# CM50267 Software Technologies for Data Science

# Lab 6 – Databases

|  |
| --- |
| - Dilbert by Scott Adams |
| Dilbert © Scott Adams 1995 |

## Deadline

Set: 16th Nov 2017

Deadline: 30 Nov 2017, 5pm

Submit: Via Moodle.

## Introduction

In this lab you will create an SQL database for the student assessment recording example we have used in the lectures. The lab is designed to be implemented using sqlite3 but can also be implemented using MySQL if you prefer and have access to a suitable server.

To use sqlite3 log on to linux.bath.ac.uk and at the command prompt type:

sqlite3 mydatabase.db

This will start the sqlite3 command line interface operating on a new database called *mydatabase.db*. The next time you run sqlite3 with the same file the database will be in the state it was when you last finished using the interface. At the sqlite3 command prompt ‘sqlite> ’ you can enter SQL queries without any further API wrapping.

The lab is structured as a series of steps that each build on those proceeding it. You should submit a document that lists the query or the output matching the labelled boxes in this document. You can use the word version of this document as your submission template a copy your text into the boxes.

Alternatively you can list all of the queries in a single file mydatabase.sql and execute the following command line:

sqlite3 –init mydatabase.sql mydatabase.db

This will run all the queries from mydatabase.sql before providing you with the prompt. Note that this does not reset the database at the start of the command.

## Task 1 (2 marks)

The first step (1.1) is not allocated any marks, but only requires you to type the provided text. It is intended to help you become familiar with the environment.

### Step 1.1 [0 Marks]

Create a table that holds the core unit information and populate it with some units.

|  |
| --- |
| sqlite> CREATE TABLE units (unitid INT, name TEXT, level INT, semester INT);  sqlite> INSERT INTO units VALUES(100,'CM60100', 6, 1), (101,'CM60101', 6, 1), (102,'XX60200', 6, 1);  sqlite> SELECT \* FROM units; |

|  |
| --- |
| 100|CM60100|6|1  101|CM60101|6|1  102|XX60200|6|1 |

Now create a table that holds the basic student information and populate it with some example students.

|  |
| --- |
| sqlite> CREATE TABLE students(studentid INT, name TEXT);  sqlite> INSERT INTO students VALUES (1001,'Rod'), (1002,'Jane'), (1003,'Freddy');  sqlite> SELECT \* FROM students; |

|  |
| --- |
| 1001|Rod  1002|Jane  1003|Freddy |

### Step 1.2 [3 Marks]

You need to enrol the students on to the appropriate units. This needs a table that links students to units.

I have gone back and made all id’s PRIMARY KEY’s

|  |
| --- |
| sqlite> CREATE TABLE enrolled(studentid INT, unitid INT, year INT); |

Now, insert the entries into this table that indicates which student is enrolled on which unit.

|  |
| --- |
| sqlite> INSERT INTO enrolled VALUES (1001,100,2016), (1001,101,2016), (1001,102,2016), (1002,100,2016), (1002,101,2016), (1002,102,2016), (1003, 102, 2016); |

We can now extract some useful information from the database.

Start by identifying the units taken by ‘Rod’:

|  |
| --- |
| sqlite> SELECT units.name FROM (students JOIN enrolled ON students.studentid=enrolled.studentid) JOIN units ON enrolled.unitid = units.unitid WHERE students.name=’Rod’; |

I’m writing in notepad and executing so there aren’t any sqlite>’s

|  |
| --- |
| 1.2a Output  CM60100  CM60101  XX60200 |

Construct the equivalent query to find the students enrolled on CM50266 Using unit CM60100

|  |
| --- |
| 1.2b sqlite> SELECT…  SELECT students.name FROM (students JOIN enrolled ON students.studentid=enrolled.studentid) JOIN units  ON enrolled.unitid = units.unitid WHERE units.name='CM60100'; |

|  |
| --- |
| 1.2c Output  Rod  Jane |

Using GROUP BY and COUNT write a query that gives the number of students taking the unit each year.

|  |
| --- |
| 1.2d sqlite> SELECT…  SELECT enrolled.year, COUNT(\*) FROM (students JOIN enrolled ON students.studentid=enrolled.studentid) JOIN units ON enrolled.unitid = units.unitid WHERE units.name='CM60100' GROUP BY enrolled.year; |

|  |
| --- |
| 1.2e Output:  2016|2 |

## Task 2 (7 Marks)

You will extend the database to include assessment. Your database should support the following features and capabilities:

