**Assignments**

**Assignments: RDBMS Fundamental**

**Q1:**

Identify the three violations of normal forms in the following table.

| **Instructor** | **Class** | **Class Number** | **Enrollment** |
| --- | --- | --- | --- |
| Lennon | Advanced Calculus | 10073 | 34 |
| McCartney | Introductory Physical Education | 10045 | 23 |
| Harrison | Auto Repair and Feminism | 10045 | 54 |
| Starr, Best | Quantum Physics | 10023 | 39 |

**Q2:**

**Based on following scenario develop an ER diagram**

The University Database

The university database stores details about university students, courses, the semester a student took a particular course (and his mark and grade if he completed it), and what degree program each student is enrolled in. The database is a long way from one that’d be suitable for a large tertiary institution, but it does illustrate relationships that are interesting to query, and it’s easy to relate to when you’re learning SQL. We explain the requirements next and discuss their shortcomings at the end of this section.

Consider the following requirements list:

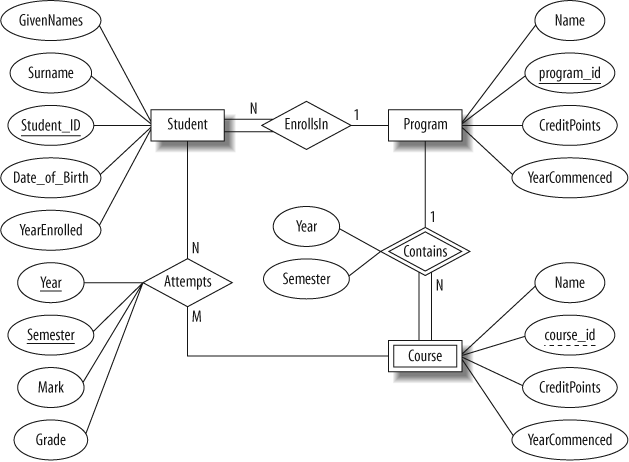
* The university offers one or more programs.
* A program is made up of one or more courses.
* A student must enroll in a program.
* A student takes the courses that are part of her program.
* A program has a name, a program identifier, the total credit points required to graduate, and the year it commenced.
* A course has a name, a course identifier, a credit point value, and the year it commenced.
* Students have one or more given names, a surname, a student identifier, a date of birth, and the year they first enrolled. We can treat all given names as a single object—for example, “John Paul.”
* When a student takes a course, the year and semester he attempted it are recorded. When he finishes the course, a grade (such as A or B) and a mark (such as 60 percent) are recorded.
* Each course in a program is sequenced into a year (for example, year 1) and a semester (for example, semester 1).
* What it doesn’t do
* Our database design is rather simple, but this is because the requirements are simple. For a real university, many more aspects would need to be captured by the database. For example, the requirements don’t mention anything about campus, study mode, course prerequisites, lecturers, timetabling details, address history, financials, or assessment details. The database also doesn’t allow a student to be in more than one degree program, nor does it allow a course to appear as part of different programs.

In our design:

* Student is a strong entity, with an identifier, student\_id, created to be the primary key used to distinguish between students (remember, we could have several students with the same name).
* Program is a strong entity, with the identifier program\_id as the primary key used to distinguish between programs.
* Each student must be enrolled in a program, so the Student entity participates totally in the many-to-one EnrollsIn relationship with Program. A program can exist without having any enrolled students, so it participates partially in this relationship.
* A Course has meaning only in the context of a Program, so it’s a weak entity, with course\_id as a weak key. This means that a Course is uniquely identified using its course\_id and the program\_id of its owning program.
* As a weak entity, Course participates totally in the many-to-one identifying relationship with its owning Program. This relationship has Year and Semester attributes that identify its sequence position.
* Student and Course are related through the many-to-many Attemptsrelationships; a course can exist without a student, and a student can be enrolled without attempting any courses, so the participation is not total.
* When a student attempts a course, there are attributes to capture the Yearand Semester, and the Mark and Grade.

**Solution:**

The ER diagram derived from our requirements is shown in [Figure 4-12](https://www.safaribooksonline.com/library/view/learning-mysql/0596008643/ch04s04.html#BAS-FIG-UNIER). Although it is compact, the diagram uses some advanced features, including relationships that have attributes and two many-to-many relationships.



*Figure 4-12. The ER diagram of the university database*

**Q3:**

Display the hire date, name and department number for all clerks.

SELECT hiredate, ename, deptno FROM emp WHERE job = ‘CLERK’;

Display the names and salaries of all employees with a salary greater than 2000.

