3.1 Write the following queries in SQL, using the university schema. (We suggest you actually run these queries on a database, using the sample data that we provide on the Web site of the book, db-book.com. Instructions for setting up a database, and loading sample data, are provided on the above Web site.)

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

b. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

c. Find the highest salary of any instructor.

d. Find all instructors earning the highest salary (there may be more than one with the same salary).

e. Find the enrollment of each section that was offered in Autumn 2009.

f. Find the maximum enrollment, across all sections, in Autumn 2009.

g. Find the sections that had the maximum enrollment in Autumn 2009.

Answer:

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

```
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.

```
select  distinct student.ID
from    (student join takes using(ID))
        join (instructor join teaches using(ID))
        using(course_id, sec_id, semester, year)
where   instructor.name = 'Einstein'
```

As an alternative to th **join .. using** syntax above the query can be written by enumerating relations in the **from** clause, and adding the corresponding join predicates on *ID*, *course_id*, *section_id*, *semester*, and *year* to the **where** clause.
Note that using natural join in place of **join .. using** would result in equating student *ID* with instructor *ID*, which is incorrect.

c. Find the highest salary of any instructor.

```
select max(salary)
from  instructor
```

d. Find all instructors earning the highest salary (there may be more than one with the same salary).

```
select   ID, name
from     instructor
where    salary = (select max(salary) from instructor)
```

e. Find the enrollment of each section that was offered in Autumn 2009. One way of writing the query is as follows.

```
select     course_id, sec_id, count(ID)
from       section natural join takes
where      semester = 'Autumn'
and        year = 2009
group by course_id, sec_id
```

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 replace **natural join** by the **natural left outer join** operation, covered later in Chapter 4. Another way is to use a subquery in the **select** clause, as follows.

```
select  course_id, sec_id,
        (select count(ID)
        from  takes
        where takes.year = section.year
              and takes.semester = section.semester
              and takes.course_id = section.course_id
              and takes.section_id = section.section_id)
        from  section
where   semester = 'Autumn'
and     year = 2009
```

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

f. Find the maximum enrollment, across all sections, in Autumn 2009. One way of writing this query is as follows:

```
select  max(enrollment)
from    (select    count(ID) as enrollment
         from      section natural join takes
         where     semester = 'Autumn'
         and       year = 2009
         group by course_id, 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 Autumn 2009. The following answer uses a **with** clause to create a temporary view, simplifying the query.

```
with sec_enrollment as (
        select    course_id, sec_id, count(ID) as enrollment
        from      section natural join takes
        where     semester = 'Autumn'
        and       year = 2009
        group by course_id, 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.

3.3 Write the following inserts, deletes or updates in SQL, using the university schema.

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

b. Delete all courses that have never been offered (that is, do not occur in the *section* relation).

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.

```
a. update instructor
       set salary = salary x 1.1
       where dept_name = 'Comp. Sci' ;
b. delete from course
      where course-id not in
           (select course_id
            from section);
c. insert into instructor
       select ID, name, dept_name, 10000
       from student
       where tot_cred > 100 ;
```

3.11 Write the following queries in SQL, using the university schema.

a. Find the names of all students who have taken at least one Comp. Sci. course; make sure there are no duplicate names in the result.

b. Find the IDs and names of all students who have not taken any course offering before Spring 2009.

c. For each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor.

d. Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.

Answer:

a. SQL query:

```
select  name
from    student natural join takes natural join course
where   course.dept = 'Comp. Sci.'
```

b. SQL query:

```
select  id, name
from    student
except
select  id, name
from    student natural join takes
where   year < 2009
```

Since the **except** operator eliminates duplicates, there is no need to use a **select distinct** clause, although doing so would not affect correctness of the query.

c. SQL query:

```
select  dept, max(salary)
from    instructor
group by dept
```

d. SQL query:

```
select  min(maxsalary)
from    (select dept, max(salary) as maxsalary
         from instructor
         group by dept)
```

3.12 Write the following queries in SQL, using the university schema.

a. Create a new course "CS-001", titled "Weekly Seminar", with 0 credits.

b. Create a section of this course in Autumn 2009, with *section_id* of 1.

c. Enroll every student in the Comp. Sci. department in the above section.

d. Delete enrollments in the above section where the student's name is Chavez.

e. Delete the course CS-001. What will happen if you run this delete statement without first deleting offerings (sections) of this course.

f. Delete all *takes* tuples corresponding to any section of any course with the word "database" as a part of the title; ignore case when matching the word with the title.

Answer:

a. SQL query:

```
insert into course
    values ('CS-001', 'Weekly Seminar', 'Comp. Sci.', 0)
```

b. SQL query:

```
insert into section
    values ('CS-001', 1, 'Autumn', 2009, null, null, null)
```

Note that the building, roomnumber and slot were not specified in the question, and we have set them to null. The same effect would be obtained if they were specified to default to null, and we simply omitted values for these attributes in the above insert statement. (Many database systems implicitly set the default value to null, even if not explicitly specified.)

c. SQL query:

```
insert into takes
    select id, 'CS-001', 1, 'Autumn', 2009, null
    from student
    where dept_name = 'Comp. Sci.'
```

d. SQL query:

```
delete from takes
where course_id= 'CS-001' and section_id = 1 and
      year = 2009 and semester = 'Autumn' and
      id in (select id
             from student
             where name = 'Chavez')
```

Note that if there is more than one student named Chavez, all such students would have their enrollments deleted. If we had used = instead of **in**, an error would have resulted if there were more than one student named Chavez.

e. SQL query:

```
delete from takes
where course_id = 'CS-001'

delete from section
where course_id = 'CS-001'

delete from course
where course_id = 'CS-001'
```

If we try to delete the course directly, there will be a foreign key violation because *section* has a foriegn key reference to *course*; similarly, we have to delete corresponding tuples from *takes* before deleting sections, since there is a foreign key reference from *takes* to *section*. As a result of the foreign key violation, the transaction that performs the delete would be rolled back.

f. SQL query:

```
delete from takes
where course_id in
      (select course_id
      from course
      where lower(title) like '%database%')
```

3.23 Consider the query:

```
select course_id, semester, year, section_id, avg (credits_earned)
from takes natural join student
where year = 2009
group by course_id, semester, year, section_id
having count (ID) >= 2;
```

Explain why joining *section* as well in the **from** clause would not change the result.

Answer: The common attributes of *takes* and *section* form a foreign key of *takes*, referencing *section*. As a result, each *takes* tuple would match at most one one *section* tuple, and there would not be any extra tuples in any group. Further, these attributes cannot take on the null value, since they are part of the primary key of *takes*. Thus, joining *section* in the **from** clause would not cause any loss of tuples in any group. As a result, there would be no change in the result.

3.24 Consider the query:

```
with dept_total (dept_name, value) as
    (select dept_name, sum(salary)
    from instructor
    group by dept_name),
dept_total_avg(value) as
    (select avg(value)
    from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

Rewrite this query without using the **with** construct.

Answer:

There are several ways to write this query. One way is to use subquerie in the where clause, with one of the subqueries having a second leve subquery in the from clause as below.

```
select distinct dept_name d
from instructor i
where
    (select sum(salary)
    from instructor
    where department = d)
    >=
    (select avg(s)
    from
        (select sum(salary) as s
        from instructor
        group by department))
```

Note that the original query did not use the *department* relation, and any department with no instructors would not appear in the query result. If we had written the above query using *department* in the outer **from** clause, a department without any instructors could appear in the result if the condition were <= instead of >=, which would not be possible in the original query.

As an alternative, the two subqueries in the where clause could be moved into the from clause, and a join condition (using >=) added.