C H A P T E R 4

SQL

Exercises

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

a. Find the total number of people who owned cars that were involved in accidents in 1989.

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'
```

c. Add a new accident to the database; assume any values for required attributes.

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'
```

d. Delete the Mazda belonging to "John Smith".

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.

4.2 Answer:

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

```
select e.employee-name, city
from employee e, works w
where w.company-name = 'First Bank Corporation' and
        w.employee-name = e.employee-name
```

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

```
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.

If people may work for several companies (not possible with the given schema, because *employee-name* is a primary key of *works*) the above solution will only list those who earn more than $10,000 per annum from "First Bank Corporation" alone.

f. 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.

```
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')
```

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

```
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
   )
```

h. 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.

```
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))
```

j. Find the company that has the most employees.

```
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)
```

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

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

4.3 Answer:

d. Give all managers of First Bank Corporation a 10-percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3-percent raise.

The SQL-92 **case** statement allows a 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'
```

If the case statement is not available, the update can be performed as follows.

```
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'
```

4.6 Answer:

a. $\{< a > \mid \exists\, b\, (< a, b > \in\ r\ \wedge\ b\ =\ 17)\}$

```
select distinct A
from r
where B = 17
```

b. $\{< a, b, c > \ \mid\ < a, b > \in\ r \wedge\ < a, c > \in\ s)\}$

```
select distinct r.A, r.B, s.C
from r, s
where r.A = s.A
```

c. $\{< a > \mid \exists\, c\, (< a, c > \in\ s\ \wedge \exists\, b_1, b_2\, (< a, b_1 > \in\ r \wedge\ < c, b_2 > \in\ r\ \wedge\ b_1\ >\ b_2))\}$

```
select distinct s.A
from s, r e, r m
where s.A = e.A and s.C = m.A and e.B > m.B
```

4.9 Answer: 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.

4.11 Answer: 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
```

4.13 Answer:

```
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
```

4.15 Answer:

a. check condition for the *works* table:-

```
check((employee-name, company-name) in
    (select  e.employee-name, c.company-name
     from    employee e, company c
     where   e.city = c.city
    )
)
```

b. check condition for the *works* table:-

```
check(
    salary < all
        (select manager-salary
         from   (select manager-name, manages.employee-name as emp-name,
                        salary as manager-salary
                 from works, manages
                 where works.employee-name = manages.manager-name)
         where  employee-name = emp-name
        )
)
```

The solution is slightly complicated because of the fact that inside the **select** expression's scope, the outer *works* relation into which the insertion is being performed is inaccessible. Hence the renaming of the *employee-name* attribute to *emp-name*. Under these circumstances, it is more natural to use assertions, which are introduced in Chapter 6.

4.16 Answer: Writing queries in SQL is typically much easier than coding the same queries in a general-purpose programming language. However not all kinds of queries can be written in SQL. Also nondeclarative actions such as printing a report, interacting with a user, or sending the results of a query to a graphical user interface cannot be done from within SQL. Under circumstances in which we want the best of both worlds, we can choose embedded SQL or dynamic SQL, rather than using SQL alone or using only a general-purpose programming language.

Embedded SQL has the advantage of programs being less complicated since it avoids the clutter of the ODBC or JDBC function calls, but requires a specialized preprocessor.