Database schema:

Department(<u>deptid</u>, dname, location)
Student(<u>snum</u>, sname, deptid, slevel, age)
Faculty(<u>fid</u>, fname, deptid)
Class(<u>cname</u>, time, room, fid)
Enrolled(<u>snum</u>, <u>cname</u>)

SQL queries:

1. Get the names of faculty teaching a class in room 'LWSN1106'.
2. Get the number of students in the department named 'Physics'.
3. Get the student numbers and names of students who have never taken a course with an instructor named 'King'.
4. Get the student numbers and names of students who have only taken classes with an instructor named 'King'.
5. Get the department names and the number of students majoring in that department, sorted in decreasing order of the number of students.
6. Retrieve names of instructors teaching computer science courses, the courses they are teaching and the number of students enrolled in each course. For this question, we assume that Computer Science courses are those whose names start with 'CS'.
7. Get student numbers, names and majors of students who do not have an 'A' in any course they are enrolled in. For this question and the next, we assume that the ENROLLED table also contains a field named 'grade' for each record.
8. Get student numbers and names of straight-A students (students who have an 'A' in all the courses they are enrolled in).
9. Get the number of the student whose name comes first among all students (i.e. first in alphabetical order).

Solutions:

1.
   ```
   SELECT DISTINCT f.fname
   FROM Faculty f, Class c
   WHERE f.fid = c.fid AND c.room = 'LWSN1106'
   ```

   This is most basic type of query involving a join of two tables. In the 'select' part of the query, we always list the fields we would like to print out. Because we need the names of faculty in this question, we select the fname field from the Faculty table. The "FROM" part of the query is where we specify the tables we need to join. Because the fname field is only available in the "Faculty" table, we include the "Faculty" table as Faculty f. In order to specify the condition that the faculty should be teaching a class in the room 'LWSN 1106', we need to join the

Faculty table with the Class table, so we also include “Class c” in the FROM clause. The actual joining of tables and the specification of the condition take place in the WHERE clause. The join is simply specified by equating the common fields in the two tables and we enforce the condition on the classroom with *c.room = ‘LWSN1106’*. We need both conditions to be met, so we use a conjunction (AND) of the two conditions. We can think of the dot (.) notation as similar to the notation in object-oriented programming languages used to get the value of an object property, i.e. we can think of “Faculty f” as declaring an object *f* of class *Faculty*, and “f.fid” as accessing the value of the “fid” field of that object. The reason we include the “DISTINCT” keyword in the SELECT clause is to eliminate duplicate tuples in the result. If we omit DISTINCT, we might get multiple tuples with the same faculty name, because a faculty member might be teaching more than one class in the same room.

2.
```
SELECT COUNT(s.snum)
FROM Student s, Department d
WHERE s.deptid = d.deptid AND d.dname=’Physics’
```

This is a simple query involving the aggregate function “COUNT” which counts the number of tuples matching a specific condition. This query is very similar to Query 1 above, which involves a join of two tables with a condition on one of the tables. The main difference is that we print the number of tuples here, instead of listing all tuples in the result. The output of this query would be a single number for the count of students in the Physics department.

3.
```
SELECT  s.snum, s.sname
FROM Student s
WHERE s.snum NOT IN(SELECT e.snum
                    FROM Enrolled e, Class c, Faculty f
                    WHERE e.cname = c.cname
                    AND c.fid = f.fid
                    AND f.fname = ‘King’)
```

Our strategy for writing this query is the following: First, find the list of students enrolled in classes taught by an instructor named ‘King’, then find the students who are not in that list. The inner query in this question gives us the list of student IDs for students enrolled in classes taught by King. We need to join the three tables *Enrolled*, *Class* and *Faculty* for the inner query through their common fields. The condition in the outer query specifies that we are only looking for students, whose IDs are not in the list of IDs we found in the inner query. Note that we can only use NOT IN (or IN) to compare things of the same domain. For example, the reason we are able to use NOT IN here is that e.snum and s.snum have the same domains, as the *snum* field in the *Enrolled* table references the *snum* field in the *Student* table.

