Introduction to SQL

Solutions for the Practice Exercises of Chapter 3

Practice Exercises

3.1

Answer:

a. Find the titles of courses in the Comp. Sci. department that have 3 credits.

```sql
select title
from course
where dept_name = 'Comp. Sci.' and credits = 3
```

b. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result. This query can be answered in several different ways. One way is as follows.

```sql
select distinct takes.ID
from takes, instructor, teaches
where takes.course_id = teaches.course_id and
takes.sec_id = teaches.sec_id and
takes.semester = teaches.semester and
takes.year = teaches.year and
teaches.id = instructor.id and
instructor.name = 'Einstein'
```

c. Find the highest salary of any instructor.

```sql
select max(salary)
from instructor
```
d. Find all instructors earning the highest salary (there may be more than one with the same salary).

\[
\text{select } \quad \text{ID, name} \\
\text{from } \quad \text{instructor} \\
\text{where } \quad \text{salary} = (\text{select max(salary)} \text{ from instructor})
\]

e. Find the enrollment of each section that was offered in Fall 2017.

\[
\text{select } \quad \text{course_id, sec_id,} \\
\quad (\text{select count(ID)} \text{ from takes}) \\
\quad \text{from } \quad \text{t} \\
\quad \text{where } \quad \text{takes.year} = \text{section.year} \\
\quad \quad \text{and} \quad \text{takes.semester} = \text{section.semester} \\
\quad \quad \text{and} \quad \text{takes.course_id} = \text{section.course_id} \\
\quad \quad \text{and} \quad \text{takes.sec_id} = \text{section.sec_id} \\
\quad \text{as enrollment from section} \\
\quad \text{where } \quad \text{semester} = \text{'Fall'} \\
\text{and } \quad \text{year} = 2017
\]

Note that if the result of the subquery is empty, the aggregate function `count` returns a value of 0.

One way of writing the query might appear to be:

\[
\text{select } \quad \text{takes.course_id, takes.sec_id, count(ID)} \\
\text{from } \quad \text{section, takes} \\
\quad \text{where } \quad \text{takes.course_id} = \text{section.course_id} \\
\quad \quad \text{and} \quad \text{takes.sec_id} = \text{section.sec_id} \\
\quad \quad \text{and} \quad \text{takes.semester} = \text{section.semester} \\
\quad \quad \text{and} \quad \text{takes.year} = \text{section.year} \\
\quad \quad \text{and} \quad \text{takes.semester} = \text{'Fall'} \\
\quad \quad \text{and} \quad \text{t} \text{akes.year} = 2017 \\
\quad \text{group by takes.course_id, takes.sec_id}
\]

But note that if a section does not have any students taking it, it would not appear in the result. One way of ensuring such a section appears with a count of 0 is to use the `outer join` operation, covered in Chapter 4.

f. Find the maximum enrollment, across all sections, in Fall 2017.

One way of writing this query is as follows:
select max(enrollment) 
from (select count(ID) as enrollment 
from section, takes 
where takes.year = section.year 
and takes.semester = section.semester 
and takes.course_id = section.course_id 
and takes.sec_id = section.sec_id 
and takes.semester = "Fall" 
and takes.year = 2017 
group by takes.course_id, takes.sec_id) 

As an alternative to using a nested subquery in the from clause, it is possible to use a with clause, as illustrated in the answer to the next part of this question.

A subtle issue in the above query is that if no section had any enrollment, the answer would be empty, not 0. We can use the alternative using a subquery from the previous part of this question, to ensure the count is 0 in this case.

g. Find the sections that had the maximum enrollment in Fall 2017.
The following answer uses a with clause, simplifying the query.

```
with sec_enrollment as ( 
    select takes.course_id, takes.sec_id, count(ID) as enrollment 
    from section, takes 
    where takes.year = section.year 
    and takes.semester = section.semester 
    and takes.course_id = section.course_id 
    and takes.sec_id = section.sec_id 
    and takes.semester = 'Fall' 
    and takes.year = 2017 
group by takes.course_id, takes.sec_id) 

select course_id, sec_id 
from sec_enrollment 
where enrollment = (select max(enrollment) from sec_enrollment)
```