SELECT ename, sal FROM emp WHERE sal > 2000;

Display the names and hire dates of all employees hired in 1981 or 1982

SELECT ename, hiredate FROM emp WHERE hiredate LIKE ‘%1981’ OR hiredate LIKE ‘%1982’;

—OR—

SELECT ename, hiredate FROM emp WHERE hiredate >= ‘1/1/1981’ AND hiredate <= ‘31/12/1982’;

Display the names and dates of employees with the column headers “Name” and “Start Date”

SELECT ename AS "Name", hiredate AS "Start Date" FROM emp;

Display the names and hire dates of all employees in the order they were hired.

SELECT ename, hiredate FROM emp ORDER BY hiredate;

Display the names and salaries of all employees in reverse salary order.

SELECT ename, sal FROM emp ORDER BY sal DESC;

Display ‘ename of department deptno earned commission $’ for each salesman in reverse salary order.

SELECT ename || ‘ of department ‘ || deptno || ‘ earned commission $’ || comm FROM emp WHERE job = ‘SALESMAN’ ORDER BY sal DESC;

Display the department numbers of all departments employing a clerk.

SELECT DISTINCT deptno FROM emp WHERE emp.job = ‘CLERK’;

Display the department number, total salary payout and total commission payout for each department.

SELECT deptno, sum(sal), sum(comm) FROM emp GROUP BY deptno;

Display the department number, total salary payout and total commission payout for each department that pays at least one employee commission.

SELECT deptno, sum(sal), sum(comm) FROM emp GROUP BY deptno HAVING sum(comm) > 0;

Display the name of each employee with his department name.

SELECT ename, dname FROM emp INNER JOIN dept ON emp.deptno = dept.deptno;

Display a list of all departments with the employees in each department.

SELECT dname, ename FROM dept LEFT OUTER JOIN emp ON dept.deptno = emp.deptno;

Display the names of each employee with the name of his/her boss.

SELECT s.ename, b.ename FROM emp s INNER JOIN emp b ON s.mgr = b.empno;

Display the employee number and name of each employee who manages other employees with the number of people he or she manages.

SELECT a.mgr, b.ename, count(a.mgr) FROM emp a INNER JOIN emp b ON a.mgr = b.empno WHERE a.mgr IS NOT NULL GROUP BY a.mgr, b.ename;

Repeat the display for the last question, but this time display the rows in descending order of the number of employees managed.

SELECT a.mgr, b.ename, count(a.mgr) FROM emp a INNER JOIN emp b ON a.mgr = b.empno WHERE a.mgr IS NOT NULL GROUP BY a.mgr, b.ename ORDER BY count(a.mgr) DESC;

Display the names and job titles of all employees with the same job as Jones.

SELECT ename, job FROM emp WHERE job = (SELECT job FROM emp WHERE ename = ‘JONES’);

Display the names and department name of all employees working in the same city as Jones.

SELECT ename, dname FROM emp INNER JOIN dept ON emp.deptno = DEPT.deptno WHERE loc = (SELECT loc FROM emp INNER JOIN dept ON emp.deptno = DEPT.deptno WHERE ename = ‘JONES’);

Display the name of the employee whose salary is the lowest.

SELECT ename FROM emp WHERE sal = (SELECT min(sal) FROM emp);

Display the names of all employees except the lowest paid.

SELECT ename FROM emp WHERE sal > (SELECT min(sal) FROM emp);

Display the names of all employees whose job title is the same as anyone in the sales dept.

Add a new Department to the DEPT table, and add a Manager and two Clerks to the EMP table that will belong to the new department.

INSERT INTO dept VALUES (50, ‘NEWDEPT’, ‘LONDON’);

INSERT INTO emp VALUES (8001, ‘FRED’, ‘MANAGER’, 7839, ‘14/01/1984’, 3100, null, 50); INSERT INTO emp VALUES (8002, ‘JIM’, ‘CLERK’, 8001, ‘18/04/1984’, 1020, null, 50); INSERT INTO emp VALUES (8003, ‘SHEILA’, ‘CLERK’, 8001, ‘08/12/1984’, 955, null, 50);

Transfer one of the new clerks to a diﬀerent department and transfer one of the previously existing clerks to your new department.

UPDATE emp SET deptno = 40, mgr = 7788 WHERE empno = 8002;

UPDATE emp SET deptno = 50, mgr = 8001 WHERE empno = 7876;

Give a unique job number to each job type.

UPDATE jobs SET job\_id = 10 WHERE job\_title = ‘ANALYST’;