4.
```
SELECT s.snum, s.sname
FROM Student s
WHERE NOT EXISTS (SELECT *
                        FROM Enrolled e, Class c, Faculty f
                        WHERE e.cname = c.cname AND
                        f.fname <> ‘King’ AND
                        f.fid = c.fid AND
                        e.snum = s.snum)
AND
s.snum IN (SELECT e2.snum
            FROM Enrolled e2)
```

The condition for this query consists of two parts: The student should not be taking any courses with an instructor named ‘King’, but the student should be in the list of students taking courses. We take care of the first part using the “NOT EXISTS” function, which ensures that the inner query following it does not return any results. For a NOT EXISTS query, we can always use * to select all fields of the tables involved in the query. In this question, the inner query finds the tuples with students from the outer query taking classes from an instructor whose name is NOT King. The NOT EXISTS function right before this inner query ensures that such tuples do not exist for the students in the main SELECT clause. Note that the part that forms the link between the inner and outer queries here is the “e.snum = s.snum” condition. The second condition on the outer query ensures that the ID of this student is in the list of IDs in the *Enrolled* table (which means this student is enrolled in at least one class).

5.
```
SELECT d.dname, COUNT(s.snum) as numStudents
FROM DEPARTMENT d,  STUDENT s
WHERE d.deptid = s.deptid
GROUP BY d.dname
ORDER BY numStudents DESC
```

This is a simple GROUP BY query, which enables grouping of query results by a specific field, in this case, the department name. The effect of grouping by department name can be thought of as running the query once for each department name and combining all results. In this query, we count the number of students in each department by using COUNT(s.snum) and storing the result in a variable named *numStudents*. The result of the query will be one line for each department, consisting of the department name followed by the number of students in that department. The last line in the query has the effect of ordering the query results in decreasing order of the number of students in each department. Note the use of the *numStudents* variable in the order by clause. In general, ORDER BY can be used with any field name and has the effect of lexicographic ordering in the case of fields with string domains.

6.
```
SELECT f.fname, e.cname, COUNT(e.snum)
FROM Faculty f, CLASS c, ENROLLED e
WHERE f.fid=c.fid AND c.cname =e.cname AND c.cname LIKE 'CS%'
GROUP BY f.fname, c.cname
```

This query shows the use of regular expressions for string comparison in SQL. This is another group by query, this time grouping by two fields: name of faculty member and a class that faculty member teaches. Grouping by these two fields together has the effect of running the query once for each (fname, cname) pair, which meet the conditions specified in the WHERE clause. We find the faculty name, class name and number of students enrolled in that class for each fname, cname pair, if cname starts with 'CS'. The wildcard '%' here matches any number of characters.

7.
```
SELECT s.snum, s.sname, d.dname
FROM Student s, Department d
WHERE s.deptid = d.deptid AND
NOT EXISTS (SELECT *
                FROM ENROLLED e
                WHERE e.snum = s.snum AND e.grade = 'A')
```

This query is quite similar to query 4, where we used NOT EXISTS to ensure no records matching the inner query exist. In this case, the inner query finds tuples such that the student in the outer query is enrolled in a class where his/her grade is A.

8.
```
SELECT s.snum, s.sname
FROM Student s
WHERE (SELECT COUNT(e.cname)
           FROM Enrolled e
           WHERE e.snum = s.snum)
            =
          (SELECT COUNT(e2.cname)
           FROM Enrolled e2
           WHERE e2.snum = s.snum AND e2.grade = 'A')
```

For this question, we use a different approach: We think of this query as "for these students, the number of classes they are taking should be equal to the number of classes in which they have the grade 'A'". Therefore, the condition involves finding the number of classes this student takes (in the first inner query) and equating it with the number of classes in which the same student (note the use of s.snum in both inner queries) has a grade of A.

9.
```
SELECT s.snum
FROM Student s
WHERE s.sname <= ALL (SELECT s2.sname
                      FROM Student s2)
```

In this query, we use the ALL operator, which is used for comparing a field with everything in another list. In this question, we use it to compare the name of the student we will output with the names of all students in the university (which is found by the inner query). Note the use of "<=", and not "<" for comparison in the query. If we used "<", we wouldn't get any results, because the student whose name we are searching for is also in the list of all students in the university.