Coursework commentary 2018–2019

CO2209 Database systems

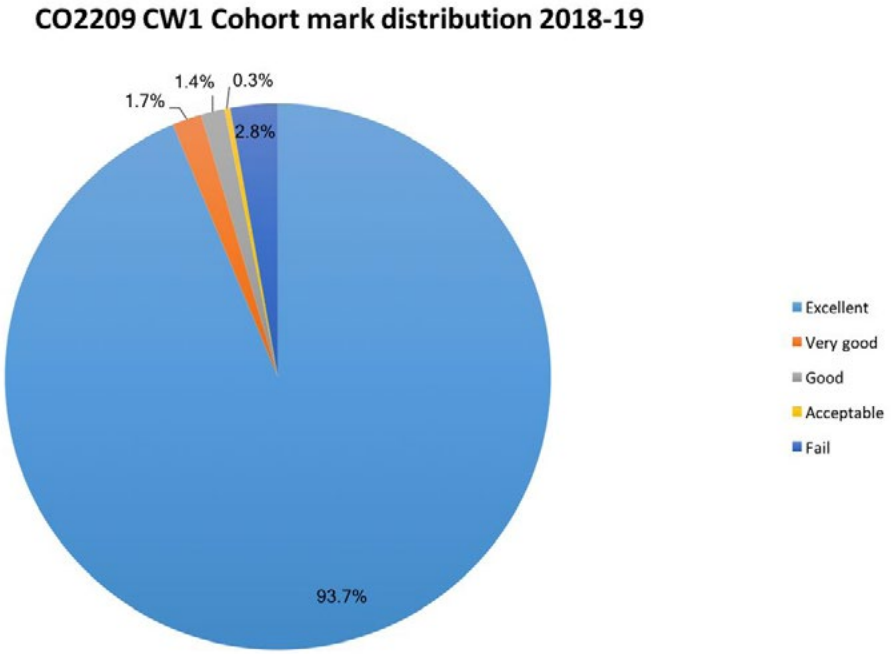
Coursework assignment 1

General remarks

Generally, results were good, with 94 per cent of students achieving an *Excellent* mark. Only those few who did not attempt large parts of the coursework got low marks, with 3 per cent failing.

Many students had serious problems downloading and installing a working version of MySQL from the Oracle site. That site does not make it easy to decide exactly what configuration of software should be downloaded, and many students spent several hours sorting out these issues. A number of students showed admirable ingenuity in going online and finding out solutions to their problems. This sort of experience is very similar to problems that students will face in their professional careers, and thus it could be argued that solving these problems was valuable preparation for employment. However, it was not the intention to make students face such challenges, and the lessons from this experience have been taken on board for the following year.

See cohort mark distribution for 2018–2019 below:



Comments on specific questions

Part A

It was hard not to get the full five marks here, although a few students thought that one or two cursory sentences or ultra-brief summaries of the MySQL manual were enough for maximum credit. Students should be guided by the number of marks a question is worth. For five marks – 1/20 of the whole coursework's value – you're probably going to need to write more than one or two sentences. It's worth noting that it is always interesting to see what a great range of practical experience we have among students taking this course, including experience with working databases.

Part B

A few students didn't realise that Unix, unlike Windows, is 'case-sensitive'. You need to name your files with this in mind, if they may find themselves being used in both systems.

Part C

It was hard to do poorly on this, although at least one student submitted table creation statements that were only a crude imitation of actual SQL. A couple of students didn't submit any outputs. The lesson here is: **read the instructions carefully**.

Part D

A few students looked at **question 3** of this part, saw the word 'defragmented', and wrote a short essay on defragmentation. But that did not answer the question. The point of this question, and the following two, was to give a concrete example of the 'logical/physical' distinction. For those who don't know: in a proper database system, physical changes to the way the data is stored – such as defragmentation, which may change the location of data on the disc – will not affect the queries on that data – except in the sense that they might run more swiftly after defragmentation. If a query makes reference to a column that is removed, it will fail. If not, not: you can delete a column, and queries on that table that do not refer to it will still run. The adding or deleting of whole rows (tuples) will not affect the validity of a query at all.

