数据库课后习题复习

一、第一章

1.1

这一章讲述了数据库系统的几个主要的优点。它有哪两个不足之处？

第一：建立数据库系统需要更多的知识，金钱，技能，和时间。

第二：数据库的复杂性可能会导致性能下降。

1.6

在 Web 查找中使用的关键字查询与数据库查询很不一样。请列出这两者之间在查询表达方式和查询结果是什么方面的主要差异。

通过提供以下内容列表来指定Web中使用的查询：没有特定语法的关键字。 结果通常是的有序列表URL，以及有关URL内容的信息摘要。 在相反，数据库查询具有允许复杂查询的特定语法指定。 在关系世界中，查询的结果总是一张表。

1.9

解释物理数据独立性的概念，以及它在数据库系统中的重要性。

物理数据独立性是修改物理数据的能力方案而无需重写应用程序。 这样修改包括从未阻止的记录存储更改为阻止的记录存储，或从顺序访问文件到随机访问文件。 这样的修改可能是在记录中添加字段； 应用程序视图从程序隐藏了此更改

1.11

请给出至少两种理由说明为什么数据库系统使用声明性查询语言，如 SQL，而不是只提供 C 或者 C ++ 的函数库来执行数据操作。

第一：SQL语言便于程序员使用和学习

第二：程序员不必担心如何编写查询确保他们将有效执行； 高效的选择执行技术留给数据库系统。 陈述式规范使数据库系统更容易正确地进行执行技术的选择

1.12

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

图 1-4 *faculty* 表

用这个表设计会导致哪些问题?

1. 如果一个部门有一位以上的讲师，则建筑物名称和预算会重复多次。 更新建筑物名称和预算可能会在某些副本上执行，而在其他副本上则无法，导致状态不一致，但不清楚实际是什么部门的建筑物名称和预算。**即会造成数据的冗余**
2. 一个部门至少需要一名教员才能进行建设和预算信息将包含在表格中。 可以使用空值如果没有讲师，但很难处理空值。
3. 如果删除部门中的所有讲师，则建筑物和预算信息也会丢失。 理想情况下，我们希望拥有部门数据库中的信息，无论部门是否是否使用关联的讲师，而不求助于空值。

1.14

解释两层和三层体系结构之间的区别。对 Web 应用来说哪一种更合适？为什么?

在两层应用程序体系结构中，该应用程序运行在客户端计算机，并直接与数据库系统通信在服务器上运行。 相反，在三层体系结构中，应用程序代码在客户端计算机上运行的计算机与应用程序服务器通信在服务器上，并且永远不要直接与数据库通信 的三层架构更适合于Web应用程序。

三层架构指的是Web服务器、应用程序服务器、DB服务器

二、第二章

2.2

考虑从 *instructor* 的 *dept_name* 属性到 *department* 关系的外码约束，给出对这些关系的插入和删除示例，使得它们破坏外码约束。

插入的示例：

Inserting a tuple:

(10111, Ostrom, Economics, 110,000)

插入到老师表中，而department表中没有经济学学院，将违反外键约束。

删除的示例：

Deleting the tuple:

(Biology, Watson, 90000)

在advisor中，至少有一名学生或老师元组的部门名称为Biology，将违反外键约束。

2.3

考虑 *time_slot* 关系。假设一个特定的时间段可以在一周之内出现多次，解释为什么 *day* 和 *start_time* 是该关系主码的一部分，而 *end_time* 却不是。

属性日期和开始时间是主键的一部分由于特定的课程很可能会在不同的日子见面，甚至可能一天见面不止一次。 但是，结束时间不是主键的一部分，**因为在特定的一天中的某一个特定时间开始的课的结束时间不可能超过1个**

2.4

但一般来说，情况可能并不总是如此（除非大学有一条规则，即两名教师不能有相同的名字，这是一个相当不友好的情况）

2.7

employee(*person-name*, *street*, *city*)
works(person-name, *company-name*, *salary*)
company(*company-name*, *city*)

图 2-14　习题 2.1、习题 2.7 和习题 2.12 的关系数据库

写出下列的查询：

a. 找到所有住在“迈阿密”城市的员工的名字”。

b. 查找工资大于10万$的所有员工的姓名。

c.找到所有住在“迈阿密”的工资超过10万$的员工的名字。

```
select person-name
from employee
where city='Miami'
```

a. $\Pi_{name}\ (\sigma_{city = \text{“Miami”}}\ (employee))$

b. $\Pi_{name}\ (\sigma_{salary > 100000}\ (employee))$

c. $\Pi_{name}\ (\sigma_{city = \text{“Miami”} \wedge salary > 100000}\ (employee))$

c有错误：应该是先employee和works做自然连接，再筛选

2.8

branch(branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
loan (loan_number, branch_name, amount)
borrower (customer_name, loan_number)
account (account_number, branch_name, balance)
depositor (customer_name, account_number)

图 2-15　习题 2.8、习题 2.9 和习题 2.13 的银行数据库

a. 查找位于“芝加哥”的所有分支机构的名称。

b. 找到所有在“市中心”分行有贷款的借款人的名字。

Answer:

a. $\Pi_{branch\_name}\ (\sigma_{branch\_city = \text{“Chicago”}}\ (branch))$

b. $\Pi_{customer\_name}\ (\sigma_{branch\_name = \text{“Downtown”}}\ (borrower \bowtie loan))$

easy!

