

注：每道题可能会有多种正确答案，本答案未覆盖所有正确答案。供参考。

## 第一章作业答案

**1.1 这一章讲述了数据库系统的几个主要的优点。它有哪些不足之处**

下面列出了与数据库系统相关的两个缺点。

- a. 建立数据库系统需要更多的知识、金钱、技能和时间。
- b. 数据库的复杂性可能导致性能不佳。

**1.4 假设你想要建立一个类似 YouTube 的视频站点。考虑 1.2 节中列出的将数据保存在文件系统各个缺点，讨论每一个缺点与存储实际的视频数据和关于视频的元数据（诸如标题、上传它的用户、标签、观看它的用户）之间的关联。**

①数据冗余和不一致。这在某种程度上与元数据有关，尽管与没有更新的实际视频数据无关。这里的关系非常少，没有一个会导致冗余。

②数据访问困难，如果视频数据只能通过几个预定义的接口访问，就像今天在视频共享网站中所做的那样，这不会是一个问题。然而，如果组织需要根据特定的搜索条件（除了简单的关键字查询）来查找视频数据，如果元数据存储的文件中，不编写应用程序就很难找到相关数据。对于查找数据的任务来说，使用数据库是很重要的。

③数据隔离。由于数据通常不是更新的，而是新创建的，所以数据隔

离不是主要问题。

④完整性问题。除了主键之外，这个应用程序中似乎不太可能存在明显的完整性约束。如果数据是分布式的，那么在执行主键约束时可能会出现問題。完整性問題可能不是一个大問題。

⑤原子性问题。上传视频时，应该自动添加视频和视频的元数据，否则数据会不一致。需要一种底层恢复机制来确保发生故障时的原子性。

⑥并发访问异常。由于数据没有更新，因此不太可能发生并发访问异常。

⑦安全问题。如果系统支持授权，这将是一个问题。

### **1.5 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方面的主要差异**

在 Web 中使用的查询是通过提供一个没有特定语法的关键字列表来指定的。结果通常是一个有序的 url 列表，以及关于 url 内容的信息片段。相反，数据库查询具有特定的语法，允许指定复杂的查询。在关系世界中，查询的结果总是一个表。

### **1.7 列出文件处理系统和 DBMS 的四个主要区别**

数据库管理系统和文件处理系统之间的一些主要区别是：

① 这两个系统都包含一组数据和一组访问该数据的程序。数据库管理系统协调数据的物理访问和逻辑访问，而文件处理系统只协调

物理访问。

- ② 数据库管理系统通过确保一个物理数据对所有授权访问它的程序可用来减少数据的重复量，而文件处理系统中一个程序写入的数据可能无法被另一个程序读取。
- ③ 数据库管理系统的设计是为了方便灵活地存取数据(即，查询)，而文件处理系统被设计成允许预先确定的数据访问(即编译程序)。
- ④ 数据库管理系统旨在协调多个用户同时访问相同的数据。文件处理系统通常被设计成允许一个或多个程序同时访问不同的数据文件。在文件处理系统中，只有当两个程序都对文件具有只读访问权限时，两个程序才能同时访问一个文件。

### **1.8 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方案而不需要重写应用程序的能力。逻辑层次的使用不需要知道物理层次结构的复杂性

### **1.9 列出数据库管理系统的五个职责。对于每个职责，说明当它不能被履行时会产生什么样的问题**

通用数据库管理器 (DBM) 有五个职责：

- a. 与文件管理器的交互
- b. 完整执行
- c. 安全执法
- d. 备份和恢复

e. 并发控制

如果一个给定的 DBM 没有满足这些职责(文中指出, 有时一个职责被设计忽略了, 例如在一个微型计算机的单用户 DBM 上的并发控制), 就会分别发生以下问题:

- a. 如果没有文件管理器的交互, 那么存储在文件中的任何东西都不能被检索到, 那么没有 DBM 可以没有这个。
- b. 一致性约束可能得不到满足, 账户余额可能低于允许的最低限度, 员工可能加班收入过多(例如, 小时数超过 80), 或者, 航空公司飞行员可能飞行时间超过法律允许的时间。
- c. 可能存在未授权用户或被授权用户访问数据库的情况, 数据库的一部分可能能够访问数据库中它们没有权限的部分。例如, 一个高中生可以获得国防密码, 或者雇员可以知道他们的主管的收入。
- d. 数据可能会永久丢失, 而不是至少以故障前存在的一致状态可用。
- e. 尽管在每个事务中执行了适当的完整性, 但一致性约束可能被违反。例如, 由于同时取款和存款, 可能会反映不正确的银行余额, 等等。