A few students, answering **question 5** wrongly believed deleting the LOCATION column would not affect the queries in Part E because LOCATION was not a Primary Key, and/or a Foreign Key. If an SQL query refers to a column that is not there, it will not work, whether or not the column in question is part of a key.

Part E

Most of the submitted coursework assignments got most of the marks for these queries. A few students ignored the instructions about submitting everything in one PDF file; or didn't number their queries; or didn't include the data generated by the query; or didn't include the natural language version of the query in their answer. Please read the instructions carefully and follow them.

Some marks were lost because the SQL did not match the required data: numbers were retrieved instead of names, or even values of chemical numbers when customer numbers were required. This is simply down to carelessness. Although doing this did not result in a loss of marks, many queries returned more information than required, in the form of extra columns. In a real system, especially one running in a distributed mode where data has to be sent over a network, you want to minimise the amount of data returned. If your user has

asked for Customer Numbers, don't burden the system by retrieving Customer Names and Addresses as well.

Some students believed that the best way to find the 'oldest' or 'minimum' of something was just to retrieve the entire list, in ascending order. Imagine doing this for a query that returns five million tables. Although it's not directly relevant to this course, note that implementing a 'minimum' or 'maximum' query by retrieving everything, with a LIMIT 1 qualification, although it answers the query – and got full marks in the coursework – might result in a huge performance difference in a real database with very large tables. Many optimizers automatically store minimum and maximum values (and other metrics) for each column, and so a query using MIN or MAX may execute much faster than the LIMIT 1 method, unless the optimizer is 'smart' enough to change the latter into the former.

There were more than a few queries where unnecessary JOINs took place. When writing a query, you should see if it can be answered using just one table; if not, can it be answered using just two tables; and so on? JOIN-ing several tables and then selecting from one of them may not be wrong – provided the JOIN preserves the needed information in the necessary table – but it's pointless, and in a real system, will slow it down.

These queries evoked the same sort of error in several students:

Query 11. "What is the DLEVEL of the most hazardous chemical(s) we stock?" This error turned up more than once:

SELECT DLEVEL

FROM CHEMICALS

WHERE DLEVEL >5;

This is called 'hardwiring'. The student has looked at the database and noted that putting '5' in the query will generate a correct answer for this particular instance of the database. But what if the database changes, so that our most hazardous chemical has a DLEVEL of 5? Queries should run correctly even if the data in the database changes – today's 'maximum' may not be tomorrow's 'maximum'.

Another example of 'hardwiring' was this answer to **Query 12:** "What is/are the DESCRIPTION(s) of the chemical(s) we stock with the most hazardous DLEVEL?"

SELECT DES

FROM CHEMICALS

WHERE DLEVEL >=8;

Where did that '8' come from?

Query 23, "List the date of the oldest order(s)" had several incorrect answers, where MAX(DATE) was used instead of MIN(DATE). It's slightly counterintuitive, but think of it this way: who is older, someone born in 2000, or someone born in 1940? And yet 1940 is smaller than 2000.

The most difficult queries were numbers 19–22. Many students got one or more of these wrong, probably because they didn't think closely enough about the question. In other words, their problem was not in the SQL, but in understanding what they needed to write the SQL for. Here are the four queries that caused the most difficulty.

19. List the CUSTNUMs and NAMES of any customer who has ordered chemical 9377.

20. List the CUSTNUMs and NAMES of any customer who has ordered any chemical **other than** chemical 9377. [Note: the answer should include

customers who have also ordered 9377, and those who have not, so long as they have ordered something else.]

21. List the CUSTNUMs and NAMES of any customer who has **only** ordered chemical 9377. (In other words, they have not ordered any other chemical.)

22. List the CUSTNUMs and NAMES of any customer who has **never** ordered chemical 9377.

These queries are all related, but each one is asking a distinctly different question, and each one needs a different SQL expression to answer it.