2.10 advisor的主码是s_id，如果一个学生可以有多个指导老师，那么主码是什么？

是s_id+i_id

2.12

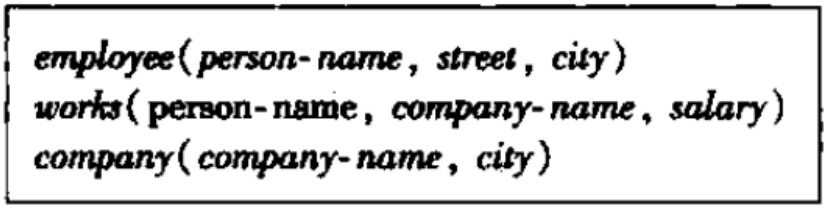

图 2-14　习题 2.1、习题 2.7 和习题 2.12 的关系数据库

写出以下的查询：

a. Find the names of all employees who work for “First Bank Corpora

tion”.

b. Find the names and cities of residence（居住城市） of all employees who work for

“First Bank Corporation”.

c. Find the names, street address, and cities of residence of all employees

who work for “First Bank Corporation” and earn more than $10,000.

a. $\Pi_{person\_name}(\sigma_{company\_name = \text{“First Bank Corporation”}}(works))$

b. $\Pi_{person\_name, city}(employee \bowtie (\sigma_{company\_name = \text{“First Bank Corporation”}}(works)))$

c. $\Pi_{person\_name, street, city}(\sigma_{(company\_name = \text{“First Bank Corporation”} \wedge salary > 10000)}(works \bowtie employee))$

Easy!

2.13

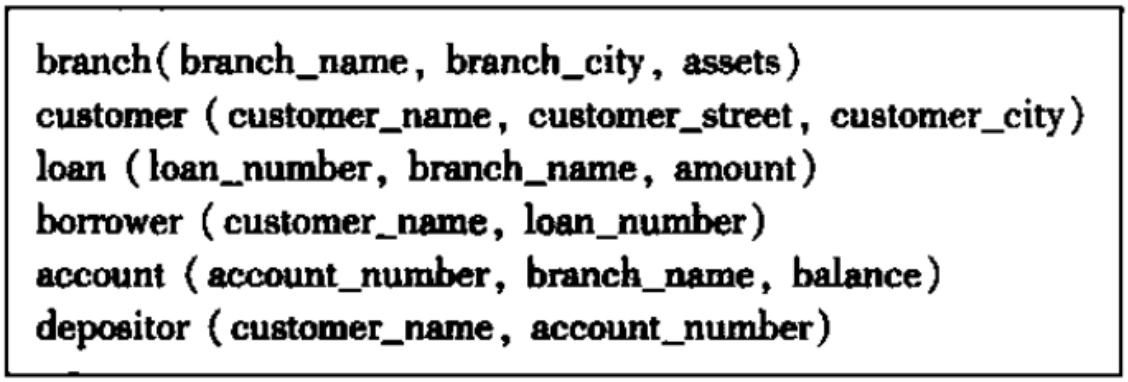

图 2-15　习题 2.8、习题 2.9 和习题 2.13 的银行数据库

a. Find all loan numbers with a loan value greater than $10,000.找出贷款额度超过10000美元的所有贷款号

b. Find the names of all depositors who have an account with a value

greater than $6,000. 查找所有存款人的姓名，他们的帐户价值大于$6,000。

c. Find the names of all depositors who have an account with a value

greater than $6,000 at the “Uptown” branch. 在“Uptown”分行找到所有持有价值大于6000$账户的储户的姓名。

a. $\Pi_{loan\_number}\ (\sigma_{amount>\ 10000}(loan)$

b. $\Pi_{customer\_name}\ (\sigma_{balance>\ 6000}\ (depositor \bowtie\ account))$

c. $\Pi_{customer\_name}\ (\sigma_{balance>\ 6000 \wedge branch\_name=\text{“Uptown”}}\ (depositor \bowtie\ account))$

2.14 列出再数据库中引入空值的两个原因

可以将null引入数据库，因为实际值**要么未知，要么不存在。**

例如，地址已更改、新地址尚不知道的雇员应 保留一个空地址。 如果员工元组有一个复合属性依赖项，而特定的员工没有依赖项，那么元组的依赖项属性应该被赋予一个NULL 价值

2.15 讨论过称化和非过程化语言的相对优点

非过程语言大大简化了查询的规范（至少，它们设计用于处理的查询类型）。 使用户不必担心查询的方式评估；这不仅减少了编程工作，而且实际上在大多数情况下，查询优化器可以比程序员WO更好地选择评估查询的最佳方法 通过试验和错误进行报复。 另一方面，程序语言在它们可以执行的计算方面要强大得多。 有些任务要么不是使用非过程语言完成，或者很难使用非过程语言表达，或者如果以非过程方式指定，则执行效率非常低