1.10 请给出至少两种理由说明为什么数据库系统使用诸如 SQL 这样的声明式查询语言来支持数据操作, 而不是只提供 C 或者 C++的函数库来执行数据操作。

①基于数据库管理系统的数据库使用方便

②应用程序开发效率高。

1.11 假设两名学生试图注册同一门课程，而该课程仅余一个开放名额。数据库系统的什么部件会防止将这最后一个名额同时分配给这两名学生？

事务管理器 并发控制机制

1.12 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？

在两层体系结构中，应用程序驻留在客户机上，通过查询语言表达式来调用服务器上的数据库系统功能。

而在一个三层体系结构中，客户机只作为一个前端并且不包含任何直接的数据库调用。客户端通常通过一个表单界面与应用服务器进行通信。而应用服务器与数据库系统通信与访问数据。

对 Web 应用来说，显然使三层体系结构更好。因为 Web 应用的访问量很大，客户机直接通过查询语言与数据库系统进行交互可能会出现阻塞（访问量太大），数据更新不及时（高并发引起），数据丢失（大数据量）等问题，通过一个应用服务器，我们可以进行负载均衡、分发等设置，由此来缓解数据库系统的压力。

1.15 描述可能被用于存储一个社会网络系统如 Facebook 中的信息的至少 3 个表

User(ID, name, age, ...)

Friendship(user1ID, user2ID, time, ...)

User\_generated\_content(ID, time, location, content, ...)

## 第二章作业答案

2.1 请考虑图 2-10 的职员数据库。这些关系上适当的主码是什么？

person-name;

person-name;

company-name.

2.2 请考虑从 instructor 的 dept\_name 属性到 department 关系的外码约束。请给出这些关系的插入和删除示例, 使得它们破坏该外码约束。

插入示例: (10111, Ostrom, Economics, 110000)

如果系表中没有经济系, 将违反外键约束。

删除示例: (Biology, Watson, 90000)

只有一个学生或每个学生的 dept\_name 为生物系, 将取消外键约束。

2.3 请考虑 time\_slot 关系。假设一个特定的时间片可以在一周之内出现不止一次, 请解释为什么 day 和 start\_time 是该关系主码的一部分, 而 end\_time 却不是。

属性 day 和 start\_time 是主键的一部分, 因为一个特定的班级很可能在好几天见面, 甚至可能一天见面不止一次。然而, end\_time 不是主键的一部分, 因为在特定日期的特定时间开始的特定班级不能在多个时间结束。

2.4 在图 2-1 所示的 instructor 示例中, 没有两位教师同名。我们

是否可以据此断定 name 可以用来作为 instructor 的超码（或主码）不。对于教员表的这个可能的实例，名称是惟一的，但通常情况下可能并不总是这样（除非大学有一个规则，两个教员不能有相同的名称，这是一个非常不可能的情况）。

## 2.5 请解释术语关系和关系模式之间在意义上的区别。

关系模式是一个类型定义，而关系是该模式的一个实例。例如，student (ss#, name) 是一个关系模式

ss#	name
123-45-6789	Tom Jones
456-78-9123	Joe Brown

是基于该模式的关系。

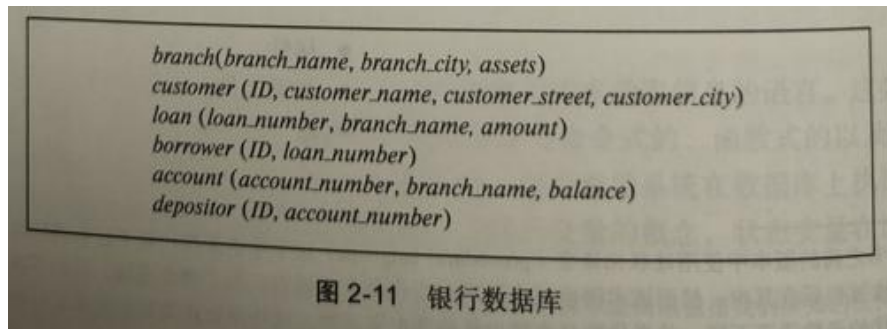
2.6 请考虑图 2-9 中模式图所示的 advisor 关系，advisor 的主码是 s\_id。假设一名学生可以有不止一位指导教师。那么，s\_id 还是 advisor 关系的主码吗？如果不是，那么 advisor 的主码应该是什么呢？

