CHAPTER 3

SQL

Solutions to Practice Exercises

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

a. Note: this is not the same as the total number of accidents in 1989. We must count people with several accidents only once.

```
select   count (distinct name)
from     accident, participated, person
where    accident.report_number = participated.report_number
and      participated.driver_id = person.driver_id
and      date between date '1989-00-00' and date '1989-12-31'
```

b. We assume the driver was "Jones," although it could be someone else. Also, we assume "Jones" owns one Toyota. First we must find the license of the given car. Then the *participated* and *accident* relations must be updated in order to both record the accident and tie it to the given car. We assume values "Berkeley" for *location*, '2001-09-01' for date and *date*, 4007 for *report_number* and 3000 for damage amount.

```
insert into accident
        values (4007, '2001-09-01', 'Berkeley')

insert into participated
        select o.driver_id, c.license, 4007, 3000
        from person p, owns o, car c
        where p.name = 'Jones' and p.driver_id = o.driver_id and
                o.license = c.license and c.model = 'Toyota'
```

c. Since *model* is not a key of the *car* relation, we can either assume that only one of John Smith's cars is a Mazda, or delete all of John Smith's Mazdas (the query is the same). Again assume *name* is a key for *person*.

```
delete car
where model = 'Mazda' and license in
    (select license
     from person p, owns o
     where p.name = 'John Smith' and p.driver_id = o.driver_id)
```

Note: The *owns*, *accident* and *participated* records associated with the Mazda still exist.

3.2 **a.** Query:

```
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. 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.

```
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. The following solution assumes that all people work for exactly one company.

```
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.

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

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

```
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.

```
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. 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.

```
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.

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

f. Query:

```
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. Query:

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

3.3 **a.** The solution assumes that each person has only one tuple in the *employee* relation.

```
update employee
set city = 'Newton'
where person_name = 'Jones'
```

b. Query:

```
update works T
set T.salary = T.salary * 1.03
where T.employee_name in (select manager_name
                          from manages)
      and T.salary * 1.1 > 100000
      and T.company_name = 'First Bank Corporation'

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

SQL-92 provides a **case** operation (see Exercise 3.5), using which we give a more concise solution:

```
update works T
set T.salary = T.salary *
    (case
        when (T.salary * 1.1 > 100000) then 1.03
        else 1.1
    )
where T.employee_name in (select manager_name
                          from manages) and
      T.company_name = 'First Bank Corporation'
```

3.4 Query:

```
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
```

3.5 We use the **case** operation provided by SQL-92:

a. To display the grade for each student:

```
select student_id,
       (case
                when score < 40 then 'F',
                when score < 60 then 'C',
                when score < 80 then 'B',
                else 'A'
        end) as grade
from marks
```

b. To find the number of students with each grade we use the following query, where *grades* is the result of the query given as the solution to part 0.a.

select *grade*, **count**(*student_id*)
from *grades*
group by *grade*

3.6 The query selects those values of *p.a1* that are equal to some value of *r1.a1* or *r2.a1* if and only if both *r1* and *r2* are non-empty. If one or both of *r1* and *r2* are empty, the cartesian product of *p*, *r1* and *r2* is empty, hence the result of the query is empty. Of course if *p* itself is empty, the result is as expected, i.e. empty.

3.7 To insert the tuple ("Johnson", 1900) into the view *loan_info*, we can do the following:

$borrower \leftarrow (\text{"Johnson"}, \perp_k) \cup borrower$

$loan \leftarrow (\perp_k, \perp, 1900) \cup loan$

such that $\perp_k$ is a new marked null not already existing in the database.