三、第三章

3.1 利用大学数据库写出以下的查询

a. 找出Comp.Sci系开设的有三个学分的课程名字

b. 找出所有由一位名叫Einstein的老师教的学生的ID；确保结果中没有重复。

c.找到教师的最高工资。

d.找到所有赚取最高工资的教师（可能有多个相同工资的教师）。

e.找出2009年秋季开设的每个课程段的选课人数

f.在2009年秋季找到所有部门的最大招生人数。

g.找出在2009年秋季拥有最多选课人数的课程段

```sql
select title
from course
where dept_name='Comp.Sci' and credits=3

select distinct student.ID
from (student join takes using(ID))join (instructor join teaches using (ID))
using (course_id,sec_id,semester,year)
where instructor.name='Einstein'

???advisor???

select max(salary)
from instructor

select ID,name
from instructor
where salary=(select max(salary) from instructor)

select course_id,sec_id,count(ID)
from section natural join takes
where semester='Autumn'
and year =2009
group by course_id,sec_id
(not using section???)


select max(enrollment)
from(select course_id,sec_id,count(ID)
      from section natural join takes
      where semester='Autumn'
      and year =2009
      group by course_id,sec_id)

with sec_enrollment as (
    select course_id, sec_id, count(ID) as enrollment
    from section natural join takes
    where semester = 'Autumn'
    and year = 2009
    group by course_id, sec_id)
select course_id, sec_id
from sec_enrollment
where enrollment = (select max(enrollment) from sec enrollment)
```

3.2

假设给你一个关系 *grade_points*(*grad_e*, *points*)，它提供从 *takes* 关系中用字母表示的成绩等级到数字表示的得分之间的转换。例如，“A”等级可指定为对应于 4 分，“A –”对应于 3.7 分，“B +”对应于 3.3 分，“B”对应于 3 分，等等。学生在某门课程(课程段)上所获得的等级分值被定义为该课程段的学分乘以该生得到的成绩等级所对应的数字表示的得分。

给定上述关系和我们的大学模式，用 SQL 写出下面的每个查询。为简单起见，可以假设没有任何 *takes* 元组在 *grade* 上取 *null* 值。

a. 根据 ID 为 12345 的学生所选修的所有课程，找出该生所获得的等级分值的总和。

b. 找出上述学生等级分值的平均值(GPA)，即用等级分值的总和除以相关课程学分的总和。

c. 找出每个学生的 ID 和等级分值的平均值。

即在大学数据库中新增了一个关系：grade_points

```
a.
select sum(credits*points)
from (takes natural join course)natural join grade_points
where id='12345'
```

但是如果一个学生没有参加任何课程，那么结果就没有任何远足，所以更好的表示是：使用左外连接，这里可以用：

```
(select sum(credits * points)
from (takes natural join course) natural join grade_points
where ID = '12345')
union
(select 0
from student
where takes.ID = '12345' and
not exists  ( select * from takes where takes.ID = '12345'))

b.
select sum(credits * points)/sum(credits) as GPA
from (takes natural join course) natural join grade_points
where ID = '12345'
```

这样的问题也是如果一个学生没有参加任何课程，就不会出现在结果里

```
select sum(credits * points)/sum(credits) as GPA
from (takes natural join course) natural join grade_points
where ID = '12345'
union
(select null as GPA
from student
where takes.ID = '12345' and
not exists ( select * from takes where takes.ID = '12345'))

c.
select ID, sum(credits * points)/sum(credits) as GPA
from (takes natural join course) natural join grade points
group by ID
```

更好的同样也是

```
union
(select ID, null as GPA
```

```
from student
where not exists ( select * from takes where takes.ID = student.ID))
```

3.3 写出SQL中的插入、删除、修改和更新语句

a. 增加Comp中每个讲师的工资10%。

b.删除所有从未提供过的课程（即没有出现在section中的）。

c.把每个在tot_cred属性上取值超过100的学生作为同系的教师插入，工资为10000美元

```
a.
update instructor
set salary=salary*1.10
where deot_name='Comp.Sci'

b.
delete from course
where course_id not in (select course_id
                        from section)

c.
insert into instructor
select ID,name,dept_name,10000
from student
where tot_cred>100
```

3.4

person (*driver_id*, *name*, *address*)
car (*license*, *model*, *year*)
accident (*report_number*, *date*, *location*)
owns (*driver_id*, *license*)
participated (*report_number*, *license*, *driver_id*, *damage_amount*)

图 3-18　习题 3.4 和习题 3.14 的保险公司数据库

a.找出2009年其车辆出过交通事故的人员总数

b.向数据库中增加一个新的事故，对每个必需的属性可以设定为任意值

c.删除"John Smith"拥有的马自达车

```
a.
select count(distinct name)
```

```
from accident natral join participated natual join person
where date between '2009-00-00' and date '2009-12-31'

b.
insert into accident values(4007,'2001-09-01','Berkeley')

c.
delete car
where model='Mazda' and license in(
select license
  from person natural join owns
  where name='J S'
)
```

3.5

假设有关系 *marks*(*ID*, *score*)，我们希望基于如下标准为学生评定等级：如果 *score* < 40 得 F；如果 40 ≤ *score* < 60 得 C；如果 60 ≤ *score* < 80 得 B；如果 80 ≤ *score* 得 A。写出 SQL 查询完成下列操作：

a. 基于 *marks* 关系显示每个学生的等级。

b. 找出各等级的学生数。

```
a.
select ID,
case
when score < 40 then 'F'
when score < 60 then 'C'
when score < 80 then 'B'
else 'A'
end
from marks

b.
with grades as
(
select ID,
case
when score < 40 then 'F'
when score < 60 then 'C'
when score < 80 then 'B'
else 'A'
end as grade
from marks
)
select grade, count(ID)
from grades
group by grade
```