不能，s\_id 不再是 advisor 的主码，因为可能存在多个元组有着相同的 s\_id，此时 s\_id 不能用来区别不同的元组。Advisor 的主码应该是 s\_id, i\_id

2.7 请考虑图 2-11 的银行数据库。假设支行名称和客户姓名能够唯一标识出支行和客户，但是贷款账户可以与多位客户相关联。

a. 适当的主码是什么？

b. 请给出你选择的主码, 并确定适当的外码。



1. Branch 主码: branch\_name
2. Customer 主码: ID 或 customer\_name
3. Loan 主码: loan\_number 外码: branch\_name
4. Borrower 主码: ID+loan\_number 外码: ID, loan\_number
5. Account 主码: account\_number 外码: branch\_name
6. Depositor 主码: ID+account\_number 外码: ID, account\_number

2.8 请为图 2-11 的银行数据库构建模式图

参考 2.7 的主码外码



### 第三章作业答案

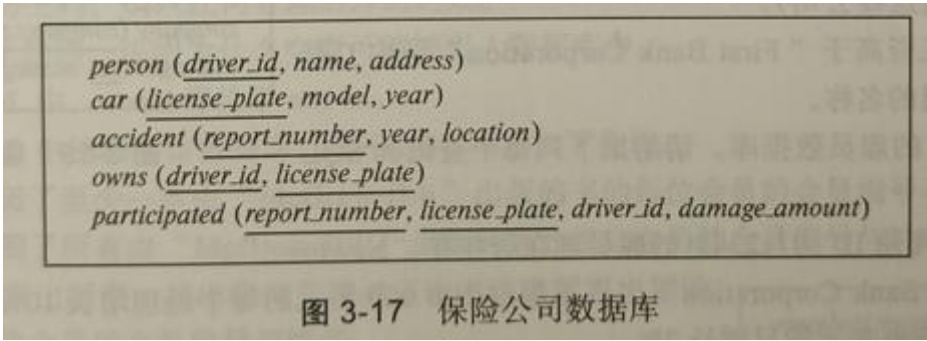
3.4 考虑图 3-17 中的保险公司数据库，其中加下划线的是主码。为这个关系数据库构造出如下 SQL 查询：

a. 找出 2017 年其车辆出过交通事故的人员总数。

```
select    count (distinct person.driver_id)  
from      accident, participated, person, owns  
where     accident.report_number = participated.report_number  
           and owns.driver_id = person.driver_id  
           and owns.license_plate = participated.license_plate  
           and year = 2017
```

b. 删除 ID 为 '12345' 的人拥有的年份为 2010 的所有汽车。

```
delete car  
where year = 2010 and license_plate in  
      (select license_plate  
       from owns o  
       where o.driver_id = '12345')
```



```
person (driver_id, name, address)  
car (license_plate, model, year)  
accident (report_number, year, location)  
owns (driver_id, license_plate)  
participated (report_number, license_plate, driver_id, damage_amount)
```

图 3-17 保险公司数据库

3.8 考虑图 3-18 中的银行数据库，其中的主码被加了下划线。请为这个关系数据库构造出如下 SQL 查询：

a. 找出银行中有账户但无贷款的每位客户的 ID。

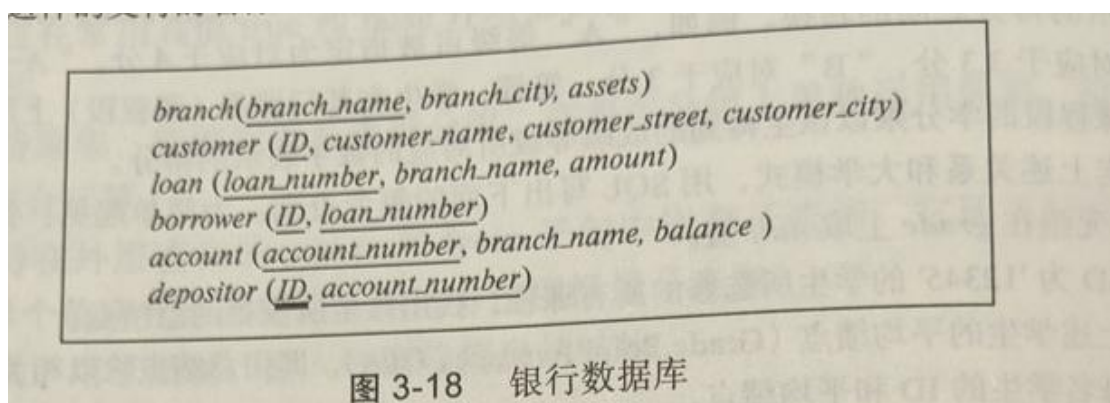
```
(select ID
from depositor)
except
(select ID
from borrower)
```

