数据库理论作业Page60

曹广杰

15352015 数据科学与计算机

授课教师：刘玉葆

2017/10/15

[3.11]

使用大学模式，用SQL进行如下查询：

classroom(*building*, *room_number*, *capacity*)
department(*dept_name*, *building*, *budget*)
course(*course_id*, *title*, *dept_name*, *credits*)
instructor(*ID*, *name*, *dept_name*, *salary*)
section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)
student(*ID*, *name*, *dept_name*, *tot_cred*)
takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
advisor(*s_ID*, *i_ID*)
time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
prereq(*course_id*, *prereq_id*)

1. 找出所有至少选修了一门Comp. Sci. 课程的学生姓名，保证结果中没有重复的名字

   分析题意：

   - 找出选修了课程x的学生；
   - 课程x属于Comp. Sci.系；

     于是我们可以通过条件限定与子查询语句的使用，实现该查询。

   此外，还可以使用关系数据库的基本结构natural join：natural join在合并多个table的时候，只处理这几个table属性相同的元组。对于完全重复的元组实行合并操作，同时对一部分属性重复的元组创建新的元组以容纳满足条件的元组的所有信息。

   这种情况下，我们将student表（含有学生name）与course（含有系name）进行join，但是course并不等同于选课——takes，所以还需要natural join这个表格。

   ```
   select distinct name
   from student natural join takes natrual join course
   where course.dept = 'Comp.Sci.'
   ```

2. 找出所有没有选修在2009年春季之前开设的任何课程的学生ID和姓名

   分析题意：

   - 查询学生信息，排除不符合条件的部分
   - 条件：选修2009年之前的课程
   - 细节：2009年春季之前，就是2009年之前，用grade属性实现限制

   句法：

- except 排除，在全集中排除一个子集。

```
select id, name
from student
except
select id, name
from student natural join takes
where grade < 2009
```

3. 找出每个系教师的最高工资值（每个系至少有一名老师）

分析题意：

- 查询T教师的最高工资
- T教师属于大学某系
- 某系不一定是哪个系

首先，查询教师的最高工资，使用max句法。而该操作将会应用到大学的各个系中（主键为dept_name）——目前，实现这种同一操作的映射只有group by可以做到。

句法：

- group by 按...分类。在每一个类中进行该语句之前的操作。

```
select dept_name, max(salary)
from instructor
group by dept_name
```

4. 在各系最高工资的集合中，选择最低的工资

分析题意：

- 在集合中选择最低的工资
- 该集合是各个系最高工资的集合
- 各个系最高工资的集合我们刚刚求算过

```
select min(salary)
from
    (select max(salary)
    from instructor
    group by dept_name)
```

[3.12]

使用大学模式，用SQL写出如下查询

1. 创建一门课程“CS-001”，名为“Weekly Seminar”，学分为0；

句法：

- insert into 添加元组。可以在后续语句中补充value值。

```
insert
into course
values('CS-001', 'Weekly Seminar', 'Comp.Sci.', 0)
```

2. 创建该课程在2009年秋季的一个课程段，*sec_id*= 1；

   分析题意：

   - 创建课程段， sec_id = 1
   - 有信息不知道

   句法：不知道的信息只要不是主键，就用null代替

```
insert
into section
values('CS-001', 1, 'Autumn', 2009, null, null, null)
```

3. 让Comp.Sci.的每一个学生都选修上该课程段；

   分析题意：

   - 让学生选课——添加选课记录
   - 学生是计算机系的学生
   - takes表示选课记录

   句法：

   - 有时候select中是可以添加常量的，这种操作可以简化查询信息的嵌套性

```
insert
into takes
select id, 'CS-001', 1, 'Autumn', 2009, null
from student
where dept_name = 'Comp.Sci.'
```

4. 删除名为Chavez的学生选修上述课程段的信息；

   分析题意：

   - 删除选课记录（限定课程段，限定学生信息）
   - 添加课程段信息，添加学生ID信息

   句法：

   - in 在...中。后接一个集合信息，当然这里使用的返回值可能是只有一个元素的集合。这种操作就是避免出现重名的情况。

```
delete
from takes
where course.id = 'CS-001'
    and section.id = 1
    and year = 2009
    and semester = 'Autumn'
    and id in
        (select id
        from student
        where name = 'Chavez')
```

