Solutions to Exercise Sheet 2

Exercise 3 Continued

h. Develop a query which employs “EXCEPT ALL” to find all those courses which appear more than once for the same year in “lecturing”. What is the reason for the repetition?

SELECT cid, year FROM lecturing
EXCEPT ALL
SELECT DISTINCT cid, year FROM lecturing;

There are 51 lines in the output table but they are not all different.
The reason for the repetition is that some courses which had multiple lecturers involved with teaching them in some years. (We will say more about this problem later in the course.)

Exercise 4: Aggregation

a. Write a query to list all courses (by their identifier “cid”) and the average enrolment over the years.

SELECT cid, ROUND(AVG(numbers)) FROM lecturing
GROUP BY cid
ORDER BY AVG(numbers);

Observe the use of the “ROUND” function, which is not strictly necessary but useful for giving a readable result.

Modify the query so that it also lists the number of different staff members (as identified by “sid”) involved in teaching it.

SELECT cid, ROUND(AVG(numbers)), COUNT(DISTINCT sid) FROM lecturing
GROUP BY cid
ORDER BY AVG(numbers);

You should observe that the “average” computed in these queries is not accurate. Some courses are listed multiple times. So, they skew the computed average. (We will say more about this problem later in the course.)

b. Develop a query which employs “GROUP BY” to find the courses which appear more than once for the same year in “lecturing”. Does this match the answer to the previous question? Why not?

SELECT cid, year FROM lecturing
GROUP BY cid, year
HAVING count(*) > 1;

We get exactly the 43 cid-year combinations where more than one lecturer was involved in the teaching. This differs from the answer to Exercise 3, part (h) because it does not list duplicate combinations of cid, year.

Exercise 5: Nested queries

a. Which courses (listed by their course title) were taken by students in 2003, judged by the existence of students taking their exams?

SELECT name FROM courses
WHERE bc IN (SELECT bc FROM allmarks03);
The `allmarks03` table only lists banner codes. We can select the corresponding names of the courses from the `courses` table. There were 72 such courses.

b. *Which courses had students taking their exams in 2003 even though they were not taught in 2003?*

```sql
SELECT name FROM courses
WHERE bc IN (SELECT bc FROM allmarks03)
EXCEPT
SELECT name FROM courses
WHERE cid in (SELECT cid FROM lecturing WHERE year = 2003);
```

We write another nested SELECT query to find the courses taught in 2003, and use `EXCEPT` to calculate the difference between the two. There were 13 such courses.

c. *Develop a query which lists those courses (by their course code “bc”) which (in 2005, say) had a higher average mark than the overall average.*

```sql
SELECT bc, AVG(mark) FROM allmarks04
GROUP BY bc
HAVING AVG(mark) > (SELECT AVG(mark)
                       FROM allmarks04);
```

There are 42 such courses.

d. *List the staff members who did not do any lecturing.*

```sql
SELECT firstname, lastname FROM staff
WHERE NOT EXISTS (SELECT * FROM lecturing WHERE lecturing.sid = staff.sid);
```

If we are happy to list only staff id numbers, we could have used `EXCEPT`:

```sql
SELECT sid FROM staff
EXCEPT
SELECT sid FROM lecturing;
```

However, to get the staff names, which are only available in the `staff` table, a nested query is useful. There are 46 such staff members.

**Exercise 6: From the May 2005 Exam**

a. *List all students (by registration number) who failed the course ‘06 02525’ in 2004.*

```sql
SELECT DISTINCT student
FROM allmarks04
WHERE bc = '06 02525' AND mark < 40;
```

There were 13.

b. *Compute the number of students and the number of courses for which marks are recorded in “allmarks04” in a single query.*

```sql
SELECT COUNT(DISTINCT student) AS "Number of students",
       COUNT(DISTINCT bc) AS "Number of courses"
FROM allmarks04;
```

Result from database:

<table>
<thead>
<tr>
<th>Number of students</th>
<th>Number of courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>657</td>
<td>76</td>
</tr>
</tbody>
</table>
c. What was the percentage of first class marks overall in 2003?

The easiest way to write this query is the following:

```sql
SELECT 100*(SELECT COUNT(*) FROM allmarks03 WHERE mark >= 70)/COUNT(*) || '%'
from allmarks03;
```

Here we are using the idea that a nested SELECT query that gives a single value as its result can be used as if it is just a value. The result from database is 28%.

Another variant of the solution that looks a bit complicated:

```sql
SELECT FirstClass.count * 100 / AllResults.count || '%'
AS "Percentage of first class marks"
FROM (SELECT COUNT(*) AS count
     FROM allmarks03) AS AllResults,
     (SELECT COUNT(*) AS count
      FROM allmarks03
      WHERE mark >= 70) AS FirstClass ;
```

Here we are using two nested queries, one to calculate the number of first class marks, and one to calculate the number of all marks.

It is not possible to do this calculation without nested queries. An attempt to do so might be as follows:

```sql
SELECT COUNT(firstclass.mark) * 100 / COUNT(marks.mark) || '%'
FROM allmarks03 AS marks, allmarks03 AS firstclass
WHERE firstclass.mark >= 70
AND ???
```

We are using two copies of the "allmarks03" table, one meant for collecting all marks and the other meant for collecting only the first class marks. However, when we use two tables in the "FROM" clause, the "SELECT" query considers all possible combinations of their rows. To stop it from considering combinations, we are forced to add a condition like "marks.student = firstclass.student" so that only one record is considered for each student. But this fails to capture our intent because students with non-first class marks will not be considered at all. Matching (or joining) the "marks" copy with the "firstclass" copy has the effect of removing all non-first class marks from the "marks" copy.

_The solutions for parts (d) and (e) will appear next week. You have another opportunity to work on them before next Thursday._