3.6 复习一下like是有大小敏感的，但是字符串有一个lower()函数，可以用来实现大小写不敏感的匹配，用这个方式来实现一个查询：找出名称中包含了sci的系，并且忽略大小写

```
select dept name
from department
where lower(dept_name) like '%sci%'
```

3.7

考虑 SQL 查询

```
select distinct p. a1
from p, r1, r2
where p. a1 = r1. a1 or p. a1 = r2. a1
```

在什么条件下这个查询选择的 *p. a1* 值要么在 *r1* 中，要么在 *r2* 中？仔细考察 *r1* 或 *r2* 可能为空的情况。

这个查询选择p.a1的值，这些值等于r1.a1或r2.a1的某些值，当且仅当r1和r2都是非空的。 如果r1和r2的一个或两个都是空的，则p，r的笛卡尔积 1和r2是空的，因此查询的结果是空的。 当然，如果p本身是空的，则结果与预期的一样，即是 空的。

3.8 重点复习！！！

考虑图 3-19 中的银行数据库，其中加下划线的是主码。为这个关系数据库构造出如下 SQL 查询：

a. 找出银行中所有有账户但无贷款的客户。

b. 找出与“Smith”居住在同一个城市、同一个街道的所有客户的名字。

c. 找出所有支行的名称，在这些支行中都有居住在“Harrison”的客户所开设的账户。

branch (*branch_name*, *branch_city*, *assets*)
customer (*customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*customer_name*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*customer_name*, *account_number*)

图 3-19　习题 3. 8 和习题 3. 15 的银行数据库

```
a.
(select customer name
from depositor)
except
(select customer name
from borrower)

b.
select F.customer_name
from customer F join customer S using(customer_street, customer_city)
where S.customer_name = 'Smith'
```

```
c.
select distinct branch name
from account natural join depositor natural join customer
where customer city = 'Harrison'
```

3.9

考虑图 3-20 的雇员数据库，其中加下划线的是主码。为下面每个查询写出 SQL 表达式：

a. 找出所有为“First Bank Corporation”工作的雇员名字及其居住城市。

b. 找出所有为“First Bank Corporation”工作且薪金超过 10 000 美元的雇员名字、居住街道和城市。

c. 找出数据库中所有不为“First Bank Corporation”工作的雇员。

d. 找出数据库中工资高于“Small Bank Corporation”的每个雇员的所有雇员。

e. 假设一个公司可以在好几个城市有分部。找出位于“Small Bank Corporation”所有所在城市的所有公司。

f. 找出雇员最多的公司。

g. 找出平均工资高于“First Bank Corporation”平均工资的那些公司。

employee(*employee_name*, *street*, *city*)
works(*employee_name*, *company_name*, *salary*)
company(*company_name*, *city*)
managers(*employee_name*, *manager_name*)

图 3-20　习题 3.9、习题 3.10、习题 3.16、习题 3.17 和习题 3.20 的雇员数据库

```
a.
select e.employee_name, city
from employee e, works w
where w.company_name = 'First Bank Corporation' and
w.employee_name = e.employee_name

b.
select *
from employee
where employee_name in
(select employee_name
from works
where company_name = 'First Bank Corporation' and salary > 10000)

c.
select employee name
from employee
where employee name not in
(select employee name
from works
where company name = 'First Bank Corporation')

d.
select employee name
```

```
from works
where salary > all
(select salary
from works
where company name = 'Small Bank Corporation')
```
如果人们可能为几家公司工作，我们希望考虑每个人的总收入，问题就更复杂了。 它可以通过使用嵌套子查询来解决，现在用with子句来解决它。
```
with emp_total_salary as
(select employee_name, sum(salary) as total_salary
from works
group by employee_name
)
select employee_name
from emp_total_salary
where total_salary > all
(select total_salary
from emp_total_salary, works
where works.company_name = 'Small Bank Corporation' and
emp_total_salary.employee name = works.employee_name
)
```

e.
最简单的是使用contain
```
select T.company_name
from company T
where (select R.city
from company R
where R.company_name = T.company_name)
contains
(select S.city
from company S
where S.company_name = 'Small Bank Corporation')
```
更好的可以用
```
select S.company_name
from company S
where not exists ((select city
from company
where company_name = 'Small Bank Corporation')
except
(select city
from company T
where S.company_name = T.company_name))
```
这个就类似于选择了生物系开的所有课的学生---NOT EXITS+EXCEPT

f.
```
select company name
from works
group by company name
having count (distinct employee name) >= all
```

```
(select count (distinct employee name)
from works
group by company name)

g.
select company name
from works
group by company name
having avg (salary) > (select avg (salary)
from works
where company name = 'First Bank Corporation')
```

3.10

考虑图 3-20 的关系数据库，给出下面每个查询的 SQL 表达式：

a. 修改数据库使“Jones”现在居住在“Newtown”市。

b. 为“First Bank Corporation”所有工资不超过 100 000 美元的经理增长 10% 的工资，对工资超过 100 000 美元的只增长 3%。

```
update employee
set city = 'Newton'
where person name = 'Jones'

update works T
set T.salary = T.salary * (case
when (T.salary * 1.1 > 100000) then 1.03
else 1.1
)
where T.employee name in (select manager name
from manages) and
T.company name = 'First Bank Corporation'
```