It is also possible to write the query without the with clause, but the subquery to find enrollment would get repeated twice in the query. While not incorrect to add distinct in the count, it is not necessary in light of the primary key constraint on takes.
3.2

**Answer:**

a. Find the total grade-points earned by the student with ID '12345', across all courses taken by the student.

```sql
select sum(credits * points)
from takes, course, grade_points
where takes.grade = grade_points.grade
    and takes.course_id = course.course_id
    and ID = '12345'
```

In the above query, a student who has not taken any course would not have any tuples, whereas we would expect to get 0 as the answer. One way of fixing this problem is to use the **outer join** operation, which we study later in Chapter 4. Another way to ensure that we get 0 as the answer is via the following query:

```sql
(select sum(credits * points)
from takes, course, grade_points
where takes.grade = grade_points.grade
    and takes.course_id = course.course_id
    and ID = '12345')
union
(select 0
from student
where ID = '12345' and not exists (select * from takes where ID = '12345'))
```

b. Find the grade point average (GPA) for the above student, that is, the total grade-points divided by the total credits for the associated courses.

```sql
select sum(credits * points)/sum(credits) as GPA
from takes, course, grade_points
where takes.grade = grade_points.grade
    and takes.course_id = course.course_id
    and ID = '12345'
```

As before, a student who has not taken any course would not appear in the above result; we can ensure that such a student appears in the result by using the modified query from the previous part of this question. However, an additional issue in this case is that the sum of credits would also be 0, resulting in a divide-by-zero condition. In fact, the only meaningful way of defining the GPA in this case is to define it as null. We can ensure that such a student appears in the result with a null GPA by adding the following **union** clause to the above query.
union
(select null as GPA
from student
where ID = '12345' and
not exists ( select * from takes where ID = '12345'))

(c. Find the ID and the grade-point average of each student)

select ID, sum(credits * points)/sum(credits) as GPA
from takes, course, grade_points
where takes.grade = grade_points.grade
and takes.course_id = course.course_id

group by ID

Again, to handle students who have not taken any course, we would have
to add the following union clause:

union
(select ID, null as GPA
from student
where not exists ( select * from takes where takes.ID = student.ID))

d. Now reconsider your answers to the earlier parts of this exercise under
the assumption that some grades might be null. Explain whether your
solutions still work and, if not, provide versions that handle nulls properly.
The queries listed above all include a test of equality on grade between
grade_points and takes. Thus, for any takes tuple with a null grade, that
student's course would be eliminated from the rest of the computation
of the result. As a result, the credits of such courses would be eliminated
also, and thus the queries would return the correct answer even if some
grades are null.

3.3

Answer:

(a. Increase the salary of each instructor in the Comp. Sci. department by
10%.

update instructor
set salary = salary * 1.10
where dept_name = 'Comp. Sci.'

(b. Delete all courses that have never been offered (that is, do not occur in
the section relation).
Chapter 3 Introduction to SQL

```
delete from course
where course_id not in
    (select course_id from section)
```

c. Insert every student whose tot_cred attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.

```
insert into instructor
select ID, name, dept.name, 10000
from student
where tot_cred > 100
```

3.4 Answer:

a. Find the total number of people who owned cars that were involved in accidents in 2017.
   Note: This is not the same as the total number of accidents in 2017. We must count people with several accidents only once. Furthermore, note that the question asks for owners, and it might be that the owner of the car was not the driver actually involved in the accident.

```
select count (distinct person.driver_id)
from accident, participated, person, owns
where accident.report_number = participated.report_number
    and owns.driver_id = person.driver_id
    and owns.license_plate = participated.license_plate
    and year = 2017
```

b. Delete all year-2010 cars belonging to the person whose ID is '12345'.

```
delete car
where year = 2010 and license_plate in
    (select license_plate
     from owns o
     where o.driver_id = '12345')
```

Note: The owns, accident and participated records associated with the deleted cars still exist.
3.5

**Answer:**

a. Display the grade for each student, based on the `marks` relation.

```sql
select ID,
    case
        when score < 40 then 'F'
        when score < 60 then 'C'
        when score < 80 then 'B'
        else 'A'
    end
from marks
```

b. Find the number of students with each grade.

```sql
with grades as
( select ID,
    case
        when score < 40 then 'F'
        when score < 60 then 'C'
        when score < 80 then 'B'
        else 'A'
    end as grade
from marks
)
select grade, count(ID)
from grades
group by grade
```

As an alternative, the `with` clause can be removed, and instead the definition of `grades` can be made a subquery of the main query.

