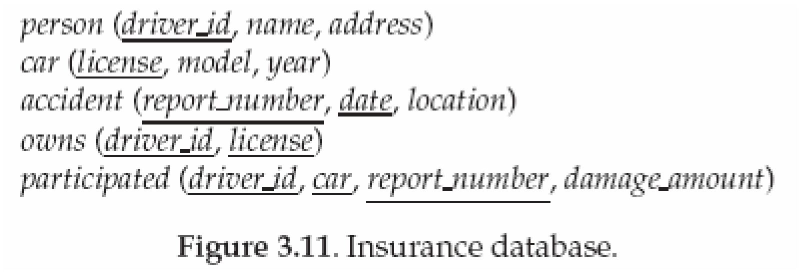
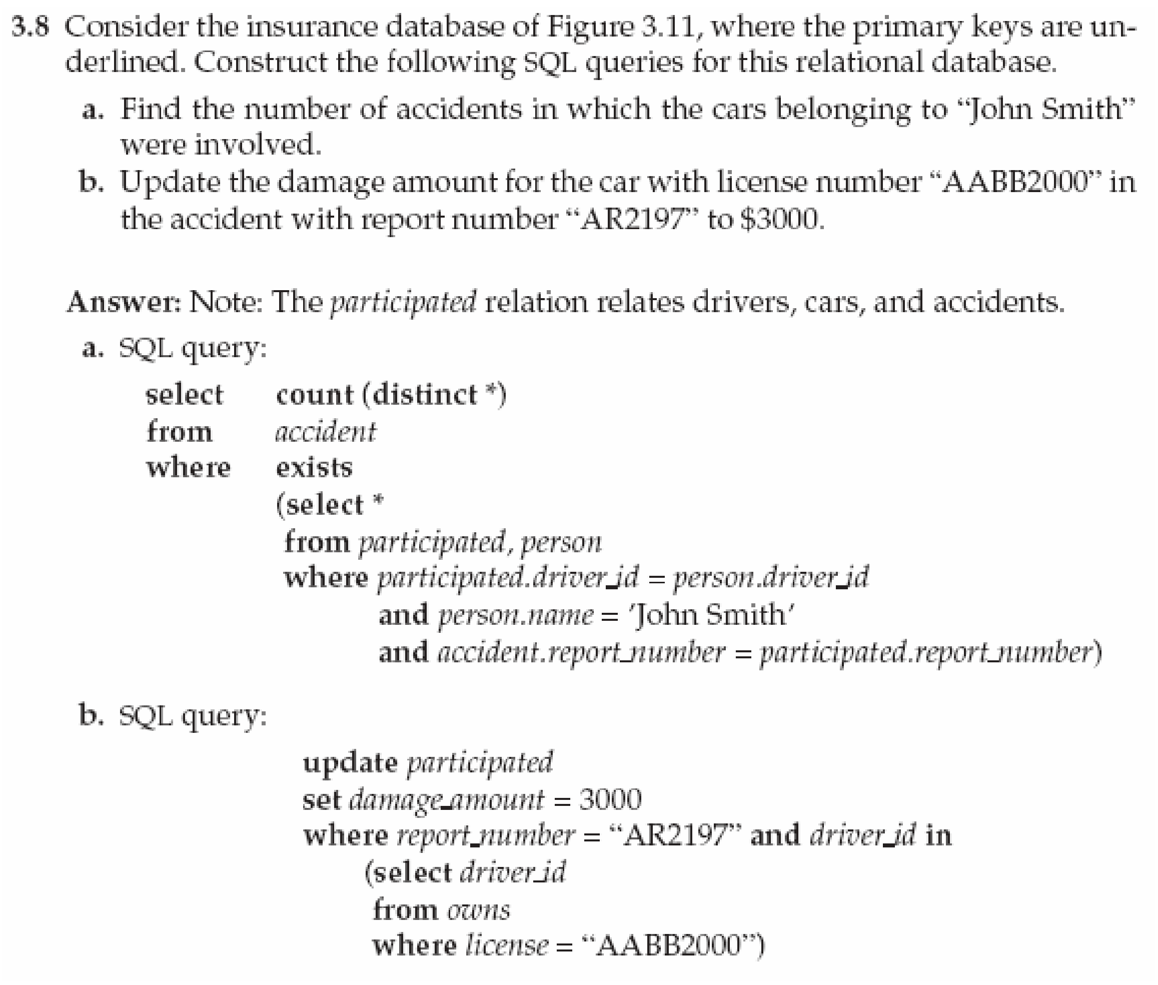