b. 找出与客户'12345'居住在同个城市、同一个街道的每位客户的 ID。

```
select F.ID
from customer as F, customer as S
where F.customer_street = S.customer_street
and F.customer_city = S.customer_city
and S.customer_id = '12345'
```

c. 找出每个这样的支行的名称：在这些支行中至少有一位居住在 Harrison 的客户开设了账户。

```
select distinct branch_name
from account, depositor, customer
where customer.id = depositor.id
and depositor.account_number = account.account_number
and customer_city = 'Harrison'
```



3.9 考虑图 3-19 的雇员数据库，其中的主码被加了下划线。请为下面的每个查询与出 SQL 表达式：

a. 找出为 “First Bank Corporation” 工作的每位雇员的 ID、姓名及所居住城市。

```
select e.ID, e.person_name, city  
from employee as e, works as w  
where w.company_name = 'First Bank Corporation' and  
       w.ID = e.ID
```

b. 找出为 “First Bank Corporation” 工作且工资超过 10000 美元的每位雇员的 ID、姓名及所居住城市。

```
select *  
from employee  
where ID in  
      (select ID  
       from works  
       where company_name = 'First Bank Corporation' and salary > 10000)
```

c. 找出没为 “First Bank Corporation” 工作的每位雇员的 ID。

```
select ID  
from works  
where company_name != 'First Bank Corporation'
```

d. 找出工资高于 “Small Bank Corporation” 的所有雇员的每位雇员的 ID。

```
select ID  
from works  
where salary > all  
      (select salary  
       from works  
       where company_name = 'Small Bank Corporation')
```

e. 假设一家公司可以位于好几个城市。找出位于 “Small Bank Corporation ” 所有所在城市的每家公司的名称。

```
select S.company_name
from company as S
where not exists ((select city
                   from company
                   where company_name = 'Small Bank Corporation')
except
(select city
 from company as T
 where S.company_name = T.company_name))
```

f. 找出雇员最多的公司名称（如果雇员最多的不止一家, 则全部列出这些公司）

```
select company_name
from works
group by company_name
having count (distinct ID) >= all
      (select count (distinct ID)
       from works
       group by company_name)
```

g. 找出平均工资高于 “ First Bank Corporation ” 平均工资的每家公司的名称。

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

```

employee (ID, person_name, street, city)
works (ID, company_name, salary)
company (company_name, city)
manages (ID, manager_id)

```

图 3-19 雇员数据库

3. 10 考虑图3-19的雇员数据库。请给出下列每个查询的SQL表达式：

a. 修改数据库使 ID 为' 12345' 的雇员现在居住在 “Newtown” 。

```

update employee
set city = 'Newtown'
where ID = '12345'

```

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

```

update works T
set T.salary = T.salary *1.03
where T.id in (select manager_id from manages) and
T.salary >100000 and
T.company_name = 'First Bank Corporation'

```

```

update works T
set T.salary = T.salary *1.1
where T.id in (select manager_id from manages) and
T.salary <= 100000 and
T.company_name = 'First Bank Corporation'

