School of Computer Science, McGill University

COMP-421 Database Systems, Winter 2018

Written Assignment 2: SQL and Indexing

Solution Outline

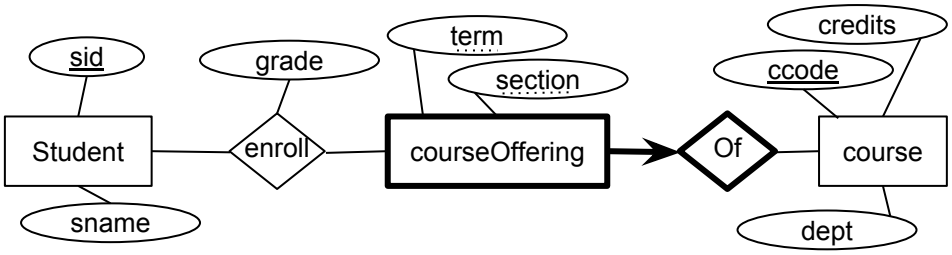
This is an individual assignment. You are required to work on your own to create the solution.
There is no late submission or extension allowed for this assignment as we need to post the solutions so that you can be prepared for the midterm.

Ex1 will be graded by an auotmated system. It is very important to read the instructions and follow them exactly. **Ex2** will be graded by the TAs.

Turn in two attachments. One for **Ex1** the tar file for automated system and another attachment for **Ex2** for the TAs to grade. For the later you may turn in a doc/pdf/txt format. Any other format, check with the TAs first.

Ex. 1 — SQL (70 Points)
Below is the ER model and corresponding relational model a simple student-course enrollment system. There are various courses under different departments, a department can chose to offer a course during a certain term (such as *winter 2018*) or it may not offer the course. Each course is also worth a certain number of credits. A course offering can have multiple sections (because there is too many students to fit into a single class). A student can enroll in the same course in different terms (if not passed previously), however they cannot enrol in different sections of the same course in a given term. These are enforced by the application. Students receive grades at the end of the semester.

A sample schema and few records have been provided as **setup.sh** Please add more records into this as you may need to test various scenarios. Use your individual database accounts to work on the assignment. **DO NOT** use your project groups database account as it is shared between all of your team members.



`student(sid, sname)`
`course(ccode, credits, dept)`
`courseoffer(term, section, ccode)`
ccode is foreign key to course.
`enroll(sid, term, section, ccode, grade)`
(term,section,ccode) is foreign key to courseoffer.
sid is foreign key to student.

Important !!

All the sql solutions will be evaluated by an automated system, which compares the output data produced by executing your query on our dataset with the expected output result for the correct query. So it is important that you include the correct column names, in the correct order, perform any ordering on output tuples as asked etc.

Double check your SQL for **typos**, for example if you spelt '*computer scence*' instead of '*computer science*', a query might not return the correct records and you will not get any points.
While the columnn and table names are not case sensitive, **the data itself can be case sensitive**. So do not write '*Computer Science*', where it was required to write '*computer science*' this can produce no results or wrong results. **For more details read the attached sql formatting guide**. If you have questions about this post it in the discussion forum for assignment 2. **Remember you will either get 0 or all points for a given SQL question !!**

For this assignment you will not create views or intermediate tables in your solution. All your answers should be comprised of only a select query. **Output ONLY the attributes in the question, following the exact order mentioned in the question**. Adding attributes not mentioned can result in a 0 score !

Unless specified, your output query should not produce duplicate results in your output resultset. Use the technique taught in class to eliminate duplicate records from the output.
Where an output ordering is asked for, remember to order the output records. The technique for this was also shown in class.