person (<u>*driver_id*</u>, *name*, *address*)
car (<u>*license*</u>, *model*, *year*)
accident (<u>*report_number*</u>, <u>*date*</u>, *location*)
owns (<u>*driver_id*</u>, <u>*license*</u>)
participated (<u>*driver_id*</u>, <u>*car*</u>, <u>*report_number*</u>, *damage_amount*)

Figure 3.11. Insurance database.



3.8 Consider the insurance database of Figure 3.11, where the primary keys are underlined. Construct the following SQL queries for this relational database.

a. Find the number of accidents in which the cars belonging to "John Smith" were involved.

b. Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

Answer: Note: The *participated* relation relates drivers, cars, and accidents.

a. SQL query:

```
select   count (distinct *)
from     accident
where    exists
         (select *
          from participated, person
          where participated.driver_id = person.driver_id
                and person.name = 'John Smith'
                and accident.report_number = participated.report_number)
```

b. SQL query:

```
update participated
set damage_amount = 3000
where report_number = "AR2197" and driver_id in
      (select driver_id
       from owns
       where license = "AABB2000")
```

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

Figure 3.12. Employee database.

3.9 Consider the employee database of Figure 3.12, where the primary keys are underlined. Give an expression in SQL for each of the following queries.

a. Find the names of all employees who work for First Bank Corporation.
b. Find all employees in the database who live in the same cities as the companies for which they work.
c. Find all employees in the database who live in the same cities and on the same streets as do their managers.
d. Find all employees who earn more than the average salary of all employees of their company.
e. Find the company that has the smallest payroll.

Answer:

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

```
select employee_name
from works
where company_name = 'First Bank Corporation'
```

b. Find all employees in the database who live in the same cities as the companies for which they work.

```
select e.employee_name
from employee e, works w, company c
where e.employee_name = w.employee_name and e.city = c.city and
    w.company_name = c.company _name
```

c. Find all employees in the database who live in the same cities and on the same streets as do their managers.

```
select P.employee_name
from employee P, employee R, manages M
where P.employee_name = M.employee_name and
    M.manager_name = R.employee_name and
    P.street = R.street and P.city = R.city
```

d. Find all employees who earn more than the average salary of all employees of their company.

The following solution assumes that all people work for at most one company.

```
select employee_name
from works T
where salary > (select avg (salary)
                from works S
                where T.company_name = S.company_name)
```

e. Find the company that has the smallest payroll.

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

3.10 Consider the relational database of Figure 3.12. Give an expression in SQL for each of the following queries.

a. Give all employees of First Bank Corporation a 10 percent raise.
b. Give all managers of First Bank Corporation a 10 percent raise.
c. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

Answer:

a. Give all employees of First Bank Corporation a 10-percent raise. (the solution assumes that each person works for at most one company.)

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

b. Give all managers of First Bank Corporation a 10-percent raise.

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

c. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

```
delete works
where company_name = 'Small Bank Corporation'
```

3.11 Let the following relation schemas be given:

$$R = (A, B, C)$$
$$S = (D, E, F)$$

Let relations $r(R)$ and $s(S)$ be given. Give an expression in SQL that is equivalent to each of the following queries.

a. $\Pi_A(r)$
b. $\sigma_{B=17}(r)$
c. $r \times s$
d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

Answer:

a. $\Pi_A(r)$

```
select distinct A
from r
```

b. $\sigma_{B=17}(r)$

```
select *
from r
where B = 17
```

c. $r \times s$

```
select distinct *
from r, s
```

d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

```
select distinct A, F
from r, s
where C = D
```