```

3. 14 考虑图 3-17 中的保险公司数据库，其中主码被加了下划线。请

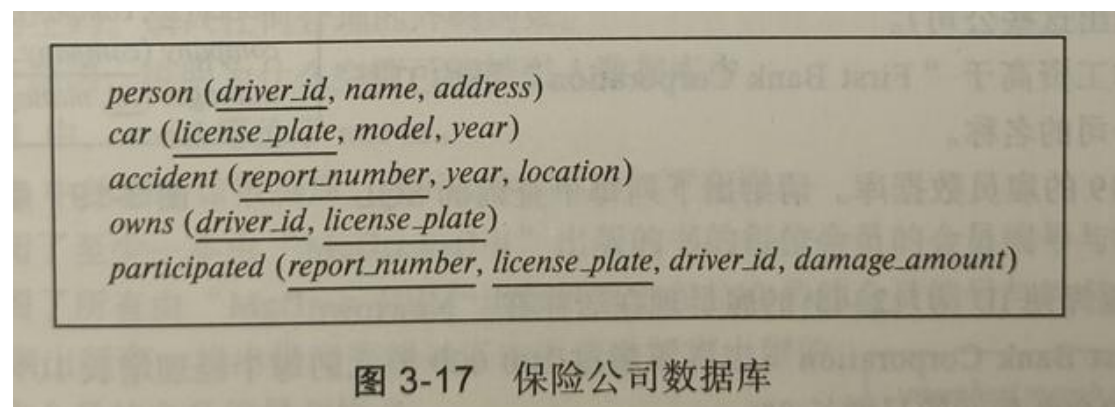
对这个关系数据库构造如下 SQL 查询。

a. 找出和名为 “John Smith” 的人的车有关的交通事故数量。

```
select count(distinct p1.report_number)
from person p, owns o, participated p1
where p.name = 'John Smith' and p.driver_id=o.driver_id and o.license=p1.license
```

b. 将编号为 “AR2197” 的事故报告中、车牌是 “AABB2000” 的车辆  
的损失额度更新为 3000 美元。

```
update participated
set damage_amount=3000
where report_number='AR2197' and license='AABB2000'
```



3. 15 考虑图 3-18 中的银行数据库, 其中主码被加了下划线。请对这个关系数据库构造如下 SQL 查询。

a. 找出在位于 “Brooklyn” 的所有支行都有账户的每位客户。

```
select distinct d.customer_name
from depositor d
where not exists
((select branch_name
from branch
```



```

where branch_city='Brooklyn')
except
(select branch_name
from depositor d2, account a
where d2.account_number=a.account_number and
d2.customer_name=d.customer_name

```

b. 找出银行的所有贷款额的总和。

```

select sum(amount)
from loan

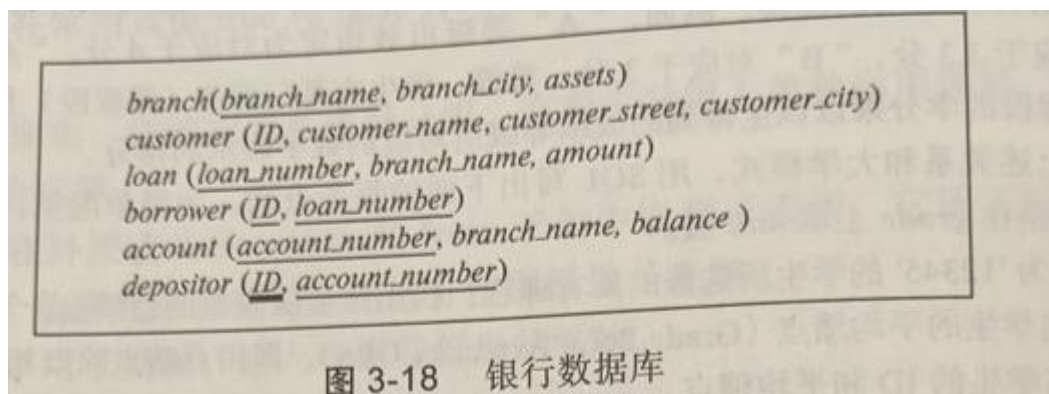
```

c. 找出资产比位于“Brooklyn”的至少一家支行要多的所有支行名称。

```

select distinct b1.branch_name
from branch b1, branch b2
where b1.assets>b2.assets and b2.branch_city='Brooklyn'

```



3. 16 考虑图 3-19 中的雇员数据库，其中主码被加了下划线。请给出下面每个查询的 SQL 表达式。

a. 找出每位这样的雇员的 ID 和姓名：该雇员所居住的城市与其工作的公司所在城市一样。

```

select e.id,e.person_name
from employee e, works w, company c