1. (2 Pts) List the course codes and credits of all 3 and 1 credit courses in the dept '*computer science*', ordering the output by the decreasing order of credits and ascending order of course code.
2. (2 Pts) List all the course codes and their credits for courses offered in the term '*winter 2018*' by the dept '*computer science*' - without using joins. Order the output by course code.
3. (3 Pts) List all the course codes and their credits for courses offered in the term '*winter 2018*' by the dept '*computer science*' - using joins. Order the output by course code.
4. (3 Pts) List all the course codes and their credits for courses offered in the term '*winter 2018*' by the dept '*computer science*' - using a correlated subquery. Order the output by course code.
5. (2 Pts) Give the course code and credits of all courses ever enrolled by the student with student id 12345678. Order the output by course code.
6. (2 Pts) Give the course code and credits of all courses enrolled by the student with student id 12345678 in the term '*winter 2018*'. Order the output by course code.
7. (2 Pts) Give list of course codes not offered in '*winter 2018*' but has been offered in '*winter 2017*'. Order the output by course code.
8. (3 Pts) List all the course codes and credits for courses that was taken by BOTH the students with student ids 12345678 and 12345679 in the term '*winter 2018*' (i.e., they are classmates - even if they took different sections of the same course). Order the output by course code.
9. (3 Pts) List all the course codes and credits for courses that were taken by the student with id 12345678 but not by the student with id 12345679 in the term '*winter 2018*' - using one of the SQL set operators taught in the class. Order the output by course code.
10. (4 Pts) List all the course codes, terms and grades for courses taken by the student with student id 12345678 which was offered by the dept '*computer science*' - using a correlated query. Order the output by course code and term.
11. (5 Pts) List the student id and names of all the ***other*** students that have taken the same course during the same term (section could be different) as that of the student with student id 12345678. Order the output by student id.
12. (2 Pts) Find the total number of students. Give the output column the name ***numstudents***.
13. (2 Pts) Find the total number of students enrolled for a course in the term '*winter 2018*'. Give the output column the name ***numstudents***.
14. (3 Pts) List the names of the department and the number of courses that each of them have (irrespective of they were offered or not). Name the later, ***numcourses***. Order the output in the decreasing order of the number of courses and then by the ascending order of the department names.

15. (5 Pts) List the course code and number of credits of all courses offered by dept '*computer science*' in '*winter 2018*' term that has at the least 5 students enrolled. Order the output by course code.
16. (6 Pts) Give the names of departments such that all the students wth a course enrollment for '*winter 2018*' term has also enrolled in at the least one of the courses offered by the dept in the '*winter 2018*' term. Order the output by department name.
17. (5 Pts) List the course codes and the number of students enrolled (name it ***numstudents***) across all sections for each course in '*winter 2018*' term. If a particular course is offered in '*winter 2018*', but has no students enrolled in it, it should show 0 for numstudents. Write this query without using any outer joins. Order the output by course code.
18. (6 Pts) Solve the above question by using an outer join, Order the output by course code. **Hint:-** use the derived table method discussed in class if you want along with the trick to manipulate *NULL* values.
19. (4 Pts) What is the average number of students enrolled in a course (across all sections) during the '*winter 2018*' term. Ignore course offering where no students were enrolled in any of its sections. Name the average column ***avgenrollment***.
20. (6 Pts) List the course code and the number of students enrolled in it (across all sections), for the course(s) offered in '*winter 2018*' term, that has the highest course enrollment in that term. If your output has multiple courses, it must be ordered by the course code. Name the number of students enrolled column as ***numstudents***.

Answer (Ex. 1) —

See attached zip file with SQLs.

Ex. 2 — Indexing (30 Points)
Consider the `enroll` table from the previous question.

`enroll(sid INTEGER, term VARCHAR(15), section INTEGER, ccode VARCHAR(10), grade VARCHAR(2))`

Here is some additional information.

- An INTEGER has 64 bits, average size of `term` is 10 bytes, average size of `ccode` is 8 bytes, and average size of `grade` is 1 byte.
- There are 100 departments and on an average 40 courses per department.
- Each year has 2 terms.
- Each term, 90% of the courses across all the departments are offered. All course offerings have only 1 section each.
- `enroll` has 4 years worth of data.
- The university has 100,000 students. (for simplicity, assume they all started together 4 years ago and no new students were added later).
- On an average each course offering has about 140 students enrolled.
- All students enroll in some course offering(s) each term.

