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，...，A_n

    FROM r_1，r_2，...，r_m

    WHERE P
  - 上述查询语句等价于关系代数表达式：

    $$\Pi_{A1, A2, \ldots, An}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$
  - 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

select子句

- SQL允许在关系以及SQL表达式结果中出现重复的元组
  - 若要强行去除重复，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

- 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：

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

where子句

- 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

- 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;
```

from子句

- 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)
```

更名运算

- 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
where T.ID= S.ID;
```

为了引用简洁

  - 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
where T.salary > S.salary and S.dept_name = 'Biology';
```

为了区分

字符串运算

- 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%’ ;
```

字符串运算

- 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;
```

重复

- 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
  - σ_θ（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σ_θ（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

重复

- 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 、x，形如

select $A_1, A_2, \ldots, A_n$

from $r_1, r_2, \ldots, r_m$

where P;

的SQL查询等价于关系代数表达式（多重集版本）：

$$\Pi_{A1, A2, \ldots, An}(\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

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∪,∩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)

集合运算

- 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);
```

集合运算

- 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

聚集函数

- 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

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.' ;
```

avg_salary
77333

上述SQL查询等价于关系代数表达式：

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

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

聚集函数

- 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

空值

- 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

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