```

where e.id=w.id and w.company\_name=c.company\_name and e.city=c.city

b. 找出所居住的城市和街道与其经理相同的每位雇员的 ID 和姓名。

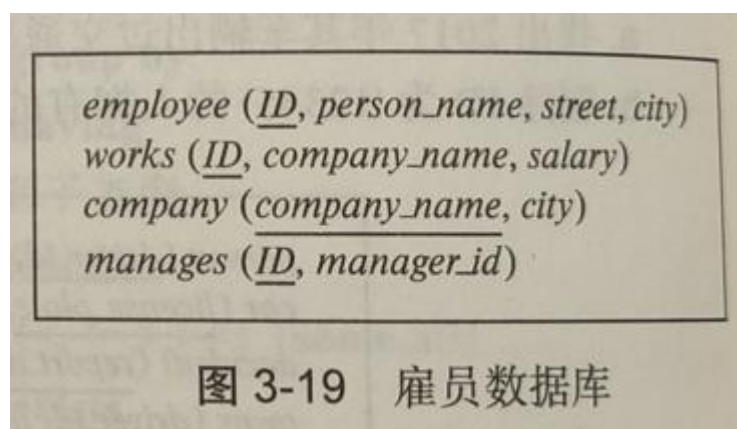
```
select e1.id,e1.person_name
from employee e1,employee e2,manages m
where e1.id = m.id and e2.id = m.manager_id and e1.street = e2.street and e1.city =
e2.city
```

c. 找出工资高于其所在公司所有雇员平均工资的每位雇员的 ID 和姓名。

```
select w1.id,e1.person_name
from works w1,employee e1
where w1.id = e1.id and salary >
(select avg(salary)
from works w2
where w1.company_name=w2.company_name)
```

d. 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)
```



3. 17 考虑图 3-19 中的雇员数据库。请给出下面每个查询的 SQL 表达



式。

a. 为 “First Bank Corporation” 的所有雇员增长 10% 的工资。

```
update works
set salary = salary * 1.1
where company_name='First Bank Corporation'
```

b. 为 “First Bank Corporation” 的所有经理增长 10% 的工资。

```
update works
set salary = salary * 1.1
where company_name='First Bank Corporation' and id in (select manager_id from
manages)
```

c. 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'
```

### 3.20 证明在 SQL 中， $\langle \rangle$ all 等价于 not in

在 SQL 中， $\langle \rangle$ all 即表示与全部元组均不相符，not in 表示在全部元组中没有相匹配的，故  $\langle \rangle$ all 与 not in 等价

### 3.21 考虑图 3-20 中的图书馆数据库。请用 SQL 写出如下查询。

a. 找出借阅了至少一本由 “McGraw-Hill” 出版的书的每位会员的会员编号与姓名。

```
select m.name
from member m, book bk, borrowed br
where m.memb_no = br.memb_nb and bk.isbn = br.isbn and
br.publisher='Mcgraw-Hill'
```

b. 找出借阅了所有由“McGraw-Hill”出版的书的每位会员的会员编号与姓名。

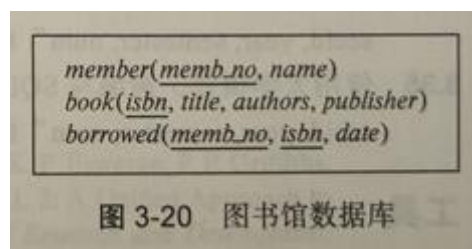
```
select m.name
from member m
where not exists
((select isbn
from book
where publisher='Mcgraw-Hill')
except
(select isbn
from borrowed
where memb_no=m.memb_no)
)
```

c. 对于每家出版商, 找出借阅了超过五本由该出版商出版的书的每位会员的会员编号与姓名。

```
select m.memb_no, m.name
from member m, borrowed br, book bk
where m.memb_no=br.memb_no and bk.isbn=br.isbn
group by publisher, m.memb_no, m.name
having count(isbn)>=5
```

d. 找出会员借阅书籍的平均数量。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, 但该会员仍应参与平均运算。

```
select count(isbn)/(select count(*) from member)
from borrowed
```



## 第四章作业答案

4.1 请考虑以下 SQL 查询，该查询旨在查找 2017 年春季讲授的所有课程的标题以及教师的姓名的列表

```
select name,title  
  
from instructor natural join teaches natural join section natural join course  
  
where semester = 'Spring' and year = 2017
```

请问这个查询有什么问题？

虽然查询在语法上是正确的，但它并不计算预期的答案，因为 dept\_name 是课程和讲师的属性。自然结合的结果是，只有当教师在她或他自己的部门教授一门课程时，成绩才会显示出来。

