3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

a. Find the titles of courses in the Comp. Sci. department that have 3 credits.

```
select title
from course
where dept name = ‘Comp. Sci.’ and credits = 3
```

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

b. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

Key Idea:

- **student** connects to **takes** through ID (student ID).
- **instructor** connects to **teaches** through ID (instructor ID).
- **takes** and **teaches** connect through course_id, sec_id, semester, and year.

b. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

Step 1: Get student-course relationships from student and takes.

⍰ (student join takes using(ID))

Step 2: Get instructor-course relationships from instructor and teaches.

⍰ (instructor join teaches using(ID))

Step 3: Match students and instructors through the details of course.

⍰ (... join ... using(course_id, sec_id, semester, year))

Step 4: Filter for instructor “ Einstein” .

⍰ where instructor.name = ‘Einstein’

Step 5: Remove duplicates and return unique student IDs.

⍰ select distinct student.ID

b. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

```
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’
```

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

c. Find the highest salary of any instructor.

```
select max(salary)
from instructor
```

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

d. Find all instructors earning the highest salary (there may be more than one with the same salary).

Key Idea:

- Use an aggregate function (**max**) to find the highest salary in the instructor table.
- Use a **subquery** to compare each instructor's salary to the highest salary.

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

d. Find all instructors earning the highest salary (there may be more than one with the same salary).

Step 1: Subquery to find the maximum salary.

- select max(salary) from instructor

Step 2: Compare each instructor’s salary to the maximum salary.

- where salary = (select max(salary) from instructor)

Step 3: Select relevant ID and name.

- select ID, name from instructor

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

d. Find all instructors earning the highest salary (there may be <u>more than one</u> with the same salary).

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

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

e. **Find the enrollment of each section that was offered in Fall 2017.**

Key Idea:

- Use the **section** table to find all sections offered in Fall 2017.
- Use a **subquery** to **count** the number of students enrolled in each section by matching the **takes** table with the details of **section**.

e. Find the enrollment of each section that was offered in Fall 2017.

Step 1: Filter sections offered in Fall 2017.

- from section where semester = ‘Fall’ and year = 2017

Step 2: Select course_id and sec_id.

- select course_id, sec_id, …

Step 3: Use a subquery to count enrollment.

-
  ```
  (select count(ID)
  from takes
  where takes.year = section.year
  and takes.semester = section.semester
  and takes.course_id = section.course_id
  and takes.sec_id = section.sec_id)
  ```

It counts the number of students in the **takes** table who match the specific **section** with the **where** clause.

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

e. **Find the enrollment of each section that was offered in Fall 2017.**

```
select course_id, sec_id,
(select count(ID)
from takes
where takes.year = section.year
and takes.semester = section.semester
and takes.course_id = section.course_id
and takes.sec_id = section.sec_id)
from section where semester = ‘Fall’ and year = 2017
```

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

f. Find the maximum enrollment, across all sections, in Fall 2017.

Key Idea:

- Use a **nested query** to calculate the enrollment for each section.
- Use an aggregate function (**max**) in the outer query to find the highest enrollment.

f. Find the maximum enrollment, across all sections, in Fall 2017.

Step 1: Filter sections offered in Fall 2017.

- where semester = ‘Fall’ and year = 2017

Step 2: Calculate the enrollment for each section.

-
  ```
  select count(ID) as enrollment
   from section natural join takes
   where semester = ‘Fall’ and year = 2017
   group by course_id, sec_id)
  ```

Combines the **section** and **takes** tables based on common columns (course_id, sec_id, semester, and year) to to find the students enrolled in each section.

Step 3: Find the maximum enrollment.

- select max(enrollment) from (…)

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

f. Find the maximum enrollment, across all sections, in Fall 2017.

```
select max(enrollment)
from (select count(ID) as enrollment
from section natural join takes
where semester = ‘Fall’ and year = 2017
group by course_id, sec_id)
```

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

g. Find the sections that had the maximum enrollment in Fall 2017.

Key Idea:

- Use a **with** clause to define a temporary relation.
- Use a **subquery** to compare each section's enrollment with the maximum enrollment and filter the results.

g. Find the sections that had the maximum enrollment in Fall 2017.

Step 1: Define a temporary relation.

```
⍰ with sec_enrollment as …
```

Step 2: Calculate the enrollment for each section.

```
⍰  select count(ID) as enrollment
   from section natural join takes
   where semester = 'Fall' and year = 2017
   group by course_id, sec_id)
```

Combines the **section** and **takes** tables based on common columns (course_id, sec_id, semester, and year) to to find the students enrolled in each section.

