3.10 Consider the employee database of Figure 3.20, where the primary keys are underlined. Give an expression in SQL for each of the following queries.

a. Find the names and cities of residence of all employees who work for First Bank Corporation.

b. Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

c. Find all employees in the database who do not work for First Bank Corporation.

d. Find all employees in the database who earn more than each employee of Small Bank Corporation.

e. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

f. Find the company that has the most employees.

g. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

Answer:

\[
\begin{align*}
\text{employee (employee\_name, street, city)} \\
\text{works (employee\_name, company\_name, salary)} \\
\text{company (company\_name, city)} \\
\text{manages (employee\_name, manager\_name)}
\end{align*}
\]

Figure 3.20. Employee database.
a. Find the names and cities of residence of all employees who work for First Bank Corporation.

```sql
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. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.
If people may work for several companies, the following solution will only list those who earn more than $10,000 per annum from “First Bank Corporation” alone.

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

As in the solution to the previous query, we can use a join to solve this one also.

c. Find all employees in the database who do not work for First Bank Corporation.
The following solution assumes that all people work for exactly one company.

```sql
select employee_name
from works
where company_name ≠ 'First Bank Corporation'
```

If one allows people to appear in the database (e.g. in employee) but not appear in works, or if people may have jobs with more than one company, the solution is slightly more complicated.

```sql
select employee_name
from employee
where employee_name not in
  (select employee_name
   from works
   where company_name = 'First Bank Corporation')
```

d. Find all employees in the database who earn more than each employee of Small Bank Corporation.
The following solution assumes that all people work for at most one company.

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

If people may work for several companies and we wish to consider the total earnings of each person, the problem is more complex. It can be solved by using a nested subquery, but we illustrate below how to solve it using the `with` clause.

```sql
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. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

The simplest solution uses the `contains` comparison which was included in the original System R Sequel language but is not present in the subsequent SQL versions.

```sql
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')
```

Below is a solution using standard SQL.
f. Find the company that has the most employees.

```sql
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. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

```sql
select company_name
from works
group by company_name
having avg (salary) > (select avg (salary)
                        from works
                        where company_name = 'First Bank Corporation')
```
4.1 Write the following queries in SQL:

a. Display a list of all instructors, showing their ID, name, and the number of sections that they have taught. Make sure to show the number of sections as 0 for instructors who have not taught any section. Your query should use an outerjoin, and should not use scalar subqueries.

b. Write the same query as above, but using a scalar subquery, without outerjoin.

c. Display the list of all course sections offered in Spring 2010, along with the names of the instructors teaching the section. If a section has more than one instructor, it should appear as many times in the result as it has instructors. If it does not have any instructor, it should still appear in the result with the instructor name set to "—".

d. Display the list of all departments, with the total number of instructors in each department, without using scalar subqueries. Make sure to correctly handle departments with no instructors.

Answer:

a. Display a list of all instructors, showing their ID, name, and the number of sections that they have taught. Make sure to show the number of sections as 0 for instructors who have not taught any section. Your query should use an outerjoin, and should not use scalar subqueries.
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

The above query should not be written using count(*) since count * counts null values also. It could be written using count(section_id), or any other attribute from teaches which does not occur in instructor, which would be correct although it may be confusing to the reader. (Attributes that occur in instructor would not be null even if the instructor has not taught any section.)

b. Write the same query as above, but using a scalar subquery, without outerjoin.

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

c. Display the list of all course sections offered in Spring 2010, along with the names of the instructors teaching the section. If a section has more than one instructor, it should appear as many times in the result as it has instructors. If it does not have any instructor, it should still appear in the result with the instructor name set to “—”.

    select course_id, section_id, ID,
        decode(name, NULL, '—', name)
    from (section natural left outer join teaches)
    natural left outer join instructor
    where semester='Spring' and year= 2010

The query may also be written using the coalesce operator, by replacing decode(...) by coalesce(name, '—'). A more complex version of the query can be written using union of join result with another query that uses a subquery to find courses that do not match; refer to exercise 4.2.

d. Display the list of all departments, with the total number of instructors in each department, without using scalar subqueries. Make sure to correctly handle departments with no instructors.

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