3.11

使用大学模式，用 SQL 写出如下查询。

a. 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姓名。

b. 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的 ID 和姓名。

c. 找出每个系教师的最高工资值。可以假设每个系至少有一位教师。

d. 从前述查询所计算出的每个系最高工资中选出最低值。

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

```
b.
select id, name
from student
except
select id, name
from student natural join takes
where year < 2009

c.
select dept, max(salary)
from instructor
group by dept

d.
select min(maxsalary)
from (select dept, max(salary) as maxsalary
from instructor
group by dept)
```

3.12

使用大学模式，用 SQL 写出如下查询。

a. 创建一门课程“CS－001”，其名称为“Weekly Seminar”，学分为 0。

b. 创建该课程在 2009 年秋季的一个课程段，*sec_id* 为 1。

c. 让 Comp. Sci. 系的每个学生都选修上述课程段。

d. 删除名为 Chavez 的学生选修上述课程段的信息。

e. 删除课程 CS-001。如果在运行此删除语句之前，没有先删除这门课程的授课信息(课程段)，会发生什么事情?

f. 删除课程名称中包含“database”的任意课程的任意课程段所对应的所有 *takes* 元组，在课程名的匹配中忽略大小写。

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

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

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

d.
```

```
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')
```

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

如果我们试图直接删除课程，就会出现外键违反，因为部分对课程有一个foriegn键引用；同样，我们必须从takebefo中删除相应的元组　重新删除部分，因为从take到section有一个外键引用。 由于违反了外键，执行删除的事务将被回滚。

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

3.13 写出创建模式的DDL

```
create table person
(driver id varchar(50),
name varchar(50),
address varchar(50),
primary key (driver_id))
```

3.14

考虑图 3-18 中的保险公司数据库，其中加下划线的是主码。对这个关系数据库构造如下的 SQL 查询：

a. 找出和“John Smith”的车有关的交通事故数量。

b. 对事故报告编号为“AR2197”中的车牌是“AABB2000”的车辆损坏保险费用更新到 3000 美元。

person (*driver_id*, *name*, *address*)
car (*license*, *model*, *year*)
accident (*report_number*, *date*, *location*)
owns (*driver_id*, *license*)
participated (*report_number*, *license*, *driver_id*, *damage_amount*)

图 3-18　习题 3.4 和习题 3.14 的保险公司数据库

```
a.
select count (*)
from accident
where exists
(select *
from participated, owns, person
where owns.driver_id = person.driver_id
and person.name = 'John_Smith'
and owns.license = participated.license
and accident.report_number = participated.report_number)

b.
update participated
set damage_amount = 3000
where report_number = "AR2197" and
license = "AABB2000")
```

3.15

考虑图 3-19 中的银行数据库，其中加下划线的是主码。为这个关系数据库构造出如下 SQL 查询：

a. 找出在"Brooklyn"的所有支行都有账户的所有客户。

b. 找出银行的所有贷款额的总和。

c. 找出总资产至少比位于 Brooklyn 的某一家支行要多的所有支行名字。

branch(*branch_name*, *branch_city*, *assets*)
customer (*customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*customer_name*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*customer_name*, *account_number*)

Figure 3.19 Banking database for Exercises 3.8 and 3.15.

```
a.
with branchcount as
(select count(*)
branch
where branch city = 'Brooklyn')
select customer name
from customer c
where branchcount = (select count(distinct branch name)
from (customer natural join depositor natural join account
natural join branch) as d
where d.customer name = c.customer name)
这里也可以用之前的那个excepy+not exists 复习的时候自己写

b.
select sum(amount)
from loan

c.
select branch name
from branch
where assets > some
(select assets
from branch
where branch city = 'Brooklyn')
```

3.16

employee (*employee_name*, *street*, *city*)
works (*employee_name*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*employee_name*, *manager_name*)

Figure 3.20 Employee database for Exercises 3.9, 3.10, 3.16, 3.17, and 3.20.

e.找出工资总和最小的公司

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

f.为"F B C "的所有经理增长10%的工资

```
update works
set salary = salary * 1.1
where employee_name in (select manager_name
from manages)
and company name = 'First Bank Corporation'
```

3.21

a. 打印借阅了任意由“McGraw-Hill”出版的书的会员名字。
b. 打印借阅了所有由“McGraw-Hill”出版的书的会员名字。
c. 对于每个出版商，打印借阅了多于五本由该出版商出版的书的会员名字。
d. 打印每位会员借阅书籍数量的平均值。考虑这样的情况：如果某会员没有借阅任何书籍，那么该会员根本不会出现在 *borrowed* 关系中。

member(*memb_no*, *name*, *age*)
book(*isbn*, *title*, *authors*, *publisher*)
borrowed(*memb_no*, *isbn*, *date*)

图 3-21　习题 3. 21 的图书馆数据库

选择性：

```
b.
select distinct m.name
from member m
where not exists
((select isbn
```

```
from book
where publisher = 'McGrawHill')
except
(select isbn
from borrowed l
where l.memb_no = m.memb_no))

c. 重点复习
select publisher, name
from (select publisher, name, count (isbn)
    from member m, book b, borrowed l
    where m.memb_no = l.memb_no
    and l.isbn = b.isbn
    group by publisher, name) as
    membpub(publisher, name, count_books)
where count_books > 5

d.
with memcount as
(select count(*)
from member)
select count(*)/memcount
from borrowed
```

3.22不使用unique重写下面的where子句