Step 3: Find the maximum enrollment.

```
⍰  select max(enrollment) from (…)
```

Step 4: Select sections with the maximum enrollment.

```
select course_id, sec_id from sec_enrollment
where enrollment = (select max(enrollment) from sec_enrollment)
```

3.1 Write the following queries in SQL, using the university schema (as shown in the lecture ppt or the textbook).

g. Find the sections that had the maximum enrollment in Fall 2017.

```
with sec_enrollment as (select course_id, sec_id, count(ID) as enrollment
from section natural join takes
where semester = 'Fall' and year = 2017
group by course_id, sec_id)
select course_id, sec_id
from sec_enrollment
where enrollment = (select max(enrollment)
from sec_enrollment)
```

3.3 Write the following inserts, deletes, or updates in SQL, using the university schema.

a. **Increase the salary of each instructor in the Comp. Sci. department by 10%.**

```
update instructor
set salary = salary * 1.10
where dept name = ‘Comp. Sci.’
```

3.3 Write the following inserts, deletes, or updates in SQL, using the university schema.

b. Delete all courses that have never been offered (i.e., do not occur in the *section* relation).

Key Idea:

- Use a **subquery** to identify courses that appear in the section table.
- Use the **NOT IN** operator to delete courses whose course_id does not match any course_id in the section table.

3.3 Write the following inserts, deletes, or updates in SQL, using the university schema.

b. Delete all courses that have never been offered (i.e., do not occur in the *section* relation).

Step 1: Find courses that have been offered.
Step 2: Find courses that have never been offered.
Step 3: Delete courses that have never been offered.

```
delete from course
where course_id not in
(select course_id from section)
```

3.3 Write the following inserts, deletes, or updates in SQL, using the university schema.

c. **Insert every student whose *tot_cred* attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.**

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

3.8

branch(*<u>branch_name</u>*, *branch city, assets*)
customer (*<u>ID</u>*, *customer_name, customer_street, customer_city*)
loan (*<u>loan_number</u>*, *branch_name, amount*)
[redacted] *loan number*)
account (*<u>account_number</u>*, *branch_name, balance*)
[redacted] *<u>account_number</u>*)

a. Find the ID of each customer of the bank who has an account but not a loan.

(select ID from depositor)
[redacted]
(select ID from borrower)

set difference

3.8

branch(branch_name, branch city, assets)
[redacted]*customer_name,* [redacted]*)*
loan (loan_number, branch_name, amount)
borrower (ID, loan number)
account (account_number, branch_name, balance)
depositor (ID, account_number)

b. Find the ID of each customer who lives on the same street and in the same city as customer '12345'.

select F.ID from customer F join customer S
[redacted] **where S.ID = '12345'**

ID	Address
10001	A
10002	A
10003	B
10004	C
12345	B

customer F

ID	Address
10001	A
10002	A
10003	B
10004	C
12345	B

customer S

join

F.ID	S.ID	Address
10001	10001	A
10001	10002	A
	......	
10003	12345	B
12345	12345	B

3.8

branch(*branch_name*, *branch city*, *assets*)
[redacted], *customer_name*, *customer_street*, [redacted])
loan (*loan_number*, *branch_name*, *amount*)
borrower (*ID*, *loan number*)
[redacted], *balance*)
[redacted]

c. Find the name of each branch that has at least one customer who has an account in the bank and who lives in “Harrison”.

select distinct branch_name from
account [redacted] **depositor** [redacted] **customer**
where customer_city = ’Harrison’

3.9

[redacted], *street*, [redacted]
[redacted], *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

a. Find the ID, name, and city of residence of each employee who works for “First Bank Corporation”.

select e.ID, e.person_name, e.city from employee e, works w
[redacted] w.company_name = ’First Bank Corporation’ and [redacted]

3.9

[redacted], *street*, [redacted]
[redacted]
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

b. Find the ID, name, and city of residence of each employee who works for First Bank Corporation” and earns more than $10000.

boolean of membership in a set

select ID, person_name, city from employee where ID [redacted] (select ID from works where company name = ’First Bank Corporation’ and salary > 10000)

3.9

[redacted], *person_name*, *street*, *city*)
[redacted], *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

c. Find the ID of each employee who does not work for “First Bank Corporation”.

select ID from works where company name <> ’First Bank Corporation’

OR

boolean of non-membership in a set

select ID from employee where ID [redacted] (select ID from works where company name = ’First Bank Corporation’)

3.9

employee (*ID*, *person_name*, *street*, *city*)
[redacted])
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

d. Find the ID of each employee who earns more than every employee of “Small Bank Corporation”.