1. Each unit will contains a number of assessments. – unit\_assessments
2. Each assessment will have a type that is either coursework or examination. - assessments
3. Each assessment will have a name. - assessments
4. Each assessment will have a maximum possible mark. - assessments
5. Each assessment will have a weighting that indicates its contribution to the overall unit mark. – unit\_assessments
6. Unit marks will be out of 100. (Add to enrolled? OR Calculate during query?)
7. When a coursework assessment becomes available to students for submission all students will have the same deadline, but some students may receive an extension to the deadline at a later point. – initial deadline: assessments; actual deadline: submissions
8. The point in time at which the students submitted a coursework assessment must be recorded. – submissions
9. It should be possible to tell if a submission has been marked and the mark awarded to a student for a given submission should be recorded. - Submissions (null value if not marked)
10. It should be possible to produce a list of the submission state for each student of a given assessment in a given year.
11. It should be possible to produce a list of the overall unit marks for all students enrolled on a given unit in a particular year.
12. It should be possible to produce a list of the submission state of all assessments for a particular student in a particular unit.
13. It should be possible to produce a list of the unit marks for all units being taken by a given student.

Not all of this functionality will be tested, but your database structure should support it.

### Step 2.1 [2 Marks]

You will need to add tables that contain the information need for assessments. And tables that relate entries in these tables to each other and the tables that you already have.

|  |
| --- |
| 2.1 sqlite> CREATE TABLE…  CREATE TABLE assessments(assessmentid INTEGER PRIMARY KEY, unitid INTEGER, name TEXT, type TEXT,  max\_mark INTEGER, weighting INTEGER, initial\_deadline TEXT,  FOREIGN KEY (unitid) REFERENCES units(unitid)  ); /\* need unitid’s to differentiate instances of EX1 so I won’t bother with something connecting units to assessments\*/  CREATE TABLE submissions(studentid INTEGER, assessmentid INTEGER, assessment\_name TEXT, submitted TEXT,  deadline TEXT, mark INTEGER,  FOREIGN KEY (studentid) REFERENCES students(studentid),  FOREIGN KEY (assessmentid) REFERENCES assessments(assessmentid)  ); /\* mark is null until the assessment is marked by the tutor \*/ |

### Step 2.2 [2 Marks]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Unit | Assessment | Deadline | | XX60200 (unitid 102) | EX1 | 15 January 2018 | | CM60100 (unitid 100) | CW1 | 24 November 2017 | | CM60100 | CW2 | 15 December 2017 | | CM60101 (unitid 101) | CW1 | 31 November 2017 | | CM60101 | EX1 | 18 January 2018 |   Table 1: Assessment Deadlines |

Populate your data base with the following:

The unit XX60200 has one assessment, an examination (EX1) it is marked out of 60.

The unit CM60100 has two assessments. Both assessments are coursework (CW1, CW2) and are marked out of 100. Each contributes 50% to the overall unit mark.

The unit CM60101 has two assessments. The first is coursework (CW1) and is marked out of 25 and contributes 25% to the overall unit mark. The second is an examination (EX1) and is marked out of 60 and contributes 75% of the overall unit mark.

|  |
| --- |
| 2.2a sqlite> INSERT…  INSERT INTO assessments (unitid, name, type, max\_mark, weighting) VALUES (102, 'EX1', 'exam', 60, 100),  (100, 'CW1', 'coursework', 100, 50), (100, 'CW2', 'coursework', 100, 50),  (101, 'CW1', 'coursework', 25, 25), (101, 'EX1', 'exam', 60, 75); |

Populate the database with records that indicate that the students have the deadlines shown in Table 1 for the various assessments. You will need at least one INSERT query per deadline. You should aim to avoid having a separate query for every student for every deadline.

|  |
| --- |
| 2.2b sqlite> INSERT…  /\* I have included an initial\_deadline column in the assessment table. I will the update that table with the deadlines above then initialise the submissions with the same deadline with a join \*/  UPDATE assessments SET initial\_deadline='15 January 2018' WHERE unitid=102 AND name='EX1';  UPDATE assessments SET initial\_deadline='24 November 2017' WHERE unitid=100 AND name='CW1';  UPDATE assessments SET initial\_deadline='15 December 2017' WHERE unitid=100 AND name='CW2';  UPDATE assessments SET initial\_deadline='31 November 2017' WHERE unitid=101 AND name='CW1';  UPDATE assessments SET initial\_deadline='18 January 2018' WHERE unitid=101 AND name='EX1';  INSERT INTO submissions (studentid, assessmentid, assessment\_name, deadline)  SELECT enrolled.studentid, assessments.assessmentid, assessments.name, assessments.initial\_deadline  FROM  (  assessments JOIN enrolled ON assessments.unitid=enrolled.unitid  ); |