3.12 Let $R = (A, B, C)$, and let r_1 and r_2 both be relations on schema R. Give an expression in SQL that is equivalent to each of the following queries.

a. $r_1 \cup r_2$
b. $r_1 \cap r_2$
c. $r_1 - r_2$
d. $\Pi_{AB}(r_1) \bowtie \Pi_{BC}(r_2)$

Answer:

a. $r_1 \cup r_2$

```
(select *
 from r1)
union
(select *
 from r2)
```

b. $r_1 \cap r_2$

We can write this using the **intersect** operation, which is the preferred approach, but for variety we present an solution using a nested subquery.

```
select *
from r1
where (A, B, C) in (select *
                    from r2)
```

c. $r_1 - r_2$

```
select *
from r1
where (A, B, C) not in (select *
                        from r2)
```

This can also be solved using the **except** clause.

d. $\Pi_{AB}(r_1) \bowtie \Pi_{BC}(r_2)$

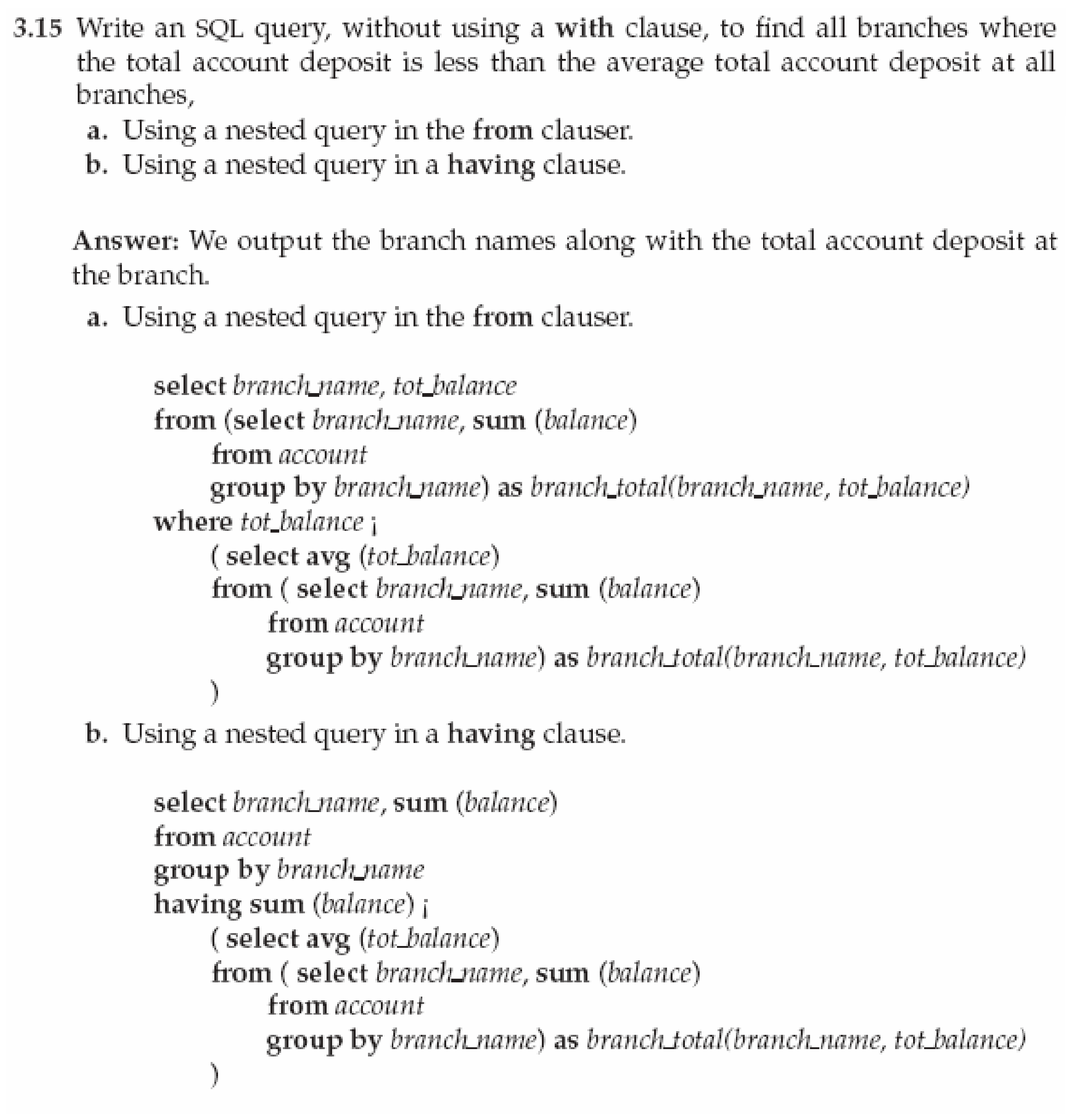
```
select r1.A, r2.B, r3.C
from r1, r2
where r1.B = r2.B
```