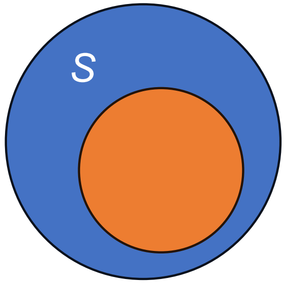
> maximum of a set

select ID from works where salary [redacted] (select salary from works where company name = ’Small Bank Corporation’)

3.9

employee (*ID*, *person_name*, *street*, *city*)
works (*ID*, *company_name*, *salary*)
[redacted]
manages (*ID*, *manager_id*)

e. Assume that companies may be located in several cities. Find the name of each company that is located in every city in which "Small Bank Corporation" is located.



Not Exists City(SBC) except City(S)

select S.company_name from company S
where [redacted] **((select city from company where company_name = 'Small Bank Corporation')**

boolean of being an empty set

[redacted]
(select city from company T where [redacted]**))**

3.9

employee (*ID*, *person_name*, *street*, *city*)
[redacted], *salary*)
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

f. Find the name of the company that has the most employees (or companies, in the case where there is a tie for the most).

select company_name from works
[redacted] company_name
[redacted] (distinct ID) [redacted] (select count(distinct ID) from works group by company_name)

>= maximum of a set

ID	company_name
10001	A
10002	A
10003	A
10004	B
10005	B

grouping →

count(distinct ID) = number of employees	company_name
3	A
2	B

3.9

employee (*ID*, *person_name*, *street*, *city*)
[redacted])
company (*company_name*, *city*)
manages (*ID*, *manager_id*)

g. Find the name of each company whose employees earn a higher salary, on average, than the average salary at “First Bank Corporation”.

select company_name from works
[redacted] company_name
[redacted] (salary) > (select avg (salary) from works where company_name = ’First Bank Corporation’)

company_name	salary
A	5000
A	6000
A	7000
B	8000
B	9000

$\xrightarrow{\text{grouping}}$

company_name	avg(salary)
A	6000
B	8500

Assignment 3

(Chap 4)

4.2 Write the following queries in SQL:

a. Display a list of all instructors, showing each instructor’s ID and the number of sections taught. Make sure to show the number of sections as 0 for instructors who have not taught any section. Your query should use an outer join, and should not use subqueries.

Step:

1. Display what? $\longrightarrow$ select ID, count(course_id)

2. From which table?

$$\begin{tikzcd}
\text{instructor's ID} \arrow[d] & \text{the number of sections taught} \arrow[d] \\
\text{instructor} & \text{teaches}
\end{tikzcd}$$

3. The total number? group by ID

Answer:

```
select ID, count(course_id)
from instructor natural left outer join teaches
group by ID;
```

b. Write the same query as in part a, but using a scalar subquery and not using outer join.

Step:

1. Display what? $\longrightarrow$ select ID, count(course_id)

2. From which table?

instructor’s ID $\downarrow$ instructor

the number of sections taught $\downarrow$ teaches

3. How to connect two tables? The same attribute ID

Answer:

```
select id, (select count(course_id)
        from teaches
        where teaches.id=instructor.id) as
num_of_course
from instructor;
```

c. Display the list of all course sections offered in Spring 2018, along with the ID and name of each instructor teaching the sec~~tion. If a section has more~~ than one instructor, that section sh~~oul~~d app~~ear as many times in~~ the result as it has instructors. If a section does not have any instructor, it should still appear in the result with the instructor name set to “ —” .

Step:

1. Display what? select course_id, sec_id, ID, coalesce(name, '—')

2. From which table? section teaches instructor

3. Condition? in Spring 2018

Answer:

```
select course_id, sec_id, ID, coalesce(name, '—')
as name
from (section natural left outer join teaches)
natural left outer join instructor
where semester = 'Spring' and year = 2018;
```

d. Display the list of all departments, with the total number of instructors in each department, without using subqueries. Make sure to show departments that have no instructors, and list those departments with an instructor count of zero.

Step:

1. Display what? select dept_name, count(ID)

2. From which table? department instructor

3. The total number? group by dept_name

Answer:

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

4.3 Outer join expressions can be computed in SQL without using the SQL outer join operation. To illustrate this fact, show how to rewrite each of the following SQL queries without using the outer join expression.

a. select * from student natural left outer join takes

Answer:

```
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);
```

If we use natural join, what is the difference?

1.natural join: the common value on the same attribute

left outer join: the common value + value only in the left table

```
select * from student natural join takes
```

```
select from student S1 where not exists
(select ID from takes T1 where T1.id = S1.id);
```

2. natural join: all attribute has values

left outer join: value only in the left table will show NULL on the attribution only in the right table

```
select ID, name, dept_name, tot_cred, NULL, NULL, NULL, NULL, NULL
```

b. select * from student natural full outer join takes

Full outer join = Left outer join union Right outer join

Answer:

```
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, sec_id, semester, year, grade
from takes T1
where not exists
(select ID from student S1 where T1.id = S1.id));
```