### Step 2.3 [1 Mark]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Assessment | | Rod (sid=1001) | | Jane (sid=1002) | | Freddy (sid=1003) | |
| Unit | Name | Submit | Mark | Submit | Mark | Submit | Mark |
| XX60200 | EX1 | 15/1/2018 | 43 | 15/1/2018 | 38 | 18/1/2018 | 48 |
| CM60100 | CW1 | 24/11/2017 | 58 | 23/11/2017 | 48 | - | - |
| CM60100 | CW2 | 15/12/2017 | 62 | 14/12/2017 | 70 | - | - |
| CM60101 | CW1 | 31/11/2017 | 20 | 30/11/2017 | 18 | - | - |
| CM60101 | EX1 | 18/1/2018 | 39 | 18/1/2018 | 53 | - | - |

Table 2: Student Assessment Submissions

Table 2 shows the results obtained by each student on each of the assessments they were required to complete. Update the database to record this information. This is likely to be one SQL query per mark.

|  |
| --- |
| 2.3 sqlite> UPDATE…  UPDATE submissions SET submitted='15 January 2018', mark=43  WHERE unitid=102 AND assessment\_name='EX1' AND studentid=1001;  UPDATE submissions SET submitted='15 January 2018', mark=38  WHERE unitid=102 AND assessment\_name='EX1' AND studentid=1002;  UPDATE submissions SET submitted='18 January 2018', mark=48 /\* Should this be 15/1/2018 like everyone else? \*/  WHERE unitid=102 AND assessment\_name='EX1' AND studentid=1003;    UPDATE submissions SET submitted='24 November 2017', mark=58  WHERE unitid=100 AND assessment\_name='CW1' AND studentid=1001;  UPDATE submissions SET submitted='23 November 2017', mark=48  WHERE unitid=100 AND assessment\_name='CW1' AND studentid=1002;    UPDATE submissions SET submitted='15 December 2017', mark=62  WHERE unitid=100 AND assessment\_name='CW2' AND studentid=1001;  UPDATE submissions SET submitted='14 December 2017', mark=70  WHERE unitid=100 AND assessment\_name='CW2' AND studentid=1002;    UPDATE submissions SET submitted='31 November 2017', mark=20  WHERE unitid=101 AND assessment\_name='CW1' AND studentid=1001;  UPDATE submissions SET submitted='30 November 2017', mark=18  WHERE unitid=101 AND assessment\_name='CW1' AND studentid=1002;  UPDATE submissions SET submitted='18 January 2018', mark=39  WHERE unitid=101 AND assessment\_name='EX1' AND studentid=1001;  UPDATE submissions SET submitted='18 January 2018', mark=53  WHERE unitid=101 AND assessment\_name='EX1' AND studentid=1002; |

### Step 2.4 [2 Marks]

Your database should now be fully populated with the complete student records from semester one.

Construct a query that will show the overall unit mark for each student who took the CM60101 unit.

|  |
| --- |
| 2.4a sqlite> SELECT…  /\* Multiply by 1.0 to make sure that calculation is done to REAL precision \*/  SELECT studentid, SUM((1.0 \* submissions.mark \* assessments.weighting) / assessments.max\_mark)  FROM (submissions JOIN assessments  ON assessments.unitid=submissions.unitid  AND submissions.assessment\_name=assessments.name)  WHERE submissions.unitid=101 GROUP BY submissions.studentid; |

|  |
| --- |
| 2.4b Output: /\* I’ve used studentid to reference students because it seemed more realistic \*/  1001|68.75 /\* Rod \*/  1002|84.25 /\* Jane \*/ |

Construct a query that will show the overall unit marks for each of the units that Rod took.

|  |
| --- |
| 2.4c sqlite> SELECT…  SELECT submissions.unitid, SUM((1.0 \* submissions.mark \* assessments.weighting) / assessments.max\_mark)  FROM (submissions JOIN assessments  ON assessments.unitid=submissions.unitid  AND submissions.assessment\_name=assessments.name)  WHERE submissions.studentid=1001 GROUP BY submissions.unitid; |

|  |
| --- |
| 2.4d Output:  100|60.0  101|68.75  102|71.6666666666667 |

KMC2017