3.14 Consider the relational database of Figure 3.12. Using SQL, define a view consisting of *manager_name* and the average salary of all employees who work for that manager. Explain why the database system should not allow updates to be expressed in terms of this view.

Answer:

```
create view salinfo as
    select manager_name, avg(salary)
    from manages m, works w
    where m.employee_name = w.employee_name
    group by manager_name
```

Updates should not be allowed in this view because there is no way to determine how to change the underlying data. For example, suppose the request is "change the average salary of employees working for Smith to $200". Should everybody who works for Smith have their salary changed to $200? Or should the first (or more, if necessary) employee found who works for Smith have their salary adjusted so that the average is $200? Neither approach really makes sense.



3.15 Write an SQL query, without using a **with** clause, to find all branches where the total account deposit is less than the average total account deposit at all branches,

a. Using a nested query in the **from** clauser.
b. Using a nested query in a **having** clause.

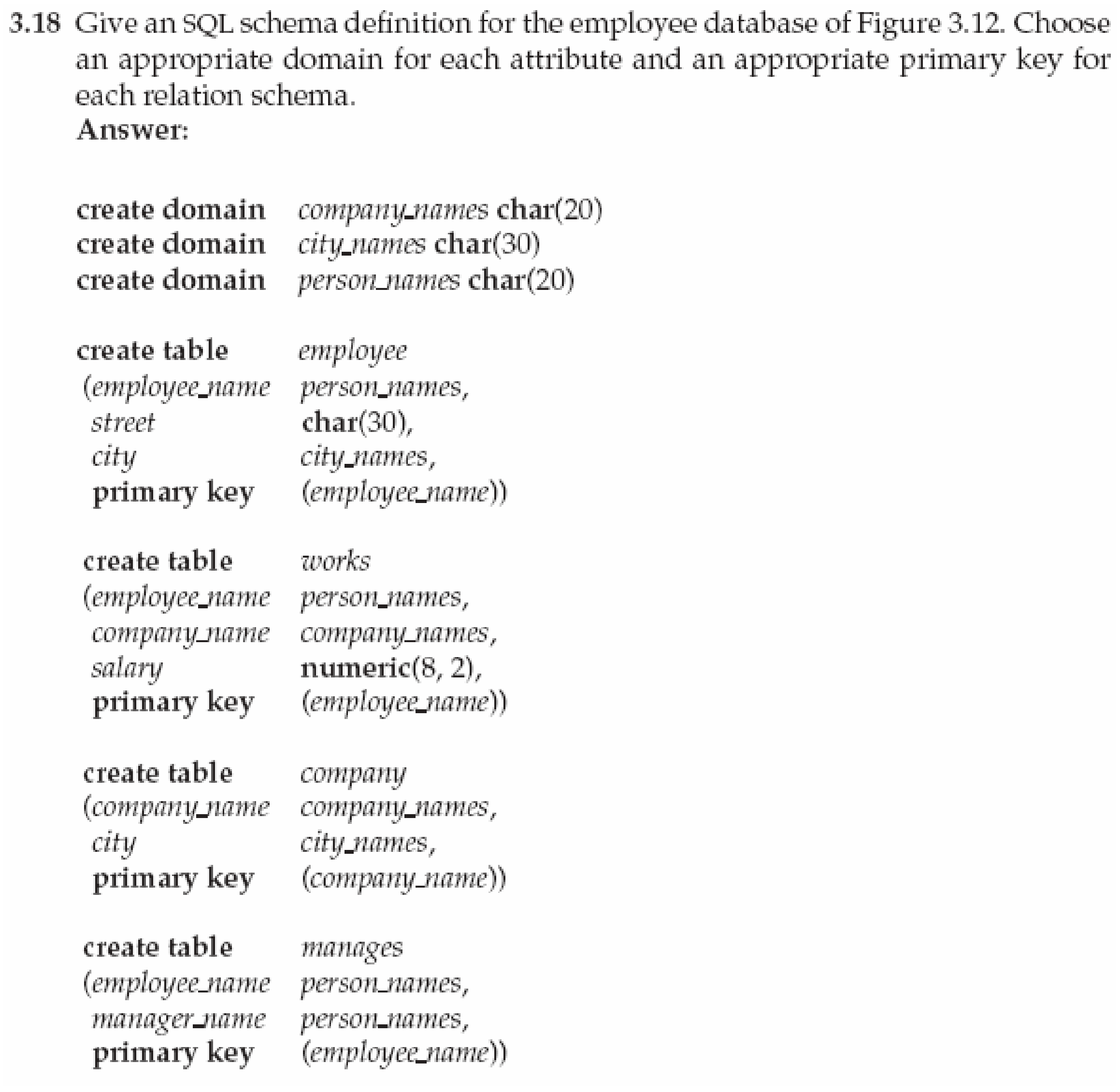
Answer: We output the branch names along with the total account deposit at the branch.

