Exercises

4.1 Consider the insurance database of Figure 4.12, where the primary keys are underlined. Construct the following SQL queries for this relational database.

- **a.** Find the total number of people who owned cars that were involved in accidents in 1989.
- **b.** Find the number of accidents in which the cars belonging to "John Smith" were involved.
- **c.** Add a new accident to the database; assume any values for required attributes.
- **d.** Delete the Mazda belonging to "John Smith".
- **e.** 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.** 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'
```

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

```
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)
```

- **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.

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 4.12. Insurance database.

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.

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

update *participated*
set *damage-amount* = 3000
where *report-number* = "AR2197" **and** *driver-id* **in**
  (**select** *driver-id*
  from *owns*
  where *license* = "AABB2000")

4.2 Consider the employee database of Figure 4.13, 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 the names and cities of residence of all employees who work for First Bank Corporation.
c. Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

d. Find all employees in the database who live in the same cities as the companies for which they work.
e. Find all employees in the database who live in the same cities and on the same streets as do their managers.
f. Find all employees in the database who do not work for First Bank Corporation.
g. Find all employees in the database who earn more than each employee of Small Bank Corporation.
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.
i. Find all employees who earn more than the average salary of all employees of their company.
j. Find the company that has the most employees.
k. Find the company that has the smallest payroll.
l. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

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 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.

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.

d. 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
```

e. 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
```

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.
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
   )
```

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))
```

i. 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.

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

Figure 4.13. Employee database.

```
select employee-name
from works T
where salary > (select avg (salary)
                from works S
                where T.company-name = S.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)
```

k. 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)
```

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 Consider the relational database of Figure 4.13. Give an expression in SQL for each of the following queries.

a. Modify the database so that Jones now lives in Newtown.
b. Give all employees of First Bank Corporation a 10 percent raise.
c. Give all managers of First Bank Corporation a 10 percent raise.
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.

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

Answer: The solution for part 0.a assumes that each person has only one tuple in the *employee* relation. The solutions to parts 0.c and 0.d assume that each person works for at most one company.

a. Modify the database so that Jones now lives in Newtown.

```
update employee
set city = 'Newton'
where person-name = 'Jones'
```

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

```
update works
set salary = salary * 1.1
where company-name = 'First Bank Corporation'
```

c. 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'
```

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.

```
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 4.11), 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'
```

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

```
delete works
where company-name = 'Small Bank Corporation'
```

4.4 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
```

4.5 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
```

4.6 Let $R = (A, B)$ and $S = (A, C)$, and let $r(R)$ and $s(S)$ be relations. Write an expression in SQL for each of the queries below:

a. $\{< a > \mid \exists\, b\, (< a, b > \in\ r \land\ b\ =\ 17)\}$
b. $\{< a, b, c > \mid\ < a, b > \in\ r \land\ < a, c > \in\ s\}$
c. $\{< a > \mid \exists\, c\, (< a, c > \in\ s \land \exists\, b_1, b_2\, (< a, b_1 > \in\ r \land\ < c, b_2 > \in\ r \land b_1 > b_2))\}$

Answer:

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

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

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

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** $e.B \ > \ m.B$

4.7 Show that, in SQL, $<>$ **all** is identical to **not in**.
Answer: Let the set S denote the result of an SQL subquery. We compare ($x <>$ **all** S) with (x **not in** S). If a particular value x_1 satisfies ($x_1 <>$ **all** S) then for all elements y of S $x_1 \neq y$. Thus x_1 is not a member of S and must satisfy (x_1 **not in** S). Similarly, suppose there is a particular value x_2 which satisfies (x_2 **not in** S). It cannot be equal to any element w belonging to S, and hence ($x_2 <>$ **all** S) will be satisfied. Therefore the two expressions are equivalent.

4.8 Consider the relational database of Figure 4.13. 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.

4.9 Consider the SQL query

select $p.a1$
from p, $r1$, $r2$
where $p.a1 = r1.a1$ **or** $p.a1 = r2.a1$

Under what conditions does the preceding query select values of $p.a1$ that are either in $r1$ or in $r2$? Examine carefully the cases where one of $r1$ or $r2$ may be empty.
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.

4.10 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)
      )
```

4.11 Suppose that we have a relation *marks*(*student-id*, *score*) and we wish to assign grades to students based on the score as follows: grade F if $score < 40$, grade C if $40 \leq score < 60$, grade B if $60 \leq score < 80$, and grade A if $80 \leq score$. Write SQL queries to do the following:

a. Display the grade for each student, based on the *marks* relation.
b. Find the number of students with each grade.

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
```

4.12 SQL-92 provides an n-ary operation called **coalesce**, which is defined as follows: **coalesce**($A_1, A_2, \ldots, A_n$) returns the first nonnull A_i in the list $A_1, A_2, \ldots, A_n$, and returns null if all of $A_1, A_2, \ldots, A_n$ are null. Show how to express the **coalesce** operation using the **case** operation.
Answer:

```
case
    when A1 is not null then A1
    when A2 is not null then A2
    . . .
    when An is not null then An
    else null
end
```

4.13 Let a and b be relations with the schemas A(*name, address, title*) and B(*name, address, salary*), respectively. Show how to express a **natural full outer join** b using the **full outer join** operation with an **on** condition and the **coalesce** operation. Make sure that the result relation does not contain two copies of the attributes *name* and *address*, and that the solution is correct even if some tuples in a and b have null values for attributes *name* or *address*.
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
```

4.14 Give an SQL schema definition for the employee database of Figure 4.13. 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))
```

4.15 Write **check** conditions for the schema you defined in Exercise 4.14 to ensure that:

a. Every employee works for a company located in the same city as the city in which the employee lives.
b. No employee earns a salary higher than that of his manager.

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 Describe the circumstances in which you would choose to use embedded SQL rather than SQL alone or only a general-purpose programming language.
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.