```
where unique (select title from course)


where( (select count(title)
from course) =
(select count (distinct title)
from course))
```

3.24

考虑查询：

```
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, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

不使用 **with** 结构，重写此查询。


```
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, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```


```
select distinct dept_name_d
from instructor i
where
(select sum(salary)
from instructor
where department = d) >=(select avg(s)
from
(select sum(salary) as s
from instructor
group by department))
```

四、第四章

4.1

用 SQL 写出下面的查询：

a. 显示所有教师的列表，列出他们的 ID、姓名以及所讲授课程段的编号。对于没有讲授任何课程段的教师，确保将课程段编号显示为 0。在你的查询中应该使用外连接，不能使用标量子查询。

b. 使用标量子查询，不使用外连接写出上述查询。

c. 显示 2010 年春季开设的所有课程的列表，包括讲授课程段的教师的姓名。如果一个课程段有不止一位教师讲授，那么有多少位教师，此课程段在结果中就出现多少次。如果一个课程段没有任何教师，它也要出现在结果中，相应的教师名置为“—”。

d. 显示所有系的列表，包括每个系中教师的总数，不能使用标量子查询。确保正确处理没有教师的系。

```
a.
select ID, name,
count(course id, section id, year,semester) as 'Number of sections'
from instructor natural left outer join teaches
group by ID, name

b.
select ID, name, (select count(*) as 'Number of sections'
from teaches T where T.id = I.id)
from instructor I

d.
select dept name, count(ID)
from department natural left outer join instructor
group by dept name
```

4.2

不使用外连接来改写下列的查询

```
a.
select * from student natural left outer join takes
can be rewritten as:

select * from student natural join takes
union
select ID, name, dept_name, tot_cred, NULL, NULL, NULL, NULL, NULL
from student S1 where not exists
(select ID from takes T1 where T1.id = S1.id)

b.
select * from student natural full outer join takes
can be rewritten as:
```

```
(select * from student natural join takes)
union
(select ID, name, dept name, tot cred, NULL, NULL, NULL, NULL, NULL
from student S1
where not exists
  (select ID from takes T1 where T1.id = S1.id))
union
(select ID, NULL, NULL, NULL, course id, section id, semester, year, grade
from takes T1
where not exists
(select ID from student S1 whereT1.id = S1.id))
```

4.3

假设有三个关系 $r(A, B)$、$s(B, C)$和 $t(B, D)$，其中所有属性声明为非空。考虑表达式：

- **r natural left outer join (s natural left outer join t)**
- **(r natural left outer join s) natural left outer join t**

a. 给出关系 r、s 和 t 的实例，使得在第二个表达式的结果中，属性 C 有一个空值但属性 D 有非空值。

b. 在第一个表达式的结果中，上述模式中的 C 为空且 D 非空有可能吗？解释原因。

a. Consider r = (a,b), s = (b1,c1), t = (b,d). The second expression would give (a,b,NULL,d).

b. 在第一个表达式的结果中，D不可能不是空的，而C是空的，因为在子表达式的自然左外连接t中，C不可能是空的，而D是n 不是空的。 在整个表达式中，当且仅当某些元组发生时，C可以为空在s中没有匹配的B值。但是在这种情况下，D也将是空的。

4.12

对于图 4-11 中的数据库，写出一个查询来找到那些没有经理的雇员。注意一个雇员可能只是没有列出其经理，或者可能有 *null* 经理。使用外连接书写查询，然后不用外连接再重写查询。

employee (*employee_name*, *street*, *city*)
works (*employee_name*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*employee_name*, *manager_name*)

```
select employee_name
from employee natural left outer join manages
where manager_name is null