5. 删除课程CS-001，如果运行删除操作之前，没有先删除授课信息（课程段），会发生什么；

   分析题意：

- 如果事先没有删除课程段信息，会因为有来自课程段table的外键索引导致无法删除。

```
delete
from course
where course.id = 'CS-001'
```

6. 删除课程名称中包括“database”的任意课程的任意课程段所对应的所有takes元组，课程名匹配中忽略大小写；

   分析题意：

   - 删除满足条件的takes元组
   - 条件：名称中包含database，不分大小写
   - takes中只有id，如果由name定义需要由course子查询

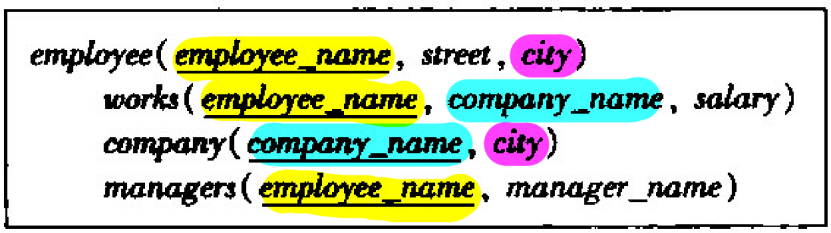
   句法：

   - lower() 转化为小写。将所有的参数都转化为小写，自然就不区分大小写。
   - like 字符串部分匹配。`%` 代表字符串未知的部分。

```
delete
from takes
where course.id in
    (select id
    from course
    where lower(title) like '%database%')
```

[3.16]

考虑3-20的雇员数据库，加下划线的是主码。实现SQL查询：



1. 找出所有为“First Bank Corporation”工作的雇员的名字；

```
select employee_name
from employee natural join company
where company.company_name = 'First Bank Corporation'
```

2. 找出数据库中生活在公司所在城市的雇员；

   分析题意：

   - 查询符合条件的雇员

- 条件：生活在公司所在城市

查询雇员的信息：

需要从employee中提取信息，employee包含雇员的全部信息，并且不存在其他的无用的信息。

与公司同一个城市：

"同一个城市"则city是一个相同的属性，我们需要此类元组的信息合并，这里使用了natural join。

“与公司”则公司是限制条件，限制公司的信息——使用子查询works。

```
select employee.employee_name as name
from employee natural join company
where company_name =
    (select company_name
    from works
    where works.employee_name = name)
```

3. 找出数据库中所有居住的街道和城市与其经理相同的雇员；

分析题意：

1. 查询符合条件的雇员信息；
2. 条件：居住的街道与city与经理相同

查询雇员信息：

从employee中查询，employee中包含street信息。

限定条件：

“经理所在的街道和城市”则可以在employee中获得，通过参数manager_name获得经理的信息。

“相同的街道与城市”则属于两个相同的属性，这时候应该使用natural join，将全集与经理的集合针对居住信息实现natural join；

“经理是否为雇员对应的经理”添加一个where语句，把两个集合中的name信息（一个是所有雇员的信息，另一个是经理的信息）进行对应即可；

```
select employee.employee_name as name
from employee natural join
    (select *
    from employee
    where employee_name in
        (select manager_name
        from managers)) as manager
where manager.manager_name =
    (select manager_name
    from manager
    where employee_name = name)
```

4. 找出工资高于其所在公司雇员平均工资的所有雇员；

分析题意：

- 找出符合条件的雇员
- 条件：工资高于所在公司平均水平
- 由于是针对每一个公司分别定制，所以使用group by语句

```
select company_name, employee_name
from works
where salary >= AVG(salary)
group by company_name
```

5. 找出工资总和最小的公司；

   分析题意：

   - 找出公司信息
   - 工资总和最小

   找出公司的信息时，获得的工资总和信息是基于不同的公司分别计算。所以应该使用group by。并对工资总数进行处理。

```
select company_name
from works
where sum(salary) <= all
    (select sum(salary)
    from works
    group by company_name)
```

[3.17]

考虑3-20关系数据库，查询表达式SQL：

1. 为“First Bank Corporation"的雇员增长10%的工资；

   分析题意：

   - 增长10%的工资
   - 操作对象为雇员，而非经理

   所以需要使用except实现子集的排除。

   句法：

   - update 更新数据关系；
   - set 操作具体的属性信息；

```
update works
set salary = 1.1*salary
where employee_name in
    (select employee_name
    from employee
    except
    select manager_name
    from managers)
```