a. Using a nested query in the **from** clauser.

```
select branch_name, tot_balance
from (select branch_name, sum (balance)
        from account
        group by branch_name) as branch_total(branch_name, tot_balance)
where tot_balance ¡
        ( select avg (tot_balance)
        from ( select branch_name, sum (balance)
                from account
                group by branch_name) as branch_total(branch_name, tot_balance)
        )
```

b. Using a nested query in a **having** clause.

```
select branch_name, sum (balance)
from account
group by branch_name
having sum (balance) ¡
        ( select avg (tot_balance)
        from ( select branch_name, sum (balance)
                from account
                group by branch_name) as branch_total(branch_name, tot_balance)
        )
```



3.18 Give an SQL schema definition for the employee database of Figure 3.12. Choose an appropriate domain for each attribute and an appropriate primary key for each relation schema.

Answer:

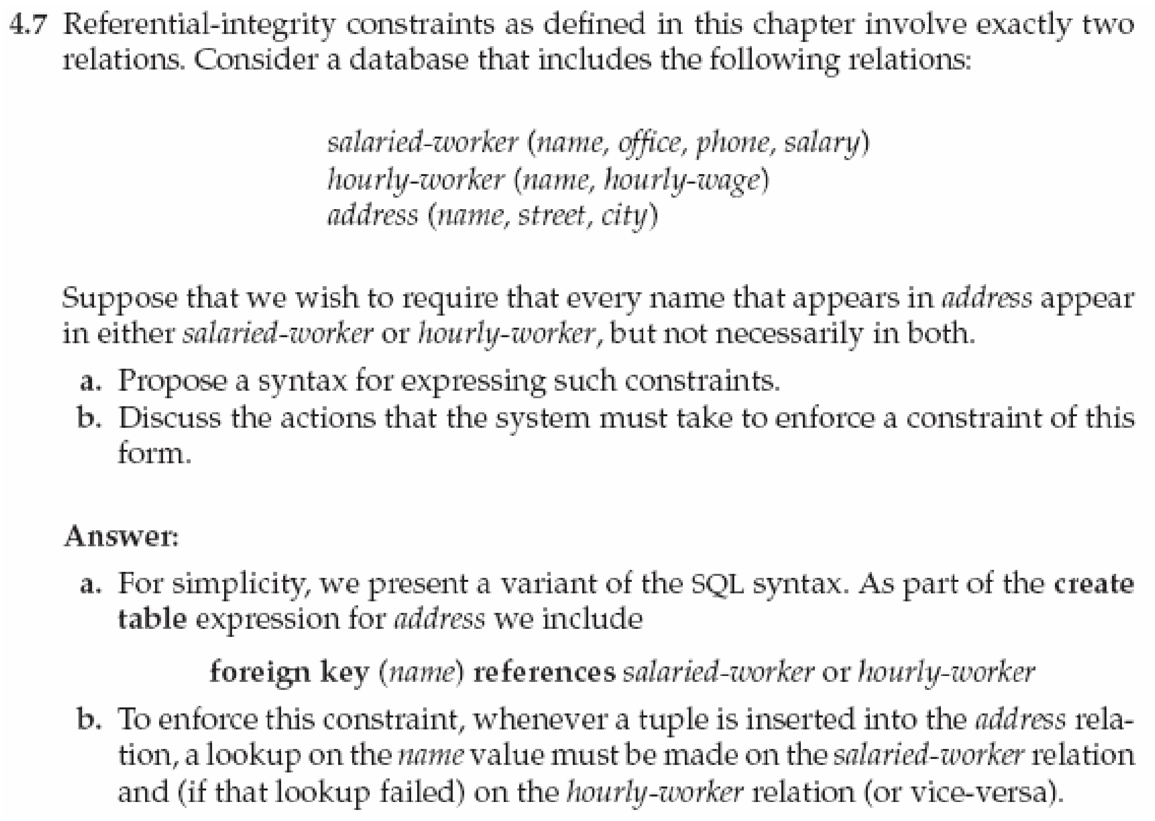
```
create domain   company_names char(20)
create domain   city_names char(30)
create domain   person_names char(20)

create table    employee
 (employee_name person_names,
  street        char(30),
  city          city_names,
  primary key   (employee_name))

create table    works
 (employee_name person_names,
  company_name  company_names,
  salary        numeric(8, 2),
  primary key   (employee_name))

create table    company
 (company_name  company_names,
  city          city_names,
  primary key   (company_name))

create table    manages
 (employee_name person_names,
  manager_name  person_names,
  primary key   (employee_name))
```



4.7 Referential-integrity constraints as defined in this chapter involve exactly two relations. Consider a database that includes the following relations:

salaried-worker (*name, office, phone, salary*)
hourly-worker (*name, hourly-wage*)
address (*name, street, city*)