4.2 用 SQL 写出下面的查询

A 请显示所有教师的列表，展示每位教师的 ID 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。

你的查询应该使用外连接，且不能使用子查询

```
select ID, count(sec_id) as Number_of_sections  
from instructor natural left outer join teaches  
group by ID
```

B 请使用标量子查询且不使用外连接来写出 a 中的查询

```

select ID,
       (select count(*) as Number_of_sections
        from teaches T where T.id = I.id)
from instructor I

```

C 请显示 2018 年春季开设的所有课程段的列表，包括讲授课程段的每位教师的 ID 和姓名。如果一个课程段有不只一位教师讲授，那么有多少位教师，此课程段在结果中就出现多少次。如果一个课程段并没有任何教师，它也要出现在结果中，相应的教师姓名被置为 ‘-’

```

select course_id, sec_id, ID,
       decode(name, null, '-', name) as name
from (section natural left outer join teaches)
     natural left outer join instructor
where semester='Spring' and year= 2018

```

D 请显示所有系的列表，包括每个系中教师的总数，不能使用子查询。请确保显示没有教师的系，并用教师计数为零的方式来列出这些系

```

select dept_name, count(ID)
from department natural left outer join instructor
group by dept_name

```

4.7 请考虑图 4-12 中的员工数据库。请给出这个数据库的 SQL DDL 定义。请指出应该保持的引用完整性约束。并把它们包括在 DDL 定义中。

```

create table employee
(
  ID varchar(5),
  person_name varchar(20),
  street varchar(20),

```

```
city varchar(20),  
primary key (ID)  
)
```

```
create table works  
(  
ID varchar(5),  
company_name varchar(20),  
salary numeric(8,2),  
primary key (ID),  
foreign key (ID) references employee(ID)  
)
```

```
create table company  
(company_name varchar(20),  
city varchar(20),  
primary key (company_name))  
foreign key (company_name) references works(company_name)  
)
```

```
create table manages  
(  
ID varchar(5),  
manager_id varchar(5),  
primary key (ID),  
foreign key (ID) references employee(ID),  
foreign key (manager_id) references employee(ID)  
)
```

Note that alternative datatypes are possible. Other choices for **not null** attributes may be acceptable.

4. 10 给定关系 a(name, address, title) 和 b(name, address, salary), 请展示如何使用带 on 条件的全外连接( full outer-join)运算来表达 a 自然全外连接 (natural full outer join)b, 而不使用自然连接 (natural join) 语法。这可以使用合并( coalesce)运算来完成。请

确保结果关系不包含 name 和 address 属性的两个拷贝, 并且即使在  $\alpha$  和  $b$  中的某些元组对于 name 或 address 属性取空值的情况下, 所给出的解决方案也是正确的。

```
select coalesce(a.name, b.name) as name,  
coalesce(a.address, b.address) as address, a.title, b.salary  
from a full outer join b on a.name = b.name and a.address = b.address
```

4. 15 请重写查询:

```
select *  
  
from section natural join classroom
```

不使用自然连接, 而是使用具有 using 条件的内连接

```
select *  
from section join classroom  
using (building,room_number)
```

4. 16 请使用大学模式来编写一个 SQL 查询, 以查找从未在大学上过课的每名学生的 ID。不使用子查询和集合运算 (使用外连接) 来执行此操作。

```
select student.id  
from student natural left outer join takes  
where course_id is null
```

```

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)

```

图 2-8 大学数据库的模式

4.18 对于图 4-12 中的数据库，请编写一个查询来找出没有经理的每位员工的 ID。请注意，一位员工可能只是没有列出经理，或者可能经理值为空。请使用外连接来编写你的查询，然后根本不使用外连接再重写查询。

```

select e.id
from employee e natural left outer join manages m
where m.manager_id is null

```

```

select e.id
from employee e
where not exists(
select *
from manager m
where m.id = e.id and m.manager_id is not null
)

```

也可以用集合运算：

```

(select e.id
from employee e)
except
(select m.id
from manages m
where m.manager_id is not null)

```

```
employee (ID, person_name, street, city)  
works (ID, company_name, salary)  
company (company_name, city)  
manages (ID, manager_id)
```

图 4-12 员工数据库

4. 20 请说明如何定义视图 `tot_credits(year, num_credits)`, 它给出每年所修的总学分数。

```
create view tot_credits(year,num_credits)  
as  
(select year,sum(credits)  
from takes natural join course  
group by year);
```