select employee_name
from employee e
where not exists
    (select employee_name
    from manages m
    where e.employee_name = m.employee_name and m.manager_name is not null)
```

4.13

在什么情况下，查询

select *
from *student* **natural full outer join** *takes* **natural full outer join** *course*

将包含在属性 *title* **上取空值的元组？**

例如：如果一个学生没有参加任何的课程，那么就会有这种空值的结果

还有一种是这个COURSE的名字就是空

4.16

如本章定义的参照完整性约束正好涉及两个关系。考虑包括如图 4-12 所示关系的数据库。假设我们希望要求每个出现在 *address* 中的名字必须出现在 *salaried_worker* 或者 *hourly_worker* 中，但不一定要求在两者中同时出现。

a. 给出表达这种约束的语法。

b. 讨论为了使这种形式的约束生效，系统必须采取什么行动。

salaried_worker (*name*, *office*, *phone*, *salary*)
hourly_worker (*name*, *hourly_wage*)
address (*name*, *street*, *city*)

图 4-12　习题 4.16 的雇员数据库

a. 我们在创建address这个表的时候，加入一个外码约束：**foreign key** (*name*) **references** *salaried worker* **or** *hourly worker*

b.每次插入一个元祖到ADDRESS里面的时候，就检查

4.17当一个经理例如Satoshi授予权限的时候，授权应当由经理完成，而不是由Satoshi完成

考虑由以下人员提供授权的情况：
用户Satoshi，而不是管理员角色。 如果我们撤销授权中本聪，例如因为中Satoshi离开公司，所有授权中本聪授予的款项也将被撤销，即使该赠予是为了工作不变的员工。如果授予是由经理角色完成的，则撤消来自聪不会导致这种级联撤销。在授权图方面，我们可以对待中本聪和角色经理作为节点。 如果补助金来自经理角色，则撤销聪的经理角色对经理的资助没有影响角色。

4.18

假定用户 *A* 拥有关系 *r* 上的所有权限，该用户把关系 *r* 上的查询权限以及授予该权限的权限授予给 public。假定用户 *B* 将 *r* 上的查询权限授予 A。这是否会导致授权图中的环？解释原因。

是的，确实会导致授权图中出现循环。 授予public导致了A to公众的优势。 给公共运营商的补助金向所有人提供授权，B已获得授权。 每项特权因此，必须在公众与所有人之间建立优势系统中的用户。 如果不这样做，那么用户将没有路径从根（DBA）。 并给出with授予选项，B可以授予选择在r到A上导致授权图中B到A的边缘。 从而，现在有一个从A to公众，从公众到B以及从B到B的循环一个

五、第五章

5.1 描述何种情况下你会选择使用嵌入式SQL，而不是用SQL或者某种通用的程序设计语言?

用SQL编写查询通常比编码容易得多通用编程语言中的相同查询。 然而并非所有查询都可以用SQL编写。 比如非声明性动作例如打印报告，与用户互动或发送结果无法从SQL内部完成对图形用户界面的查询。

在我们想要两全其美的情况下，我们可以选择嵌入式SQL或动态SQL，而不是单独使用SQL或仅使用通用编程语言。

嵌入式SQL的优点是程序不那么复杂因为它避免了ODBC或JDBC函数调用的混乱，但是需要专门的预处理器。

5.4

说明如何用触发器来保证约束“一位教师不可能在一个学期的同一时间段在不同的教室里教课”。（要知道，对关系 *teaches* 或 *section* 的改变都可能使该约束被破坏。）

5.5

写一个触发器，用于当 *section* 和 *time_slot* 更新时维护从 *section* 到 *time_slot* 的参照完整性约束。注意，我们在图 5-8 中写的触发器不包含更新操作。

```
create trigger timeslot_check1 after insert on section
    referencing new row as nrow
    for each row
    when (nrow.time_slot_id not in (
            select time_slot_id
            from time_slot)) /* time_slot 中不存在该 time_slot_id */
    begin
      rollback
    end;
    create trigger timeslot_check2 after delete on time slot
    referencing old row as orow
    for each row
    when (orow.time_slot_id not in (
            select time_slot_id
            from time_slot) /* 在 time_slot 中刚刚被删除的 time_slot_id */
    and orow.time_slot_id in (
            select time_slot_id
            from section)) /* 在 section 中仍含有该 time_slot_id 的引用 */
    begin
      rollback
    end;
```

图 5-8　使用触发器来维护参照完整性

5.7

考虑图 5-25 中的银行数据库。视图 *branch_cust* 定义如下：

```
create view branch_cust as
    select branch_name, customer_name
    from depositor, account
    where depositor.account_number = account.account_number
```

假设视图被物化，也就是，这个视图被计算且存储。写触发器来维护这个视图，也就是，该视图在对关系 *depositor* 或 *account* 的插入和删除时保持最新。不必管更新操作。

例如在更新时候的触发器：

```
define trigger insert_into_branch_cust_via_depositor
after insert on depositor
referencing new table as inserted for each statement
insert into branch_cust
    select branch_name, customer_name
    from inserted, account
    where inserted.account_number = account.account_number
```

即每次插入的时候都要考虑在这个视图中同步插入

define trigger *delete_from_branch_cust_via_depositor*
after delete on *depositor*
referencing old table as *deleted* **for each statement**
delete from *branch_cust*
select *branch_name, customer_name*
from *deleted, account*
where *deleted.account_number = account.account_number*

5.8

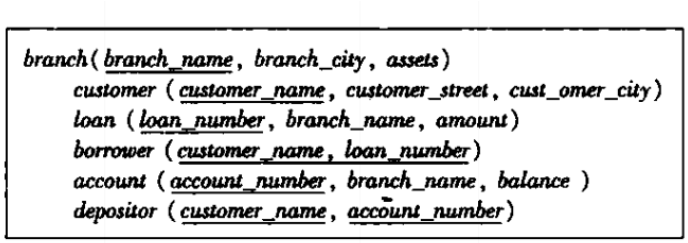

图 5-25　习题 5.7、习题 5.8 和习题 5.28 中用到的银行数据库

考虑图 5-25 中的银行数据库。写一个 SQL 触发器来执行下列动作：在对账户执行 delete 操作时，对账户的每一个拥有者，检查他是否有其他账户，如果没有，把他从 *depositor* 关系中删除。

```
create trigger check-delete-trigger after delete on account
referencing old row as orow
for each row
delete from depositor
where depositor.customer_name not in
    ( select customer_name from depositor
    where account_number <> orow.account_number )
end
```

5.10 重点复习

对于给定的一个关系 $S(student, subject, marks)$，写一个查询，利用排名操作找出总分数排在前 n 位的学生。

```
select student, sum(marks) as total,
    rank() over (order by (total) desc ) as trank
from S
groupby student
having trank ≤ n
```

5.15

考虑有两个关系的雇员数据库

employee(*employee_name*, *street*, *city*)
works(*employee_name*, *company_name*, *salary*)

这里主码用下划线标出。写出一个查询找出这样的公司，它的雇员的平均工资比“First Bank Corporation”的平均工资要高。

a. 使用合适的 SQL 函数。

b. 不使用 SQL 函数。

