，支持union, union ALL, intersect和Minus；但不支持Intersect ALL和Minus ALL
- ❑ 在SQL Server 2000中，只支持union和union ALL

聚集函数

- ❑ 聚集函数是以值的一个集合（集或多重集）为输入，返回单个值的函数。SQL提供了五个固有聚集函数：
 - 平均值：avg
 - 最小值：min
 - 最大值：max
 - 总和：sum
 - 计数：count
 - 其中，sum和avg的输入必须是数字集，但其他运算符还可作用在非数字数据类型的集合上，如字符串

- ❑ 除了上述的五个基本聚集函数外，还有分组聚集（group by）。group by子句中给出的一个或多个属性是用来构造分组的，在group by子句中的所有属性上取值相同的元组将被分在一个组中
- ❑ having子句类似于where子句，但其是对分组限定条件，而不是对元组限定条件。having子句中的谓词在形成分组后才起作用，因此可以使用聚集函数

□ 例1, 找出Computer Science系教师的平均工资

```
select avg (salary) as avg_salary  
from instructor  
where dept_name= 'Comp. Sci.' ;
```

上述SQL查询等价于关系代数表达式:

$$g_{avg(salary)} (\sigma_{dept_name = 'Comp. Sci'} (instructor))$$

<u>avg_salary</u>

77333

聚集函数

□ 例2，找出每个系的平均工资

```
select dept_name avg (salary) as avg_salary  
from instructor  
group by dept_name ;
```

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

注意：任何没有出现在group by子句中的属性，如果出现在select子句中的话，它只能出现在聚集函数内部，否则这样的查询就是错误的！

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000



□ 例3，找出教师平均工资超过42 000美元的系

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

注意：与select子句的情况类似，任何出现在having子句中，但没有被聚集的属性必须出现在group by子句中，否则这样的查询就是错误的！

dept_name	avg(salary)
Physics	91000
Elec. Eng.	80000
Finance	85000
Comp. Sci.	77333
Biology	72000
History	61000

- 在第2章中，我们提到过SQL允许使用null值表示属性值信息缺失。我们在谓词中可以使用特殊的关键词null测试空值, 也可以使用is not null测试非空值

- 例，找出instructor关系中元组在属性salary上取空值的教师名

```
select name
```

```
from instructor where salary is null;
```

- 空值的存在给聚集运算的处理也带来了麻烦。聚集函数根据以下原则处理空值：

- 除了count (*) 外所有的聚集函数都忽略输入集合中的空值
- 规定：空集的count运算值为0，其他所有聚集运算在输入为空集的情况下返回一个空值

□ 例，计算所有教师工资总和

```
select sum(salary)  
from instructor;
```

- sum运算符会忽略输入中的所有空值
- 如果，*instructor*关系中所有元组在*salary*上的取值都为空，则sum运算符返回的结果即为null

- ❑ 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

- ❑ 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);
```


- 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: $<, \leq, >, =, \neq$

$$(5 < \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{true}$$

$$(5 < \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 = \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true}$$

$$(5 \neq \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true} \text{ (因为 } 0 \neq 5)$$

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

- 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: $C \langle comp \rangle all\ r \Leftrightarrow \forall t \in r (C \langle comp \rangle t)$

$$(5 < all \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = false$$

$$(5 < all \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array}) = true$$

$$(5 = all \begin{array}{|c|} \hline 4 \\ \hline 5 \\ \hline \end{array}) = false$$

$$(5 \neq all \begin{array}{|c|} \hline 4 \\ \hline 6 \\ \hline \end{array}) = true \text{ (因为 } 4 \neq 5, 6 \neq 5)$$

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

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);
```

- ❑ 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
 - $\text{exists } r \Leftrightarrow r \neq \emptyset$
 - $\text{not exists } r \Leftrightarrow r = \emptyset$
- ❑ 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)`”

□ 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

- 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)
```

←--- 定义临时关系

```
select budget  
from department, max_budget  
where department.budget = max_budget.value;
```

←--- 使用临时关系

- 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

数据库的修改-删除

- ❑ 除了数据库信息的抽取外，SQL还定义了增加、删除和更新数据库信息的操作
- ❑ 删除请求的表达与查询非常类似。我们只能删除整个元组，而不能只删除某些属性上的值。SQL用如下语句表示删除：