Suppose that we wish to require that every name that appears in *address* appear in either *salaried-worker* or *hourly-worker*, but not necessarily in both.

a. Propose a syntax for expressing such constraints.
b. Discuss the actions that the system must take to enforce a constraint of this form.

Answer:

a. For simplicity, we present a variant of the SQL syntax. As part of the **create table** expression for *address* we include

foreign key (*name*) **references** *salaried-worker* or *hourly-worker*

b. To enforce this constraint, whenever a tuple is inserted into the *address* relation, a lookup on the *name* value must be made on the *salaried-worker* relation and (if that lookup failed) on the *hourly-worker* relation (or vice-versa).

4.10 Consider an employee database with two relations

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

where the primary keys are underlined. Write a query to find companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

a. Using SQL functions as appropriate.
b. Without using SQL functions.

Answer:

```
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")
```

4.12 Compare the use of embedded SQL with the use in SQL of functions defined in a general-purpose programming language. Under what circumstances would you use each of these features?
Answer: SQL functions are primarily a mechanism for extending the power of SQL to handle attributes of complex data types (like images), or to perform complex and non-standard operations. Embedded SQL is useful when imperative actions like displaying results and interacting with the user are needed. These cannot be done conveniently in an SQL only environment. Embedded SQL can be used instead of SQL functions by retrieving data and then performing the function's operations on the SQL result. However a drawback is that a lot of query-evaluation functionality may end up getting repeated in the host language code.