```
a.
create function avg_salary(cname varchar(15))
returns integer
declare result integer;
select avg(salary) into result
from works
where works.company_name = cname
return result;
end
select company_name
from works
where avg_salary(company_name) > avg_salary("First Bank Corporation")

b.
select company_name
from works
group by company_name
having avg(salary) > (select avg(salary)
    from works
    where company_name="First Bank Corporation")
```

5.16

使用 **with** 子句而不是函数调用来重写 5. 2. 1 节的查询，返回教师数大于 12 的系的名称和预算。

5.2.1节中的查询：

门想要这样一个函数：给定一个系的名字，返回
数目。我们可以如图 5-5 所示定义函数。这个
在返回教师数大于 12 的所有系的名称和预算

```
select dept_name, budget
from department
where dept_count(dept_name) > 12;
```

```
create function dept_count(dept_name varchar(20))
    returns integer
    begin
    declare d_count integer;
        select count(*) into d_count
        from instructor
        where instructor.dept_name= dept_name
    return d_count;
    end
```

现在写成：

```
with instr_count (dept_name, number) as
(select dept_name, count (ID)
from instructor
group by dept_name)
select dept_name, budget
from department, instr_count
where department.dept_name = instr_count.dept_name
and number > 12
```

5.17

把使用嵌入式 SQL 与在 SQL 中使用定义在一个通用程序设计语言中的函数这两种情况进行比较。在什么情况下你会考虑用哪个特性?

SQL函数主要是一种扩展SQL处理复杂数据类型（例如图像）属性的能力，或者执行复杂和非标准的操作。 当执行必要的操作（例如显示结果和与之交互）时，嵌入式SQL很有用需要用户。 仅在SQL中无法方便地完成这些操作环境。 通过检索数据然后在SQL上执行函数的操作，可以使用嵌入式SQL代替SQL函数。结果。 但是缺点是很多查询评估功能可能最终会在宿主语言代码中重复出现。

5.21

假设有两个关系 r 和 s，r 的外码 B 参照 s 的主码 A。描述如何用触发器实现从 s 中删除元组时的 on delete cascade 选项。

我们为每个关系定义触发器，这些关系的主键由某个其他关系的外键引用。 触发将是从引用关系中删除元组时，将激活该函数的触发器执行的操作将是访问所有引用关系，并删除其中所有其外键属性值的元组与已删除的元组中的主键属性值相同参照关系。 这组触发器将处理删除级联操作

5.24

对 SQL 聚集函数 sum、count、min 和 max 中的每一个，说明在给定多重集合 S_1 和 S_2 上的聚集值的条件下，如何计算多重集合 $S_1 \cup S_2$ 上的聚集值。

在上述的基础之上，在给定属性 $T \supseteq S$ 上的分组聚集值的条件下，对下面的聚集函数，给出计算关系 $r(A, B, C, D, E)$ 的属性的子集 S 上的分组聚集值的表达式。

a. sum、count、min 和 max。

b. avg。

c. 标准差。

Answer: Given aggregate values on multisets S_1 and S_2, we can calculate the corresponding aggregate values on multiset $S_1 \cup S_2$ as follows:

- $\textbf{sum}(S_1 \cup S_2) = \textbf{sum}(S_1) + \textbf{sum}(S_2)$
- $\textbf{count}(S_1 \cup S_2) = \textbf{count}(S_1) + \textbf{count}(S_2)$
- $\textbf{min}(S_1 \cup S_2) = \textbf{min}(\textbf{min}(S_1), \textbf{min}(S_2))$
- $\textbf{max}(S_1 \cup S_2) = \textbf{max}(\textbf{max}(S_1), \textbf{max}(S_2))$

Let the attribute set $T = (A, B, C, D)$ and the attribute set $S = (A, B)$. Let the aggregation on the attribute set T be stored in table *aggregation_on_t* with aggregation columns *sum_t*, *count_t*, *min_t*, and *max_t* storing **sum, count, min** and **max** resp.

a. The aggregations *sum_s*, *count_s*, *min_s*, and *max_s* on the attribute set S are computed by the query:

```
select A, B, sum(sum_t) as sum_s, sum(count_t) as count_s,
            min(min_t) as min_s, max(max_t) as max_s
from aggregation_on_t
groupby A, B
```

b. The aggregation *avg* on the attribute set S is computed by the query:

```
select A, B, sum(sum_t)/sum(count_t) as avg_s
from aggregation_on_t
groupby A, B
```

c. For calculating standard deviation we use an alternative formula:

$$stddev(S) = \frac{\sum_{s \in S} s^2}{|S|} - avg(S)^2$$

which we get by expanding the formula

$$stddev(S) = \frac{\sum_{s \in S} (s^2 - avg(S))^2}{|S|}$$

If S is partitioned into n sets $S_1, S_2, \ldots S_n$ then the following relation holds:

$$stddev(S) = \frac{\sum_{S_i} |S_i|(stddev(S_i)^2 + avg(S_i)^2)}{|S|} - avg(S)^2$$

Using this formula, the aggregation **stddev** is computed by the query:

select *A*, *B*,
(**sum**(*count_t* $*$ (*stddev_t**stddev_t+ *avg_t** *avg_t*))/**sum**(*count_t*)) -
(**sum**(*sum_t*)/**sum**(*count_t*))
from *aggregation_on_t*
groupby *A*, *B*