3.6

**Answer:**

```sql
select dept_name
from department
where lower(dept_name) like '%sci%'
```

3.7

**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 `rl` and `r2` are non-empty. If one or both of `rl` and `r2` are
branch (branch_name, branch_city, assets)
customer (ID, customer_name, customer_street, customer_city)
loan (loan_number, branch_name, amount)
borrower (ID, loan_number)
account (account_number, branch_name, balance)
depositor (ID, account_number)

---

**Figure 3.17** Banking database

empty, the Cartesian product of \( p, r1 \) and \( r2 \) is empty, hence the result of the query is empty. If \( p \) itself is empty, the result is empty.

3.8

**Answer:**

a. Find the ID of each customer of the bank who has an account but not a loan.

\[
\begin{aligned}
\text{(select } & \text{ID} \\
\text{from } & \text{depositor}) \\
\text{except} \\
\text{(select } & \text{ID} \\
\text{from } & \text{borrower}) \\
\end{aligned}
\]

b. Find the ID of each customer who lives on the same street and in the same city as customer '12345'.

\[
\begin{aligned}
\text{select } & \text{F.ID} \\
\text{from } & \text{customer as F, customer as S} \\
\text{where} & \text{F.customer.street} = \text{S.customer.street} \\
& \text{and F.customer.city} = \text{S.customer.city} \\
& \text{and S.customer.ID} = '12345' \\
\end{aligned}
\]

c. Find the name of each branch that has at least one customer who has an account in the bank and who lives in “Harrison”.

\[
\begin{aligned}
\text{select } & \text{distinct branch.name} \\
\text{from } & \text{account, depositor, customer} \\
\text{where} & \text{customer.id} = \text{depositor.id} \\
& \text{and depositor.account_number} = \text{account.account_number} \\
& \text{and customer.city} = 'Harrison' \\
\end{aligned}
\]
3.9

**Answer:**

a. Find the ID, name, and city of residence of each employee who works for “First Bank Corporation”.

```sql
select e.ID, e.person_name, city
from employee as e, works as w
where w.company_name = 'First Bank Corporation' and w.ID = e.ID
```

b. Find the ID, name, and city of residence of each employee who works for “First Bank Corporation” and earns more than $10000.

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

This could be written also in the style of the answer to part a.

c. Find the ID of each employee who does not work for “First Bank Corporation”.

```sql
select ID
from works
where company_name <> 'First Bank Corporation'
```

If one allows people to appear in `employee` without appearing also in `works`, the solution is slightly more complicated. An outer join as discussed in Chapter 4 could be used as well.

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

d. Find the ID of each employee who earns more than every employee of “Small Bank Corporation”.

select ID
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. But note that the fact that ID is the primary key for works implies that this cannot be the case.

e. Assume that companies may be located in several cities. Find the name of each company that is located in every city in which "Small Bank Corporation" is located.

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

f. Find the name of the company that has the most employees (or companies, in the case where there is a tie for the most).

select company_name
from works
group by company_name
having count (distinct ID) >= all

(select count (distinct ID)
from works
group by company_name)

g. Find the name of each company 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')
3.10

**Answer:**

a. Modify the database so that the employee whose ID is '12345' now lives in “Newtown”.

```sql
update employee
set city = 'Newtown'
where ID = '12345'
```

b. Give each manager of “First Bank Corporation” a 10 percent raise unless the salary becomes greater than $100000; in such cases, give only a 3 percent raise.

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

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

The above updates would give different results if executed in the opposite order. We give below a safer solution using the `case` statement.

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