Now assume there exists an indirect, clustered type II B+-tree index on `sid` of `enroll` and an unclustered type II B+-tree indirect index on `(term, ccode)` columns of `enroll`. A single data entry must always fit into one leaf page (it may not spread over more than one leaf page).

Further, rids of indexes take 10 bytes, internal page pointers are 6 bytes, page size is 4000 bytes. Leaf pages are filled on an average 60%, intermediate pages can have a fill factor in the range 50 - 100%. The root might have any fill factor.

1. For both indices calculate:
   (a) (6 Points) *the avg. number of rids per data entry, the size of the data entry and the total number of data entries.*
   (b) (4 Points) *the number of leaves.*

(c) (5 Points) *maximum and minimum possible number of intermediate nodes in the index (for the given possible fill factor range of 50-100%) and the height of the tree in each case.*

Answer (Ex. 2) —

1.

For the index on `sid`:

(a) Total number of courses = 100 dept × 40 courses = 4000 courses.
Number of courses offered in a given term = 4000 courses × 0.90 offered = 3600 courses.
On an average each course offering has 140 students enrolled, therefore the average number of courses taken by a student in a term is is $140 \times 3600/100,000 \approx 5$ courses per term.
Each student enrolls in 5 courses × 2 terms × 4 years = 40 courses offering enrollments overall per student. = 40 rids per data entry.
Size of data entry = sizeof(sid) + 40 rids × sizeof(rid) = $8 + 40 \times 10 = 408$ bytes.
Total number of data entries = total number of students (as all students enroll in some courses) = 100,000 data entries.

(b) index size in bytes = 408 bytes per data entry × 100,000 = 40,800,000 bytes.
Number of leaf pages required to store them, given a page size of 4000 bytes and 60% usage, $= 40,800,000 bytes/(4000 \times 0.60) = 17,000$ leaves.

(c) The number of page pointers an intermediate node can hold at 50% fill factor is 4000 × 0.50 / (8 byte sid + 6 byte page pointer) ≈ 142 page pointers.
To hold these page pointers we will need $17000/142 \approx 120$ intermediate nodes at max.
Similarly, if intermediate nodes are 100% full, it can hold $4000/(8 + 6) = 285$ page pointers, which will require a total of $17000/285 \approx 60$ intermediate nodes at minimum.
In either case, we just need one more level to store a root, making the height of the tree 2.

For the index on `(term, ccode)`:

(a) It is given that each course offering has on an average 140 students enrolled. = 140 rids per data entry.
Size of data entry = sizeof (term) + sizeof(ccode) + 140 rids × sizeof(rid) = 10 + 8 + 140 × 10 = 1418 bytes.
Total number of data entries = total number of course offerings across all years = number of courses offered in a term × number of terms in a year × number of years = 3600 × 2 × 4 = 28,800 data entries.

(b) index size in bytes = 1418 bytes per data entry × 28,800 = 40,838,400 bytes.
Number of leaf pages required to store them at 60% usage is = 40,838,400 bytes / (4000 × 0.60) = 17,016 leaves.

(c) The number of page pointers an intermediate node can hold at 50% fill factor is 4000 × 0.50 / (10 byte term + 8 byte ccode + 6 byte page pointer) ≈ 83 page pointers.
This requires $17,016/83 \approx 205$ intermediate nodes immediately above leaves. Plus another $205/83 \approx 3$ intermediate nodes above this. For a total of 205 + 3 = 208 intermediate nodes. Height of the tree in this case is 3.
Similarly, if intermediate nodes are 100% full, it can hold $4000/(10 + 8 + 6) \approx 167$ page pointers, which requires $17,016/167 \approx 102$ intermediate nodes at a minimum. These can be handled with just one root node. Hence height of the tree is 2 in this case.