```
delete from r  
where P;
```

其中P代表一个谓词，*r*代表一个关系

- 例1，从instructor关系中删除与Finance系教师相关的所有元组

```
delete from instructor  
where dept_name = 'Finance' ;
```


数据库的修改-删除

- 例2，从instructor关系中删除所有这样的教师元组，他们在位于Watson大楼的系工作

```
delete from instructor
where dept_name in (select dept_name
                    from department
                    where building = 'Watson' );
```

数据库的修改-删除

- 例3，删除工资低于大学平均工资的教师记录

```
delete from instructor
where salary < (select avg (salary)
                from instructor);
```

- 问题：当我们从instructor关系中删除元组时，平均工资就会改变
- SQL中的解决方案：
 - 首先，计算出平均工资，并找出要删除的所有元组
 - 然后，删除上述找到的所有元组（不重新计算平均工资，也不重新测试元组是否符合删除条件）
 - 在同一SQL语句内，除非外层查询的元组变量引入内层查询，否则内层查询只进行一次

数据库的修改-插入

- ❑ SQL允许使用insert语句，向关系中插入元组，形式如下：

```
insert into r [(c1, c2, ...)]  
  values (e1, e2, ...);  
insert into r [(c1, c2, ...)]  
  select e1, e2, ... from ...;
```

- 例1，假设我们要插入的信息是Computer Science系开设的名为“Database Systems”的课程CS-437，它有4个学分

```
insert into course  
  values ( 'CS-437' , 'Database Systems' , 'Comp. Sci.' , 4);
```

- SQL允许在insert语句中指定属性，所以上述语句还可写为：

```
insert into course (course_id, title, dept_name, credits)  
  values ( 'CS-437' , 'Database Systems' , 'Comp. Sci.' , 4);
```

数据库的修改-插入

- 若上例中， Database Systems” 课程的学分未知，插入语句还可写为：

```
insert into course
values ( 'CS-437' , 'Database Systems' , 'Comp. Sci.' , null );
```

```
insert into course (course_id, title, dept_name)
values ( 'CS-437' , 'Database Systems' , 'Comp. Sci.' );
```

- 假设我们想让Music系每个修满144学分的学生成为Music系的教师，其工资为18000美元

```
insert into instructor
select ID, name, dept_name, 18000
from student
where dept_name = 'Music' and tot_cred > 144;
```


数据库的修改-更新

- SQL允许使用update语句，在不改变整个元组的情况下改变其部分属性的值，形式如下：

```
update r
set <c1=e1 , [c2=e2, ... ]>
[where <condition>] ;
```

- 例1，假设给工资超过100 000美元的教师涨3%的工资，其余教师涨5%
我们可以写两条update语句

```
update instructor
set salary = salary * 1.03
where salary > 100000;
update instructor
set salary = salary * 1.05
where salary <= 100000;
```



注意：这两条update语句的顺序十分重要。如果调换顺序，可能导致工资略少于100 000美元的教师将增长8%的工资

数据库的修改-更新

- 针对上例查询，我们也可以使用SQL提供的case结构，避免更新次序引发的问题，形式如下：

```
case
when pred1 then result1
when pred2 then result2
. . .
when predn then resultn
else result0
end
```

因此上例查询可重新为：

```
update instructor
set salary = case
    when salary <= 100000 then salary * 1.05
    else salary * 1.03
end
```

❑ SQL查询语句的通用形式:

select <[distinct] $c1, c2, \dots$ >

from < $r1, \dots$ >

[where <*condition*>]

[group by < $c1, c2, \dots$ > [having <*cond2*>]]

[order by < $c1$ [desc] , [$c2$ [desc|asc], \dots]]>

❑ SQL查询语句执行顺序:

from → where → group (aggregate) → having → select → order by

❑ SQL支持关系上的基本集合运算, 包括并、交、和差运算

总结

- ❑ SQL支持聚集，可以把关系进行分组，还支持在分组上的集合运算
- ❑ SQL支持在外层查询的where和from子句中嵌套子查询
- ❑ SQL提供了用于更新、插入、删除信息的结构

谢谢！