2. 为“First Bank Corporation”的经理增长10%的工资；

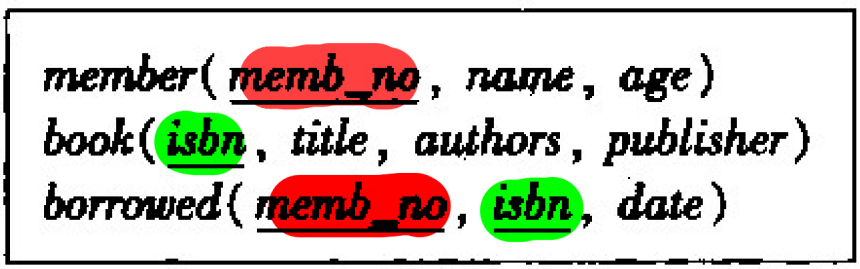
```
update works
set salary = 1.1*salary
where employee_name in
    (select manager_name
    from manager)
```

3. 删除“Small Bank Corporation”的雇员在works关系中的所有元组；

```
delete
from works
where company_name = 'Small Bank Corporation'
```

[3.21]

考虑3-21的图书馆数据库，实现查询SQL：



1. 打印借阅了任意由”McGraw-Hill“出版社出版的书的会员名字；

   分析题意：

   - 查询符合条件的会员名字（member）
   - 借阅了由“McGraw”出版的书——即借书的ID在出版社ID集合中

```
select name
from member natural join borrowed
where isbn in
    (select isbn
    from book
    where publisher = 'McGraw-Hill')
```

2. 打印借阅了所有由”McGraw-Hill“出版社出版的书的会员名字；

   分析题意：

   - 查询符合条件的会员名字
   - 条件：借阅了该出版社的所有的书

   借阅了所有的书，即出版社出书的集合是借书集合的子集。而事实上，这样的问题最后也就转化为集合的比较问题。

   而集合的比较则无非加减，这里笔者使用了减法except，用not exists表示是否是空集——即两个集合是否相等。

```
select name
from member natural join borrowed as borrowed_note
where
    not exists
    (select isbn
    from book
    where publisher = 'McGraw-Hill'
    except
    select isbn
    from borrowed_note)
```

一点拓展思考：

在数据库处理的时候，毫无疑问，我们应对的具体对象就是数据。

- 数据可能是单独的，在数据库中根据条件查找信息的时候，每一个信息都符合的条件；
- 数据可能是集合的，限定数据的信息不再是对每一个元组都有效，反而提供的用于比较的参考数据也是集合。于是我们最后用于限定条件的信息就转化为集合的运算。

集合运算无非也就是子集与非子集的关系，因为集合所表达的信息最后都用于与当前所给出的集合条件进行比较。而包含关系我们已经在上文中进行了讨论，并最终得到了解决。

综上，在目前数据库的结构体系下，SQL所拥有的语言是可以处理目前所遇到的数据的任何问题的。（转载请声明出处）

3. 对于每一个出版商，打印借阅了多于5本由该出版商出版的书的会员的名字；

分析题意：

- 查询符合条件的会员信息
- 条件：借阅特定的书的数量大于5
- 特定：由某出版社出版

查询会员信息需要从member中读取。

借阅书的限定条件：

借阅书的条件由两个信息综合限定，读者（书是这个读者借的）与出版社，而这两个信息又各自独立，分别在两个table中存在，因此想要限定就一定要有关联因素。当然这里可以使用natural join语句。

由此也可以看出，使用natural join语句的使用情况就是在筛选两个table中的交集信息时，限定条件也分别处于两个table中。 *（转载请声明出处）*

```
select name
from member as guest
where 5 <
    (select count(*)
    from borrowed natural join book
    where memb_no = guest.memb_no
        and publisher = 'McGraw-Hill')
```

4. 打印每一位会员借阅书籍量的平均值。

分析题意：

- 计算集合中元素的平均值
- 集合元素为每一位会员借阅量
- 获取某一位会员的借阅量需要在borrowed中对书的ID进行统计

- 推广到多个会员中操作使用——group by *（转载请声明出处）*

```
select AVG(book_cnt)
from
    (select count(isbn)
    from borrowed
    group by memb_no) as book_cnt
```