姓名：胡瑞康

学号：22336087

Fig.0大学数据库

```
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. 使用 Fig.0 的大学模式（university schema）编写的 SQL 查询：

a) 查找每位至少选修过一门计算机科学课程的学生的 ID 和姓名，确保结果中没有重复的姓名。
首先从学生表`student`中选择学生的`id`和`name`。
通过内连接`takes`表，可以关联学生和他们所选的课程。
接着，通过内连接`course`表，可以获取到课程的详细信息。
最后，通过`where`语句过滤出那些课程部门为"computer science"（计算机科学）的课程。
使用`distinct`确保查询结果中学生姓名不重复。

```sql
select distinct student.id, student.name
from student
join takes on student.id = takes.id
join course on takes.course_id = course.course_id
where course.dept_name = 'computer science';
```

b) 查找每位未选修过 2017 年之前开设的任何课程的学生的 ID 和姓名。
首先从学生表`student`中选取所有学生的`id`和`name`。
`where not exists`子句用于排除那些在2017年或之前已经选修过课程的学生。
在子查询中，通过连接`takes`和`section`表，我们检查每个学生是否在2017年之前选修过课程。

```sql
select student.id, student.name
from student
where not exists (
    select 1 from takes
    join section on takes.course_id = section.course_id and takes.sec_id =
section.sec_id
    where student.id = takes.id and section.year < 2017
);
```

c) 对于每个系（department），找到该系教师的最高薪水。
在教师表`instructor`中，对每个部门`dept_name`进行分组，并计算每个部门教师的最高薪水。
使用`max`函数来找出每个分组中的最大薪水。

```sql
select dept_name, max(salary) as max_salary
from instructor
group by dept_name;
```

d) 找到在所有系（departments）中计算出的每个系最高薪水的最低值。
首先内部查询使用了与上一查询相同的逻辑来为每个系找到最高薪水。
外部查询则从这些最高薪水中找出最低的一个，使用`min`函数。

```sql
select min(max_salary) as min_of_max_salaries
from (
    select dept_name, max(salary) as max_salary
    from instructor
    group by dept_name
) as max_salaries;
```

2. 考虑以下查询

```sql
with dept_total (dept_name, value) as
(seleet dept_name, sum(salary)
from instructor
group by dept_name),
depttotal_avg(value) as
(seleet avg(value) from dept_total)
select dept_name from dept_total, dept_total_avg
where dept_total.value>= dept_totalavg.value,
```

请将此查询重新编写，而不使用WITH子句。

首先计算每个部门教师薪资的总和，并将结果作为一个临时表`dept_total`。
接着计算`dept_total`中所有值的平均值，作为第二个临时表`dept_total_avg`。
最后，查询从`dept_total`中选择那些薪资总和大于或等于平均薪资的部门名称

```sql
select dept_name
from (
    select dept_name, sum(salary) as value
    from instructor
    group by dept_name
) as dept_total,
```

```
(
    select avg(value) as value
    from (
        select dept_name, sum(salary) as value
        from instructor
        group by dept_name
    ) as dept_total
) as dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

3.不使用"unique"构造，重写where语句：

`where unique (select title from course)`
首先在课程表`course`中对课程标题`title`进行分组，然后通过`having count(*) = 1`来筛选出那些唯一的标题。
外部查询计算这些唯一标题的数量，如果数量大于0，则条件为真。

```
where (select count(*) from (select title from course group by title having
count(*) = 1) as unique_titles) > 0;
```

Fig.1银行数据库

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

4. 考虑 Fig.1 中的银行数据库，其中主键已经用下划线标记。构建以下针对这

个关系数据库的 SQL 查询。
a) 查找银行的每位顾客的 ID，他们拥有账户但没有贷款。
从存款人表`depositor`中选择具有账户的所有顾客的ID。通过使用`where not exists`子句，查询检查每位顾客是否在借款人表`borrower`中存在记录。如果不存在，即该顾客没有贷款，该顾客ID就会被选中。使用`distinct`确保结果中ID的唯一性。

```
select distinct d.id
from depositor d
where not exists (
    select 1
    from borrower b
    where d.id = b.id
);
```

b) 查找与顾客 '12345' 同住在相同街道和城市的每位顾客的 ID。
首先从客户表`customer`中找到ID为'12345'的顾客的街道和城市，然后查询所有与这个地址相同，但ID不是'12345'的其他顾客的ID。通过两个嵌套的子查询获取'12345'顾客的街道和城市，确保仅选择住在同一地点的其他顾客。

```sql
select c.id
from customer c
where c.customer_street = (
    select customer_street
    from customer
    where id = '12345'
)
and c.customer_city = (
    select customer_city
    from customer
    where id = '12345'
)
and c.id <> '12345';
```

c) 查找每个至少有一位在银行拥有账户并住在“Harrison”的顾客的分行的名称。
通过连接账户表`account`、存款人表`depositor`和客户表`customer`来找到所有在'Harrison'城市有居住并且拥有账户的顾客。通过指定`c.customer_city = 'Harrison'`来筛选客户，最后选择对应分行的名称，使用`distinct`来确保分行名称的唯一性。

```sql
select distinct a.branch_name
from account a
join depositor d on a.account_number = d.account_number
join customer c on d.id = c.id
where c.customer_city = 'Harrison';
```

5. 考虑 Fig.1 中的银行数据库，其中主键已经用下划线标记。构建以下针对这

个关系数据库的 SQL 查询。
a) 找到每位在“Brooklyn”的每个分行都有账户的顾客。
首先选择所有存款人`depositor`的ID。使用`where not exists`子句来排除那些在“Brooklyn”的某个分行没有账户的顾客。内部的双重`not exists`逻辑确保只有在所有“Brooklyn”分行都有账户的顾客被选中。

```sql
select distinct d.id
from depositor d
where not exists (
    select 1
    from branch b
    where b.branch_city = 'Brooklyn'
    and not exists (
        select 1
        from account a
        join depositor d2 on a.account_number = d2.account_number
        where a.branch_name = b.branch_name
        and d2.id = d.id
    )
);
```

b) 找到银行中所有贷款金额的总和。
计算贷款表`loan`中所有贷款金额的总和。通过使用`sum`函数聚合所有贷款的金额，得到总和。

```
select sum(amount) as total_loan_amount
from loan;
```

c) 找到具有资产超过至少一个位于“Brooklyn”的分行的资产的所有分行名称。
首先在子查询中计算出位于“Brooklyn”的所有分行中资产最少的值。然后，主查询选择所有资产超过这个最小值的分行的名称。这确保选中的每个分行都至少比“Brooklyn”中任一分行的资产多。

```
select b1.branch_name
from branch b1
where b1.assets > (
    select min(b2.assets)
    from branch b2
    where b2.branch_city = 'Brooklyn'
);
```